# GENERAL

1 Getting Started ................................. 1
   1.1 Instructions for download and installation ........................................ 1

2 Operating system .................................. 9
   2.1 Windows ....................................... 9
   2.2 Linux ........................................ 9
   2.3 MacOS ......................................... 15

3 Digital Accessibility .............................. 19
   3.1 Accessibility Statement for nextnano Software ..................................... 19
   3.2 Accessibility Evaluation Report for nextnano Software .......................... 20

4 Support Ticket System ............................ 25
   4.1 Motivation ..................................... 25
   4.2 Channels ...................................... 25
   4.3 How to set up your free customer account .......................................... 26

5 Downloads .......................................... 27
   5.1 nextnano software .................................. 27
   5.2 nextnanomat ..................................... 28
   5.3 nextnano++ & nextnano<sup>3</sup> .................................. 28
   5.4 nextnano.NEGF ..................................... 29
   5.5 nextnano.MSB ..................................... 29

6 Release Notes ...................................... 31
   6.1 nextnanomat Release Notes ......................................................... 31
   6.2 nextnano++ Release Notes ......................................................... 31
   6.3 nextnano<sup>3</sup> Release Notes .................................................... 31
   6.4 nextnano.NEGF Release Notes ..................................................... 31

7 nextnanomat ........................................ 33
   7.1 GUI tabs ......................................... 33
   7.2 Settings ......................................... 59
   7.3 Main menu functions ................................................................. 63
   7.4 License activation ......................................................... 73
   7.5 Color maps ......................................... 73

8 nextnano++ ......................................... 75
   8.1 Introduction ....................................... 75
   8.2 Input File ......................................... 76
   8.3 Simulation output ......................................................... 219
   8.4 Material Database ......................................................... 219
   8.5 Command line arguments ..................................................... 230
   8.6 Convergence ......................................... 234
   8.7 Settings for maximizing performance ............................................ 241
We are currently rewriting our documentation. Please note that this manual is not complete yet but will be improved step by step.

nextnano - the next generation 3D nano device simulator

Welcome to our 3D nano device simulator nextnano.

nextnano++ is a Schrödinger-Poisson-current solver and simulates quantum wells, quantum wires, quantum dots, ...

nextnano++ features

- includes group IV materials (Si, Ge, SiGe) and all III-V materials, its ternaries and quaternaries;
- the nitrides are available in the zinc blende and wurtzite crystal structure
- flexible structures and geometries (1D, 2D and 3D)
- fully quantum mechanical electronic structure, based on the 8-band $k \cdot p$ model
- strain, piezo and pyroelectric charges
- growth directions along $[001], [011], [111], [211], \ldots$ in short along any crystallographic direction
- equilibrium and nonequilibrium, calculation of current close to equilibrium (semi-classical)
- magnetic fields

nextnano++ (written in C++) is the successor of the nextnano$^3$ code (written in Fortran).

The previous documentation of the nextnano++ software can be found here: https://www.nextnano.com/nextnanoplus/

1.1 Instructions for download and installation

Many thanks for your interest in the nextnano software.

1. Register
   First, you have to register here.
   - Register
   You will then receive an email with a link for the following password protected downloads, and instructions on how to activate your free 1-month evaluation license.

2. Download
   You can download the latest beta version of nextnano from our Downloads website:
Our most recent zip files contain executables for Windows, Linux and MacOS. Experienced users can use Wine to run nextnanomat on Linux. However, it does not work smoothly on Linux. Therefore, we strongly recommend Windows.

Older versions are also listed here:

a) nextnano_2020_12_09.zip (currently version 2020-12-09)
b) nextnano_2020_12_09_no_installer.zip (currently version 2020-12-09)

a) is recommended (with installer).
b) is without installer. No installation is required. Just unzip the file and start nextnanomat.exe. nextnanomat is the graphical user interface while the nextnano++ and nextnano³ are the actual scientific software doing the calculations.

A previous development version is this one:

a) nextnano_beta.zip (currently version 2021-03-15)
b) nextnano_beta_no_installer.zip (currently version 2021-02-18)

A previous stable version is this one:

a) nextnano.zip (currently version 2020-04-14)
b) nextnano_no_installer.zip (currently version 2020-04-14)

You can typically install several versions of nextnano on the same computer. There is no need to uninstall previous versions if you want to keep them.

**Note:** You can download the most recent version from here: nextnano_latest.zip. This zip file is generated automatically once we check in new source code. It is therefore not much tested and could be unstable. It includes binaries for Windows, Ubuntu and MacOS, and sample input files. It does not include an installer. This zip file is only suited for experienced users.

3. Start nextnanomat

nextnanomat (nextnanomat.exe) is the Front End & Workflow Manager that runs the nextnano++ and nextnano³ software.

4. License Activation

When you activate the license, please enter exactly your License Key into the field License key:

Then a license file is generated. You need this automatically generated license License_nnp.lic for both the nextnano++ and nextnano³ software. If your institution does not allow Online Activation, you can use Offline activation. Your evaluation license (free-of-charge) is valid for 1 month and can be activated on 2 computers. If you need a prolongation, please do not hesitate to contact us.

The license activation procedure is documented here: License Activation

The location of the license files can be checked using:

nextnanomat ==> Tools ==> Options ==> Licenses.

5. Number of licenses

For university and government institutions, the license fee covers all group members of the research group purchasing the software.
6. Example input files (Tutorials)

   The zip file contains many example input files.
   It is best to follow the tutorials of our Workshop.
   We are happy to provide any further tutorial input files that you need. All tutorials on this website List of tutorials are contained in the installatin folder.
   The complete list of tutorials can be found here: List of all tutorials
   If you can tell us what you want to simulate, we can point you to the most relevant examples.

7. Gnuplot

   We also recommend installing the free gnuplot software:

   Download Gnuplot

   Then you can generate plots from within our GUI nextnano mat conveniently.

8. First steps

   Start e.g. with this Laser Diode example. This input file solves the Schrödinger, Poisson and drift-diffusion current equation for a multi-quantum well laser diode.

   Detailed documentation: 1D InGaAs Multi-quantum well laser diode
Figure 1.1.2: nextnano++ and nextnano³ use the same license file: License_nnp.lic
Figure 1.1.3: Conduction band edge, valence band edge, and quasi-Fermi levels for electrons and holes of a Laser Diode
The number of quantum wells can be adjusted in the input file using the variable \texttt{NUMBER\_OF\_WELLS}.

![Figure 1.1.4: Template tab to define parameter sweeps. Here, the variable \texttt{NUMBER\_OF\_WELLS} is varied using the List of values 4, 5, 6, 7, 8. By clicking on Create input files, 5 input files are generated, each having a different number of quantum wells.](image)

This laser diode input file is included in the zip file of the latest update.

9. Facebook

Follow us on Facebook to stay informed about the latest news of the nextnano software:

facebook.com/nextnano

10. YouTube Videos

We also have a YouTube Channel:

There are two short videos that give you a first impression. If you are unsure whether to install the nextnano software, this video might be of help:

- Tutorial – How to install the software
- Tutorial – How to start a simulation
Figure 1.1.5: Energy and position resolved electron density $n(x, E)$ of the Laser Diode.
11. Python package

We offer a Python package for postprocessing the results. However, *nextnanopy* is currently suited for experienced Python and experienced *nextnano* users only.

```
pip install nextnanopy
```

12. Feedback and Support

Just let us know if you have any further questions, or if you have any feedback on our software. Please send support questions to support [at] nextnano.com or use our *nextnano* support Help Center. You will then receive a support ticket.

The *nextnano* team
CHAPTER TWO

OPERATING SYSTEM

2.1 Windows

nextnanomat and the nextnano software has been developed for Windows.

2.2 Linux

2.2.1 nextnanomat on Linux

*We strongly recommend using nextnanomat on Windows.*

nextnanomat is programmed in C#, and can thus be executed on any operating system. It is, however, developed on and optimized for Windows. On Linux, you have to either install Wine or Mono and then execute nextnanomat.

Please note that we developed nextnanomat for Windows and not for Linux, therefore many things might not work as expected.

Status 2020-04-07: Recommended to use Wine as described below.

Status 2018-05-03: Recommended to use Wine as described below.
Option 1: Wine

The following instructions are for Ubuntu version 20.10 (Groovy Gorilla). Commands for the Wine installation on various distributions are given in Wine Installation. A detailed description on how to install WineHQ on Ubuntu can be found here: Install WineHQ package on Ubuntu

The following commands are needed to install WineHQ on Ubuntu 20.10 (Groovy Gorilla). For older versions of Ubuntu, please see here: Wine Installation

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo apt-add-repository 'deb https://dl.winehq.org/wine-builds/ubuntu/ groovy main'
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Download the zip file of nextnano, i.e. the version without installer.

- Previous version: https://www.nextnano.com/nextnano3/restricted/download/update/nextnano_no_installer.zip

Unzip this file.

- Open a Terminal window and change to the directory where the `nextnanomat.exe` is located.
- Type: `wine nextnanomat.exe`
- There will be a pop-up message that a Mono package is missing which can be installed automatically. Click “Install”.
- There will be a pop-up message that a Gecko package is missing which can be installed automatically. Click “Install”.
- If nextnanomat does not work, close the terminal and open a new terminal and then try again.

The first thing you have to do when nextnanomat opens is to activate your license using:

```
$ wine ClientSideServerActivation.exe -O ~/Documents -UF "Homer" -UL "Simpson" -UE →"homer.simpson@nextnano.com" -E "homer.simpson@nextnano.com-2021-12-31-l1s4" -In →"Springfield Nuclear Power Plant" -A "license.nextnano.cloud" -Exe
   ClientSideActivator.exe
```

You obviously have to replace `homer.simpson@nextnano.com-2021-12-31-l1s4` with your name and your License key.

If everything works, you will find the license file `License_nnp.lic` in your home directory.

If it does not work, use Offline Activation:

```
$ wine ClientSideActivator.exe -O user_fingerprint.txt -UF Homer -UL Simpson -UE →homer.simpson@nextnano.com -In "Springfield Nuclear Power Plant" -E homer. →simpson@nextnano.com-2020-05-12-dt34
```

Then the generated fingerprint file `user_fingerprint.txt` which contains your

- Computer Name
- Disk Hash
- CPU Hash

has to be sent to support [at] nextnano.com. We will then send you your license key within 24 hours.
Please note that we developed nextnano\textsuperscript{mat} for Windows and not for Linux, therefore some things might not work as expected. Optionally, one can execute the nextnano++ and nextnano\textsuperscript{3} executables on Linux using Wine without nextnano. You have to type in: \ldots (Add an example here\ldots)

Known problems are:

- License activation: Somehow it was not possible to download the license file because nextnano\textsuperscript{mat} did not have Write Access to the Documents folder.
- On very old CPUs, the 64-bit version of nextnano++ does not work. In this case, please select the 32-bit version of nextnano++. Tools $\Rightarrow$ Options $\Rightarrow$ Simulation $\Rightarrow$ nextnano++ executable $\Rightarrow$ <path>/<nextnano++\bin 32bit\nextnano++_Microsoft_32bit_serial.exe
- Somehow the settings are not saved, i.e. each time one starts nextnano\textsuperscript{mat}, the license has to be activated anew.
- The program crashes.

Things that have to be fixed for future updates are:

- Use a monospaced font for the Input tab. (It seems that the default font for Windows is not found and then another default font (Tahoma) which is not monospaced is chosen.) One can choose a different font for the editor. Tools $\Rightarrow$ Options $\Rightarrow$ Editor $\Rightarrow$ Editor font

The following instructions are for Ubuntu version 18.04 LTS.

Run commands in a terminal:

```
sudo apt update  # Update system

sudo apt install wine-stable winetricks  # Install required packages (without "-stable" in versions before Ubuntu 16.10)

winetricks dotnet45  # Install .net support, follow along on-screen instructions, ignore warnings

winetricks corefonts  # Install basic fonts of Windows
```

Finally run winecfg $\Rightarrow$ Select “Windows 8” in Applications $\Rightarrow$ Windows Version and reboot afterwards.

To start the application, use wine /<your_directory_name>/nextnanomat.exe

Example: ”wine '/home/ubuntu/nextnanoTestversion/nextnanomat.exe' ”

Option 2: Mono

(This documentation should be updated for the latest Ubuntu and Mono versions.) It is not clear if it still works!

```
mono nextnanomat.exe &
```

On some Linux distributions (e.g. Ubuntu), Mono is already preinstalled.

The current Mono version uses .NET Framework version 4.0 by default. nextnano\textsuperscript{mat}, however, still requires the installation of the .NET Framework version 2.0. Therefore the following error occurs:

```
The assembly mscorlib.dll was not found or could not be loaded.
It should have been installed in the `/usr/lib/mono/2.0/mscorlib.dll' directory.
```

The following solution works:
All nextnano3 features seem to work on Linux Ubuntu. However, what does not work satisfactorily is the coloring of the 2D plots. They look too bright, so one can hardly recognize the results. A quick fix is to activate “Show grid”. Then the figure looks better.

If you have any feedback on these instructions, please let us know, then we can keep our documentation up to date.

We can also provide a command line version of nextnano for the various Linux distributions on request.

You can contact us at support [at] nextnano.com.

2.2.2 nextnano++ & nextnano3 on Linux

The nextnano++ and nextnano3 simulations can be executed from the Linux terminal by directly using the Linux executables, or by using Wine and the Windows executables. In order to make the programs executable, one needs to run the command `chmod a+x *.exe`, where (*) is replaced by the corresponding nextnano++ or nextnano3 Linux executable. For nextnano3 simulations, one also has to set the environment variable NEXTNANO by running the command `$ export NEXTNANO="/<directory_name>/nextnano/<date>"`. For license activation on Linux with Wine, please refer to License Activation. It is also possible to request a license.txt file in order to run simulations on Linux without any use of Wine. Should this be the desired option, one needs to modify the name of the license file in the below commands.

Option 1: Linux executables

The terminal commands for two nextnano++ and nextnano3 sample input files can then be given as follows, to be executed from the nextnano folder directory:

```
"nextnano3/Linux gcc 64bit/nextnano3/Linux_gcc_64bit.exe" -log -license
-"License/License_nnp.lic" -outputdirectory "Output/<name_of_input_file>" -
-inputfile "Sample files/nextnano3 sample files/1D_simple_GaAs_QW.in"
```

```
"nextnano++/bin 64bit/nextnano++_Linux_gcc_64bit.exe" --license
-"License/License_nnp.lic" --outputdirectory "Output" -log
-"Sample files/nextnano++ sample files/Quantum Mechanics examples/QW_finite_1D_nnp.in"
```

In order to simulate other input files, one simply needs to change the name/directory of the last argument. Detailed documentation on command line features can be found on the pages nextnano++ command line arguments and nextnano3 command line arguments. Further information and more examples can be found in the README/Linux file and the shell scripts included in the nextnano folder.

Option 2: Wine and Windows executables

In order to run simulations using the Windows executables, one can simply add the above commands after the command `wine` or `wine64` and modify the name/directory of the executable to choose one of the Windows executables.

Please feel free to contact us at support@nextnano.com regarding any questions.
2.2.3 Wine Installation

Below are the commands for installing Wine on various Ubuntu distributions.

Ubuntu 20.10 (Groovy Gorilla)

On Ubuntu 20.10, the following commands will install the stable branch of Wine:

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo add-apt-repository "deb https://dl.winehq.org/wine-builds/ubuntu/ groovy main"
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Ubuntu 20.04 (Focal Fossa)

On Ubuntu 20.04, the following commands will install the stable branch of Wine:

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo add-apt-repository "deb https://dl.winehq.org/wine-builds/ubuntu/ focal main"
sudo apt update
sudo apt install --install-recommends winehq-stable
```

Ubuntu 18.04 (Bionic Beaver)

On Ubuntu 18.04, the following commands are instead used:

```
sudo dpkg --add-architecture i386
wget -nc https://dl.winehq.org/wine-builds/winehq.key
sudo apt-key add winehq.key
sudo apt-add-repository "'deb https://download.opensuse.org/repositories/Emulators:/Wine:/Debian/xUbuntu_18.04/ ./'"
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DFA175A75104960E
sudo apt update
sudo apt install --install-recommends winehq-stable
```
Debian 10

On Debian 10, the following commands will install Wine:

```bash
sudo dpkg --add-architecture i386
sudo apt update
sudo apt -y install gnupg2 software-properties-common
wget -qO- https://dl.winehq.org/wine-builds/winehq.key | sudo apt-key add -
sudo apt-add-repository https://dl.winehq.org/wine-builds/debian/
wget -O -q https://download.opensuse.org/repositories/Emulators::Wine::Debian/Debian_10/Release.key | sudo apt-key add -
echo "deb http://download.opensuse.org/repositories/Emulators::Wine::Debian/Debian_10 ./" | sudo tee /etc/apt/sources.list.d/wine-obs.list
sudo apt update
sudo apt install --install-recommends winehq-stable
```

RHEL/CentOS 8

We have successfully installed Wine 5 on RHEL 8 using the following commands:

(These build Wine from source, hence the installation takes considerably longer.)

```bash
sudo -i
dnf clean all
dnf update
dnf groupinstall 'Development Tools'
dnf install libX11-devel freetype-devel zlib-devel libxcb-devel libxslt-devel
    libg crypt-devel libxml2-devel gnutls-devel libjpeg-turbo-devel
    libtiff-devel gstreamer1-devel dbus-devel fontconfig-devel
cd /opt
wget https://dl.winehq.org/wine/source/5.0/wine-5.0.tar.xz
tar -Jxf wine-5.0.tar.xz
cd wine-5.0
##For 32-Bit Systems:
./configure
##For 64-Bit Systems:
./configure --enable-win64
make
make install
```
Configuring Wine

After installing Wine, if you encounter errors about Mono, a useful trick is to delete the folder .Wine and force Wine to download and install Gecko and Mono automatically. Normally, Wine will suggest installing these the first time it is used. Furthermore, opening the wine uninstaller and removing interfering applications might solve some problems. In the end, we are able to use Wine without installing Winetricks or dotnet manually. If you are working on a remote machine, you might need to enable X11 forwarding to see the windows created during the Wine configuration. This is done by connecting via ssh -X and modifying the ssh.config file under etc/ssh accordingly.

2.3 MacOS

There are three options:

A: Run simulations from Terminal.

B: Run simulations with nextnanopy.

C: Run simulations with nextnanomat using Wine or Mono (not yet tested).

We strongly recommend the option B.

Basic simulation is possible from (A) Terminal. (B) nextnanopy Python package can not only run the simulations but also sweep variables and postprocess the results. For the options A and B, you need nextnano executables compiled for Mac, which we can provide you on request. Please feel free to contact us at support [at] nextnano.com. For the option C, Mono requires the Mac executables but Wine uses the ones compiled for Windows.

As of 1 Jul. 2021, we have not yet tested the option (C) for macOS Big Sur.

2.3.1 Option A: Run simulations from Terminal

The terminal commands for two nextnano++ and nextnano3 sample input files can then be given as follows, to be executed from the nextnano folder:

```bash
./<nextnano3 mac executable name>.exe -log -license "License/License_nnp.lic" -outputdirectory "Output/<name_of_input_file>" -inputfile "Sample files/nextnano3 sample files/1D_simple_GaAs_QW.in"
```

```bash
./<nextnano++ mac executable name>.exe --license "License/License_nnp.lic" --outputdir "Output" -log -inputfile "Sample files/nextnano++ sample files/Quantum Mechanics examples/QW_finite_1D_nnp.in"
```

Detailed documentation on command line features can be found in Command line arguments (nextnano++) and Command line arguments (nextnano3).

2.3.2 Option B: Run simulations with nextnanopy

nextnanopy (see Introduction) is our Python package for running simulations, sweeping variables and postprocessing results.
Install Python

You can install Python package Anaconda to establish a Python environment including NumPy, Matplotlib and an IDE called “Spyder”. With this, you can use nextnanopy from a graphical user interface.

Here, we explain an alternative way to install a Python package from Terminal via Homebrew. We have tested this with ARM64 Mac with macOS 11.4 (Big Sur).

1. In a macOS Terminal, type in:

```bash
# install Command Line Tools, if not installed on your machine
xcode-select --install

# install Homebrew, if not installed (cf. Homebrew website)
/bin/bash -c \\
'$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)'

# update Homebrew
brew update

# search for available Python packages
brew search python

# install Python3
brew install python3
```

2. Unversioned commands ‘python’, ‘pip’ etc. pointing to ‘python3’, ‘pip3’ etc., respectively, are installed into, e.g., `/opt/homebrew/opt/python@3.9/libexec/bin`. It is useful to set this path to ~/.zprofile.

```bash
# open ~/.zprofile with a text editor and write
"""eval "$(/opt/homebrew/bin/brew shelleven)"
""""export PATH=/opt/homebrew/opt/python@3.9/libexec/bin:$PATH"

# apply the changes
source ~/.zprofile

# make sure that the version 3.9 or later has been installed
python --version

# upgrade pip (NOT update)
pip install --upgrade pip
```

3. Using pip, please install NumPy and Matplotlib which are required for nextnanopy.

```bash
pip install numpy
pip install matplotlib
```

Install nextnanopy

You can either manually or automatically install nextnanopy. For more details, please refer to Installation. Manual installation using Git looks like:

```bash
## manual installation
# go to a folder where you want to store local repository of nextnanopy project
cd <folder name>

# clone source code from Github
git clone https://github.com/nextnanopy/nextnanopy.git

# build nextnanopy
cd nextnanopy/
python setup.py install
```
For the automatic installation, you can use pip:

```
pip install nextnanopy
```

### Configure nextnanopy

Open the file `config_nextnano.py` with an text editor to adjust the paths to your license, output and executable installation folders:

```python
open config_nextnano.py
# (adjust the paths)
# (save the file)
# run the config file to apply changes
python config_nextnano.py
```

### Running nextnanopy

Please see *Tutorials* and sample Python scripts to learn how to run a simulation with nextnanopy. nextnanopy repository includes sample Python scripts under `/nextnanopy/templates`.

#### 2.3.3 Option C: Run simulations with nextnanomat using Wine or Mono

This option is for those who wish to use GUI nextnanomat to run simulations.

nextnanomat is programmed in C#, and can thus be executed on any operating system. It is, however, developed on and optimized for Windows. On macOS, you have to install either Wine or Mono to run nextnanomat.

Wine was available from Mountain Lion 10.8 until Mojave 10.14. We confirmed that using Wine one could run nextnanomat.exe on Mojave 10.14.

**Wine did not work on Catalina 10.15 or later. However, Wine version 6.0.1 released on 7 Jun. 2021 is said to support wine64 on Apple M1. We will test once the built package becomes available.**

The following is for macOS 10.14 Mojave.

### Installation procedure

1. Install *XQuartz* (version 2.7.7 or later).
   - Please open Xquartz and check if it starts without errors.
2. Install *Wine Stable* for macOS.
   - On the website, both `.pkg` files and `.tar.gz` files are provided. Installation from `.pkg` files is handy. In this case, however, only the 32-bit version of nextnano software can be used, and currently one has to make sure that appropriate `libiomp5.dll` files are located in the same directory as “nextnano3.exe” and “nextnano++.exe” (even for the serial version of nextnano). If you need `libiomp5.dll` files, please contact us.
   - If you install Tarball for “Wine Stable” (32 + 64-bit), the 64-bit version is also available.
3. Install winetricks.
   - Launch *Wine stable* from Applications or Launchpad.
   - Terminal window shows up with a short introduction of important commands.

2.3. MacOS
ruby -e \
  "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew install winetricks

1. Fonts and configuration
   - Launch Wine stable again.
   - Run:
     
     winetricks corefonts  # install basic fonts of Windows
     winecfg  # Configure Wine

     • Setting window shows up. Select “Windows 10” for Windows version in the “Application” tab.
     • Press “OK” button.

Running nextnanomat

1. Launch Wine Stable from Application or Launchpad.
2. Run:

   wine /<your directory>/nextnanomat.exe

3. Activate the license with your email address (only once) and have fun!

If you have any feedback on these instructions, please let us know by sending an email to support [at] nextnano.com. It helps us keeping our documentation up to date.
The company nextnano GmbH is producing scientific software for a very specific audience. Due to its small company size, targeting a full accessibility especially retroactively is impossible. However, we aim to consider accessibility in any new software developments.

3.1.1 Conformance status

The Web Content Accessibility Guidelines (WCAG) define requirements for designers and developers to improve accessibility for people with disabilities. It defines three levels of conformance: Level A, Level AA, and Level AAA. The nextnano software is partially conformant with WCAG 2.1 Level A. Partially conformant means that some parts of the content do not fully conform to the accessibility standard. On some functionality conformance up to Level AA is achieved.

3.1.2 Feedback

We welcome your feedback on the accessibility of the nextnano software. Please let us know if you encounter accessibility barriers:

- Phone: +49-8121-7603205
- Support widget on this website
- support(at)nextnano.com
3.1.3 Compatibility with assistive technology

Tested and recommended on Windows 10. Partially compatible with screen reader software NVDA. Fully compatible with Windows built-in Screen Magnifier up to 400%, App Zoom up to 175% and color inversion. Not compatible with dark high contrast themes.

3.1.4 Limitations and alternatives

Cognitive and Learning Disabilities

Not suited due to scientific purpose of software and challenging cognitive performance for meaningful usage.

Hearing impairment

No limitations. Software does not include sound or audio.

Low vision

**Graphical user interface (GUI) nextnanomat**

- Font style and size of Editor tab adjustable.
- GUI compatible with screen magnifier at least up to 400%.
- Colormap choices for most common types of color blindness – Protanopia, Deuteranopia and Tritanopia.
- Screen reader compatibility, only partially given.

**Alternative 1: Without using GUI**

- Possibility to execute the scientific program by command line only (screen reader compatibility)
- Common output format → Visualization by any preferred visualization software possible.

**Alternative 2: nextnano GmbH also accepts simulation requests (charged) → Employees can simulate requested structures for you.**

3.1.5 Assessment approach

The company nextnano GmbH assessed the accessibility of the nextnano software by self-evaluation.

3.2 Accessibility Evaluation Report for nextnano Software

- **Executive Summary**
- **Scope of Review**
- **Reviewers**
- **Review Process**
- **Results**
  - Interpretative summary of review results
  - Detailed results
    - Audio
    - Video
3.2.1 Executive Summary

This report describes the conformance of the nextnano Software with W3C’s Web Content Accessibility Guidelines (WCAG). Based on this evaluation, the nextnano Software is close to meeting WCAG 2.1, Conformance Level A. Partially conforming to Level AA as well. Detailed review results are available below.

3.2.2 Scope of Review

Full software: nextnano Graphical User Interface (GUI) nextnanomat Version 4.3.1.0, date 2021-Dec-14.

3.2.3 Reviewers

Carola Burkl, Developer of nextnanomat (GUI)

3.2.4 Review Process

Conformance was tested for WCAG 2.1 Level A and AA. Manual review based on Praxent Accessibility Guidelines adapted for software. Compatibility with Windows built-in accessibility tools tested. Compatibility with screen reader NVDA tested.

3.2.5 Results

Interpretative summary of review results

Mostly conformant on WCAG 2.1 Level A. On some functionality conformance up to Level AA is achieved.

Detailed results

Audio

No audio
Video

No video

Content

- Meaningful Order (1.3.2) [A] → yes
- Sensory Capabilities (1.3.3) [A] → yes
- Under three flashes (2.3.1 & 2.2.2) [A] → yes
- Images of text (1.4.5) [AA] → yes
- Text Size & Spacing (1.4.12 & 1.4.4) [AA] → partially
  
  Compatible with screen magnifier at least up to 300% without loss of functionality or content.
- Content on Hover & Focus (1.4.13) [AA] → no
  - Content present on hover cannot be dismissed.
  - Hover content does not block region of mouse pointer.
- Alternative Text (1.1.1) [A] → partially
  - All images and non-text content of software have alternative text. → yes
  - Visual output of simulation results is not text interpreted. But result data can be viewed as text file as well.
- Page titles (2.4.2) [A] → yes
- Descriptive headings & labels (2.4.6) [AA] → partially
  - Buttons of main menu → yes (same functionality as buttons of tab pages)
  - Buttons of tab pages → no
- Language Changes (3.1.2) [AA] → no

Navigation & Architecture

- Clear text links (2.4.4) [A] → yes
- Multiple Paths (2.4.5) [AA] → yes
  
  Functionality of buttons can be accessed via buttons or navigation menu

Interface Elements

- Form Labels & Instructions (3.3.2) [A] → yes
  
  All forms and input fields are labeled
- Label in name (2.5.3) [A] → partially
- Consistent Identification (3.2.4) [AA] → no
- Name, Role, Value (4.1.2) [A] → yes, although name is not always descriptive
- Consistent Identification (3.2.4) [AA] → not relevant: no corresponding representation in non-html software
Interactions

• Focus order (2.4.3) [A] → no
• No focus change (3.2.1) [A] → yes
• No input change (3.2.2) [A] → yes
• Error identification (3.3.1) [A] → yes
• Keyboard focus visible (2.4.7) [AA] → yes
• Error suggestions (3.3.3) [AA] → yes
• Error preventions (3.3.4) [AA] → not relevant: no corresponding representation in software
• Pointer gestures (2.5.1) [A] → not relevant: no high precise pointer movements necessary
• Pointer Cancellations (2.5.2) [A] → no
• Motion actuation (2.5.4) [A] → not relevant: no motion functionality is used
• Keyboard only (2.1.1) [A] → no
  – Inside a tab → yes.
  – But to change tabs → no.
  – Logical tab order → partially
• No keyboard traps (2.1.2) [A] → no
• Character Key Shortcuts (2.1.4) [A] → yes (only active at focus)
• Adjustable Time (2.2.1) [A] → not relevant: no existing time limits

Visibility

• Use of color (1.4.1) [A] → no
  – Visualization of 2D or 3D data in the output tab uses heat maps to represent data values.
  – However, the color maps include choices for different types of color blindness as well as monochromatic options.
• Orientation (1.3.4) [AA] → yes (GUI can be resized to custom preference)
• Text & Image contrast (1.4.3) [AA] → yes
• Additional element contrast (1.4.11) [AA] → no

Code

• Clean code (4.1.1) [A] → not relevant: no html in software
• Reflow (1.4.10) → yes
Testing

- Assistive technologies
  - Screen reader NVDA tested
  - Screen magnifier (Windows) tested, up to 400%
  - Color Inversion (Windows) tested
  - Display increased App size (Windows) tested, up to 175%
  - Make text bigger (Windows) – does not work well
  - High contrast (Windows)
    * Black background does not work
    * Other background works but is not recommended
- Tested and recommended on Windows 10

3.2.6 References

1. Web Content Accessibility Guidelines (WCAG) Overview  https://www.w3.org/WAI/intro/wcag
2. Web Content Accessibility Guidelines 2.1  https://www.w3.org/TR/WCAG21/
3. Techniques for WCAG 2.1  https://www.w3.org/WAI/WCAG21/Techniques/
4. Accessibility Evaluation Resources  http://www.w3.org/WAI/eval/
4.1 Motivation

We are using a support system on all our channels. You have plenty of possibilities to get in contact with us. Whether you prefer writing emails, using the widget on our website or the Help Center directly, all requests are collected centrally and will be processed by their number - fair and square. Note, unless you purchased a special support option only technical support is included in your license fee. (E.g. no simulation requests or adapting your input files.)

This support system also offers the possibility of giving feedback. Critique and suggestions are highly appreciated.

4.2 Channels

4.2.1 Email

Both, emails to support(at)nextnano.com and support(at)nextnano.eu will automatically create a support ticket.

4.2.2 Widget

The easiest way, if you encounter a problem or do not find what you search for on our website - just click on the widget in the bottom right corner and fill out the form.
4.2.3 Help Center

In the help center https://nextnano.atlassian.net/servicedesk you can raise a new request or view your current/closed support tickets. It is all kept together, media files and conversation on your request.

4.3 How to set up your free customer account

This only needs to be done once. If you encounter any problems, don’t hesitate to contact us! Just click on the link View your support ticket in the first mail you receive, after creating a ticket (see Figure 4.3.1). You will be leaded through the process of setting up your help center account. Note, you don’t need an account for the communication on your ticket, but you will need one to access files which are provided by our support team.

Figure 4.3.1: Email confirmation after creating a support request.
This page contains the downloads for nextnano\textsuperscript{mat}, nextnano\textsuperscript{++} (includes nextnano.MSB), nextnano\textsuperscript{3} and nextnano.NEGF. nextnano\textsuperscript{mat} can be download on its own without a license.

A trial of nextnano\textsuperscript{++} and nextnano\textsuperscript{3} can be obtained from the registration page here.

Release notes of each version can be found here.

5.1 nextnano software

These downloads contain the latest versions of nextnano\textsuperscript{mat}, nextnano\textsuperscript{++} (includes nextnano.MSB) and nextnano\textsuperscript{3}. The download for nextnano.NEGF is further below.

\textbf{Note:} It is recommended to use the download with the installer.

If you wish to use the download without the installer, unzip the file and start nextnanomat.exe. The download without the installer may be useful if you do not have access to administrator rights on your computer.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Platform & Release Date & Download & Remark \\
\hline
Windows 10 & 2022-06-09 & Link & With Installer \\
\hline
Windows 10, MacOS, Linux & 2022-06-09 & Link & No Installer \\
\hline
Windows 10 & 2022-04-14 & Link & With Installer \\
\hline
Windows 10, MacOS, Linux & 2022-04-14 & Link & No Installer \\
\hline
Windows 10 & 2021-03-15 & Link & With Installer \\
\hline
Windows 10 & 2021-03-15 & Link & No Installer \\
\hline
Windows 10 & 2020-12-09 & Link & With Installer \\
\hline
Windows 10 & 2020-12-09 & Link & No Installer \\
\hline
Windows 10 & 2020-04-14 & Link & With Installer \\
\hline
Windows 10 & 2020-04-14 & Link & No Installer \\
\hline
\end{tabular}
\end{table}

5.1.1 32-bit

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Platform & Release Date & Download & Remark \\
\hline
Windows 10 & 2021-03-15 & Link & With Installer \\
\hline
Windows 10 & 2020-04-14 & Link & With Installer \\
\hline
Windows 10 & 2020-04-14 & Link & No Installer \\
\hline
\end{tabular}
\end{table}
5.2 nextnanomat

This package only includes nextnanomat. These downloads are for nextnanomat, the graphical user interface (GUI).

<table>
<thead>
<tr>
<th>Platform</th>
<th>Version</th>
<th>Release Date</th>
<th>Download</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>&gt;= 2022-06-08</td>
<td>Link</td>
<td>Latest (automatic build)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2022-06-02</td>
<td>Link</td>
<td>Stable</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2020-04-14</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2018-08-31</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2018-04-12</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2017-06-30</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2017-03-06</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2016-09-19</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2016-07-13</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2015-01-30</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2013-03-27</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2011-09-23</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2011-08-04</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2011-04-20</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2011-01-27</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-11-25</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-10-07</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-08-13</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-08-05</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-07-19</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-06-14</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-06-09</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-05-21</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-05-19</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-05-14</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-05-12</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-04-15</td>
<td>Link</td>
<td>Beta</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2010-02-05</td>
<td>Link</td>
<td>Beta</td>
</tr>
</tbody>
</table>

5.3 nextnano++ & nextnano³

This package only includes nextnano++ and nextnano³. nextnano++ and nextnano³ are the scientific softwares performing the simulations. nextnano++ is written in C++ while nextnano³ is written in Fortran.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Version</th>
<th>Release Date</th>
<th>Download</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>&gt;= 2022-06-08</td>
<td>Link</td>
<td>Latest (automatic build)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>&gt;= 2022-06-08</td>
<td>Link</td>
<td>Latest (Intel)</td>
</tr>
</tbody>
</table>
5.4 nextnano.NEGF

This package only includes nextnano.NEGF.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Version</th>
<th>Release Date</th>
<th>Download</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2022-04-29</td>
<td>Link</td>
<td>Stable</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2022-03-30</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2020-11-16</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2020-06-22</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
</tbody>
</table>

5.5 nextnano.MSB

(deprecated) This package only includes nextnano.MSB. nextnano.MSB is no longer maintained. Its functionality is fully included in nextnano++. Please use nextnano++ instead.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Version</th>
<th>Release Date</th>
<th>Download</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2017-12-20</td>
<td>Link</td>
<td>Stable (deprecated)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2017-02-07</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2017-01-20</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2016-11-13</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
<tr>
<td>Windows 10</td>
<td>n/a</td>
<td>2015-05-13</td>
<td>Link</td>
<td>Beta (deprecated)</td>
</tr>
</tbody>
</table>

Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.
Note: This page is still under development. Check in soon to see the updated release notes.

6.1 nextnanomat Release Notes

6.2 nextnano++ Release Notes

6.3 nextnano³ Release Notes

6.4 nextnano.NEGF Release Notes

6.4.1 Release 1.0.1

Release notes summary:

Linear alloy grading in a given layer

```
<Layer> <!-- all the material parameters will be linearly interpolated between the <Material1> and the <Material2> -->
  <Material1>mat1</Material1>
  <Material2>mat2</Material2>
  <Thickness unit="nm">5.0</Thickness>
</Layer>
```

In-plane nonparabolicity for multiband models

The in-plane nonparabolicity for multiband models can now be activated with the command

```
<Materials>
  ...
  <InPlaneNonParabolicity>yes</InPlaneNonParabolicity>
  ...
</Materials>
```
Update to individual scattering rates

The individual scattering rates of LO-phonon, alloy disorder and interface roughness can now be outputted
7.1 GUI tabs

7.1.1 Input

The input file tab supports both ASCII and XML format.

General features:

• Syntax highlighting
• Autocomplete
• Context-sensitive online help
• Use of variables

The syntax and the supported features depend on the software used for calculations. For detailed information see related software documentation:

• nextnano++
• nextnano^3
• nextnano.MSB
• nextnano.NEGF

7.1.2 Template

This web page describes the template feature of nextnanomat.

In the Template tab you can overview all variables which are defined in the input file. You can sweep a chosen variable by a list of values or a range of values. Additionally there is the option to compare the sweep results by the post-processing feature: It collects specified data from sweep files and stores them in a separate file, so that e.g. interband transition energy can be visualized as a function of well width.

An example how the template can be used to sweep over a variable, e.g. the quantum well width, is shown in this Exciton energy in quantum wells - Tutorial.
Variable definition in input file (required)

```plaintext
<Variable> = <Value> <Comment>
```

with the following syntax definition:

**Variable (required)** A `<Variable>` starts with a `$` or `%` sign (% should be preferred), can contain the characters A-Z, a-z, 0-9 and the underscore (_). It is case-sensitive.

- For `nextnano++`, the definition of variables and examples can be found in the `nextnano++` syntax documentation.
- For `nextnano`, the definition of variables and examples can be found in the `nextnano` macro documentation.

**Value (required)** Any string after the `=` sign without `#` and `!` characters is considered as `<Value>`. It is used as default value for the Template and `Template (Beta)` user interface.

**Comment (optional)** Starts with a `#` or `!` sign and is used as a description of the variable in the Template user interface. The comment may also include the following keywords (including brackets):

- `# (DoNotShowInUserInterface)` This variable is not shown to the user in the Template user interface.
- `# (DisplayUnit:<Unit>)` A string to show the unit for the variable (optional)
- `# (ListOfValues:<default>)` Default values for List of values input control
- `# (RangeOfValues:From=1,To=10,Step=1)` Default values for Range of values input control
- `# (HighlightInUserInterface)` Highlights variable in Template user interface in yellow

**Note:** Every line that starts with a `$` (not nn3) or `%` (after stripping leading spaces) and includes a `=` is considered a variable definition.

---

**Example**

```plaintext
$well_width = 6 # Variable for quantum well width. (DisplayUnit:<nm>) (RangeOfValues:From=6,To=18,Step=2) (HighlightInUserInterface)
```

---

**Sweep over a variable + optional post-processing**

If an input file includes at least one variable definition, it is possible to automatically sweep over it. The Template Tab of `nextnanomat` can be seen in Figure 7.1.2.1, already pre-filled for a variable sweep. To reproduce, please follow these steps:

1. The input file has to be chosen as the template file. Thus all correct variable definitions are loaded into the list.
2. Choose a sweeping variable and complete the required fields, e.g. List of values for QW_Separation.
3. The input files can be created by clicking the Create input files button.
4. They will be added to the batch list. To start the simulations you have to switch to the Simulation-tab, see Figure 7.1.2.2.
Figure 7.1.2.1: Template tab of nextnanomat with double quantum well input file loaded.
Figure 7.1.2.2: Batchlist of successfully simulated template sweep.
The sweep of a variable is now completed.

When the simulations are done, you have the possibility to use the post-processing feature. Its input fields are also located in the Template tab and can be seen in Figure 7.1.2.3. To use post-processing,

5. Choose the output file (*.dat) and the column number of the variable you want to compare, e.g. the file `energy_spectrum_quantum_region_Gamma.dat` displayed in Figure 7.1.2.4 and Figure 7.1.2.5

6. Additionally state the maximum number of values (rows) which will be compared. (It is not allowed to exceed the existing number of rows.)

7. Now push the button **Create file with combined data** and

8. Visualize your results in the Output-tab, see Figure 7.1.2.6.

For the steps 5 and 6 it is necessary to check the data structure of the simulation result you are interested in. Go to Output tab, select the data file and toggle **Text view**.

The result plots the chosen data in relation of the sweeping variable, see Figure 7.1.2.6.
Figure 7.1.2.5: Text view of same output file.

Figure 7.1.2.6: Result of post-processing. Electron energy eigenvalue in relation of quantum well separation.
Postprocessing of existing sweep

If you have already done some sweeps in the past, you can also use the post-processing feature of nextnano mat retroactively. Note, additional to the post-processing steps described above, you need to load the template file and select and fill in the variable values which have been used for the sweep.

7.1.3 Template (Beta)

Introduction

The Template (Beta) tab has a similar functionality to the Template tab. It allows convenient sweeps of an input file through the use of variables. In comparison to the standard Template feature, the Beta version allows nested sweeps of multiple variables. Its use is intended for experienced users who need to investigate their structure in complex dependency of more than one variable.

The input file

Any input file, which contains variables, can be used. At least two variables should be defined to use the full functionality of Template beta. Different options are provided to load an input file. It can be imported from the currently active input tab, searched within the file explorer or by inserting the path. Once opened, the defined variables should be shown on the right hand side. If changes to the file have been made by/with an external application, it can easily be updated through the reload button.
How to create a sweep

Follow the next steps:

1. (If the white panel is empty, click on the Add node button.)
2. Select the node New Sweep.
3. In the property panel you can define the parameters for this sweep:
   1. Choose a variable (e.g. $TEMPERATURE$).
   2. Set a correlating name for the sweep (e.g. “TempSweep”, Figure 7.1.3.3).
   3. Select List of values or Range of values and define the values for the sweep.
4. To add a second, third etc. sweep, repeat the steps above.
   1. To create a nested sweep, select a parent node before clicking on the Add node button. The new node will be nested below the selected one. (e.g. “BiasSweep”, Figure 7.1.3.4)
   1. If no node is selected, the new sweep will be on the same level and they will not be correlated to each other.

Save input files and run a sweep

In this example we use three different temperature values and a nested bias sweep with four different bias values for each temperature (see Figure 7.1.3.3 and Figure 7.1.3.4). Thus 3x4 = 12 different input files are to be created. Each input file receives the name and the values of both sweep parameters for identification.

Before running the simulation, there are different options for the simulation output:

Either the option Single folders or Structured folders can be selected. The first option generates one unique simulation output folder for each input file (ergo, for this example separate 12 folders, see Figure 7.1.3.6), whereas the structured option generates nested output folders matching the sweep structure. For the structured option a global index can be included which enumerates each sweep (Figure 7.1.3.7). And/Or a parent folder - for each sweep - can be added (Figure 7.1.3.8).

Lastly you can choose whether you want to create the input files only temporary to directly execute them or if you want to store them on the disk for later use.

After setting all your output preferences, click on the Create Input files button. You should see a confirmation after your input files are created successfully:

Next, go to the Simulation tab to find the created input files and execute them.

7.1.4 Simulation

The processing of simulations is done here:

- The simulations can be started, paused and stopped.
- There is a batch list, where input files can be added or removed. It is also possible to add complete directories plus sub-directories by the option Add Directory Tree to Batch List of the Simulation- menu tab.
- Information of the processed simulation is provided. Such as Start time, Duration and Status.
- The .log-file of the currently running simulation is written below. This file, with information on the calculation, is also stored in the output folder.
Figure 7.1.3.3: Parent sweep

Figure 7.1.3.4: Nested sweep
Figure 7.1.3.5: Output options

Figure 7.1.3.6: Output option a - Single folders.

Figure 7.1.3.7: Output option b - Structured folders plus global index.

Figure 7.1.3.8: Output option c - Structured folders plus parent folder.
Figure 7.1.3.9: Created input files in simulation tab.

7.1.5 Output

- **Navigation**
  - Menu
  - Panels for 1D, 2D and 3D

- **Visualization**
  - Supported Output Formats
  - Text view
  - 1D View
  - 2D View
  - 3D View

- **Features**
  - Overlay
  - Hide constant values
  - Show differences
  - Snap to gridpoints
  - Special fullscreen
  - Auto select color map
  - Fix middle color to specific value

- **Export functionality**
- Gnuplot export
  * Moving plot-files to another device
  * Optimize your Gnuplot Graph
- Customized
  * Paraview export
  * Python scripts

- Further information
  - Output settings
  - Clean up output folder
  - Color maps

The old documentation for this tab is available here: https://www.nextnano.com/dokuwiki/doku.php?id=nnm: output

Navigation

Menu

![Menu Image](image)

Figure 7.1.5.1: Menu, containing the most important functions to visualize the output data.

By right-clicking on the active output tab - which shows the output path or filename - you can create additional output tabs, for example to visualize different output directories. However these tasks are quite memory intensive, so use this feature with caution. We recommend to use a single output tab.

Panels for 1D, 2D and 3D

More specific visualization functions have their own panel. (For convenience some of them can be hidden.) Depending on the dimension of a data file, different functionality is available. For example, whether the graph should be represented by a dots or a line is only relevant for 1D data. Color maps and slices are only available for 2D and 3D data. The position and function of the different panels is shown in the following pictures.
Figure 7.1.5.2: Panels for a 1D data file

Figure 7.1.5.3: Panels for a 2D data file
Visualization

Supported Output Formats

The workflow manager nextnanomat can display the following output formats:

- **.txt** The file is displayed as a text.
- **.dat** The file is displayed as a graph, e.g. scalar field $f(x)$ or vector field $\mathbf{F}(x)$

\[
\begin{array}{c|c}
    x_1 & f(x_1) \\
    x_2 & f(x_2) \\
    \vdots & \vdots \\
    x_n & f(x_n) \\
\end{array}
\]

or

\[
\begin{array}{c|c|c|c|c}
    x_1 & f_1(x_1) & f_2(x_1) & \cdots & f_m(x_1) \\
    x_2 & f_1(x_2) & f_2(x_2) & \cdots & f_m(x_2) \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    x_n & f_1(x_n) & f_2(x_n) & \cdots & f_m(x_n) \\
\end{array}
\]

- **. mtx or . mat** Matrix format for e.g. a matrix, a table, a $f(x,y)$ graph. The x and y axes are labeled with integer numbers.

\[
\begin{bmatrix}
    A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\
    A_{21} & A_{22} & A_{23} & \cdots & A_{2n} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    A_{m1} & A_{m2} & A_{m3} & \cdots & A_{mn} \\
\end{bmatrix}
\]

- **.vtr** - 2D/3D VTK data format (rectilinear grid) - scalar field $f(x,y,z)$ or vector field $\mathbf{F}(x,y,z)$. They can be viewed using the Paraview software which is a full 3D visualization software while nextnanomat only displays 2D slices of 3D data files.

- **.fld** - 1D/2D/3D AVS data format (rectilinear grid) - scalar field $f(x,y,z)$ or vector field $\mathbf{F}(x,y,z)$

If a file extension is unknown it is treated as if it were a .txt file.

Text view

If text view is toggled, the content of the selected file is displayed. This view is read-only. For editing, the file can be exported to a custom texteditor. (We do not recommend to edit data files!)

1D View

Within graph view, one-dimensional data is displayed as curves. Whether data values are represented as points or lines is up to the user. Antialiasing can be switched on or off via Output settings. Additionally the line thickness, background color or the usage of a grid can be customized.

If a data file contains multiple columns, each column is represented by its own curve. The list of available columns is shown within the column view panel. Only checked curves will be displayed. Selected curves are highlighted to allow fast recognition of related data.

Helpful features for one-dimensional data visualization are:

- Overlay
- Hide constant values
• Show differences
• Snap to gridpoints
• Special fullsize
• Gnuplot export

2D View

Two-dimensional data is displayed as a heat map, also known as pseudo-coloring. Each value of the two-dimensional grid is mapped to a distinct color. The color difference perceived allows the visual interpretation of value differences. Therefore the change of the color gradient needs to be perceived linearly by the human eye. Some color maps, like the rainbow map, introduce artefacts and have misleading visual perception, which increases the risk for misinterpretation of scientific data. Find more information about Color maps.

Helpful features for two-dimensional data visualization are:
• Auto select color map
• Fix middle color to specific value
• Gnuplot export of surface plots

3D View

Real three-dimensional plots are not supported by nextnanomat. Instead 2D planes of the 3D data are visualized through heat maps, same as for 2D data. An additional panel allows the selection of the displayed plane (xy, xz or yz) and the position of the slice. This panel can be seen in Figure 7.1.5.13. If three-dimensional visualization is aimed for, we recommend exporting data to Paraview.

Helpful features for three-dimensional data visualization are:
• All of the 2D features
• Paraview export

Features

Overlay

For the best experience when visually analyzing the results of the simulation, it is sometimes necessary to look at different files at the same time. We call this the Overlay feature.

1. Select the plots you want to memorize in the column view panel.
2. Click the Add to Overlay button, see Figure 7.1.5.4. Alternatively, use the keys a (add) or + (Numpad only).
3. Select another output file.
4. (optional) add plots of multiple files to the overlay list.

Your memorized plots will be displayed in gray on top of the currently selected file.

You can check and edit the content of your overlay in the overlay list panel, this panel can be shown or hidden by clicking the Show list of overlays button. Remove selected curves with the Remove from Overlay button, shown in Figure 7.1.5.4, or use the Del/Entf or d (delete) key on your keyboard.

You can also export the overlay graphs to one combined image file, for further information refer to Gnuplot export.
Figure 7.1.5.4: Workflow to add or remove files from Overlay.
**Hide constant values**

**Note:** Be careful when using this feature. We recommend to always disable it after usage.

This feature hides any parts of the curves where the value does not change within 5 grid points (two to the left and right). Its purpose is for visualizing wave functions and probability densities on top of bandedges. The constant part which represents the energy level will be hidden. This allows to focus on the changing parts as well as for the underlying bandedge to be seen. The influence of this feature can be seen in Figure 7.1.5.5 and Figure 7.1.5.6.

**Show differences**

Whenever exactly two columns are selected in the column view panel, the value difference between these two curves is shown while the mouse is hovered over the data, see Figure 7.1.5.7.

**Snap to gridpoints**

If snap to grid points button is toggled (default: on) a small cursor cross follows the nearest data curve while the mouse cursor is hovered over the data. The coordinates and value displayed in the upper right corner of nextnanomat match the data values of the curve, see Figure 7.1.5.8.

**Special fullsize**

By using this feature, the view of the output diagram will be matched perfectly to a single graph. Either the selected graph of the column view, or it is iterating through all available columns of the currently displayed file. This feature is useful for example if a file contains multiple columns with a highly different range of values. Displaying all graphs at the same time (normal fullsize mode) can lead to certain curves appearing to be zero. By selecting such curve and clicking the special fullsize button, the y-axis limits are set to its respective minimum and maximum and the curve is displayed correctly. An example for this use-case can be seen in Figure 7.1.5.9.

Furthermore this feature is also useful for any other file containing more than one column. Iterating through the columns and displaying each curve in its optimal frame, allows for fast and convenient data evaluation, see Figure 7.1.5.10. "example one high number of curves"

**Auto select color map**

If auto selection of a color map is active, a diverging color map is selected for data files spanning from negative to positive values. Else a linear color map is chosen.

**Fix middle color to specific value**

This feature is aimed for the visualization of diverging data in combination with a diverging color map. It fixes the mapping of the neutral middle color to a specific value and thus ensures a symmetric color representation of diverging data.

**Note:** To be able to support custom color maps, this feature is not limited to predefined diverging color maps. To avoid unintentional usage we recommend to enable this feature on demand only.
Figure 7.1.5.5: Hide constant values on.
Figure 7.1.5.7: Arrow showing value difference of two data curves.

Figure 7.1.5.8: Snap to grid points feature enabled.

Figure 7.1.5.9: Special fullsize for a diverging range of values.

Figure 7.1.5.10: Special fullsize for a high number of curves.
The feature is enabled by checking the corresponding check box. If enabled, it is active whenever a diverging color map is assumed (error prone to enable flexibility) and the data file contains the specified value. Else it is inactive and grayed out. The visual feedback on the activity of the feature can be seen in Figure 7.1.5.11 and Figure 7.1.5.12.

Example usage of the feature can be seen in Figure 7.1.5.13.

**Export functionality**

Most of the export options are available from the context menu (right-click on the visualization). Some specific options can be accessed from the output menu button Export and Open in specific Format. For the latter option the custom defined paths to installed applications are used (with exception of the Gnuplot application). These paths need to be defined in the Output settings.
Gnuplot export

Gnuplot is a free graphing utility, which allows scientists to visualize data interactively. If installed it can be used to export 1D, 2D & 3D nextnano results. Either the currently selected file can be exported (by usage of the context menu) or the contents of the overlay list can be combined (1D) and exported (by usage of the output menu button). An example of such a combined file can be seen in Figure 7.1.5.14. The plot style for Gnuplot exports can be customized in Options: Gnuplot settings.

In 2D you have some additional options for your gnuplot file, displayed in Figure 7.1.5.15. The plot can be displayed as a color map, analog to the implementation of nextnano, or as a surface plot, which is a pseudo three-dimensional plot (Figure 7.1.5.16). Title and labels are optional and if they aren’t specified they will be taken directly from the file (if provided). If remember settings is checked, the next time the panel will be pre-filled with these settings. If you always use the same settings and don’t want this dialog to be displayed each time, you can chose the Create Gnuplot file − last used (*.plt) option in the context menu.

Gnuplots are interactive, see Figure 7.1.5.17, which makes them suitable to create animations e.g. for presenta-
Figure 7.1.5.15: Additional options for more-dimensional Gnuplot export.

Figure 7.1.5.16: Gnuplot surface representation of Figure 7.1.5.15. (Probability density of the 10th wavefunction in a hexagonal structure.)
tions. Furthermore they can be saved as vector graphics. To understand the dependency between plot files and their raw data files, please refer to the next section.

Figure 7.1.5.17: Interactive surface plot.

Moving plot-files to another device

1D-plots are linked to the original .dat file(s). So if you want to move your plot to another device, you can either save your plot as .pdf/.svg/.png file directly in gnuplot (recommended), or if you want to move the original .plt-file you also have to move all necessary .dat files (the paths can be adjusted when opening the .plt file with a text editor).

Warning: If you move the 1D plot file without data files or without adjusting the paths, it will be broken. (Won’t open when file is double-clicked.)

The data of 2D/3D-plots is directly stored within the .plt file, so there is nothing to consider when moving these files. (So why isn’t this done for 1D-plots alike? Because with the overlay feature you have the possibility to export a nearly unlimited number of .dat files into a single .plt file. If all this data would be duplicated and transferred into the .plt file, it would simply get to big.)

Optimize your Gnuplot Graph

Collection of some useful gnuplot commands. To edit these commands, open the .plt file within a text editor.

Semi-log plot

```
set logscale x
set logscale y
```

Change the line thickness (lw 4)

```
plot 'D:\bandedges.dat' linetype rgb "#FF0000" pt 5 ...
plot 'D:\bandedges.dat' linetype rgb "#FF0000" pt 5 lw 4 ...
```

Change font size of the x axis (20)

```
set xlabel "position (nm)" font "sans - serif"
set xlabel "position (nm)" font "sans - serif,20"
```

Use subscript and superscript (enhanced vs. noenhanced)

```
plot 'D:\density_hole.dat' using 1:2 title ^="p (10^18 cm^-3)" enhanced ...
plot 'D:\density_hole.dat' using 1:2 title ^="p (10^18 cm^-3)" noenhanced ...
```

Change range \([x_{min}, x_{max}]\) and \([y_{min}, y_{max}]\) of the graph

```
set xrange [-0.3:100.3]
set xrange [0:100]
set yrange [-1.5:2.5]
set yrange [-1.0:0.5]
```

Set/Remove grid
**set** grid
**unset** grid

**Change thickness of the border (lw)**

**set** border lw 2
**set** border lw 3

**Set/Remove legend**

**set** key on
**set** key off

**Set/Remove box around legend (box)**

**set** key on ... box
**set** key on ... nobox

**Increase font size in legend**

**set** key on ... font "sans-serif,14"
**set** key on ... font "sans-serif,18"

**Specify the location of legend**

**set** key left top inside ...
**set** key right bottom outside ...

**Remove line from legend (notitle)**

`plot 'D:\bandedges.dat' using 1:2 title "E_c" ...
plot 'D:\bandedges.dat' using 1:2 notitle "E_c" ...

**Add Greek letter to line in legend ("/{Symbol G}" enhanced)**

using 1:2 title "/{Symbol G} [eV]\" enhanced

produces .
e, l, m, q produce , , , , respectively.

**Add a label to the point (x, y) in the plot**

**set** label "label" at 0.5, 1.5

**Add an arrow**

**set** arrow from 1.5, 0.3 to 4, 2

**Graph Title**

**set** title "title" font "sans-serif,18"

**Generate high quality graphs**

1. In gnuplot window: Click on Export plot to file
2. Save as SVG files (.svg)
3. Open the saved .svg file with Inkscape
4. File Export .PNG Image... Select Drawing Export
Customized

In **Options: Data export** custom paths to installed applications (e.g. Paraview, notepad++) can be set, to allow convenient export of output files for the purpose of visualization or postprocessing.

**Paraview export**

1. Install the free software **Paraview** on your computer

2. Within nextnanomat:
   a. In **Options: Data export**, write the path to the Paraview executable in your computer, e.g. `C:\Program Files\ParaView 5.6.0-Windows-msvc2015-64bit\bin\paraview.exe`
   b. Select a 3D `.vtr` file of the simulation output folder and click on the Export and open in specific format button. Choose Open File with Paraview (Paraview will open automatically.)

3. Within Paraview:
   a. (The selected file should be highlighted automatically.) Click on **Apply**.
   b. Other settings:
      I. Representation: Surface
      II. Cell/Point Array Status: Choose the array to be displayed. For example, one can display the file `bandedges.vtr` and choose the array `Gamma` that corresponds to the Gamma conduction band. Click on **Apply**.
      III. Coloring: Choose the array to be displayed. (Gamma, for example)

Now you can play with the tool. Rotating, changing opacity and adding filters. Paraview is a very rich tool, hence it requires some time to explore all its capabilities. We recommend investing some time to learn about its filters:

- First, the file should be highlighted and then click on **Filter Alphabetical**.
  Some interesting ones are:
  - Edit cells by region
  - Calculator
  - Contour

Using **File Save State**, the values of all variables can be stored on your PC. These states can be reloaded, which is useful for the generation of scripts.

For further information, we recommend a good tutorial from **TACC**.

**Python scripts**

Custom written python scripts can be called directly from the Output tab. The parent folder of the python script, Path to the currently displayed output file as well as the path to the simulation folder are transferred as system arguments, see **Figure 7.1.5.18**. (If you need other specific information to be exported, just mail your request to the nextnano support team or use the widget on this website.) By using those system arguments in your python script, individual postprocessing of specific output files is reduced to a one-click effort.
Figure 7.1.5.18: Example of transferred system arguments when calling a python script.

Further information

References, you might find helpful...

Output settings

Options: View/Output

Clean up output folder

Clean Up Simulation Output Folder
Color maps

7.2 Settings

referred to and found in menu under Tools -> Options

7.2.1 Options: Simulation

7.2.2 Options: Material database

7.2.3 Options: Licenses

7.2.4 Options: Editor

7.2.5 Options: View/Output

7.2.6 Options: Expert settings

7.2.7 Options: Gnuplot settings

7.2.8 Options: Custom executable

In the settings shown in Figure 7.2.8.1 a custom executable can be defined, to be executed by the workflow manager nextnanomat.

Note: This is no default use-case for nextnanomat and should only be used by advanced users. Additionally, as it is a nice-to-have feature only, support regarding this feature is limited.

Most executables which can be started by command line, can be started by nextnanomat. This allows the usage of the batch list for polling and job execution control, as well as the visualization of output results (if the format is supported) or comfortable editing of input files. In many cases a combination with our cluster computing feature HTCondor is possible.

After defining the path to the executable as well as the working directory, a keyword identifier is necessary.

In case your executable needs input files: You need to find a common string within these files, thus nextnanomat can choose the correct executable for each job execution.

In case your executable does not need input files: You still need to create some dummy files as placeholder or job submission files. The keyword can be arbitrary.

For example the HelloWorld executable (see Figure 7.2.8.1), which is executing a fixed script independent of input variables, can be started by submission of a text file containing HelloWorld. This enables the workflow manager nextnanomat to start and monitor the execution and to collect the results.

Optionally a path to a folder structure or the database can be defined, if the executable needs additional files during execution. Specific command line arguments can be added at the bottom of this settings tab. The variable $INPUTFILE will insert the filename of the actual input file at the specified position of the command line arguments.

If it does not work as wished for, we recommend to switch the flag output additional debug info on (Expert Settings). The log then contains the command line arguments nextnanomat uses to start the application. Thus allows you to debug easily.
Figure 7.2.1.1: Settings for simulation and threading.
Figure 7.2.2.1: Settings for material databases.
Figure 7.2.3.1: Settings for nextnano Licenses.
Example: How to run TiberCAD from nextnanomat

To run TiberCAD you need to finish its installation and put your TiberCAD license file into the respective folder. Move the example or your personal input files to a folder with write permission. Choose the following settings in the custom executable tab, Figure 7.2.8.2:

Note: A special folder has to be created, which contains all input .tib, geometry .geo and mesh .msh files on the same hierarchy level. The path to that folder needs to specified in the settings for the custom executable.

This enables the editing - including syntax highlighting - of the input files within nextnanomat. And furthermore the parallel execution of input files (see Figure 7.2.8.3) and visualization of the output.

7.2.9 Options: Data export

write the path to the Paraview executable in your computer, e.g. C:\Program Files\ParaView 5.6.0-Windows-msvc2015-64bin\bin\paraview.exe

7.3 Main menu functions

Drop down functions of main menu, which do not clone a button functionality of any tab page.
Figure 7.2.5.1: Settings for output tab and graph style.
Figure 7.2.6.1: Settings for advanced user experience.

7.3. Main menu functions
Figure 7.2.7.1: Settings for Gnuplot export and graph style.
Figure 7.2.8.1: Settings for custom executable.

7.3. Main menu functions
Figure 7.2.8.2: Settings to run TiberCAD as a custom executable.
7.3.1 Activate license

The license activation procedure is documented here: License Activation

7.3.2 Convert nextnano³ input file to nextnano++

How to use this automatic conversion:

In the menu select Tools ==> Convert nextnano³ input file to nextnano++

When you use the automatic conversion of nextnano³ input file into nextnano++, you will find that it probably does not work completely.

If you save and run the nextnano++ input file that has been converted and that has the suffix _nnp.in, very likely some errors appear indicating which line(s) to change. Then some manual adjustments are needed, but the rough structure should help a lot for the conversion.

7.3.3 Clean Up Simulation Output Folder

After some months of simulating, the output folder can get quite huge. There is a feature to conveniently clean up the output folder with additional options, e.g. creating a backup of the input files. In the main menu, click on Tools and select Clean Up Simulation Output Folder. The dialog can be seen in Figure 7.3.3.1. To check multiple files at the same time, you need to select multiple files first and click on one checkbox afterwards.

7.3.4 Generate nextnanopy Config File

If you want to execute a nextnano product using nextnanopy, a config file is needed. This file contains all relevant information and paths to the licenses, nextnano executables and material database.

To make it more comfortable for you to use both nextnanomat and nextnanopy or switch to nextnanopy, you can export the settings already stored within nextnanomat into the format needed by nextnanopy.

Taken from the nextnanopy documentation on https://github.com/nextnanopy/nextnanopy/blob/master/docs/examples/Example0_Set_up_the_configuration.rst:

Per default the config file needs to be located at your home directory (usually C:\Users\Your_User\nextnanopy-config). When you import nextnanopy for the first time, it will
Figure 7.2.9.1: Settings for data export.
Figure 7.3.1: Tools functions

Figure 7.3.3.1: Dialog to help clean up the simulation output folder.

7.3. Main menu functions
automatically generate the configuration file with few default parameters. If this file already exists, it will not modify it. You can set up this configuration file only once and you do not need to worry about it anymore, except when you renew your license or update the nextnano version.

Thus, if you want to have your custom paths as specified in nextnanomat, just export a config file using this function and save it into your home directory. Then nextnanopy will not overwrite that file. After a new installation of a nextnano update, you can easily update the paths for nextnanopy as well, by using this function again.

Example config file:

```
[nextnano++]
exe = C:\Program Files\nextnano\2021_12_24\nextnano++\bin 64bit\nextnano++_Intel_˓64bit.exe
license = C:\Users\homer.simpson\Documents\nextnano\License\License_nnp.lic
database = C:\Program Files\nextnano\2021_12_24\nextnano++\Syntax\database_nnp.in
outputdirectory = D:\nextnano output
threads = 3

database = C:\Program Files\nextnano\2021_12_24\nextnano3\Syntax\database_nn3.in
threads = 0
debuglevel = -1
cancel = -1
softkill = -1

database = C:\Program Files\nextnano\2021_12_24\nextnano.NEGF\Material_Database.xml
outputdirectory = D:\nextnano output
threads = 3

database = C:\Program Files\nextnano\2021_12_24\nextnano.MSB\Materials.xml
outputdirectory = D:\nextnano output
debug = 0
```

7.3.5 Generate System Snapshot for Troubleshooting

By clicking this function, a text file - containing most relevant information for debugging - will be created. Save this file and send it to the nextnano Support Team to assist you.
7.4 License activation

Activate license

7.5 Color maps

On this page you will find some background information about scientific visualization of data. Additionally, the color map choices and customization options of the nextnano software are explained. The references and sources used to generate these maps can be found in the last section.

7.5.1 Introduction

Color maps for scientific software need to intuitively represent data without visual distortion.

7.5.2 Implementation

7.5.3 References
8.1 Introduction

8.1.1 General remarks

nextnano++ is a console application that is run from within the nextnanomat (nextnanomat) software (GUI). Alternatively, it can be executed from the command line (Command line arguments). The input file specifies the device that shall be simulated.

8.1.2 nextnanomat

nextnanomat (nextnanomat) is a convenient graphical user interface for nextnano++. In addition, it can visualize 1D, 2D and 3D simulations results.

8.1.3 Input file

The input file specifies all properties of the device, such as geometry, material composition, grid, contacts,... Furthermore, it sets all parameters that are needed to define the program flow of nextnano++. The keywords that can be used for this purpose are defined in the syntax (Syntax definition) of the input file.

8.1.4 Output

nextnano++ exports its results to a directory and in a certain format that have to be specified in the section (Simulation output) of the input file.

8.1.5 Examples

The nextnano++ installation provides some example input files (nextnano++ Tutorials) (C:\Program Files\nextnano\2020_12_09\Sample files\nextnano++ sample files) that can be run with nextnanomat, to get familiar with the program.
8.1.6 Material database

All material properties that are needed for simulation are specified as material parameters in database files (*Material Database*), which are provided with the nextnano++ installation. The database covers a large amount of *Zincblende* (all III-V and diamond-type like Si, Ge, ...), *Wurtzite* (GaN, AlN, InN, ...) materials, and their alloys.

If you have further questions, see the *Frequently Asked Questions (FAQ)* or contact support [at] nextnano.com.

8.2 Input File

8.2.1 Syntax definition

Documentation of input file syntax, use of variables and parser validation syntax.

**Use of variables, IF statements and tags**

- Syntax validation
- Variables: Operators and functions
- Advanced syntax documentation for validation files

The input file is almost white space independent. For example,

\[
\begin{align*}
\text{x} &= 5 \\
\text{y} &= 6 \\
\text{z} &= [1, 2]
\end{align*}
\]

and

\[
\begin{align*}
\text{x} &= 5 \\
\text{y} &= 6 \\
\text{z} &= [1, 2]
\end{align*}
\]

have the same effect. Similarly,

```
band{
```

and

```
band {
```

do the same. However, adding a line break as

```
band
{
```

is not allowed here due to parser limitations.

Next, we define variables. Variables are defined as

```
# This is a comment line.
x = 3.0 # This is comment, too. # Note that a comment runs until the end of the line.
```
Thus a variable name always starts with a dollar sign $ and is followed by a character, and then by an arbitrary number of characters, numbers, or underscores. Variables always have global scope and can be used for math as:

\[ y = sq r (y) * x \]

Note that mathematical variables are either doubles or double arrays, but get converted to integers if needed (only if no rounding is needed!). But elementwise mathematical operations between vectors or between scalars and vectors are not supported.

In addition, it is also possible to define string variables as:

\[
\begin{align*}
\text{name} &= \text{"some text"} \quad \# \text{ quoted string constant} \\
\text{id} &= \text{hello} \quad \# \text{ unquoted string constant (must not start with a number)}
\end{align*}
\]

Similar to variable names, also unquoted string constants must start with a letter (or underscore) in order to avoid conflicts with e.g. numbers in scientific notation. Please use quoted string constants instead if e.g. names starting with a number are needed.

Leading and trailing blanks are trimmed, and multiple string constants as

\[
\begin{align*}
\text{"aa b"} \\
\text{"c"}
\end{align*}
\]

or

\[
\begin{align*}
\text{"aa b"} \\
\text{"c"}
\end{align*}
\]

are automatically concatenated as

\[
\text{"aa b c"}
\]

with blanks inserted in between. Using the + operator, string variables can be concatenated (without inserted blanks) with each other or with double constants, double variables, or quoted (!) string constants as:

\[
\begin{align*}
\text{id} &= \text{"world"} \\
\text{id2} &= \text{"world"} \\
\text{num} &= 3 \\
\text{concat} &= \text{id + _} \text{id2 + num + 5} \quad \# \text{ result: "hello_world35"}
\end{align*}
\]

Double values are rounded into an integer first, before being concatenated to a string variable. Note that quoted string constants can only be added from the right.

Finally, variables can also be used to define conditional comments and conditional blocks.

Conditional comments have one of the following forms

\[
\begin{align*}
\text{#IF $x} \\
\text{#if $x}
\end{align*}
\]

and allow enabling/disabling of individual lines. Note that $x must be defined and a number, otherwise an error message will occur. In this example, the text is always commented out, unless $x is defined with value $x != 0 (actually 0.0, no rounding or truncation is being performed here!).

Similarly, conditional blocks have the form

\[
\begin{align*}
\text{#} \text{IF x} \\
\text{#if x}
\end{align*}
\]
and allow enabling/disabling of whole blocks. Note that the use of

```plaintext
!ELSE
!ELIF
```

is optional and that nesting conditional blocks is not allowed. Also here, \$x \neq 0\) needs to defined with value \$x \neq 0\) to be considered TRUE. And the debug statements

```plaintext
!IF($x)

name # 'This is a nested conditional line.'

!ELSE

name

!ENDIF
```

(see below) are also executed in the conditional block regions which are false.

Also note that all variables are removed from the final symbol table. Finally, remember that the parser is case sensitive.

Next, we define groups and attributes. Their name follows (except for the leading dollar symbol \$) the same convention as variable names. Validation files may also contain groupnames starting with a question mark \?. We have here the following syntax:

```plaintext
\$\text{groupname}\$

\$\text{attribute1} = \text{value}\$

\$\text{group2}\$

\$\text{attribute1} = \text{value}\#	ext{ Each group has its own scope !}

\$\text{group2}\$

\$\text{attribute2} = \text{value}\#	ext{ but attributes are unique.}

\$\text{group2}\$

\$\text{groupname}\$
```

Note that the order of groups is relevant, but the order of the attributes in a group is ignored. Also note that groups may be empty as:

```plaintext
\$\text{emptygroup}\$
```

The curly brackets \{\} belonging to each group are checked for correctness.

Tags

In addition, it is possible to add tags to explicitly check the current scope. For example,
or

```
groupName{
  ...
}
```

will have no effect, while

```
groupName{
  ...
}
```

will cause an error message, since the assumed scope and the actual scope do not match.

Input files may also be decorated at the root level (i.e. outside of any group) with XML tags such as

```
<id>
<id/>
```

or also:

```
<id/>
```

Here, \texttt{id} follows (except for the leading dollar symbol) the same convention as variable names. For backwards compatibility, in addition, also the empty (non-XML) tag \texttt{<>} is still available to e.g. check root level group closure. Please note that, whereas the simulator completely ignores the content of XML tags, they may have special meaning for calling programs such as \texttt{nextnanomat} and thus should not be altered without understanding their use. Practically, this means that, outside of groups, you may decorate input files/templates or also databases with XML tags in any way you wish. Just make sure to comment out stuff to be ignored by \texttt{nextnano++} with double comments \texttt{##} (to avoid possible collisions with conditional ifs) in order to add things such as:

```
$mass = 0.067 # m_0</unit> </variables>
```

At the root level, one can use the empty tag

```
<>
```

to check for the root scope. This is optional and not required. Tags with these brackets \texttt{<...>} are ignored by the parser and can be used to provide additional meta data. That is, everything right (or left) of such symbols is executed normally, as if there was just a ; (optional separator) instead of each tag.

Example:

```
<>
```

It might look like an XML tag but it is much simpler. Nesting and matching tags are not checked. No blanks or special characters except underscores _ may be used within tags \texttt{<...>}.

There exist different types of attributes. Allowed are

- real numbers
  ```
x 12.121
  ```
- integers
  ```
i 12
  ```
- vectors of real numbers

8.2. Input File
Attributes may also (like variables) be initialized with values of variables or the results of computations. But note that unlike variables, attributes may neither be redefined nor be used in mathematical expressions.

Next, there are also a couple of debug statements available, that can be used at any (reasonable) point inside of an input file or validation file:

\begin{verbatim}
!VARABS # prints all variables with their values into the standard output
!TABLE # prints the entire symbol table into the standard output
\end{verbatim}

Example:

\begin{verbatim}
$QW_WIDTH  6
$QW_SEPERATION  4
$QW_min  20
$QW_max  26
\end{verbatim}

But note that result of these debug statements obviously depends on their location in the file. Additionally, all variables and their values that are used in a simulation are written to the output folder into a file called

- variables_input.txt (for variables used in the input file)
- variables_database.txt (for variables used in the database file)

**Syntax validation**

**How to use validation and validation files**

The syntax is listed in the files input.val and database.val that are located in the Syntax/ folder. In fact, this is obsolete. These files are no longer distributed to the customer. The contents of these files are now contained inside the source code. If one starts nextnano++ with the argument -p, i.e. nextnano++.exe -p, then the syntax definition files input_syntax.txt and database_syntax.txt are written out. They are not needed. They have just informative character to display available input file syntax to the user.

As we have seen, the syntax of input files and validation files is except for some details mostly identical. As in content format languages such as XML or HTML, an almost unlimited hierarchy of objects is possible, as long as the resulting file is wellformed (means conforms with the syntax). But of course we would like now to impose constraints on what can be in an input file. This is easily done with a validation file. If desired, every input file can validated with respect to the constraints of any validation file. In this section, we describe the format used to define such constraints.

We start out with the definition of attributes. Here are some examples:
Thus, \( x_1, x_2, x_3, \) and \( x_4 \) are attributes of the type \textit{real} number. \( x_2 \) has also a mandatory minimum value defined, \( x_3 \) has also a mandatory minimum value, and \( x_4 \) must not be zero. Furthermore, \( x_3 \) is an optional attribute that may be omitted, while \( x_1, x_2, \) and \( x_4 \) are mandatory. Also note that as usual the order of the modifiers in the curly brackets \{\} is ignored.

Similarly, we have for \textit{integers}:

Vectors of \textit{real numbers} can be defined as:

Here, the modifier DIM states an explicit dimensionality of the vector, \texttt{MINDIM} and \texttt{MAXDIM} an explicit minimum and maximum dimensionality, and \texttt{MODDIM=3} requires that the dimension of the vector can be divided by 3.

Similarly, the corresponding definition for \textit{integer vectors} are:

Note that \texttt{MIN} and \texttt{MAX} can also applied to check the elements of vectors. In this case, the same bounds apply to all elements of a vector.

When validating vectors or intvectors, numerical variable values are automatically converted, therefore no extra brackets as \{123\} are needed for one-dimensional objects.

Next, strings are defined as

and \textit{choices} as

Finally, \textit{enumerations} are defined as

When validating strings, choices, or enums, numerical variable values are automatically converted into strings, therefore no extra quotes as "123" are needed.

Note that string, name string, and function string are essentially identical, except that string and name string forbid some characters such as > which could conflict with file I/O, and that name string further requires some of the characters a-zA-Z_.

Each attribute definition must be defined inside of a \textit{group definition} as

\texttt{\begin{verbatim}
x1{ TYPE real[2] } x2{ TYPE integer MIN 0 } x3{ TYPE integer MIN 0 MAX \{1.2 OPT"" \} x4{ TYPE real[2] NOTZERO"" }
\end{verbatim}}

\texttt{\begin{verbatim}
i1{ TYPE integer[2] } i2{ TYPE integer MIN } i3{ TYPE integer MIN MAX \{2 OPT"" }
\end{verbatim}}

\texttt{\begin{verbatim}
\end{verbatim}}

\texttt{\begin{verbatim}
\end{verbatim}}

\texttt{\begin{verbatim}
\end{verbatim}}

\texttt{\begin{verbatim}
\end{verbatim}}

\texttt{\begin{verbatim}
color{ TYPE string[2] VAL"white"}
\end{verbatim}}
An empty group is defined using a group definition that does not contain any attribute definitions, while a group definition inside another group definition indicates that the one group is supposed to contain the other group.

Of course, there are special modifiers for group definitions as well. Here we have available

```
shape_cuboid{
  TYPE = group
  OPT = ""
  x{
    TYPE = vec3
    DIM = 2
  }
  y{
    TYPE = vec3
    DIM = 2
  }
  z{
    TYPE = vec3
    DIM = 2
  }
}
```

Here, ITEMS=3 means that the group needs to be defined 3 times in the current scope, while MINITEMS=5, MODITEMS=4, and MAXITEMS=3 have the expected meanings. Note that the use of MINITEMS or ITEMS together with OPT may lead to constraints that cannot be fulfilled.

In addition, it also possible to specify dependencies between groups. Here is an example:

```
groupname1{
  TYPE = group
  ITEMS = 3
}
```

This means, that in the group `groupname` exactly one of the three children groups `x1`, `x2`, and `x3` must be present. At the same time, if `x3` is present, it may occur up to 5 times.

Similary,

```
groupname{
  TYPE = group
  ONE{
    TARGETS = "x1 x2 x3"
  }
  x1{
    TYPE = group
    OPT = ""
    MAXITEMS = 3
  }
  x2{
    TYPE = group
    OPT = ""
    MAXITEMS = 3
  }
  x3{
    TYPE = group
    OPT = ""
    MAXITEMS = 5
  }
}
```

means, that at least some (even all but not none) of the children groups `x1`, `x2`, `x3` must be present in the current scope.

Finally, by writing

```
groupname{
  TYPE = group
  MAXONE{
    TARGETS = "x1 x2 x3"
  }
  x1{
    TYPE = group
    OPT = ""
    MAXITEMS = 3
  }
  x2{
    TYPE = group
    OPT = ""
    MAXITEMS = 3
  }
  x3{
    TYPE = group
    OPT = ""
    MAXITEMS = 5
  }
}
```
we can guarantee that *atmost one* of the children groups \( x_1, x_2, x_3 \) is present in the current scope.

Finally, using

```plaintext
getline{ TYPE= X1 X2 X3
   NONE{ TARGETS="x1 x2 x3" }
   x1{ TYPE= OPT="" MAXITEMS= }
   x2{ TYPE= OPT="" MAXITEMS= }
   x3{ TYPE= OPT="" MAXITEMS= }
}
```

we can enforce that *none* of the children groups \( x_1, x_2, x_3 \) are present. This feature is only useful for inherited groups.

Similarly, the conditions ?TWO, ?THREE, ?MAXTWO, ?MAXTHREE, ?COND_TWO, ?COND_THREE, ?COND_MAXTWO, ?COND_MAXTHREE can be used to check for 2 or 3 occurrences.

In addition, it is also possible to define conditional dependencies as

```plaintext
getline{ TYPE= X1 X2 X3
   COND_SOME{ COND="n" TARGETS="x1 x2 x3" }
   n{ TYPE= OPT="" MAXITEMS= }
   x1{ TYPE= OPT="" MAXITEMS= }
   x2{ TYPE= OPT="" MAXITEMS= }
   x3{ TYPE= OPT="" MAXITEMS= }
}
```

which that if group \( n \) is present in the current scope, then at least *some* of the children groups \( x_1, x_2, x_3 \) must also be present in the current scope.

Similarly,

```plaintext
?COND_NONE{ COND="n" TARGETS="x1 x2 x3" }
?COND_ONE{ COND="n" TARGETS="x1 x2 x3" }
?COND_MAXONE{ COND="n" TARGETS="x1 x2 x3" }
?COND_SOME{ COND="n" TARGETS="x1 x2 x3" }
?COND_ALL{ COND="n" TARGETS="x1 x2 x3" }
```

have the expected meaning as well.

Finally we also have the following tests

```plaintext
?EXISTS{ TARGET="global/preamble/elements/simulation3D" }
?EXISTS_NOT{ TARGET="global/preamble/elements/simulation3D" }
```

for the presence or absence of an object.

As the last examples show, both the \texttt{TARGET} attribute as well as the \texttt{COND} attribute may also contain a path to the element in question, as for example:

```plaintext
?COND_ALL{ COND="/global/preamble/elements/simulation3D" TARGETS="x1 x2 x3" }
```

Here, a path beginning with a slash / is an absolute path that is searched from the root level.

**Warning:** Paths in \texttt{COND} and \texttt{TARGET} do not result into an exhaustive search of the symbol table yet!
Variables: Operators and functions

For the use with number variables, the following operators (sorted with decreasing precedence) and functions are available in the parser:

**Operators**

The following operators are supported.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>round arithmetic brackets</td>
<td>( )</td>
</tr>
<tr>
<td>power (exponentiation)</td>
<td>^</td>
</tr>
<tr>
<td>unary minus and unary plus</td>
<td>- +</td>
</tr>
<tr>
<td>arithmetic multiplication, division, remainder</td>
<td>* / %</td>
</tr>
<tr>
<td>arithmetic plus and minus</td>
<td>+ -</td>
</tr>
<tr>
<td>arithmetic comparisons</td>
<td>&lt; &lt;= &gt; &gt;=</td>
</tr>
<tr>
<td>arithmetic comparisons</td>
<td>== !=</td>
</tr>
</tbody>
</table>

Note that you have to define a separate variable beforehand if you want to use these operators in conditional statements. For instance, arithmetic comparison works as

```plaintext
$a = 3
$b = 1
$greater $a > $b
#IF $greater ...
```

**Functions**

The following mathematical functions are supported.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt()</td>
<td>square root $\sqrt{}$</td>
</tr>
<tr>
<td>cbrt()</td>
<td>cubic root $\sqrt[3]{}$</td>
</tr>
<tr>
<td>exp()</td>
<td>exponential function $\exp()$</td>
</tr>
<tr>
<td>log()</td>
<td>natural logarithm $\log$</td>
</tr>
<tr>
<td>log2()</td>
<td>decadic logarithm (base 2) $\log_2$</td>
</tr>
<tr>
<td>log10()</td>
<td>decadic logarithm (base 10) $\log_{10}$</td>
</tr>
<tr>
<td>sinh()</td>
<td>hyperbolic sine $\sinh()$</td>
</tr>
<tr>
<td>cosh()</td>
<td>hyperbolic cosine $\cosh()$</td>
</tr>
<tr>
<td>tanh()</td>
<td>hyperbolic tangent $\tanh()$</td>
</tr>
<tr>
<td>asinh()</td>
<td>inverse hyperbolic sine $\sinh^{-1}()$</td>
</tr>
<tr>
<td>acosh()</td>
<td>inverse hyperbolic cosine $\cosh^{-1}()$</td>
</tr>
<tr>
<td>atanh()</td>
<td>inverse hyperbolic tangent $\tanh^{-1}()$</td>
</tr>
<tr>
<td>erf()</td>
<td>error function $\text{erf}()$</td>
</tr>
<tr>
<td>erfc()</td>
<td>complementary error function $\text{erfc}()$</td>
</tr>
<tr>
<td>gamma()</td>
<td>Gamma function $\Gamma()$</td>
</tr>
<tr>
<td>fdm3half()</td>
<td>complete Fermi–Dirac integral $F_{-3/2}$ of order -3/2 (includes the $1/\Gamma(-1/2)$ prefactor)</td>
</tr>
<tr>
<td>fdmhalf()</td>
<td>complete Fermi–Dirac integral $F_{-1/2}$ of order -1/2 (includes the $1/\Gamma(1/2)$ prefactor)</td>
</tr>
<tr>
<td>fdzzero()</td>
<td>complete Fermi–Dirac integral $F_{0}$ of order 0 (includes the $1/\Gamma(1)$ = 1 prefactor)</td>
</tr>
<tr>
<td>fdphalf()</td>
<td>complete Fermi–Dirac integral $F_{1/2}$ of order 1/2 (includes the $1/\Gamma(3/2)$ prefactor)</td>
</tr>
<tr>
<td>fdp3half()</td>
<td>complete Fermi–Dirac integral $F_{3/2}$ of order 3/2 (includes the $1/\Gamma(5/2)$ prefactor)</td>
</tr>
<tr>
<td>abs()</td>
<td>absolute value $</td>
</tr>
</tbody>
</table>

continues on next page
Here, the results of the comparison operators and of the comparison functions `ispositive()`, ... `isnotnegative()` are either 1 for `TRUE` or 0 for `FALSE`. Please be careful when comparing the results of floating point computations and use `round()` if necessary, since e.g. $(1/3)\times 3$ has the value 0.99999999... and not 1.0.

Also please note that while conditional comments and conditional blocks consider any nonzero number as `true`, the use of positive numbers for `TRUE` is preferred, since then the arithmetic operators `+` and `*` can be used in conjunction with the comparison functions to implement logical expressions. E.g. you can define:

```
$\pi = 4 \times a t a n(1)
```

This will give 3.1415926535897932384626433832795029. You can also specify:

```
$\pi = 3.1415926535897932384626433832795029
```

Variable evaluation occurs already during parsing of the input/database file and thus before the beginning of the actual simulation. The input file after variable evaluation and the database file after the variable evaluation and possible modification by `database{}` in the input file (which are the real inputs of the simulation) are written into files

- `simulation_input.txt`
- `simulation_database.txt`.

In case of problems, or when many variables are used, it is highly recommended to review the file `simulation_input.txt` for possible mistakes. Similarly, `simulation_database.txt` will tell you (and our customer support) which values of material parameters were actually used for the simulation.

**Array variables**

Array variables can be subscripted using round brackets as (base 1):

```
$element7 = $vector(7)
```

If the array subscript is out of range, a run-time error will occur.

In addition, for the use with array variables, the following function is available:

```
dim()
```

All variables are removed from the final symbol table.

**Further remarks**

Except within comments, input files are strictly 7-bit ASCII. That is, no umlauts, diacritics, etc. in strings, names, etc. This is an inherent limitation of the parser. Command line parameters, file paths, and file names may contain all characters except \ / ? * ^ & ‘ " : ; and control characters (e.g. `newlines`). Unfortunately, e.g. on (US localized) windows, file names or file paths containing characters outside of code page 1252 (https://en.wikipedia.org/wiki/Windows-1252) may not be found or properly processed. Similar issues also may arise for other Windows localizations or for other operating systems. In order to avoid such problems, please make to sure to avoid characters outside of code page 1252 for all file names and file paths.
Advanced syntax documentation for validation files

only for developers

Validation files also offer the feature of parser-level group inheritance. Suppose, we have defined a group structure:

```plaintext
something{
  TYPE=gro
section_base{
    TYPE=gro
      ITEMS=0
      OPT=1
    x{
      TYPE=real
      OPT=1
    }
    y{
      TYPE=real
      OPT=1
    }
    z{
      TYPE=real
      OPT=1
    }
    another_group{
      TYPE=gro
        # more stuff in here
    }
section_base{
    TYPE=gro
      OPT=1
    w{
      TYPE=real
      OPT=1
    }
}
```

In this case, the group `section` is inherited from the group `section_base`.

Inheritance means here the following procedure:

- Since the file is parsed once from the top to the bottom, first the group `section_base` is generated in the symboltable of the validation file. Note that while `section_base` is abstract here (it may not occur), there is no need for a base group to abstract.
- Once the group `section` is encountered, it is constructed by deep copying all (parser-level) subgroups of `section_base` (thus `x`, `y`, `z`, `another_group` with all their content) and calling it `section`.
- Only afterwards the content of `section` itself (in this case is `w`) will be added.

Obviously, the following rules and restrictions to inheritance then apply:

- Base group and derived group must both reside in the same group or both at root level.
- Inheritance occurs in the parser, not in the validator. Thus, from a purely technical point it is also possible to inherit off group `x()` in the example above, with the derived group having a different `TYPE` (even `TYPE=GROUP`). But since all parser-items of the base group like `TYPE` or `ITEMS` are not copied, any such operation is useless for constructing a validation file and not recommended.
- For the same reason, deriving from a `TYPE=GROUP` into a different `TYPE` is not useful either.
- Next, since derivation does not copy `ITEMS`, `OPT`, `MINVAL`,... definitions of the base group, they must be defined in all derived groups as well.
- Finally, any derived group may as well be base group for other inherited groups. However, self-inheritance or inheritance from a group not encountered yet during parsing is not possible.

### 8.2.2 Keywords

All keywords (groups and attributes) for nextnano++ are described in this section. The most frequently used keywords are also gathered for a quick reference in this file.
**global{}**

Global settings simulation domain.

- `simulate1D()` / `simulate2D()` / `simulate3D()`
- `crystal_zb{}` / `crystal_wz{}`
- `substrate{}`
- `temperature`
- `temperature_dependent_bandgap`
- `temperature_dependent_lattice`
- `periodic{}`
- `magnetic_field{}`

**simulate1D{} / simulate2D{} / simulate3D{}**

Specify simulation dimension: 1D simulation along the x direction

```
global{
    simulate1D()
}
```

Specify simulation dimension: 2D simulation in the (x,y) plane

```
simulate2D()
```

Specify simulation dimension: 3D simulation

```
simulate3D()
```

**crystal_zb{} / crystal_wz{}**

Specify orientation of crystal coordinate system with respect to simulation coordinate system. For zinc blende materials there are three-digit Miller indices. The Miller indices define a plane. There exists a vector that is perpendicular to this plane, e.g. in zinc blende blende materials, the [hkl] vector is always perpendicular to the (hkl) plane. However, for wurtzite, this is not necessarily true. For instance, although the [0001] vector is perpendicular to the (0001) plane, in general does not hold that the vector that is perpendicular to the (hkil) plane is defined by [hkil]. Note: For a 1D simulation, the heterostructure is always grown along the x axis. For a 2D simulation, always the (x,y) plane is used.

**Zinc blende**

```
crystal_zb{
    x_hkl = [1, 0, 0]  # Specify (hkl) plane perpendicular to x axis
    y_hkl = [0, 1, 0]  # Specify (hkl) plane perpendicular to y axis
}
```

The x axis of the simulation coordinate system is perpendicular to this (hkl) plane of the crystal, here: (1 0 0). The y axis of the simulation coordinate system is perpendicular to this (hkl) plane of the crystal, here: (0 1 0). The Miller indices (here: (0 0 1)) for the z axis are determined automatically. For zinc blende it holds: The vector [hkil] is perpendicular to the (hkil) plane.

Another example:
x axis of simulation coordinate system is perpendicular to (3 1 1) plane of crystal coordinate system, i.e. the x axis is along [311] direction. y axis of simulation coordinate system is perpendicular to (0 -1 1) plane of crystal coordinate system, i.e. the y axis is along [0-11] direction. The Miller indices (here: [2, -3, -3]) for the z axis are determined automatically, i.e. (2 -3 -3) plane, i.e. the z axis is along [311] direction.

Wurtzite

Usually for wurtzite, the four-digit Miller-Bravais indices (h k i l) are used. We also use this notation but omit the ‘i’ because i = - h - k. The three integer values (Miller indices) that are given for x_hkl refer to a plane and not to a direction. The x direction is then the one that is perpendicular to this plane.

This vector along the x axis has indices that are in general not identical to the Miller indices in wurtzite.

In wurtzite, the c/a ratio of the two lattice constants ‘c’ and ‘a’ is important. The ideal one, \( c/a = \sqrt{8/3} = 1.63299 \ldots \) is not the one present in GaN, AlN or InN, i.e. in real materials. For the rotation of the crystal coordinate system to the simulation coordinate system, a specific c/a ratio has to be assumed. By default, we use the one of the substrate material. If you want to use the ideal c/a ratio, you have to specify rotation_c_a_ratio_use_substrate = no.

Additionally, one can specify a custom value for the c/a ratio. If no customized value is specified, rotation_c_a_ratio = sqrt(8/3) (default). The actually used rotation matrix is written to the log file.

Exactly two of these three axes have to be specified, the third one is calculated internally.
**substrate{}**

Specify substrate (and thus its lattice constant), i.e. the material on which all layers are grown. Consequently, all layers are strained with respect to the (unstrained) substrate. The substrate is also necessary to determine the symmetry properties of the $k_{||}$ integration.

```plaintext
code: nextnano Manual, Release latest
substrate{
    name *GaAs*
}
```

or for ternary $\text{Al}_x\text{Ga}_{1-x}\text{As}$ ($\text{Al}_{0.3}\text{Ga}_{0.7}\text{As}$)

```plaintext
code: nextnano Manual, Release latest
substrate{
    name *Al$_x$Ga$_{1-x}$As* # AlGaAs
    alloy_x 0.3 # x = 0.3
}
```

or for quaternary $\text{Al}_x\text{Ga}_{y}\text{In}_{1-x-y}\text{As}$ ($\text{Al}_{0.3}\text{Ga}_{0.1}\text{In}_{0.6}\text{As}$)

```plaintext
code: nextnano Manual, Release latest
substrate{
    name *Al$_x$Ga$_y$In$_{1-x-y}$As* # AlGaInAs
    alloy_x 0.3 # x = 0.3
    alloy_y 0.1 # y = 0.1
}
```

**temperature**

Specify simulation temperature. Temperature should be larger than 1 µK ($T \geq 1 \times 10^{-6}$ K)

```plaintext
code: nextnano Manual, Release latest
temperature 300 # T in Kelvin (here: 300 K)
```

**temperature_dependent_bandgap**

```plaintext
code: nextnano Manual, Release latest
temperature_dependent_bandgap yes/no # turns on/off temperature dependence of band gaps (optional, default is yes)
```

Note: The band gaps (and thus the conduction band edges) are adjusted when the temperature changes. This has the consequence that the conduction band offsets are temperature dependent whereas the valence band offsets are not temperature dependent. Due to Varshni formula a correction is calculated (if necessary interpolated) which is then added to the (interpolated) band gap. For instance:

\[
E_{\text{gap}}(T > 0 \text{ K}) = E_{\text{gap}}(T = 0 \text{ K}) + E_{\text{Varshni correction}}(T)
\]

\[
E_{\text{Varshni correction}}[A_xB_{1-x}C] = x - \frac{\alpha_{AC}T^2}{T + \beta_{AC}} + (1 - x) - \frac{\alpha_{BC}T^2}{T + \beta_{BC}} - x(1 - x) - \frac{\alpha_{bow,ABC}T^2}{T + \beta_{bow,ABC}}
\]

**temperature_dependent_lattice**

```plaintext
code: nextnano Manual, Release latest
temperature_dependent_lattice yes/no # turns on/off temperature dependence of lattice constants (optional, default is yes)
```
**periodic{}**

Note: If the grid is periodic along any of the three directions, then the Poisson and the strain equations are solved automatically with periodic boundary conditions along the appropriate directions. For the Schrödinger equation periodic boundary conditions are taken only if the quantum region extends over the whole simulation region along the relevant direction. Objects (e.g. circles, rectangles, ...) extending over an edge of the simulation region will not automatically be continued on the opposite side. Periodic boundaries are not implemented for magnetic fields.

```
periodic{               # (optional)
  x [yes/no]          # periodic along x direction (default is no) (2D or 3D)
  y [yes/no]          # periodic along y direction (default is no) (2D or 3D)
  z [yes/no]          # periodic along z direction (default is no) (3D only)
}
```

Note: For all nextnano versions older than 2019-04-16, the `periodic()` flag was specified within the `grid()` section and not within the `global()` section.

**magnetic_field{}**

Specify magnetic field.

- **strength**
  - **value** any float $\geq 0.0$
  - **default** 0.0

  magnetic flux density $B$ in units of Tesla $[T] = [Vs/m^2]$

- **direction**
  - **value** integer array of dimension 3
  - **default**

  (only 3D) orientation of magnetic field vector with respect to $(x,y,z)$ simulation coordinate system

  Note: In 2D, the specifier `direction` must not be present because the direction is automatically set internally to be perpendicular to the simulation plane in 2D, i.e. $(x,y)$ plane $\Rightarrow [0, 0, 1]$.

  In 1D (Zeeman splitting of energy levels only), the specifier `direction` must not be present because the direction is parallel to the simulation direction $(x)$.

  Defining `magnetic_field{}` in the input file results into a different (slower) solver being called. Thus, omit this section altogether in case no magnetic field is needed.

  Note: The Hall effect for the drift-diffusion current calculation is not implemented yet.

```
magnetic_field{
  strength 5.3 # magnetic flux density $B$ in units of Tesla
  --[T] = [Vs/m^2]
  direction [3, 1, 1] # (only 3D) orientation of magnetic field
  --vector with respect
  --to $(x,y,z)$ simulation coordinate system
  --here: [311]
}
```
For single-band: For a normal single-band calculation, all eigenstates are internally treated to be two-fold spin degenerate (e.g. when calculating the quantum density). In contrast, if `magnetic_field()` is defined, the Pauli equation is solved in all 2D and 3D single-band quantum regions instead of the single-band Schrödinger equation. So the states are no longer two-fold spin degenerate (unless the magnetic field is zero). Note that even if the magnetic field strength is explicitly set to zero, still the Pauli equation is solved in which case both spin eigenfunctions will be calculated although they have the same energy. Thus, to maximize speed in the absence of magnetic fields, leave `magnetic_field()` undefined. Then the dimension of the matrix from which the eigenfunctions are calculated has only half the size (and thus only half of the eigenfunctions have to be calculated), and it is is real instead of Hermitian.

Currently, the magnetic field is input to the single-band and the multi-band k.p Hamiltonian but is not input to the drift-diffusion current equation. Thus the classical Hall effect is not taken into account. As the magnetic field modifies the eigenenergies and probability densities, and thus the quantum density, the current will be different compared to the zero-field case because the current is proportional to the density.

**structure**

definition of device structure (including doping{})

- **Output definitions**
  - `output_region_index[]`
  - `output_material_index[]`
  - `output_contact_index[]`
  - `output_user_index[]`
  - `output_alloy_composition[]`
  - `output_impurities[]`
  - `output_generation[]`

- **Structure definitions**
  - `Region object shapes`
  - `Contact definitions`
  - `Material definitions`
  - `doping and generation rate profile`

- **Repeating regions**
  - `array_x[], array_y[], array_z[]`
  - `repeat_profiles`
  - `array2_x[], array2_y[], array2_z[]`
Output definitions

The following syntaxes specifies the output file. These are put under \texttt{structure{}}.

\textbf{output\_region\_index{}}

output (last) region number and (last) material region number for each grid point

\begin{verbatim}
output\_region\_index{}  # output (last) region number and (last) material region number for each grid point
  boxes [yes/no]      # (optional) For each grid point, in 1D two points are printed out to mimic
                    # abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)
\end{verbatim}

\textbf{output\_material\_index{}}

output material number according to material database for each grid point

\begin{verbatim}
output\_material\_index{}  # output material number according to material database for each grid point
  boxes [yes/no]      # (optional)
\end{verbatim}

\textbf{output\_contact\_index{}}

output contact number for each grid point

\begin{verbatim}
output\_contact\_index{}  # output contact number for each grid point
  boxes [yes/no]      # (optional)
\end{verbatim}

\textbf{output\_user\_index{}}

output (last) user defined index for each grid point

\begin{verbatim}
output\_user\_index{}  # output (last) user defined index for each grid point
  boxes [yes/no]      # (optional)
\end{verbatim}

\begin{enumerate}
\item The user index array is preinitialized with value 0 everywhere.
\item The regions are processed in order of their definition, and only regions which have a user index defined are considered, (i.e. regions without user index do not affect the user index array).
\end{enumerate}

Hints: Set a user index to e.g. 0 if you want a region to e.g. merely delete the user index inside. And use variables together with expressions such as $\text{index} = \text{index} + 1$ to generate consecutive index values from region to region.

Download example input file here.

This example shows how to create an incremental enumeration of regions using variables, and also how to keep identical number across clusters of regions.
output_alloy_composition()

output alloy composition for each grid point

```plaintext
output_alloy_composition{
  # output alloy composition for each grid
  boxes [yes/no] # (optional)
}
```

output_impurities()

output doping concentration for each grid point in units of \([10^{18}/cm^3]\)

```plaintext
output_impurities{
  # output doping concentration for each grid
  # point in units of [10^18/cm3]
  boxes [yes/no] # (optional)
}
```

output_generation()

output generation rate for each grid point in units of \([10^{18}/(cm^3 s)]\)

```plaintext
output_generation{
  # output generation rate for each grid
  # point in units of \([10^18/(cm^3 s)]\)
  boxes [yes/no] # (optional)
}
```
Structure definitions

Every region needs to have a certain shape, which can be defined by several objects. It consists of a certain material and/or contact, and it can have a doping profile.

Any subsequently defined region overwrites previously defined ones in the overlapping area. For exclusive properties such as material and contact, this implies a substitution of the old value.

Concerning doping, the new profile is added to any previously defined one.

Geometric objects may also be defined such that they are partially, mostly, or completely outside of the simulation region. Only the parts of structures which are inside of the simulation region will be used, everything else is ignored.

The following structures are supported. These are put under `structure{ region{} }`.

Region object shapes

- 1D:
  - `line{}`

- 2D:
  - `rectangle{}`
  - `circle{}`
  - `triangle{}`
  - `trapezoid{}`
  - `polygon{}`
  - `regular_polygon{}`
  - `hexagon{}`
  - `semiellipse{}`

- 3D:
  - `cuboid{}`
  - `sphere{}`
  - `cylinder{}`
  - `obelisk{}`
  - `hexagon_obelisk{}`
  - `cone{}`
  - `semiellipsoid{}`
  - `regular_prism{}`
  - `hexagonal_prism{}`
  - `polygonal_prism{}`
  - `pyramid{}`
  - `regular_pyramid{}`
  - `hexagonal_pyramid{}`
  - `polygonal_pyramid{}`
line{}

1D object. a line from start to end point along the specified direction

Example

```cpp
line{
  # 1D object
  x [10.0, 20.0] # a line from 10 nm to 20 nm along x direction
}
```

rectangle{}

2D object, a rectangle defined by two lines along the x and y directions

Example

```cpp
rectangle{
  # 2D object, a rectangle defined by two lines along the x and y directions
  x [10.0, 20.0] # a line from 10 nm to 20 nm along x direction
  y [0.0, 5.0] # a line from 0 nm to 5 nm along y direction
}
```

cuboid{}

3D object, a cuboid defined by three lines along the x, y and z directions

Example

```cpp
cuboid{
  # 3D object, a cuboid defined by three lines along the x, y and z directions
  x [10.0, 20.0] # a line from 10 nm to 20 nm along x direction
  y [0.0, 5.0] # a line from 0 nm to 5 nm along y direction
  z [0.0, 5.0] # a line from 0 nm to 5 nm along z direction
}
```

circle{}

2D object, a circle is defined by its center and radius

Example

```cpp
circle{
  # 2D object, a circle is defined by its center and radius
  center( x 10.5 y 14.0 ) # same as for regular polygon
  radius 10.0 # radius
}
```
sphere{}

3D object, a sphere is defined by its center and radius

Example

```c
sphere{
  # 3D object, a sphere is defined by its center and radius
  center{ x = 10.5  y = 14.0  z = 1.0 } # similar as for circle
  radius = 10.0 # radius
}
```

cylinder{}

3D object, e.g. a cylinder with a freely oriented axis

Example

```c
cylinder{
  # 3D object, e.g. a cylinder with a freely oriented axis
  axis_start = [50.0, 50.0, 30.0] # coordinates of starting point of cylinder axis
  axis_end = [50.0, 50.0, 60.0] # coordinates of ending point of cylinder axis
  radius = 20.0 # radius of cylinder
}
```

Download example input file here.
trapezoid()

2D object e.g. a simple trapezoid along the x axis

Example

```
trapezoid{
  # 2D object e.g. a simple trapezoid along the x axis
  → along the x axis
  base_x = [ 5, 15 ] # base line extends in x direction
  → from 5 to 15 nm
  base_y = [25, 25] # base line has a constant y
  → coordinate y = 25 nm
  top_x = [ 8, 12 ] # top line extends in x direction
  → from 8 to 12 nm
  top_y = [30, 30] # top line has a constant y
  → coordinate y = 30 nm
}
```

Note: Exactly one of the elements base_x and base_y has to be set by two equal numbers to define the base line. The same holds for top_x and top_y to define the top line.

obelisk()

3D object, e.g. an obelisk parallel to the (x,y) plane with top below bottom

Example

```
obelisk{
  # 3D object, e.g. an obelisk parallel to the (x,y) plane with top below bottom
  → parallel to the (x,y) plane with top below bottom
  base_x = [ 11, 19 ] # extension of base plane in x
  → direction, i.e. from 11 to 19 nm.
  base_y = [ 9, 21 ] # extension of base plane in y
  → direction, i.e. from 9 to 21 nm.
  base_z = [10, 16] # base plane at z = 10 nm
  top_x = [12, 18] # extension of top plane in x
  → direction, i.e. from 12 to 18 nm.
  top_y = [13, 19] # extension of top plane in y
  → direction, i.e. from 13 to 19 nm.
  top_z = [22, 22] # top plane at z = 22 nm
}
```

Note: Exactly one of the elements base_x, base_y and base_z has to be set by two equal numbers to define the base plane. The same holds for top_x, top_y and top_z to define the top line.

Download example input file here.
hexagon_obelisk{}

3D object, an obelisk with its base and top planes given by hexagons

Example

```
hexagon_obelisk{
  # 3D object, an obelisk with its base and top planes given by hexagons
  --and top planes given by hexagons
  --possible orientations of the hexagon within the rectangularly defined planes
  permute [yes|no] # (optional) switch between two possible orientations of the hexagon within the rectangularly defined planes
}
```

semiellipse{}

2D object, e.g. a simple semiellipse along the x axis

Example

```
semiellipse{
  # 2D object, e.g. a simple semiellipse along the x axis
  --semiellipse along the x axis
  base_x [45, 55] # extension of base plane in x direction, i.e. from 45 to 55 nm.
  base_y [5, 5] # base line at y = 5 nm
  top [50, 15] # top coordinate of the semiellipse
  --(x,y) = (50,15) in units of [nm]
}
```

Note: Exactly one of the elements base_x, and base_y has to be set by two equal numbers to define the base line.
**semiellipsoid{}**

3D object, e.g. a semiellipsoid parallel to the (y,z) plane with top below bottom

**Example**

```
semiellipsoid{} # 3D object, e.g. a semiellipsoid
→ parallel to the (x, y) plane with top below bottom
base_x  \[9, 21\] # extension of base plane in x
→ direction, i.e. from 9 to 21 nm.
base_y \[11, 20\] # extension of base plane in y
→ direction, i.e. from 11 to 20 nm.
base_z \[10, 10\] # base plane at z = 10 nm
top \[11,15,24\] # top coordinate of the
→ semiellipsoid (x,y,z) = (11,15,24) in units of [nm]
```

**Note:** Exactly one of the elements `base_x`, `base_y`, and `base_z` has to be set by two equal numbers to define the base plane.

Download example input file [here](#).

---

**cone{}**

3D object, e.g. a cone parallel to the (x,z) plane

**Example**

```
cone{} # 3D object, e.g. a cone parallel to
→ the (x,z) plane
base_x \[5, 20\] # extension of base plane in x
→ direction, i.e. from 5 to 20 nm.
base_y \[20, 20\] # base plane at y = 20 nm
```

(continues on next page)
Note: Exactly one of the elements base_x, base_y, and base_z has to be set by two equal numbers to define the base plane.

triangle()

2D object, a triangle defined by its 3 vertices

Example

definition{ # 2D object, a triangle defined by its 3 vertices. vertex{ x 10.5 y 14.0 } # a vertex P is defined by its x and y coordinates: P=(x,y). vertex{ x 0.0 y 0.0 } # vertex{ x 5.0 y 10.0 } # }

polygon()

2D object, a polygon defined by its vertices. If the first and the last defined vertex are not identical, then they are joined with a line.

Example

definition{ # 2D object, a polygon defined by its vertices. If the first and the last defined vertex are not identical, then they are joined with a line. vertex{ x 10.5 y 14.0 } # a vertex P is defined by its x and y coordinates: P=(x,y). Multiple vertices can and must be defined for a polygon. # Vertices must be ordered either clockwise or counterclockwise, otherwise the behavior during structure generation will be undefined. }

(base_z [7, 19] # extension of base plane in z direction, i.e. from 7 to 19 nm. top [10, 30, 11] # top coordinate of the cone (x,y,z). diminution 0.0 # (optional) minimum value is 0.0 (i.e. cone), maximum value is 1.0 (i.e. cylinder). # diminution = 0.5 corresponds to "half diameter of base diameter", default is 0.0 (i.e. cone). )
**polygonal_prism**

3D object (= 2D polygon with extension into the perpendicular direction; vertices define the circumference of the prism.)

Example

```plaintext
polygonal_prism{
  # 3D object (= 2D polygon with extension into the perpendicular direction; vertices define the circumference of the prism.)
  z [0, 10] # define the extent in the desired height direction. Here: Height is defined with respect to z direction.
  vertex{ x 10.5 y 14.0 } # a vertex P is defined by its x and y coordinates: P=(x,y). Multiple vertices can and must be defined for a polygon.
  # Vertices must be ordered either clockwise or counterclockwise, otherwise the behavior during structure generation will be undefined.
  axis [0, 1, 1] # (optional) inclination (shear) of prism structure
  # (Obviously, cyclic permutation of x, y, z are possible.)
}
```

**regular_polygon**

2D object, a polygon with equal angles and equal side lengths. It is defined by its center, one vertex and the number of facets.

Example

```plaintext
regular_polygon{
  # 2D object, a polygon with equal angles and equal side lengths. It is defined by its center, one vertex and the number of facets.
  center{ x 10.5 y 14.0 } # The center point M is defined by its x and y coordinates: M=(x,y).
  corner{ x 20.0 y 30.0 } # A corner vertex P is defined by its x and y coordinates: P=(x,y). Only one corner must be specified. By modifying the corner coordinates the whole polygon can easily be rotated around its center.
  number_of_facets 7 # number of facets (= number of vertices), must be >= 3
}
```

**regular_prism**

3D object (= 2D regular_polygon with extension into the perpendicular direction; center and/or corner define the circumference of the prism.)

Example

```plaintext
regular_prism{
  # 3D object (= 2D regular_polygon with extension into the perpendicular direction; center and/or corner define the circumference of the prism.)
  z [0, 10] # define the extent in the desired height direction. Here: Height is defined with respect to z direction.
  center{ x 10.5 y 14.0 } # The center point M is defined by its x and y coordinates: M=(x,y).
}
```

(continues on next page)
corner( x 20.0 y 30.0 ) # A corner vertex P is defined by
--its x and y coordinates: P=(x,y). Only one corner must be specified. By
--modifying the corner coordinates the whole polygon can easily be
--rotated around its center.

number_of_side_facets 7 # number of side facets (= number
--of vertices), must be >= 3
axis [0, 1, 1] # (optional) inclination (shear)
--of prism structure
--of x, y, z are possible.)
}

hexagon{}

2D object, a polygon with equal angles and equal side lengths and 6 facets. It is defined by its center
and one corner vertex.

Example

hexagon{ # 2D object, a polygon with equal
--angles and equal side lengths and 6 facets. It is defined by its center
--and one corner vertex.
center( x 10.5 y 14.0 ) # same as for regular_polygon
corner( x 20.0 y 30.0 ) # same as for regular_polygon
}

hexagonal_prism{}

3D object (= 2D hexagon with extension into the perpendicular direction; center and/or corner define
the circumference of the prism.)

Example

hexagonal_prism{ # 3D object (= 2D hexagon with
--extension into the perpendicular direction; center and/or corner define
--the circumference of the prism.)
z [0, 10] # define the extent in the
--desired height direction. Here: Height is defined with respect to z
--direction.
center( x 10.5 y 14.0 ) # same as for regular_polygon
corner( x 20.0 y 30.0 ) # same as for regular_polygon
axis [0, 1, 1] # (optional) inclination (shear)
--of prism structure
--of x, y, z are possible.)
}

Note: Per default, all prisms (polygonal_prism, regular_prism, hexagonal_prism) are assumed
to extend along the respective layer thickness direction (i.e. normal to the defining coordinate plane). But, using
the axis vector, an arbitrary axis (inclination) direction for the prism can be defined in the simulation system.
The axis vector does not need to be normalized, however, its orientation defines which side of the prism layer is
the base to be used as reference for the inclination. For example,

regular_prism{
z [50, -70] # automatically reordered to [-70, 50]
center( x 10 y 10 )
}
defines a regular octahedral prism extending primarily in the z direction (end surfaces are x-y planes at \( z = -70 \) and \( z = +50 \)). Since the axis points upwards in z direction \( (z = 120) \), the base surface to be taken as reference is the lower x-y plane at \( z = -70 \). There, the octagon center is at \( \{ x = 10 \  y = 10 \} \) with an octagon corner at \( \{ x = 30 \  y = 40 \} \) With the axis vector defined as above, we then find for the x-y plane at \( z = +50 \)

- the octagon center at \( \{ x = 10+15 \  y = 10+25 \} \) and
- the octagon corner at \( \{ x = 30+15 \  y = 40+25 \} \).

In analogy to polygon, we provide pyramidal structures.

### polygonal_pyramid()

**Example**

```plaintext
class polygonal_pyramid{
  # 3D object
  z = [-70, 70]  # same as for polygonal_prism
  vertex{ x = 10.5  y = 14.0 }  # a vertex P is defined by its x and y coordinates: P=(x,y). Multiple vertices can and must be defined for a polygon.  # Vertices must be ordered either -clockwise or counterclockwise, otherwise the behavior during structure generation will be undefined.  
  apex{ x = 10  y = 10  z = 120 }
}
```

### regular_pyramid()

**Example**

```plaintext
class regular_pyramid{
  # 3D object
  z = [-70, 70]  # same as for regular_prism
  center{ x = 10  y = 10 }  # same as for regular_prism
  corner{ x = 70  y = 70 }  # same as for regular_prism
  number_of_side_facets = 8  # same as for regular_prism
  apex{ x = 10  y = 10  z = 120 }
}
```

### hexagonal_pyramid()

**Example**

```plaintext
class hexagonal_pyramid{
  # 3D object
  z = [-70, 70]  # same as for hexagonal_prism
  center{ x = 10  y = 10 }  # same as for hexagonal_prism
  corner{ x = 70  y = 70 }  # same as for hexagonal_prism
  apex{ x = 10  y = 10  z = 120 }
}
```
Note: Similar to the prismatic structures, use x, y, and z at the beginning of the respective primitive to define the extent in the desired height direction, use vertex, center, and/or corner to define the circumference of the base of the pyramid, and apex to define the position of the apex of the pyramid.

Note that, for polygonal_pyramid (as for polygon), the vertices must be ordered either clockwise or counterclockwise, otherwise the behavior during structure generation will be undefined.

Also note that if the apex is located outside of the interval defined by x, y, or z at the beginning in the height direction, the pyramid will be truncated. Also, the pyramid will point upwards if the apex is above the center of said interval (and the lower plane is used as base), and will point downwards if the apex is below the center (and the upper plane is used as base). And in case a symmetric regular pyramid is desired, please make sure to laterally align the apex with the center point.

For example

```
regular_pyramid{
    z [70, -70]
    center{x 10 y 10}
    corner{x 70 y 70}
    number_of_side_facets 8
    apex{x 10 y 10 z 120}
}
```

defines a regular octahedral pyramid with base at z = -70, centered there at \{ x = 10 y = 10 \} and a corner there at \{ x = 70 y = 70 \}. The apex of the pyramid would be at \{ x = 10 y = 10 z = 120 \}, making the structure rotationally symmetric, except that the pyramid is truncated at z = +70. Thus, a rotationally symmetric truncated octahedral pyramid has been defined.

### pyramid{}

3D object, e.g. a pyramid with 4 freely defined corner points

**Example**

```
pyramid{
    # 3D object, e.g. a pyramid
    --with 4 freely defined corner points
    point1 { 50.0, 20.0, 30.0 } # coordinates of first point of
    --pyramid
    point2 { 50.0, 50.0, 80.0 } # coordinates of second point of
    --of pyramid
    point3 { 80.0, 50.0, 50.0 } # coordinates of third point of
    --pyramid
    point4 { 50.0, 80.0, 30.0 } # coordinates of fourth point of
    --of pyramid
}
```

### Contact definitions

#### contact{}

**Example**

```
contact{
    # {optional}
    name "source" # This region will be defined as a
    --contact. In this case the contact is called "source".
}
```
Material definitions

Binary, ternary and quaternary materials are possible, with several choices of alloy functions. Depending on the dimension of the simulation domain, different options are available.

**binary{}**

binary material

**Example**

```
binary{
  name "GaAs"  # binary material name for this region
}
```

**ternary_constant{}**

ternary material with constant alloy profile

**Example**

```
ternary_constant{
  name "Al(x)Ga(1-x)As"  # ternary material name for this region with constant alloy profile
  alloy_x 0.2  # x content of the alloy (minimum value is 0.0, maximum value is 1.0)
}
```

**ternary_linear{}**

ternary material name which varies linearly along the line from start to end point

**Example**

```
ternary_linear{
  name "In(x)Al(1-x)As"  # ternary material name for this region with linear alloy profile
  alloy_x [0.8, 0.2]  # start and end value of x content
  x [75.0, 125.0]  # x coordinates of start and end point [nm]
  y [10.0, 20.0]  # y coordinates of start and end point [nm] (2D or 3D only)
  z [10.0, 20.0]  # z coordinates of start and end point [nm] (3D only)
  # This defines an alloy profile, which varies linearly along the line from the point (75,10,10) to the point (125,20,20)
}
```
and stays constant in the perpendicular planes.

(3D quantum dot)

ternary_pyramid{}

ternary material name with pyramidal alloy profile

Example

```python
ternary_pyramid(# (e.g. for InGaAs quantum dots) starting point and direction (3D only)
   name = "In[ Ga] As" # ternary material name for this region with pyramidal alloy profile
   alloy_x = [0.28, 0.80] # c_{min} and c_{max} value of x content (minimum value is 0.0, maximum value is 1.0)
   # vary alloy concentration from apex/axis x = 0.80 (In0.80Ga0.20As)
   # to plane through apex perpendicular to axis x = 0.28 (In0.28Ga0.72As) (see figure below)
   # apex located at point (20.0, 20.0, 11.0) (top of inverted pyramid)
   # direction of center axis (0, 0, 1), i.e. along z axis
   # the same pyramidal profile as (0,0,-1).
}
```

Note: The indium content is given by the following formula, which considers an additional lateral variation of the indium content:

\[ c = c_{min} + (c_{max} - c_{min}) \cos^2 \phi \]

where \( \phi \) is the angle to the center axis. The formula is based on the model proposed by Tersoff (N. Liu et al., PRL 84, 334 (2000)). For simplicity the alloy profile is still isotropic around the center axis of the quantum dot. The indium content depends solely on the angle to the center axis, with high indium content for small angles as indicated by the light regions in the figure shown below.

(3D quantum dot)
ternary_trumpet()

ternary material with “trumpet” alloy profile

Example

```plaintext
ternary_trumpet{
    # (e.g. for InGaAs quantum dots)
    # ternary material name for this region with "trumpet" alloy profile
    alloy_x [0.2, 0.5] # :math:`c_{\text{min}}` and :math:`c_{\text{max}}`
    x [20.0, 0] # x coordinate of apex and x component of axis direction [nm]
    y [20.0, 0] # y coordinate of apex and y component of axis direction [nm]
    z [11.0, 1] # z coordinate of apex and z component of axis direction [nm]
    # apex located at point (20.0,20.0,11.0) (top of inverted pyramid)
    # direction of center axis (0,0,1), i.e. along z axis
    # The profile is symmetric with respect to the inverse of the direction of the center axis,
    # i.e. (0,0,1) will lead to the same trumpet profile as (0,0,-1).
    z0 1.25 # parameter to vary the shape of the alloy profile [minimum value is 1e-10]
    rho0 0.6 # parameter to vary the shape of the alloy profile [minimum value is 1e-10]
}
```

**Note:** The indium content is given by the formula:

\[
c = c_{\text{min}} + (c_{\text{max}} - c_{\text{min}}) \exp\left[-\sqrt{x^2 + y^2} \exp\left(-\frac{z}{z_0}\right)\right]/\rho_0
\]

The formula is based on the more refined model proposed by Migliorato (M.A. Migliorato et al., PRB 65, 115316 (2002)). This profile resembles the horn of a trumpet and is thus called ‘trumpet’. The maximum indium concentration is on the center axis of the quantum dot. The parameters \(z_0\) and \(\rho_0\) can be used to vary the shape of the alloy profile while keeping the average indium content fixed.

(3D quantum dot)
ternary_import{}

ternary material which uses imported alloy profile

Example

```plaintext
ternary_import{
  name = "In(x)Al(1-x)As" # ternary material name.
  -- for this region which uses imported alloy profile
  import_from = "import_alloy_profile1D" # reference to imported data in "import{}. The imported profile must have exactly one data component (x).
}
```

quaternary_import{}

quaternary material which uses imported alloy profile

Example

```plaintext
quaternary_import{
  name = "Al(x)Ga(y)In(1-x-y)As" # quaternary material name.
  -- name for this region which uses imported alloy profile
  import_from = "import_alloy_profile1D" # reference to imported data in import{}. The imported profile must have exactly two data components (x,y).
}
```

quinternary_import{}

quinternary material which uses imported alloy profile

Example

```plaintext
quinternary_import{
  # analogous for quaternaries:
}
```

quaternary_constant{}

quaternary material with constant alloy profile

Example

```plaintext
quaternary_constant{
  name = "Al(x)Ga(y)In(1-x-y)As" # quaternary material name.
  -- for this region with constant alloy profile
  alloy_x = 0.2 # x content of the alloy.
  -- (minimum value is 0.0, maximum value is 1.0)
  alloy_y = 0.5 # y content of the alloy.
  -- (minimum value is 0.0, maximum value is 1.0)
}
```

Note: For quaternaries of type AxByC1-x-yH, \( x + y \leq 1 \) must hold.
The interpolation of $AxByC1-x-yH$ is done according to eq. (E.10) in PhD thesis of T. Zibold apart from changes in sign of bowing parameters. The interpolation of $AxB1-xCyD1-y$ is done according to eq. (E.15) in PhD thesis of T. Zibold apart from changes in sign of bowing parameters.

**quaternary_linear{}**

quaternary material with linear alloy profile

**Example**

```plaintext
quaternary_linear{
  name  "Al(Ga(1-x)In(1-x-y)As)" # quaternary material name for this region with linear alloy profile
  alloy_x [0.2, 0.5] # start and end value of x
  --content (minimum value is 0.0, maximum value is 1.0)
  alloy_y [0.1, 0.3] # start and end value of y
  --content (minimum value is 0.0, maximum value is 1.0)
  x [20.0, 20.0] # x coordinates of start and end point [nm]
  y [20.0, 20.0] # y coordinates of start and end point [nm]
  z [11.0, 20.0] # z coordinates of start and end point [nm]
} quaternary_pyramid{}
```

quaternary material with pyramid alloy profile

**Example**

```plaintext
quaternary_pyramid{
  name  "Al(Ga(1-x)In(1-x-y)As)" # quaternary material name for this region with pyramidal alloy profile
  alloy_x [0.2, 0.5] # minimum and maximum value of x content
  alloy_y [0.1, 0.3] # minimum and maximum value of y content
  x [20.0, 0] # x coordinate of apex and x component of axis direction [nm]
  y [20.0, 0] # y coordinate of apex and y component of axis direction [nm]
  z [11.0, 1] # z coordinate of apex and z component of axis direction [nm]
  x [20.0, 11.0] (top of inverted pyramid) # apex located at point (20.0, 11.0), i.e. along z axis # direction of center axis (0, 0, 1), i.e. along z axis # The profile is symmetric with respect to the inverse of the direction of the center axis, i.e. (0,0,-1) will lead to the same pyramidal profile as (0,0,1).
} quaternary_linear{}
```
quaternary_trumpet{}

quaternary material with “trumpet” alloy profile

**Example**

```plaintext
quaternary_trumpet{
  "Al(\(x\))Ga(\(y\))In(1-x-y)As" # quaternary material name for this region with "trumpet" alloy profile
  alloy_x [0.2, 0.5] # minimum and maximum value of x content
  alloy_y [0.1, 0.3] # minimum and maximum value of y content
  x [20.0, 0] # x coordinate of apex and x component of axis direction [nm]
  y [20.0, 0] # y coordinate of apex and y component of axis direction [nm]
  z [11.0, 1] # z coordinate of apex and z component of axis direction [nm]
  0) (top of inverted pyramid) # direction of center axis (0,0,1), i.e. along z axis
  1), i.e. along z axis # The profile is symmetric with respect to the inverse of the direction of the center axis,
  same trumpet profile as (0,0,-1). # i.e. (0,0,1) will lead to the same trumpet profile as (0,0,-1).
  z0 1.25 # parameter to vary the shape of the alloy profile (minimum value is 1e-10)
  rho0 0.6 # parameter to vary the shape of the alloy profile (minimum value is 1e-10)
}
```

analogous for quinternaries:

quinternary_constant{}
quinternary_linear{}
quinternary_pyramid{}
quinternary_trumpet{}

**integrate()**

spatial integration of profiles in the region.

**Example**

```plaintext
integrate{
  profiles in this region. # spatial integration of profiles in this region.
  electron_density() # integrate electron density.
  hole_density() # integrate hole density.
  piezo_density() # integrate piezo charge density.
  pyro_density() # integrate pyro charge density.
  polarization_density() # integrate the polarization.
  charges density: (= piezo + pyro)
}
```

(continues on next page)
Note: Due to the finite discretization of the space, it is advised to define the region for integration slightly larger than the region of actual interest, especially if there is a significantly high density at the boundaries of the integration region.

### doping and generation rate profile

In each region, we can specify the doping profile and generation rate profile. These features are explained below:

```plaintext
structure{ doping{} generation{} }

doping{}
```

Specifications that define information on doping.

- **Syntaxes**
  - Specifications of doping profile
  - Print out
  - Remove
- **Example**
  - 1D
  - 3D

Note: Information on impurities - which can be donors or acceptors or fixed charge - is specified here: *impurities[]*

#### Syntaxes

**Specifications of doping profile**

The doping profile is assigned to a certain region. The following syntaxes are put under `structure{ region{ doping{} } }`

- constant
- linear
- gaussian1D
- gaussian2D
- gaussian3D
• import (import doping profile from external file)

  constant  constant doping profile over the region

  Example

  ```plaintext
  constant{
    name  "Si"  # name of impurity
    conc  1.0e18  # doping concentration [cm-3]
    # (applies to 1D, 2D and 3D)
    add   yes  # (optional) yes or no (default = yes)
  }
  ```

  Note: the name of impurities must be specified in the same input file in impurities{}.

linear  linearly varying doping profile along the line from start to end point

  Example

  ```plaintext
  linear{
    name  "n-Si-in-GaAs"  # name of impurity
    conc  [1e18,2e18]  # start and end value of doping [cm-3]
    x [50.0,100.0]  # x coordinates of start and end point
    y [50.0,100.0]  # y coordinates of start and end point (2D or 3D only)
    z [50.0,100.0]  # z coordinates of start and end point (3D only)
    # This defines a doping profile, which varies linearly along the line from the point (50,50,50)
    # to the point (100,100,100) and stays constant in the perpendicular planes.
    add   yes  # (optional) yes or no (default = yes)
  }
  ```

gaussian1D  Gaussian distribution function in one direction, constant in perpendicular directions

  Example

  ```plaintext
  gaussian1D{
    # Gaussian distribution function in one direction, constant in perpendicular directions
    name  "p-B-in-Si"  # name of impurity
    conc  1.0e18  # maximum of doping concentration [cm-3]
    dose  1e12  # dose of implant [cm-2] (integrated density of gaussian function), typical ranges are from 1e11 to 1e16.
    # Either dose or conc has to be specified, but not both simultaneously.
    # conc = dose / ( SQRT(2*pi) * sigma_x )
    x 50.0  # x coordinate of Gauss center (ion's projected range Rp, i.e. the depth where most ions stop) [nm]
    sigma_x 5.0  # root mean square deviation in x [nm]
    direction  "statistical fluctuation of Rp"  # (3D only)
    y  # (2D or 3D only)
    sigma_y  # (3D only)
    z  # (3D only)
  }
  ```

(continues on next page)
gaussian2D  Gaussian distribution function in two directions, constant in perpendicular direction (2D or 3D only)

Example

```plaintext
gaussian2D{
  # Gaussian distribution function in two directions, constant in perpendicular direction (2D or 3D only)
  name  "p-B-in-Si"  # name of impurity
  conc  1.0e18        # maximum of doping concentration
  dose  1.0           # dose of implant [cm-1]
  (integrated density of 2D gaussian function)
  # Either dose or conc has to be specified, but not both simultaneously.
  x     50.0          # x coordinate of Gauss center [nm]
  sigma_x 5.0         # root mean square deviation in x
  direction  [nm]
  y     50.0          # y coordinate of Gauss center [nm]
  sigma_y 5.0         # root mean square deviation in y
  direction  [nm]
  z     50.0          # (3D only)
  sigma_z 5.0         # (optionally) the appropriate standard deviations (sigma) have to be specified.
  add  yes            # (optional) yes or no (default = yes)
}
```

gaussian3D  Gaussian distribution function in three directions (3D only)

Example

```plaintext
gaussian3D{
  # Gaussian distribution function in three directions (3D only)
  name  "p-B-in-Si"  # name of impurity
  conc  1.0e18        # maximum of doping concentration
  dose  1.0           # dose of implant [cm-3]
  (integrated density of 3D gaussian function)
  x     50.0          # x coordinate of Gauss center [nm]
  sigma_x 5.0         # root mean square deviation in x
  direction  [nm]
  y     50.0          # y coordinate of Gauss center [nm]
  sigma_y 5.0         # root mean square deviation in y
  direction  [nm]
  z     50.0          # z coordinate of Gauss center [nm]
  sigma_z 5.0         # root mean square deviation in z
  direction  [nm]
  # All three x, y, z and the appropriate standard deviations (sigma) have to be specified.
}
```

Note: This profile corresponds to LSS theory (Lindhard, Scharff, Schiott theory) for doping - Gaussian distribution of ion implantation.
add \[\text{yes}\] # (optional) yes or no (default = \[\rightarrow\yes\])

import import generation profile from external file

import{
  # import generation
  \[\rightarrow\text{profile from external file}\]
  \[\rightarrow\text{name} \text{\"p-B-in-Si\"} \# impurity name uses\]
  \[\rightarrow\text{imported alloy profile}\]
  \[\rightarrow\text{import from} \text{\"import_doping_profile\"} \# reference to imported\]
  \[\rightarrow\text{data in import{}. The file being imported must have exactly one}\]
  \[\rightarrow\text{data component.}\]
}

Print out

These doping profile can be printed out by output_impurities{} keyword under structure{}:

output_impurities{}

\begin{verbatim}
structure{
  output_impurities{
    # output doping concentration for
    each grid point in units of [10^18/(cm^3)]
    boxes \[\rightarrow\text{yes/no}\] # (optional)
  }
}
\end{verbatim}

Remove

It is also possible to remove a doping from a specific region.

remove{}

\begin{verbatim}
structure{
  region{
    doping{ remove{ }} \# remove doping from this
    \[\rightarrow\text{region}, to keep certain regions free from doping.}\]
  }
  # region
}
\end{verbatim}

Note: doping{} and generation{} is always additive per default (add = yes) (unless import is different),
i.e. each profile adds to the already existing dopants/fixed charges/generation at a given point. At the same
time, using remove{}, all species of the already existing doping or generation concentrations can be removed.
However, there is also the problem that remove{} removes all species of dopants/fixed charges at a given point.
Thus, removing e.g. only donors but not acceptors is difficult. This problem is solved by the new “add = yes/
no” flag, which the user can specify for each profile (and thus for the species of that profile), whether the profile
should add to (which is the default) or replace the already existing concentration of the profile species.

For import{}, this flag has not been implemented yet.
Example

1D

Figure 8.2.2.1 shows two impurity profiles based on LSS theory. More information can be found in this tutorial: Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider’s code.

3D

Figure 8.2.2.2 shows a 3D generation profile that is defined inside a 20 nm x 20 nm x 50 nm cube where the 50 nm are the z direction. The generation rate profile is homogeneous with respect to the (x,y) plane, it only varies along the z direction.

The generation rate profile is constant between z = 10 nm and z = 25 nm with a rate of 1 x 1018 \([1/cm^3]\) . It has Gaussian shape from z = 25 nm to z = 45 nm (gaussian1D). It is zero between z = 0 nm and z = 10 nm, as well as between z = 45 nm and z = 50 nm.

<table>
<thead>
<tr>
<th></th>
<th>z = 0 ~ 10 nm</th>
<th>z = 10 ~ 25 nm</th>
<th>z = 25 ~ 45 nm</th>
<th>z = 45 ~ 50 nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>generation rate [1/cm^3]</td>
<td>0.0</td>
<td>constant (1.0 \times 1018)</td>
<td>Gaussian (center = 25nm, sigma_z=6.0 nm)</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Here is the structure part of the input file that generates the above generation profile. The whole input file is available [here](#).

```plaintext
structure{
  output_impurities{} # output doping concentration
  for each grid point in units of [10^18/(cm3)]

  region{
    everywhere()
    binary{ name GaAs }
    contact{ name cocont }
  }

  region{
    binary{ name GaAs }
    cuboid{
      (continues on next page)
    }
  }
}
```

---

8.2. Input File
Figure 8.2.2.2: Three-dimensional doping profile. (Image generated by Paraview.)
region{
    binary{ name = GaAs }
    cuboid{
        x = [0E0, 20E0]
        y = [0E0, 20E0]
        z = [0E0, 10E0]
    }
    doping{
        constant{
            name = "n-Si-in-GaAs"
            conc = 1.0E18
            # doping concentration [1/cm3] (applies to 1D, 2D and 3D)
        }
    }
}
region{
    binary{ name = GaAs }
    cuboid{
        x = [0E0, 20E0]
        y = [0E0, 20E0]
        z = [10E0, 25E0]
    }
    doping{
        gaussian1D{
            name = "n-Si-in-GaAs"
            conc = 1.0E18
            # maximum of generation rate [1/cm3]
            z = 25
            # z coordinate of Gauss center (ion's projected range Rp, i.e. the depth where most ions stop) [nm]
            sigma_z = 6.0
            # root mean square deviation in z direction (statistical fluctuation of Rp) [nm]
        }
    }
}

**generation()**

- **Syntaxes**
  - Specifications of generation rate profile
  - Print out
  - Remove
- **Example**
  - 3D

Specifications that define information on generation rate.
**Syntaxes**

**Specifications of generation rate profile**

The generation rate profile is assigned to a certain region. The following syntaxes are put under `structure{
region{ generation() } }`.

- `constan`t constant generation rate over the region

  ```
  constant{
    rate 1.0e18 # generation rate [1/cm^3 s]
    add yes # (optional) yes or no (default =
  }
  ```

- `linear` linearly varying generation rate along the line from start to end point

  ```
  linear{
    rate [1e18,2e18] # start and end value of
    x [50.0,100.0] # x coordinates of start and end
    y [50.0,100.0] # y coordinates of start and end
    z [50.0,100.0] # z coordinates of start and end
    # This defines a generation rate 
    # profile, which varies linearly along the line from the point (50, 
    # 50,50) to the point (100,100,100) 
    add yes # (optional) yes or no (default =
  }
  ```

- `gaussian1D` Gaussian distribution function in one direction, constant in perpendicular directions

  ```
  gaussian1D{
    rate 1.0e18 # maximum of generation rate [1/
    dose 1e12 # dose of implant [cm-2]
    # (integrated density of gaussian function), typical ranges are
    # from 1e11 to 1e16. # Either rate or dose has to be
    # specified, but not both simultaneously.
  }
  ```

(continues on next page)
# rate = dose / ( SQRT(2*pi) * sigma_x )

x = 50.0        # x coordinate of Gauss center (ion's projected range Rp, i.e. the depth where most ions stop) [nm]
sigma_x = 5.0   # root mean square deviation in x direction [nm]
y =             # (2D or 3D only)
sigma_y =       # (2D or 3D only)
z =             # (3D only)
sigma_z =       # (3D only)

# Only one out of x, y, z and the appropriate standard deviation (sigma) has to be specified.
add = yes        # (optional) yes or no (default = yes)

Note: This profile corresponds to LSS theory (Lindhard, Scharff, Schiott theory) for doping - Gaussian distribution of ion implantation.

**gaussian2D** Gaussian distribution function in two directions, constant in perpendicular direction (2D or 3D only)

**Example**

```plaintext
gaussian2D{
  # Gaussian distribution function in two directions, constant in perpendicular direction (2D or 3D only)
  rate = 1.0e18       # maximum of generation rate [1/cm^3 s]
  dose = 1.0          # dose of implant [cm^-1] (integrated density of 2D gaussian function)
  x = 50.0            # x coordinate of Gauss center [nm]
  sigma_x = 5.0       # root mean square deviation in x direction [nm]
  y = 50.0            # y coordinate of Gauss center [nm]
  sigma_y = 5.0       # root mean square deviation in y direction [nm]
  z =                 # (3D only)
  sigma_z =           # (3D only)
  # Exactly two out of x, y, z and the appropriate standard deviations (sigma) have to be specified.
  add = yes            # (optional) yes or no (default = yes)
}
```

**gaussian3D** Gaussian distribution function in three directions (3D only)

**Example**

```plaintext
gaussian3D{
  # Gaussian distribution function in three directions (3D only)
  rate = 1.0e18       # maximum of generation rate [1/cm^3 s]
  dose = 1.0          # dose of implant [dimensionless] (integrated density of 3D gaussian function)
  x = 50.0            # x coordinate of Gauss center [nm]
  sigma_x = 5.0       # root mean square deviation in x direction [nm]
  y = 50.0            # y coordinate of Gauss center [nm]
  sigma_y = 5.0       # root mean square deviation in y direction [nm]
  z =                 # (3D only)
  sigma_z =           # (3D only)
}
```
y = 50.0 # y coordinate of Gauss center [nm]
sigma_y = 5.0 # root mean square deviation in y的方向 [nm]
z = 50.0 # z coordinate of Gauss center [nm]
sigma_z = 5.0 # root mean square deviation in z的方向 [nm]

All three x, y, z and the appropriate standard deviations (sigma) have to be specified.
add = yes # (optional) yes or no (default = yes)

import import generation profile from external file

import{
    generation profile from external file.
    import_from = "import_generation_profile" # reference to imported data in import{}. The file being imported must have exactly one data component.
}

Print out

These generation rate profile can be printed out by output_generation{} under structure{}:

    output_generation{}

structure{
    output_generation{ # output generation rate for each grid point in units of [10^18/(cm^3 s)]
        boxes = yes/no # (optional)
    }
}

Remove

It is also possible to remove a generation rate from a specific region.

remove{}

structure{
    region{
        generation{ remove{} } # remove generation rate from this region, to keep certain regions free from generation rate.
        # region
    } # region
} # structure

Note: doping{} and generation{} is always additive per default (add = yes) (unless import is different), i.e. each profile adds to the already existing dopants/fixed charges/generation at a given point. At the same time, using remove{}, all species of the already existing doping or generation concentrations can be removed. However, there is also the problem that remove{} removes all species of dopants/fixed charges at a given point. Thus, removing e.g. only donors but not acceptors is difficult. This problem is solved by the new “add = yes/no” flag, which the user can specify for each profile (and thus for the species of that profile), whether the profile should add to (which is the default) or replace the already existing concentration of the profile species.

For import{}, this flag has not been implemented yet.
Example

3D

Figure 8.2.2.3 shows a 3D generation profile that is defined inside a 20 nm x 20 nm x 50 nm cube where the 50 nm are the z direction. The generation rate profile is homogeneous with respect to the (x,y) plane, it only varies along the z direction.

![Generation rate profile 3D](image_url)

Figure 8.2.2.3: Three-dimensional generation rate profile. (Image generated by Paraview.)

The generation rate profile is constant between \( z = 10 \) nm and \( z = 25 \) nm with a rate of \( 1 \times 10^{18} \) \( \text{1/(cm}^3\text{s)} \). It has Gaussian shape from \( z = 25 \) nm to \( z = 45 \) nm (gaussian1D). It is zero between \( z = 0 \) nm and \( z = 10 \) nm, as well as between \( z = 45 \) nm and \( z = 50 \) nm.

<table>
<thead>
<tr>
<th>generation rate ([1/(\text{cm}^3\text{s})])</th>
<th>( z = 0 \sim 10 ) nm</th>
<th>( z = 10 \sim 25 ) nm</th>
<th>( z = 25 \sim 45 ) nm</th>
<th>( z = 45 \sim 50 ) nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>generation</td>
<td>0.0</td>
<td>constant (1.0 x 1018)</td>
<td>Gaussian (center = 25 nm, ( \sigma_z = 6.0 ) nm)</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Here is the structure part of the input file that generates the above generation profile. The whole input file is available [here](link).

```plaintext
structure{
    output_generation{
        # output generation rate for each grid point
        in units of [10^18/(cm^3 s)]
    }
}
```

(continues on next page)
Repeating regions

The following specifiers can be used to define a periodically repeated pattern. Experimentally, a periodic geometry might have been generated by e.g. etching. This feature is useful for multi-quantum wells, superlattices, Quantum Cascade Lasers, Bragg reflectors, etc. Also, it is possible to make the number of layers a variable ($NUM_QUANTUM_WELLS$) in a corresponding template. Please also note to adjust the grid accordingly (grid{}), e.g. if a nonuniform grid is chosen.

Note: array_x{} was previously called repeat_x{} (and max was called num with max=num-1). repeat_x{} is deprecated and should be replaced with array_x{}. 

122 Chapter 8. nextnano++
array_x{}, array_y{}, array_z{}

These copy the region object. The loop runs from \(-\text{shift}\cdot\text{min}\) to \(\text{shift}\cdot\text{max}\).

**Example**

```plaintext
array_x{
  # (optional)
  shift 11.0 # repeat region in x direction by shifting it 11.0 nm (in units of [nm])
  max 3 # repeat region in x direction by applying the shift 3 (=max) times (Here, 4 regions will be set: The original one, and 3 shifted ones.) # max = 0 does not set a repeated region; negative values are not allowed.
  min 2 # (optional, default is 0) repeat region in negative x direction 2 times, i.e. the region object will be shifted 2 times by -shift.
  # min = 0 does not set a repeated (=shifted) region; negative values are not allowed.
  # In this example, the region is repeated 2 (=min) times into the negative direction and 3 (=max) times into the positive direction.
}
array_y{
  # (optional, 2D and 3D only)
}
array_z{
  # (optional, 3D only)
}
```

**repeat_profiles**

`repeat_profiles` specifies which profiles are shifted. If `repeat_profiles` is not defined, shift all profiles (default).

**Example**

```plaintext
repeat_profiles

-# having a periodic light absorbing mask

# periodic generation may be caused by

# having a periodic light absorbing mask

# (continues on next page)
```

If you want to shift alloy/doping/generation profiles independent of each other you have to define separate regions (`region`) for each, for instance a separate region for doping where you add `array_x` and `shift`.

(=> `region{ doping[...{ array_x{ shift = ... } }]`)

For instance, two identical layers containing 16 quantum dots each, can be easily generated by specifying only one quantum dot geometry.

```plaintext
region{
  cone{ # Here, the quantum dot has the shape of a cone.
    base_x [1.0,7.0] # extension of base plane in x direction, i.e.
    from 1.0 to 7.0 nm
    base_y [1.0,7.0] # extension of base plane in y direction, i.e.
    from 1.0 to 7.0 nm
}
```

(continues on next page)
base_z = [6.0, 6.0] # base plane at z = 6.0 nm

top = [4.0, 4.0, 10.0] # top coordinate of the cone (x,y,z) = (4.0,4.0,10.0) in units of [nm]
diminution = 0.25 # cone: diminution = 0.0, cylinder: diminution

Note: Edges of the element base_x, base_y, and base_z should be treated by topological features of the element base_x.

ternary_linear{
    name = "Al\(_x\)Ga\(_{1-x}\)As" # Al\(_x\)Ga\(_{1-x}\)As
    alloy_x = [0.25, 1.0] # vary alloy composition from x = 0.25 (Al\(_{0.25}\)Ga\(_{0.75}\)As) to x = 1.0 (AlAs)
    z = [10, 6] # vary alloy content from z = 10 nm to z = 6 nm
}

array_x{
    shift = 11.0 max 3 # repeat region in x direction by shifting it
    by 11.0 nm (4 objects in total: The original region and 3 shifted ones.)
}
array_y{
    shift = 11.0 max 3 # repeat region in y direction by shifting it
    by 11.0 nm (4 objects in total: The original region and 3 shifted ones.)
}
array_z{
    shift = 20.0 max 1 # repeat region in z direction by shifting it
    by 20.0 nm (2 objects in total: The original region and 1 shifted one.)
}
repeat_profiles{ "alloy" }
array2_x{}, array2_y{}, array2_z{}

These are the second hierarchy of repetitions.

Example

```plaintext
region{
    binary{ name = "InAs" }
    array_x{ shift = 20 num = 5 }
    array_y{ shift = 20 num = 5 }
    array2_x{ shift = 150 num = 3 }
    array2_y{ shift = 150 num = 3 }
    repeat_profiles = 'other printing'
    circle{
        center{ x = 100 y = 100 }
        radius = 30
    }
    doping{
        gaussian2D{
            name = 'conc1 e18 x = 100 y = 00 sigma_x = 7 sigma_y = 7'
            add = yes
        }
    }
}
```

This produces the following:

For repeated structures which extend beyond the bounds of the simulation regions, please make sure that min and max are large enough to also include objects which are partially outside of the simulation region.

**Note:** When periodic{...} is used, objects extending over an edge of the simulation region will not automatically be continued on the opposite side. If such objects are present in a periodic simulation, for each periodic coordinate direction (x, y or z), please either define a repetition (using the size of the simulation region as shift with max = 1 and/or min = 1 as needed), or extend an already present repetition to the edge of the simulation region (by increasing min and max as needed).

**Warning:** Special care has to be taken when using remove{} or add = no for doping{}/fixed charge/generation{} in some repeated regions. Namely, repeated regions are created by sequentially
creating multiple instances of a given region at the different positions defined by the \texttt{array\_*} and \texttt{array2\_*} statements. But the order in which these instances are created depends on undocumented implementation details and thus may change from release to release. For additive dopants/fixed charges/generation, or for repeated regions which do not self-overlap, the final structure and profiles do not depend on this undocumented creation order and thus no problems will occur. However, for repeated regions which self-overlap (e.g. due to small region shifts), using \texttt{remove\{\}} or \texttt{add = no} results in the final structure and profiles being dependent on that creation order and often being different from the user’s intentions. Therefore, in case of doubt, please visually inspect your structure and profiles to avoid such issues.

\textbf{grid{} }

Specifications of the non-uniform rectangular grid lines.

- \texttt{xgrid{} / ygrid{} / zgrid{} }

\textbf{xgrid{} / ygrid{} / zgrid{} }

These grid lines set the total device size, so at least two lines have to be present for each relevant direction.

\begin{verbatim}
grid{
  xgrid{} # x position of line and adjacent grid spacing
  -line{ pos 0 spacing 0.5 } # 0.5 nm spacing at —
  -position 0 nm
  -line{ pos 50 spacing 0.1 }  # 0.1 nm spacing at —
  -position 50 nm
  -line{ pos 100 spacing 0.5 }  # 0.5 nm spacing at —
  -position 100 nm
}
\end{verbatim}

\texttt{pos}

\textbf{value} any float

\begin{itemize}
  \item position of the line [nm]
\end{itemize}

\texttt{spacing}

\textbf{value} any float $\geq 0.0$

\begin{itemize}
  \item grid spacing adjacent to this line [nm]
  \item Minimum spacing is $10^{-3}$ nm, i.e. 1 pm.
\end{itemize}
Note: This example where at 75 nm an abrupt change in grid spacing is enforced is a bit dangerous. This can have impacts on numerical stability and numerical accurateness as discretized equations are only approximations to the continuous equations being solved. For nonuniform grids, higher order terms for the first or second derivates might get too large compared to slightly nonuniform grids. Abrupt changes in grid spacing rather should be avoided if possible. A corresponding warning message is written to the log file for duplicate grid points and jumps in grid points.

Note: The program generates a grid that has the specified spacing adjacent to the specified grid lines. If the grid is not equidistant, an exponential grid is generated. In this case, the calculated spacing corresponds only approximately to the specified spacing. The grid spacing used might have influence on the numerical convergence of the equations.

The actually generated grids are also written into files grid_x.dat, grid_y.dat, and grid_z.dat. Especially for more complicate grid definitions or grid definitions employing variables, it is recommended to review these files to make sure that the results match your expectations. You can also use nextnanomat → Output → Show grid to visualize the grid lines.

Repeating grid lines In the same way as regions (=== region{}) can be repeated, also grid lines can be repeated which might be useful for multi-quantum wells, superlattices, Quantum Cascade Lasers, Bragg reflectors, etc.

Also, it is possible to make the number of repetitions a variable in a corresponding template.

array{} copies the line object located at pos a few times, namely min times into the negative direction and max times into the positive direction, i.e. the loop runs from -shift*min to shift*max.
The grid line definitions `line{}` are processed in the order of their occurrence in order to obtain a list of default grid lines. Here, within each `line{}` definition, duplicate grid lines created by `array{}` and `array2{}` are ignored. After all `line{}` definitions have been processed, the list of default grid lines is ordered in ascending position (but the order of doubled lines remains unchanged), and intermediate grid lines are added as suggested by the respective grid spacings. Note that `array{}` and `array2{}` may accidentally cause doubled grid lines between different `line{}` definitions. In this case, resulting jumps in grid spacings may be nonobvious and should be checked in the output.

```
xgrid{
  min_pos = 100.0 # (optional) nm
  max_pos = 200.0 # (optional) nm
  line{...
}
```

Using the optional variables `min_pos` and `max_pos` (defaults are -Infinity and +Infinity), a (half-)range can be defined for each coordinate direction within which all grid lines to be used are located. Grid lines located outside (e.g. due to a position defined by a formula or by a repetition) will be ignored. Please note that specifying a (half-)range in itself does not specify a grid line for the respective range limit(s). Similarly, there is no automatic insertion of grid lines as controlled by the respective spacings between a grid line defined inside of a (half-)range and an adjacent grid line defined outside of the (half-)range.

Per default, all defined grid lines are used for the computation, with the boundaries of the simulation region being defined by the smallest and largest coordinate positions of all grid lines defined for a respective coordinate direction. However, there are circumstances where structural features are partially or completely outside of the simulation region, and thus no grid lines are allowed for these structural features. Here, in order to facilitate reuse of coordinate calculations between region definitions and grid definitions, it is helpful to set `min_pos` and `max_pos` to the desired extent of the simulation region in order to prevent unwanted grid lines from being generated. All grid line definitions from `line{}` which fall outside of the interval `[min_pos, max_pos]` are ignored. Thus, if the such defined interval is [100, 200], an individual line with `pos = 250` would be ignored. Similarly, when a line is repeated using `array{}` and/or `array2{}` multiple times, the line repetitions falling outside of the interval would be ignored.

### Jumps in grid spacing

By default, jumps in grid spacings are not allowed. Optionally, they can be switched on.

```
xgrid{
  allow_spacing_jumps = yes
  line{...
}
```

For meaning of `array{}`, see `array_x{}` in `structure{}`.

For meaning of `array2{}`, see `array2_x{}` in `structure{}`.
Here, left of the point at x=50 nm, a grid spacing of 0.5 nm is used, and right of the point at x=50 nm, a grid spacing of 0.1 nm is used which is a jump in grid spacing. allow spacings jumps = yes is not recommended due to a loss in numerical accuracy and numerical stability. The only purpose of this feature is to generate input files that are compatible to input files of the nextnano³ software which supports jumps by default.

**import{}**

Specifications for importing data from a file or generating them from an analytic function, e.g. electrostatic potential, alloy profile, strain profile, doping profile, generation rate profile, electron or hole Fermi level profile.

Once a file has been imported or a function has been defined, it can be used several times, e.g. the same file could include the alloy concentration of a ternary for different region objects.

Data with dimensionality deviating from the simulation dimension can also be imported, e.g. an absorption profile for solar cell modeling.

```text

<table>
<thead>
<tr>
<th>directory</th>
<th>value</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>example</td>
<td>“D:\import_files\”</td>
<td></td>
</tr>
</tbody>
</table>

Name of directory where files to be imported are located (if data are imported from files)

```text

```text

<table>
<thead>
<tr>
<th>file{}</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
</tbody>
</table>

Name for referencing the imported data in the input file, e.g. “imported_potential_profile_2D”

```text

```text

| filename | example | “D:\any_filename.fld” |

Name of file which is imported. If an absolute path is included here, the entry in directory specified above is ignored.

```text

```text

| alternative | “any_filename.fld” |

Name of file which is imported. The file should be located here: directory\filename

```text

```text

<table>
<thead>
<tr>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>option options</td>
</tr>
</tbody>
</table>

Format of the file to be imported. At the moment only AVS format and a simple .dat format are supported.

```text

```text

| number_of_dimensions | option options | 1 2 3 |

Explicit specification of the number of dimensions. Can be only used for .dat files.
(Optional, default value is given by dimensionality of simulation defined in global{})

```text

```text

<table>
<thead>
<tr>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>option examples</td>
</tr>
</tbody>
</table>

Default 1
Imported data is multiplied by this scaling factor (optional).
Useful to correct unit of imported data, e.g. conversion from [Joule] to [eV].

**analytic_function**{} define analytic functions to be imported here. Does not need to be defined if data are imported from files.

**name**

**example** "Analytic_Potential"
Name for referencing the imported function in the input file, e.g. "analytic_function_gaussian"

**function**

**example** \(1+2^x+2+\exp(y)\)
String defining the function in case only one component needs to be defined, otherwise use component.

---

**Note:** Syntax allowed for functions:

- white spaces are ignored
- valid operators are "+", "-", "\^", "/" and "\^\^"
- multiplication signs always have to be spelled out (i.e. "5\times x" is valid, "5x" is not)
- variable names are fixed to "x", "y" and "z" (capital letters are also allowed)
- additional functions also available (e.g. \("\exp\", "\sqrt\", "\sin\", see full list below), have to be followed by brackets ("\(\exp(x)\)" is valid, "\(\exp x\)" is not)
- global variables are allowed if preceded by "$" (e.g. "$\pi$"
- exponential notation ("2e-3" or "4E10") is allowed

---

**label**

**example** "potential_label"
Label (optional) to be displayed in legend in case only one component is defined, otherwise see component.

**component**{} In case multiple components are needed, define one component group for each component.

**function_i**

**example** "1D_import"
String defining the function for this component.

**label**

**example** "component_label"
Label (optional) to be displayed in legend for this component.

**output_imports**{} Output all imported data including scale factor.

The filenames correspond to the entry given in name = ....
The files will be written to a folder called Imports/.

The following operators (sorted with decreasing precedence):
and functions are defined in analytic_function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt()</td>
<td>square root $\sqrt{}$</td>
</tr>
<tr>
<td>cbrt()</td>
<td>cubic root $\sqrt[3]{\cdot}$</td>
</tr>
<tr>
<td>exp()</td>
<td>exponential function $\exp()$</td>
</tr>
<tr>
<td>log()</td>
<td>natural logarithm $\log$</td>
</tr>
<tr>
<td>ln()</td>
<td>natural logarithm $\ln$</td>
</tr>
<tr>
<td>log2()</td>
<td>decadic logarithm (base 2) $\log_2$</td>
</tr>
<tr>
<td>log10()</td>
<td>decadic logarithm (base 10) $\log_{10}$</td>
</tr>
<tr>
<td>sin()</td>
<td>sine $\sin()$</td>
</tr>
<tr>
<td>cos()</td>
<td>cosine $\cos()$</td>
</tr>
<tr>
<td>tan()</td>
<td>tangent $\tan()$</td>
</tr>
<tr>
<td>asin()</td>
<td>arcsine $\sin^{-1}()$</td>
</tr>
<tr>
<td>acos()</td>
<td>arccosine $\cos^{-1}()$</td>
</tr>
<tr>
<td>atan()</td>
<td>arctangent $\tan^{-1}()$</td>
</tr>
<tr>
<td>sinh()</td>
<td>hyperbolic sine $\sinh()$</td>
</tr>
<tr>
<td>cosh()</td>
<td>hyperbolic cosine $\cosh()$</td>
</tr>
<tr>
<td>tanh()</td>
<td>hyperbolic tangent $\tanh()$</td>
</tr>
<tr>
<td>asinh()</td>
<td>inverse hyperbolic sine $\sinh^{-1}()$</td>
</tr>
<tr>
<td>acosh()</td>
<td>inverse hyperbolic cosine $\cosh^{-1}()$</td>
</tr>
<tr>
<td>atanh()</td>
<td>inverse hyperbolic tangent $\tanh^{-1}()$</td>
</tr>
<tr>
<td>erf()</td>
<td>error function $\text{erf}()$</td>
</tr>
<tr>
<td>erfc()</td>
<td>complementary error function $\text{erfc}()$</td>
</tr>
<tr>
<td>gamma()</td>
<td>Gamma function $\Gamma()$</td>
</tr>
<tr>
<td>fdm3half()</td>
<td>complete Fermi–Dirac integral $F_{-3/2}()$ of order -3/2 (includes the $1/\Gamma(-1/2)$ prefactor)</td>
</tr>
<tr>
<td>fdmhalf()</td>
<td>complete Fermi–Dirac integral $F_{-1/2}()$ of order -1/2 (includes the $1/\Gamma(1/2)$ prefactor)</td>
</tr>
<tr>
<td>fdzero()</td>
<td>complete Fermi–Dirac integral $F_0()$ of order 0 (includes the $1/\Gamma(1) = 1$ prefactor)</td>
</tr>
<tr>
<td>fdphalf()</td>
<td>complete Fermi–Dirac integral $F_{1/2}()$ of order 1/2 (includes the $1/\Gamma(3/2)$ prefactor)</td>
</tr>
<tr>
<td>fdp3half()</td>
<td>complete Fermi–Dirac integral $F_{3/2}()$ of order 3/2 (includes the $1/\Gamma(5/2)$ prefactor)</td>
</tr>
<tr>
<td>abs()</td>
<td>absolute value $</td>
</tr>
<tr>
<td>floor()</td>
<td>floor function $\text{floor}(x)$: largest integer $\leq x$</td>
</tr>
<tr>
<td>ceil()</td>
<td>ceiling function $\text{ceil}(x)$: smallest integer $\geq x$</td>
</tr>
<tr>
<td>round()</td>
<td>rounds the number to the nearest integer</td>
</tr>
<tr>
<td>sign()</td>
<td>sign function $\text{sign}(\cdot)$</td>
</tr>
<tr>
<td>iszero()</td>
<td>check if value is zero $\text{iszero}(\cdot)$</td>
</tr>
<tr>
<td>isnotzero()</td>
<td>check if value is not zero $\text{isnotzero}(\cdot)$</td>
</tr>
<tr>
<td>ispositive()</td>
<td>check if value is positive $\text{ispositive}(\cdot)$</td>
</tr>
<tr>
<td>isnotpositive()</td>
<td>check if value is not positive $\text{isnotpositive}(\cdot)$</td>
</tr>
<tr>
<td>isnegative()</td>
<td>check if value is negative $\text{isnegative}(\cdot)$</td>
</tr>
<tr>
<td>isnotnegative()</td>
<td>check if value is not negative (corresponds to $\text{heaviside}(\cdot)$)</td>
</tr>
<tr>
<td>heaviside()</td>
<td>Heaviside step function (corresponds to $\text{isnotnegative}(\cdot)$)</td>
</tr>
</tbody>
</table>

**impurities{}**

Specifications that define impurities (donors, acceptor and fixed charges)

- donor{} / acceptor{} 
- charge{}
donor{ } / acceptor{ }

name
value <String> from the database.

energy
value <Float>

degeneracy
value <Integer>

The energy separation from the conduction or valence band edge is given in units of electron-volts, $eV$. These energies are meant as ionization energies, e.g. a donor with an energy level right below the conduction band edge would be specified by a small positive energy level. Degeneracy of impurity levels affects their degree of ionization. The degeneracy of donors is usually assumed to be equal to 2, for acceptors it is equal to 4.

- **shallow donors: degeneracy factor 2** Outer s orbital is onefold occupied (neutral state). There is one possibility to get rid of one electron but there are two to incorporate one (spin up, spin down).

- **shallow acceptors: degeneracy factor 4** The $sp^3$ orbital is threefold occupied. Thus, one possibility to incorporate an electron, four possibilities to get rid of one.

More details on degenerate impurity levels can be found in e.g. [ChuangOpto1995]. Note that in nitride semiconductors crystallizing in the wurtzite structure the degeneracy factor may vary from 4 to 6 because of a small valence band splitting.

Example:

```plaintext
impurities{
  donor{ name = "n-P-in-Si" energy = 0.045 degeneracy = 2 }
  donor{ name = "n-As-in-Si" energy = 0.054 degeneracy = 2 }
  acceptor{ name = "p-B-in-Si" energy = 0.045 degeneracy = 4 }
}
```

_Cheat parameter:_ energy = -1000 (for instance), that means, all electrons are fully ionized from the donors (similar for holes/acceptors). This might be useful for low temperatures like 4 K where usually the degree of ionization is very small. By using -1000 one can force them to be completely ionized. If full ionization is assumed, i.e. energy = -1000, then the degeneracy factor effectively becomes irrelevant. This can be seen from eqs. (1.4) – (1.7) in PhD thesis of Stefan Birner.

```plaintext
impurities{
  donor{ name = "fully-ionized" energy = -1000 degeneracy = 2 }
  acceptor{ name = "fully-ionized" energy = -1000 degeneracy = 4 }
}
```

charge{ }

name
value <String>

type
value positive, negative

default

It can be used to put positive or negative charges into the device (e.g. to describe interface charges)

Example:
### Table 8.2.2.2: Donor levels (n-type) in units of $eV$ relative to conduction band edge

<table>
<thead>
<tr>
<th>Donor Name</th>
<th>Energy</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-As-in-Si</td>
<td>0.054</td>
<td>DESSIS</td>
</tr>
<tr>
<td>n-Sb-in-Si</td>
<td>0.039</td>
<td>DESSIS</td>
</tr>
<tr>
<td>n-N-in-Si</td>
<td>0.045</td>
<td>DESSIS</td>
</tr>
<tr>
<td>n-N-in-SiC</td>
<td>0.10</td>
<td>DESSIS</td>
</tr>
<tr>
<td>n-Si-in-GaAs</td>
<td>0.0058</td>
<td></td>
</tr>
<tr>
<td>n-Si-in-AlAs</td>
<td>0.007</td>
<td>300 K, Landolt-Boernstein</td>
</tr>
<tr>
<td>n-Si-in-Al0.27Ga0.73As</td>
<td>0.006</td>
<td>Landolt-Boernstein</td>
</tr>
</tbody>
</table>

More parameters can be found in the nextnano³ database file `database.in` or at this link

### Table 8.2.2.3: Acceptor levels (p-type) in units of $eV$ relative to valence band edge

<table>
<thead>
<tr>
<th>Acceptor Name</th>
<th>Energy</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-In-in-Si</td>
<td>0.16</td>
<td>DESSIS</td>
</tr>
<tr>
<td>p-Al-in-SiC</td>
<td>0.20</td>
<td>DESSIS</td>
</tr>
<tr>
<td>p-C-in-GaAs</td>
<td>0.027</td>
<td>Landolt-Boernstein 1982</td>
</tr>
</tbody>
</table>

More parameters can be found in the nextnano³ database file `database.in` or at this link

**classical()**

specifies which bands enter classical densities and dopant densities.
# valence bands
HH{ output_bandedge{ ... } }
LH{ output_bandedge{ ... } }
SO{ output_bandedge{ ... } }

# Further output definitions
output_bandedges{
  profiles = "Gamma H H H"
  averaged = yes
}

output_bandgap{
  averaged = yes
}

output_carrier_densities{}
output_ionized_dopant_densities{}
output_intrinsic_density{
  boxes = yes
}
output_energy_resolved_densities{}

energy_distribution{
  min = -5.0
  max = 5.0
  energy_resolution = 0.1
  only_quantum_regions = yes
}

energy_resolved_density{
  min = -5.0
  max = 5.0
  energy_resolution = 0.1
  only_quantum_regions = yes
}

emission_spectrum{
  output_spectra{
    emission = yes
    gain = yes
    absorption = yes
    stimulated_emission = yes
    spectra_over_energy = yes
    spectra_over_wavelength = yes
    spectra_over_frequency = yes
    spectra_over_wavenumber = yes
    photon_spectra = yes
    power_spectra = yes
  }
  refractive_index = 3.0
}

output_photon_density = yes
output_power_density = yes

# Carrier statistics for classical densities
carrier_statistics =

# Irradiation
irradiation{
    min_wavelength = 300.0
    max_wavelength = 2000.0
    min_energy = 0.5
    max_energy = 5.0
    energy_resolution = 0.01

    illumination{
        direction_x = 1

        database_spectrum{
            name = "Solar-ASTM-G173-global"
            concentration = 300
        }
    }

    reflectivity{
        database_spectrum{
            name = "Al0.8Ga0.2As"
        }
    }

    absorption{
        database_spectrum{
            name = "GaAs"
        }
    }

    output_spectra{
        illumination = yes
        reflectivity = yes
        absorption = yes

        spectra_over_energy = yes
        spectra_over_wavelength = yes
        spectra_over_frequency = yes
        spectra_over_wavenumber = yes
    }
}

photo_generation{
    output = yes
    output_spectrum = yes
    output_energy_resolved = yes
    output_quantum_efficiency = yes
}

output_light_field = yes

Below are various variables and functions to be used inside classical{}. 

8.2. Input File
Conduction bands

**Gamma** at Gamma point

- **output_bandedge** output band edge of this band in a single file [eV]
  - **averaged**
    - **value** yes or no
    - **default** yes
      - yes: for each grid point the band edge will be averaged between neighboring material grid points. (averaged = yes is similar to ‘boxes = no’. Note that ‘boxes’ is related to output of material grid points while averaged is related to output of simulation grid points.)
      - no: abrupt discontinuities at interfaces (in 1D two points, in 2D four points, in 3D eight points for each grid point)

**Warning:** 2D and 3D simulations can produce a lot of output data (~order of GB). In order to avoid excessive consumption of your hard disk, it is strongly recommended to use averaged = yes for 2D and 3D simulations.

**L** at X point; does not apply to wurtzite materials

- **output_bandedge** analogous usage as above

**X** at X point; does not apply to wurtzite materials; for all materials other than Si, Ge, GaP

- **output_bandedge** analogous usage as above

or

**Delta** at Delta point; for Si, Ge, GaP

- **output_bandedge** analogous usage as above

Valence bands

**HH** heavy hole

- **output_bandedge** analogous usage as above

**LH** light hole

- **output_bandedge** analogous usage as above

**SO** split-off hole (or crystal-field split-off hole in wurtzite)

- **output_bandedge** analogous usage as above

Further output definitions

**output_bandedges** output all (relevant) band edges and (relevant) Fermi levels in one file named ‘bandedges.’ [eV].

**profiles** Enumerate relevant band edges for output. If profiles are not defined, all band edges are written out.

- **options** Gamma X Delta L HH LH SO electron_fermi_level hole_fermi_level
- **example** profiles = "Gamma HH LH"
averaged

value yes or no

default yes

• yes: for each grid point the band edges will be averaged between neighboring material
  grid points. (averaged = yes is similar to ‘boxes = no’. Note that ‘boxes’ is related to
  output of material grid points while averaged is related to output of simulation grid
  points.)

• no: abrupt discontinuities at interfaces (in 1D two points, in 2D four points, in 3D
  eight points for each grid point)

output_bandgap{} output band gaps for Gamma, L, X (or Delta) bands with reference to the highest
valence band edge. Additionally the difference between the lowest conduction band and the
highest valence band edges is written out: MIN(Gamma,L,X (or Delta)) - MAX(hh,lh,so) [eV]

averaged

value yes or no

default yes

• yes: for each grid point the band gaps will be averaged between neighboring material
  grid points. (averaged = yes is similar to ‘boxes = no’. Note that ‘boxes’ is related to
  output of material grid points while averaged is related to output of simulation grid
  points.)

• no: abrupt discontinuities at interfaces (in 1D two points, in 2D four points, in 3D
  eight points for each grid point)

output_carrier_densities{} output electron and hole densities \([10^{18}/cm^3]\]

The units for the output file total_charges.txt are

• electrons /cm^2 (1D simulation)
• electrons /cm (2D simulation)
• electrons (3D simulation)

output_ionized_dopant_densities{} output ionized dopant densities \([10^{18}/cm^3]\]

The ionized acceptor and donor densities are written to these files:

• density_acceptor_ionized.dat
• density_donor_ionized.dat

output_intrinsic_density{} output intrinsic density \([1/cm^3]\]

boxes

value = yes or no

(optional) For each grid point, in 1D two points are printed out to mimic abrupt discontinuities
at interfaces (in 2D four points, in 3D eight points)

output_energy_resolved_densities{} output energy-resolved densities \(n(x, E), p(x, E)\) in units of
\([cm^{-3}eV^{-1}]\) in 1D, \([cm^{-3}eV^{-1}]\) in 2D, and \([cm^{-3}eV^{-1}]\) in 3D.

energy_distribution{} output integrated electron and hole density as a function of energy, \(n(E), p(E)\) in units of \([cm^{-2}eV^{-1}]\) in 1D, \([cm^{-1}eV^{-1}]\) in 2D, and \([eV^{-1}]\) in 3D.

min minimum energy [eV] (required)

value double

default -10.0

max maximum energy [eV] (required)
value double
default 5.0

energy_resolution energy spacing [eV] (optional)
value double
default 0.1

only_quantum_regions
value yes or no
default no

consider only quantum regions (optional), can be used to suppress contributions from outside the quantum regions of interest. This works even if quantum mechanics is not enabled in run{}

Note: Note that energy_distribution{}, which directly calculates the space-integrated energy-resolved density, is logically independent from the two groups energy_resolved_density{} and emission_spectrum{}. Incidentally, it also runs much faster and needs much less memory for the same energy spacing.

energy_resolved_density{} output electron and hole density as a function of energy and position, \(n(x, E), p(x, E)\) in units of \([\text{cm}^{-3}\text{eV}^{-1}]\) in 1D, \([\text{cm}^{-3}\text{eV}^{-1}]\) in 2D, and \([\text{cm}^{-3}\text{eV}^{-1}]\) in 3D.

min minimum energy [eV] (required)
value double
default -10.0

max maximum energy [eV] (required)
value double
default 5.0

energy_resolution energy spacing [eV] (optional)
value double
default 0.1

only_quantum_regions
value yes or no
default no

consider only quantum regions (optional), can be used to suppress contributions from outside the quantum regions of interest. This works even if quantum mechanics is not enabled in run{}

Note:

- min, max always refer to a zero point at the (local) conduction band edge, and not to the photon energy.
- max should be set high enough above 0 to contain all occupied electron states and min should be set far enough below -bandgap to contain all occupied hole states.
- The respective values for energy_resolution should be set smaller than \(k_BT\) if one wishes to fully resolve the structures of the integrated densities and/or of the emission spectra.
However, while setting energy\_resolution in energy\_distribution{} as low as 0.001 eV has little influence on program execution time, using similarly small values for energy\_resolution in energy\_resolved\_density{} will result in massive slowdowns (and in 3D also in massive memory use), since the computational effort for obtaining emission spectra grows quadratically with the number of energy bins.

**emission\_spectrum{}** output emission spectrum calculated from energy-resolved densities $n(x, E)$ and $p(x, E)$ computed by energy\_resolved\_density{}. Radiative recombination rate reads $R_{\text{radiative}}(x, E) = C(x) \int dE_e \int dE_h n(x, E_e)p(x, E_h)\delta(E_e - E_h - E)$, where $C(x)$ [cm$^3$/s] is the (material-dependent) radiative recombination parameter. “spectra” and “density” in the following refer to the integrals of $R_{\text{radiative}}$ over position and energy, respectively.

**output\_spectra{}** computed types of spectra:

- **emission**
  - value yes or no
  - default yes

- **gain**
  - value yes or no
  - default yes

- **absorption**
  - value yes or no
  - default yes

- **stimulated\_emission**
  - value yes or no
  - default no

Unit selection - the following apply to all of the above

- **spectra\_over\_energy**
  - value yes or no
  - default yes

- **spectra\_over\_wavelength**
  - value yes or no
  - default yes

- **spectra\_over\_frequency**
  - value yes or no
  - default no

- **spectra\_over\_wavenumber**
  - value yes or no
  - default no

Unit selection - the following apply to all of the above

- **photon\_spectra**
  - value yes or no
  - default yes

Unit selection - the following apply to all of the above
power_spectra
value yes or no
default no

refractive_index
value double
default 1.0

averaged-out refractive index $n_r$ (optional, dimensionless). Refractive index used for calculating gain and absorption. The absorption/gain is multiplied by the factor $1/n_r^2$. The values for the optical dielectric constant from the database are not used yet at this point.

output_photon_density
value yes or no
default no

output emitted photon density in units of $[\text{cm}^{-3}\text{s}^{-1}]$ (optional)
(emitted_photon_density.dat)

output_power_density
value yes or no
default no

output emitted power density in units of $[\text{W/cm}^3]$ (optional)
(emitted_power_density.dat)

Carrier statistics for classical densities

carrier_statistics
options fermi_dirac maxwell_boltzmann
default fermi_dirac

Optionally, one can use Maxwell-Boltzmann statistics for the classical densities (not recommended as this is only an approximation which is only applicable in certain cases).

In order to maintain consistency, also the (integrated) energy distribution (density_vs_energy) and the classical emission spectra and densities are computed using the same statistics. Use together with quantum regions is possible but not recommended, and convergence of the current-Poisson or quantum-current-Poisson equation may become worse (please readjust convergence parameters accordingly).

Note:

* $n = N_c \mathcal{F}_{1/2} \left( \frac{E_F - E_n}{k_B T} \right)$ (electron density for fermi_dirac)
* $p = N_c \mathcal{F}_{1/2} \left( \frac{E_F - E_p}{k_B T} \right)$ (hole density for fermi_dirac)
* $n = N_c \exp \left( \frac{E_F - E_n}{k_B T} \right)$ (electron density for maxwell_boltzmann)
* $p = N_c \exp \left( \frac{E_F - E_p}{k_B T} \right)$ (hole density for maxwell_boltzmann)
* where $\mathcal{F}_n(E)$ is a Fermi-Dirac integral of the order $n$. 
**Irradiation**

Output of comparison spectra, compute light field in the device, and photo-generation.

Note that spectral data can be alternatively defined in the database (see also database.nnp.optional for list of predefined illumination spectra), in the database section of the input file, or imported from external files.

Lorentzian, Gaussian and Planck illumination spectra are fully additive, i.e. several of each can be added as needed in order to synthesize more complex illumination spectra.

**irradiation{}**

Definition of energy grid. Specify either wavelength or photon energy.

minimum and maximum photon wavelength

- **min_wavelength**
  - value: real value >= 10 and <= 1e6
  - units: [nm]

- **max_wavelength**
  - value: real value >= 10 and <= 1e6
  - units: [nm]

minimum and maximum photon energy

- **min_energy**
  - value: real value >= 1e-6 and <= 100
  - units: [eV]

- **max_energy**
  - value: real value >= 1e-6 and <= 100
  - units: [eV]

resolution of energy grid

- **energy_resolution**
  - value: real value >= 1e-6
  - units: [eV]

```plaintext
irradiation{
  min_energy 0.5
  max_energy 5.0
  # min_wavelength = 300.0
  # max_wavelength = 2000.0
  energy_resolution 0.01

  illumination{

direction_x
  - value: integer +1/-1
  - units: (dimensionless)
  ascending/descending illumination in x direction

direction_y
  - value: integer +1/-1
```

8.2. Input File
units (dimensionless)
ascending/descending illumination in y direction
direction_z
value integer +1/-1
units (dimensionless)
ascending/descending illumination in z direction

Database spectrum
Several spectra (solar spectra, CIE illuminants, absorption, reflectivity, ...) can be found in the database file database_nnp_optional.

database_spectrum{}
database unit W/m², relative intensities (e.g. CIE illuminants) are normalized to 1.0 W/m²

name
value string
concentration
value real value >= 0.0
default 1.0

illuminaton{
direction_x

database_spectrum{
name "Solar-ASTM-G173-global"
# name = "CIE-D75"
concentration 300 # e.g. 300 suns
}

# or
import_spectrum{
import_from "filename"
cutoff yes # yes/no: If yes, set values outside definition interval to zero.
# (default=?)
absolute_intensities yes # yes/no (default: yes)
# If no, spectrum does not contain absolute values,
# normalize intensity to 1 [W/cm² nm⁻¹]
before concentration
concentration 300 # e.g. 300 suns
}

Planck spectrum

planck_spectrum{
irradiance 10000.0 # in [W/m²], for complete(!) Planck spectrum; real value
>= 0.0
temperature 5000.0 # real value >= 1e-6
}

Lorentzian spectrum
lorentzian_spectrum{
  irradiance = 10000.0 # in [W/m^2], for complete(!) Lorentzian spectrum;
  real value >= 0.0

  # Specify either wavelength and width, or ...
  wavelength = 500.0 # real value >= 10.0 in [unit:nm]
  width = 100.0 # real value >= 1e-3 in [unit:nm]

  # ... specify energy and gamma.
  energy = 2.5 # real value >= 1e-6 in [unit:eV]
  gamma = 1.0 # real value >= 1e-6 in [unit:eV]
}

Gaussian spectrum

 gaussian_spectrum{
  irradiance = 1000.0 # in [W/m^2], for complete(!) Gaussian spectrum; real
  value >= 0.0

  # Specify either wavelength and width, or ...
  wavelength = 500.0 # real value >= 10.0 in [unit:nm]
  width = 100.0 # real value >= 1e-3 in [unit:nm]

  # ... specify energy and gamma.
  energy = 2.5 # real value >= 1e-6 in [unit:eV]
  gamma = 1.0 # real value >= 1e-6 in [unit:eV]
}

Constant spectrum

constant_spectrum{
  irradiance = 10000.0 # in [W/m^2], integrated as min_energy...max_energy
}

reflectivity{}

  database_spectrum{
    name = "Al0.8Ga0.2As"
  }

  # or

  import_spectrum{
    import_from = "filename"
    cutoff = yes # yes/no: If yes, set values outside definition
    interval to zero. # (default=?)
  }

  # or

  constant_spectrum{
    reflectivity = 0.5 # real value >= 0.0 and <= 1.0 (dimensionless)
  }
}

absorption{}

  database_spectrum{
    (continues on next page)
name "GaAs"

# or

import_spectrum{
  import_from "filename"
  cutoff yes # yes/no: If yes, set values outside definition to zero.
  # (default=?)
}

# or

constant_spectrum{
  absorption 0.5 # real value >= 0.0 [1/cm]
}

output_spectra{
  illumination yes # yes/no (default: yes)
  reflectivity yes # yes/no (default: yes)
  absorption yes # yes/no (default: yes)
  spectra_over_energy yes # yes/no (default: yes)
  spectra_over_wavelength yes # yes/no (default: yes)
  spectra_over_frequency yes # yes/no (default: yes)
}

photo_generation{
  output yes # yes/no (default: yes)
  output_spectrum yes # yes/no (default: yes)
  output_energy_resolved yes # yes/no (default: yes)
  output_quantum_efficiency yes # yes/no (default: yes)
}

Output generated carriers $G(x)$ and $G(x, E)$ due to photon absorption.

The light field is calculated as a function of position and energy.
strain{}

The documentation for this keyword is also available here (old documentation layout).

Below are various variables and functions that can be used inside strain{}.

Specifying the strain model

**pseudomorphic_strain{}** Homogeneous strain for 1D layer structures (analytical calculation).

This feature also works in 2D or 3D but the user must be sure that the model makes sense from a physical point of view (i.e. the 2D/3D structure should consist of different layers along the growth direction whereas the layers must be homogenous along the two perpendicular directions).

**minimized_strain{}** Minimization of the elastic energy for 2D and 3D geometries (numerical calculation).

It can also be used for 1D simulations. In this case, the results will be equivalent to the analytical model pseudomorphic_strain{}.

**import_strain{}**

**import_from**

value string
ex ample "strain_tensor"

Reference to imported data in import{}.

The data being imported must have exactly 6 components. The expected order of strain tensor components is: \( \varepsilon_{xx} \varepsilon_{yy} \varepsilon_{zz} \varepsilon_{yz} \varepsilon_{zx} \varepsilon_{xy} \)

**coordinate_system**

value string
default simulation
e x ample crystal

The imported strain tensor is with respect to the simulation or crystal coordinate system (optional parameter).

**Note:** In order to calculate the strain, one has to provide a substrate with respect to which the layers are strained. This can be done with the keyword global{} ==> substrate{...}.

Piezo- and pyroelectric charges

**piezo_density**

value yes or no
default yes

Calculate piezoelectric charge density and take it into account while solving the Poisson equation.

If no strain is solved, this flag is ignored.

**pyro_density**

value yes or no
default yes
Calculate pyroelectric charge density and take it into account while solving the Poisson equation. If material system is not wurtzite, this flag is ignored. The pyroelectric charge density due to spontaneous polarization applies to wurtzite only. In order to obtain pyroelectric charges, it is not necessary to calculate strain. Pyroelectric charges are only present in wurtzite materials but not in zinc blende.

second_order_piezo

value yes or no

default no

Include 2nd order piezoelectric coefficients in the calculation

Note: Not fully implemented for wurtzite, only “standard growth directions” supported for wurtzite as the most general formula was not known to us at the time of implementation.

Specify growth direction (for pseudomorphic strain model)

growth_direction

example growth_direction = [1, 0, 0]

• Vector in crystal coordinate system
• Can be specified in a 2D or 3D simulation but not in a 1D simulation (x axis is taken by default in 1D)
• If not set, x axis of simulation coordinate system is taken by default.

Solver definitions

linear_solver{}

iterations

value integer

default 10000

Number of iterations for linear equation solver in strain algorithm

abs_accuracy

value double

default

• 1e-8 [GPa] (1D)
• 1e-8 [GPa nm] (2D)
• 1e-8 [GPa nm2] (3D)

rel_accuracy

range [0.0, 0.01]

default 1e-12 (dimensionless)

use_cscg

value yes or no

Composite step conjugate gradient solver (try this one if standard solver fails to converge)
Debugging

debglevel

  options -1 0 1 2 3

  default 2

The higher this integer number, the more information on the numerical solver is printed to the screen output

Output definitions

output_strain_tensor{ }

  output (symmetric) strain tensor: \( \varepsilon_{ij} = (u_{ij} + u_{ji})/2 \) [dimensionless]

  crystal_system

    value yes or no

    default no

    output strain tensor in crystal coordinate system

  simulation_system

    value yes or no

    default yes

    output strain tensor in simulation coordinate system (useful if simulation coordinate system differs from crystal coordinate system)

    Note: The ordering of the strain tensor components is: \( \varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \varepsilon_{xy}, \varepsilon_{xz}, \varepsilon_{yz} \)

boxes

  value yes or no

  For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)

output_stress_tensor{ }

  output (symmetric) stress tensor: \( \sigma_{ij} = C_{ijkl} \varepsilon_{kl} \) [GPa]

  crystal_system

    value yes or no

    default no

    output stress tensor in crystal coordinate system

  simulation_system

    value yes or no

    default yes

    output stress tensor in simulation coordinate system (useful if simulation coordinate system differs from crystal coordinate system)

    Note: The ordering of the stress tensor components is: \( \sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{xz}, \sigma_{yz} \)

boxes

  value yes or no
**output_hydrostatic_strain** prints out the hydrostatic strain, i.e. the trace of the strain tensor
\[ \text{Tr}[\varepsilon_{ij}] = \varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} \] [dimensionless]

boxes

value yes or no

Note: The hydrostatic strain output is in percent (This is different compared to nextnano³.)

**output_distortion_tensor** outputs distortion tensor \( u_{ij} \) (which can be nonsymmetric for certain growth directions) \( u_{xx} u_{yy} u_{zz} u_{xy} u_{yx} u_{xz} u_{zx} u_{yz} u_{zy} \) [dimensionless]

crystal_system

value yes or no

default no

output distortion tensor in crystal coordinate system

simulation_system

value yes or no

default yes

output distortion tensor in crystal coordinate system

boxes

value yes or no

**output_displacement** outputs displacement vector [nm]

crystal_system

value yes or no

default no

output displacement vector in crystal coordinate system

simulation_system

value yes or no

default yes

output displacement vector in simulation coordinate system

boxes

value yes or no

**output_force_density** outputs force density vector field \( f_i \) [nN/nm³] (at moment output may be not fully correct; not tested sufficiently)

crystal_system

value yes or no

default no

output force density vector field in crystal coordinate system

simulation_system

value yes or no

default yes

output force density vector field in simulation coordinate system

boxes
output_elastic_energy_density()  output elastic energy density \( \frac{1}{2} C_{ijkl} \varepsilon_{ij} \varepsilon_{kl} \) [eV/nm³] The integrated elastic energy is printed out in log file.

boxes

output_polarization_charges()  prints out piezo and pyroelectric charge densities \( \times 10^{18}/\text{cm}^3 \) in case they were calculated. Pyroelectric charges due to spontaneous polarization apply to wurtzite only. Piezoelectric charges can be calculated for both zinc blende and wurtzite in case the strain was calculated. The piezo charge density is written to: density_piezoelectric_charge.dat \( (\rho_{pz}) \) For diamond like crystal structures that have an inversion center such a Si or Ge, piezoelectric charges do not exist.

The pyro charge density is written to: density_pyroelectric_charge.dat \( (\rho_{py}) \) It applies to wurtzite only and is independent of strain and is due to spontaneous polarization. If both, piezo and pyroelectric charge densities were calculated, the sum of both charge densities (total polarization charge density) is written to: density_polarization_charge.dat \( (\rho_{pol} = \rho_{pz} + \rho_{py}) \)

To summarize:

- zincblende: density_piezoelectric_charge.dat \( (\rho_{pz}) \)
- wurtzite: density_piezoelectric_charge.dat \( (\rho_{pz}) \), density_pyroelectric_charge.dat \( (\rho_{py}) \), density_polarization_charge.dat \( (\rho_{pol} = \rho_{pz} + \rho_{py}) \)

output_polarization_vector()  prints out piezo and pyroelectric polarization vector [C/cm²]. Pyroelectric polarization due to spontaneous polarization apply to wurtzite only. The piezoelectric polarization vector depends on strain and it is zero if no strain is present.

crystal_system

default  no

output polarization vector in crystal coordinate system

simulation_system

default  yes

output polarization vector in simulation coordinate system

boxes

default  no

output_sparse_matrix()  output sparse matrix used in strain solver

type  enumerate desired types of sparse matrix output in .mtx format (optional).

options values zero_nonzero, zero_nonzero_absolute, all

default values

- values: output sparse matrix as it is
- zero_nonzero: output matrix containing ‘0’ and ‘1’ for zero and non-zero entries of sparse matrix
- zero_nonzero_absolute: output matrix containing ‘0’ and ‘1’ for zero and non-zero absolute values of entries of sparse matrix
- all: output all types listed above
Output of material parameters

**output_lattice_constants** Output lattice constants.

    boxes
    \text{value \ y\ or \ n}

**output_elastic_constants** Output elastic constants.

    boxes
    \text{value \ y\ or \ n}

**output_piezo_constants** Output piezoelectric constants.

    boxes
    \text{value \ y\ or \ n}

**output_second_order_piezo_constants** Output 2nd order piezoelectric constants.

    boxes
    \text{value \ y\ or \ n}

**output_pyro_constants** Output pyroelectric constants, i.e. spontaneous polarization constants.

    boxes
    \text{value \ y\ or \ n}

**poisson**

Specifications for the Poisson equation.

- \text{charge\_neutral}
- \text{newton\_solver}
- \text{linear\_solver}
- \text{bisection}
- \text{import\_potential}
- \text{electric\_field}
- \text{reference\_potential}
- \text{debuglevel}
- \text{output\_potential}
- \text{output\_electric\_field}
- \text{output\_electric\_displacement}
- \text{output\_electric\_polarization}
- \text{output\_dielectric\_tensor}
- \text{output\_sparse\_matrix}
charge_neutral

  value  yes or no
  default yes

The initial electrostatic potential $\phi$ entering the Newton solver is determined by requiring local charge neutrality. It is determined using the bisection algorithm. If charge_neutral = no is set and neither a potential is imported nor a fixed electric field is set, then the potential is set at non-Dirichlet points to the Fermi level(s) (both Fermi levels are equal at the beginning).

newton_solver{ }

The Newton solver is used for solving the nonlinear Poisson equation. It is solved with a Newton iteration using inexact line search. The Poisson equation is nonlinear because the charge carrier density $\rho$ depends on the electrostatic potential $\phi$, i.e. $\rho(\phi)$. For each Newton step a system of linear equations, $A \cdot x = b$, is solved with a linear solver, in order to obtain a gradient. This gradient is used for the inexact line search. Generally, low temperature simulations make the Poisson equation extremely nonlinear at the beginning of the iteration and thus require more line search steps than usual. Using debuglevel = 2 displays information on the line search steps (search_steps): In the .log file of your simulation, you can find more information on the convergence of the Newton solver. Parameters for solver of nonlinear poisson equation are as follows:

iterations
  value  any integer > 1
  default 40

Number of iterations for Newton solver

search_steps
  value  any integer between 1 and 50
  default 30

residual
  value  any float
  default 1e3 cm$^{-2}$ (1D)
  1e1 cm$^{-1}$ (2D)
  1e-4 [dimensionless] (3D)

gradient_shift
  value  any float between the interval -1e6 and 1e-6
  default 1e-13

linear_solver{ }

Parameters for linear equation solver in Newton algorithm.

iterations
  value  any integer > 1
  default 1000

number of iterations for linear equation solver

abs_accuracy
  value  any float > 0
### Parameters for bisection search

Used for the initial solution of the Poisson equation when `charge_neutral = yes` is set. Bisection is performed in order to achieve local charge neutrality at each grid point:

\[
\rho = p - n + \text{sum}(ND, \text{ionized}) - \text{sum}(NA, \text{ionized}) = 0
\]

Thus, a true classical charge neutrality is computed for classical carrier and doping situations. Additionally, bisection is also used to determine the electrostatic potential at which contacts become charge neutral, which is also needed for e.g. ohmic contacts and Schottky contacts using Schottky barriers, but not for Fermi contacts as inside them the electrostatic potential is initialized exactly the same way as specified outside/between of contacts. The bisection for contacts is performed in any case, i.e. independently from the bisection used when `charge_neutral = yes` is set. The bisection method is a well known algorithm for finding the root of a function. The delta is the so-called convergence tolerance parameter. Specifically in nextnano++ we use this method to find the initial solution of the Poisson equation that generally converges very fast using the default parameters and no extra tuning is required.

### delta

- Default: \(1 \times 10^{-3} \text{ cm}^{-1}(2D)\)
- \(1 \times 10^{-8} \text{ [dimensionless]}(3D)\)

### rel_accuracy

- Value: any float between 0.0 and 0.01
- Default: \(1 \times 10^{-13}\)

### dkr_value

- Value: any float < 0.5
- Default: 0.0

(“magic parameter” to speed up calculations, affects preconditioning. Negative values are ignored but will switch to a slightly slower but more stable preconditioner.)
Range of bisection search.

iterations

value any integer > 1
default 40

residual

value any integer > 1
default 1e3 cm^-3

For the other bisection used for contacts, the values from the input file are internally modified in that iterations is always increased to be at least 40 and residual is reduced to be at most 1e3 cm^-3, i.e. the contact setup ignores bisection definitions which are weaker than these default settings. Note: In GaN, the intrinsic density at T=300 K is of the order 1e-10 cm^-3, in AlN even smaller, so the residual needs to be adjusted in some cases. Making the default values smaller may result in significantly longer initialization times, especially in 3D, and will provide no benefit for other materials than wide band gap semiconductors (e.g. nitrides). Also note that low temperatures cause low densities.

import_potential{ }

Import electrostatic potential from file or analytic function and use it as initial guess for solving the Poisson equation. If no Poisson equation is solved, the imported data determines the electrostatic potential that is used throughout the simulation, i.e. in this case an electrostatic potential can be read in that is fixed during the rest of the simulation and is used as input to the Schrödinger equation and for the calculation of the densities. The solution obtained from a problem solved previously using a different meshing is accepted.

import_from

value path string

Reference to imported data in import{}. The data may have more than one component (e.g. vector field).

component_number

value integer

default 1

If imported data is a vector field, one may want to specify the component.

electric_field{ }

strength

value any float

default 0.0 V/m

Defines a constant electric field in the structure. If electric_field is defined, and the absolute value is larger than zero, then it is being used for the electrostatic potential calculation.

direction

value 3D float vector

default [1.0, 0.0, 0.0]

Orientation of electric field vector with respect to (x, y, z) simulation coordinate system. For 1D simulations, the direction can be omitted and in this case the default will be used.
reference_potential

    value  any float
    default 0.0

If electric_field is defined, this value in units of [V] is being added to the electrostatic potential.

dbglevel

    value  any integer between -1 and 3
    default 1

The higher this integer number, the more information on the numerical solver is printed to the screen output. Increasing the respective debuglevel to 2 or more significantly increases the volume of the diagnostic output displayed in nextnanomat (or a shell window). As result of the additional I/O load, particularly 1D simulations will slow down correspondingly (especially for current and poisson)

output_potential

Prints out the electrostatic potential in [eV].

output_electric_field

Prints out the electric field in kv/cm.

output_electric_displacement

Prints out the output electric displacement

output_electric_polarization

Prints out the output electric polarization

output_dielectric_tensor

Prints out the output dielectric tensor in simulation coordinate system, as it is used while setting up the sparse matrix for the Poisson solver.

    boxes
    
    value yes or no
    default no

For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at interfaces (in 2D four points, in 3D eight points)
output_sparse_matrix{ }

Prints out the output sparse matrix used in Poisson solver.

  type
    value  string list
    default values

Options for the string value list are as follows * values- output sparse matrix as it is (also imaginary part, if sparse matrix is complex valued) * zero_nonzero- output matrix containing '0' and '1' for zero and non-zero entries of sparse matrix (same for imaginary part, if sparse matrix is complex valued) * zero_nonzero_absolute- output matrix containing '0' and '1' for zero and non-zero absolute values of entries of sparse matrix * all- output all types listed above

quantum()

Specifications that define quantum models, i.e. how the Schrödinger equation should be solved.

quantum{
  debuglevel  
  allow_overlapping_regions  
  region{ }
  exchange_correlation{ }
}

```plaintext
quantum{
  debuglevel
  allow_overlapping_regions  no

  #----------------
  # Quantum regions
  #----------------
  region{
    name "qr1"
    quantize_x{}
    quantize_y{}
    quantize_z{}
    no_density yes
    x [10.0, 20.0]
    y [10.0, 20.0]
    z [10.0, 20.0]

    # Boundary conditions
    boundary{
      x ...[1,1]
      y ...[1,1]
      z ...[1,1]
      classical_boundary_x no
      classical_boundary_y no
      classical_boundary_z no
      num_classical_x [1,1]
      num_classical_y [1,1]
      num_classical_z [1,1]
    }
  }
}
```

(continues on next page)
# Output definitions

```plaintext
output_wavefunctions{
  max_num = 10
  all_k_points = yes
  structured = no
  amplitudes = "S X Y Z C B _ H H _ L H _ S O"
  probabilities = "yes"
  scale = 0.7
  in_one_file = yes
  energy_shift = both
  include_energies_in_shifted_files = yes
}
output_subband_densities{
  max_num = 10
  in_one_file = yes
}
output_sparse_matrix{
  type = all
  structured = no
}
output_rotated_inverse_mass_tensor{
  boxes = yes
  structured = no
}
```

# Quantum models and solver definitions

```plaintext
# Eigensolvers (choose one)
lapack()
arpack()
accuracy = 1e-6
iterations = 200
preconditioner = chbsh

cutoff = 0.3
abs_cutoff = 2.5
order_chebyshev = 20

# Dispersion

dispersion{
  path{
    name = "100"
    point{
      k = [1.0, 0.0, 0.0]
      k = [1.0, 1.0, 0.0]
    }
    spacing = 0.5
    num_points = 10
  }
  lines{
    name = "lines"
    spacing = 0.5
    k_max = 1.0
  }
  full{
    name = "3D"
    kxgrid{
      line{
        ...
      }
    }
  }
}
```

(continues on next page)
pos -1
    spacing 0.02
}
)
kygrid{
    line{
        pos -1
        spacing 0.02
    }
}
kzgrid{
    line{
        pos -1
        spacing 0.02
    }
}
}
}
}
}

superlattice{
    name "superlattice"
    num_points_x 10
    num_points_y 15
    num_points_z 20
    num_points 20
}
}
}
L{ }
}
X{ }
}
Delta{ }
}
HH{ }
}
LH{ }
}
SO{ }
}
kp_6band{
    kp_parameters{
        use_Luttinger_parameters no
        approximate_kappa no
    }
}
lapack{} #arpack()
k_integration{
    relative_size 0.2
    num_points 5
    num_subpoints 2
    max_symmetry no
    force_k0_subspace yes
}
}
kp_8band{
    num_electrons 6
    num_holes 12
    accuracy 1e-8
    iterations 200
    kp_parameters{
        use_Luttinger_parameters no
        from_6band_parameters no
        approximate_kappa no
        evaluate_S no
        rescale_S_to 1.0
    }
    k_integration{
        (s a m e s k p_6band)
    }
    lapack{}
    #arpack_inv{}
    shift_window 0
    shift 0.2
    abs_shift 2.5
    linear_solver{
        iterations 500
        abs_accuracy 1e-9
        rel_accuracy 1e-9
        use_cscg no
        force_diagonal_preconditioner no
    }
    #advanced settings for 8-band k.p quantum density
    shift_min_CB 0.0
    shift_max_VB 0.0
    tunneling yes
    classify_kspace 0
    threshold_classification 0.5
    full_band_density no
}
#Matrix elements definitions
#---------------------------------
interband_matrix_elements{
    KP6_Gamma{
        direction [1,1,0]
    }
    HH_Gamma( HH_Gamma( ) < HH_i | Gamma_j >)
    LH_Gamma( LH_Gamma( ) < LH_i | Gamma_j >)
(continued from previous page)

```c
SO_Gamma{[1,1]}  # < SO_i | Gamma_j >
HH_Delta{[1,1]}  # < HH_i | Delta_j >
LH_Delta{[1,1]}  # < LH_i | Delta_j >
SO_Delta{[1,1]}  # < SO_i | Delta_j >
HH_X{[1,1]}      # < HH_i | X_j >
LH_X{[1,1]}      # < LH_i | X_j >
SO_X{[1,1]}      # < SO_i | X_j >
HH_L{[1,1]}      # < HH_i | L_j >
LH_L{[1,1]}      # < LH_i | L_j >
SO_L{[1,1]}      # < SO_i | L_j >

output_matrix_elements  yes
output_transition_energies  yes

intraband_matrix_elements{
  Gamma{
    direction  [1,1,0]
  }
  Delta{[1,1]}
  X{[1,1]}
  L{[1,1]}
  HH{[1,1]}
  LH{[1,1]}
  SO{[1,1]}
  KP6{[1,1]}
  KP8{[1,1]}

  output_matrix_elements  yes
  output_transition_energies  yes
  output_oscillator_strengths  yes
}

dipole_moment_matrix_elements{
  Gamma{
    direction  [1,1,0]
  }
  Delta{[1,1]}
  X{[1,1]}
  L{[1,1]}
  HH{[1,1]}
  LH{[1,1]}
  SO{[1,1]}
  KP6{[1,1]}
  KP8{[1,1]}

  output_matrix_elements  yes
  output_transition_energies  yes
  output_oscillator_strengths  yes
}

transition_energies{
  Gamma{}
  KP6_Gamma{}
  HH_Gamma{}
  LH_Gamma{}
  SO_Gamma{}
  Delta{}
  HH_Delta{}
  LH_Delta{}
  SO_Delta{}
```

(continues on next page)
X{}
HH_X{}
LH_X{}
SO_X{}
L{}
HH_L{}
LH_L{}
SO_L{}
HH{}
LH{}
SO{}
KP6{}
KP8{}

```
calculate_lifetime{
  phonon_energy = 0.036
}

bulk_dispersion{
  path{
    name = "from_Gamma_to_L"
    position{
      x = 5.5
      y = 10.0
      z = -1.1
    }
    shift_holes_to_zero = yes
    point{
      k = [1.0, 0.0, 0.0]
    }
    spacing = 0.5
    num_points = 10
  }
  lines{
    name = "lines"
    position{
      x = 5.5
      y = 10.0
      z = -1.1
    }
    shift_holes_to_zero = yes
    spacing = 0.5
    k_max = 1.0
  }
  full{
    name = "3D"
    position{
      x = 5.5
      y = 10.0
      z = -1.1
    }
    shift_holes_to_zero = yes
    kxgrid{
      line{
        pos = -1
      }
    }
  }
}
```
debuglevel

value any integer between -1 and 3

default 1

The higher this integer number, the more information on the numerical solver is printed to the screen output. Increasing the respective debuglevel to 2 or more significantly increases the volume of the diagnostic output displayed in nextnano (or a shell window). As result of the additional I/O load, particularly 1D simulations will slow down correspondingly (especially for current{ } and poisson{ }).

allow_overlapping_regions

value yes or no

default no

Overlapping quantum regions computing the same band(s) are not allowed. Note that, in case such overlap is allowed, the quantum densities of the respective regions are added in the overlap region and a too high density will be computed. Thus, please only allow such overlap when the quantum densities are known to be extremely small in the overlap region.
region{ }

Inside the quantum region, the Schrödinger equation is solved.

name
value “string”

Provides the name of the quantum region.

quantize_x{}
In 2D or 3D simulation, the Schrödinger equation is solved within the slices perpendicular to x-direction. This results in the reduction of the calculation time.

For example, if a 2D simulation has 100 grids in x-direction and 50 grids in y-direction, the normal calculation solves the eigenvalue problem of a (100x50) x (100x50) matrix. When quantize_x{} is specified, on the other hand, nn++ solves the 1D Schrödinger equations along y-direction at each grid point in x-direction so 100 eigenvalue problems of 50x50 matrixes are solved. Thus the runtime of the eigenvalue solver could be roughly estimated as (number of x-grids)\(^{-1}\) times, but we should note that the runtime also depends on the number of eigenvalues to be calculated.

Currently, only one-band (Gamma, X, Delta, LH, HH, etc.) without k-integration and without magnetic field is supported, and QM output is limited to local spectra and occupations. If strain is enabled, deformation potentials are ignored. Similarly, quantum boundary conditions are always Neumann or periodic, irrespective of what is specified in the input file. And quantum decomposition regions cannot be used for CBR or optics.

Only one quantization direction (x, y, z) can be simultaneously specified when quantum decomposition is used. Typically, the quantization direction is the growth direction.

Note that a similar number of states should be requested as for a corresponding 1D simulation (i.e. much less than normally needed in 2D or 3D), and that lateral (i.e. orthogonal to the quantization direction) grid spacings can be much larger than for “normal” quantum simulation, as the density from quantum decomposition is NOT affected by wide lateral grid spacings.

quantize_y{}
The same as quantize_x{}, but the slices are in y-direction.

quantize_z{}
The same as quantize_x{}, but the slices are in z-direction.

no_density
value yes or no
default no

Tells if to not calculate quantum mechanical charge density.

x
value 2D float vector

Provides the extension of quantum region in x direction in nanometers (nm)

y
value 2D float vector

Provides the extension of quantum region in y direction in nanometers (nm). To be used for 2D or 3D calculations only.

z
value 2D float vector

Provides the extension of quantum region in x direction in nanometers (nm). To be used for 3D calculation only.

• boundary{ }
Specifies the boundary condition for Schrödinger equation along various axis dimensions. In general, Dirichlet boundary conditions correspond to \( f = \text{constant} \) and Neumann boundary conditions correspond to \( df/dx = \text{constant} \). Quantum densities may
exhibit pathological density values on the boundary (e.g. 0 in the case of Dirichlet boundary conditions). Using \texttt{classical\_boundary\_x}, \texttt{classical\_boundary\_y}, \texttt{classical\_boundary\_z}, the computation of a classical density can be enforced on the respective boundary points for the respective band(s). (The quantum calculation itself and respective results such as wavefunctions are not affected by this setting). Using \texttt{num\_classical\_x}, \texttt{num\_classical\_y}, \texttt{num\_classical\_z} you can explicitly specify the number of points to be cut at each side.

\begin{verbatim}
x value dirichlet/neumann/cbr
default neumann

y value dirichlet/neumann/cbr
default neumann

z value dirichlet/neumann/cbr
default neumann

classical\_boundary\_x
value yes or no
default no

classical\_boundary\_y
value yes or no
default no

classical\_boundary\_z
value yes or no
default no

classical\_boundary\_x
value 2D integer vector
default [1, 1]

classical\_boundary\_y
value 2D integer vector
default [1, 1]

classical\_boundary\_z
value 2D integer vector
default [1, 1]
\end{verbatim}

\textbf{Note:} Periodic boundary conditions along the appropriate direction(s) are taken automatically if \texttt{global \{ \ldots periodic{ x/y/z = yes} \}} is specified \textbf{and} if the quantum region extends over the whole simulation region along the appropriate direction. In this case, the \texttt{dirichlet} or \texttt{neumann} specifications under \texttt{quantum\{ \ldots \{region\{ \ldots \{boundary\{ \ldots \} \} \} \} \}} are ignored along the appropriate direction(s).

* \texttt{output\_wavefunctions\{ \}} Provides options for output of wavefunction data

\begin{verbatim}
max_num
\end{verbatim}
value  any integer between 1 and 9999

default  1.0

**all_k_points**

value  yes or no

default  false

Prints out the wavefunctions for all \( k \) points (1D: \( k_1 = (k_y, k_z) \), 2D: \( k_1 = k_z \)) that are used in the \( k\_integration \) or \( dispersion \). Enabling this option can produce a large number of output files.

**structured**

value  yes or no

default  no

The whole output for \( quantum \) is written in subdirectory \( Quantum/ \). If enabled, additional subdirectories are created in subdirectory \( Quantum/ \) to organize the structure of the output files in a meaningful way. It is recommended to set this parameter to yes if a lot of output files are created, e.g. in case \( all_k_points = yes \), and both amplitudes and probabilities are printed out.

**amplitudes**

value  string

default  "no"

Prints out the wavefunctions \( \psi \) in units of 1D: \( nm^{-1/2} \), 2D: \( nm^{-1} \), 3D: \( nm^{-3/2} \).

**options**

"yes" : for \( k.p \) it is equivalent to \( S_X_Y_Z \)

"no" : no output is done for amplitudes.

"S_X_Y_Z" : prints out the wavefunctions \( \psi \) with respect to the basis \( |S+\rangle |S-\rangle |X+\rangle |Y+\rangle |Z+\rangle |X-\rangle |Y-\rangle |Z-\rangle \) corresponding to the x, y, z of the simulation coordinate system (and not crystal coordinate system) and + and - correspond to the spin projection along the z axis of the crystal system.

"CB_HH_LH_SO" : prints out the wavefunctions \( \psi \) with respect to the basis \( |cb+\rangle |cb-\rangle |hh+\rangle |lh+\rangle |lh-\rangle |hh-\rangle |so+\rangle |so-\rangle \). This basis is the same as used in L. C. Lew Yan Voon, M. Willatzen, *The k.p method* (2009) (Table 3.4); G. Bastard, *Wave Mechanics Applied to Semiconductor Heterostructures* (1988) and B. A. Foreman, PRB 48, 4964 (1993).

If multiple choices are required type them together inside a string like

```
amplitudes "S_X_Y_Z,CB_HH_LH_SO"
```

**probabilities**

value  string

default  yes

Prints out the wavefunctions \( |\psi|^2 \) in units of 1D: \( nm^{-1} \), 2D: \( nm^{-2} \), 3D: \( nm^{-3} \).

**options**

yes : for \( k.p \) it is the sum of the squares of all components of a spinor

no : no output

S_X_Y_Z : same as for the amplitudes \( (k.p \) only)
CB_HH_LH_SO : same as for the amplitudes \((k,p)\) only

If multiple choices are required type them together inside a string like

```plaintext
probabilities = "yes"
```

scale

| value  | float
|--------|
| default| 1.0

scale factor for output of amplitudes and probabilities

in_one_file

| value  | yes or no
|--------|
| default| yes

Prints out the amplitudes into one file and the probabilities into one file. If no is chosen, for each eigenvalue a separate file is written out.

energy_shift

| value  | string
|--------|
| default| both

options shifted : prints out the amplitudes and the probabilities shifted by the energy.

not_shifted : prints out the amplitudes and the probabilities as they are (an integral over volume is equal to 1).

both : prints out the amplitudes and the probabilities with and without energy shift.

include_energies_in_shifted_files

| value  | yes or no
|--------|
| default| yes

Selects if the energy levels are added in output of shifted amplitudes and probabilities or not. If no is selected a separate file with energy levels is written out.

**Note:** The energy spectrum (i.e. the eigenvalues) are always written into the files `energy_spectrum_*.dat`. The projections of the eigenfunctions on the basis states of the bulk Hamiltonian are written into the files `spinor_composition_*.dat`.

- output_subband_densities

  | value  | any integer between 0 and 9999
  |--------|
  | default| 1

number of subband densities to be printed out. If `max_num` is not present, the subband density is written out for each eigenvalue.

in_one_file

| value  | yes or no
|--------|
| default| yes
Prints out the subband densities into one file. If no is chosen, for each subband density a separate file is written out. This feature only makes sense for 1D simulations.

- **output_sparse_matrix{ }** Provides options for output of sparse matrix, used in Schroedinger equation solver, in .mtx format

  **type**
  
<table>
<thead>
<tr>
<th>value</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>values</td>
</tr>
</tbody>
</table>

  **options values**
  
  - zero_nonzero : output matrix containing ‘0’ and ‘1’ for zero and non-zero entries of sparse matrix (same for imaginary part, if sparse matrix is complex valued)
  
  - zero_nonzero_absolute : output matrix containing ‘0’ and ‘1’ for zero and non-zero absolute values of entries of sparse matrix
  
  - all : output all types listed above

  **structured**
  
<table>
<thead>
<tr>
<th>value</th>
<th>yes or no</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>no</td>
</tr>
</tbody>
</table>

  Whole output is written in subdirectory Quantum/. IF yes is selected additional subdirectories are created in subdirectory Quantum/ to organize the structure of the output files in a meaningful way.

- **Gamma{ }** Solves single-band effective mass Schrödinger equation for the Gamma conduction band.

  **num_ev**
  
<table>
<thead>
<tr>
<th>value</th>
<th>integer &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>0</td>
</tr>
</tbody>
</table>

  Provides the number of eigenvalues to be calculated.

  **force_complex_solver**
  
<table>
<thead>
<tr>
<th>value</th>
<th>yes or no</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>no</td>
</tr>
</tbody>
</table>

  Set flag to yes, when optics/ is complaining about real-valued quantum solvers.

- **lapack{ }** LAPACK eigensolver is used to solve dense matrix problem (should be used for 1D and small 2D systems). For 1D simulations without periodic boundary conditions a tridiagonal LAPACK solver is used for the single-band Hamiltonian as default.

- **arpack{ }** ARPACK eigensolver is used to solve eigenvalue problem using sparse matrix routines. It ARPACK should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30). Memory usage of arpack (and also arpack_inv) only depends on the number of eigenvectors requested, and is not influenced by the type of preconditioner used. Essentially, for each requested eigenvector (i.e. wave function), additional temporary space corresponding to 2.5 eigenvectors is needed during runtime. Among the preconditioners, chebyshev preconditioning and legendre preconditioning are comparably fast, but require both the specification of a cutoff energy under (above) which all eigenvalues of interest are assumed to be located. If this assumption is violated, only spurious parts of the energy spectrum will be computed. On the other hand, setting the cutoff energy too generous will slow down convergence. Since the energy spectrum often shifts during the Quantum-Poisson iteration, a more generous initial cutoff energy is also needed for the first Quantum-Poisson iteration step. If this initial
cutoff energy is not provided, much slower but more predictable polynomial preconditioning will be used for the first Quantum-Poisson iteration step instead of the specified chebyshev / legendre preconditioner. Alternatively, this slower polynomial preconditioning can also be used for the entire Quantum-Poisson iteration. In this case, no cutoff energies need to be specified at all. Generally, it is advisable to use polynomial preconditioning when simulating a new structure until the distribution of the eigenvalues, the location of the Fermi level(s), and the required numbers of eigenvalues are better known. Performance of all preconditioners can be further tuned by changing the order of the respective polynomial used, with optimal values typically lying between 10 and 30. arpack will terminate once the desired accuracy has been reached or the specified number of iterations has been exceeded. In the latter case, not all requested eigenvectors may have been calculated, or convergence may be incomplete.

accuracy
  value  any float > 0
  default  1e-10 for LAPACK 1e-7 for ARPACK
accuracy of eigenvalue

iterations
  value  any integer > 1
  default  500
number of iterations for eigenvalue solver

preconditioner
  value  0 or polynomial, 1 or chebyshev, 2 or legendre
  default  1 or chebyshev
Polynomial preconditioner is the slowest but does not require to specify cutoff energy whereas chebyshev or legendre preconditioner requires you to specify cutoff energy.

order_polynomial
  value  any integer > 1
  default  15
order of the polynomial used for polynomial preconditioning

order_chebyshev
  value  any integer > 1
  default  20
order of the polynomial used for Chebyshev preconditioning

order_legendre
  value  any integer > 1
  default  20
order of the polynomial used for Legendre preconditioning

cutoff
  value  any float >= 0.001
  default  0.3 # [eV]

abs_cutoff
  value  any float >= 0.001
  default  0.0 # [eV]
**Note:** The default behaviour of ARPACK eigensolver is the following: When the Schrödinger equation is solved for the first time, the polynomial preconditioner is used, because there is no suitable cutoff energy known. In all later Quantum-Poisson iterations the chebyshev preconditioner will be used (up to two times faster) with a cutoff energy slightly above the highest eigenvalue, which was calculated in the last iteration.

- **dispersion** \{ \} calculate the $k_{||}$ and $k_{superlattice}$ (if applicable) dispersion. The energy dispersion $E(k)$ along the specified paths and for the specified $k$ space resolutions are completely independent from the $k$ space resolution that was used within the self-consistent cycle where the k.p density has been calculated. The latter is specified in $k\_integration$. For more details, see `dispersion`.

- **L**\{ \} solves single-band Schrödinger equation for the $L$ conduction band. The options are the same as `Gamma`.

- **X**\{ \} solves single-band Schrödinger equation for the $X$ conduction band. The options are the same as `Gamma`.

- **Delta**\{ \} solves single-band Schrödinger equation for the $Delta$ conduction band. The options are the same as `Gamma`.

- **HH**\{ \} solves single-band Schrödinger equation for the **heavy hole** valence band. The options are the same as `Gamma`.

- **LH**\{ \} solves single-band Schrödinger equation for the **light hole** valence band. The options are the same as `Gamma`.

- **SO**\{ \} solves single-band Schrödinger equation for the **split-off hole** valence band. The options are the same as `Gamma`.

- **kp_6band**\{ \} solves 6-band k.p Schrödinger equation for the **heavy, light and split-off hole** valence band. The options are the same as `Gamma` with some additional options, which are

  - **kp\_parameters**\{ \} advanced manipulation of k.p parameters from the database.

  **use_Luttinger\_parameters**

  **value** yes or no

  **default** no

  By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (L, M, N). If enabled then it uses Luttinger parameters ($\gamma_1, \gamma_2, \gamma_3$) instead.

  **approximate\_kappa**

  **value** yes or no

  **default** no

  By default the $\kappa$ for zinc blende crystal structure is taken from the database or input file. If this is enabled then the solver is forced to approximate kappa through others 6-band k.p parameters, even though kappa is given in database or input file.

- **lapack**\{ \} LAPACK eigensolver: solves dense matrix problem (for 1D and small 2D systems only)

- **arpack**\{ \} ARPACK eigensolver (default) ARPACK should be faster for large matrices ($N > 1000$) where only a few eigenvalues are sought (~5-30).

  - **k\_integration**\{ \}
Provides options for integration over \(k_{||}\) space for \(k.p\) density calculations (for 1D and 2D only). By default the quantum mechanical charge density is calculated \(\text{no_density} = \text{no}\). Therefore, \(k\_integration()\) is required. If you do not need a quantum mechanical density, e.g. because you are not interested in a self-consistent simulation, the calculation is much faster if you use \(\text{no_density} = \text{yes}\). Then you can omit \(k\_integration()\) and only the eigenstates for \(k_{||} = (k_y, k_z) = (0,0) = 0\) are calculated.

**relative_size**

- **value**: float between 0.0 and 1.0
- **default**: 1.0

Range of \(k_{||}\) integration relative to size of Brillouin zone. Often a value between 0.1-0.2 is sufficient.

**num_points**

- **value**: integer > 1
- **default**: 10

Number of \(k_{||}\) points, where Schrödinger equation has to be solved (in one direction). In 1D, the number of Schrödinger equations that have to be solved depends quadratically on \(\text{num_points}\). In 2D, the number of Schrödinger equations that have to be solved depends linearly on \(\text{num_points}\).

**num_subpoints**

- **value**: integer > 1
- **default**: 5

Number of points between two \(k_{||}\) points, where wavefunctions and eigenvalues will be interpolated.

**max_symmetry**

- **value**: 1 or no 2 or \(C_2\) 3 or full
- **default**: full

no does not use symmetry of Brillouin zone to reduce number of \(k_{||}\) points.

\(C_2\) uses up to \(C_2\) symmetry of Brillouin zone to reduce number of \(k_{||}\) points.

full uses full symmetry of Brillouin zone to reduce number of \(k_{||}\) points. For example for a cubic \(k\) space the 1/8th of the zone.

**force_k0_subspace**

- **value**: yes or no
- **default**: no

If set to yes, \(k_{||}\) integration in quantum() is modified in that only states for point \(k = 0\) are computed exactly, whereas all other \(k\) points are computed in the subspace of the \(k = 0\) wavefunctions. As a result of this approximation, computational speed is much improved (you may even be able to also enlarge the number of eigenvalues). In case you are planning to use this approximation for final results, please make sure to check whether the resulting loss of accuracy in density is acceptable.

- **kp_8band**

  It solves 8-band \(k.p\) Schrödinger equation for the Gamma conduction band and the heavy, light and split-off hole valence bands.
num_electrons
    value integer >= 0
    default 0
number of electron eigenvalues
num_holes
    value integer >= 0
    default 0
number of hole eigenvalues
accuracy
    value any float > 0
    default 1e-7
accuracy of eigenvalue
iterations
    value any integer > 1
    default 500
number of iterations for eigenvalue solver
    - kp_parameters

    Provides options for advanced manipulation of k.p parameters from database.

use_Luttinger_parameters
    value yes or no
    default no
By default the solver uses the DKK (Dresselhaus-Kip-Kittel) parameters (\(L, M, N\)). If enabled then it uses Luttinger parameters \((\gamma_1, \gamma_2, \gamma_3)\) instead.

from_6band_parameters
    value yes or no
    default no
By default the 8-band k.p parameters are taken from database or input file. If enabled then it evaluates the 8-band k.p parameters from 6-band k.p parameters, Kane parameter \(E_P\) and temperature dependent band gap \(E_g\).

approximate_kappa
    value yes or no
    default no
By default the \(\kappa\) for zinc blende crystal structure is taken from the database or input file. If this is enabled then the solver is forced to approximate kappa through others 8-band k.p parameters, even though kappa is given in database or input file.

evaluate_S
    value yes or no
    default no
By default $S$ ($S_1$, $S_2$ for wurtzite) $k.p$ parameter(s) is (are) taken from database or input file. If enabled it evaluates $S$ ($S_1$, $S_2$ for wurtzite) $k.p$ parameter(s) from effective mass $m_e (m_{e,par}, m_{e,perp}$ for wurtzite), Kane parameter(s), spin-orbit coupling(s) and temperature dependent band gap.

rescale_S_to

value float for zinc blende crystal structure

2D float vector for wurtzite crystal structure

set $S$ for zinc blende crystal structure to specified value and rescale $E_p$, $L', N^+$ in order to preserve electron’s effective mass.

set $S_1$, $S_2$ for wurtzite crystal structure to specified values respectively and rescale $E_{P1}$, $E_{P2}$, $L_1'$, $L_2'$, $N_1^+$, $N_2^+$ in order to preserve electron’s effective masses.

– k_integration{ } Provides options for integration over $k_{||}$ space for $k.p$ density calculations (for 1D and 2D only) same as kp_6band{ k_integration{ }}

lapack{ } LAPACK eigensolver: solves dense matrix problem (for 1D and small 2D systems only)

arpack_inv{ } ARPACK shift invert eigensolver. ARPACK should be faster for large matrices (N > 1000) where only a few eigenvalues are sought (~5-30).

shift_window

value integer

default 0

When LAPACK is used, shifts the window of computed states by the specified number of states up (for positive integers) or down (for negative integers). Adjust when the computed states are not centered around the band gap.

shift

value float $>=0$

default 0.1 # [eV]

energy shift relative to band edges in arpack_inv.

abs_shift

value float $>=0$

default 0.0 # [eV]

energy shift on an absolute energy scale in arpack_inv.

– linear_solver{ }

Provides parameters for linear equation solver in arpack_inv shift invert preconditioner

iterations

value integer $>1$

default 10000

number of iterations in arpack_inv. Occasionally, using even larger values than 10000 may be necessary to avoid diagonalization failure.

abs_accuracy

value float between 0.0 and 0.01

default 1e-8
absolute accuracy in arpack_inv.

rel_accuracy

value float between 0.0 and 0.01

default 1e-8

relative accuracy in arpack_inv.

use_cscg

value yes or no

default no

When arpack_inv is used, forces the slower but occasionally more robust CSCG (Composite Step Conjugate Gradient) linear solver to be used rather than the cg (Conjugate Gradient) linear solver. May occasionally prevent a diagonalization failure.

force_diagonal_preconditioner

value yes or no

default no

When arpack_inv is used, forces the use of a slower but more robust diagonal preconditioner. As result, total runtime and stability of the arpack_inv solver may actually become much better and diagonalization failures may be avoided.

shift_min_CB

value float

default 0.0

(relevant only if classify_kspace = 0) Shifts the minimum of the conduction band to manipulate cutoff energy and thereby the quantum density classification.

shift_max_VB

value float

default 0.0

(relevant only if classify_kspace = 0) Shifts the maximum of the valence band to manipulate cutoff energy and thereby the quantum density classification.

tunneling

value yes or no

default yes

(relevant only if classify_kspace = 0) Choice of the (position-dependent) cutoff energy. yes defines the cutoff energy at max((minimum of the conduction band in the structure), (position-dependent valence band edge)), while no sets it to min((maximum of the valence band in the structure), (position-dependent conduction band edge)).

classify_kspace

value 0, 1, 2, or 3

default 0

Choice of the classification method in the 8-band k.p quantum density calculation.

- classify_kspace = 0: Eigenstates are classified by comparing the zone-center eigenvalues with the (possibly position-dependent) cutoff energies. For the definition of cutoff energies, see shift_min_CB, shift_max_VB, and tunneling.
classify_kspace = 1: Eigenstates are classified by comparing the zone-center spinor composition with threshold_classification.

classify_kspace = 2: Eigenstates are classified at each in-plane k vector (1D simulation) and at each k value (2D simulation) using spinor composition averaged with the neighbouring k points.

classify_kspace = 3: Eigenstates are classified at each in-plane k vector (1D simulation) and at each k value (2D simulation) using spinor composition averaged with the neighbouring k points, but skipping the average if any of the neighbouring k points has the opposite sign of charge. The resulting quantum density will be different from the case classify_kspace = 2 if electron-hole hybridization occurs (e.g. type-II broken-gap superlattices).

threshold_classification

value \(0.0 \leq \text{float} \leq 1.0\)

default 0.5

(relevant only if classify_kspace \(\geq 1\)) Classify states to electrons if the electron spinor composition is greater than this threshold and otherwise to holes.

full_band_density

value yes or no

default no

Calculate density by filling all states above Fermi level with holes and subtracting a negative background charge (lapack only). This ignores classify_kspace.

- interband_matrix_elements

  Provides the option to calculate interband matrix elements between wave functions of two different bands.

  KP6_Gamma\{\} \(\sum_k (kp_{6k,i}|\Gamma_j)\), with k = 1 .. 6 indexing the component of the six-component k.p wave function and i, j indexing the wave function numbers. kp_6band{ } and Gamma{ } calculation must be present.

  HH_Gamma\{\} Matrix element of the transition between the heavy hole valence band and the gamma conduction band \(\langle HH_i|\Gamma_j\rangle\)

  LH_Gamma\{\} Matrix element of the transition between the light hole valence band and the gamma conduction band \(\langle LH_i|\Gamma_j\rangle\)

  SO_Gamma\{\} Matrix element of the transition between the split-off hole valence band and the gamma conduction band \(\langle SO_i|\Gamma_j\rangle\)

  HH_Delta\{\} Matrix element of the transition between the heavy hole valence band and the Delta conduction band \(\langle HH_i|\Delta_j\rangle\)

  LH_Delta\{\} Matrix element of the transition between the light hole valence band and the Delta conduction band \(\langle LH_i|\Delta_j\rangle\)

  SO_Delta\{\} Matrix element of the transition between the split-off hole valence band and the Delta conduction band \(\langle SO_i|\Delta_j\rangle\)

  HH_X\{\} Matrix element of the transition between the heavy hole valence band and the X conduction band \(\langle HH_i|X_j\rangle\)

  LH_X\{\} Matrix element of the transition between the light hole valence band and the X conduction band \(\langle LH_i|X_j\rangle\)

  SO_X\{\} Matrix element of the transition between the split-off valence band and the X conduction band \(\langle SO_i|X_j\rangle\)

  HH_L\{\} Matrix element of the transition between the heavy hole valence band and the L conduction band \(\langle HH_i|L_j\rangle\)
LH_L{ } Matrix element of the transition between the light hole valence band and the L conduction band $\langle LH_i | L_j \rangle$

SO_L{ } Matrix element of the transition between the split-off valence band and the L conduction band $\langle SO_i | L_j \rangle$

output_matrix_elements = yes/no  Output matrix elements.

output_transition_energies = yes/no  Output transition energies.

• intraband_matrix_elements{ } Calculate intraband matrix elements $\langle i | \epsilon \cdot \hat{p} | j \rangle$ for wave functions within one band. The light polarization direction $\epsilon$ is automatically normalized in the program. $\hat{p} = i \hbar \nabla$ is the momentum vector.


Gamma{ } Calculates the matrix element $\langle \Gamma_i | \epsilon \cdot \hat{p} | \Gamma_j \rangle$.

direction

value  3D integer vector

default [1, 0, 0]

It defines the polarization direction $\epsilon$. From it a vector of unit length is calculated, which enters the calculation. In 1D simulation it can be omitted and [1,0,0] is then assumed.

Delta{ } Calculates the matrix element $\langle \Delta_i | \epsilon \cdot \hat{p} | \Delta_j \rangle$. See Gamma{ } for direction option.

X{ } Calculates the matrix element $\langle X_i | \epsilon \cdot \hat{p} | X_j \rangle$. See Gamma{ } for direction option.

L{ } Calculates the matrix element $\langle L_i | \epsilon \cdot \hat{p} | L_j \rangle$. See Gamma{ } for direction option.

HH{ } Calculates the matrix element $\langle HH_i | \epsilon \cdot \hat{p} | HH_j \rangle$. See Gamma{ } for direction option.

LH{ } Calculates the matrix element $\langle LH_i | \epsilon \cdot \hat{p} | LH_j \rangle$. See Gamma{ } for direction option.

SO{ } Calculates the matrix element $\langle SO_i | \epsilon \cdot \hat{p} | SO_j \rangle$. See Gamma{ } for direction option.

KP6{ } Calculates the matrix element $\sum_k \langle kp6_{i,k} | \epsilon \cdot \hat{p} | kp6_{j,k} \rangle$, $k = 1 \ldots 6$. See Gamma{ } for direction option.

KP8{ } Calculates the matrix element $\sum_k \langle kp8_{i,k} | \epsilon \cdot \hat{p} | kp8_{j,k} \rangle$, $k = 1 \ldots 8$. See Gamma{ } for direction option.

output_matrix_elements = yes/no  Output matrix elements.

output_transition_energies = yes/no  Output transition energies.

output_oscillator_strengths = yes/no  Output oscillator strengths. Currently, only a simple formula is used, i.e. the free electron mass is used and not the real effective mass one.

• dipole_moment_matrix_elements{ } Calculate dipole moment matrix elements $\langle i | \epsilon \cdot \hat{d} | j \rangle$ for wave functions within one band. The light polarization direction $\epsilon$ is automatically normalized in the program. $\hat{d} = e \hat{r}$ is the dipole moment vector.


Gamma{ } Calculates the matrix element $\langle \Gamma_i | \epsilon \cdot \hat{d} | \Gamma_j \rangle$.

direction
value 3D integer vector
default [1, 0, 0]

It defines the polarization direction $\epsilon$. From it a vector of unit length is calculated, which enters the calculation. In 1D simulation it can be omitted and [1,0,0] is then assumed.

**Delta{ }** Calculates the matrix element $\langle \Delta_i | \epsilon \cdot \hat{d} | \Delta_j \rangle$. See Gamma{ ... } for direction option.

**X{ }** Calculates the matrix element $\langle X_i | \epsilon \cdot \hat{d} | X_j \rangle$. See Gamma{ ... } for direction option.

**L{ }** Calculates the matrix element $\langle L_i | \epsilon \cdot \hat{d} | L_j \rangle$. See Gamma{ ... } for direction option.

**HH{ }** Calculates the matrix element $\langle HH_i | \epsilon \cdot \hat{d} | HH_j \rangle$. See Gamma{ ... } for direction option.

**LH{ }** Calculates the matrix element $\langle LH_i | \epsilon \cdot \hat{d} | LH_j \rangle$. See Gamma{ ... } for direction option.

**SO{ }** Calculates the matrix element $\langle SO_i | \epsilon \cdot \hat{d} | SO_j \rangle$. See Gamma{ ... } for direction option.

**KP6{ }** Calculates the matrix element $\sum_k \langle kp6_{k,i} | \epsilon \cdot \hat{d} | kp6_{k,j} \rangle$, $k = 1,\ldots,6$. See Gamma{ ... } for direction option.

**KP8{ }** Calculates the matrix element $\sum_k \langle kp8_{k,i} | \epsilon \cdot \hat{d} | kp8_{k,j} \rangle$, $k = 1,\ldots,8$. See Gamma{ ... } for direction option.

**output_matrix_elements = yes/no** Output matrix elements.

**output_transition_energies = yes/no** Output transition energies.

**output_oscillator_strengths = yes/no** Output oscillator strengths. Currently, only a simple formula is used, i.e. the free electron mass is used and not the real effective mass one.

- **transition_energies{ }** Calculate transition energies (energy difference) between two states in certain bands. Use this if you want to calculate transition energies but do not want to calculate the matrix elements. Note that the matrix elements defined above also include specifiers for transition energies: output_transition_energies = yes". 
  
  - Gamma{ }
  - KP6_Gamma{ }
  - HH_Gamma{ }
  - LH_Gamma{ }
  - SO_Gamma{ }
  - Delta{ }
  - HH_Delta{ }
  - LH_Delta{ }
  - SO_Delta{ }
  - X{ }
  - HH_X{ }
  - LH_X{ }
  - SO_X{ }
  - L{ }
- HH_L { }
- LH_L { }
- SO_L { }
- HH { }
- LH { }
- SO { }
- KP6 { }
- KP8 { }

- calculate_lifetime{ } Calculate the lifetimes of the state due to LO phonon scattering. For more information check R. Ferreira, G. Bastard, PRB 40, 1074 (1989) and Section 2.1.3 of the PhD thesis of G. Scarpa, Technische Universität München.

phonon_energy
  value any float > 0.0
  default 0.01

LO phonon energy

- bulkDispersion{ ... } Calculate bulk k.p dispersion of the material at a specific position in the simulation domain. For more details, see bulk_dispersion{}.

exchange_correlation{ }

Provides options to calculate exchange-correlation effects. This is not calculated by default.

type
  value string
  options lda : Include exchange-correlation effects in the LDA approximation (Local Density Approximation)
  lsda : Include exchange-correlation effects in the LSDA approximation (Local Spin Density Approximation)

initialSpinPol
  value float between 0.0 and 1.0
  default 0.0

Breaks spin up/down symmetry if no magnetic field is present.

outputSpinPolarization{} output spin polarization [dimensionless]

output_exchange_correlation{} output exchange correlation potentials in [eV].

quantum{ dispersion }

dispersion{}

calculate the \( k_{||} \) and \( k_{\text{superlattice}} \) (if applicable) dispersion. The energy dispersion \( E(k) \) along the specified paths and for the specified \( k \) space resolutions are completely independent from the \( k \) space resolution that was used within the self-consistent cycle where the k.p density has been calculated. The latter is specified in k_integration{ }.

path{ } Calculate dispersion along custom path in k-space. Multiple instances are allowed.

type
  value string
  options lda : Include exchange-correlation effects in the LDA approximation (Local Density Approximation)
  lsda : Include exchange-correlation effects in the LSDA approximation (Local Spin Density Approximation)

initialSpinPol
  value float between 0.0 and 1.0
  default 0.0

Breaks spin up/down symmetry if no magnetic field is present.

outputSpinPolarization{} output spin polarization [dimensionless]

output_exchange_correlation{} output exchange correlation potentials in [eV].
value string

name of the dispersions which also defines the names of the output files.

point{ }

specifies points in the path through k-space. At least two k points have to be defined.
Line between two such points is called segment.

k

value 3D float vector

k-point represented by vector \([k_x, k_y, k_z]\). The units are \(nm^{-1}\).

For 1D simulation the \(k||\) space is a \(k_y - k_z\) plane so \(k_y, k_z\) can be freely chosen.

\(k_x\) can only be different from zero, if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain.

for 2D simulation the \(k||\) space is a \(k_z\) axis so \(k_z\) can be freely chosen. \(k_x\) can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole y-domain.

for 3D simulation the \(k||\) space is empty. \(k_x\) can only be different from zero if a periodic boundary condition along the x-direction is defined and the quantum region extends over the whole x-domain. \(k_y\) can only be different from zero if a periodic boundary condition along the y-direction is defined and the quantum region extends over the whole y-domain. \(k_z\) can only be different from zero if a periodic boundary condition along the z-direction is defined and the quantum region extends over the whole z-domain.

spacing

value float

specifies approximate spacing for intermediate points in the path segments in \(nm^{-1}\).

Excludes num_points.

num_points

value integer \(> 1\)

specifies number of points (intermediate + two corner points) for each single path segment. Excludes spacing.

lines{ }

calculate dispersions along some predefined paths of high symmetry in k-space, e.g. \([100]\), \([110]\), \([111]\) and their equivalents (in total maximally 13).

name

value string

name of the dispersions which also defines the names of the output files.

spacing

value float

specifies approximate spacing for intermediate points in the path segments in \(nm^{-1}\).

k_max

value float

specifies a maximum absolute value (radius) for the k-vector in \(nm^{-1}\).

full{ }

calculate dispersion in 1D/2D/3D k-space depending on simulation dimensionality and periodic boundary conditions.

name

value string
name of the dispersion which also defines the name of the output file.

**kxgrid** () specifies a \texttt{grid} \ldots in \textit{k}-space for a 1D/2D/3D plot of the energy dispersion \( E(k_x, k_y, k_z) \). allowed only, if simulation is periodic along x-direction and current quantum region extends over the whole x-domain. The options are same as \texttt{grid}

**kygrid** () The options are same as \texttt{kxgrid} ()

**kzgrid** () The options are same as \texttt{kxgrid} ()

**superlattice** () (convenience keyword) Calculate superlattice dispersion \( E(k_{SL}) \) along periodic directions. The intervals are set automatically to \([-\pi/L_i, \pi/L_i]\], where \( L_i \) is the simulation domain range along periodic directions with \( i = x, y, z \).

**name**

value string

name of the dispersion which also defines the name of the output file.

**num_points_x**

value any integer > 1

specifies number of points along x direction in \textit{k} space where dispersion is calculated. The simulation must be periodic along the x direction in direct space.

**num_points_y**

value any integer > 1

specifies number of points along y direction in \textit{k} space where dispersion is calculated. The simulation must be periodic along the y direction in direct space.

**num_points_z**

value any integer > 1

specifies number of points along z direction in \textit{k} space where dispersion is calculated. The simulation must be periodic along the z direction in direct space.

**num_points**

value any integer > 1

(convenience keyword) Specifies number of points along all appropriate directions in \textit{k} space.

**output_dispersions** () output all defined dispersions.

**max_num**

value any integer between 1 and 9999

number of bands to print out

**output_masses** () output effective masses calculated from the dispersions.

**max_num**

value any integer between 1 and 9999

output effective masses calculated from the dispersions.
**bulk_dispersion**

Calculate bulk $k.p$ dispersion of the material at a specific position in the simulation domain.

- **path**{} calculate bulk $k.p$ dispersion along custom path in k-space. Multiple instances are allowed.
  - **name**
    - **value** string
      - name of the dispersion which also defines the name of the output file.
  - **position**{}
    - specifies the point $(x,y,z)$ in the simulation domain, where the dispersion has to be calculated.
      - **x**
        - **value** float
          - for 1D simulations (x)
      - **y**
        - **value** float
          - for 2D simulations (x,y)
      - **z**
        - **value** float
          - for 3D simulations (x,y,z)
  - **shift_holes_to_zero**
    - **value** yes or no
      - default no
      - If enabled, shifts the whole dispersion, so that the energy for the Gamma point for the highest hole band is equal to 0.0 [eV].
    - **point**{}
      - specifies points in the path through k-space. At least two k points have to be defined.
        - Line between two such points is called segment.
      - **k**
        - **value** 3D float vector
          - k-point represented by vector $[k_x, k_y, k_z]$. The units are $nm^{-1}$.
      - **spacing**
        - **value** float
          - specifies approximate spacing for intermediate points in the path segments in $nm^{-1}$.
          - Excludes **num_points**.
    - **num_points**
      - **value** integer > 1
        - specifies number of points (intermediate + two corner points) for each single path segment. Excludes **spacing**.
    - **lines**{}
      - calculate dispersions along some predefined paths of high symmetry in k-space, e.g. [100], [110], [111] and their equivalents (in total maximally 13).
      - **name**
        - **value** string
          - name of the dispersions which also defines the names of the output files.

---

**8.2. Input File**
**position** specifies the point (x,y,z) in the simulation domain, where the dispersion has to be calculated.

- **x**
  - value float
  - for 1D simulations (x)

- **y**
  - value float
  - for 2D simulations (x,y)

- **z**
  - value float
  - for 3D simulations (x,y,z)

**shift_holes_to_zero**
- value yes or no
- default no

If enabled shifts the whole dispersion, so that the energy for the Gamma point for the highest hole band is equal to 0.0 [eV].

**spacing**
- value float

specifies approximate spacing for intermediate points in the path segments in nm⁻¹.

**k_max**
- value float

specifies a maximum absolute value (radius) for the k-vector in nm⁻¹.

**full** calculate bulk k.p dispersion in 3D k-space. Multiple instances are allowed.

**name**
- value string

name of the dispersions which also defines the names of the output files.

**position** specifies the point (x,y,z) in the simulation domain, where the dispersion has to be calculated.

- **x**
  - value float
  - for 1D simulations (x)

- **y**
  - value float
  - for 2D simulations (x,y)

- **z**
  - value float
  - for 3D simulations (x,y,z)

**shift_holes_to_zero**
- value yes or no
- default no
If enabled shifts the whole dispersion, so that the energy for the Gamma point for the highest hole band is equal to 0.0 [eV].

\texttt{kxgrid{} } specifies a grid\{...\} in k-space for a 1D/2D/3D plot of the energy dispersion E(kx, ky, kz). allowed only, if simulation is periodic along x-direction and current quantum region extends over the whole x-domain. The options are same as \texttt{grid{}}.

\texttt{kygrid{} } The options are same as \texttt{kxgrid{}}.
\texttt{kzgrid{} } The options are same as \texttt{kxgrid{}}.
\texttt{output_bulk_dispersions{} } output all defined bulk k.p dispersions.
\texttt{output_masses{} } output effective masses calculated from the dispersions.

\texttt{contacts{}}

Definition of the contacts

At simulation boundaries, a Neumann boundary condition is applied to the Poisson equation, \( \frac{\partial \phi}{\partial x} = 0 \).

Five types of contacts are possible:

- **Schottky**
  A Schottky contact requires the specification of a Schottky barrier. A Schottky contact implies
  - Dirichlet boundary conditions for the electrostatic potential (Poisson equation)
  - Dirichlet boundary conditions for the Fermi levels

  The Dirichlet value for the potential within the contact is determined by requiring that the energetic distance between the Fermi level and the conduction band edge is equal to the value of the Schottky barrier. Note: A Schottky contact can also be used to model the effect of Fermi level pinning due to surface states.

- **Charge neutral**
  As for the Schottky contact a charge neutral contact implies
  - Dirichlet boundary conditions for the electrostatic potential (Poisson equation)
  - Dirichlet boundary conditions for the Fermi levels

  The Dirichlet value for the potential within the contact is determined by requiring local charge neutrality for each grid point of the contact. Note: For instance, this contact model can be used to calculate the energy levels in a quantum well or quantum cascade laser (QCL) as a function of applied bias.

- **Ohmic**
  From 2019-01-23 on, we changed the definition of ohmic. Ohmic contacts now behave per default like charge neutral contacts but additionally have a shift parameter.

- **Zero field** (was called ohmic before) – \textit{not recommended}
  A zero field contact implies
  - Neumann boundary conditions for the Poisson equation (i.e. zero electric field, or more precisely \( \mathbf{D} = 0 \) where \( \mathbf{D} \) is the dielectric displacement)
  - Dirichlet boundary conditions for the Fermi levels in the current equation.

  Note: Quantum regions extending into zero field contacts will cause carrier densities higher than those in metals and Fermi levels in the keV range. The cause of this is the nonphysical way zero field contacts are calculated. (Essentially, by enforcing a Neumann zero-field condition at the contact.) Note: Until 2019-01-23, zero_field was called ohmic.

- **Fermi**
  A Fermi contact implies Dirichlet boundary conditions for the Fermi levels. Fermi contacts are boundary conditions to the current equations only. Fermi contacts are “invisible” for the Poisson and Schrödinger
equation, i.e. the Poisson and Schrödinger equations are solved in the Fermi contact region with the material parameters that are defined in this region. No boundary conditions are imposed on the electrostatic potential, i.e. Neumann is used for the Poisson equation (default).

Please note the following important computational limitation: For almost all types of contacts involving the Poisson equation (ohmic, schottky with barrier defined, charge_neutral, zero_field), the material under the contact is used as reference for computing charge neutrality conditions, ohmic properties, or barrier heights, not the material adjacent to the contact. Thus, please make sure that an appropriate material is defined under the contact to be available as reference. The only exceptions to this behavior are fermi contacts, since these contacts are only involved the current equations, and schottky contacts with defined work_function, since then no reference material is needed.

The bias affects the value of the Fermi levels as it determines the Dirichlet value of the Fermi levels within the contacts. For example, by specifying a bias of 0.7 V, the Fermi level in the corresponding contact is set to a value of -0.7 eV.

```
contacts{

contact no. 1: Specify "Schottky barrier" to model Fermi level pinning due to surface states: For Schottky contacts, it is also possible to define the vacuum work function of the contact materials instead of the Schottky barrier. In this case, the Schottky-Mott rule will be used to determine the barrier height of the contact. However, please note that, due to Fermi level pinning, experimentally measured Schottky barrier heights may be quite different. The corresponding vacuum energy level () for all Schottky contacts is predefined as 6.3 [eV] in correspondence to the band offsets in the database, but can be modified as well if necessary.

``schottky``{

    name "air"
    bias 0.0 # [V]
    barrier 0.7 # [eV] Specify either barrier or work_function but not both.
    work_function 0.7 # [eV] (optional)
}

vacuum_level 6.3 # [eV] (optional, relevant for work_function only) (default = 6.3 eV, estimated from Si affinity of 4.05 eV and Si band offset in database)
```

Contact no. 2: Specify charge neutral contact.

```
charge_neutral{

    name "charge_neutral"
    bias 0.0 # [V]
}
```

Contact no. 3: Specify ohmic contacts.

```
ohmic{

    name "source" # name of contact
    bias 0.0 # [V]
    shift 0.0 # [eV] (optional, default = 0.0) Can be used to shift the band structure up (or down) in order to increase (or decrease) the ohmic behavior of the contact. # Warning: The effect of shift can be very large, best start with \pm 1\ kBT for p-doped/n-doped material, (that is \pm 0.025 eV at 300 K) and check the result.
}
```

Contact no. 4: Specify zero field contacts (was called ohmic before 2019): (We are not sure whether there is a physical scenario for zero_field contacts, but we leave them in the code in case somebody wants them. In any case, users should keep in mind that they may induce huge space charges which often result into convergence problems.)

---

Chapter 8. nextnano++
**contact no. 5**: Specify Fermi levels.

```plaintext
contact no. 5: Specify Fermi levels.

fermi{
    name  =  "fermi_1",
    bias  =  0.7  # [V]
}
```

It is possible to define Dirichlet regions for the electron or the hole quasi-Fermi level only (as compared to type fermi{} which always creates a Dirichlet region for both). This feature may be used to e.g. fix the quasi-Fermi level of majority carriers in a region while leaving the quasi-Fermi level of the minority carriers free to float there, as for instance in a bipolar device. Note that overlapping contact regions still cannot be defined, thus also no overlapping e.g. fermi_electron{} and fermi_hole{} contact regions with different bias. Each contact definition will replace (overwrite) previously defined contact region definitions at each respective grid point.

**contact no. 6**: Specify Fermi level for electrons.

```plaintext
contact no. 6: Specify Fermi level for electrons.

fermi_electron{
    name  =  "fermi_el",
    bias  =  0.7  # [V]
}
```

**contact no. 7**: Specify Fermi level for holes.

```plaintext
contact no. 7: Specify Fermi level for holes.

fermi_hole{
    name  =  "fermi_hl",
    bias  =  0.7  # [V]
}
```

Directory names

```plaintext
long_directory_names = no  # yes/no (default: no)
```

no: bias subdirectories are enumerated as `bias_*****` independently of the numbers of contacts defined.

yes: bias subdirectories are named `bias_000_001_***_...` which, for inputs with a large number of contacts, could result in issues with too long file paths.

See file `bias_points.log` in each subdirectory for the actual bias values used.

```plaintext
}
```

Each contact (ohmic, schottky, fermi, charge_neutral, zero_field) can contain an optional bias sweep.

Instead of specifying

```plaintext
bias  =  0.5  # [V]
```

one can alternatively define a voltage sweep (e.g. 0.0, 0.1, 0.2, ..., 2.0)

```plaintext
bias  =  [0.0, 2.0]  # start and end bias in [V]
steps  =  20  # (optional) number of voltage sweep steps, integer between 1 and 999
```

or one can alternatively define a sequence of bias points for each contact:
bias \([0.0, 1.8, 1.9, 2.0]\) # sequence of biases [V]
# (up to 100 bias points)
steps \(1\) # (optional) number of voltage
# sweep steps between
# consecutive bias points
# (default is 1)

If step is defined to be greater than 1, the space between the consecutive bias points is also subdivided correspondingly. Consecutive identical bias points are skipped. A reversal in the direction of the change in bias is allowed but currently probably not useful. If bias is defined as a scalar (or vector of length 1), only one bias value is calculated (and the value of steps is ignored).

The output file bias_points.log contains the mapping between bias values and bias index for all bias points. It is possible to define a bias sweep at several contacts.

At each grid point, only one type of contact can exist. For overlapping contact regions, the last defined contact on this grid point is used.

currents{}

Specifications for the current equation such as mobility models and recombination models.

currents{
  mobility_model
  high_field_saturation{
    alpha_electrons \(0.0\) # >= 0.0
    beta_electrons \(2.0\) # >= 0.001
    vsat_electrons \(1.0\) # >= 1.0
    alpha_holes \(0.0\) # >= 0.0
    beta_holes \(2.0\) # >= 0.001
    vsat_holes \(1.0\) # >= 1.0
  }
  recombination_model{ # required
    SRH \(\text{yes}\) # optional, default: no
    Auger \(\text{yes}\) # optional, default: no
    radiative \(\text{yes}\) # optional, default: no
    enable_generation \(\text{yes}\) # optional, default: yes
  }
  output_fermi_levels()
  output_fermi_level_differences()
  output_mobilities()
  output_recombination()
  output_currents()
  output_velocities()
  output_power_density()
  # linear equation solver in current equation
  linear_solver{
    iterations \(200\)
    abs_accuracy \(1e-30\)
    rel_accuracy \(1e-13\)
  }
}
dkr_value = 0.1
use_cscg = no
force_diagonal_preconditioner = no
force_iteration = no
accuracy = 0

devloglevel = 1

insulator_bandgap = 0.5 # [eV]
minimum_density = le11 # [cm^-3]
minimum_density_factor = [1.0, 1.0] # [cm^-3]
maximum_density = le30 # [cm^-3]
maximum_density_factor = [1.0, 1.0] # [cm^-3]
minimal_recombination = yes

**mobility_model**

    value "string"
    options constant, masetti, arora, minimos, simba
    default constant

The following mobility models are supported.

- constant mobility model (*mobility_constant{}*).
- Masetti mobility model (*mobility_masetti{}*).
- Arora mobility model (*mobility_arora{}*).
- MINIMOS 6 mobility model (*mobility_minimos{}*).
- SIMBA mobility model (*mobility_simba{}*).

**high_field_saturation{}**

optional
experimental feature (use with care!)

    alpha_electrons
        value double >= 0.0
        example 0.0

    beta_electrons
        value double >= 0.001
        example 2.0

    vsat_electrons
        value double >= 1.0
        example 1.0

    alpha_holes
Specify $\alpha$, $\beta$ and $v_{sat}$ values for electrons and holes, respectively.

Model for velocity saturation

The parameters derive from an extended Canali model. The driving force $F$ of the respective carriers is currently the gradient of the respective quasi-Fermi level.

Setting $\alpha$ to zero and $\beta$ to 2 will yield the Hänsch model.

Note that convergence may be much more difficult to achieve when high field saturation (hfs) is enabled. Therefore, this feature should remain disabled by default.

\[
\mu(F) = \frac{(\alpha + 1)\mu_{low}}{\alpha + \left(1 + (\alpha + 1)\frac{v_{sat}}{v_{sat}^h}\right)^\frac{\beta}{\alpha}}
\]

$\mu_{low}$: low field mobility

$v_{sat}$: saturation velocity

**recombination_model{}**

*required*

**SRH (optional)**

value yes or no

default no

Shockley-Read-Hall recombination (**recombination{}**)

**Auger (optional)**

value yes or no

default no

Auger recombination (**recombination{}**)

**radiative (optional)**

value yes or no

default no

radiative recombination (direct recombination) (**recombination{}**)

**enable_generation (optional)**

value yes or no

default yes

Generation can be switched on (default: yes) or off for the enabled processes.
For each enabled of the three processes, generation is enabled/disabled using enable\_generation. Thus, enabling only generation without also enabling recombination is not possible (enable\_generation = yes has no effect then). If radiative recombination is calculated (radiative = yes), then the photo\_current is included in the file IV\_characteristics.dat. Additionally, the internal quantum efficiency is written to the file internal\_quantum\_efficiency.dat.

**Output definitions**

**output\_fermi\_levels()**

prints out the Fermi levels for electrons and holes [eV]

**output\_fermi\_level\_differences()**

prints out the difference of electron and holes Fermi levels, $\Delta E_F = E_{Fn} - E_{Fp}$ [eV]

By overlaying the Fermi level difference over the band gaps, you may e.g. determine where and involving which bands lasing may occur.

**output\_mobilities()**

prints out the electron and hole mobilities [cm\(^2\)/Vs]

**output\_recombination()**

prints out the recombination rates [1e18/(cm\(^3\)/s)]

Note: If you want to output the generation rate, you have to specify this in structure\{ generation[] ==> output\_generation() \}.

**output\_currents()**

prints out the electron and hole current densities [A/cm\(^2\)]

Note: The electron, hole, and total currents (integrated over the contacts surfaces) are always written into the files IV\_electrons.dat, IV\_holes.dat, and IV\_characteristics.dat (in units of [A/cm\(^2\)] (1D), [A/cm] (2D), [A] (3D)), respectively. If radiative recombination is used, the file IV\_characteristics.dat also contains the photo current. In all IV\_*.dat files, the first columns indicate the voltages at each contact. Typically, the first column should be the one that is swept as it is then easier to plot the results within nextnano\_mat as the first column is the x axis in such a plot. You can switch the columns by reordering the contacts, see contacts[]. The consumed power is written in IV\_Power.dat in units of [W/cm\(^2\)] (1D), [W/cm] (2D), [W] (3D), respectively. The emitted power column is added if the energy resolved density integration is enabled.

**output\_velocities()**

prints out electron and hole drift velocities [cm/s]
output_power_density()

prints out power density (only Joule heating) \([\text{W/cm}^3]\)

linear_solver()

Parameters for linear equation solver in current equation.

  iterations
    value integer
    default 10000
    example 200
  number of iterations

  abs_accuracy
    value double
    default 1e-30 [eV]
    example 1e-30 [eV]
  absolute accuracy of Fermi level, use a small value to force convergence

  rel_accuracy
    value double, 0.0 <= rel_accuracy <= 0.01
    default 1e-13 [dimensionless]
  relative accuracy

  dkr_value
    value double, must be <= 0.5
    default -1.0 [dimensionless]
    example 0.1 [dimensionless]
  magic parameter to speed up calculations, affects preconditioning

  use_cscg
    value yes or no
    default no
  Forces the slower but occasionally more robust CSCG (Composite Step Conjugate Gradient) linear solver to be used rather than the cg (Conjugate Gradient) linear solver. May occasionally prevent a diagonalization failure.

  force_diagonal_preconditioner
    value yes or no
    default no
Only for debugging purposes, enabling will make code much slower or prevent convergence. Forces the use of a slower but more robust diagonal preconditioner. Only for debugging purposes, enabling will make code much slower or prevent convergence. Please try setting it to yes in case preconditioning fails or the linear solver diverges. If set to yes, iterations may have to be further increased.

**force_iteration**

value  yes  or  no

default  no

Only for debugging purposes, enabling will make code much slower or prevent convergence.

**extended_accuracy**

value  0  or  1

default  0

If set to 1, then current equation is solved using slower but more accurate solver (only implemented for nonperiodic 1D simulations). Experimental feature, will change in the future.

**debuglevel**

value  integer value between [-1,3]  
default  1

The higher this integer number, the more information on the numerical solver is printed to the screen output. Increasing the respective debuglevel to 2 or more significantly increases the volume of the diagnostic output displayed in nextnanomat (or a shell window). As result of the additional I/O load, particularly 1D simulations will slow down correspondingly (especially for currents[] and poisson[ ]).

**insulator_bandgap**

value  double  
default  1.0 # [eV]  
example  0.5 # [eV]

$I_{\text{gap}}$ affects initial solution of Fermi level.

A large value (relative to band gap) of $I_{\text{gap}}$ lets Fermi level drop continuously. A small value of $I_{\text{gap}}$ lets Fermi level drop in barrier and makes it flat in small bandgap regions. A better, more meaningful, name for insulator_bandgap might have been initial_energy_scale.

The drift-diffusion current equation reads $\text{div} (\mu n \nabla E_F) = G - R$. In order to calculate the density $n$, we have to know the quasi-Fermi level $E_F$. Approximately, the intrinsic density exponentially depends on the band gap $E_{\text{gap}}$. Therefore we apply a trick, and use $\text{div} \exp \left( \frac{E_{\text{gap}}}{I_{\text{gap}}} \right) \nabla E_F = 0$ in order to find a first approximation to the quasi-Fermi level, where $I_{\text{gap}}$ can be entered in the input file (insulator_bandgap) to adjust the convergence behavior of the initial solution.
**minimum_density**

value double

default 1e10 \(\text{[cm}^{-3}\)\]

eexample 1e11 \(\text{[cm}^{-3}\)

Improves condition number of current equation matrix.

Note: \(1e12 \text{[cm}^{-3}\) seems to be too high.

\(1e-10\) is actually the smallest possible value, smaller values are just automatically increased to \(1e-10\) without warning. Minimum charge carrier density (lower limit) for both electrons and holes that can appear in drift-diffusion current equations. The minimum density might have to be increased in order to obtain convergence for the drift-diffusion current equations. The minimum density should be as low as possible. The minimum density can be chosen as large as possible but should be smaller than the minimum density in the converged result. As the drift-diffusion current is proportional to the charge carrier density, this eventually also sets the lower limit of the current. The minimum density is a useful flag for structures where regions are present that have almost no density (e.g. a barrier, or insulator). If the density in such an insulator is below \(1 - 10^3 \text{[cm}^{-3}\), the product of \(\mu n\) in the drift-diffusion current equation varies over several orders of magnitude. Consequently, the matrix used in the linear solver is not well conditioned. Here, the current through these insulating regions is basically zero which has implications on the convergence behavior of the drift-diffusion current equations. Increasing the minimum density will help in these cases. A useful value for the minimum density of a certain material depends on the band gap because its intrinsic density also depends on the band gap. A wide-band gap material has a much lower intrinsic density than a low-band gap material.

Minority carriers in highly doped semiconductors or carriers in undoped wide band gap semiconductors have extremely small equilibrium densities (much less than \(1.0 \text{ cm}^{-3}\)), resulting in complete breakdown of the solvers for current equation due to underflow. Unfortunately, it is not clear whether quadruple precision would cure this problem. Also, it seems unphysical to believe that one carrier per kilometre can be physically relevant (aside from the rather practical issue that real-life minority carrier densities are not in thermal equilibrium and thus never become as small as predicted). Therefore, we modified the code (2019-01-23) to make sure that the minimum density parameter as specified for the current equation is not smaller than \(10^{-10} \text{ cm}^{-3}\) (this value corresponds to a conductivity 10 orders of magnitude lower than the best insulators). At the same time, we left the minimum value in the syntax definition unchanged as \(0.0\) in order to avoid users getting annoying error messages when they experimentally set the value to zero (zero will just be silently increased to \(1e-10\)). For systems without extremely low carrier densities, results will not be affected by this change.

Separate minimum densities for electrons and holes are optionally available:

**minimum_density_factor**

value double array

default \([1.0 , 1.0]\) \(\text{[cm}^{-3}\)

eexample \([1.0 , 0.01]\) \(\text{[cm}^{-3}\)

Here, the two numbers for \textit{minimum_density_factor} defines scaling factors by which the minimum density is multiplied for \textit{electrons} and for \textit{holes}. For example, a minimum density of \(1e10 \text{ [cm}^{-3}\) for \textit{electrons} and \(1e8 \text{ [cm}^{-3}\) for \textit{holes} could be defined e.g. as

```
minimum_density 1e10 # [cm^-3]
minimum_density_factor [1.0 , 0.01] # [cm^-3]
```

or

```
minimum_density 1e9 # [cm^-3]
minimum_density_factor [10 , 0.1] # [cm^-3]
```

or, most conveniently,
Irrespective of the definition, the actually used values for the minimum densities are output close to the beginning of the log file.

**maximum_density**

value double

default 1e30 # [cm^-3]

similar meaning as minimum_density

**maximum_density_factor**

value double array

default [ 1.0 , 1.0 ] # [cm^-3]

analogous to minimum_density

**Note:** Note that the minimum/maximum densities only affect the current operators (∇μn∇) and corresponding currents (en∇), thus, they have no direct influence on recombination (unless minimal_recombination = yes), computed densities, Poisson equation, etc. Also, when restricting effective densities in the current equations from above and/or below, please think first whether and in which way such restriction affects the physics: Increasing minimum densities make insulating regions less insulating, whereas decreasing maximum densities make conducting regions less conducting.

**minimal_recombination**

value yes or no

default no

If enabled, the minimum densities will also apply to the recombination/generation terms of the current equation.

**cbr{}**

Specifications that define CBR (Contact Block Reduction method) calculation, i.e. ballistic current calculations.

This method is based on the following publications: [BirnerCBR2009], [MamaluyCBR2003]

At a glance: CBR current calculation

- full 1D, 2D and 3D calculation of quantum mechanical ballistic transmission probabilities for open systems with scattering boundary conditions
- Contact Block Reduction method:
  - only incomplete set of quantum states needed (~ 100)
  - reduction of matrix sizes from $O(N^3)$ to $O(N^2)$
- ballistic current according to Landauer–Büttiker formalism
The CBR method is an efficient method that uses a limited set of eigenstates of the decoupled device and a few propagating lead modes to calculate the retarded Green’s function of the device coupled to external contacts. From this Green’s function, the density and the current is obtained in the ballistic limit using Landauer’s formula with fixed Fermi levels for the leads.

It is important to note that the efficiency of the calculation and also the convergence of the results are strongly dependent on the cutoff energies for the eigenstates and modes. Thus it is important to check during the calculation if the specified number of states and modes is sufficient for the applied voltages. To summarize, the code may do its job very efficiently but is far away from being a black box tool.

```cbr{
  name "qr" # CBR quantum region
  lead{ name "lead_1" } # lead quantum region
  rel_min_energy -0.01 # lower boundary (relative)
  rel_max_energy 0.3 # upper boundary (relative)
  abs_min_energy 2.5 # lower boundary (absolute)
  abs_max_energy 2.6 # upper boundary (absolute)
  delta_energy 1e-6 # energy grid resolution
  ildos yes # outputs integrated LDOS
  two_particle_options [1, 0, 0] # for one particle model
}
```

**name**

- **value** “string”
- **example** qr_device

refers to quantum region to which CBR method will be applied \((d\text{-dimensional})\)

**lead{}**

- **name**
  - **value** “string”
  - **example** qr_lead1

  Provides the name of the quantum region of the lead.

Refers to lead quantum region. Make sure that the lead region specified here has dimension \(d - 1\).

**rel_min_energy**

- **value** double
- **default** -0.01 # [eV]

Lower boundary for transmission energy interval relative to lowest eigenvalue
rel_max_energy

value double
default 1.01 # [eV]
Upper boundary for transmission energy interval relative to highest eigenvalue

abs_min_energy

value double
default 0.0 # [eV]
Lower boundary for transmission energy interval on an absolute energy scale

abs_max_energy

value double
default 0.0 # [eV]
Upper boundary for transmission energy interval on an absolute energy scale

Note:
Specify either
• rel_min_energy and rel_max_energy or
• abs_min_energy and abs_max_energy.

delta_energy

value double
default 1e-4 # [eV]
This value determines the resolution of the transmission curve $T(E)$.

ildos

value yes or no
default no
Outputs integrated local density of states.
two_particle_options

value double array

default [1, 0, 0, ...] # ??? what is default

11 values for two-particle model [., ..., .]

```cpp
numStates2_ = (int)two_particle_options_[0];
const double epsRel = two_particle_options_[1];
const DVector3 r1 = two_particle_options_[2]*uNanometer,
                    two_particle_options_[3]*uNanometer;
const DVector3 r2 = two_particle_options_[5]*uNanometer,
                    two_particle_options_[6]*uNanometer;
const double delta = two_particle_options_[8]*uEVolt; // splitting
cost double z = two_particle_options_[9]*uEVolt; // tunneling

// [prefactor] = Q^2/[cEps0], [cEps0] = Q/L*V => [prefactor] = Q L V = eV_...
const double prefactor = two_particle_options_[10]*sqr(cEcharge)/
                        (2*pi*epsRel*cEps0);
```

Example Figure 8.2.2.4 shows the calculated transmission from lead 1 to lead 3 as a function of energy $T_{13}(E)$. Full line: All eigenfunctions of the decoupled device are taken into account. Dashed line: Only the lowest 7% of the eigenfunctions are included. Here, Neumann boundary conditions are used for the propagation direction. The vertical line indicates the cutoff energy, i.e. the highest eigenvalue that is taken into account.

![Figure 8.2.2.4: The transmission calculated with the CBR method using all eigenstates and only 7% of the eigenstates. In the latter case, the transmission is still very accurate for the lower energies.](image)

Additional notes Special boundary conditions for CBR method

- Along propagation direction Neumann boundary conditions are applied to the Schrödinger equation.
- Perpendicular to the propagation direction Dirichlet boundary conditions are applied to the Schrödinger equation.
**Note:** Physically speaking, the lead quantum cluster must be a two-dimensional surface in a 3D simulation, a one-dimensional line in a 2D simulation and a zero-dimensional point in a 1D simulation.

**optics()**

Specifications for the optics keyword, e.g. for the calculation of the optical absorption.

There is a nice optics tutorial on the optical absorption feature of nextnano++.

```
 optics{
  debuglevel = 2
  region{
    name = "optical_active"
    # Input values
    # (sigma+) circularly polarized light around the x axis
    polarization{ name = "y+i+z"
      re = [0, 1, 0]
      im = [0, 0, 1]
    }
    refractive_index = 10.0 # [] (optional)
    normalization_volume = 100.0 # [nm^dim] (optional)
    # Setting for k|| and eigenstate summation
    make_spin_degenerate = no
    spin_align = yes
    interband = yes
    intraband = yes
    energy_threshold = 1e-6 # [eV]
    transition_threshold = 1e-6 # [eV]
    occupation_threshold = 0 # [eV]
    k_integration{
      relative_size = 0.1
      num_points = 4
      # Specify either num_subpoints or num_totalsubpoints.
      num_subpoints = 16
      num_totalsubpoints = 128
      symmetry = 1
    }
    # Treatment of occupation
    occupation_ignore = no
    occupation_interpolate_invfermi = yes
    occupation_const_fermilevel = no
    # Output settings
    output_energies = no
    output_occupations = no
  }
}(continues on next page)
```
output_transitions = no
output_spinor_components = no

output_spectra{
    output_components = no
    spectra_over_energy = yes
    spectra_over_wavelength = no
    spectra_over_frequency = no
    spectra_over_wavenumber = no
}

spontaneous_emission = yes

energy_min = 1.4 # [eV]
energy_max = 1.7 # [eV]
energy_resolution = 0.001 # [eV]
energy_broadening_lorentzian = 0.005 # [eV]
energy_broadening_gaussian = 0.005 # [eV]

dipole_approximation = no # preliminary

}

debuglevel

value integer value \geq -1 \text{ and } \leq 4.

default 0

The larger this value, the more verbose the diagnostic output will be.

region{}

name

value "string"

example optical_active

name of the quantum region for which optics is calculated

polarization{}

name "string"

re real part

im imaginary part

Define polarizations of incoming light for which optical absorption should be calculated.

Examples:

# linearly polarized light in x direction.
# name is used for the file names of the output.
# polarization( name "x" re [1,0,0] )

# linearly polarized light in y direction
# polarization( name "y" re [0,1,0] )

(continues on next page)
# linearly polarized light in z direction

```
polarization{
  name "z" re [0,0,1]
}
```

# TM mode.
# This naming might be useful when analyzing heterostructure grown in x direction.

```
polarization{
  name "TM" re [1,0,0]
}
```

# TE mode

```
polarization{
  name "TEy" re [0,1,0]
}
```

```
polarization{
  name "TEz" re [0,0,1]
}
```

# (sigma+) circularly polarized light around the x axis

```
polarization{
  name "y+iZ" re [0,1,0] im [0,0,1]
}
```

# (sigma-) circularly polarized light around the x axis

```
polarization{
  name "y-iz" re [0,1,0] im [0,0,-1]
}
```

# an example for an arbitrary polarization direction

```
polarization{
  name "x1y1z2" re [1,1,2]
}
```

We can generally write the electric field of a traveling wave propagating to $k$ direction as follows:

$$E(r; t) = (E_x \hat{x} + E_y \hat{y} + E_z \hat{z}) \exp [i(k \cdot r - \omega t)]$$

$$\begin{bmatrix}
  E_x \\
  E_y \\
  E_z
\end{bmatrix} \exp [i(k \cdot r - \omega t)] = \begin{bmatrix}
  \text{Re}(E_x) \\
  \text{Re}(E_y) \\
  \text{Re}(E_z)
\end{bmatrix} + i \begin{bmatrix}
  \text{Im}(E_x) \\
  \text{Im}(E_y) \\
  \text{Im}(E_z)
\end{bmatrix} \exp [i(k \cdot r - \omega t)]$$

where $E_{x/y/z}$ are complex numbers.

re=[ , , ] and im = [ , , ] correspond to the first and second column in the last line.

refractive_index

- **optional**
- **optional**
- **value** double
- **unit** dimensionless
- **example** 10.0

Specify an alternative refractive index for the substrate material.

normalization_volume

- **optional**
- **optional**
- **value** double
- **unit** [nm^dim]
- **example** 100.0

Specify an alternative normalization volume for the optical spectra. The default is the volume of the simulated device.

Settings for $k_{||}$ and eigenstate summation
**make_spin_degenerate**

- **value** yes or no
  - optional optional
  - default no

Effect of the flag is throwing away every second state from the respective quantum solver, i.e. the states are made spin degenerate and are counted twice. Use only when all states are pairwise identical.

**spin_align**

- **value** yes or no
  - default ?

*yes:* aligns spin-degenerate states in a virtual magnetic field prior to the optics calculations.

*no:* Spin-degenerate states enter the optics calculation in a random superposition.

**interband**

- **value** yes or no
  - default yes

Compute optical valence band to conduction band transitions.

**intraband**

- **value** yes or no
  - default yes

Compute optical valence band to valence band and conduction band to conduction band transitions.

**energy_threshold**

- **value** double
  - default 1e-6 [eV]
  - example 1e-8 [eV]

Only transitions between states with at least this energy difference are regarded when computing optical spectra. 1e-8 [eV] should be reasonable in most cases.

**transition_threshold**

- **value** double
  - default 1e-6 [eV]
  - example 1e-4 [eV]

Only transitions between states with at least this optical intensity are regarded when computing optical spectra. A value of 0 can be safely used as a default. Higher values can reduce computational time but may neglect weak optical transitions.

**occupation_threshold**

- **value** double; 0 <= 1
  - default 0 [eV]
  - example 1e-8 [eV]

Only transitions between states with at least this occupation are regarded when computing optical spectra. A value of 0 can be safely used as a default. Higher values can reduce computational time but may neglect weakly occupied states.

**k_integration{}**

- relative_size
value  double, 0.001 <= relative_size <= 1.0
  default  0.1
  example  0.3

size of $k_{||}$ space integration

num_points
  value  integer
  units  ?
  default  4
  example  8

number of points in $k_{||}$ space where transition intensities are computed

Note: Specify either num_subpoints or num_totalsubpoints.

num_subpoints
  default  ?
  example  16
  example  256

number of interpolation points for $k_{||}$ space integration (relative to num_points)

num_totalsubpoints
  default  128
  example  128

total number of interpolation points for $k_{||}$ space integration (relative to num_points)

symmetry
  value  integer; 1, 2, 3 or 4
  default  1
  example  4

rotational $k_{||}$ space symmetry

force_k0_subspace
  value  yes or no
  default  no

If set to yes, $k_{||}$ integration in quantum{} is modified in that only states for point $k = 0$
are computed exactly, whereas all other k points are computed in the subspace of the $k = 0$
wavefunctions. As a result of this approximation, computational speed is much improved (you
may even be able to also enlarge the number of eigenvalues). In case you are planning to use
this approximation for final results, please make sure to check whether the resulting loss of
accuracy in optical spectra is acceptable.

Treatment of occupation

occupation_ignore
  value  yes or no
  default  no

Ignore the occupation of states when computing optical spectra: Valence bands and conduction bands
are considered to be fully occupied and fully empty, respectively.
occupation_interpolate_invfermi

value yes or no
default yes

less important. This option may increase accuracy of numerical $k_{||}$ space integration.

occupation_const_fermilevel

value yes or no
default no

In semi-classical current calculations, the quasi-Fermi level may depend on position. Optical spectra on the other hand are computed in a completely quantum mechanical way. To resolve this inconsistency, the Fermi level is averaged. Note that both options violate the physical assumption in some way and are only a valid approximation for almost constant Fermi levels. no: In this case, the inconsistency is resolved by computing a position dependent occupation number.

Output settings

output_energies

value yes or no
default no

Output energy dispersion for every transition.

output_occupations

value yes or no
default no

Output occupation dispersion for every transition.

output.transitions

value yes or no
default no

Output transition strength for every transition.

output.spinor_components

value yes or no
default no

Output the spinor components for each state at each $k_{||}$ point (only relevant in multi-band $k \cdot p$ calculations).

Note: In 1 dimensional systems the axis of quantization for the angular momentum is x, in 3D z.

output.spectra

output.components

value yes or no
default no

Output Im $\epsilon$ for every transition.

spectra.over.energy

value yes or no
default yes
Output spectra with respect to the energy.

\texttt{spectra\_over\_wavelength}
\begin{itemize}
\item \texttt{value} yes or no
\item \texttt{default} no
\end{itemize}
Output spectra with respect to the wavelength.

\texttt{spectra\_over\_frequency}
\begin{itemize}
\item \texttt{value} yes or no
\item \texttt{default} no
\end{itemize}
Output spectra with respect to the frequency.

\texttt{spectra\_over\_wavenumber}
\begin{itemize}
\item \texttt{value} yes or no
\item \texttt{default} no
\end{itemize}
Output spectra with respect to the wavenumber.

\texttt{spontaneous\_emission}
\begin{itemize}
\item \texttt{value} yes or no
\item \texttt{default} no
\end{itemize}
Calculate spontaneous emission rate using the momentum matrix element obtained by 8-band kp model. (This feature is not yet implemented in 3D simulation.)

\texttt{energy\_min}
\begin{itemize}
\item \texttt{value} double
\item \texttt{default} ? # [eV]
\item \texttt{example} 1.4 # [eV]
\end{itemize}
lower energy bound for optical spectra

\texttt{energy\_max}
\begin{itemize}
\item \texttt{value} double
\item \texttt{default} ? # [eV]
\item \texttt{example} 1.7 # [eV]
\end{itemize}
upper energy bound for optical spectra

\texttt{energy\_resolution}
\begin{itemize}
\item \texttt{value} double
\item \texttt{default} 0.001 # [eV]
\end{itemize}
spectral resolution

\textbf{Energy broadening} (Lorentzian or Gaussian)

\texttt{energy\_broadening\_lorentzian}
\begin{itemize}
\item \texttt{value} real number \geq 0.000001
\item \texttt{unit} [eV]
\item \texttt{default} 0.0 (i.e. switched off if undefined)
Set the broadening to value greater than 0.0 to make the Lorentzian broadening

\[ \mathcal{L}(E - E_0) = \frac{1}{\pi} \frac{\Gamma/2}{(E - E_0)^2 + (\Gamma/2)^2} \]

included to the calculation of the optical spectrums. The specified value is read as the FWHM \( \Gamma \).

**energy_broadening_gaussian**

- **value**: real number \( \geq 0.000001 \)
- **unit**: [eV]
- **default**: 0.0 (i.e. switched off if undefined)

Set the broadening to value greater than 0.0 to make the Gaussian broadening

\[ \mathcal{L}(E - E_0) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(E - E_0)^2}{2\sigma^2} \right) \]

included to the calculation of the optical spectrums. The specified value is read as the FWHM \( \Gamma = 2\sqrt{\ln 2} \cdot \sigma \).

(In 1D and 2D, both Lorentzian and Gaussian can be used simultaneously. In 3D, either of these broadenings must be included.)

**dipole_approximation**

- **value**: yes or no
- **default**: no

*(preliminary)* Specify whether to calculate the matrix elements corresponding to optical transitions with dipole approximation. Note: \( <\psi_1|p \cdot \epsilon|\psi_2> \) elements are calculated by default, with dipole approximation: \( p' \cdot \epsilon <\psi_1|\psi_2> \) (without scaling: \( \epsilon \) is polarization direction, and \( p \) is momentum operator, \( p' \) is the integral for the Bloch central functions)

For more details, see [Eissfeller2008].

**opticaldevice** (not tested)

```plaintext
opticaldevice{
    name = "optical_active"
    line_broadening = 1
    line_width = 1.0 # [eV]
    photon_energy = 1.0 # [eV]
}
```

**name**

- **value**: “string”
- **example**: `optical_active`

name of quantum region for which optical generation should be calculated
**line_broadening**

- value integer
- example 1

broadening type: for Lorentzian it is 1

**line_width**

- value double
- example 1.0

for Lorentzian broadening the linewidth in [eV]

**photon_energy**

- value double
- example 1.0

the mid energy of the radiation in [eV]

**database{}**

specification of material parameters

The documentation for this keyword is available [here](#) (old documentation layout).

**output{}**

- **Visualization**
  - **Sections and Ranges**
- **VTK format for rectilinear grid**
- **AVS format for rectilinear grid**
- **Output of material parameters**

Options for the visualization of the output data and output of material parameters.

**Visualization**

Specification of options for the visualization of the data with certain programs like

- **Origin** (1D/2D)
- **VTK** VTK format (2D/3D)
- **AVS/Express** (2D/3D)

- **directory**
  - value "string"
  - example "./output/quantum_well_1D"
path for output files

**mandatory_path**

value yes or no
default no

If `mandatory_path` is set to yes, the (relative or absolute) output directory specified in `output{ directory = ... }` is used, and any directory specified in the command line (as e.g. done by `nextnanomat`) is ignored.

If it is set to no, the directory specified in the command line (as e.g. done by `nextnanomat`) is used as base path, to which a relative path specified in directory then is appended. On the other hand, an absolute path specified in directory is ignored.

In all cases, a subdirectory named as the input file is further appended to the output path, unless `-n` or `--noautooutdir` is set as command line option (`nextnanomat` sets this option automatically).

Also note that the location of the log (*.log) file is not affected by these settings.

### Warning:
Please make sure that a mandatory output directory is set such that no important files (or the input directory) are overwritten. Be especially careful when accepting input files from others, and do not run simulations using administrative privileges.

#### format2D

value string

**options**

- VTKAscii
- AvsBinary_one_file
- AvsAscii_one_file
- AvsBinary
- AvsAscii
- VTKAscii_AvsBinary_one_file
- VTKAscii_AvsAscii_one_file
- VTKAscii_AvsBinary
- VTKAscii_AvsAscii
- Origin

**default** AvsBinary_one_file

Output file format for 2D simulations

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTKAscii</td>
<td>VTK XML ASCII format (.vtr, r = rectilinear grid)</td>
</tr>
<tr>
<td>AvsBinary_one_file</td>
<td>AVS/Express file format - header (ASCII), coordinates and variables (both binary) are written into a single .fld file</td>
</tr>
<tr>
<td>AvsAscii_one_file</td>
<td>AVS/Express file format - header (ASCII), coordinates and variables (both ASCII) are written into a single .fld file</td>
</tr>
<tr>
<td>AvsBinary</td>
<td>AVS/Express file format (AVS steering files *.v, and *.fld, *.coord, *.dat data files) - data files in binary format</td>
</tr>
<tr>
<td>AvsAscii</td>
<td>AVS/Express file format (AVS steering files *.v, and *.fld, *.coord, *.dat data files) - data files in ASCII format</td>
</tr>
<tr>
<td>VTKAscii_AvsBinary_one_file</td>
<td>VTKXML + AvsBinary_one_file</td>
</tr>
<tr>
<td>VTKAscii_AvsAscii_one_file</td>
<td>VTKXML + AvsAscii_one_file</td>
</tr>
<tr>
<td>VTKAscii_AvsBinary</td>
<td>VTKXML + AvsBinary</td>
</tr>
<tr>
<td>VTKAscii_AvsAsciiOrigin</td>
<td>VTKXML + AvsAscii</td>
</tr>
<tr>
<td>Origin</td>
<td>Origin file format (Origin steering files *.plt, data files *.dat)</td>
</tr>
</tbody>
</table>

#### format3D

value string
**Options** see format2D

Output file format for 3D simulations. Options are identical to the 2D case.

**write_avs_v**

**options** yes or no

**default** no

Output of AVS steering file .v.

**write_origin_plt**

**options** yes or no

**default** no

Output of Origin steering file .plt.

**write_gnuplot_plt**

**options** yes or no

**default** no

Currently, gnuplot format is only implemented for energy resolved densities in 1D, energy resolved photogeneration in 1D, and light field (and may create huge (!) files).

**use_gnuplot_one_file**

**options** yes or no

**default** no

If yes, all information (metadata and data) necessary for the gnuplot figure is contained in one file.

**silent**

**value** yes or no

**default** yes

If set to no, print additional warnings concerning output.

---

**Sections and Ranges**

Ranges in sections may contain only one grid point. If no point is found inside the range interval, as e.g. for zero-length intervals such as [50.1, 50.1], the closest grid point is used.

All section commands are ignored for energy resolved densities, energy resolved photogeneration, and light field.

**section{}** Output a smaller section of the simulation area (zoom).

**Example**

```plaintext
section{
    name = "part" # name of section enters file name
    range_x = [0, 20] # range in x direction [nm]
    range_y = [-5, 5] # range in y direction [nm] (2D or 3D only)
    range_z = [2, 10] # range in z direction [nm] (3D only)
}
```

**section1D{}** Output a 1D section of the simulation area (1D slice) (2D or 3D only).

**Example**
section1D{
  name "x"  # name of section enters file name
  x 10.0  # 1D slice at x = 10 nm
  y 10.0  # 1D slice at y = 10 nm
  z 10.0  # 1D slice at z = 10 nm (3D only)
  range_x [0, 20]  # (optional) range in x direction [nm]
  range_y [-5, 5]  # (optional) range in y direction [nm]
  range_z [2, 10]  # (optional) range in z direction [nm] (3D only)
}

Note:

• 2D usage:
  - x, range_y
    1D slice at x = ... nm within the range from y = ... nm to y = ... nm or
  - y, range_x
    1D slice at y = ... nm within the range from x = ... nm to x = ... nm

• 3D usage:
  - x, y, range_z or
    1D slice at x = ... nm and y = ... nm within the range from z = ... nm to z = ...
    nm
  ...

If range is left out, the section extends over the whole simulation area.

section2D{}  Output a 2D section of the simulation area (2D slice) (3D only).

Example

section2D{
  name "center"  # name of section enters file name
  x 10.0  # 2D slice at x = 10 nm
  y 20.0  # 2D slice at y = 20 nm
  z 10.0  # 2D slice at z = 10 nm
  range_x [0, 20]  # (optional) range in x direction [nm]
  range_y [-5, 5]  # (optional) range in y direction [nm]
  range_z [2, 10]  # (optional) range in z direction [nm]
}

Note:

• 3D usage:
  - x, range_y, range_z
    2D slice at x = ... nm within the range from y = ... nm to y = ... nm and from z
    = ... nm to z = ... nm or
  - y, range_x, range_z
    2D slice at y = ... nm within the range from x = ... nm to x = ... nm and from z
    = ... nm to z = ... nm or
  - z, range_x, range_y
2D slice at $z = \ldots$ nm within the range from $x = \ldots$ nm to $x = \ldots$ nm and from $y = \ldots$ nm to $y = \ldots$ nm

**only_sections**

value yes or no  
**default** no

If set to no, print additional warnings concerning output.

Currently, for fields living on the position grid, the entire field is always output in addition to its sections defined by `output{}`. However, if `only_sections` is set to yes, the entire field will not be output anymore but only its sections defined by `output{}`. Thus, if no sections are defined, also no fields will be output. These files can be used to restrict field output to the actual regions of interest, or also to suppress most file I/O (if no sections are defined).

This setting has no effect on RAM usage or on the fields used in the calculation, it just affects what is written into files. Also note that quantities living on e.g. an energy grid, integrative quantities like I-V curves, or files needed for resuming operation are not influenced by this setting.

**Examples**

```makefile
output{  
    directory "./output"  
    section1D{  
        name "x"  
        y 10  
        z 10  
    }  
    section2D{  
        name "y"  
        y 10  
        range_x [-20, 220.5]  
        range_z [-20, 220.5]  
    }  
}
```

```makefile
output{  
    directory "./output"  
    section{  
        name "zoom"  
        range_x [0,20]  
        range_y [-5,5]  
    }  
    section1D{  
        name "y"  
        y 10  
        range_x [-20, 220.5]  
    }  
}
```
VTK format for rectilinear grid

===> VTK - The Visualization Toolkit

The .vtr format can be read by the following software:

- VisIt visualization tool (free)
- ParaView (open source)
- ImageVis3D (open source)

AVS format for rectilinear grid

The .fld format can be read by the following software:

- AVS/Express visualization tool (commercial)

The main file of AVS format has .fld extension. Here is an example:

```plaintext
# AVS/Express field file  # necessary header
#
ndim  3  # number of dimensions

dim1  6  # number of nodes along 1st dimension
dim2  6  # number of nodes along 2nd dimension
dim3  6  # number of nodes along 3rd dimension

# of dimensions "ndim"
space  3  # must be equal to "ndim"
veclen  1  # number of components of vector field, "1" = scalar field

# are supported.
data  # data type. Currently only "double" and "integer"

field  # type of mapping. Only rectilinear field is supported.
label  # label for each vector field component
unit  # unit of each vector field component (internally in tool not used at the moment)

1 file  filetype  skip  offset  stride  #
<---defines where 1st component of vector field is saved. Numbering must be ascending, starting with "1"
<---and number of "variable" "i" lines must be equal to "veclen". Supported file types are "ascii" and "binary".
<---"skip" defines how many lines in file have to be skipped before data item appears.
<---"offset" defines how many columns in line have to be skipped before searched data items appear.
<---"stride" defines how many steps have to be made before next data item appears.

1 file  filetype  skip  offset  stride  #
<---contains information about where and how nodes of 1st coordinate are stored
1 file  filetype  skip  offset  stride  #
<---contains information about where and how nodes of 2nd coordinate are stored
1 file  filetype  skip  offset  stride  #
<---contains information about where and how nodes of 3rd coordinate are stored

<---numbering must be ascending, starting with "1" and number of "coord" "i" lines
<---must be equal to "ndim"
```

(continues on next page)
The following shows an example of a file that can be imported using `import{}`. This example shows how to import ordered data via AVS format 3D_origin-format.fld file into `nextnano++`:

```
# AVS/Express field file
#
ndim = 3
dim1 = 3
dim2 = 3
nspace = 3
veclen = 2
data =
dim2 = [0, 0, 0, 1, -1, 5, 0, 0, 2, -2, 0, 5, 0, 4, -4, 5, 5, 0, 5, -5, 10, 5, 0, 6, -6, 0, 10, 0, 7, -7, 5, 10, 0, 8, -8, 10, 10, 0, 9, -9, 0, 0, 5, 10, -10, 5, 0, 5, 11, -11, 10, 0, 5, 12, -12, 0, 5, 5, 13, -13, 5, 5, 5, 14, -14, 10, 5, 5, 15, -15, 0, 10, 5, 16, -16, 5, 10, 5, 17, -17, 10, 10, 5, 18, -18, 0, 0, 5, 10, -19, 5, 0, 10, 20, -20, 10, 0, 10, 21, -21, 0, 5, 10, 22, -22, 5, 5, 10, 23, -23, 10, 5, 10, 24, -24, 0, 10, 10, 25, -25, 5, 10, 10, 26, -26, 10, 10, 10, 27, -27,
```

Note that the order of the values matters.
Output of material parameters

material_parameters{}

kp_parameter{}  Output k · p parameters of materials in quantum regions where 6-band or 8-band k · p Hamiltonian was solved. The Dresselhaus-Kip-Kittel (DKK) parameters (L, M, N), which are used internally in the code, are printed out. Also the Luttinger (\(\gamma_1, \gamma_2, \gamma_3, \kappa\)) (for zinc blende) or Rashba-Sheka-Pikus (\(A_1, A_2, \ldots, A_6\)) parameters (for wurtzite) are printed out, respectively. Additionally for 8-band k · p calculations the S, E_P, F and B parameters are printed out. For further information, consult Chapter 3 of the PhD thesis of S. Birner.

boxes

  value  yes or no

  (optional) For each grid point, in 1D two points are printed out to mimic abrupt discontinuities at material interfaces (in 2D four points, in 3D eight points)

charge_carrier_masses{}  Output effective masses of all energy bands used in the simulations in units of \([m_0]\).

boxes  as above

spin_orbit_coupling_energies{}  Output spin-orbit coupling energy for zinc blende (1 parameter) or crystal-field splitting and spin-orbit coupling energies for wurtzite (3 parameters) in units of \([eV]\).

boxes  as above

static_dielectric_constants{}  Output static relative dielectric constants for zinc blende (1 parameter) and wurtzite (3 parameters) (unitless).

boxes  as above

deformation_potentials{}  Output the deformation potentials for zinc blende and wurtzite in units of \([eV]\).

boxes  as above

run{}

Specifications for the program execution (program flow)

Note:  There are two syntaxes for run{}.

New syntax  recommended syntax for versions newer than 2020-05-01. We specify new syntax in this documentation.

Deprecated syntax  used in versions older than 2020-04-30. Nevertheless, it will still work in later versions. Deprecated syntax is specified here.

- structure_only{}  
- strain{}  
- poisson{}  
- quantum{}  
- current_poisson{}  
- quantum_poisson{}  

structure_only()  

If present, calculation is aborted after structure setup, similarly to when the command line flag \(-s\) or \(--structure\) is set. But differently from the command line flag, if last_region is present, partial structure initialization is performed. This is useful for debugging your structure definition, e.g. if you have a 2D or 3D simulation with many material regions, contact regions, doping regions and generation regions overlapping each other in a complicated way. The files in the output directory Structure/ will then reflect this partial initialization. (Note that in case not all regions are used here, some initialization and output steps related to strain, poisson, current, quantum, cbr, optics, etc. will be omitted in order to avoid inconsistencies.)

\begin{verbatim}
last_region{ <integer> }

value  any integer \(\geq 1\)

default 1000000
\end{verbatim}

Example:

\begin{verbatim}
run{
    structure_only{
        last_region 5
    }
}
\end{verbatim}

The simulation prints out the structure up to the (last) region index 5.

strain()  

It solves the strain equation

poisson()  

It solves the Poisson equation

quantum()  

It solves the Schrödinger equation. Exchange–correlation effects (optional) can be included and are calculated from the quantum density. Then the Schrödinger equation is solved again but this time including the exchange-correlation potential energy.
current_poisson()

It solves the coupled current and Poisson equations self-consistently.

fast_poisson

value yes or no

default no

If enabled, Newton iterations for Poisson in the middle of the classical current-Poisson iteration will be limited to 1. Note that enabling this setting may also influence stability of convergence or change the optimal value for alpha_fermi. Typically, yes increases the number of iterations but significantly reduces the overall execution time.

multi_stage_solve

value yes or no

default no

Flag in order to solve classical current equation first with recombination/generation switched off in order to get a good starting point, and then with recombination/generation switched on (if any recombination models are switched on). Can be used to improve convergence in some situations but may increase runtime in others.

system_solve

value yes or no

default no

Alternative new iteration method for classical current-Poisson. This Newton method may provide better convergence for some systems (but may require different values of convergence parameters). yes results in Fermi levels and potential being simultaneously updated as a system of unknowns during the iteration. Irrespective of its value, system_solve always takes the value of current_repetitions into account.

iterations

value any integer >= 1

default 100

Number of iterations for current-Poisson solver

fermi_limit

value any float between 0.0 and 10.0

default 2.0

Defines how far the quasi-Fermi levels can move above the highest / below the lowest contact. Except in case of huge bandgaps and extreme photogeneration, the defaults should not require any change. At the same time, in the absence of any externally induced photogeneration, these values could be set to zero in order to stabilize the iteration.

current_repetitions

value any integer >= 1

default 1

Number of current-density iterations. The current equations are repeatedly solved for the quasi-Fermi levels with the densities fixed. The current equation for the electrons and for the holes are solved independently with a common and fixed recombination term. For each iteration, the densities are adjusted according to the new quasi-Fermi levels of the previous iteration. current_repetitions defines number of these repetitions. If generation/recombination is present, using a value > 1 (e.g. 5) may stabilize the iteration and sometimes enable faster convergence (larger alpha_fermi may also be possible then).
limit_repetitions

    value  yes or no
    default no

    If enabled, the current-density loop is exited early as soon as residual_fermi is reached by the quasi-Fermi levels.

residual

    value  any float > 0.0
    default 1e5 cm$^2$ (1D)
    1e3 cm$^{-1}$ (2D)
    1e-3 [dimensionless] (3D)

    Residual occupation changes.

residual_fermi

    value  any float > 0.0
    default 1e-5 [eV]

    Residual Fermi level changes. This value is also used during quantum_current_poisson().

alpha_fermi

    value  any float between 1e-5 and 1.0
    default 1.0

    Dimensionless underrelaxation parameter for Fermi level. The final quasi-Fermi level for electrons after each iteration is calculated as follows:

    \[ E_{F,n} = (E_{F,n} \text{ of previous iteration}) \times (1 - \alpha_{\text{fermi}}) \times (E_{F,n} \text{ of actual iteration}) \times \alpha_{\text{fermi}} \]

    This Fermi level is then input to the next iteration. The same holds for the Fermi level \( E_{F,p} \) for holes. The value of \( \alpha_{\text{fermi}} \) will change due to \( \alpha_{\text{scale}} \) during the iterations. The actually used \( \alpha_{\text{fermi}} \) is now included in iteration_current_poisson.dat and iteration_quantum_current_poisson_details.dat.

alpha_iterations

    value  any integer \( \geq 1 \)
    default 1000

    Number of alpha iterations

alpha_scale

    value  any float between 0.1 and 1.0
    default 0.998

    Alpha scale. Both for classical and for quantum iterations, \( \alpha_{\text{fermi}} \) will be reduced further as:

    \[ \alpha_{\text{fermi}} \leftarrow \max(\alpha_{\text{fermi}} \times \alpha_{\text{scale}}, 1e-5) \]

    at each iteration step once the number of iterations exceeds alpha_iterations.

Use this feature to improve convergence (particularly convergence of Fermi levels) towards the end of the iteration. Note that decreasing \( \alpha_{\text{fermi}} \) too fast (a problem with older
versions) will result in the iteration stalling (only the residuals of the densities but none of the Fermi levels decrease). The total current equation may then not be properly conserved.

**output_log**

- **value**: yes or no
- **default**: yes

**NOTE:** Both conditions specified by `residual` and `residual_fermi` must hold in order to consider a calculation as converged.

**quantum_poisson{ }**

It solves the Schrödinger-Poisson equations self-consistently. When `quantum_poisson{}` is desired, note that additionally either `poisson{}` or `current_poisson{}` is required.

**iterations**

- **value**: integer
- **default**: 30

  number of iterations, i.e. self-consistency cycles

**residual**

- **value**: any float > 0.0
- **default**: 1e5 cm^{-2} (1D), 1e3 cm^{-1} (2D), 1e-3 [dimensionless] (3D)

  residual of the integrated total charge carrier density changes. Note that this is dimension dependent and default is: 1e5/cm^{2} (1D), 1e3/cm (2D), 1e-3 [dimensionless] (3D). This applies to exact Schrödinger equation, not to subspace Schrödinger equation.

**Note:** If you do not include enough eigenstates, the convergence behavior might be affected as the occupation of the eigenstates is not considered in a useful way.

**use_subspace**

- **value**: yes or no
- **default**: yes

  solve Schrödinger equation within subspace of eigenvectors of previous iteration as long as achieved residual is larger than desired `residual * residual_factor` and at least in every second iteration

**subspace_iterations**

- **value**: any integer between 1 and 1000

**subspace_residual_factor**

- **value**: any float >= 2.0
- **default**: 1e12

  controls the number of subspace iterations

  It holds for `use_subspace = yes`:

  ```
  if (residual in densities > residual * subspace_residual_factor) {
    use only approximate quantum solutions
  }
  ```
Use subspace_iterations > 1 to further reduce computational load (i.e. runtime) from exact quantum solutions (the best value is system-dependent). Note that the number of iterations may not change or even increase. In rare cases (e.g. when a huge number of eigenvalues is computed), selecting use_subspace = no may be faster.

**alpha_potential**

- **value** any float between 0.001 and 1
- **default** 1.0

In case of stubborn convergence problems which do not appear to have any root cause such as not enough eigenvalues and which appear not to respond to any change in other parameters, try using a mildly smaller value than 1.0 such as 0.5.

Using values smaller than 1.0 per default is not recommended, as the run time is expected to increase as $1/\alpha_{\text{potential}}$ for normally converging input files.

**output_log**

- **value** yes or no
- **default** yes

Output of convergence of Schrödinger-Poisson equation (residuals for quantum_poisson) into the logfile iteration_quantum_poisson.dat

**quantum_current_poisson**

It solves the Schrödinger-Current-Poisson equations self-consistently. When quantum_current_poisson is desired, note that additionally either poisson or current_poisson is required and current_poisson must be defined in the input file.

- **iterations**
  - **value** integer
  - **default** 30
  
  see quantum_poisson

- **residual**
  - **default** 1e5 (1D) / 1e3 (2D) / 1e-3 (3D)
  
  see quantum_poisson

- **use_subspace**
  - **value** yes or no
  - **default** yes
  
  see quantum_poisson

- **subspace_iterations**
  - **value** any integer between 1 and 1000
  
  see quantum_poisson

- **subspace_residual_factor**
  - **value** any float $\geq$ 2.0
default 1e12

see quantum_poisson()

fermi_limit

value any float between 0.0 and 10.0

default 2.0

see quantum_poisson()

current_repetitions

default 2

number of current-density iterations. The current equation is repeatedly solved for the quasi-Fermi levels. For each iteration, the densities are adjusted according to the new quasi-Fermi levels of the previous iteration. current_repetitions defines number of these repetitions. If generation/recombination is present, using a value > 1 (e.g. 5) may stabilize the iteration and sometimes enable faster convergence (larger alpha_fermi may also be possible then).

limit_repetitions

value yes or no

default yes

If enabled, the current-density loop is exited early as soon as residual_fermi is reached by the quasi-Fermi levels.

residual_fermi

value any float > 0.0

default 1e-5 [eV]

alpha_fermi

value any float between 1e-5 and 1.0

default 1.0

The Fermi level is underrelaxed between repetitions using an underrelaxation parameter for the Fermi levels. It should be used only once an oscillation of residuals is observed while self-consistently solving the Poisson and Schrödinger (and current) equations to improve convergence. For further information, please read comments on alpha_fermi parameter above.

alpha_iterations

value any integer >= 1

default 1000

number of alpha iterations

alpha_scale

value any float between 0.1 and 1.0

default 0.998

Both for classical and for quantum iterations, alpha_fermi will be reduced further as $alpha_fermi \leftarrow \max( alpha_fermi \times alpha_scale , 1e-5)$ at each iteration step once the number of iterations exceeds alpha_iterations. Use this feature to improve convergence (particularly convergence of Fermi levels) towards the end of the iteration. Note that decreasing alpha_fermi too fast (a problem with older versions) will result in the iteration stalling (only the residuals of the densities but none of the Fermi levels decrease). The total current equation may then not be properly conserved.

alpha_potential
value any float between 0.001 and 1

default 1.0

In case of stubborn convergence problems which do not appear to have any root cause such as
not enough eigenvalues and which appear not to respond to any change in other parameters, try
using a mildly smaller value than 1.0 such as 0.5.

Using values smaller than 1.0 per default is not recommended, as the run time is expected to
increase as 1/\(\alpha_{\text{potential}}\) for normally converging input files.

output_log

value yes or no

default yes

Output of convergence of (quantum) current-Poisson equation (residuals for
quantum_current_poisson) into the logfile iteration_quantum_current_poisson.dat

Note: Both conditions specified by residual and residual_fermi are only checked between iterations
but not between repetitions.

optics{}

calculate optical properties, see optics{}

Examples

```
run{
    structure_only{} # If present, calculation is aborted
    →after structure setup.
}
```

```
run{} # just sets up the device geometry
```

```
run{
    strain{} # solves the strain equation
}
```

```
run{
    strain{} # solves the strain equation
    quantum{} # and then the Schrödinger equation
}
```

```
run{
    strain{} # solves the strain equation
    poisson{} # solves the Schrödinger and Poisson
    quantum_poisson{} # solves the Schrödinger and Poisson
    →equations self-consistently
}
```

```
run{
    strain{} # solves the strain equation
    current_poisson{} # solves the coupled current and Poisson
    →equations self-consistently
}
```

(continues on next page)
quantum_current_poisson{}  # solves the Schrödinger, Poisson and _
--current equations self-consistently
}  

run{
  quantum{}  # solves the Schrödinger equation
  optics{}  # calculates optical properties
}

Using the new syntax (quantum_poisson{}, quantum_current_poisson{}), the classical computations (poisson{} or current_poisson{}) can be specified independent from the quantum calculation to be performed, e.g. it is now possible to combine poisson{} with quantum_current_poisson{} to bypass the classical current calculations.

Restrictions

• **Poisson**: Only maximally one of poisson{} and current_poisson{} can be defined, which defines the classical equation to be solved (also as first stage before possibly solving any quantum mechanics). If neither is set, only fixed potentials will be used.

• **Quantum**: If quantum mechanics is desired, one of quantum{}, quantum_density{}, quantum_poisson{}, and quantum_current_poisson{} must be set.

• The quantum equations to be solved - only quantum, quantum with self-consistent density/exchange, self-consistent quantum-Poisson, and self-consistent quantum-current-Poisson - are only defined by the choice of quantum{}, quantum_density{}, quantum_poisson{}, and quantum_current_poisson{}, irrespective of the choice of the classical solution method. Note that one of poisson{} and current_poisson{} must be set when quantum_poisson{} or quantum_current_poisson{} is desired. Use poisson{} in conjunction with quantum_current_poisson{} to skip classical current calculations.

• Quantum with self-consistent density/exchange is solved by selection of quantum_density{} (users can change parameters in there as needed).

Further remarks

• 2019-01-24: At the end of current_poisson{}, Poisson is now solved once to make the band structure consistent with the Fermi levels. In case of incomplete convergence, the partly converged output is then more in line with physical intuition.

• Input residuals and tolerances are rescaled to various internal units (often in a dimension-dependent manner, i.e. they are different for 1D, 2D and 3D simulations) before being passed to low-level numerical routines like ARPACK, LAPACK, BLAS, nonlinear solvers, etc. Therefore, diagnostic output from low-level numerical solvers usually contains values which are completely different from those which are output by the high-level physics routines or output into files.

• **There are logfiles that track the convergence behavior of the iterations during the simulation.** The convergence information for the respective self-consistent equations can be plotted. It is best to use a logarithmic scale.

  – *iteration_current_poisson.dat*

  current_poisson{}

  Convergence of coupled Current-Poisson with classical densities

  – *iteration_quantum_density.dat*

  quantum_density{}

  Convergence of Schrödinger equation with self-consistent density/exchange
8.3 Simulation output

Here, we will add soon more information on the content of the output file names.

For each simulation run, a new output folder is created in the simulation output folder. The created folder has the name of the input file. In addition date-time is added to the folder name if the option is selected in Options->Expert settings of nextnanomat (this option is recommended in order to avoid overwritten existing output data).

The created output folder contains:

- the **input file** (.in).
- a folder ‘...’ which gives material parameters used in the calculation.
- a folder ... (only if the strain option is activated).
- Several files related to the sweep made. For a voltage sweep, it contains ....
- a **log file** is created at the end of the simulation, containing all the information displayed during the simulation.

8.4 Material Database

8.4.1 Zincblende

https://www.nextnano.com/nextnanoplus/software_documentation/database/zincblende.htm

8.4.2 Wurtzite

https://www.nextnano.com/nextnanoplus/software_documentation/database/wurtzite.htm

8.4.3 Recombination

recombination{}

This section specifies the coefficients related to recombination processes. These are used when the current equation is solved. In nextnano++, the following recombination processes are included:

- **Shockley-Read-Hall (SRH) recombination**
- **Auger recombination**
- **Radiative recombination**
Example

```c
binary_zb {
    name = Si # material name, e.g. Si, GaAs, InP, ...

    recombination{
        SRH{ # Shockley-Read-Hall recombination
            tau_n = 1.0e-9 # [s] zero doping scattering time for electrons
            nref_n = 1.0e19 # [cm^-3] reference doping concentration for electrons
            tau_p = 1.0e-9 # [s] zero doping scattering time for holes
            nref_p = 1.0e18 # [cm^-3] reference doping concentration for holes
        }
        Auger{ # Auger recombination
            c_n = 2.8e-31 # [cm^6/s]
            c_p = 9.9e-31 # [cm^6/s]
        }
        radiative{ # direct recombination
            c = 2.0e-10 # [cm^3/s]
        }
    }
}
```

**Shockley-Read-Hall (SRH) recombination**

SRH model models the generation/recombination process that is assisted by impurities. The recombination/generation rates depend on the deviation of the carrier concentration from the equilibrium value and the scattering rates depend on the doping concentration.

\[
R_{SRH} = \frac{p \cdot n - n_i^2}{\tau_p(n + n_i) + \tau_n(p + p_i)}
\]

\[
\tau_{p/n} = \frac{\tau_{p/n,0}}{1 + \frac{N_A + N_D}{N_{n/p,ref}}}
\]

<table>
<thead>
<tr>
<th>τ₀</th>
<th>τ₀ (zero doping scattering time for electrons in [s])</th>
</tr>
</thead>
<tbody>
<tr>
<td>n, ref</td>
<td>nref_n (reference doping concentration for electrons in [cm^-3])</td>
</tr>
<tr>
<td>p₀</td>
<td>τ₀ (zero doping scattering time for holes in [s])</td>
</tr>
<tr>
<td>p, ref</td>
<td>nref_p (reference doping concentration for holes in [cm^-3])</td>
</tr>
</tbody>
</table>
Auger recombination

\[
R_{\text{Auger}} = (C_n n + C_p p) \cdot (n p - n_i^2)
\]

More information on physics: Auger recombination processes in semiconductor heterostructures.

For devices with an extremely high carrier concentration the Auger process is the dominant recombination channel. The process involves three particles and therefore scales with the third power of the carrier densities.

The phonon-assisted Auger recombination rate, which plays an important role especially at high carrier injection, respectively high doping levels, will be modeled in the program by the following equation:

\[
R_{\text{Auger}} = (C_n n + C_p p) \cdot (n p - n_i^2)
\]

Radiative recombination

\[
R_{\text{radiative}} = C(n p - n_i^2)
\]

The simplest process for the generation and recombination of electron-hole pairs is the direct process via the emission or absorption of a photon (radiative recombination). This is important for light emitting devices.

8.4.4 Mobility models

**mobility_constant**

The documentation for this keyword is available [here](#) (old documentation layout).

**mobility_minimos**

The documentation for this keyword is available [here](#) (old documentation layout).
8.4.5 database_nnp_optional.in

This database file contains optional optical data such as

- standard solar spectra
- large collection of CIE illuminants and light sources
- reflectivity spectra
- absorption spectra.

If you wish to use this data, just insert the data of interest to your database file or into a `database/` section of your input file.

**Optical emission spectra**

in units of W/(m^2 nm)

(as function of wavelength in [nm])

**Solar spectra**

The following solar spectra are already predefined and do not need to be included into database or input files.

```plaintext
# extraterrestrial solar spectrum ASTM E-490 (1366.1 W/m^2 integrated)
# added cutoff at 119.5 nm and 1000000 nm to keep integrated irradiance finite
# name = "Solar-ASTME490"

# ASTM G-173-03 solar spectrum - extra terrestrial reference (airmass 0.0)
# added cutoff at 280 nm and 4000 nm to keep integrated irradiance finite
# name = "Solar-ASTM-G173-ETR"

# ASTM G-173-03 solar spectrum - air mass 1.5 global tilt (1000.4 W/m^2 integrated)
# added cutoff at 280 nm and 4000 nm to keep integrated irradiance finite
# name = "Solar-ASTM-G173-global"

# ASTM G-173-03 solar spectrum - air mass 1.5 direct + circumsolar (900.1 W/m^2 integrated)
```

(continues on next page)
CIE luminants and light sources

The following CIE luminants and light sources are already predefined and do not need to be included into database or input files.

- CIE illuminant A (tungsten - 2856 K) with additional cutoff at 300 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-A"

- CIE illuminant D50 (horizon daylight - 5003 K) with additional cutoff at 300 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-D50"

- CIE illuminant D55 (mind-morning/mid-afternoon daylight - 5503 K) with additional cutoff at 300 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-D55"

- CIE illuminant D65 (noon daylight - 6504 K) with additional cutoff at 300 nm and 830 nm (irradiance NOT normalized)
  
  # name = "CIE-D65"

- CIE illuminant D75 (North sky daylight - 7504 K) with additional cutoff at 300 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-D75"

- CIE fluorescent FL1 (normal, daylight - 6430 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-FL1"

- CIE fluorescent FL2 (normal, cool white - 4230 K - most representative) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-FL2"

- CIE fluorescent FL3 (normal, white - 3450 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
  
  # name = "CIE-FL3"
# CIE fluorescent FL4 (normal, warm white - 2940 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL4"

# CIE fluorescent FL5 (normal, daylight - 6350 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL5"

# CIE fluorescent FL6 (normal, light white - 5150 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL6"

# CIE fluorescent FL7 (broad band, D65 simulator - 6500 K - most representative) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL7"

# CIE fluorescent FL8 (broad band, D50 simulator, Sylvania F40 Design 50 - 5000 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL8"

# CIE fluorescent FL9 (broad band, cool white - 5150 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL9"

# CIE fluorescent FL10 (three narrow bands, Philips TL85, Ultralume 50 - 5000 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL10"

# CIE fluorescent FL11 (three narrow bands, Philips TL84, Ultralume 40 - 4000 K - most representative) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL11"

# CIE fluorescent FL12 (three narrow bands, Philips TL83, Ultralume 30 - 3000 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL12"

# CIE fluorescent FL3.1 (standard halophosphate - 2932 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.1"
# CIE fluorescent FL3.2 (standard halophosphate - 3965 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.2"

# CIE fluorescent FL3.3 (standard halophosphate - 6280 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.3"

# CIE fluorescent FL3.4 (DeLuxe - 2904 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.4"

# CIE fluorescent FL3.5 (DeLuxe - 4086 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.5"

# CIE fluorescent FL3.6 (DeLuxe - 4894 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.6"

# CIE fluorescent FL3.7 (three bands - 2979 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.7"

# CIE fluorescent FL3.8 (three bands - 4006 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.8"

# CIE fluorescent FL3.9 (three bands - 4853 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.9"

# CIE fluorescent FL3.10 (three bands - 5000 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.10"

# CIE fluorescent FL3.11 (three bands - 5854 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.11"

# CIE fluorescent FL3.12 (multi-band - 2984 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.12"

# CIE fluorescent FL3.12 (multi-band - 3896 K) with additional cutoff at 380 nm→ and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.12"

# CIE fluorescent FL3.13 (multi-band - 5045 K) with additional cutoff at 380 nm→ and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.13"

# CIE fluorescent FL3.14 (D65 simulator JIS Z 8716:1991 - 6509 K) with additional→ cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-FL3.14"

# CIE illuminant LED-B1 (phosphor-converted blue - 2733 K) with additional cutoff→ at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-B1"

# CIE illuminant LED-B2 (phosphor-converted blue - 2998 K) with additional cutoff→ at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-B2"

# CIE illuminant LED-B3 (phosphor-converted blue - 4103 K) with additional cutoff→ at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-B3"

# CIE illuminant LED-B4 (phosphor-converted blue - 5109 K) with additional cutoff→ at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-B4"

# CIE illuminant LED-B5 (phosphor-converted blue - 6598 K) with additional cutoff→ at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-B5"

# CIE illuminant LED-BH1 (red and phosphor-converted blue mixed - 2851 K) with→ additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-BH1"

# CIE illuminant LED-RGB1 (red, green, and blue mixed - 2840 K) with additional→ cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-RGB1"

# CIE illuminant LED-V1 (phosphor-converted violet - 2724 K) with additional
cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-V1"

# CIE illuminant LED-V2 (phosphor-converted violet - 4070 K) with additional
cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-LED-V2"

# CIE recommended indoor illuminant ID50 (5096 K) with additional cutoff at 300 nm
and 780 nm (irradiance NOT normalized)
# name = "CIE-ID50"

# CIE recommended indoor illuminant ID65 (6596 K) with additional cutoff at 300 nm
and 780 nm (irradiance NOT normalized)
# name = "CIE-ID65"

# CIE high pressure discharge lamp HP1 (sodium - 1959 K) with additional cutoff at
380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-HP1"

# CIE high pressure discharge lamp HP2 (color-enhanced sodium - 2506 K) with
additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-HP2"

# CIE high pressure discharge lamp HP3 (metal halide - 3144 K) with additional
cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-HP3"

# CIE high pressure discharge lamp HP4 (metal halide - 4002 K) with additional
cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-HP4"

# CIE high pressure discharge lamp HP5 (metal halide - 4039 K) with additional
cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "CIE-HP5"
Light sources

The following light sources are predefined (from: R. W. G. Hunt et al., Measuring Color, Wiley 2011), and do not need to be included into database or input files.

```plaintext
# low pressure sodium lamp (MB - 1726 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "Lamp-SOX"

# high pressure mercury lamp (MB - 5592 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "Lamp-MB"

# high pressure mercury lamp (MBF - 3538 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "Lamp-MBF"

# high pressure mercury lamp (MBTF - 3652 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "Lamp-MBTF"

# high pressure mercury lamp (HMI - 5988 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "Lamp-HMI"

# Xenon lamp (6044 K) with additional cutoff at 380 nm and 780 nm (irradiance NOT normalized)
# name = "Lamp-Xenon"
```

Optical reflectivity

(as function of wavelength in [nm])

```plaintext
optical_reflectivity {
  name  "Si-polished-wafer"
  cutoff no
  at{ wavelength  250  reflectivity  0.672612594 }
  at{ wavelength  260  reflectivity  0.705174   }
  at{ wavelength 1000 reflectivity  0.316252445 }
}

optical_reflectivity {
  name  "Al0.8Ga0.2As"
  ...
}
```
Optical absorption coefficient

in units of (1/cm)
(as function of wavelength in [nm])

```
optical_absorption{
    name "Si";
    cutoff no

    at{ wavelength 250 absorption 1.84E+06 }
    at{ wavelength 260 absorption 1.97E+06 }
    ...
    at{ wavelength 1450 absorption 3.20E-08 }
}

name "Silicon";
}

optical_absorption{
    name "Ge";

}

name "Germanium";
}

optical_absorption{
    name "GaAs";

}

optical_absorption{
    name "InP";

}

optical_absorption{
    name "GaN";

}

optical_absorption{
    name "InN";

}

optical_absorption{
    name "In0.20Ga0.80N";

}
```
8.5 Command line arguments

Command line usage:

`nextnano++_Intel_64bit.exe [runmode] [options] filename1 [filename2 ...]`

`filename1` is the input file you want to simulate.

Available optional runmodes are:

- `-v, --version` Show version number only.
- `-h, --help` Show command line usage only.
- `-p, --parse` Parse input file(s) and quit.
- `-s, --structure` Parse input file(s), generate structure(s), and quit.
- `-r, --resume` Resume previous execution saved using `--autosave`.

Available options are:

- `-l license_file, --license license_file` Use license file `<license_file>`.
  
  Example: `--license "E:\My Documents\nextnano\License\License_nnp.lic"`

- `-log, --logfile` Redirect output of input file(s) into separate log file(s).
  
  Generates a `*.log` file of screen output (standard output). The file will be written to: `<inputfilename>_log.log`

- `-i input_directory, --inputdirectory input_directory` Specify input directory `<input_directory>`.

- `-o output_directory, --outputdirectory output_directory` Specify output directory `<output_directory>`.

- `-n, --noautooutdir` Do not create output directory(ies) with same name(s) as input file(s).
  
  (= no automatic output directory)

Multi-threading

- `-t i, --threads i` Set number of parallel threads. Here, `i` threads are specified, any integer value between 0 and 1023 is allowed.

  *Not displayed and effective in serial executables. Currently we do not provide serial executables any more.*

  Using `--threads 0` is equivalent to not specifying `--threads` at all, i.e. the code does not attempt to change the number of threads used.

  Maximum value for `--threads` is the number of CPU cores, or possibly twice that number if *Hyper-threading* is enabled.

  For default value of 0, OpenMP system supplied maximal value is used.

  If set (e.g. using `nextnanomat` Expert Settings), the number of parallel OpenMP threads is set to the supplied value. If the desired value is too large for the CPU, the maximum value available for the CPU is set. If not set or set to 0, the default value as specified by the environment is used (usually 1 or all available). The actually used value is output near the beginning of the log file.

  For example, on an i7-8700 CPU (6 cores and 12 threads with *Hyper-threading* on), the optimal number for best performance is 4. Using the extra
threads from Hyper-threading rather hurts performance, and issues like memory speed seem to require a further reduction to less than 6 threads. With 4 threads, CPU load is about 45-50% on the tested CPU. This feature may also be useful for HTCondor to reduce background load, or to limit individual load for multiple parallel nextnano processes

-b i, --blas_threads i  
Set number of parallel threads in BLAS, LAPACK, etc. Here, i threads are specified, any integer value between 0 and 1023 is allowed.

Allows to separately set the number of BLAS (MKL) threads (MKL = Intel Math Kernel Library).

Maximum value for --blas_threads is typically the number of CPU cores.

Default value is 0 (Then uses the same number as the global number of threads which can be set by -t or --threads.)

For default value of 0, and if --threads is not specified or 0, the MKL library supplied maximal value is used.

Note: Additional notes on multi-threading

When only running one job at a time, setting --threads and --blas_threads to the number of CPU cores typically gives best performance. To force serial execution of each job, set both --threads and --blas_threads to 1.

Note that (the number of threads times the number of parallel jobs) and also (the number of BLAS threads times the number of parallel jobs) should not exceed the number of cores in order to avoid performance penalties from oversubscribing the CPU. Limited memory bandwidth may even impose lower limits on notebooks and lower grade desktop PCs.

Values for --threads and --blas_threads larger than the system supplied maximal values are automatically adjusted downwards. If unexpected values are automatically set (see logfile for output), please also check your environment variables such as OMP_NUM_THREADS or MKL_NUM_THREADS.

---

-g, --generate  
Generate additional debug information. Also outputs syntax definition files input_syntax.txt and database_syntax.txt. Additionally the files keywords_nnp.xml and database_nnp.xml are created, which are used by nextnatomat for its auto completion feature.

-l, --single  
Run multiple input files as single task.

-a, --autosave  
Autosave current execution for use with --resume.

Note that --resume controls reading from and --autosave controls writing into backup files. Both can be set independently or together.

Example:  
nextnano --license License_nnp.lic -log --outputdirectory "H:\nextnano\Output" QuantumDot.in

Soft kill  
If the user places or creates a file called SOFT_KILL (without file extension) into the root output folder of the currently running simulation, a softkill will be performed, i.e. the program exits the iteration cycle and writes the output.

The concrete effects are the following:

1. As soon as the SOFT_KILL file is detected (may take a while), any running classical or quantum iteration will be terminated early, but all (incomplete) results will be written into files. Note that the detection is only performed at the beginning of each iteration step.

2. If the SOFT_KILL file is detected in the classical current-poisson equation, no quantum or optical calculations will be performed afterwards, i.e. only classical (incomplete) results will be written into files.
3. After any detection, subsequent sweeps will still be executed but their data will be incomplete in the same way. (We also could prevent further sweeps if this is the preferred approach.)

4. The SOFT_KILL file is not being removed at the end of the simulation. However, old SOFT_KILL files are automatically removed at the beginning of the simulation and thus will not cause any trouble.

5. If there are multiple simulations running in parallel (or being scheduled sequentially), separate SOFT_KILL files need to be placed in the respective root output folders.

Further remarks

Priorities in descending order

1. Full (absolute) paths with file names have the highest priority, e.g. H:\nextnano\...

2. Input and output directories (both relative and absolute), defined in command line, have priority over absolute directory paths (not file paths) defined in input file.

Rules

Default input directory is the directory, where the input file is located (not the current working directory). It can be redefined in command line (--inputdirectory) or in the input file (import). By default the output of the simulation is written into an automatically generated directory with the same name as the input file. This default behavior can be suppressed using the command line flag --noautooutdir. If no output directory is defined in the command line or input file, the output of the simulation is written into the current working directory (including the automatically generated directory unless it is not suppressed). Relative input and output directory paths defined in the command line are relative to the current working directory. Relative paths to directories, defined in the command line and in the input file are always concatenated. Command line definitions have priority over definitions in the input file. If in the command line a relative or absolute path (--inputdirectory / --outputdirectory) is defined, the corresponding absolute directory path in the input file is ignored.

Examples

* --inputdirectory in command line is not defined

```plaintext
import{} # if no directory is specified, # the directory where the input file is located # is taken as the input directory directory  "D:\import_files" # absolute path # root path "import_files" # relative path with respect # to current working directory 

file{}

# specified directory is ignored.
filename  "D:\any_filename.fld" # absolute path. The above specified directory is ignored.
# root path. The above specified directory is ignored.
"any_directory\any_filename.fld" # relative path
# concatenated with path specified by directory.
"any_filename.fld" # file is searched in directory }
```

* --inputdirectory in command line is defined, e.g.

--inputdirectory D:\inputdir # absolute path
--inputdirectory \inputdir # root path
--inputdirectory inputdir # path relative to current directory

```plaintext
import{} # if no directory is specified, # the directory specified in the command line # is taken as the input directory directory  "D:\inputdir" # absolute path is ignored because of definition in command line (continues on next page)
```
The whole output of a simulation is written out in a directory named as the input file. This can be suppressed by command line flag \texttt{--noautooutdir}.

\begin{itemize}
\item \texttt{--outputdirectory} in command line is \emph{not defined}
\end{itemize}

\begin{verbatim}
output{
  # if no directory specified, # the current directory is taken as output directory
directory  "D:\simulation_output"
  # absolute path
  # root path is ignored because of --definition in command line
  # path specified in command line
  "any_directory" # relative path concatenated with path specified by command line and/or path specified by directory.
  "any_filename" # file is searched in directory defined by the command line and directory
}
\end{verbatim}

\begin{itemize}
\item \texttt{--outputdirectory} in command line is \emph{defined}, e.g.
\end{itemize}

\begin{verbatim}
output{
  # if no directory specified, # the directory specified in command line is taken as output directory
  directory  "D:\\outputdir" # absolute path
  "\\outputdir" # root path
  "outputdir" # relative (to current directory) path
}
\end{verbatim}

\textbf{Resume} One can resume calculation in \texttt{nextnano++}, if it was abnormally stopped, providing the simulator by potential, (quasi-)Fermi level for electrons and holes profiles, which are periodically saved in output directory while simulation. For this an additional argument \texttt{-r} or \texttt{--resume} has to be passed to the executable. This option can be used, if one wants to specify potential, (quasi-)Fermi level for electrons and holes profiles as input for simulation. All of three files:

\begin{itemize}
\item potential.raw
\item FermiLevel_el.raw
\item FermiLevel_hl.raw
\end{itemize}
have to exist in output directory of simulation. By passing to the simulation all three *.raw files no Poisson and current equations are solved. Only densities, band edges and quantum mechanics, if switched on, are calculated.

If the equations are solved self-consistently, then the profiles in *.raw are considered only as start values for all equations.

If exchange correlations are calculated, the exchange potentials are saved in xcpot_*.raw, which can be read in in resumed simulation.

8.6 Convergence

8.6.1 Introduction

Simulations of Schrödinger-Poisson converge self-consistently, and almost automatically, thanks to an algorithm proposed some years ago by Alex Trellakis, one of our talented developers. This algorithm was implemented in nextnano++ and has been very successful for several devices.

However, when the current equation is included in this system, the convergence to the solution becomes a challenge, due to the nature of this equation. For some devices, the system of equations becomes very unstable and a certain ability to reach the convergence is required.

Especially for systems where the carrier density fluctuates from large values to almost zero in certain regions or interfaces, the process of obtaining convergence becomes more critical and acting in a strategic way is very helpful.

8.6.2 Setting the input file for performing self-consistent current-Schrödinger-Poisson computations

Self-consistent current-Schrödinger-Poisson computations can be specified in the section run{} of the input file, through the statements

- current_poisson{}
- quantum_current_poisson{}

The first statement is mandatory, and it provides a first estimate of the electrostatic potential and the (quasi-)Fermi levels, even before including the quantum calculations to the system.

In principle, this is the minimum information required to start the simulations. All numerical parameters are adjusted automatically internally in the code until the solution is found or the maximum number of iterations is reached. Unfortunately, given the huge variety of devices the program can simulate, universal parameters are not possible to be predicted in advance. For this reason, in order to give to the user more control of the convergence process, some parameters can optionally be specified within the subsection quantum_current_poisson{}. Some examples are the following parameters: alpha_Fermi, residual, residual_fermi and iterations.

It is not our purpose to describe each of these parameters in this document, but to provide some guidance how to control the numerical process with the minimum effort as possible. The list of all parameters, its description, range of values and default can be found on the section run.
8.6.3 Talking about convergence

Before proceeding it is important to discuss what the expression “to get convergence” means. Actually, nextnano++ has to solve six groups of equations: current (also called, continuity) equation, Schrödinger equation and Poisson equation for electrons and for holes. As default the values of carrier densities, Fermi levels and potential are kept iteratively consistent from one step to the other. Internally the program computes for each equation a so-called cost function, that represents a metric of how close the obtained solutions are close to the “exact” one. For example, one way the cost function can be defined is by the difference of left and the right of each equation. Then, after each iteration the results of the cost function are called residuals.

Getting convergence means to find the conditions that minimize the cost functions. A good analogy of this process is the task of finding the deepest location of a valley in a mountain chain. In order to reach this valley, having some strategy concerning the necessary moves in some direction can reduce the time and the number of steps to conclude this task. If each step is too large, we can overfly the valley, if it is too small, we can take a long time to reach it. This is the role of the alpha_Fermi parameter in the current_poisson and quantum_current_poisson solvers: large values of alpha_Fermi can make the minimum invisible, and if it is too small can take a long time for simulations. Additionally, especially when the value of alpha_Fermi is small, it is possible that the number of iterations, given by the parameter iterations, is not enough to reach this minimum.

This analogy with a mountain chain is actually very simplistic, because the program deals with finding a minimum of cost functions in a multi-dimensional space and a non-linear system of equations, which makes this task more complex and, for this reason, provides more accurate results than any analytical model.

Keeping this in mind, setting the right parameters is usually an iterative process. One procedure that can be used for reducing the simulation time is by displaying the results “on-the-fly” within our graphical interface (nextnanomat) in two different ways.

The first method is through the numerical values displayed in the “Simulation” tab of the graphical interface (nextnanomat). The evolution of the residuals is printed out as soon they are computed. If, after certain time, some of the residuals are not reducing from one iteration to the next one, it is recommended to stop the simulation and restart a new one with different parameters.

The second method is by plotting the files iteration_current_poisson.dat and iteration_quantum_current_poisson.dat. By default, these files are generated automatically by the program, unless “output_log = no” is specified in current_poisson{} and quantum_current_poisson{} subsections. They can be displayed in the browser menu for the “Output” tab of nextnanomat. As in the previous method, if the residuals are reducing too slow, it is recommended to restart a new simulation that can accelerate the process.

8.6.4 Recommended strategy

As mentioned before, for some devices, the value of the parameters appearing in current_poisson{} and quantum_current_poisson{} subsections that bring the algorithm into a quick convergence belong to a very small region of the parameter-space, and tuning these parameters can require certain ability and time. The program contains internally several default parameters that are suitable for many devices, but due to the huge variety of configurations that a device can present, it is possible that, for some devices, they shall be adjusted manually.

The next following recommended steps can assist on the control of the simulation: they are not universal, but they can provide some ideas about the procedure.

1. Simplify the system

   Start finding a suitable electrostatic potential. In another words, comment out in the run{} section of the input file all lines except strain{} and current_poisson{} subsections.

2. Set minimum_density or maximum_density

   Set the minimum_density to a large value, i.e. 1e12 or even larger, if necessary. This parameter can be found within the current{} section of the input file. Nevertheless, for some conditions where the density of carriers is expected to be low, the values for minimum_density and maximum_density shall be reduced, for example to 1e-2 and 1e16, for example. In this situation, the most critical value is
the maximum_density. One typical example where the maximum_density shall be reduced is when simulating where the current in expected be almost zero, like in a diode or transistor operating under the threshold bias.

3. Adjust parameters of current_poisson simulation

A complete control of the simulation can be obtained by choosing new target residuals (residual and residual_fermi) and the number of iterations (iterations). The smaller the residuals, the larger the runtime will be. Choose a certain number of iterations, and after the simulation verify if it is necessary to increase this number, by reading the log file.

In the latest versions of nextnano++, a new method was developed that can reduce the simulation time. In order to activate the new method, please, set fast_poisson = yes inside current_poisson{}.

After each simulation it is recommended to reduce gradually the value of the minimum_density, for example, by a factor of ten, until the system again does not converge. At this point, change the value of alpha_fermi and current_iterations until the code converge again. In the next section an intuitive approach of how these parameters can be smartly changed will be presented.

For fast simulations, choose the value of alpha_Fermi as large as possible (the maximum value is 1.0). If this value generates overflow, a message will appear in the graphical interface and the simulation will stop. In this case, it is recommended to reduce the value of alpha_Fermi, for example, to 0.5, 0.1, 0.05, and so on. It rarely converges for values close to 1 (the default value).

If it is still not converging, increase the minimum_density, and simulate again using alpha_Fermi equal to 0.5 or less. There is no recipe valid for all devices.

At the end of this process, for certain values of the residuals, the value of the minimum_density shall be as small as possible. Taking some time to find the larger value of alpha_Fermi that brings the system to convergence will speed up the rest of the simulations.

4. Self-consistent quantum calculations

Follow the same procedure as before, but do not change the parameters of the current_poisson{} subsection. Usually, it is a good strategy to start with larger residuals within the quantum_current_poisson{} subsection than the one used in current_poisson{}.

Having obtained some initial results, even before reaching convergence, it is always helpful to check if the occupation number of all bands decays to zero, or at least, several orders of magnitude from the initial values. If necessary, increase the number of events in the specific band where the occupation number is not small enough. Keep in mind that the self-consistent solution shall contain all information about the states that can be populated for the system under a certain condition (for example, under a certain applied bias).

5. Alternative solution

Under certain conditions, a self-consistent simulation of these three groups of equations, can result for some systems in a numerically unstable solution. For this reason, the option of limiting how far the quasi-fermi levels can move above the highest contact of below the lowest one has been implemented. This is still a new feature under development and it is only recommended in the case of devices presenting materials with huge bandgaps and extremely photogenerated. For this kind of simulations, set fermi_limit to a value in the range 0 and 10 eV. The default is 2 eV.

8.6.5 Getting some intuition…

As mentioned above, depending on the nature of the device and the specific operation conditions (temperature or bias), it is necessary to guide the tool to get convergence. Let us see some practical examples.

Here we will illustrate how the evolution of the residuals in a current-Schrödinger-Poisson can evolve during the convergence process for two different devices. The images correspond to the plot of the data from interaction_current_poisson.dat, and iteration_quantum_current_poisson.dat files, that can be found in the output folder of the simulation.
Figure 8.6.5.1 corresponds to the residual evolution of a system that converges faster: all residuals drop around one order of magnitude every ten iterations. The default parameters within the code brings the system almost automatically to the minimum of the residuals.

In contrast, Figure 8.6.5.2 shows the final result for a different device after the system gets convergence. In this case, in the input file were specified that \texttt{residual	extunderscore fermi} is equal to $10^{-7}$ eV and residual (density) as $10^5$/cm$^3$. The value of \texttt{alpha	extunderscore Fermi} is 0.01. Although it was specified a total of 2000 iterations, the convergence was achieved in around 400 steps. It is important to notice that only after 200 iterations the system starts reducing the residuals in several orders of magnitude.

For some devices, setting the values of \texttt{alpha	extunderscore iterations} and \texttt{alpha	extunderscore scale} can result in a better performance. The value of \texttt{alpha	extunderscore iterations} is related to the moment where the \texttt{alpha	extunderscore Fermi} shall start to gradually reduce, and the value \texttt{alpha	extunderscore scale} is the rate of reduction between two successive iterations. There is no rule for the direction they should be changed. It is necessary to test some cases and look at the effect on the residuals.

![Figure 8.6.5.1: Residual evolution for a system A exhibiting quick convergence.](image)

Sometimes the number of iterations is not enough to reach the convergence. Figure 8.6.5.3 and Figure 8.6.5.2 plot the results of the same system B but differ in their number of iterations. Figure 8.6.5.3 is simulated with only 150 iterations. As it was shown in Figure 8.6.5.2, only after 180 iterations the residuals start to decrease. Hence Figure 8.6.5.3 does not show converging behaviour. In this kind of simulations, there are no criteria for knowing at which point this will happen: it requires experience or can be done by trial and error.

A pseudo-non-convergence can also happen when small residuals are specified in the input file. Returning to the Figure 8.6.5.2 it can be observed that, choosing \texttt{residual	extunderscore fermi} as $10^{-10}$ eV would probably result in a non-convergence: the \texttt{residual	extunderscore fermi} does not decrease at a high rate after 350 iterations. Then, increasing
Figure 8.6.5.2: Residual evolution for a system B with slow convergence. In the input file were specified \( \text{residual}_\text{fermi} = 10^{-7} \text{ eV} \), \( \text{residual (density)} = 10^5 \text{ /cm}^3 \), and \( \text{iterations} = 2000 \). The \( \text{alpha}_\text{Fermi} \) parameter was set to 0.01.
Figure 8.6.5.3: Residual evolution of system B with 150 iterations, exhibiting a pseudo-non-convergence behavior. Specifications in the input file: residual_fermi = 10^{-7} \text{ eV}, residual (density) of 10^5 \text{ /cm}^3, and iterations = 150. The value of alpha_{Fermi} is 0.01.
the number of iterations in this case would not solve the problem.

Another situation is when the value of $\alpha_{\text{Fermi}}$ is too small: it looks like the residuals do not decrease, like in Figure 8.6.5.4. In this example, $\alpha_{\text{Fermi}}$ was reduced from 0.01 (value used for Figure 8.6.5.2 and Figure 8.6.5.3) to 0.0001, and after 2000 iterations the system does not converge. Here we used the system B of the previous two images.

![Figure 8.6.5.4: Residual evolution for a system exhibiting pseudo-non-convergence. Specifications in the input file: residual_fermi = $10^{-7}$ eV, residual (density) = $10^5 /\text{cm}^3$, and iterations = 2000. The $\alpha_{\text{Fermi}}$ parameter was reduced to 0.0001.](image)

There are other patterns for finding convergence, but here only the most relevant ones have been shown.
8.6.6 Sweeping parameters

It is very common to use a sweep of specific variables within the input file, for example bias or any other user defined parameter.

It is important to have in mind that any change in the input file is equivalent to a simulation of a new system (for example when modifying doping), or the operation condition (temperature or bias). There is no mathematical reason that the solutions of two systems shall be similar. In other words, it is not expected that all solutions using different conditions will converge under the same criteria, for the entire range of variation of the sweep parameters. Eventually, for example, a sweep of bias from 0 to 8 Volts can use the same parameters for the whole simulation, but this is not the most common case.

A good strategy is to start the sweep of the parameters and verify at which value the solution does not longer converge. For saving time it is recommended to split the range of variation in two parts, and to follow the simulation only using the values of the parameter (for example, bias) that have still not converged. Trying to make the solution converge for a wide range of values for the sweep variable, using with a unique set of residuals and alpha_fermi, can become a very hard task, without the recommended range splitting.

8.6.7 ... and when nothing works

Our concern, in the development of our code, is to make it as accurate and fast as possible. Some simulations can be performed in a simple notebook, especially for 1D simulations.

Unfortunately, for some devices under specific conditions, making the system of Current-Schrödinger-Poisson converge in few iterations is a very specialized and time-consuming task. Observing the needs of our customers, nextnano is offering our customers the opportunity to perform this task on demand. Please, consult our schedules and fees when an extra assistance is required. Our experts in simulation can assist you to boost your project!

This document is copywritten by nextnano and it is distributed for free to nextnano customers. The company does not allow sharing, publishing or distributing by any means without its written permission.

Edited by Maria Cecilia da Silva Figueira.

8.7 Settings for maximizing performance

nextnano++ whose version is later than 2021_12_24 uses significantly more parallelization than previous versions.

The following settings are recommended unless a notebook or an ancient PC is used. This is illustrated for the example of a CPU 8 cores and 16 threads:

1. If many small (e.g. 1D) runs are performed at a time:
   • max. number of simulations = number of CPU cores (not CPU threads!), i.e. 8 in the present case.
   • max. number of threads per simulation = 1
   • process priority: on a dedicate machine, or if you are not bothered by CPU load: use “normal” instead of “below normal”

1. If only individual runs are performed at a time, which optionally may be large (e.g. 3D):
   • max. number of simulations = 1
   • max. number of threads per simulation = number of CPU cores (not CPU threads!), i.e. 8 in the present case
   • process priority: on a dedicate machine, or if you are not bothered by CPU load: use “normal” instead of “below normal”

The max. number is specified from “Tools->Options->Simulation” in nextnanomat. The process priority is specified from “Tools->Options->Expert settings”.

On Linux, corresponding optimal settings apply.
Also note that especially 3D simulations may write huge amounts of data (GBytes) to disk, i.e. using SSDs is highly recommended.
9.1 Introduction

Not all keywords have been documented in the new manual.
The old documentation for the nextnano³ software is available here: https://www.nextnano.com/nextnano3/

9.2 Input File

9.2.1 Macro features

Comment signs  Everything in the same line following after such a symbol is treated as a comment.

- #
  Note: This is the same comment sign as used in the nextnano++ software.
- !
- //
- /*This is a comment.*/
  The end tag */ is optional.
  Note: No line breaks are allowed for the comments /* ... */.
- <This is a comment.>

Note: However, the following are exceptions and are treated differently.

- #IF
- !IF

Both are special statements for conditional comments (see below).

Note: XML tags can be used in the input file, e.g. to provide additional meta information. They are ignored by nextnano³ but could be used in e.g. nextnanomat. Everything that is between these brackets < ... > is replaced with blanks. This can be used to define XML tags such as:

```xml
<variables>
  # comment
  %TEMPERATURE = 300.0 # (DisplayUnit:K)
  %WIDTH = 10.0 # (DisplayUnit:nm)
  %HEIGHT = 20.0 unit "nm" # (DisplayUnit:nm)
</variables>
```

```xml
<variables>
  %TEMPERATURE = 300.0 </variables> # (DisplayUnit:K)
```
Macro 1: Variables

The % symbol is used to define a variable such as \%VARIABLE = that should be replaced with another string or value. Variables are case-sensitive.

A variable is only defined and initialized if the first character in a line starts with the % symbol.

This macro distinguishes between *strings* (default), i.e. simple string replacements where 1 and 1.0 are treated as a *string*) and *functions* (i.e. arithmetic operations if \%FunctionParser = yes) where both 1 or 1.0 are treated as a *real* number. Both can be combined but it is important to understand the differences to avoid errors. For instance, for some *specifiers*, the input parser expects an *integer* number and not a *real* number. So real numbers have to be converted back to integers in some cases. This macro is supported by the nextnano\textsuperscript{mat} GUI.

String feature

*Find & Replace* of variables

Example: \%WIDTH = 10.0

nextnano\textsuperscript{3} supports a simple *Find & Replace* feature where *strings* in the input file can be replaced with another *string*.

**How to use it** Just put something like the following lines anywhere into your input file. We recommend that you put it at the beginning of the input file for better readability.

```
# By default, we do not evaluate functions.
# Everything is just a simple "Find & Replace String" operation.
#---------------------------------------------------------------
%width = 10.0 # quantum well width
-> (DisplayUnit:nm)
%length = 10  # quantum well length
-> (DisplayUnit:nm)
%size = 10.0  # quantum well size
-> (DisplayUnit:nm)
%z_coordinates = -21.0 101.0  # from z = -21 to z = 101
-> (DisplayUnit:nm)
%well_material = \[GaAs\]  # quantum well material: GaAs
%barrier_material = \"AlAs\"  # quantum barrier material: AlAs
%zmin = -10.0  # zmin = -10 nm
-> (DisplayUnit:nm)
%zmax = 10.0  # zmax = 10 nm
-> (DisplayUnit:nm)

# derived variable: quantum_region = zmin zmax = -10.0 10.0
%quantum_region = %zmin %zmax  # (DisplayUnit:nm)
%hkl_z_direction = 0 1 1  # \[0 1 1\] growth direction

# The whole string including blanks after the first '=' sign is used.
%Region1 = region_number 1  # geometry
->
```

The % sign indicates that you define a *variable*. When the input file is processed, all occurrences of %width, %z-coordinates, %well_material, ... are replaced by the according strings.

A *comment* sign (e.g. 
\#) is allowed in this line. Blanks are allowed within the string which is useful for reading in arrays of numbers. Almost everything is allowed.

**Restrictions**
• The variables and their definitions are case-sensitive.
• A variable name must not contain a - sign, e.g.

\[ \%cb\text{mass} = 0.067 \ # \ (\text{DisplayUnit:m0}) \]

is not allowed. Use underscores _ instead.
• In one line, only one variable can be initialized.

**Function feature**

**Find, Evaluate & Replace**

Example: \%WIDTH = \%LENGTH / 2.0

The statement \%FunctionParser = yes switches the function parser on (default: off).

The following operators

<table>
<thead>
<tr>
<th>Round arithmetic brackets</th>
<th>( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power (exponentiation)</td>
<td>^</td>
</tr>
<tr>
<td>Arithmetic multiplication, division</td>
<td>* /</td>
</tr>
<tr>
<td>Arithmetic plus and minus</td>
<td>+ -</td>
</tr>
</tbody>
</table>

and functions are supported:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt()</td>
<td>square root (\sqrt{\cdot})</td>
</tr>
<tr>
<td>exp()</td>
<td>exponential function (\exp(\cdot))</td>
</tr>
<tr>
<td>log()</td>
<td>natural logarithm (\log)</td>
</tr>
<tr>
<td>log10()</td>
<td>decadic logarithm (base 10) (\log_{10})</td>
</tr>
<tr>
<td>sin()</td>
<td>sine (\sin(\cdot))</td>
</tr>
<tr>
<td>cos()</td>
<td>cosine (\cos(\cdot))</td>
</tr>
<tr>
<td>tan()</td>
<td>tangent (\tan(\cdot))</td>
</tr>
<tr>
<td>asin()</td>
<td>arcsine (\sin^{-1}(\cdot))</td>
</tr>
<tr>
<td>acos()</td>
<td>arccosine (\cos^{-1}(\cdot))</td>
</tr>
<tr>
<td>atan()</td>
<td>arctangent (\tan^{-1}(\cdot))</td>
</tr>
<tr>
<td>sinh()</td>
<td>hyperbolic sine (\sinh(\cdot))</td>
</tr>
<tr>
<td>cosh()</td>
<td>hyperbolic cosine (\cosh(\cdot))</td>
</tr>
<tr>
<td>tanh()</td>
<td>hyperbolic tangent (\tanh(\cdot))</td>
</tr>
<tr>
<td>abs()</td>
<td>absolute value (</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
# If '%FunctionParser = yes', then from now on we evaluate all functions.
# If '%FunctionParser = no' (or not present), no functions are evaluated.
# "Find & Evaluate & Replace String" operation.
#------------------------------------------------------------
%
%FunctionParser [yes] # This is an important line!
%a 4.0 # a = 4.0
%b 3 # b = 3
%function1 sqrt(\%a) / \%b 2 # Evaluate: sqrt(a) / b / 2 = 2.0 - 1.5 = -0.5
%xalloy %function1 # Insert evaluated value for %function1.
```

(continues on next page)
# Strings containing blanks are also possible. 
# They must be in quotation marks " " or apostrophes ".
%string1 = "GaAs"
%string2 = "GaAs"
%string3 = "zero-potential"

# Logical variables that are .TRUE. or .FALSE. can be defined 
# and used in conditional #IF statements.
%Logical1 = .TRUE.
%Logical2 = .FALSE.

We have to be careful about the value of a variable. Should it be regarded as a string (default), or as a function (%FunctionParser = yes)?

---
# Treat variables as strings (default): 
# -------------------------------------
# e.g. %variable1 = %zmin %zmax # '%zmin %zmax' is treated as a string
# %variable2 = 2.5e0 # '2.5e0' is treated as a string
# %variable3 = 2.5 # '2.5' is treated as a string, without exponent 'e0'
---
%radius = 5e0 # quantum dot radius = 5 nm
(DisplayUnit:nm)
%grid_spacing = 0.5 # grid spacing
(DisplayUnit:nm)
---
# From now on, evaluate all functions:
# ------------------------------------
# e.g. %variable1 = %zmin %zmax # '%zmin %zmax' is treated as a formula
# (which will produce an error in this case)
# %variable2 = 2.5 # '2.5' is converted to a real number including exponent 'e0'
---
%FunctionParser = yes # use function parser
%radius = 5 # quantum dot radius = 5 nm
(DisplayUnit:nm)
%boundary = %radius + 1 # device_width/2 = 5 nm + 1 nm
(DisplayUnit:nm)
---

Special treatment of integers The parser distinguishes between integers and real numbers. Thus we have to be careful when using functions. This means that we have to convert the real values back to integers. Integers will be rounded down up >= .5 or rounded down if < .5, i.e. they will be rounded to the nearest integer.

- Option a) (recommended)

---
# If you need an integer, then use INT(...).
# %nodes contains an integer value.
---
%nodes = INT( %width / %grid_spacing )

- Option b) (deprecated)

---

# If you need an integer, then simply call the variable '%INT(<name>)'.
# %INT(nodes) contains an integer value.
%INT(nodes) = %width / %grid_spacing - 1

• Option c) (deprecated)

# If you need an integer, then simply call the variable '%INT(<name>)'.
# %nodes is real number.
# %INT(nodes) contains an integer value.
%nodes = %width / %grid_spacing - 1
%INT(nodes) = %nodes

Integer numbers can be defined using quotation marks "--" or INT():

%NumberOfEigenvalues = "10" # 10 eigenvalues
%NumberOfEigenvalues = INT(10) # 10 eigenvalues

Integer arrays can be defined using quotation marks "--":

%hkl_z_direction = "0 1 1" # [0 1 1] growth direction

Conditional comments

#IF or !IF statements can also be used to define conditional comments, i.e. variables can be used to define conditional comments.

#IF or !IF are not case-sensitive, i.e.
# if or !if also work.

#IF %TemperatureDependentBandGap varshni-parameters-on = yes

Such a statement has the following meaning:

• If the value of %TemperatureDependentBandGap is either .TRUE. or 1 or any other nonzero value, then the statement is included in the input file.

• If the value of %TemperatureDependentBandGap is either .FALSE. or 0 or undefined, then the statement is simply ignored as if it were a comment.

In this example, the text is always commented out, unless %IncludeHoles is defined with a value %IncludeHoles /= 0 or /= .FALSE..

%IncludeHoles = 1 # or %IncludeHoles = 1

!IF %IncludeHoles $quantum-model-holes
!IF %IncludeHoles $model-number = 1
!IF %IncludeHoles $model-name = effective-mass
!IF %IncludeHoles $cluster-numbers = 1
!IF %IncludeHoles $valence-band-numbers = 1 2 3
!IF %IncludeHoles $number-of-eigenvalues-per-band = 10
!IF %IncludeHoles $end_quantum-model-holes

In the following, the evaluated value of %IncludeElectrons is 0, therefore the three lines starting with !IF %IncludeElectrons are treated as comments.
If the variable definition

```
%IncludeHoles =
%IncludeElectrons =
```

appears after the line

```
%FunctionParser = yes
```

quotation marks " or apostrophes ' must be used for the logical strings .TRUE./.FALSE. to indicate that they have to be treated as strings:

```
%IncludeHoles = "TRUE" # or
%IncludeHoles = 'TRUE'
```

Variable replacement

- Once the input file has been processed, the input file (*.in) is copied to the output folder. Additionally, this output folder contains a file called *.in.mo_macro which contains the input file with all variables having been replaced with their values.
- All variables and their values that are used in a simulation are written to the output folder into a file called variables_input.txt.

Macro 2: Namelist (deprecated)

Alternatively, namelists are supported for macros. The input syntax for this macro is implemented as a NAMELIST which is a standard Fortran feature.

nextnano³ supports a simple Find & Replace feature where strings in the input file can be replaced with another string. The first lines of the input file can contain the following additional entries.

```
& macro filename = 'macro.in' /

macro Find_and_Replace_1 = '$WIDTH', '0d0 10.0d0'
Find_and_Replace_2 = 'x-nodes = 9', 'x-nodes = 29'
Find_and_Replace_3 = 'GaAs', 'InP'
/
```

These three lines have the following meaning:
- Find $WIDTH and replace it with 0d0 10.0d0.
- Find x-nodes = 9 and replace it with x-nodes = 29.
- Find GaAs and replace it with InP.

Download example file: macro.in

Example 1

```
! ...
!

(continues on next page)
```
A macro can be used to simply search for a specific string and to replace it with another string.

Example: Replace the material 'GaAs' with 'InAs'.

```
material-name = GaAs
--> material-name = InAs
```

Replace the quantum well width '$WIDTH' with '10d0':

```
x-coordinates = -$WIDTH $WIDTH
--> x-coordinates = -10d0 10d0
```

Replace the string '2*$WIDTH' with '20d0':  
(Mathematical operations are not supported yet. They are treated as strings.)
```
x-coordinates = 0d0 2*$WIDTH
--> x-coordinates = 0d0 20d0
```

Replace the string '$WIDTH+$SIZE' with '30d0':  
(Mathematical operations are not supported yet. They are treated as strings.)
```
x-coordinates = 0d0 $WIDTH+$SIZE
--> x-coordinates = 0d0 30d0
```

Replace the quantum well width '$WIDTH' with '0d0 10d0':
```
x-coordinates = $WIDTH
--> x-coordinates = 0d0 10d0
```

Simulate along x, y or z direction:
```
Find_and_Replace_1 = '$DIRECTION','x' ! simulation along x
Find_and_Replace_2 = '$DOMAINTYPE','1 0 0' ! simulation along x

Find_and_Replace_1 = '$DIRECTION','y' ! simulation along y
Find_and_Replace_2 = '$DOMAINTYPE','0 1 0' ! simulation along y
```

Example 2
```
$macro filename = '' ! (optional)
-macro file to be read in, use '' for no macro input file
Find_and_Replace_1 = '$WIDTH','0d0 10.0d0' ! is ignored
-if macro file is read in
Find_and_Replace_2 = 'x-nodes = 9','x-nodes = 29' ! is ignored
-if macro file is read in
Find_and_Replace_3 = 'GaAs','InP' ! is ignored
-if macro file is read in
```

9.2. Input File
Currently up to 40 Find & Replace definitions can be specified.

9.2.2 The input file keywords

Here you can find detailed descriptions about the keywords and specifiers for the input file. The general definition file is `keywords.val` where valid keywords are defined.

Keywords for setting up the device geometry, material and general settings

- `$input_filename`
- `$global-parameters` (temperature)
- `$simulation-dimension`
- `$simulation-flow-control`
- `$strain-minimization-model`
- `$electric-field`
- `$magnetic-field`
- `$numeric-control`

Keywords for setting up the device geometry

- `$domain-coordinates`
- `$regions`
- `$region-cluster`
- `$grid-specification`

Keywords for setting up material properties

- `$material`
- `$binary-zb-default`
- `$binary-wz-default`
- `$ternary-zb-default`
- `$ternary-wz-default`
- `$alloy-function`
- `$import-data-on-material-grid`

Keywords for doping
• $doping-function
• $impurity-parameters
• $material-interfaces
• $interface-states

Keywords for output
• $global-settings (output directory)
• $output-bandstructure
• $output-densities
• $output-strain
• $output-1-band-schroedinger
• $output-kp-data
• $output-current-data
• $output-raw-data
• $output-grid
• $output-geometry
• $output-material
• $output-file-format
• $output-section

Keywords for quantum calculations
• $quantum-regions
• $quantum-cluster
• $quantum-model-electrons
• $quantum-model-holes
• $quantum-bound-states
• $quantum-dot-layer-density
• $optical-absorption
• $tighten

Keywords for current calculations
• $poisson-boundary-conditions
• $voltage-sweep
• $current-regions
• $current-cluster
• $current-models
• $simple-drift-models
• $Monte-Carlo (not available yet)

Keywords for recombination
• $Auger-recombination
• $direct-recombination (radiative)
• $SRH-recombination (Shockley-Read-Hall)
• $quantumstate-recombination-rates

Keywords for mobility models
• $mobility-model-constant
• $mobility-model-arora
• $mobility-model-dar (Darwish)
• $mobility-model-lom (Lombardi)
• $mobility-model-masetti
• $mobility-model-minimos
• $mobility-model-simba

Keywords for electrolytes
• $electrolyte
• $electrolyte-ion-content
• $buffer-solutions
• $buffer-constant-A(T)

Keywords for NEGF
• $CBR-current (ballistic)
• $global-parameters-NEGF
• $scattering-mechanisms
• $contact-type
• $damping-parameters
• $roughness-profile
• $left-contact-potential-profile
• $right-contact-potential-profile
• $potential-profile (deprecated)
• $mass-profile (deprecated)
• $alloy-profile (deprecated)
• $nonparabolicity-profile (deprecated)
• $dielectric-profile (deprecated)
• $doping-function-NEGF (deprecated)
• $NEGF-spintronics (for NEGF spintransport code)

Other keywords
• $warnings
9.2.3 Keywords

$simulation-dimension

Simulation dimension and orientation

Simulation coordinate system

In general, the underlying coordinate system for the simulation is a three-dimensional cartesian coordinate system (x,y,z). Dependent on the type of simulation, i.e. whether a one-, two- or three-dimensional simulation has to be performed, the corresponding number of axes has to be selected out of this triplet of coordinate directions. For these specifications, the keyword $simulation-dimension together with its specifiers dimension and orientation have to be used in the input file.

Specification of the simulation domain dimension and its orientation

dimension

  type integer
  presence required
  options 1 (1D)
           2 (2D)
           3 (3D)

Specify here if a 1D, 2D or 3D simulation is performed.

orientation

  type integer array of dimension 3
  presence required
  options for 1D simulation: 1 0 0 1 0 0 0 1
                         for 2D simulation: 1 1 0 1 0 1 1
                         for 3D simulation: 1 1 1

The specifier orientation defines the orientation of the simulation domain relative to the (x,y,z) coordinate system. A three-dimensional array containing 0 and/or 1 is expected. The relevant axes in this array are selected using 1.

Specify here if the simulation is performed along a direction, in a plane or in a volume.

  • simulation direction (1D: x, y or z axis)
  • simulation plane (2D: (x,y), (x,z) or (y,z) plane)
  • simulation volume (3D)

Example 1 (1D)

$simulation-dimension

  dimension 1 ! 1D simulation
  orientation 0 0 1 ! z axis

$end_simulation

Example 2 (2D)

$simulation-dimension

  orientation 1 1 0

$end_simulation

Example 3 (3D)

$simulation-dimension

  orientation 1 1 1

$end_simulation
Example 2 (2D)

```plaintext
$simulation ! 2D simulation
dimension 2 ! (x,y) plane
orientation 1 1 0
$end_simulation
```

Example 3 (3D)

```plaintext
$simulation ! 3D simulation
dimension 3
orientation 1 1 1
$end_simulation
```

$regions

- **General specifiers**
- Details of specification
  - 1-dimensional objects (only possible in 1D simulations)
  - 2-dimensional objects
  - 3-dimensional objects

**General specifiers**

To build up a geometry, there are - dependent on the dimension - various basic geometry elements available. These geometry elements are specified within the keyword $regions and can be clustered to a bigger object later on.

```plaintext
$regions

integer          required
integer          required
character        required
double_array     optional
double_array     optional
double_array     optional
double           optional
double_array     optional
double_array     optional
$end_regions
```

region-number
**region-number**

- **type**: integer (≥ 1)
- **presence**: required
- **example**: 4

An integer number to refer to geometry element. Numbering must be unique. All region numbers together must form a dense set $1, 2, 3, \ldots, \text{maxnumber}$.

**region-priority**

- **type**: integer (≥ 1)
- **presence**: required
- **example**: 2

A positive integer to set overwriting priority. In case of overlapping regions, the region with higher priority (= higher numerical value) overwrites the region with lower priority.

**base-geometry**

- **type**: character
- **presence**: required
- **example**: rectangle

Type of geometry object.

- **3D**: cuboid, sphere, ...
- **2D**: rectangle, circle, ...
- **1D**: line

These are specified in detail later.

**x-coordinates**

- **type**: double_array
- **presence**: optional
- **unit**: [nm]
- **example**: -10.0 10.0

This is for cuboid, rectangle, line. The values represent $x_{\text{min}}, x_{\text{max}}$ from left.

**y-coordinate**

- **type**: double_array
- **presence**: optional
- **unit**: [nm]
- **example**: -10.0 10.0

This is for cuboid, rectangle, line. The values represent $y_{\text{min}}, y_{\text{max}}$ from left.

**z-coordinates**

- **type**: double_array
- **presence**: optional
- **unit**: [nm]
- **example**: -10.0 10.0

This is for cuboid, rectangle, line. The values represent $z_{\text{min}}, z_{\text{max}}$ from left.

**center**

- **type**: double_array

9.2. Input File
presence  optional
unit  [nm]
example  2.0
This is for circle (2D), sphere (3D).

radius

type  double
unit  [nm]
presence  optional
example  3.0
This is for circle (2D), sphere (3D).

base-coordinates

type  double_array
presence  optional
unit  [nm]
example  10.0  20.0  10.0  20.0  15.0  15.0
This is for obelisk, cone, truncated-cone, trapezoid. The values represent xmin
xmax ymin ymax zmin zmax of obelisk (cone) base plane. (one pair must be equal)

top-coordinates

type  double_array
presence  optional
unit  [nm]
example  10.0  20.0  10.0  20.0  30.0  30.0
This is for obelisk, cone, truncated-cone, trapezoid, semiellipsoid: The
values represent xmin xmax ymin ymax zmin zmax of obelisk (cone) base plane (one pair
must be equal) or x y z for semiellipsoid.

corner-coordinates

type  double_array
presence  optional
unit  [nm]
example  10.0  10.0  10.0  10.0  30.0  10.0  20.0  20.0  10.0
Triangle corner coordinates, interpreted via orientation and similar for polygon. For triangular
prism: x1 y1 z1 x2 y2 z2 x3 y3 z3

semi-ellipse-base

type  double_array
presence  optional
unit  [nm]
example  60.0  120.0  40.0  40.0
For semiellipse: base line ellipse. These are two pairs of xmin, xmax ymin, ymax zmin,
zmax - one pair equal numbers

semi-ellipse-top

type  double_array
presence: optional

unit [nm]

exa[mp]le 100.0 20.0

For semiellipse: top coordinate pair e.g. for coordinate orientation (101) -> (x,z)

Details of specification

- 1-dimensional objects (only possible in 1D simulations)
  - line
- 2-dimensional objects
  - rectangle
  - circle
  - triangle
  - trapezoid
  - polygon
    - regular-polygon (not implemented yet but could be added -> use polygon instead)
    - hexagon (not implemented yet but could be added -> use polygon instead)
    - semiellipse
- 3-dimensional objects
  - cuboid
  - sphere
  - cylinder (not implemented yet but could be added -> use truncated-cone instead)
  - obelisk
  - hexagonal-obelisk
  - cone
  - truncated-cone
  - semiellipsoid
  - regular-prism (not implemented yet but could be added)
  - hexagonal-prism (not implemented yet but could be added -> use hexagonal-obelisk instead)
  - triangular-prism
  - polygonal-prism (not implemented yet but could be added)
  - pyramid (not implemented yet but could be added -> use obelisk instead)
  - regular_pyramid (not implemented yet but could be added -> use obelisk instead)
  - hexagonal_pyramid (not implemented yet but could be added -> use hexagonal-obelisk instead)
  - polygonal_pyramid (not implemented yet but could be added)
1-dimensional objects (only possible in 1D simulations)

line

```
$regions
number  1
geometry line
priority 1
coordinates x
$end_regions
```

- Chosen coordinates must be consistent with simulation orientation.

2-dimensional objects

rectangle

```
$regions
number  1
geometry rectangle
priority 1
x-coordinates x
y-coordinates y
$end_regions
```

- Two pairs of delimiting coordinates are required. Whether these have to be \(x\)-coordinates and \(y\)-coordinates as in the example above, or another combination (e.g. \(x, z\)) depends on the simulation orientation which is specified already.

circle

```
$regions
number  1
geometry circle
priority 1
center x y
radius r
$end_regions
```

- The circle is defined by a center with coordinates \((x,y)\) and a radius \(r\).

triangle (can also be specified using polygon)

```
$regions
number  1
geometry triangle
priority 1
corner coordinates x1 y1 x2 y2 x3 y3
$end_regions
```

- The corner coordinates refer to the plane, specified by the simulation orientation.
polygon

$regions
  number       1
  geometry     polygon
  priority     1
  coordinates  x1 y1 x2 y2 x3 y3 x4 y4 x5 y5 ...
$end_regions

• The corner coordinates of a polygon refer to the plane, specified by the simulation orientation.
• The vertices may be listed clockwise or anticlockwise. The first point could be repeated, i.e. the first point and the last point may be identical but this is not required.
• The input polygon may be a compound polygon consisting of several separate subpolygons. If so, the first vertex of each subpolygon must be repeated.

trapezoid (can also be specified using polygon)

$regions
  number       1
  geometry     trapezoid
  priority     1
  coordinates  -15.0 15.0 -20.0 -20.0 ! xmin xmax ymin...
  ymin coordinates  -25.0 25.0 -40.0 -40.0 ! xmin xmax ymin...
$end_regions

• For each base-coordinates and top-coordinates, 2 values must be equal (meaning only 2 values can be equal), e.g. ymin=ymax=-40.0. Then one boundary of the object is a line at y=-40 nm. Base and top planes of the trapezoid must be in a coordinate plane. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has identical values (ymin = ymax). This also implies that base and top lines are parallel to either the x or the y axis.

The example given above shows the blue trapezoid at the bottom of this figure. The top line extends in x direction from -15 to 15 nm. The top line has a constant y coordinate of y = -20 nm The base line extends in x direction from -25 to 25 nm. The base line has a constant y coordinate of y = -40 nm.
semiellipse

$\textbf{semiellipse}$

- **semi-ellipse-base**: Here, 2 values must be equal (meaning only 2 values can be equal), e.g., $y_{\text{min}}=y_{\text{max}}=40.0$. Then one boundary of the object is a line at $y=40$ nm. Base plane of the semiellipse must be in a coordinate plane. This plane is identified by the implicit rule, that a pair of coordinate values (e.g., $y_{\text{min}} y_{\text{max}}$) has identical values ($y_{\text{min}} = y_{\text{max}}$).

- **semi-ellipse-top**: This defines a point which determines the height of the semiellipse. In our example ($y_{\text{min}} = y_{\text{max}}$), the plane is in the (x,z)-coordinate plane. Top coordinates specify an arbitrary point “above” the ellipse, representing the base of the semiellipse.

- Top coordinate must be lower than upper coordinate of base line border in direction of axis mentioned above.

- Top coordinate must be higher than lower coordinate of base line border in direction of axis mentioned above.

- e.g.) in case of $y_{\text{min}} = y_{\text{max}}$, $x_{\text{min}} < x_{\text{1}} < x_{\text{max}}$ and ($y_{\text{1}} < y_{\text{min}}=y_{\text{max}}$ or $y_{\text{1}} > y_{\text{min}}=y_{\text{max}}$)

**Example 1**

semi-ellipse-base = 60.0 120.0 40.0 40.0  
semi-ellipse-top = 100.0 20.0

From these data, the following points are extracted: (point: (x,y))

- base: (60,40) (120,40)
- top: (100,20)

**Example 2**

semi-ellipse-base = 60.0 120.0 40.0 40.0  
semi-ellipse-top = 100.0 60.0

We changed the $y_{\text{1}}$ coordinate of semi-ellipse-top from 20.0 to 60.0.
**Example3**

semi-ellipse-base = 40.0 40.0 20.0 80.0  
semi-ellipse-top = 100.0 60.0

Here we changed semi-ellipse-base: Now the two x coordinates have identical values.

**Example4**

semi-ellipse-base = 40.0 40.0 50.0 70.0  
semi-ellipse-top = 100.0 60.0

Here we changed the ymin and ymax coordinates of semi-ellipse-base: 
Now the y extension of the semiellipse is restricted from ymin = 50 nm to ymax = 70 nm. The baseline is at the fixed value for x = 40 nm.

**Example5**

semi-ellipse-base = 40.0 40.0 40.0 80.0  
semi-ellipse-top = 60.0 60.0
Here we built a circle out of 2 semi-ellipses. However, it is obviously easier to use `circle` instead.

**Example 6**

\[
\begin{align*}
\text{semi-ellipse-base} &= 42.0 \ 42.0 \ 40.0 \ 80.0 \\
\text{semi-ellipse-top} &= 60.0 \ 60.0 \\
\text{semi-ellipse-base} &= 38.0 \ 38.0 \ 40.0 \ 80.0 \\
\text{semi-ellipse-top} &= 20.0 \ 60.0
\end{align*}
\]

Same as example 5 but this time, we moved the baselines a little bit apart from each other to make example 5 easier to understand.
### 3-dimensional objects

#### cuboid

<table>
<thead>
<tr>
<th>$regions$</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
</tr>
<tr>
<td>geometry</td>
</tr>
<tr>
<td>priority</td>
</tr>
<tr>
<td>coordinates</td>
</tr>
</tbody>
</table>

*The surfaces of the cuboid are assumed to be in coordinate planes of the simulation coordinate system. The coordinates above specify the six coordinate planes which limit the cuboid.*

#### sphere

<table>
<thead>
<tr>
<th>$regions$</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
</tr>
<tr>
<td>geometry</td>
</tr>
<tr>
<td>priority</td>
</tr>
<tr>
<td>center</td>
</tr>
<tr>
<td>radius</td>
</tr>
</tbody>
</table>

*The sphere is defined by a center with coordinates $(x,y,z)$ and a radius $r$.*
oblisk

\begin{verbatim}
$regions
  number           1
  geometry         obelisk
  priority         1
  base-coordinates x x x x x x x x x x
  top-coordinates x x x x x x x x x x
$end_regions

- Base and top plane of the obelisk have to be in parallel coordinate planes. These planes are
  identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has the same
  value (ymin = ymax).
- In this example, the plane is in the (x,z)-coordinate plane. The remaining four coordinates
  specify a rectangle in the corresponding plane.
\end{verbatim}

truncated-cone and cylinder

\begin{verbatim}
$regions
  number           1
  geometry         truncated-cone
  priority         1
  base-coordinates x x x x x x x x x x
  top-coordinates x x x x x x x x x x
$end_regions

- A cone with its apex cut off by a plane is called a truncated cone. In our implementation, the
  truncated cone is bounded by two ellipses of different size that are aligned parallel to each other.
- Base and top plane of the truncated cone have to be in parallel coordinate planes. This plane is
  identified by the implicit rule, that a pair of coordinate values (e.g. xmin xmax) has the same
  value (xmin = xmax).
- In this example, the plane is in the (y,z)-coordinate plane. ymin ymax and zmin zmax specify
  the diameter of the truncated cone top and base in the y and z direction, respectively. This
  corresponds to the specification of ellipses in the base and top plane.
\end{verbatim}

How to specify a cylinder?

A cylinder is specified as a special case of a truncated-cone where the two boundary planes are
circles. For a truncated cone, one specifies base and top coordinates.

Let us assume we have a spherical cylinder of diameter 10 nm and height 15 nm. Then the base
and top coordinates would be, for example, base-coordinates = (10.0 20.0 10.0 20.0
15.0 15.0) = (10,20,10,20,15,15)
top-coordinates = (10.0 20.0 10.0 20.0 30.0 30.0
(xmin,xmax,ymin,ymax,zmin,zmax) = (10,20,10,20,30,30)

In other words, the x and y-coordinates specify the principal axes of the bottom and top ellipse (or cir-

cle) of the cylinder, respectively, and the z-coordinates specify the planes in which these two ellipses
lie.
cone

$regions

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>region_number</td>
<td>1</td>
</tr>
<tr>
<td>region_geometry</td>
<td>cone</td>
</tr>
<tr>
<td>region_priority</td>
<td>1</td>
</tr>
<tr>
<td>base_coordinates</td>
<td>xmin xmax</td>
</tr>
<tr>
<td></td>
<td>ymin ymax</td>
</tr>
<tr>
<td></td>
<td>zmin zmax</td>
</tr>
<tr>
<td>coordinates</td>
<td>x y z</td>
</tr>
</tbody>
</table>

$end_regions

Base plane of the cone has to be in parallel to the coordinate system planes. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. xmin xmax) has the same value (xmin = xmax). In this example, the plane is in the (y,z)-coordinate plane. ymin ymax and zmin zmax specify the diameter of the cone base in the y and z direction, respectively. This corresponds to the specification of an ellipse in the base plane. The top of the cone, the apex, is defined by the point (x,y,z).

The cone does not have to be right circular (where circular means that the base is a circle and right means that the axis passes through the center of the base at right angles to its plane).

Oblique cones are allowed, in which the axis does not pass perpendicularly through the center of the base.

The following figure shows several cones and truncated-cones (including the special case of cylinders).

The cones are defined as follows:

!-------------------------------------------------------------------------
! This is a cylinder. The base and top planes are parallel to the (y,z) plane.
!-------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>region_number</td>
<td>1</td>
</tr>
<tr>
<td>region_geometry</td>
<td>cylinder</td>
</tr>
<tr>
<td>region_priority</td>
<td>1</td>
</tr>
<tr>
<td>base_coordinates</td>
<td>xmin xmax</td>
</tr>
<tr>
<td></td>
<td>ymin ymax</td>
</tr>
<tr>
<td></td>
<td>zmin zmax</td>
</tr>
<tr>
<td>coordinates</td>
<td>x y z</td>
</tr>
</tbody>
</table>

(continues on next page)
This is a right circular cone. The base plane is parallel to the (y,z) plane.

This is a cone where the projection of the apex onto the base plane is located outside the base plane. The base plane is parallel to the (y,z) plane.

This is a truncated cone. The base plane is parallel to the (y,z) plane.
## semiellipsoid

Base plane of the semiellipsoid must be in a coordinate plane. This plane is identified by the implicit rule, that a pair of coordinate values (e.g. ymin ymax) has identical values (ymin = ymax).

In this example, the plane is in the (x,z)-coordinate plane. Top coordinates specify an arbitrary point "above" the ellipse, representing the base of the semiellipsoid.

**Example** A 3D sphere can be constructed from two semiellipsoids. However, it is obviously easier to use sphere instead.

In this example, the bottom planes of the two half-spheres are at z = 5 nm. The upper half-sphere extends from 5 nm to 6 nm, the lower half-sphere from 5 nm to 4 nm. The extensions in x and y directions for both half-spheres are from 4 nm to 6 nm.

Consequently, the sphere has a diameter of 2 nm.
**hexagonal-obelisk**

Base and top plane of the hexagonal-obelisk have to be in parallel coordinate planes. These planes are identified by the implicit rule, that a pair of coordinate values (e.g. \( z_{\text{min}} \), \( z_{\text{max}} \)) has the same value (\( z_{\text{min}} = z_{\text{max}} \)). In this example, the plane is in the (x,y)-coordinate plane. The remaining four coordinates specify a rectangle in the corresponding plane.

This geometry element is useful for wurtzite.

Many thanks to Lu Fu-Fa (Institute of Technology (CCIT), Taiwan, R.O.C.) for useful suggestions regarding the implementation of this geometry element.

Two hexagonal-obelisk shapes are possible:

- **‘hexagonal-cylinder’ with 6-fold rotational symmetry axis oriented along the z direction (i.e. \( z_{\text{min}} = z_{\text{max}} \)).**
  Requirements: - \( x_{\text{min}}(\text{base}) = x_{\text{min}}(\text{top}) \) - \( x_{\text{max}}(\text{base}) = x_{\text{max}}(\text{top}) \) - \( y_{\text{min}}(\text{base}) = y_{\text{min}}(\text{top}) \) - \( y_{\text{max}}(\text{base}) = y_{\text{max}}(\text{top}) \)

  **Note:** This condition should be fullfilled:

  - \( y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base}) > (x_{\text{max}}(\text{base}) - x_{\text{min}}(\text{base})) / 0.866 \)
  where \( 0.866 = \cos \pi / 6 \)

- **‘hexagonal-pyramid’ with 6-fold rotational symmetry axis oriented along the z direction (i.e. \( z_{\text{min}} = z_{\text{max}} \)).**
  Requirements: - \( x_{\text{min}}(\text{top}), x_{\text{max}}(\text{top}), y_{\text{min}}(\text{top}), y_{\text{max}}(\text{top}) \) arbitrary

  **Note:** This condition should be fullfilled:
For both shapes it holds:

- height of pyramid/cylinder: \(z_{\text{max}}(\text{top}) - z_{\text{max}}(\text{base})\) \((z_{\text{min}} = z_{\text{max}})\)
- width in x direction (distance between two parallel planes): \(x_{\text{max}}(\text{base}) - x_{\text{min}}(\text{base})\)
- width in y direction (distance between two corners) if the condition \(y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base}) > (x_{\text{max}}(\text{base}) - x_{\text{min}}(\text{base})) / 0.866\) is fulfilled:
  - If the above cited condition is not fulfilled, then the width is: \(y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base})\)
  - width along y > width along x if \(y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base}) > x_{\text{max}}(\text{base}) - x_{\text{min}}(\text{base})\)
- center of hexagonal base plane on x axis: \(0.5(x_{\text{min}}(\text{base}) + x_{\text{max}}(\text{base}))\)
- center of hexagonal base plane on y axis: \(0.5(y_{\text{min}}(\text{base}) + y_{\text{max}}(\text{base}))\): \(y_{\text{min}}(\text{base})\) and \(y_{\text{max}}(\text{base})\) can be used to shift the hexagon along the y axis.

Two sides of the hexagonal base plane are aligned parallel to the y axis. To rotate the hexagonal base plane by 30 degrees, the user has to specify values for \(x_{\text{min}}(\text{top})\), \(x_{\text{max}}(\text{top})\), \(y_{\text{min}}(\text{top})\), \(y_{\text{max}}(\text{top})\) so that it holds:

\[x_{\text{max}}(\text{top}) - x_{\text{min}}(\text{top}) > y_{\text{max}}(\text{top}) - y_{\text{min}}(\text{top})\]

In this case it holds:

- width in x direction (distance between two corners) if the condition \(x_{\text{max}}(\text{base}) - x_{\text{min}}(\text{base}) > (y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base})) / 0.866\) is fulfilled:
- width in y direction (distance between two parallel planes): \(y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base})\)
- width along y < width along x if \(x_{\text{max}}(\text{base}) - x_{\text{min}}(\text{base}) > y_{\text{max}}(\text{base}) - y_{\text{min}}(\text{base})\)
- center of hexagonal base plane on x axis: \(0.5(x_{\text{min}}(\text{base}) + x_{\text{max}}(\text{base}))\): \(x_{\text{min}}(\text{base})\) and \(x_{\text{max}}(\text{base})\) can be used to shift the hexagon along the x axis.

If the 6-fold rotational axis is oriented along the x (i.e. \(x_{\text{min}} = x_{\text{max}}\)) or y directions (i.e. \(y_{\text{min}} = y_{\text{max}}\)), cyclic permutations hold for the above statements.

**Example**

- Hexagonal shaped pyramid with flat top plane:
- Hexagonal shaped pyramid:
- Hexagonal shaped "cylinder":

### triangular-prism

```plaintext
$regions
number = 1
base-geometry = triangular-prism
priority = 1
coordinates = x_1 y_1 z_1 x_2 y_2 z_2 x_3 y_3 z_3 x_4 y_4 z_4 x_5 y_5 z_5 x_6 y_6 z_6
$end_regions
```

Restrictions: `triangular-prism` must be oriented so that the triangles are perpendicular to either the x, y or z directions.

Example: Triangles perpendicular to z direction. Then it must hold:

**corner-coordinates**

- \(z_1 = z_2 = z_3\)
In addition it holds:
• $x_1 = x_4$, $y_1 = y_4$
• $x_2 = x_5$, $y_2 = y_5$
• $x_3 = x_6$, $y_3 = y_6$

Example

```
$regions
  number  = 1
  geometry = triangle
  priority = 1
  coordinates = x1 y1 z1
                 x2 y2 z2
                 x3 y3 z3
                 x4 y4 z4
$end_regions
```

$region-cluster

Regions can be clustered to bigger objects by the keyword $region-cluster. Any region must be assigned to a cluster which is labeled by its cluster-number.

Also the so-called default region must be assigned to a cluster. The default region is the rest of the simulation domain which is not filled out by defined regions. It might happen that the default region is an empty region. Nevertheless, it must be assigned to a cluster.

In any case it holds:

\[
\text{default region number} = \text{maximum region number defined} + 1
\]

The regions which have to be clustered to one cluster are specified by their numbers like this region-number = 2 3 5.
Explanation of specifiers

**cluster-number**

- **type**: integer $>= 1$
- **presence**: required

An integer number to refer to clustered geometry element.

**region-numbers**

- **type**: integer array containing integers $>= 1$
- **presence**: required

Region numbers belonging to this cluster `cluster-number`.

**apply-constant-el-Fermi-level**

- **type**: double
- **presence**: optional
- **unit**: [eV]
- **example**: 0.2

Applies a constant electron Fermi level $E_{F,n}$ to this cluster. The energetic distance of the conduction band edges with respect to this Fermi level determines the electron density.

**apply-constant-hl-Fermi-level**

- **type**: double
- **presence**: optional
Applies a constant hole Fermi level $E_F$ to this cluster. The energetic distance of the valence band edges with respect to this Fermi level determines the hole density.

**Example**

We have defined 5 regions under keyword $regions$, region 2 and 4 are clustered to form cluster-number 2 (i.e. region 2 and 4 now will consist of the same material as a material kind is assigned to a cluster and not to a region). region-number 6 is an undefined default region (Assumption: We have defined only 5 regions.) and must be the default rest. Now there are two possibilities:

**Possibility 1:**

```
$region
  number  1  ! e.g. GaAs
  number  2  ! e.g. AlAs
  number  3  ! e.g. GaAs
  number  4  ! e.g. InP, 6 = default
$end_region
```

Here, the default region has the same material properties as cluster-number 4. The default region can consist of some undefined region (the remaining part of the simulation area which is not filled yet) or it can be empty if the whole simulation area is already filled with other objects.

**Possibility 2:**

```
$region
  number  1  ! e.g. GaAs
  number  2  ! e.g. AlAs
  number  3  ! e.g. GaAs
  number  4  ! e.g. InP
  number  5  ! e.g. InP, 6 = default
$end_region
```

Here, the default region has its own material properties which have to be defined later under keyword $material$. Again, the default region can consist of some undefined region (the remaining part of the simulation area which is not filled yet) or it can be empty if the whole simulation area is already filled with other objects.

**Useful application for a default region** If one has a complicated geometry such as a quantum dot which is surrounded by a GaAs cap layer, one can define the complicated geometry as usual and for the (complicated) surrounding material one just uses the default option.

**Note:** Be careful in 3D when you specify for strain calculation strain-minimization (see $strain-minimization-model$) to define your substrate as a separate cluster.
For quantum mechanical calculations, you have to define quantum regions (\textit{quantum-regions}) and quantum clusters on which certain quantum models (\textit{quantum-model-electrons}, \textit{quantum-model-holes}) are applied. As for the \textit{regions} and \textit{current-regions}, \textit{quantum-regions} can be \textit{clustered}. At least cluster number 1 has to be specified if the quantum mechanical properties should be calculated at all.

The syntax of the \textit{region-cluster}, \textit{quantum-cluster}, and \textit{current-cluster} is very similar. For more information on these specifiers see \textit{Region-cluster} and \textit{Current-cluster}.

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|}
\hline
\textit{quantum-cluster} & \textit{current-cluster} \\
\hline
\textit{cluster-number} & \textit{cluster-number} \\
\hline
\textit{type} integer & \textit{type} integer \\
\hline
\textit{presence} required & \textit{presence} required \\
\hline
\textit{region-numbers} & \textit{region-numbers} \\
\hline
\textit{type} integer array containing integers & \textit{type} integer array containing integers \\
\hline
\textit{presence} required & \textit{presence} required \\
\hline
\textit{deactivate-cluster} & \textit{deactivate-cluster} \\
\hline
\textit{type} character & \textit{type} character \\
\hline
\textit{presence} optional & \textit{presence} optional \\
\hline
\textit{value} yes or no & \textit{value} yes or no \\
\hline
\textit{default} no & \textit{default} no \\
\hline
\textit{apply-constant-el-Fermi-level} & \textit{apply-constant-el-Fermi-level} \\
\hline
\textit{type} double & \textit{type} double \\
\hline
\textit{presence} optional & \textit{presence} optional \\
\hline
\textit{unit} [eV] & \textit{unit} [eV] \\
\hline
\textit{example} 0.2 & \textit{example} 0.2 \\
\hline
\end{tabular}
\end{center}
\end{table}

Explanation of specifiers

\textbf{cluster-number}

\textit{type} integer \(>= 1\)

\textit{presence} required

An integer number to refer to clustered geometry element.

\textbf{region-numbers}

\textit{type} integer array containing integers \(>= 1\)

\textit{presence} required

Region numbers belonging to the cluster \textit{cluster-number}.

\textbf{deactivate-cluster}

\textit{type} character

\textit{presence} optional

\textit{value} yes or no

\textit{default} no

Flag to switch off quantum cluster. Switching off quantum calculation means classical simulation. This is a very convenient way to turn the quantum calculation off and on. For a first test, it is recommended to a classical calculation as it is much faster. Then you can check whether the geometry and the doping regions are defined correctly, and if the strain has the expected influence on the conduction and valence band edges. Once you are convinced that you have set up the structure correctly, you can do a quantum mechanical simulation.

\textbf{apply-constant-el-Fermi-level}

\textit{type} double

\textit{presence} optional

\textit{unit} [eV]

\textit{example} 0.2

Applies a constant electron Fermi level \(E_{F,n}\) to this quantum cluster. The energetic distance of the conduction band edges with respect to this Fermi level determines the electron density.

\textbf{apply-constant-hl-Fermi-level}

\textit{type} double

\textit{presence} optional
unit \[\text{[eV]}\]

**example** \[-0.1\]

Applies a constant hole Fermi level $E_{F,p}$ to this quantum cluster. The energetic distance of the valence band edges with respect to this Fermi level determines the hole density.

**Examples**

Here, only one quantum cluster is defined.

```
!-------------------------------------------------------------
$quantum
cluster-number 1
$end_quantum
!-------------------------------------------------------------
```

Here, two quantum regions are combined to form a common quantum cluster.

```
!-------------------------------------------------------------
$quantum
cluster-number 1
region-numbers 1 2
$end_quantum
!-------------------------------------------------------------
```

Here, three separate quantum clusters are defined that are treated independently, i.e. three different Schrödinger equations are solved. Typically, it is best to have only one quantum cluster. However, in some cases, using separate clusters can lead to faster calculations. But this should be used with care, as often the wavefunctions with higher energies are distributed over the whole device.

```
!-------------------------------------------------------------
$quantum
cluster-number 1
region-numbers 1
cluster-number 2
region-numbers 2
cluster-number 3
region-numbers 3
$end_quantum
!-------------------------------------------------------------
```

**$current-cluster**

For current calculations, you have to define current regions (**$current-regions**) and current clusters on which certain current models (**$current-models**) are applied. As for the **regions** and **quantum-regions**, **current-regions** can be **clustered**. At least cluster number 1 has to be specified if the current has to be calculated at all.

The syntax of the **region-cluster**, **quantum-cluster**, and **current-cluster** is very similar. For more information on these specifiers see **$region-cluster** and **$quantum-cluster**.

```
$current
cluster-number integer optional
integer integer_array required
character
$end_current
```

Explanation of specifiers

**cluster-number**

* type integer $\geq 1$
* presence required

An integer number to refer to clustered geometry element.
region-numbers
  type integer array containing integers >= 1
  presence required
Region numbers belonging to cluster cluster-number.
deactivate-cluster
  type character
  presence optional
  value yes or no
  default no
Flag to switch off current clusters. Switching off current calculation means equilibrium simulation.
This is a very convenient way to turn the current calculation off and on.

Example

```
!----------------------------------------------
$current
| cluster-number | 1 |
| region-numbers | 1 |
| deactivate-cluster | no |
| !deactivate-cluster | yes |
$end_current
!----------------------------------------------
```

$current-regions

In order to make it possible to combine several different modes of current calculation in one device, there is an extra set of regions and clusters for the calculation of the current. The syntax is the same as for the other regions ($regions, $quantum-regions).

If no current region is specified the whole device is numbered as current region 1, i.e. this keyword does not have to be present.

```
$current
| integer | optional |
| character | required |
| integer | required |
| double_array | optional |
| double_array | optional |
| double_array | optional |
| double_array | optional |
| double_array | optional |
| double_array | optional |
| double_array | optional |
$end_current
```
So far only the drift-diffusion model has been implemented but you never know what comes next.

**Explanation of specifiers**

- **model-number**
  - **type**: integer \( \geq 1 \)
  - **presence**: required
  - **value**: 1
  
  An integer number to refer to current model. Currently, only one model is implemented, so you have to choose 1.

- **transport-model-name**
  - **type**: character
  - **presence**: required
  - **value**: simple-drift-model
  
  This is the only option currently.

- **cluster-numbers**
  - **type**: integer array
  - **presence**: required
  - **value**: 1
  
  Currently, only one current cluster supported so far.

**Example**

```
!----------------------------------------------
$current-models

$current-models

model-number

<table>
<thead>
<tr>
<th>type</th>
<th>integer ( \geq 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>presence</td>
<td>required</td>
</tr>
<tr>
<td>value</td>
<td>1</td>
</tr>
</tbody>
</table>

transport-model-name

<table>
<thead>
<tr>
<th>type</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>presence</td>
<td>required</td>
</tr>
<tr>
<td>value</td>
<td>simple-drift-model</td>
</tr>
</tbody>
</table>

cluster-numbers

<table>
<thead>
<tr>
<th>type</th>
<th>integer array</th>
</tr>
</thead>
<tbody>
<tr>
<td>presence</td>
<td>required</td>
</tr>
<tr>
<td>value</td>
<td>1</td>
</tr>
</tbody>
</table>

!----------------------------------------------
```

**Impurity-parameters**

To specify the properties of impurities used in the simulation.
Specify here the properties of the impurities used for doping.

**impurity-number**

- **type** integer
- **presence** required
- **example** 1

1, 2, 3, … are unique impurity numbers labeled in $doping-function$.

Optionally, a name for the impurity can be provided.

**impurity-name**

- **type** character
- **presence** optional
- **example** n-As-in-Si
- **example** $0.010\cdot x+0.050\cdot (1-x)$ (unit: eV)
- **example** $0.010\cdot x+0.050\cdot (1-x)-0.020\cdot x\cdot (1-x)$ (unit: eV) Here, a bowing factor of 0.020 eV is included.

Specify an arbitrary name. The name will be part of some output files. (for later use; it is planned to read in impurity parameters from the database.)

Special feature: If **impurity-name** contains the character x, this string is then interpreted as an equation where x is the alloy content. This way, a position dependent impurity level can be simulated, e.g. for graded Al(x)Ga(1-x)N layers. In this example, the impurity level would be 10 meV for AlN (x=1) and 50 meV for GaN (x=0). Note that this has influence on CPU time, i.e. the calculation will take longer because the result of the equation has to be evaluated during the calculation.

(For later use: It is planned to read in impurity parameters from the database.)

Several types of impurities are supported.

**impurity-type**

- **type** character
- **presence** required
- **options** n-type, p-type, trap

Specifies the type of an impurity. n-type means that the impurity is treated as a donor, p-type as an acceptor. Option trap is not supported so far.

Each impurity can have several different energy levels.

**number-of-energy-levels**

- **type** integer
- **presence** required
- **example** 1
- **example** 1 2

Number of different energy levels for this impurity. Each energy level is specified in **energy-levels-relative**.

Energy levels

**energy-levels-relative**

- **type** double array
- **presence** required
- **unit** eV
A large negative value, e.g. -1000.0 eV implies full ionization.

Energy levels relative to “nearest” band edge (n-type -> conduction band, else valence band) in units of eV. As many energies as energy levels. These energies are meant as ionization energies, e.g. a donor with an energy level right below the conduction band edge would be specified by a small positive energy level.

When impurity levels are relatively deep compared to the thermal energy kBT at room temperature, incomplete ionization must be considered.

Note: ‘Cheat’ parameter: energy-levels-relative = -1000.0, for instance, that means, all electrons are fully ionized from the donors (similar for holes/acceptors). This might be useful for low temperatures like 4 K where usually the degree of ionization is very small. By using -1000.0 one can force them to be completely ionized.

The energy levels of the donors and acceptors relative to the lowest conduction band edge and highest valence band edge can be output using dopant-energy-levels = yes (see $output-densities$).

See also our tutorial on Doped semiconductors to learn more about partial ionization.

Degeneracy of the specified energy levels

degeneracy-of-energy-levels

type integer array
presence required
example 2
example 4 4

Shallow donors have degeneracy factor 2: Outer s orbital is onefold occupied (neutral state). There is one possibility to get rid of one electron but there are two to incorporate one (spin up, spin down).

Shallow acceptors have degeneracy factor 4: The sp³ orbital is threefold occupied. Thus, one possibility to incorporate an electron, four possibilities to get rid of one.

More details on degenerate impurity levels can be found in e.g. [ChuangOpto1995]. Note that in nitride semiconductors crystallizing in the wurtzite structure the degeneracy factor may vary from 4 to 6 because of a small valence band splitting.

If full ionization is assumed, i.e. energy-levels-relative = -1000.0, then the degeneracy factor effectively becomes irrelevant. This can be seen from eqs. (1.4) - (1.7) in [Birner-PhD2011].

transition-times-cb-to-levels

type double array
presence optional
unit ?

Transition times tau1, tau2, tau3, … from conduction band(s) to energy levels; required in case of trap: times from conduction band to discrete levels.

transition-times-levels-to-vb

type double array
presence optional
Transition times $\tau_1$, $\tau_2$, $\tau_3$, ... from energy levels to valence bands band(s); required in case of trap: times from discrete levels to valence bands.

Note: Currently no interlevel transition times implemented. Can be added provided there are also models which can handle such things.

Example

```plaintext
$impurity

! n-type in GaAs

<table>
<thead>
<tr>
<th>number</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>As</td>
</tr>
<tr>
<td>type</td>
<td>n</td>
</tr>
<tr>
<td>energy levels</td>
<td>1</td>
</tr>
<tr>
<td>energy relative</td>
<td>0.0058</td>
</tr>
<tr>
<td>levels</td>
<td>2</td>
</tr>
</tbody>
</table>

! n-type in GaAs (fully ionized)

<table>
<thead>
<tr>
<th>number</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>As</td>
</tr>
<tr>
<td>type</td>
<td>n</td>
</tr>
<tr>
<td>energy levels</td>
<td>1</td>
</tr>
<tr>
<td>energy relative</td>
<td>-1000.0</td>
</tr>
<tr>
<td>levels</td>
<td>2</td>
</tr>
</tbody>
</table>

! p-type in GaAs

<table>
<thead>
<tr>
<th>number</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>As</td>
</tr>
<tr>
<td>type</td>
<td>p</td>
</tr>
<tr>
<td>energy levels</td>
<td>1</td>
</tr>
<tr>
<td>energy relative</td>
<td>0.027</td>
</tr>
<tr>
<td>levels</td>
<td>4</td>
</tr>
</tbody>
</table>

! p-type in GaAs (fully ionized)

<table>
<thead>
<tr>
<th>number</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>As</td>
</tr>
<tr>
<td>type</td>
<td>p</td>
</tr>
<tr>
<td>energy levels</td>
<td>1</td>
</tr>
<tr>
<td>energy relative</td>
<td>-1000.0</td>
</tr>
<tr>
<td>levels</td>
<td>4</td>
</tr>
</tbody>
</table>

$end_impurity
```

Database values

```plaintext
relative -1000.0 ! energy in units of [eV]
-ionization 0.054 ! n-As-in-Si
            0.045 ! n-P -in-Si
```

(continues on next page)
\begin{verbatim}
0.039 ! n-Sb-in-Si
0.045 ! n-N -in-Si
0.006 ! n-Si-in-Al0.27Ga0.73As
0.0058 ! n-Si-in-GaAs
0.007 ! n-Si-in-AlAs
0.10 ! n-N -in-SiC
0.20 ! p-Al-in-SiC
0.045 ! p-B -in-Si
0.16 ! p-In-in-Si
0.027 ! p-C -in-GaAs
\end{verbatim}

More parameters can be found in the database file \texttt{database.nn3.in} or at this website: http://www.ioffe.ru/SVA/NSM/Semicond/

\textbf{$\$material$-interfaces$}

To specify additional charges at material interfaces, one has to specify

- material interfaces
- interface state properties.

See also documentation for keyword \texttt{Sinterface-states}.

\begin{verbatim}
$material
integer optional
integer_array required
integer_array required
$end_material

Explanation of specifiers

\textbf{interface-number}

\textbf{type} integer \\
\textbf{presence} required

An integer number to refer to interface number. Dense numbering (1, 2, \ldots) as usual.

\textbf{apply-between-material-numbers}

\textbf{type} integer \\
\textbf{presence} required

Two integer numbers to refer to interface between geometry clusters, i.e. it contains the \textit{material numbers} of the adjoining regions.

Is 3 4 equivalent to 4 3 in 1D? This becomes relevant for 2D and 3D simulations.

\textbf{state-numbers}

\textbf{type} integer \\
\textbf{presence} required

Identification numbers of interface states (e.g. fixed charge, trap or electrolyte) defined under keyword \texttt{Sinterface-states}.

\textbf{Note:} There can only be one unique \texttt{interface-number} for the the interface between \texttt{integer1} and \texttt{integer2}.
Here is an example which does not make sense (!):

```
interface-number  1
numbers           3 4
interface-number  2
numbers           3 4 ! '3 4' has already been assigned
                  ! in 'interface-number = 1'
```

$\textbf{magnetic-field}$

One can apply a magnetic field only for 2D and 3D simulations. For 1D simulations this is not possible.

```
$magnetic-field$  
character          optional 
double             required
integer_array      required
$magnetic-field$  
character          optional 
double             optional
integer            optional
$magnetic-field$  
character          optional 
integer_array      optional
$end_magnetic$     
```

Example

```
!-------------------------------------------------!
$magnetic-field$ !
$magnetic-field$-on  yes  ! yes/no
$magnetic-field$-strength  4.0  ! [T]
$magnetic-field$-direction  0 0 1  ! along z direction
potential           yes  ! yes/no
numbers             2 3  !
$end_magnetic$     
```

Here, a magnetic field of 4 T is applied.

- **magnetic-field-on**
  - type character
  - presence required
  - options yes or no
  - default no

Flag for switching magnetic field on or off.

- **magnetic-field-strength**
  - type double
  - unit [T]
  - example 2.5

The magnetic field strength refers to the magnetic flux density $B$ which has the SI unit $[T] = [Vs/m^2]$. Note that the SI unit for the magnetic field strength $H$ is $[A/m]$. It is possible to specify a negative value which inverts the vector specified here: magnetic-field-direction.
magnetic-field-direction

**type** integer array of dimension 3

**example** 0 0 1

The 3 (integer) indices \( x, y, z \) of a vector \( \mathbf{x} = (x, y, z) \) which is parallel to the magnetic field vector.

The direction of the magnetic field must be perpendicular to simulation orientation, e.g. if

```plaintext
$simulation$

dimension 2

orientation 0 1 1 ! simulation in (y,z) plane

$end_simulation$
```

then the magnetic field direction must be

```plaintext
magnetic-field-direction 1 0 0 ! along x direction
```

For a 3D simulation, the magnetic field can have any direction. The vector \( \mathbf{B} \) must be specified with respect to the \( xyz \) simulation system, i.e. you cannot specify the four-digit Miller-Bravais indices as in the case for wurtzite. It is with respect to the \( x, y \) and \( z \) coordinate axes that were specified under $domain-coordinates$: If not specified, default values specified in database_nn3.in are taken.

```plaintext
x y z := hkl-x-direction-zb, hkl-y-direction-zb, hkl-z-direction-zb

or

x y z := hkil-x-direction, hkil-y-direction, hkil-z-direction
```

(output, only needed for magnetic field sweep)

**output-magnetic-vector-potential**

**type** character

**options** yes or no

**default** no

The magnetic vector potential \( \mathbf{A}(x, y, z) \) is defined with respect to the symmetric gauge:

\[
\mathbf{A}(r) = -\frac{1}{2} (r \times \mathbf{B})
\]

**exclude-region-cluster-numbers**

**type** integer array

**example** 2 4

Here, cluster numbers 2 and 4 are exempted from magnetic field, i.e. \( B=0 \). This is useful for Aharonov–Bohm effect (not fully implemented yet).

**Magnetic field sweep**

It is possible to sweep over the magnetic field strength, i.e. to vary the strength of the magnetic field stepwise. This is similar to electric field sweeps ($selectric-field$), voltage sweeps ($voltage-sweep$) and doping concentration sweeps ($doping-function$).

**magnetic-field-sweep-active**

**type** character

**options** yes or no

**default** no

Specify yes if you want to sweep over several values of the magnetic field.
magnetic-field-sweep-size
  type double
  example 0.5
  example [T]
Here, the magnetic field increases in steps of 0.5 [T]. A negative value can be used to decrease the magnetic field.

magnetic-field-sweep-number-of-steps
  type integer
  example 10
Specify here the number of steps for the magnetic field sweep.
Example

```plaintext
$[magnetic field]
  on = yes ! yes/no
  strength = 0.0 ! [T]
  direction = 0 0 1 ! along z direction
  active = yes ! yes/no
  step-size = 0.5 ! [T]
  number-of-steps = 10 !
$end[magnetic field]
```

Here, the magnetic field increases from 0 [T] to 5 [T] in steps of 0.5 [T], i.e. 11 simulations are performed. The output is labeled with _ind000.dat, _ind001.dat, _ind002.dat,... where the index refers to the number of the magnetic field sweep step. The output for the eigenvalues as a function of applied magnetic field can be found here: magnetic_ev_vb1_qc1_sg1_deg1.dat.

In this particular example, the heavy hole valence band edge energies (vb1) have been written out as a function of magnetic field. The first column contains the strength of the magnetic field in units of [T]. The second column contains the 1\textsuperscript{st} eigenvalue for the specified electric field in units of [eV], the third column contains the 2\textsuperscript{nd} eigenvalue for the specified electric field in units of [eV],...

Note: The magnetic field is only implemented for the single-band effective mass model but not for the multi-band \( \mathbf{k} \cdot \mathbf{p} \) model. For the combination of \( \mathbf{k} \cdot \mathbf{p} \) and magnetic field, please use the nextnano++ software.

### $Auger-recombination

Any values specified here in the input file overwrite the values specified in the material database.

Please refer to the database section of $Auger-recombination for more details on the meaning of these parameters.
**$direct-recombination**

Any values specified here in the input file overwrite the values specified in the material database.

Please refer to the database section of $direct-recombination for more details on the meaning of these parameters.

```plaintext
$direct=[[4]]
```

**$SRH-recombination**

Any values specified here in the input file overwrite the values specified in the material database.

Please refer to the database section of $SRH-recombination for more details on the meaning of these parameters.

```plaintext
$SRH=[[4]]
```

**$mobility-model-constant**

See also explanations under section database ==gt $mobility-model-constant.

The only difference is that number-of-parameters is optional instead of required.

```plaintext
$mobility=[[4]]
```
$\text{mobility-model-minimos}$

See also explanations under section database $\Rightarrow$ $\text{mobility-model-minimos}$. The only difference is that number-of-parameters is optional instead of required.

```
$\text{mobility-model-minimos}$

<table>
<thead>
<tr>
<th>$\text{mobility}$</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{integer}$</td>
<td>optional</td>
</tr>
</tbody>
</table>

$\text{mobility-model-simba}$

See also explanations under section database $\Rightarrow$ $\text{mobility-model-simba}$. The only difference is that number-of-parameters is optional instead of required.

```
$\text{mobility-model-simba}$

<table>
<thead>
<tr>
<th>$\text{mobility}$</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{integer}$</td>
<td>optional</td>
</tr>
</tbody>
</table>

$\text{mobility-model-arora}$

See also explanations under section database $\Rightarrow$ $\text{mobility-model-arora}$. The only difference is that number-of-parameters is optional instead of required.

```
$\text{mobility-model-arora}$

<table>
<thead>
<tr>
<th>$\text{mobility}$</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{integer}$</td>
<td>optional</td>
</tr>
</tbody>
</table>

$\text{mobility-model-dar}$

See also explanations under section database $\Rightarrow$ $\text{mobility-model-dar}$. The only difference is that number-of-parameters is optional instead of required.

```
$\text{mobility-model-dar}$

<table>
<thead>
<tr>
<th>$\text{mobility}$</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{integer}$</td>
<td>optional</td>
</tr>
</tbody>
</table>
$mobility-model-lom

See also explanations under section database ==> $mobility-model-lom.

The only difference is that number-of-parameters is optional instead of required.

```
$mobility-model-lom
optional
integer optional
$end_mobility
```

$mobility-model-masetti

See also explanations under section database ==> $mobility-model-masetti.

The only difference is that number-of-parameters is optional instead of required.

```
$mobility-model-masetti
optional
integer optional
$end_mobility
```

$optical-absorption

- Optical absorption
- Solar cells
  - Solar spectrum
  - Absorption coefficient
  - Reflection coefficient
  - Transmission coefficient
  - Solar cell output
- Black body spectrum
- Intraband absorption in the single-band case

This keyword allows to calculate optical absorption and solar cells.

```
$optical-absorption
character required
character optional
character optional
character optional
integer optional
double optional
double optional
double optional
integer optional
character optional
double optional
```

(continues on next page)
This flag can be set to `yes` for Step 1 and Step 2, and `no` for Step 3 (see below).

### Optical absorption

The following are only relevant for solar cell simulations.

#### destination-directory

- **type**: character  
  - **presence**: required
  - **example**: `optics/`

Directory for output of data files.

### Optical absorption

- **calculate-optics**
  - **type**: character
  - **options**: `yes` or `no`
  - **default**: `no`

Choose `yes` if you want to calculate the optical absorption (Step 3).

A tutorial is available that describes this keyword: Optical absorption of an InGaAs quantum well

**Number of quantum cluster for which absorption is calculated. If this specifier is not present, the quantum cluster 1 is taken.**

**kind-of-absorption**

**type** character

**options** interband-only intra-vb-only intra-cb-only intra-sg-only

- interband-only
  Considers only interband transitions between holes and electrons.
  - heavy hole \(\leftrightarrow\) Gamma band
  - light hole \(\leftrightarrow\) Gamma band
  - split-off hole \(\leftrightarrow\) Gamma band

- intra-vb-only
  Considers only intraband transitions within the valence bands.
  - heavy hole \(\leftrightarrow\) light hole
  - heavy hole \(\leftrightarrow\) split-off hole
  - light hole \(\leftrightarrow\) split-off hole
  - heavy hole \(\leftrightarrow\) heavy hole
  - light hole \(\leftrightarrow\) light hole
  - split-off hole \(\leftrightarrow\) split-off hole

- intra-cb-only
  Considers only intraband transitions within the conduction band (Gamma band).
  - Gamma band \(\leftrightarrow\) Gamma band

- intra-sg-only
  Considers only intraband transitions within the same band (single-band for Gamma, L, X, heavy hole, light hole, split-off hole band)
  - Gamma band \(\leftrightarrow\) Gamma band
  - L band \(\leftrightarrow\) L band
  - X band \(\leftrightarrow\) X band
  - heavy hole \(\leftrightarrow\) heavy hole
  - light hole \(\leftrightarrow\) light hole
  - split-off hole \(\leftrightarrow\) split-off hole

This is a simple algorithm taking only account the energy levels and wave functions at \(k_\parallel = 0\) (for single-band case). It only works for 1D and 2D simulations so far. It can also be used for the \(k \cdot p\) wavefunctions as shown in this tutorial: *Intersubband transitions in InGaAs/AlInAs multiple quantum well systems*. In this case, the correct \(k \cdot p\) density and \(k\)-dependent matrix elements and nonparabolicity and anisotropy of the energy dispersion \(E(k_x, k_y)\) is taken into account.

In order for this flag to work, the following must be present in the input file (`$output-1-band-schroedinger`):

```
$output -1 -b a n d -s c h r o e d i n g e r . . .
matrixelements yes ! or any other value apart from 'no'
```
The equation used is described here: *Intraband absorption in the single-band case*

- **inter-sg-only**
  Similar as **intra-sg-only** but for interband transitions. It currently does not work for \( k \cdot p \) wavefunctions.

Specifications for absorption energy range: lower/upper boundary for photon energy interval

- **e-min-photon**
  - type double
  - unit [eV]
  - example 1.0
  lower boundary for photon energy

- **e-max-photon**
  - type double
  - unit [eV]
  - example 2.0
  upper boundary for photon energy

- **num-energy-steps**
  - type integer
  - example 1000
  Number of energy steps between \( e_{\text{min-photon}} \) and \( e_{\text{max-photon}} \).
  This number determines the resolution of the absorption curve \( \alpha(E) \) where \( E \) is the energy in units of \([eV]\).

**Note:** The number of energy grid points = num-energy-steps + 1
because the first grid point (e-min-photon) is also included.

The energy grid spacing is

\[
\Delta E = (e_{\text{max-photon}} - e_{\text{min-photon}}) / \text{num-energy-steps}.
\]

Distinguish between calculating and reading in \( k_\parallel \) points.

- **read-in-k-points**
  - type character
  - options yes or no
  - default no

Flag to distinguish between Step 2 and Step 3. For Step 3, in order to avoid calculating the \( k_\parallel \) points again, one can simply read them in from a previous simulation, and then calculate and output the optical absorption.

Energy of lowest/highest eigenvalue considered for calculation, i.e. energies are calculated in the interval \([e_{\text{min-state}}, e_{\text{max-state}}]\). Here, the eigenvalue solver is called with these energy values. Alternatively, the eigenvalue solver can be called with a certain number of eigenvalues requested, i.e. one either has to specify an energy interval or the number of eigenvalues. Depending on the eigenvalue solver used, different options are possible.

- **e-min-state**
  - type double
  - unit [eV]
lowest eigenvalue

**e-max-state**

type double

unit [eV]

default 5.0

eample 0.3

highest eigenvalue

**Broadening of absorption curve**

This is only relevant for the calculation of the absorption curve.

**smoothing-of-curve**

type character

options Lorentzian, Gaussian, yes, no

default yes

For Lorentzian or Gaussian, we introduce an artificial broadening (smoothing) of the curve (Lorentzian or Gaussian broadening). If yes, both Lorentzian and Gaussian broadening will be calculated and written out. If no, no broadening (smoothing) of the curve is assumed.

**smoothing-damping-parameter**

type double > 0.0

unit [eV]

default 0.005

The artificial parameter for smoothing of absorption curve is smoothing-damping-parameter. It is usually denoted as \( \Gamma \) and is the Full Width at Half Maximum (FWHM).

**Lorentzian lineshape**

The Lorentzian function is given by

\[
L(E) = \frac{1}{\pi} \frac{\Gamma/2}{(E - E_{ij})^2 + (\Gamma/2)^2}
\]

where \( \Gamma/2 \) is the scale parameter Lorentzian half-width, i.e. Half Width at Half Maximum (HWHM). It describes the shape of certain types of spectral lines (lineshape). Note that the definition of the Lorentzian function includes a factor \( 1/\pi \).

- \( E_{ij} \) is the transition energy between the states \( i \) and \( j \) and specifies the location of the peak in the Lorentzian function.
- \( \Gamma \) is specified in the input file via smoothing-damping-parameter. It is the Full Width at Half Maximum (FWHM).
- \( \alpha = \Gamma/2 \) is the Half Width at Half Maximum (HWHM).

```
!--------------------------------------------
! for k.p algorithm only:
! First, the absorption curve is calculated.
! Then the broadening is applied.
!--------------------------------------------

absorption_NoSmoothingV = absorptionV
absorptionV = 0.0
```

(continues on next page)
DO  i=1,num-energy-steps+1 ! Loop over all energy grid points E(i)...
       and determine absorption alpha(i)=alpha(E).
DO  j=1,num-energy-steps+1 ! This loop is essentially an integration...
       over energy dE.
E_weight = (Lorentzian( E_gridV(j),  E_gridV(i), smoothing-
       →damping-parameter ) - 6
           Lorentzian( E_gridV(j), - E_gridV(i), smoothing-
       →damping-parameter ) ) * DeltaEnergy
absorptionV(i) = absorptionV(i) + absorption_NoSmoothingV(j) * E_
       →weight
END DO
END DO

The following specifiers are only used for the k · p optical absorption algorithm but not for the simple single-band intersubband transition optical absorption algorithm.

**E_P**

<table>
<thead>
<tr>
<th>type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>example</td>
<td>20.0</td>
</tr>
<tr>
<td>unit</td>
<td>[eV]</td>
</tr>
<tr>
<td>status</td>
<td>currently not implemented, value from the database is used</td>
</tr>
</tbody>
</table>

$E_P$ is Kane’s matrix element $E_P = | < S | p | X > |^2$. It should be around 20 eV and depends on the material. The $E_P$ parameter is given in the database by the specifier 8x8kp-parameters. $E_P$ can be converted into the $P$ parameter by the following equation: $E_P = \frac{2m_0}{\hbar^2} P^2$. In our model the $E_P$ parameter is only relevant for interband transitions. It enters into the matrix element prefactor (matrix_element_prefac) which is described in Section 1.1.1 Inter-band transitions of the documentation: Absorption, Matrix elements, Inter-band transitions, Intra-band transitions (pdf). $E_P$ has the same value for all materials in this implementation. In principle it could have been read in from the database rather than specifying it within the keyword `$optical-absorption`.

**Polarization**

$m$ is equivalent to $\tan(\theta)$ or $\tan^{-1}(\theta)$ depending whether $P_1/P_2$ is $x$ or $y$. so instead of $\theta$ one can directly use the formula below:

$$P = m \cdot P_1 + e^{i\omega \pi} \cdot P_2$$

This is in fact more general, as it also describes circular polarizations which lead to complex coefficient.

**polarization-vector-1**

<table>
<thead>
<tr>
<th>type</th>
<th>double array</th>
</tr>
</thead>
<tbody>
<tr>
<td>example</td>
<td>1.0 0.0 0.0</td>
</tr>
</tbody>
</table>

$x$ $y$ $z$ coordinates (in simulation system) for first in-plane vector $P_1$

**polarization-vector-2**

<table>
<thead>
<tr>
<th>type</th>
<th>double array</th>
</tr>
</thead>
<tbody>
<tr>
<td>example</td>
<td>0.0 1.0 0.0</td>
</tr>
</tbody>
</table>

$x$ $y$ $z$ coordinates (in simulation system) for second in-plane vector $P_2$

**magnitude-relation-1-2**

<table>
<thead>
<tr>
<th>type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>[]</td>
</tr>
<tr>
<td>default</td>
<td>0.5</td>
</tr>
</tbody>
</table>
example 1.0

relation of magnitudes \( m = |E_1|/|E_2| \)

phase

<table>
<thead>
<tr>
<th>type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>[ ]</td>
</tr>
<tr>
<td>default</td>
<td>0.0</td>
</tr>
<tr>
<td>example</td>
<td>0.5</td>
</tr>
</tbody>
</table>

phase \( \omega: E_2 \Rightarrow \exp(i\omega \pi)E_2 \)

\[ P = m \cdot P_1 + e^{i\omega \pi} \cdot P_2 \]

Examples

- x-polarized light

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In this case, polarization-vector-1 is ignored as \( |E_1| \) is set to be zero.

- z-polarized light

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In this case, polarization-vector-1 is ignored as \( |E_1| \) is set to be zero.

- circularly polarized light in the (x,y) plane

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In this case, polarization-vector-1 is not ignored as \( |E_1| = |E_2| \).

- quantum well, interband absorption

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- quantum well, intraband absorption

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: Intraband absorption only for z-polarized light.

Fermi levels

<table>
<thead>
<tr>
<th>fermi_in_el</th>
</tr>
</thead>
<tbody>
<tr>
<td>type double</td>
</tr>
<tr>
<td>unit [eV]</td>
</tr>
<tr>
<td>default 0.0</td>
</tr>
<tr>
<td>example 0.1</td>
</tr>
</tbody>
</table>

Fermi levels
Optional input for Fermi level of electrons (default: calculated quasi-Fermi level for electrons)

\[
\text{fermi}_\text{in}_\text{hl}
\]

- **type**: double
- **unit**: [eV]
- **default**: 0.0
- **example**: -1.0

Optional input for Fermi level of holes (default: calculated quasi-Fermi level for holes)

\[
\text{device-thickness}
\]

- **type**: double
- **unit**: [m]
- **default**: thickness of device or quantum cluster? (Check this!)
- **example**: 1e-6

Optional input of device thickness for normalization of absorption

\[
\text{k-space-symmetry}
\]

- **type**: character
- **options**: default, none, four-fold
- **default**: default

Symmetry of \( k_\parallel = (k_x, k_y) \) space to be discretized. If any symmetry is present, less \( k_\parallel \) points have to be calculated. By default, the appropriate symmetry is chosen taking into account any crystal rotations with respect to the simulation axes, as well as nonsymmetric strains.

**Note:** In order to save CPU time, we recommend the following procedure:

Instead of calculating

- \( k \cdot p \) eigenstates and
- optical absorption

within one simulation, it is more efficient to divide this into 3 steps.

1. **Step 1:** Calculate eigenstates for \( k_\parallel = 0 \).

   - \[ \boxed{\text{optics} \ no} \]

   This is very quick. Solve \( k \cdot p \) to determine lowest and highest eigenvalue so that we know what to specify for \( \text{e-min-state} \) and \( \text{e-max-state} \) in Step 2.

2. **Step 2:** Calculate eigenstates for all \( k_\parallel \) vectors and save all of the wavefunctions to file so that they can be read in and used many times in Step 3.

   - \[ \boxed{\text{strain}\_\text{calculation} \ yes} \]
   - \[ \boxed{\text{parallel} \ 1700} \]
   - \[ \boxed{\text{Brillouin zone discretization}} \]

   \[ \boxed{\text{optics} \ yes} \]
   - \[ \boxed{\text{quantum}\_\text{cluster} \ 1} \]
   - \[ \boxed{\text{red}\_\text{in}-\text{k}\_\text{points} \ no} \]
   - \[ \boxed{\text{e-min-state} \ -1.7} \]
   - \[ \boxed{\text{e-max-state} \ 0.3} \]

9.2. Input File
Read in raw data (potential, quasi-Fermi levels, strain if applicable) and all \( k \cdot p \) wavefunctions and output \( k || \) points. The strain calculation is very fast for a 1D simulation. Instead of reading it in, it could be directly calculated (recommended).

The user specifies the total number of \( k || \) points that are present in \( k || \) space. However, internally the code modifies this number according to the following algorithm:

- number of \( k \) points in positive x direction (without Gamma point): \( N_{kx} \)
- number of \( k \) points in positive y direction (without Gamma point): \( N_{ky} = N_{kx} \)

\[
\text{total_number_of_k||} = (2 \times N_{kx} + 1) \times (2 \times N_{ky} + 1)
\]

In this example (\( \text{num-kp-parallel} = 1700 \)):

\[
N_{kx} = N_{ky} = 20
\]

\[
\Rightarrow \text{total_number_of_k||} = 41 \times 41 = 1681
\]


Read in \( k \) points, calculate and output optical absorption for specific polarization of incident light.

If one wants to repeat the calculation for another polarization, one only needs to change the polarization vector and repeat Step 3. It is not necessary in this case to recalculate Step 1 or Step 2. Step 3 also outputs the energy dispersion \( E(k ||) = E(k_x, k_y) \). (Check: Why not Step 2?)

Output Results

Figure 9.2.3.1: Conduction and valence band edges of the quantum well

The units of the optical absorption are \([\text{m}^{-1}]\) and not arbitrary units as indicated in the figure.

The electric susceptibility tensor \( \chi \) is contained in the file `susceptibility_tensor.dat`.
Figure 9.2.3.2: Electron states in the quantum well

Figure 9.2.3.3: Hole states in the quantum well
Note: As this tensor is complex, for each component, two values are written out.

- re: real part
- im: imaginary part

The relevant part for the absorption is only the imaginary part.

The units of the $k_{\parallel}$ space grid coordinates $k_x$ and $k_y$ are \([\text{Å}^{-1}]\) and the energy units are \([\text{eV}]\).

The files
- el\_dispersion\_100.dat
- el\_dispersion\_110.dat
- hl\_dispersion\_100.dat
- hl\_dispersion\_110.dat

show the same data but with slices along the

- [10] (i.e. $k_{\parallel} = (k_x, k_y = 0)$ and
- [11] (i.e. $k_{\parallel} = (k_x = k_y)$ directions in $k_{\parallel}$ space.

Here all electron and all hole eigenvalues are contained in one file, respectively.

Restrictions

- Only Dirichlet boundary conditions are supported so far.
- Step 2 and Step 3 only work if:

```plaintext
  [10] in  yes
```
Figure 9.2.3.5: Energy dispersion $E(k_x, k_y)$ of the highest hole eigenstate (ground state).
Solar cells

For solar cells, we have this tutorial: GaAs Solar Cell

Example files for solar spectra, absorption coefficient, transmission and reflectivity coefficients can be found in the installation folder:

- C:\Program Files\nextnano\nextnano3\Syntax\Solar cell files\absorption\n- C:\Program Files\nextnano\nextnano3\Syntax\Solar cell files\reflectivity
- C:\Program Files\nextnano\nextnano3\Syntax\Solar cell files\solar spectra\

The following specifiers are relevant for solar cell simulations (photovoltaics).

incident-light-along-direction

type character
options x, y, z, -x, -y, -z

default along simulation direction in 1D

In a 1D simulation, this specifier is optional. For 2D and 3D, a direction must be specified.

Solar spectrum

import-solar-spectrum

type character
options yes or no

default no

For a solar cell simulation, one has to read in a solar spectrum, e.g. AM 1.5, or AM 1.0 (AM = air mass). They can be obtained from NREL website, e.g. ASTM-E490: https://www.nrel.gov/grid/solar-resource/spectra-astm-e490.html (AMST = American Society for Testing and Materials)

file-solar-spectrum

type character
example H:\solar_cells\ASTMG173_AM10.dat AM 1.0 spectrum (extraterrestrial)
example H:\solar_cells\ASTMG173_AM15.dat AM 1.5 spectrum
example H:\solar_cells\ASTMG173_AM15G.dat AM 1.5G spectrum (G = global, i.e. including diffuse light)

The file must consist of two columns (wavelength and spectrum), the units are [nm] and [W/m^2*nm^-1].

<table>
<thead>
<tr>
<th>wavelength[nm]</th>
<th>AM1.5[W/m^2*nm^-1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Concentration of sun light

number-of-suns

type integer

default 1.0 our sun
example 0.0 (no sun, dark)
example 2.5 2.5 suns
The number of suns can be set to increase the power of the solar spectrum in order to model concentrator solar cells.

**Absorption coefficient**

```
import-absorption-spectrum
type character
options yes or no
default no
```

The file must consist of two columns (wavelength and absorption coefficient), the units are \([\text{nm}]\) and \([\text{cm}^{-1}]\).

<table>
<thead>
<tr>
<th>wavelength [nm]</th>
<th>absorption [1/cm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Reflection coefficient**

Fraction of incident photons that are reflected from surface for a particular wavelength.

```
import-reflectivity-spectrum
type character
options yes or no
default no
```

The file must consist of two columns (wavelength and reflection coefficient), the units are \([\text{nm}]\) and [ ].

<table>
<thead>
<tr>
<th>wavelength [nm]</th>
<th>reflectivity [ ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Transmission coefficient**

Fraction of incident photons that are transmitted through the device for a particular wavelength (relevant for very thin devices).

```
import-transmission-spectrum
type character
options yes or no
default no
```

9.2. Input File
The file must consist of two columns (wavelength and transmission coefficient), the units are [nm] and [ ].

```
<table>
<thead>
<tr>
<th>wavelength[nm]</th>
<th>transmission[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

**Solar cell output**

All output is twofold:

- one is with respect to wavelength in units of [nm]
- one is with respect to photon energy in units of [eV] (indicated by _eV*.dat)

The files are:

- Absorption coefficient
  - optics/Absorption_coefficient.dat (as read in from file but now in units of [m^-1])
  - optics/Absorption_coefficient_interpolated.dat (interpolated on wavelength grid of solar spectrum but now in units of [m^-1])
- Reflectivity
  - optics/Reflectivity.dat (as read in from file)
  - optics/Reflectivity_interpolated.dat (interpolated on wavelength grid of solar spectrum)
- Transmission
  - optics/Transmission.dat (as read in from file)
  - optics/Transmission_interpolated.dat (interpolated on wavelength grid of solar spectrum)
- Solar spectrum
  - optics/SolarSpectralIrradiance_sun0001.dat (as read in from file)
  - optics/PhotonFlux_sun0001.dat (photon flux density calculated from solar spectrum)
- Total number of of photons in the solar spectrum above an energy value contributing to the maximum photocurrent for a solar cell made with a specific band gap:
  - optics/PhotonFlux_BandGap_eV_sun0001.dat (calculated from solar spectrum)
  - optics/PhotoCurrent_BandGap_eV_sun0001.dat (calculated from solar spectrum)
- Spectral response
  - optics/SpectralResponse_sun0001.dat external and internal spectral response
- Quantum efficiency
  - optics/QuantumEfficiency_sun0001.dat external and internal quantum efficiency
- Generation rate
  - optics/GenerationRateLight_AVS_sun0001.fld 2D plots $G(x, \lambda)$ and $G(x, E)$
  - optics/GenerationRateLight_sun0001.dat 1D plot $G(x)$
  - optics/GenerationRate_eV_sun0001.dat 1D plot $G(E)$ where $E$ is the energy
  - optics/GenerationRate_Wavelength_sun0001.dat 1D plot $G(\lambda)$
- Current-voltage characteristics
- current/IV_characteristics_new.dat


- The following information can be found in the .log file, such as
  - short-circuit current $I_{sc}$
  - open-circuit voltage $U_{oc}$
  - ideal conversion efficiency $\eta$
  - ...

Solar cell results

| short-circuit current: $I_{sc}$ = 281.473346 [A/m^2] |
| open-circuit voltage: $U_{oc}$ = -1.012500 [V] |

All these results are approximations.

In practice, a good fill factor is around 0.8.

They are only correct if a lot of voltage steps have been used (i.e. a high resolution).

Example for a solar cell simulation

````
$optical

spectrums

suns = 1

$end_optical
```

9.2. Input File 303
Black body spectrum

**calculate-black-body-spectrum**

*type* character

*options* yes or no

*default* no

Flag for calculating *black body spectrum* according to Planck’s law, e.g. to compare the solar spectrum to the spectrum of a black body at $T = 5778$ K.

- The spectral energy density
- the spectral *radiance* (which is emitted per m$^2$ and per unit solid angle sr ($sr = \text{steradian}$)) and
- the spectral *irradiance* (which is received per m$^2$)

is calculated.

**Note:** spectral irradiance = spectral radiance $\cdot \pi$

spectral energy density = spectral radiance $\cdot 4\pi/c$

There are several output files, i.e. output with respect to

- **wavelength** $\lambda$ in units of [m],

  - BlackBody_SpectralEnergyDensity_wavelength.dat
    
    | Wavelength[nm] | SpectralEnergyDensity[kJ/m$^3$/m] |
  
  - BlackBody_SpectralRadiance_wavelength.dat
    
    | Wavelength[nm] | SpectralRadiance[kW/m$^2$/nm/sr] |
  
  - BlackBody_SpectralIrradiance_wavelength.dat
    
    | Wavelength[nm] | SpectralIrradiance[kW/m$^2$/nm] |

- **angular frequency** $\omega = 2\pi\nu$ in units of [1/s],

  - BlackBody_SpectralEnergyDensity_angular_frequency.dat
    
    | AngularFrequency_omega[10$^{15}$/s] | SpectralEnergyDensity[10$^{-15}$J/m$^3$/s$^{-1}$] |
  
  - BlackBody_SpectralRadiance_angular_frequency.dat
    
    | AngularFrequency_omega[10$^{15}$/s] | SpectralRadiance[10$^{-12}$W/m$^2$/s$^{-1}$/sr] |
  
  - BlackBody_SpectralIrradiance_angular_frequency.dat
    
    | AngularFrequency_omega[10$^{15}$/s] | SpectralIrradiance[10$^{-12}$W/m$^2$/s$^{-1}$] |

- **frequency** $\nu$ in units of [Hz],

  - BlackBody_SpectralEnergyDensity_frequency.dat
    
    | Frequency[THz] | SpectralEnergyDensity[10$^{-15}$J/m$^3$/Hz] |
  
  - BlackBody_SpectralRadiance_frequency.dat
Frequency[THz] SpectralRadiance[10^-12W/m^2/sr]

- BlackBody_SpectralIrradiance_frequency.dat

Frequency[THz] SpectralIrradiance[10^-12W/m^2/Hz]

- BlackBody_SpectralEnergyDensity_energy.dat

AngularFrequency_omega[10^15/s] SpectralEnergyDensity[kJ/m^3/eV]

- BlackBody_SpectralRadiance_energy.dat

AngularFrequency_omega[10^15/s] SpectralRadiance[kW/m^2/eV/sr]

- BlackBody_SpectralIrradiance_energy.dat

AngularFrequency_omega[10^15/s] SpectralIrradiance[kW/m^2/eV]

The file BlackBody_Info.txt contains some additional information about the calculated black body spectrum.

**Intraband absorption in the single-band case**

In the following we assume a single band with a parabolic energy band dispersion.

Tutorials showing results are available here:

- Intersubband absorption of an infinite quantum well
- Intersubband transitions in InGaAs/AlInAs multiple quantum well systems

For a 1D heterostructure grown along the \(x\) direction, the absorption coefficient \(\alpha\) reads (see e.g. [ChuangOpto1995] or p.53 in [FaistQCL2013])

\[
\alpha(\omega) = \frac{e^2 \omega}{\varepsilon_0 c n_r c} \sum_i \sum_j (\pi_i - \pi_j) x_{ij}^2 \frac{\Gamma/2}{(E_j - E_i - \hbar \omega)^2 + (\Gamma/2)^2}
\]

or, equivalently in energy,

\[
\alpha(E) = \frac{e^2 E}{\varepsilon_0 c n_r c} \sum_i \sum_j (\pi_i - \pi_j) x_{ij}^2 \frac{\Gamma/2}{(E_j - E_i - \hbar \omega)^2 + (\Gamma/2)^2}
\]

where

- \(\omega = E / \hbar\) is the frequency in units of \([s^{-1}]\)
- \(E\) the energy in units of \([J]\)
- \(\varepsilon_0\) the elementary charge in units of \([A As]\)
- \(\varepsilon_0\) is the vacuum permittivity in units of \([A As/V m]\)
- \(c\) is speed of light in vacuum in units of \([m/s]\)
- \(n_r = \sqrt{\varepsilon_r}\) is the refractive index ([]) assumed to be homogeneous. So we take the average of the quantum region (check this).
- \(\pi_i = \frac{1}{L} \sigma_i = \frac{1}{L} \int n_i(x) dx\) is the averaged electron density of subband \(i\) in units of \([m^{-3}]\), where \(L\) is the length of the quantum region and \(\sigma_i = \int n_i(x) dx\) is the subband density in units of \([m^{-2}]\)
- \(x_{ij} = <i|\sigma|j>\) is the dipole moment between initial state \(i\) and final state \(j\) in units of \([m]\)
- \(\Gamma\) is the energy linewidth (broadening) in units of \([J]\) in terms of full-width at half maximum (FWHM).

This equation includes a Lorentzian broadening which includes a factor of \(1/\pi\).

We can also define the position dependent absorption coefficient

9.2. Input File 305
\[ \alpha(\omega, x) = \frac{\epsilon_0 \epsilon_r}{\epsilon_0 \epsilon_r + \mu_0 \sigma} \sum_i \sum_j (n_i(x) - n_j(x)) x_i x_j \frac{G}{2} \left( \frac{1}{x_i^2} - \frac{1}{x_j^2} \right) \]

where

- \( n_i(x) \) is the electron density of state \( i \) at position \( x \) in units of \( [m^{-3}] \).

The units of both \( \alpha(\omega) \) and \( \alpha(\omega, x) \) are \( [m^{-1}] \). In plots, typically \( [cm^{-1}] \) is used.

If we integrate \( \alpha(\omega, x) \) over position \( x \) in the whole quantum region of length \( L \), and divide by the length of the quantum region \( L \), we obtain \( \alpha(\omega) \) as defined above,

\[ \alpha(\omega) = \frac{1}{L} \int \alpha(\omega, x) dx. \]

So \( \alpha(\omega) \) as defined in the beginning of this section, where we averaged the density \( \pi_i \), is the averaged absorption in the quantum region and equivalent to the definition given here.

Finally, we note that this also works for the \( k \cdot p \) wavefunctions:

\[ \alpha(\omega, x, k) = \frac{\epsilon_0 \epsilon_r}{\epsilon_0 \epsilon_r + \mu_0 \sigma} \sum_i \sum_j (n_i(x, k) - n_j(x, k(x))) x_{ij}(k(x)) \frac{G}{2} \left( \frac{1}{x_{ij}(k(x))} - \frac{1}{x_{ij}(k(x))} \right) \]

where

- \( n_i(x, k) \) is the electron density of state \( i \) at position \( x \) and vector \( k(x) = (k_x, k_y) \) (in 1D) or \( k(x) = k_z \) (in 2D).

### $buffer-solutions$

Buffer solutions: To control pH values in electrolytes.

It is possible to overwrite the database entries of $buffer-solutions by this keyword and its specifiers in the input file. You can also define a completely new buffer which is not contained in the database. If you overwrite a buffer that is contained in the database, it is necessary that you include all specifiers, including the ones that you do not want to overwrite. The program then uses all the values given in the input file and ignores all database entries.

<table>
<thead>
<tr>
<th>$buffer$</th>
<th>character</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$buffer$</td>
<td>integer</td>
<td>required</td>
</tr>
<tr>
<td>$buffer$</td>
<td>double_array</td>
<td>required</td>
</tr>
<tr>
<td>$buffer$</td>
<td>character</td>
<td>required</td>
</tr>
<tr>
<td>$buffer$</td>
<td>character</td>
<td>required</td>
</tr>
<tr>
<td>$buffer$</td>
<td>character</td>
<td>optional ! only necessary for some$buffer$</td>
</tr>
<tr>
<td>$buffer$</td>
<td>character</td>
<td>optional ! only necessary for some$buffer$</td>
</tr>
<tr>
<td>$buffer$</td>
<td>double_array</td>
<td>required ! pKa at 25° C ( = 298.15 K)</td>
</tr>
<tr>
<td>$buffer$</td>
<td>double_array</td>
<td>required ! d pKa / d T</td>
</tr>
<tr>
<td>$buffer$</td>
<td>double_array</td>
<td>required ! charge on the conjugate</td>
</tr>
<tr>
<td>$end_buffer$</td>
<td></td>
<td>optional</td>
</tr>
</tbody>
</table>

\( pK_a \) and \( d pK_a / d T \).

**Example 1: MOPS**

```plaintext
$buffer
buffer-name: MOPS ! MOPS (C7H15NO4S) + NaOH
valency: +1.0 ! (C7H14NO4S)^- Na^+

1 1.0 ! C7H15NO4S <-> (C7H14NO4S)^- + H^+

pKa: 7.31 ! pKa at 25° C ( = 298.15 K)
! Note: This pKa is thermodynamic value.
! The working pKa' is 7.20.

dpKa_dT: -0.011 ! d pKa / d T

$end_buffer
```

(continues on next page)
Involved ions and molecules:

- **buffer-name:** MOPS (C$_7$H$_{15}$NO$_4$S) + NaOH
- **ion-valency:** (C$_7$H$_{14}$NO$_4$S)$^-$ and Na$^+$
- **ion-name-1:** C$_7$H$_{15}$NO$_4$S $\rightleftharpoons$ (C$_7$H$_{14}$NO$_4$S)$^- +$ H$^+$
- **ion-name-2:** Na$^+$
- **z$_{\text{acid}}$:** 0 = C$_7$H$_{15}$NO$_4$S

**Example 2: PBS (phosphate buffer)**

The phosphate buffer is special (and thus more complicated) because it consists of three $pK_a$ values (and it thus has four different ions).

Involved ions and molecules:

- **ion-valency:** (H$_2$PO$_4$)$_^-$, (HPO$_4$)$_{2^-}$, (PO$_4$)$_{3^-}$ and Na$^+$
- **ion-name-1:** NaH$_2$PO$_4$ $\rightleftharpoons$ (H$_2$PO$_4$)$_^-$ + Na$^+$
- **ion-name-2:** Na$_2$HPO$_4$ $\rightleftharpoons$ (HPO$_4$)$_{2^-} + 2$ Na$^+$
- **ion-name-3:** (PO$_4$)$_{3^-}$
- **ion-name-4:** Na$^+$
- **z$_{\text{acid}}$:** 0 = H$_3$PO$_4$, -1 = (H$_2$PO$_4$)$_^-$, -2 = (HPO$_4$)$_{2^-}$

For the explanation of the specifiers, please also check the description in the database of the keyword $buffer$-solutions.
$buffer$-constant-$A(T)$

Constant $A(T)$ used for buffer calculations: The $pK_a$ value depends on the ionic strength.

Example 1

![---](continues on next page)
$atomic-layers$ is only necessary for the heterostructure tight-binding code but not for the bulk tight-binding code.

<table>
<thead>
<tr>
<th>Character</th>
<th>Optional !</th>
<th>Character</th>
<th>Required !</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>Optional !</td>
<td>Character</td>
<td>Optional !</td>
</tr>
<tr>
<td>Character</td>
<td>Optional !</td>
<td>Character</td>
<td>Optional !</td>
</tr>
<tr>
<td>Character</td>
<td>Optional !</td>
<td>Character</td>
<td>Optional !</td>
</tr>
<tr>
<td>Character</td>
<td>Optional !</td>
<td>Integer</td>
<td>Optional !</td>
</tr>
<tr>
<td>Double_array</td>
<td>Optional !</td>
<td>Double_array</td>
<td>Required !</td>
</tr>
<tr>
<td>Double</td>
<td>Optional !</td>
<td>Double</td>
<td>Optional !</td>
</tr>
<tr>
<td>Double</td>
<td>Optional !</td>
<td>Double</td>
<td>Optional !</td>
</tr>
<tr>
<td>Double</td>
<td>Optional !</td>
<td>Integer</td>
<td>Optional !</td>
</tr>
<tr>
<td>Integer</td>
<td>Optional !</td>
<td>Integer</td>
<td>Optional !</td>
</tr>
<tr>
<td>Integer</td>
<td>Optional !</td>
<td>Integer</td>
<td>Optional !</td>
</tr>
<tr>
<td>Integer</td>
<td>Optional !</td>
<td>Integer</td>
<td>Optional !</td>
</tr>
<tr>
<td>Character</td>
<td>Optional !</td>
<td>Character</td>
<td>Optional !</td>
</tr>
</tbody>
</table>

$atomic-layers$ is only necessary for the heterostructure tight-binding code but not for the bulk tight-binding code.
Example: $atomic-layers

$atomic-layers

!--------------------------------------
! layer 1: cation
! layer 2: anion
!--------------------------------------

**Note:**

- The total number of layers in [001] superlattice direction must be integer multiple of 4 (e.g. 4, 8, 12, …) because a unit cell consists of 4 atomic layers.
- Convert number of layers (atomic layers) into [nm] units: position = layer-number * LatticeConstant / 4
- Convert number of molecular layers into [nm] units: position = layer-number * LatticeConstant / 2
- The lattice constant refers to the constant distance between unit cells in a crystal lattice. A unit cell consists of 2 molecular layers in diamond/zinc blende structure.
- It must hold: ion-1-content + ion-2-content = 1.0

Example: $tighten

$tighten

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
Example: $tighten

$atomic-layers

!-----------------------------------------------------------------!
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>L</td>
<td>! L</td>
</tr>
<tr>
<td>sections</td>
<td>0.5 0.5 0.5</td>
<td>! '1', '2', '3'</td>
</tr>
<tr>
<td>points</td>
<td>1000</td>
<td>!</td>
</tr>
<tr>
<td>units of (2pi/a) point</td>
<td>0.5 0.0 0.5</td>
<td>! k vector in_</td>
</tr>
<tr>
<td>units of (2pi/a) point</td>
<td>0.5 0.0 0.5</td>
<td>! k vector in_</td>
</tr>
<tr>
<td>dependence</td>
<td>2.0</td>
<td>!</td>
</tr>
<tr>
<td>eigenvectors</td>
<td>no</td>
<td>! 'yes' / 'no'</td>
</tr>
<tr>
<td>scale</td>
<td>1.0</td>
<td>!</td>
</tr>
<tr>
<td>left</td>
<td>0.0</td>
<td>! [eV]</td>
</tr>
<tr>
<td>right</td>
<td>0.0</td>
<td>! [eV]</td>
</tr>
<tr>
<td>level</td>
<td>1</td>
<td>! '1', '2', '3'</td>
</tr>
<tr>
<td>states</td>
<td>The full</td>
<td>!</td>
</tr>
<tr>
<td>geometry</td>
<td>no</td>
<td>! 'yes' / 'no'</td>
</tr>
<tr>
<td>Hamiltonian</td>
<td>no</td>
<td>! 'yes' / 'lower'</td>
</tr>
<tr>
<td>second / 'no'</td>
<td>'yes' / 'first'</td>
<td></td>
</tr>
<tr>
<td>option</td>
<td>! 'eigenvalues-'</td>
<td></td>
</tr>
<tr>
<td>only, 'eigenvectors-and-density', 'eigenvectors-and-g-factor', 'no-eigenvalues'</td>
<td>! used with 2</td>
<td></td>
</tr>
<tr>
<td>psi</td>
<td>1</td>
<td>!</td>
</tr>
<tr>
<td>above, index of ist of 4 bands =</td>
<td></td>
<td></td>
</tr>
<tr>
<td>plot</td>
<td>4</td>
<td>!</td>
</tr>
<tr>
<td>anion</td>
<td>no</td>
<td>! 'no' / 'yes'</td>
</tr>
<tr>
<td>(default is no. For testing only, cation and anion material parameters of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>database are exchanged.)</td>
<td></td>
<td>!</td>
</tr>
<tr>
<td>factor</td>
<td>1.0</td>
<td>! (default is 1.</td>
</tr>
<tr>
<td>0. scaling factor for potentials (0.0 &lt;= ... &lt;= 1.0)</td>
<td>0.0 = no</td>
<td></td>
</tr>
<tr>
<td>potential, i.e. free electron, 1.0 = full periodic potential, anything in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>between is a mixture</td>
<td>! ! Note: This</td>
<td></td>
</tr>
<tr>
<td>feature can be used to obtain the free-electron dispersion in a zincblende</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lattice</td>
<td></td>
<td>!</td>
</tr>
<tr>
<td>factors</td>
<td>0.0, 0.0, 0.136</td>
<td>! 10 values:</td>
</tr>
</tbody>
</table>

9.2. Input File
\[ U_{0,s} \text{ and } U_{0,a} \rightarrow \text{can be chosen both as 0.0. They just determine the reference energy.} \]
\[ \rightarrow \text{The subscripts } s \text{ and } a \text{ refer to the symmetric and antisymmetric form factors.} \]
\[ \rightarrow \text{The lattice constant is also needed!!!} \]
\[ G_{\nu} \rightarrow \text{The reciprocal G vectors up to this subscript, e.g. } G_0 (0), G_3 (3), G_4 (4), G_8 (8), \ldots G_{11} (11), G_{12} (12); \text{ The subscript is the length squared } |G|^2. \]
\[ \rightarrow \text{G}_0 = (0 0 0) \] (1 vector, total number of G vectors: 1)
\[ \rightarrow \text{G}_3 = (1 1 1) \] (8 vectors, total number of G vectors: 9)
\[ \rightarrow \text{G}_4 = (2 0 0) \] (6 vectors, total number of G vectors: 15)
\[ \rightarrow \text{G}_8 = (2 2 0) \] (12 vectors, total number of G vectors: 27)
\[ \rightarrow \text{G}_{11} = (3 1 1) \] (24 vectors, total number of G vectors: 51)
\[ \rightarrow \text{G}_{12} = (2 2 2) \] (8 vectors, total number of G vectors: 59)
\[ ! \text{end_tighten} \]
\[ ! \text{end_tighten} \]

**Note:** The standard parameter files which are specified as follows

- is then: input-directory/filename-material-parameters
- is then: input-directory/filename-distance-parameters

are located under

C:\Program Files (x86)\nextnano\nextnano3\Syntax\Tight-binding files\

**Parametrisation**

Scholz and Sarma parametrisations are given in the articles [JancuPRB1998] and Sarma2002.

**Syntax**

**calculate-tight-binding-tighten** do tight-binding calculation with tighten

- **options** yes or no
- **default** no (no tight-binding calculation)

**tighten-method**

- **options** rashba2tighten-tighten generate input file for tighten and do tight-binding calculation with tighten rashba2tighten generate input file for tighten only tighten tight-binding calculation ==> tighten only

**tight-binding-model**

- **options** Scholz (default) (only for III-V materials) Sarma (for II-VI materials)

**destination-directory** directory name for tight-binding output. This is where all output goes.
**input-directory** directory name for tight-binding input, relative to executable path.

- **default**: ./
- **example**: TightBinding/

**filename-tighten**

- Name of tight-binding input file (will be generated by rashba2tighten, and read in again by tighten).
- This file is written to destination-directory/ and will be read in again by the tighten algorithm.
- If tighten-method = tighten, the full filename is then: input-directory/filename-tighten

- **default**: tighten.in

**filename-k-vectors**

- This file contains user input for lattice structure and tight-binding parameters.
- It will be generated by rashba2tighten, and read in again by tighten program.
- If tighten-method = tighten, the full filename is then: input-directory/filename-k-vectors

- **default**: k_vectors.dat

**k-vectors-sample-type**

- This file contains the k vectors for the tight-binding Hamiltonian that will be diagonalized, i.e. for which the eigenenergies and eigenfunctions will be calculated.
- The columns in this file are:

<table>
<thead>
<tr>
<th>REAL(loop_index_over_k)</th>
<th>0.5*(kx+ky)</th>
<th>0.5*(ky+kz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5*(kx+kz)</td>
<td>kx</td>
<td>ky</td>
</tr>
<tr>
<td></td>
<td>kz</td>
<td></td>
</tr>
</tbody>
</table>

**options**

- **band** for band structure plot along some predefined lines (the lines are different for the bulk and superlattice code, see below)
- **user-defined-path** as defined in Brillouin-zone-path = ...
- **user-defined-sections** as defined in Brillouin-zone-sections = ...

For the superlattice tight-binding code the relevant options are:

- **band**
  - for band structure plot along the lines (superlattice code): Gamma ==> (along Delta to) X = 0.5 G1 ==> (along Z to) M = 0.5 (G1+G2) ==> (along Sigma to) Gamma ==> Z = 0.5 G3
  - reciprocal primitive vectors: G1 = (2pi sqrt(2)/a_lateral) * (1 0 0) ==> along (1 0 0) in-plane direction
  - G2 = (2pi sqrt(2)/a_lateral) * (0 1 0) ==> along (0 1 0) in-plane direction
  - G3 = (2pi /a_vertical) * (0 0 1) ==> along superlattice direction
  - a_lateral: in-plane lattice constant
  - a_vertical: out-of-plane lattice constant

Some predefined paths are:

- **k_parallel_100-Gamma-k_superlattice**
  - X = -0.5 G1 ==> (along Delta to) Gamma ==> Z = 0.5 G3
\[ k_{\text{parallel}}_{110} - \text{Gamma} - k_{\text{superlattice}} \quad M = -0.5 (G1+G2) \implies \]
(\text{along Sigma to}) \quad \text{Gamma} \implies Z = 0.5 G3 \\

110G 3 values, namely (000), (kx0,0,0), (0,kx0,0), [110] direction and [1-10] direction \\

100D “100D” - “001D” \\

two 2 values, namely (000) and the one specified (kx0,ky0,kz0) \\
cirG circle Gamma=(000) with radius kmod=|k0l, surrounding Gamma point at distance specified. \\
xyz number of k points values from (0,0,0) => (kx0,ky0,kz0) \\

Lprojected-Gamma-Xprojected \\
L-Gamma-X \\
k1-Gamma-k2 \\
k1-k2 \\

R-Z-Gamma-X-M-A See p. 404, Fig. 8.37 of C. Hamaguchi, Basic Semiconductor Physic, 2nd edition and Fig. 8.29 \\
Gamma-X-R-Z-Gamma-M-A-Gamma See Fig. 8 of A. Sawamura et al., Optical Materials Express 8, 1569 (2018) \\

For the bulk tight-binding code the relevant options are: \\
band for band structure plot along the lines (bulk code): same as L-Gamma-X-W-L-K-Gamma for the bulk tight-binding code \\

Some predefined paths are: \\
L-Gamma-X-W-K-L-W-X-K-Gamma \\

• same as [JancuPRB1998] and Fig. 1 of A. Sawamura et al., Optical Materials Express 8, 1569 (2018) \\
• (The points U and K have the same energies but the path to these points from Gamma or X is not equivalent,) \\
• L-Gamma is along Lambda axis, i.e. along [1,1,1] direction (L is at 2pi/a (0.5,0.5,0.5); Gamma is at 2pi/a (0,0,0) ). The distance between these points is pi/a sqrt(3). \\
• Gamma-X is along Delta axis, i.e. along [1,0,0] direction (X is at 2pi/a (1,0,0) ) \\
• X-W is along V \\
• W-K is along \\
• K-L is along \\
• L-W is along Q \\
• W-X is along V \\
• X-K is along \\
• K-Gamma is along Sigma \\

L-Gamma-X-W-L-K-Gamma e.g. S. Sapra et al., PRB 66, 205202 (2002) (same as band) \\
L-Gamma-X-K-Gamma
L-Gamma-X U K Gamma e.g. Tom P. Pearsall, Quantum Photonics (Band structure plots for pseudopotential calculations) (to do: eliminate the path between U and K)

L-Gamma-X

X-Gamma-Z U' L Gamma K (for strain)

110G 3 values, namely (000), (k0,kx0,0), (-k0, kx0,0)

110X 3 values, namely (001), (k0,kx0,1), ( kx0,-k0,0,1)

110D 3 values, namely (00kz0), (k0,kx0,kz0), (-k0, kx0,kz0)

110L 3 values, namely (0.5,0.5,0.5), (0.5+kx0,0.5+kx0,0.5-2kx0),

(0.5+kx0,0.5-kx0,0.5)

100G 3 values, namely (000), (k0,0,0),0(ky0,0,0)

two 2 values, namely (000) and (k0,ky0,kz0)

cirG circle Gamma=(000) in kz=0 plane with radius kmod=|k0|

cirX circle X=(001) in kz=1 plane with radius kmod=|k0|

cirL circle L=(111) in plane lambda*(1,1,-2) + mu*(1,-1,0) with radius kmod=|k0|

xyz number_of_k_points values from ( 0 , 0 , 0 ) ==> (k0,ky0,kz0)

xff number_of_k_points values from ( 0 ,ky0,kz0) ==> (kx0,ky0,kz0)

ffz number_of_k_points values from (kx0,ky0, 0 ) ==> (kx0,ky0,kz0)

For the bulk tight-binding code of graphene the relevant options are:

M'_K_Gamma_M_K' band structure of graphene along the path M' ==>
K ==> Gamma ==> M ==> K' ==> M'

K_Gamma_M_K' band structure of graphene along the path K ==> Gamma
===> M ==> K'

Brillouin-zone-path This is a path through the Brillouin zone passing through all these k points in this order. The number of entries must be a multiple of 3.

example

```
  path  0.5  0.5  0.5  ! L
         0.0  0.0  0.0  ! Gamma
         1.0  0.0  0.0  ! X
         | | | | | | ! ...
```

Brillouin-zone-sections

example

```
  sections  0.5  0.5  0.5  ! L  ! Section 1: This is a path through the Brillouin zone between these two k points. L ==> Gamma
         0.0  0.0  0.0  ! Gamma ! 0.0  0.0  0.0  ! Gamma ! Section 2: Then a path through the Brillouin zone between these two k points is taken. Gamma ==> X
         1.0  0.0  0.0  ! X
         0.0  0.0  0.0  ! Gamma ! Section 3: Then a path through the Brillouin zone between these two k points is taken. Gamma ==> L
```

(continues on next page)
Brillouin zone of zincblende semiconductor

- Define Brillouin zone boundaries in units of \(2\pi/a\) where \(a\) is the lattice constant.
- The first Brillouin zone comprises those points in reciprocal space that are closer to the origin (i.e. the Gamma point) than to any other point of the reciprocal lattice.
- The high symmetry points are called:
  - \(L\) \([1/2, -1/2, 1/2]\) \(~[1-11]\)
  - Gamma \([0, 0, 0]\) (origin, i.e. center of the 1st Brillouin zone)
  - \(X\) \([1, 0, 0]\) \(~[100]\)
  - \(U\) \([1, 1/4, -1/4]\)
  - \(K\) \([0, 3/4, 3/4]\)
  - Gamma \([0, 0, 0]\)
  - \(W\) \([1, sqrt(2)/2, 0]\)
- The line from [Gamma] to [X] along the high-symmetry direction \([100]\) is called Delta.
- The line from [Gamma] to [L] along the high-symmetry direction \([111]\) is called Lambda.
- The line from [K] to [X] along the high-symmetry direction \([. . .]\) is called Sigma. (should be checked)
- The distance in \(k\) space from [Gamma] to [L] is \(sqrt(3)\pi/a\).
- If one goes from [Gamma] to [X] along the line [Delta], the cubic symmetry splits the 8 bands into 2 bands.
- At the zone center [Gamma], the energy value is 8-fold degenerate.
- At the [X] point, there are two bands, each of which is 4-fold degenerate.
- If one goes from [Gamma] to [L] along the line [Lambda], the cubic symmetry splits the 8 bands into 4 bands, the upper and lower one is 1-fold degenerate, the two intermediate ones are each 3-fold degenerate.
- The electronic wave function at the [Gamma] point in the center of the Brillouin zone sees the maximum symmetry of the fcc lattice.
- There are 48 operations (rotations, reflections, inversion) that leave the structure invariant.

**number-of-k-points** number of \(k\) vectors for which to calculate eigenvalues (only relevant for band, \(k_{parallel\_100}\)-Gamma-\(k\) superlattice, \(k_{parallel\_100}\)-Gamma-\(k\) superlattice, \(xyz\) so far)

**example** 1000

With the following flags with can specify a \(k\) vector in the Brillouin zone.
The superlattice k.p dispersion can be calculated along an arbitrary line from the k point ‘k-direction-from-k-point’ to the Gamma point and then to the k point ‘k-direction-to-k-point’. Either k-direction-from-k-point or k-direction-to-k-point or both can be zero. If both are zero, then only the Gamma point is calculated. k-direction-from-k-point can be omitted.

You can use this flag to specify a customized plot for the E(k) dispersion, e.g. along a line from [110] to the Gamma point and then to the [001] point.

\[
\begin{array}{ccc}
  \text{point} & -0.5 & -0.5 & 0.0 \text{! k vector} \\
  \text{in units of [2pi/a]} & \\
  \text{point} & 0.0 & 0.0 & 0.0 \text{! k vector} \\
  \text{in units of [2pi/a]} & \\
\end{array}
\]

**power-of-distance-dependence**

default 2.0

ti = t0*(d0/di)\^\eta It seems that this value is not used inside the code, unscaled matrix elements are used.

**calculate-eigenvectors**

value yes or no
default no

**scale**

default 1.0

test 5.0

scale output of wave functions and \(|\psi|^2\) to improve visualization of \(|\psi|^2\) in the band edges plot

**potential-energy-left**

default 0.0 [eV]

Add potential energy to band edges (value at first layer).

**potential-energy-right**

default 0.0 [eV]

Add potential energy to band edges (value at last layer). The values in between are interpolated for each layer. This way an electrostatic potential or electric field can be included.

**debug-level**

default 1

generate standard output, this is appropriate to generate an input file if unit cell has many atoms. NOTE: only first element of star is printed.

**options**

- 2 get very detailed output, excluding Hamiltonian
- 3 get very detailed output, including Hamiltonian to stdout

**filename-states**

default states.in

The full filename is then: input-directory/filename-states

This file is located in the folder: C:\Program Files (x86)\nextnano\nextnano3\Syntax\Tight-binding files

**calculate-only-lattice-geometry**

default no
option  yes Calculate only lattice geometry but not tight-binding Hamiltonian.

output-Hamiltonian
  default  no Hamiltonian matrix not written to file
  options
    • yes whole Hamilton matrix is written column wise
    • lower lower triangle is written row wise

calculate-k-derivatives
  default  no do not calculate k-derivatives of H(k)
  options
    • yes calculate first and second
      • first calculate d/dk H(k)
      • second calculate d²/dkikj H(k)

Note:  ham_1deriv_output_file
       ham_2deriv_output_file
       ... files that contain nonzero matrix elements of 1st (and 2nd) k-derivative of H in same format as
       ham_output_file,
       BUT ONLY SPIN_UP/SPIN-UP PART since SPIN-DN/SPIN-DN is identical and SPIN-DN/SPIN-
       UP=0
       ‘der1.dat’ ! ham_1deriv_output_file = ‘der1.dat’
       ‘der2.dat’ ! ham_2deriv_output_file = ‘der2.dat’

tighten-option
  default  eigenvalues-only calculate only eigenvalues (eigval)
  options
    • eigenvectors-and-density calculate eigenvectors and density
      of tight-binding Hamiltonian (ev+dens)
    • eigenvectors-and-g-factor calculate eigenvectors and g-
      factor (ev+g)
    • no-eigenvalues no diagonalization

number-of-band-for-Fermi-energy
  range  \[ N \]
  default  0 i.e. highest valence band number
  example  8
  zero of energy is taken at this band for Gamma point (or first k-vector)
  This specifier only applies to the bulk tight-binding algorithm and the pseudopotential code.

number-of-electron-eigenvalues
  range  \[ N \]
  default  11

number-of-electron-eigenvalues
  range  \[ N \]
default 15
This specifier only applies to the superlattice tight-binding algorithm.
Include 11 electron and 15 hole energies in E(k) energy dispersion plot.

number-of-band-for-psi

default 1
used with 2 above, index of 1st of 4 bands w $|\psi|^2$ (actually 8 bands due to spin degeneracy)

Note:
• If ieigenvalues_flag=2, wave functions are calculated at Gamma for every second of number-of-bands-to-plot/2 consecutive bands, starting with number-of-band-for-psi, where -1 and 0 = top of valence band, 1 and 2 = 1st conduction band. Choose an odd number.
• If ieigenvalues_flag=3, g-factor tensor for this plus following (nondegenerate) band is calculated

number-of-bands-to-plot

default 4
This is the number of bands that are plotted in the files
• Gamma_psi_squared.dat $|\psi|^2$, ... are not shifted
• Gamma_psi_squared_shift.dat $|\psi|^2$, ... are shifted by their eigenenergies with respect to the energy dispersion plot
• Gamma_psi_squared_shift_bandedges.dat $|\psi|^2$, ... are shifted by their eigenenergies with respect to the band edges plot
starting from the band indicated with number-of-band-for-psi.
This number does not take into account spin. E.g. if you specify ‘4’, the program will print out 8 bands, i.e. twice as much, i.e. spin is taken into account automatically by the program.

rescale-to-unstrained-k-points

default no Do not rescale band structure to unstrained k points. It can be useful to keep the high symmetry points fixed on the graph in order to compare different strains.
option yes Rescale band structure to unstrained k points. This moves the high symmetry points if strained.
This flag is only relevant for the tight-binding code of graphene.

Necessary input files

Material parameters

TB_material_parameters.in
Material parameters that also occur in the normal nextnano database:
c11 c12 elastic constants $10^{-2}$ [GPa]
a lattice constant [Angstrom] tight binding parameters [eV]

Tight-binding material parameters
number of electrons on cation and number of electrons on anion (e.g. 3 for Ga and 5 for As in GaAs; 4 for Si)

orbital energies [eV], e.g. Esc is s-orbital energy of cation

orbital energies [eV], e.g. Esa is s-orbital energy of anion

Edct2 and Edat2 are used instead of Ed to allow for d-orbital splitting

absolute deformation potential [eV]

difference in absolute valence band deformation potentials (in eV) between intrinsic Scholz values and Van de Walle/Needs values.

Inclusion of this term guarantees that calculated av agrees with Van de Walle/Needs.

absolute deformation potential [eV]

difference in absolute valence band deformation potentials (in eV) between intrinsic Scholz values and Van de Walle/Needs values.

on-site and intersite spin-orbit couplings

so\_p\_c so\_p\_a

so\_d\_c so\_d\_a

Gamma\_av energy of unstrained top of valence band edge at Gamma point;

Gamma\_ac energy of unstrained conduction band edge minimum at Gamma point [eV]

Both are with respect to an absolute energy scale (crude estimate only)

The last two values are not used for the tight-binding calculation. However, these are the relevant values that are contained in the output files band\_edges\_nm.dat and band\_edges.dat. They are the values of the unstrained band edges on an absolute energy scale. They are crude estimates only. As already said, they are not input to the actual tight-binding calculation but they are similar to the results of a bulk tight-binding calculation.

The empirical tight-binding material parameters for Ge are the ones of JancuPRB1998, Table II with the following exceptions: Es, Ep, Ed, Es* are shifted by +1.7683 eV.

Distance parameters

These parameters are relevant for strained materials or for alloys.

strain dependent shift of on-site-d energies in [eV] (deformation potential). It is chosen to be identical for cation and anion.

bdef = bd * ed\_Scholz
States

states.in

This noneditable namelist file will be read in by the tighten program and contains the following information:

&state_description

number_state_ref = 10

state_ref_name(1) = 's'
state_ref_name(2) = 'px'
state_ref_name(3) = 'py'
state_ref_name(4) = 'pz'
state_ref_name(5) = 'se'
state_ref_name(6) = 'dxy'
state_ref_name(7) = 'dyz'
state_ref_name(8) = 'dzx'
state_ref_name(9) = 'dx2y2'
state_ref_name(10) = 'dz2r2'

number_coup_ref = 21

coup_ref_name(1) = 'sss'
coup_ref_name(2) = 'sps'
coup_ref_name(3) = 'pss'
coup_ref_name(4) = 'pps'
coup_ref_name(5) = 'ppp'
coup_ref_name(6) = 'seses'
coup_ref_name(7) = 'sess'
coup_ref_name(8) = 'sses'
coup_ref_name(9) = 'seps'
coup_ref_name(10) = 'pses'
coup_ref_name(11) = 'sds'
coup_ref_name(12) = 'dss'
coup_ref_name(13) = 'pds'
coup_ref_name(14) = 'dps'
coup_ref_name(15) = 'pdp'
coup_ref_name(16) = 'dpp'
coup_ref_name(17) = 'seds'
coup_ref_name(18) = 'dse'
coup_ref_name(19) = 'dds'
coup_ref_name(20) = 'ddp'
coup_ref_name(21) = 'ddd'

/
Generated output files

- k_vector.dat
- tighten.in

These are the input files for tighten.

- out_structure.txt
- band_edges_unstrained_nm.dat Gamma conduction band edge and topmost valence band edge (units: position [nm], energy [eV], band gap [eV])
- band_edges_unstrained_layers.dat Gamma conduction band edge and topmost valence band edge (units: atomic layer, energy [eV])
- E(k)_tighten_new_bandedges.dat energy dispersion E(k) where the x axis is either in units of [1/Angstrom] or integer numbers indicating the number of k points (not shifted, energies correspond to tight-binding material parameters in input file)
- E(k)_tighten_new.dat energy dispersion E(k) where the x axis is either in units of [1/Angstrom] or integer numbers indicating the number of k points (shifted so that topmost valence band edge equals 0 eV)
- E(k)_tighten.dat energy dispersion E(k) (original output) (shifted so that topmost valence band edge equals 0 eV)

Output files of tighten (bulk)

- E(k)_tighten_bulk_new_noshift.dat energy dispersion E(k) (original output) (not shifted, energies correspond to tight-binding material parameters in input file)
- E(k)_tighten_bulk_new.dat energy dispersion E(k) (shifted so that valence band edge of first k vector equals 0 eV)
- E(k)_tighten_bulk.dat energy dispersion E(k) (original output) (shifted so that valence band edge of Gamma k vector equals 0 eV (default) (shifted so that valence band edge of first k vector equals 0 eV (debug-level = 10))

If k-vectors-sample-type = xyz, then the k vectors are in units of [1/nm] of the files out_ek_tighten_bulk_new*.dat.

hamtightenout_bulk.txt
driver_file_sl.in

'hamtightenout.dat' ham_output_file = ‘hamout.dat’

Strain

```bash
$simulation
  strain-calculation strain
```

Include biaxial strain for superlattice tighten code.

```bash
strain-calculation no-strain
```

No strain is considered for superlattice tighten code, i.e. strain tensor is zero and each layer has equilibrium lattice constant in growth direction.

In any case, the lattice constant in the lateral growth direction is the one of the substrate material specified in
for both, homogeneous-strain and no-strain.

(Internally, strain001V and the trace of the strain tensor are set to zero. The vertical lattice constant is the equilibrium lattice constant of the layer material.)

### $tight$-binding

This is preliminary. It is not implemented yet.

```plaintext
! Contributors: nextnano Development Team

$domain

$tight

! --> TIGHT BINDING DRIVER INPUT

character  required ! 'yes' / 'no'
character  optional ! 'yes' / 'no'

! --> TIGHT BINDING INPUT

integer  optional ! 0 (no screen output), 1 (minimum screen output), 2 (detailed screen output)
character  required ! root path for the whole TB simulation, e.g. 'TB_data/'
character  required ! name of the TB parse file, e.g. 'data_driver'
character  required ! name of the reference states and couplings file, e.g. 'state.dat'

! SUPERCCELL

double_array  required ! x,y,z [nm]
double_array  required ! x,y,z [nm]

! --> TIGHT BINDING INPUT

integer  required ! e.g. 1
character  required ! e.g. 'yes' / 'no', e.g. 'yes'
double  required ! (%), e.g. 20.0
character  required ! e.g. 'yes' / 'no', e.g. 'yes'
character  required ! alloy -> 'yes' / 'no', e.g. 'no'

! cyclic permutations

character  required !

! --> BASIS INPUT

integer  required ! 0 (none), 1 (generate xmol xyz file), 2 (generate basis file for strain calculation + xmol xyz file), 3 (calculate strain + generate xmol xyz file), 4 (add strain from data file + generate xmol xyz file)
character  required !
strain  character  required ! 'yes' / 'no'
strain  character  required ! e.g. 'strain1'
strain  character  required ! e.g. 'basis_coord.dat'
strain  character  required ! e.g. 'basis_coord.dat'
strain  character  required ! e.g. 'TB_potential.dat'
strain  character  required ! 'yes' / 'no', e.g. no
strain  character  required ! e.g. 'TB_potential.dat'

! --> SPARSE MATRIX INDEX INPUT

character  required ! 'upper'
```

(continues on next page)
### $warnings

Warnings can be switched on or off. It is recommended to use the default settings which are specified in the database file `database_nn3.in`. Currently, changing the default value does not have much effect.

<table>
<thead>
<tr>
<th>$warnings</th>
<th>logical</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$end_warnings</td>
<td>logical</td>
<td>required</td>
</tr>
</tbody>
</table>

Specify here if warnings should be written to the `.log` file.

**warnings**

- **type** logical
- **presence** required
- **options** `.FALSE.`, `.TRUE.`

**default** uses value specified in `database_nn3.in`

**Example**

```bash
$warnings !
warnings ! switch on warnings
$end_warnings !
```

(continued from previous page)
$\textsc{SNEGF-spintronics}$

NEGF for spintronics: Spin transport based on the Nonequilibrium Green's functions (NEGF) method.

This part is based on the PhD thesis of [KubisPhD2009].

This is preliminary. It is not useful for nondevelopers.

\begin{verbatim}
$\text{SNEG}$
\begin{tabular}{ll}
  coordinate & character \ required! \\
  character   & character \ optional! \\
  character   & character \ optional! yes/no \\
  character   & character \ optional! yes/no \\
  integer     & integer \ optional! \\
  integer     & integer \ optional! number of grid points \\
\end{tabular}

--in energy
\begin{tabular}{ll}
  double     & double \ optional! minimum of considered \\
\end{tabular}

--energies in [eV]
\begin{tabular}{ll}
  double & double \ optional! BIA [eVn] \\
  double & double \ optional! SIA (Rashba) [eVnm] \\
  double_array & double_array \ optional! B_x, B_y, B_z [T] \\
  double_array & double_array \ optional! B_x, B_y, B_z [T] \\
  double_array & double_array \ optional! B_x, B_y, B_z [T] \\
  double & double \ optional! [eV] \\
  double & double \ optional! [eV] \\
  double & double \ optional! [eV] \\
  contacts & contacts \ character \ optional! yes/no \\
  contacts & contacts \ character \ optional! yes/no \\
  contacts & contacts \ character \ optional! yes/no \\
  contacts & contacts \ character \ optional! yes/no \\
  contacts & contacts \ character \ optional! yes/no \\
  contacts & contacts \ character \ optional! yes/no \\
\end{tabular}

$\text{end$\textsc{SNEG}}$
\end{verbatim}

$\text{output-grid}$

Write grid coordinates to output file. (This keyword is not very useful.)

\begin{verbatim}
$\text{output}$
\begin{tabular}{ll}
  integer_array & integer_array \ required \\
  character     & character \ required \\
\end{tabular}

$\text{end-output}$
\end{verbatim}

Example

\begin{verbatim}
!----------------------------------!
$\text{output}$
\begin{tabular}{l}
  coordinate A 1 0 0 ! specify coordinate axis (1 0 0) or (0 1 0) or (0 0 1) \\
  file       A*.xyz     ! specify output file name \\
  coordinate A 0 1 0 !
\end{tabular}

$\text{end-output}$
!----------------------------------!
\end{verbatim}

The following keywords are not documented yet in the new manual, i.e. they are linked to the old manual.
$domain-coordinates

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/domain-coordinates.htm

$material

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/material.htm

$strain-minimization-model

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/strain-minimization-model.htm

$poisson-boundary-conditions

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/poisson-boundary-conditions.htm

$quantum-bound-states

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-bound-states.htm

$quantum-dot-layer-density

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-dot-layer-density.htm

$quantum-state-recombination-rates

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/quantumstate-recombination-rates.htm

$output-strain

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/output-strain.htm

$output-bandstructure

The documentation for this keyword is available here:  https://www.nextnano.com/nextnano3/input_parser/keywords/output-bandstructure.htm
$output-1-band-schroedinger

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-1-band-schroedinger.htm

$output-kp-data

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-kp-data.htm

$output-densities

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-densities.htm

$output-current-data

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-current-data.htm

$output-file-format

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-file-format.htm

$output-section

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-section.htm

$output-material

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-material.htm

$output-raw-data

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-raw-data.htm

$output-geometry

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/output-geometry.htm
$doping-function

The documentation for this keyword is available here:  [https://www.nextnano.com/nextnano3/input_parser/keywords/doping-function.htm](https://www.nextnano.com/nextnano3/input_parser/keywords/doping-function.htm)

$interface-states

The documentation for this keyword is available here: [https://www.nextnano.com/nextnano3/input_parser/keywords/interface-states.htm](https://www.nextnano.com/nextnano3/input_parser/keywords/interface-states.htm)

$global-parameters

Global parameters are general parameters which are valid all over the device. See also $global-parameters under database section.

```
$global
  double optional
  character optional
  integer optional
$end_global
```

The lattice temperature is given in Kelvin. The lowest allowed temperature can be set in $program_restrictions.

**Note:** $global-parameters is optional. If it is not present, the default value for the temperature is taken from $global-parameters of the database file. Currently, the default temperature is set to 300 K.

**Note:** The band gaps (namely the conduction band edges only) are adjusted when the temperature changes. This feature can be switched off. See $numeric-control: varshni-parameters-on = no

In addition, the lattice constants depend on temperature. This can be switched off as well. See $numeric-control: lattice-constants-temp-coeff-on = no

**Temperature sweep**

It is possible to sweep over the temperature, i.e. to vary the temperature stepwise. This is similar to

- electric field sweep ($electric-field$)
- magnetic field sweep ($magnetic-field$)
- voltage sweep ($voltage-sweep$)
- doping concentration sweep ($doping-function$)
- alloy sweep ($alloy-function$).

The output is labeled with ..._ind000.dat,..._ind001.dat,..._ind002.dat,... where the index refers to the number of the temperature sweep step.

**Example**

```
!--------------------------------------------!
$global
  temperature 300.0 ! [K]
  active yes ! yes/no
  size -10.0 ! [K], i.e. in this case -10
$end_global
```

(continues on next page)
In this example, the temperature is varied starting from 300 K, and then reducing the temperature 20 times in steps of \(-10\). Obviously, increasing the temperature is also possible. If you do not want to write out all data specified in the output section for every step, you have to enter an integer number greater than one. If you specify 1, then output files will be generated for each temperature sweep. This is useful if you want to fill out all unused space on your hard disk.

Restrictions

130/140: same as 13/14 but including self-consistent Poisson-Schrödinger.

See tutorial Electron concentration in doped semiconductors (Si, Ge, GaAs) for details.

$global-settings

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parseter/keywords/global-settings.htm

$simulation-flow-control

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parseter/keywords/simulation-flow-control.htm
**$input-filename**

Specification of input file name

This is optional (and deprecated). It is much more convenient to start the simulation using command line argument --inputfile "D:\test\Quantum Well.in". The only reason for the existence of this keyword is to start a simulation without command line arguments, e.g. for testing purposes. The name of the input file to be run can be written into the file keywords.val within the keyword $input_filename which must be the first (!) entry in this file. Apart from this modification keywords.val must not be changed.

The specification of the input filename is done as a first entry after $input_filename in the file keywords.val as follows:

```plaintext
$input_filename  optional
   !!!character  optional
$end_input_filename  optional
```

Example 1

```plaintext
!--------------------------------------------!
$input_filename  optional !
   !!!character  optional ! Reads in "input_file1.in".
!ID_Quantum_well.in  character  optional ! Reads in "ID_Quantum_well.
   ..in" if the comment sign "!" is removed.
$end_input_filename  optional !
!--------------------------------------------!
```

A comment can be inserted using ! or #.

Example 1

```plaintext
!-------------------------------------------------------------------!
$input_filename  optional !
   "D:\My\nextnano\input\input_file1.in"  character  optional !
   ..Reads in "input_file1.in".
$end_input_filename  optional !
!-------------------------------------------------------------------!
```

**$grid-specification**

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/grid-specification.htm

**$import-data-on-material-grid**

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/import-data-on-material-grid.htm

**$binary-zb-default**

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/binary-zb-default.htm
$binary-wz-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/binary-wz-default.htm

$ternary-zb-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/ternary-zb-default.htm

$ternary-wz-default

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/ternary-wz-default.htm

$alloy-function

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/alloy-function.htm

$numeric-control

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/numeric-control.htm

$CBR-current

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/CBR-current.htm

$quantum-regions

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-regions.htm

$quantum-model-electrons

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-model-electrons.htm

$quantum-model-holes

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/quantum-model-holes.htm
$\text{voltage-sweep}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/voltage-sweep.htm

$\text{electric-field}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/electric-field.htm

$\text{simple-drift-models}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/simple-drift-models.htm

$\text{Monte-Carlo}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/Monte-Carlo.htm

$\text{NEGF}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/NEGF.htm

$\text{electrolyte}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/electrolyte.htm

$\text{electrolyte-ion-content}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/electrolyte-ion-content.htm

The documentation of the keywords for the nextnano$^3$ software is available here: https://www.nextnano.com/nextnano3/input_parser/keywords/keywords.htm

### 9.3 Database

#### 9.3.1 The database keywords

Here you can find detailed descriptions about the keywords and specifiers in the default parameter database input file `database_nn3.in` and the general definitions file `database_nn3_keywords.val`.

The scheme is

- `keywords.val` (general definitions) $<==>$ `input_file.in` (input file)
- `database_nn3_keywords.val` (general definitions) $<==>$ `database_nn3.in` (database file)
Valid keywords for the parameters of the default parameter database are listed in `database_nn3_keywords.val`. This is the place to enter new keywords if one wants to add new parameters. In principle, one should only enter into this database keywords, which are marked as `required`. Through this it is warranted that corresponding entries in the database actually do occur. The input parser checks this, but only their existence. Additionally, the new keywords have to be entered into `keywords.val`. E.g. for zinc blende materials the keyword is `$binary_zb_default`. On this basis, the entries in the database `database_nn3.in` have to be complemented for the newly declared values for all affected known materials. This also includes the adaptation of changes in the data types of already existing parameters (e.g. `double => double_array`).

Material parameters can be found in the review articles of Vurgaftman and Meyer ([VurgaftmanJAP2001], [VurgaftmanJAP2003]).

### 9.3.2 Keywords

#### $binary-zb-default

The documentation for this keyword is available here: [https://www.nextnano.com/nextnano3/input_parser/database/binary-zb-default.htm](https://www.nextnano.com/nextnano3/input_parser/database/binary-zb-default.htm)

#### $binary-wz-default

The documentation for this keyword is available here: [https://www.nextnano.com/nextnano3/input_parser/database/binary-wz-default.htm](https://www.nextnano.com/nextnano3/input_parser/database/binary-wz-default.htm)

#### $ternary-zb-default

Ternary zinc blende parameters

Parameters for zinc blende type ternary alloys. This set of parameters refers to the binary constituents and their material parameters. Here, the bowing parameters for interpolation between these binaries are specified.

A bowing parameter $b$ is defined as follows for the material parameter $Q$. Note that there is a minus sign in front of the term $b x(1-x)$.

$$Q[A_x B_{1-x} C] = x \cdot Q[AC] + (1-x) \cdot Q[BC] - b \cdot x \cdot (1-x)$$

$b$ is defined as $b = 4Q(A_{0.5}B_{0.5}C) - 2(Q[AC] + Q[BC])$.

The advantage of the bowing model is that it requires knowledge of the relevant quantity only at a composition $x=0.5$ together with the values for the binaries.

Please check the keywords section for more details: $ternary-zb-default$

For the meaning of the material parameters and its units, please check the keywords section for $binary-zb-default$ for more details. The units for binary material parameters and for the bowing parameters are the same.

```plaintext
$ternary-zb-default

character required
character required
character required
double_array optional
double_array optional
double_array optional
double_array optional
double_array optional
double_array optional
double_array optional
double_array optional
double_array optional
double_array optional
```

(continues on next page)
Example 1

If no bowing parameters are specified, linear interpolation is done.

Example 2

If a bowing parameter is nonzero, bowing is used. Bowing can be quite complicated, especially if

- valence band offset (average valence band edge energy)
- spin-orbit splitting energy
- band gap

are bowed simultaneously.

Note that you can use `band-shift` to shift the resulting band edges by the desired amount.

It is hard to fine bowing parameters for a default material database that works for every alloy parameter.
Example 3

You can define special ternaries for certain conditions, e.g. low temperature or high In content.

\[ \text{ternary-type e.g. } \text{Al}(x)\text{Ga}(1-x)\text{As-zb-default, must be a defined ternary material. This string is usually a known material. If a material-type with material-model = ternary-zb-default is specified within the$\text{material}$ keyword which is unknown, you have to provide a complete set of input data for this material type. In this case the material-type must be equal to string. However, the binary constituents can still be either known or unknown binary materials.} \]

\[ \text{binary}(x) \text{ and binary}(1-x) \text{ must be a binary material of type binary-zb-default, e.g. AlSb-zb-default, must be a defined binary material. This string can be either a known binary or an arbitrary name. In case this binary is not a known material, you will be prompted for all material parameters.} \]

\[ \text{band-shift} \]
type double
presence optional
value 0.0

If nonzero, the resulting ternary material is shifted by this amount (independent of the alloy-conent). So the default value should be 0.0.

**bow-valence-band-energies**

<table>
<thead>
<tr>
<th>type</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>presence</td>
<td>optional</td>
</tr>
<tr>
<td>value</td>
<td>0.0</td>
</tr>
</tbody>
</table>

This bowing applies to the average valence band edge energy and not to the valence band maximum.

### Sternary-wz-default

Ternary wurtzite parameters

Parameters for wurtzite type ternary alloys. This set of parameters refers to the binary constituents and their material parameters. Here, the bowing parameters for interpolation between these binaries are specified.

```plaintext
$tternary
... $ternary

... "average" valence band edge energy E_v (without splittings)
... zincblende, this is double
... zincblende, this does not exist
... used in wurtzite
... zinblende, this is double

$end_ternary
```

Here, no further explanations are given. The meaning is analogous to $Sternary-zb-default.
$default-materials

Default materials.

<table>
<thead>
<tr>
<th>$default-materials</th>
<th>character</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$default-materials</td>
<td>character</td>
<td>required</td>
</tr>
<tr>
<td>$default-materials</td>
<td>character</td>
<td>required</td>
</tr>
<tr>
<td>$default-materials</td>
<td>character</td>
<td>required</td>
</tr>
<tr>
<td>$end_default</td>
<td>character</td>
<td>optional</td>
</tr>
</tbody>
</table>

Explanation of specifiers.

**material-name**
- **type** character
- **presence** required
- **value** e.g. Si

An arbitrary name can be chosen, e.g. Si or silicon.

**material-type**
- **type** character
- **presence** required
- **value** e.g. Si-zb-default

The material parameters of Si-zb-default have to be defined in the database.

**material-model**
- **type** character
- **presence** required
- **value** binary-zb-default, binary-wz-default, ternary-zb-default, ternary-wz-default; not supported yet: quaternary-wz-default

Zinc blende (zb) and wurtzite (wz) binary and ternary alloys are supported. Materials with diamond crystal structure can be specified using zinc blende. SiGe alloys are called “ternaries” even if only two atoms are involved.

For alloys (ternaries) only.

**material-property**
- **type** character
- **presence** optional
- **value** swap-(x)-and-(1-x)

If material is a ternary material, this value can be set to swap-(x)-and-(1-x). This allows to specify 2 different types of SiGe alloys that contain the same material parameters but \( x \) and \( y = (1-x) \) are interchanged.

Here, material-name is swapped but both refer to the same material-type.

You can also define your own materials, e.g. GaAs with parameters for 4 K, or explicit parameters for a ternary, e.g. Al\(_x\)In\(_{1-x}\)P with a fixed value of \( x=0.52 \): Al\(_{0.52}\)In\(_{0.48}\)P

9.3. Database
Further examples

```
$default

![Example function. Quaternaries are not supported yet.]
```

$default-material-models

Default material models.

```
$default

<table>
<thead>
<tr>
<th>$default</th>
<th>$end_default</th>
</tr>
</thead>
</table>
```

Zinc blende and wurtzite materials are supported. Binary and ternaries are possible. The latter need an alloy function. Quaternaries are not supported yet.

Example

```
$default

! Zinc blende
```

(continues on next page)
More information on the Physics: Auger recombination processes in semiconductor heterostructures

For devices with an extremely high carrier concentration the Auger process is the dominant recombination channel. The process involves three particles and therefore scales with the third power of the carrier densities. The phonon-assisted Auger recombination rate, which plays an important role especially at high carrier injection, respectively high doping levels, will be modeled in the program by the following equation:

$$R_{Aug} = (C_n n + C_p p)(np - n_i^2)$$

where $n$ is the electron density, $p$ is the hole density and $n_i$ is the intrinsic density.

### $Auger-recombination$

**Explanation of specifiers.**

**material-name**

*type* character 

*presence* required 

*value* e.g. GaAs

Name of material to which this set of parameters applies. Name has to be listed in $default-materials$. 

**number-of-parameters**

*type* integer 

*value* required
presence  required
value  e.g. 2

Control parameter if the number of parameters provided is the same as demanded.

There are two sets of parameters, one for electrons (\(n\)) and one for holes (\(p\)).

\textbf{n-C}

\texttt{type double}
\texttt{presence required}
\texttt{value e.g. 1.0e-30}
\texttt{unit [cm^6/s]}

The \(C_n\) parameter for electrons as specified in the equation above. The order of magnitude is around \(10^{-30}\) [cm^6/s].

\textbf{n-bow-C}

\texttt{type double}
\texttt{presence optional}
\texttt{value e.g. 0.0}
\texttt{unit [cm^6/s]}

For ternary alloys there are also bowing parameters possible. \(n\text{-bow-C} = 0.0\) means zero bowing, i.e. linear interpolation is used.

\textbf{p-C}

\texttt{unit [cm^6/s]}

Same as \texttt{n-C} but for holes.

\textbf{p-bow-C}

\texttt{unit [cm^6/s]}

Same as \texttt{n-bow-C} but for holes.

\textbf{Example}

```plaintext
$Auger

name   2
parameters  2
C      2.8e-31 ! [cm^6/s]
C      9.9e-31 ! [cm^6/s]

name   2
parameters  2
C      1.0e-30 ! [cm^6/s]
C      1.0e-30 ! [cm^6/s]

name   2
parameters  2
b              0.0 ! [cm^6/s]
b              0.0 ! [cm^6/s]

$end_Auger
```

There is also a keywords section in the input file for \texttt{$Auger-recombination} which you can use to overwrite default material parameters.
$direct-recombination$

The simplest process for the generation and recombination of electron-hole pairs is the direct process via the emission or absorption of a photon (radiative recombination). This is important for light emitting devices. The local spontaneous emission rate is approximated as

\[ R_{dir} = C(np - n_i^2) \]

where \( C \) is called the \textit{bimolecular recombination coefficient} (often abbreviated with the letter \( B \)), \( n \) is the electron density, \( p \) is the hole density and \( n_i \) is the intrinsic density.

This simple equation includes the full spectrum of photons generated by spontaneous band-to-band transition processes. By definition, the radiative recombination rate (= spontaneous electron recombination rate) per unit volume of an electron-hole pair is equivalent to the spontaneous photon generation rate per unit volume, because obviously, each time an electron recombines with a hole radiatively, a photon is emitted.

```
$direct
  character   required
  integer    integer
  double     optional
$end_direct
```

Explanation of specifiers.

\textbf{material-name}

- \textbf{type} \ character
- \textbf{presence} \ required
- \textbf{value} \ e.g. \ GaAs

Name of material to which this set of parameters applies. Name has to be listed in $default-materials$.

\textbf{number-of-parameters}

- \textbf{type} \ integer
- \textbf{presence} \ required
- \textbf{value} \ e.g. \ 1

Control parameter if the number of parameters provided is the same as demanded.

\textbf{C-opt}

- \textbf{type} \ double
- \textbf{presence} \ required
- \textbf{value} \ e.g. \ 7.2e-10
- \textbf{unit} \ [cm^3/s]

The parameters are specified as shown in the tables above. The order of magnitude is around \( 10^{-10} \) [cm^3/s].

\textbf{bow-C-opt}

- \textbf{type} \ double
- \textbf{presence} \ optional
- \textbf{value} \ e.g. \ 0.0
- \textbf{unit} \ [cm^3/s]

For ternary alloys there are also bowing parameters possible. \( bow-C-opt = 0.0 \) means zero bowing, i.e. linear interpolation is used.

Example
There is also a keywords section in the input file for $direct-recombination which you can use to overwrite default material parameters.

$SRH-recombination

The generation/recombination process can be assisted by impurities. This is modeled by the Shockley-Read-Hall model (SRH). The recombination/generation rates depend on the deviation of the carrier concentration from the equilibrium value and the scattering rates depend on the doping concentration.

$$R_{SRH} = \frac{pn-n_i^2}{\tau_p(n+n_i)+\tau_n(p+n_i)}$$

where

$$\tau_{pa} (N_D + N_A) = \frac{\tau_{p0}\tau_{n0}}{1+\frac{N_D}{N_{n,ref}}}$$

and $n$ is the electron density, $p$ is the hole density and $n_i$ is the intrinsic density. $N_D$ and $N_A$ are the donor and acceptor concentrations, respectively.

$N_{n,ref}$ and $N_{p,ref}$ are the reference doping concentrations for electrons and holes, respectively.

$\tau_{p0}$ and $\tau_{n0}$ are the zero doping scattering times for electrons and holes, respectively.

Explanation of specifiers.

**material-name**

- **type** character
- **presence** required
- **value** e.g. GaAs

Name of material to which this set of parameters applies. Name has to be listed in $default-materials.$
number-of-parameters
  type  integer
  presence  required
  value  e.g. 4

Control parameter if the number of parameters provided is the same as demanded.

There are two sets of parameters, one for electrons (n) and one for holes (p).

n-N-ref
  type  double
  presence  required
  value  e.g. 7.1e15
  unit  [cm^-3]

The $N_{a,ref}$ parameter for electrons as specified in the equation above.

n-tau
  type  double
  presence  required
  value  e.g. 4.26e-4
  unit  [s]

The $\tau_n$ parameter for electrons as specified in the equation above.

n-bow-N-ref
  type  double
  presence  optional
  value  e.g. 0.0
  unit  [cm^-3]

For ternary alloys there are also bowing parameters possible. $n\text{-bow-N-ref} = 0.0$ means zero bowing, i.e. linear interpolation is used.

p-N-ref
  unit  [cm^-3]

Same as n-N-ref but for holes.

p-tau
  unit  [s]

Same as n-tau but for holes.

p-bow-N-ref
  unit  [cm^-3]
Same as \textit{n-bow-N-ref} but for holes.

\textbf{p-bow-tau}

\textbf{unit} $[s]$

Same as \textit{n-bow-tau} but for holes.

Example

\begin{verbatim}
$SRH
$mobility-model-arora
$mobility-model-constant
$end_SRH
\end{verbatim}

There is also a keywords section in the input file for \textit{$SSRH$-recombination} which you can use to overwrite default material parameters.

\textbf{$mobility-model-arora$}

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parset/database/mobility-model-arora.htm

\textbf{$mobility-model-constant$}

The constant mobility model is due to lattice scattering (phonon scattering) and depends only on the temperature. The lattice atoms oscillate about their equilibrium sites at finite temperature leading to a scattering of carriers which results in a temperature dependent mobility $\mu_{\text{const}}$. $\mu_L$ is the mobility due to bulk phonon (lattice) scattering. For all semiconductors the temperature dependent lattice mobility is modeled by a power law. The parameter values used in this model for electrons and holes, respectively, are taken from the PhD thesis of V. Palankovski, \textit{Simulation of Heterojunction Bipolar Transistors (TU Vienna)}.

This model is suited for undoped structure.

\textbf{Note:} The $\gamma$ exponents $n$-\textit{gamma-lattice-temp}, $p$-\textit{gamma-lattice-temp} have opposite sign in both the PhD thesis of V. Palankovski and in the MINIMOS documentation compared to the implementation of \textit{nextnano}$^3$. 

344 Chapter 9. \textit{nextnano}$^3$
In this model the mobility is constant and depends only on the temperature $T$. The parameters in the database are given for electrons and holes,

$$\mu_{\text{const}}(T) = \mu_L \left( \frac{T}{T_0} \right)^{-\gamma},$$

where $T_0 = 300$ K.

<table>
<thead>
<tr>
<th>$\text{mobility}$</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-mu-lattice-temp</td>
<td>double optional</td>
</tr>
<tr>
<td>$n$-gamma-lattice-temp</td>
<td>double optional</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{mobility}$</th>
<th>optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-mu-lattice-temp</td>
<td>double optional</td>
</tr>
<tr>
<td>$n$-gamma-lattice-temp</td>
<td>double optional</td>
</tr>
</tbody>
</table>

Explanation of specifiers.

**material-name**

- **type** character
- **presence** required
- **value** e.g. Si

Name of material to which this set of parameters applies. Name has to be listed in `default-materials`.

**number-of-parameters**

- **type** integer
- **presence** required
- **value** e.g. 4

Control parameter if the number of parameters provided is the same as demanded.

There are two sets of parameters, one for electrons ($n$) and one for holes ($p$).

**n-mu-lattice-temp**

- **type** double
- **presence** required
- **value** e.g. 1417
- **unit** [cm$^2$/Vs]

$\mu_L,n$ is the bulk phonon mobility for electrons.

**n-gamma-lattice-temp**

- **type** double
- **presence** required
- **value** e.g. 2.5
- **unit** [ ]
\( \gamma_n \) is the exponent of the temperature dependence for electrons.

**Bowing parameters**

**n-bow-mu-lattice-temp**
- **type** double
- **presence** optional
- **value** e.g. 0.0
- **unit** \( [\text{cm}^2/\text{Vs}] \)

For ternary alloys there are also bowing parameters possible. \( n\text{-bow-mu-lattice-temp} = 0.0 \) means zero bowing, i.e. linear interpolation is used.

**n-bow-gamma-lattice-temp**
- **type** double
- **presence** optional
- **value** e.g. 0.0
- **unit** \( [\text{}] \)

**p-mu-lattice-temp**
- **unit** \( [\text{cm}^2/\text{Vs}] \)

Same as **n-mu-lattice-temp** but for holes, \( \mu_{L,p} \).

**p-gamma-lattice-temp**
- **unit** \([\text{}]\)

Same as **n-gamma-lattice-temp** but for holes, \( \gamma_{L,p} \).

**p-bow-mu-lattice-temp**
- **unit** \( [\text{cm}^2/\text{Vs}] \)

Same as **n-bow-mu-lattice-temp** but for holes.

**p-bow-gamma-lattice-temp**
- **unit** \([\text{}]\)

Same as **n-bow-gamma-lattice-temp** but for holes.

Example

```plaintext
$mobility

  name parameters temp     ! [cm^2/Vs] PhD thesis V.
--Palankovski
   n-bow-mu-lattice-temp   2.2     ! [{}] PhD thesis V.
   n-bow-gamma-lattice-temp 0.9     ! [{}] PhD thesis V.
--Palankovski but opposite sign compared to MINIMOS
```

(continues on next page)
For the Darwish mobility model, see [DarwishIEEE1997] and [Klaassen1992].

$\textbf{\$mobility-model-dar}$

For the Darwish mobility model, see [DarwishIEEE1997] and [Klaassen1992].
(continued from previous page)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Bowing parameter</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>Bowing parameter</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
Note: The mobility models `mobility-model-dar` (Darwish) has been developed for silicon only.

Note: In `nextnano` we use the nominal dopant concentration as specified in the input file and not the ionized one.

Example

```plaintext
$mobility

ame
parameters 70
B 3.61e7 ! [cm/s]
```

(continues on next page)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.70e4</td>
<td>(cm^2/Vs) (V/cm)^1/2</td>
<td></td>
</tr>
<tr>
<td>9.68e16</td>
<td>1/cm^3</td>
<td></td>
</tr>
<tr>
<td>0.72169</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.19778</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.41372</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7643</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.0e3</td>
<td>V/cm</td>
<td></td>
</tr>
<tr>
<td>3.828</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7643</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.5502</td>
<td>cm^2/Vs</td>
<td></td>
</tr>
<tr>
<td>470.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.19778</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.01552</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.80618</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.495</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.85e-21</td>
<td>V/s</td>
<td></td>
</tr>
<tr>
<td>0.89233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.41372</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.0e3</td>
<td>V/cm</td>
<td></td>
</tr>
<tr>
<td>4.0e20</td>
<td>1/cm^3</td>
<td></td>
</tr>
<tr>
<td>1.51e7</td>
<td>cm/s</td>
<td></td>
</tr>
<tr>
<td>4.18e3</td>
<td>(cm^2/Vs) (V/cm)^1/2</td>
<td></td>
</tr>
<tr>
<td>0.0119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>470.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.23e17</td>
<td>1/cm^3</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.719</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.495</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.828</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.97e13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.36e20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.89233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.41372</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.19778</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.28227</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.005978</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.80618</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.89233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.36e7</td>
<td>cm^2/Vs</td>
<td></td>
</tr>
<tr>
<td>2.495</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.828</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For the Lombardi mobility model, see [LombardiIEEE1988].

$mobility-model-lom$

For the Lombardi mobility model, see [LombardiIEEE1988].
Note: The mobility models `mobility-model-lom` (Lombardi) has been developed for silicon only.

Note: In nextnano we use the nominal dopant concentration as specified in the input file and not the ionized one.

Example

```plaintext
[---------------------------------------]
$mobility
[---------------------------------------]
```

(continues on next page)
### mobility-model-masetti

The documentation for this keyword is available here: [https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-masetti.htm](https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-masetti.htm)
$\textbf{mobility-model-minimos}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-minimos.htm

$\textbf{mobility-model-simba}$

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/mobility-model-simba.htm

$\textbf{mobility-models}$

Several mobility models can be chosen that consider phonon scattering, impurity scattering and electric field dependence.

\begin{verbatim}
$\textbf{mobility}$  required
<table>
<thead>
<tr>
<th>character</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>required</td>
</tr>
</tbody>
</table>
$\textbf{end.mobility}$
\end{verbatim}

\textbf{Note:} The mobility models mobility-model-lom (Lombardi) and mobility-model-dar (Darwish) have been developed for silicon only.

\textbf{Example}

\begin{verbatim}
!--------------------------------------------------------!
$\textbf{mobility}$  
| name     | 0  | SIMBA, parallel E field dependence according to model 0 |
| number   | 1  |

| name     | 1  | SIMBA, parallel E field dependence according to model 1 |
| number   | 2  |

| name     | 2  | SIMBA, parallel E field dependence according to model 2 |
| number   | 3  |

| name     | 3  | SIMBA, parallel E field dependence according to model 3 |
| number   | 4  |

| name     | 4  | SIMBA, parallel E field dependence according to model 4 |
| number   | 5  |

| name     | 5  | SIMBA, parallel E field dependence according to model 5 |
| number   | 6  |

| name     | 6  | with perpendicular E field dependence |
| number   | 7  |

| name     | 7  | with perpendicular E field dependence |
| number   | 8  |
\end{verbatim}

(continues on next page)
$transport-models

Charge carrier models.

$transport

type character required
model-name character required
presence integer required
value simple-drift-model

Transport model for drift-diffusion equation.

model-name

type character
presence required
value simple-drift-model
More information available here: $simple-drift-models

model-type-number

  type integer
  presence required
  value 1

Only 1 model implemented so far.

Example

```plaintext
$transport
  name model-type-number
  integer presence required
  value 1
$end_transport

$buffer-constant-A(T)

Constant $A(T)$ used for buffer calculations: The p$K_a$ value depends on the ionic strength.

```plaintext
$buffer
  constant-A(T) optional
  constant-C optional
c constate-dto-Kelvin double required
$end_buffer

Example

```plaintext
$buffer
  constant-A(T) Kelvin 273.15 ! Kelvin = Celsius + 273.15
$end_buffer

The left column of the specifier $T_A(T)$ contains the temperature in degrees of Centigrade (Celsius) between 0° C and 100° C.

The right column of the specifier $T_A(T)$ contains the corresponding value of the constant A as a function of temperature T, i.e. $A(T)$.
The values are taken from page 30 of [Beynon1996].
They can also be approximated by a second-order polynomial ([Beynon1988]):
\[ A(T) = 0.4918 + 0.0006614T + 0.000004975T^2 \]
If the keyword $buffer-constant-A(T)$ is present in the input file, the values for this keyword in the database are overwritten.

**Physical significance of this parameter**
The ionic strength of an electrolyte influences the p\(K_a\) value of the buffer. This dependence can be described by the following equation (sometimes known as the Debye-Hückel relationship) where the constant \(A(T)\) enters.

\[
pK_a' = pK_a + (2z_a - 1)[AI^{1/2}/(1 + I^{1/2}) - 0.1I]
\]
where \(I\) is the ionic strength and \(z_a\) is the charge on the conjugate acid species. \(pK_a'\) is the modified \(pK_a\) value.

The value of \(A\) (sometimes called Debye-Hückel parameter) is about 0.5 but it is temperature dependent.

Internally, the program takes the temperature \(T_0\) that is given in the input file under the keyword $global-parameters$ (in units of Kelvin) and interpolates linearly between the two appropriate neighboring \(A(T)\) values to find the value for \(A(T_0)\). The conversion between temperature in Kelvin and Centigrade is done by the constant:

\[
constant-Centigrade-to-Kelvin = 273.15
\]

**Example**

<table>
<thead>
<tr>
<th>Lattice-temperature (°C)</th>
<th>A(T) (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.4989</td>
</tr>
<tr>
<td>20</td>
<td>0.5070</td>
</tr>
</tbody>
</table>

=> Internally the program calculates the value for \(A(T = 15° C)\) = \((0.4989 + 0.5070)/2 = 0.50295\).

The following interpolation formula is used:

\[
A(T) = A(T_i) + \text{slope} \times (T - T_i)
\]

where

\[
\text{slope} = \frac{A(T_i+1) - A(T_i)}{T_i+1 - T_i}
\]

and it holds:

\[
T_i+1 \text{ and } T_i \text{ are the closest temperature points above and below the specified temperature lattice-temperature.}
\]

- If the lattice-temperature is smaller than the smallest value of \(A(T)\), the smallest \(A(T)\) value is taken.
- If the lattice-temperature is larger than the largest value of \(A(T)\), the largest \(A(T)\) value is taken.

The value of \(A\) always depends on temperature. This can only be switched off by specifying only one value of \(T\) and \(A(T)\) in the database or in the input file. The values for \(T\) and \(A(T)\) that are specified in the database can be overwritten in the input file. For details, have a look at the input file keyword $buffer-constant-A(T)$.
$buffer-solutions

The documentation for this keyword is available here: https://www.nextnano.com/nextnano3/input_parser/database/buffer-solutions.htm

$physical-constants

The base system for units is SI.

The following physical constants are used within the nextnano³ code.

```plaintext
$physical
required
double required ! [As] = [C] ~
--e: elementary charge

double required ! [kg] ~
--m0: electron mass
double required ! [Js] h_
--bar: Planck’s constant
double required ! [m/s] (exact) ~
--c: speed of light in vacuum
double required ! [J/K] k_
--B: Boltzmann constant
double required ! [As/Vm] = [F/m] (exact) ~
--epsilon0: electric constant
double required ! [1/mol] N_
--A: Avogadro number
$end_physical
```

Example

```
!-----------------------------------------!
$physical
charge = -1.6021766208e-19 ! [C] = [As] -1.
--constant = 6.626070040e-34 ! [Js] 6.
--light = 2.99792458e+8 ! [m/s] (exact)
--permittivity = 8.854187817e-12 ! [As/Vm] (exact) 8.854187817.
--number = 6.022140857e+23 ! [] 6.
$end_physical
```

These SI units were taken on 2019-05-10 from https://physics.nist.gov/cuu/Constants/index.html. The number in parentheses is the numerical value of the standard uncertainty referred to the corresponding last digits of the quoted result.

Further constants

- reduced_planck_constant (Planck constant over 2 pi) is calculated internally inside the program:
  \[
  \text{planck-constant}/(2\pi) = \hbar/2\pi
  \]

```
reduced_planck_constant = 1.054571800139113e-034 ! [Js]
  --(calculated internally from other constants)
  1.054571800(13) e-34 ! [Js]
  --(NIST)
```

- bohr_radius is calculated internally inside the program:
**Derived constants**

- \([h_{\text{bar}}^2/(2m_0)]\)
  
  \[
  \text{h2b2m}_m2 = 6.104264214606464 \times 10^{-39} \quad \text{[m]}
  \]
  
  \[
  \text{h2b2m}_eVAA2 = 3.27.2113860281805 \quad \text{[eV]}
  \]

- From the Boltzmann constant \(k_B\), one obtains \(k_B T\) at room temperature in units of \([\text{eV}]\):
  
  \[
  k_B = 0.0258519910 \quad \text{[eV]} \quad (\text{[K]} \quad 298.15 \quad \text{[THz]} \quad 300)
  \]

For input scaling factors, see \(\text{input-scaling-factors}\).

**Conversion factors**

- \(\mu m \leftrightarrow \text{eV}: h \cdot c / e \cdot 10^9 = 1.23984197\)
  
  Example
  
  \[
  1.23984 / 8.4 \mu m = 0.1476 \text{ eV}
  \]
  
  \[
  1.23984 / 0.1476 \text{ eV} = 8.4 \mu m
  \]

- \(\mu m \leftrightarrow \text{THz}: c \cdot 10^6 = 299.792458\)
  
  Example
  
  \[
  299.79 / 8.4 \mu m = 35.69 \text{ THz}
  \]
  
  \[
  299.79 / 35.69 \text{ THz} = 8.4 \mu m
  \]
• $10^{18}$ cm$^{-3}$ $\leftrightarrow$ M: 602.21415

Example

\[
\frac{30.11}{602.2} = 0.050 \text{ M}
\]

\[
0.050 \text{ M} \times 602.2 = 30.11
\]

• $10^{18}$ cm$^{-3}$ $\leftrightarrow$ mM: 0.60221415

Example

\[
\frac{30.11}{0.6022} = 50 \text{ mM}
\]

\[
50 \text{ mM} \times 0.6022 = 30.11
\]

$\text{input-scaling-factors}$

The base system for units is SI. However, most input quantities are assumed to be given in scaled form. For physical constants, see $\text{physical-constants}$.

```
$\text{input}$
```

<table>
<thead>
<tr>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
</tbody>
</table>

```
$\text{end_input}$
```

Example

```
$\text{input}$
```

<table>
<thead>
<tr>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
<tr>
<td>double required</td>
</tr>
</tbody>
</table>

```
$\text{end_input}$
```

(continues on next page)
! spin-orbit coupling energies,\( \langle \rangle \)  
\( k_p^2_wz = 1.6021766208e-19 \)  
\( \rightarrow \) deformation potentials [eV]  
\( k_p^2A_wz = 1.0 \)  
\( \rightarrow \) k.p parameter A1,A2,A3,A4,A5,A6 \( \langle \rangle \), multiplied internally with \( [h_bar^2/(2*m0)] \)  
\( k_p^2B_wz = 6.104264214606464e-39 \)  
\( \rightarrow \) k.p parameter B1,B2,B3 \( [h_bar^2/(2*m0)] \) \( [J/m^2] \)  

It is assumed that input numbers are scaled to these units.

**Note:** \( [h_bar^2/(2*m0)] = kp_k^2_zb = kp_k^2B_wz \)

### $global-parameters

Global parameters are general parameters which are valid all over the device. See also $global-parameters in Keywords section.

<table>
<thead>
<tr>
<th>$global</th>
<th>double</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$end_global</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Lattice temperature**

**lattice-temperature**

**type** double  
**presence** required  
**unit** [K]  
**value** e.g. 300.0

**Note:** The band gaps, i.e. the conduction band edges, are adjusted when the temperature changes. This feature can be switched off.

See $numeric-control: varshni-parameters-on

In addition, the lattice constants depend on temperature. This can be switched off as well.

See $numeric-control: lattice-constants-temp-coeff-on

Example

```plaintext
!-----------------------------------------!
$global                           
| temperature | 300.0 | ! 300 K  
$end_global                       
!-----------------------------------------!
```

9.3. Database
$program_restrictions

These are general restrictions to the program.

<table>
<thead>
<tr>
<th>$program_restrictions</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_dimension_domain</td>
<td>integer</td>
</tr>
<tr>
<td>max_dimension_domain</td>
<td>integer</td>
</tr>
<tr>
<td>minimum-grid-factor</td>
<td>double</td>
</tr>
<tr>
<td>maximum-grid-factor</td>
<td>double</td>
</tr>
<tr>
<td>delta-to-treat-grid-factor-as-one</td>
<td>double</td>
</tr>
</tbody>
</table>

Settings for domain dimensions.

**min_dimension_domain**

- **type**: integer
- **presence**: required
- **value**: 1

The minimum is 1, i.e. for one-dimensional simulations. A value of 0 makes sense for bulk simulations. However, this is not supported yet.

**max_dimension_domain**

- **type**: integer
- **presence**: required
- **value**: 3

The maximum is 3, i.e. for three-dimensional simulations.

Settings for grid spacing.

**minimum-grid-factor**

- **type**: double
- **presence**: required
- **value**: 0.1

Be careful, geometric series grow up very rapidly.

**maximum-grid-factor**

- **type**: double
- **presence**: required
- **value**: 10.0

Be careful, geometric series grow up very rapidly.

**delta-to-treat-grid-factor-as-one**

- **type**: double
- **presence**: required
value 1.0e-3
To avoid numerical inaccuracies, everything in the interval 1.0 ± Δ is treated as 1.0.

minimum-grid-width
  type double
  presence required
  value 0.01

Only a guess, if exceeded a warning message is printed to standard output. Currently, this value is irrelevant and is ignored in the program, i.e. no warning message is printed.

maximum-grid-width
  type double
  presence required
  value 200.0

Only a guess, if exceeded a warning message is printed to standard output. Currently, this value is irrelevant and is ignored in the program, i.e. no warning message is printed.

maximum-number-of-grid-points
  type integer
  presence required
  value 1000000 = 100 * 100 * 100

Only a guess, if exceeded a warning message is printed to standard output.

lowest-temperature-allowed
  type double
  presence required
  value 0.0

Note: Convergence is more difficult to achieve if the temperature is set to a very small value or to even 0.0 [K].

Example

```
!-----------------------------------------------!
$program_restrictions
  min_dimension_domain = 1
  max_dimension_domain = 3
  min_grid-factor = 0.1
  max_grid-factor = 10.0
  minimum-grid-width = 1.0e-3
  minimum-grid-width = 0.01
  minimum-grid-width = 200.0
  maximum-number-of-grid-points = 1000000
  lowest-temperature-allowed = 0.1 [K]
  lowest-temperature-allowed = 0.0 [K]
$end_program_restrictions
!-----------------------------------------------!
```

9.3. Database 363
$input_filename

Only to specify input file name.

Any keyword with valid syntax. Must be set to optional. The first string in the third line is the name of the input file.

$domain-coordinates-defaults

Specify axis for growth direction with respect to simulation coordinate system.

Define the possible values for growth-coordinate-axis in keyword $domain-coordinates. They are only relevant for

Example

$zb-default-expectations

Number of expected parameters for zinc blende materials.

Specify here how many parameters are expected.

number-of-8x8kp-parameters

  type integer
We need 6 parameters for the 8-band $k \cdot p$ model in zinc blende.

\textbf{number-of-6x6kp-parameters}

\begin{itemize}
  \item \textbf{type} integer
  \item \textbf{presence} required
  \item \textbf{value} 6
\end{itemize}

We need 4 parameters for the 6-band $k \cdot p$ model in zinc blende.

\textbf{number-of-uniax-vb-def-pots}

\begin{itemize}
  \item \textbf{type} integer
  \item \textbf{presence} required
  \item \textbf{value} 2
\end{itemize}

Example

\begin{verbatim}
1------------------------------------------------------------!
$zb          6                                          !
  number_of_parameters       6                              !
  number_of_kp_parameters    4                              !
  number_of_uniax_vb_pots    2                              ! b, d
$end_zb                        !
1------------------------------------------------------------!
\end{verbatim}

\textbf{Swz-default-expectations}

Number of expected parameters for wurtzite materials.

\begin{verbatim}
$swz [-----------------------------]
  number_of_8x8kp_parameters   integer required
  number_of_6x6kp_parameters   integer required
  number_of_uniax_vb_pots      integer required
$end_swz                      !
\end{verbatim}

Specify here how many parameters are expected.

\textbf{number-of-8x8kp-parameters}

\begin{itemize}
  \item \textbf{type} integer
  \item \textbf{presence} required
  \item \textbf{value} 13
\end{itemize}

We need 13 parameters for the 8-band $k \cdot p$ model in wurtzite.

\textbf{number-of-6x6kp-parameters}

\begin{itemize}
  \item \textbf{type} integer
  \item \textbf{presence} required
  \item \textbf{value} 9
\end{itemize}

We need 9 parameters for the 6-band $k \cdot p$ model in wurtzite.

\textbf{number-of-uniax-vb-def-pots}
We need 6 parameters for the uniaxial valence band deformation potentials in wurtzite (6 numbers usually: d1, d2, d3, d4, d5, d6).

Example

```
!---------------------------------------!
$zb
! parameters 13
parameters 9
pots 6  ! d1, d2, d3, d4, d5, d6
$end_zb
!---------------------------------------!
```

**$zb-restrictions**

Some restrictions apply for zinc blende materials.

```
$zb
integer  required
integer_array  required
integer_array  required
double_array  required
integer_array  required
integer_array  required
double_array  required
integer  required
integer  required
integer  required
$end_zb
```

Explanations

**miller-size**

type integer

presence required

value 3

There are three Miller indices altogether that define the (hkl) plane. Coincidently, in zincblende the [hkl] vector is perpendicular to the (hkl) plane.

**miller-default-direction-of-x**

type integer_array

presence required

Three-digit Miller indices of the (hkl) plane perpendicular to x axis of simulation coordinate system.

**miller-default-direction-of-y**

type integer_array

presence required

Three-digit Miller indices of the (hkl) plane perpendicular to y axis of simulation coordinate system.
This corresponds to x axis and y axis, respectively, in simulation coordinate system, i.e. the x axis is perpendicular to the (1 0 0) plane, i.e. x axis is along the [100] direction in zinc blende, and the y axis is perpendicular to the (0 1 0) plane, i.e. y axis is along the [010] direction in zinc blende.

These value can be overwritten in $domain-coordinates (hkl-x-direction, hkl-y-direction, hkl-z-direction).

Direction cosines

```
direction-cosines
```

- **type**: double_array
- **presence**: required
- **value**: 0.0 0.0 0.0

Direction cosines between lattice vectors. \(g_1 \cdot g_2, g_2 \cdot g_3, g_1 \cdot g_3\) \(\ldots\) unit vectors in lattice directions.

Direction cosine refers to the cosine of the angle between any two vectors. Direction cosines are useful for forming direction cosine matrices that express one set of orthonormal basis vectors in terms of another set, or for expressing a known vector in a different basis. For zinc blende, we use:

\[
g_{ik} = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

Three-digit Miller indices of the \((hkl)\) plane:

- **miller-direction-of-cx**
  - **type**: integer_array
  - **value**: 1 0 0

Corresponds to x axis in crystal coordinate system, i.e. the x axis is perpendicular to the (1 0 0) plane, i.e. x axis is along the [100] direction in zinc blende.

- **miller-direction-of-cy**
  - **type**: integer_array
  - **value**: 0 1 0

Corresponds to y axis in crystal coordinate system, i.e. the y axis is perpendicular to the (0 0 1) plane, i.e. y axis is along the [010] direction in zinc blende.

- **miller-direction-of-cz**
  - **type**: integer_array
  - **value**: 0 0 1

Corresponds to z axis in crystal coordinate system, i.e. the z axis is perpendicular to the (0 0 1) plane, i.e. z axis is along the [001] direction in zinc blende.

These are the default orientations.

- **lattice-constants-for-cxyz**
  - **type**: double_array
  - **value**: 1.0 1.0 1.0

9.3. Database 367
Lattice constants to interpret the Miller indices.

Define how the bands are labeled.

**gamma-is-cb-number**

```
  type integer
  value 1
```

The conduction band minimum at the Gamma point has the number 1.

**l-is-cb-number**

```
  type integer
  value 2
```

The conduction band minimum at the L point has the number 2.

**x-is-cb-number**

```
  type integer
  value 3
```

The conduction band minimum at the X point (or Delta in Si or Ge) has the number 3.

Example

```
!--------------------------------------------------!
$zb
  size 3
  x 1 0 0
  y 0 1 0
  cosines 0.0 0.0 0.0 ! g1*2, g2*3, g3*3,
  ! g1 ... unit vectors in lattice directions
  ex 1 0 0
  ey 0 1 0
  ez 0 0 1
  xyz 1.0 1.0 1.0
  number 1
  number 2
  number 3
$end_zb
!--------------------------------------------------!
```

**$wz-restrictions**

Some restrictions apply for wurtzite materials.

```
$wz
  integer required
  integer_array required
  integer_array required
  double_array required
  integer_array required
  integer_array required
  double_array required
$end_wz
```

Explanations

**miller-size**

```
  type integer
```
There are four Miller-Bravais indices altogether that define the \((hkil)\) plane. 

**Note:** They do not define the [hkil] direction.

Usually for wurtzite, the four-digit Miller-Bravais indices \((h k i l)\) are used.

\[
miller-default-direction-of-x
\]

**type** integer_array  
**presence** required 

Four-digit Miller indices of the \((hkil)\) plane perpendicular to x axis of simulation coordinate system.

\[
miller-default-direction-of-y
\]

**type** integer_array  
**presence** required 

Four-digit Miller indices of the \((hkil)\) plane perpendicular to y axis of simulation coordinate system.

This corresponds to x axis and y axis, respectively, in simulation coordinate system, i.e. the x axis is perpendicular to the \((1 0 -1 0)\) plane and the y axis is perpendicular to the \((-1 2 -1 0)\) plane, respectively.

These value can be overwritten in $domain-coordinates (hkil-x-direction, hkil-y-direction, hkil-z-direction).

**Note:** It holds for the four-digit Miller-Bravais indices \((h k i l)\): \(i = -(h + k)\), i.e. \(i\) is not independent.

Direction cosines

\[
direction-cosines
\]

**type** double_array  
**presence** required 

**value** -0.5 0.0 0.0 

Direction cosines between lattice vectors. \(g1*g2, g2*g3, g1*g3\) \(g_i\ldots\) unit vectors in lattice directions.

Direction cosine refers to the cosine of the angle between any two vectors. Direction cosines are useful for forming direction cosine matrices that express one set of orthonormal basis vectors in terms of another set, or for expressing a known vector in a different basis. For wurtzite, we use:

\[
g_{ik} = (1 -0.5 0) \\
(0 0 1)
\]

Four-digit Miller indices of the \((hkil)\) plane:

\[
miller-direction-of-cx
\]

**type** integer_array  
**value** 1 0 -1 0 

Corresponds to x axis in cartesian crystal coordinate system, i.e. the x axis is perpendicular to the \((1 0 -1 0)\) plane.

\[
miller-direction-of-cy
\]
type integer_array
value -1 2 -1 0

Corresponds to y axis in cartesian crystal coordinate system, i.e. the y axis is perpendicular to the (-1 2 -1 0) plane.

miller-direction-of-cz

type integer_array
value 0 0 0 1

Corresponds to z axis in cartesian crystal coordinate system, i.e. the z axis is perpendicular to the (0 0 0 1) plane, i.e. axis parallel to sixfold rotational axis in wurtzite which is coincidently also the [0001] direction.

These are the default orientations.

lattice-constants-for-cxyz

type double_array
value 1.0 1.0 1.6329931618554520654648560498039

Lattice constants to interpret the Miller-Bravais indices: 1.0, 1.0, \( \sqrt{3} \). Here, we take the ideal wurtzite ratio of \( c/a = \sqrt{3}/2 \).

In wurtzite, there are three coordinate axis in the basal plane, \( a_1, a_2, a_3 \), and the c direction perpendicular to it. There are different definitions for it.

- \( a_1 = [10\bar{1}0], a_2 = [-1\bar{2}\bar{1}0], a_3 = [...] \), \( c = [0001] \) (used by nextnano3) \( \Rightarrow a_1 = \sqrt{3}/2 ax - a/2 y, a_2 = ay, c = cz \)
- \( a_1 = [2\bar{1}10], a_2 = [-1\bar{2}\bar{1}0], a_3 = [-1\bar{1}20], c = [0001] \) \( \Rightarrow a_1 = [2\bar{1}10] a/\sqrt{6}, a_2 = [-1\bar{2}\bar{1}0] a/\sqrt{6}, c = [0001] c = [0, 0, 0, 3 \lambda ] a/\sqrt{6}, \) where \( \lambda = \sqrt{2/3} c/a \).

Example

```
$begin_wz
$end_wz
```

Nextnano Manual, Release latest

Chapter 9. nextnano3
$region-default

Default priority in regions.

<table>
<thead>
<tr>
<th>region</th>
<th>integer</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Minimum priority.

$region

<table>
<thead>
<tr>
<th>region</th>
<th>integer</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$end_region

Minimum priority.

$region

<table>
<thead>
<tr>
<th>region</th>
<th>integer</th>
<th>presence</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$end_region

Change only with care. 0 is a good idea. A negative value might also work.

Example

!----------------------------!
$region
        priority = 0
$end_region
!----------------------------!

$known-doping-functions

Several doping functions are possible. For more information see $doping-function.

$known

<table>
<thead>
<tr>
<th>known</th>
<th>integer</th>
<th>character</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$end_known

Example

!------------------------------------!
$known
        number = 1
        name = constant doping profile

        number = 2
        name = Gaussian shaped doping profile

        number = 3
        name = linear doping profile

$end_known
!------------------------------------!

9.3. Database
Several functions are possible. For more information see $alloy-function.

$known-function-names

Alloy functions.

function-number

  type  integer
  presence  required
  value  1, 2, ...

Dense numbering of integers, starting from 1, 2, 3, ...

function-name

  type  character
  presence  required
  value  constant, linear, import-alloy-profile, alloy-profile-defined-by-function,... (see below)

Name of alloy profile.

Example

(continues on next page)
**9.3. Database**

(continued from previous page)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

$\text{end\_known}$

---------------------------------------------
Several impurity types are possible.

$\text{known-impurity-types}$

Example

```bash
$\text{known}$

\begin{verbatim}
\text{name } \text{number} \text{name } \text{number} \text{name } \text{number}
\end{verbatim}

$\text{end_known}$

$\text{interface-state-limitations}$

Several doping functions are possible. For more information see $\text{doping-function}$.

Example

```bash
$\text{interface}$

\begin{verbatim}
\text{name } \text{charge density due to trap states}
\text{name } \text{fixed charge density}
\text{name } \text{charge density due to electrolyte (site-binding model)}
\text{name } \text{charge density due to gas adsorption}
\end{verbatim}

$\text{end_interface}$
```
$\text{quantum-model-electrons}$

Several quantum models for the electrons are possible.

Example

```
$\text{quantum-model-electrons}$

<table>
<thead>
<tr>
<th>name</th>
<th>character required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>integer required</td>
</tr>
</tbody>
</table>

$\text{end_quantum}$
```

$\text{quantum-model-holes}$

Several quantum models for the holes are possible.

Example

```
$\text{quantum-model-holes}$

<table>
<thead>
<tr>
<th>name</th>
<th>character required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>integer required</td>
</tr>
</tbody>
</table>

$\text{end_quantum}$
```

$\text{separation-models-electrons}$

More details about the separation model can be found in the glossary.

Example

```
$\text{separation-models-electrons}$

<table>
<thead>
<tr>
<th>name</th>
<th>character required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>integer required</td>
</tr>
</tbody>
</table>

$\text{end_separation}$
```

(continues on next page)
More details about the separation model can be found in the glossary.

Example

$end_method

More details about the method of Brillouin zone integration can be found in the glossary.

Example

$end_method
$k$-range-determination-methods

This makes only sense for $k \cdot p$ calculations in 1D ($k_{||} = (k_x, k_y)$) and 2D ($k_{||} = (k_z)$) but not in 3D.

1. Solve Schrödinger equation for $(k_x, k_y) = (0, 0)$.
2. Define a set of $k_{||}$ that one needs and solve $k \cdot p$ Schrödinger equation for every $k_{||}$.

Two models are supported.

<table>
<thead>
<tr>
<th>model-name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bulk-dispersion-analysis</td>
<td>1</td>
</tr>
</tbody>
</table>

Here, the range for $k_{||}$ is determined automatically by the program using the bulk energy dispersion $E(k)$. More information...

<table>
<thead>
<tr>
<th>model-name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-max-input</td>
<td>2</td>
</tr>
</tbody>
</table>

A maximum value $k_{\text{max}}$ of $k_{||}$ has to be specified in the input file.

Example

```plaintext
$\text{-----------------------------}!
| $k$-range-definition             |
| model-name                        |
| value bulk-dispersion-analysis                |
| model-type-number                  |
| value 1                              |

$\text{-----------------------------}!
| $k$-range-definition             |
| model-name                        |
| value $k$-max-input               |
| model-type-number                  |
| value 2                              |
```

$tight-binding-parameters

This part is preliminary. It is not working yet.
### Interaction between s and p orbitals

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>spin orbit (p)</td>
<td>double</td>
<td>optional ! ion 2</td>
</tr>
<tr>
<td>spin orbit (d)</td>
<td>double</td>
<td>optional ! ion 2</td>
</tr>
</tbody>
</table>

! Interaction between s and d orbitals (in our case: zero)

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>onsite (s) [eV]</td>
<td>integer</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>onsite (p) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>onsite (s*) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>onsite (d) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>onsite (s) [eV]</td>
<td>double</td>
<td>optional ! ion 2 _</td>
</tr>
<tr>
<td>onsite (p) [eV]</td>
<td>double</td>
<td>optional ! ion 2 _</td>
</tr>
<tr>
<td>onsite (s*) [eV]</td>
<td>double</td>
<td>optional ! ion 2 _</td>
</tr>
<tr>
<td>onsite (d) [eV]</td>
<td>double</td>
<td>optional ! ion 2 _</td>
</tr>
</tbody>
</table>

! Tight-binding couplings

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>onsite (s) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (p) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (s*) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (d) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (s) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (p) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (s*) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
<tr>
<td>onsite (d) [eV]</td>
<td>double_array</td>
<td>required ! [eV],[]</td>
</tr>
</tbody>
</table>

! Spin orbitals

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>spin orbit (s) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>spin orbit (p) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>spin orbit (s*) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
<tr>
<td>spin orbit (d) [eV]</td>
<td>double</td>
<td>required ! ion 1 _</td>
</tr>
</tbody>
</table>

$end_tight$
Example

!----------------------------------------------!
! Tight-binding couplings
! Interaction between s and d orbitals (in our case: zero)

! States

<table>
<thead>
<tr>
<th>State</th>
<th>p</th>
<th>0.0195</th>
<th>Si spin orbit (p)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>d</td>
<td>0.0000</td>
<td>Si spin orbit (d)</td>
</tr>
</tbody>
</table>

! Interaction between s and p orbitals

! Tight-binding couplings

<table>
<thead>
<tr>
<th>Pair</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s</td>
<td>s</td>
<td>p</td>
<td>p</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>p</td>
<td>s</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>d</td>
<td>s</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>s</td>
<td>p</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>d</td>
<td>p</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>21</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>sss</td>
<td>-1.9413</td>
<td>3.6720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>2.7836</td>
<td>2.4880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-1.6933</td>
<td>0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-2.7998</td>
<td>1.8690</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>2.7836</td>
<td>2.4880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>4.1068</td>
<td>2.1870</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>2.8428</td>
<td>1.9190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-2.1073</td>
<td>1.8300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-1.6933</td>
<td>0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>2.8428</td>
<td>1.9190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-3.3081</td>
<td>0.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-0.7003</td>
<td>2.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-2.7998</td>
<td>1.8690</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-2.1073</td>
<td>1.8300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-0.7003</td>
<td>2.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-1.2327</td>
<td>2.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-1.5934</td>
<td>3.7110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>1.9977</td>
<td>2.0930</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>1.9977</td>
<td>2.0930</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>2.5145</td>
<td>2.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sss</td>
<td>-2.4734</td>
<td>2.0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.351259e-10 ! m -> A ! don't change! very sensitive

$end_tight

(continues on next page)
$\textbf{Warnings}$

Warnings can be switched on or off.

\begin{tabular}{ll}
\texttt{$\textbf{warnings}$} & \texttt{required} \\
\texttt{logical} & \texttt{required} \\
\texttt{$\textbf{end\_warnings}$} & \texttt{required} \\
\end{tabular}

Specify here if warnings should be written to the \texttt{.log} file.

\begin{verbatim}
warnings

  type logical
  presence required
  options .FALSE.

  default uses value specified in database_nn3.in
\end{verbatim}

Example

\begin{verbatim}
!---------------------------------------------------------------------!
$warnings
warnings = .FALSE.
$end_warnings
! switch off warnings

!---------------------------------------------------------------------!
\end{verbatim}

\section*{9.4 Command line arguments}

Command line usage:

\begin{verbatim}
nextnano3_Intel_64bit.exe --inputfile 1D_HEMT.in --database database_nn3.in --license License_nnp.lic --outputfolder 1D_HEMT/
\end{verbatim}

Currently nextnano$^3$ provides the following options:

\begin{itemize}
  \item \texttt{-i input\_file, --inputfile input\_file} Name of input file to be read in.
    \begin{itemize}
      \item \texttt{--inputfile 1D_GaAs_HEMT.in}
      \item \texttt{--inputfile "1D GaAs HEMT.in"}
    \end{itemize}
    Blanks in file names are supported if using quotation marks " ".
    Default value: Entry in keywords.val.

  \item \texttt{-d database\_file, --database database\_file} Name of material parameter database file to be read in.
    \begin{itemize}
      \item \texttt{--database database_modified_my_Si_parameters.in}
      \item --database "my database_nn3.in"
    \end{itemize}
\end{itemize}
Blanks in file names are supported if using quotation marks " ".

Default value: Entry in file `database_nn3_keywords.val`, i.e. `database_nn3.in`.

- ` license_file, --license license_file` Name of license file to be read in.

Example:

- `--license license.dat`
- `--license License_nnp.lic`
- `--license H:\Documents\nextnano\license\license.dat`
- `--license "H:\My Documents\nextnano\license\license.dat"

Default value: `License_nnp.lic`

- ` o output_directory, --outputdirectory output_directory` Directory name where to output results.

Example:

- `--outputdirectory my_output_foldername`
- `--outputdirectory my_output_foldername/`
- `--outputdirectory "my output folder/"
- `--outputdirectory "my output\my folder/"
- `--outputdirectory "<name_of_input_file>"
- `--outputdirectory "<name_of_input_file>\output"
- `--outputdirectory "E:\My Documents\nextnano_output\<name_of_input_file>"

In the last three examples the string `<name_of_input_file>` is provided and not the name of the input file. This special string `<name_of_input_file>` will be replaced automatically with the actual name of the input file.

Blanks in folder names are supported if using quotation marks " ".

- ` t i, --threads i` Number of parallel threads (i = integer) to benefit from parallelization on multi-core CPUs.

Example:

- `--threads 1 uses 1 thread (default)`
- `--threads 4 uses 4 threads (only works with an executable that had been compiled with OpenMP option)`
- `--threads 0 automatic decision, i.e. use value specified in input file, or use default value if not specified in input file`
This setting only affects parallelization of e.g. $k_{||}$ vectors or energy grid points (NEGF). Parallelization of calls to MKL are still parallelized automatically.

---

---debuglevel i

Generate additional debug information ($i = \text{integer}$).

Example:

```
--debuglevel 3
```

The integer number has the following meaning:

- $0$: no debug information at all (default)
- $1$: modest debug information
- $2$: more debug information
- $3$: even more debug information
- $n$: any other integer number is also possible, some have specific meanings (for developers only)
- $> 1000$: Here, the files keywords_nn3.xml and database_nn3_keywords_nn3.xml are created, which are used by nextnanomat for its auto completion feature.
- $< 0$: automatic decision, i.e. use value specified in input file, or use default value if not specified in input file

---

--log

Generate a *.log file of screen output (standard output).

The file will be written to: $<\text{outputdirectory}>/ <\text{inputfilename}> \_\log.log$

---

--cancel i

Automatically cancel simulation after $i$ minutes ($\text{kill}$, i.e. program stops immediately).

If value is $\leq 0$, the simulation is not canceled.

Example:

```
--cancel 10
```

---

--softkill i

Automatically cancel simulation after $i$ minutes ($\text{soft kill}$, i.e. program exits iteration cycle and writes output).

If value is $\leq 0$, the simulation is not canceled.

Example:

```
--softkill 10
```

Alternative option: If the user saves a file called SOFTWARE (without file extension) into the output folder of the currently running simulation, a soft kill will be performed.

---

--system operating_system

Flag indicating on which operating system the executable is executed.

Example:

```
--system default (default)
--system windows
--system linux
```
Available options are default, windows, linux, unix and mac.

When default is specified, the corresponding setting of your executable is applied.

Typically, it is not required to specify this flag as the executables had been compiled for the respective operating systems (and thus this setting has already been applied automatically).

Currently, the internal settings for linux, unix, mac are identical.

`--condor` Flag that is necessary if job is submitted to HTCondor.

`-p, --parse` Parse input file and quit.

`--parse-keywords` Parse keywords.val file. By default, the content of the file keywords.val is read in from within the source code.

`--parse-database` Parse database_nn3_keywords.val file. By default, the content of the file database_nn3_keywords.val is read in from within the source code.

`-v, --version` Show version number only.

`-h, --help` Display available command line options.

**Note:** For versions before 2021, the double-hyphen flags `--` are not supported. Please use instead the appropriate single-hyphen flags `-`:

- `-inputfile`
- `-database`
- `-license`
- `-outputdirectory`
- `-help` (Displays allowed options.)

---

### 9.5 Tutorials

Tutorials for the nextnano³ software can be found in their own section: nextnano³ Tutorials.
For much more accurate simulations of QCLs and RTDs, we recommend using nextnano.NEGF. See also this video: nextnanoNEGF_video_ITQW_2017_1080p.mp4

Figure 10.1: This video (mpg4, 23 MB) shows how a QCL works. The first 10 seconds summarise the features of our software, the remaining 20 seconds show the animated simulation results. At 20 seconds, the QCL starts to lase: The electron density (figure on the right) shows the population inversion, i.e. the higher state has a larger electron population than the lower lasing state.

**Note:** We think it makes sense to

1. **first** get familiar with the nextnanomat user interface (using the nextnano++ software) and
2. **then** have a look at the nextnano.NEGF software.

You can solve the Schrödinger equation for a QCL heterostructure subject to an electric field using the nextnano++ software.

You can find simple QCL wavefunction calculation (including intersubband transition matrix elements) examples here:

C:\Program Files\nextnano\2020_12_09\Sample files\nextnano++ sample files\Start with these input files:

- 1DQCL_simple_nnp.in, documented here: https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_QCL_simple.htm
- THzQCL_Andrews_Vienna_MatSciEng2008_nnp.in, documented here:
Once you have gained some familiarity with the nextnano mat user interface, you can eventually start using the nextnano.NEGF software.

**Note:**

1. Install the nextnano software.
   
   Instructions for download and installation

2. Once done, install the nextnano.NEGF software.

You can download the nextnano.NEGF software from here:

Unzip the file, store it to a useful location on your hard disk, and set the path to the nextnano.NEGF.exe executable in nextnano mat ==› Tools ==› Options ==› Simulation ==› nextnano.NEGF executable file.

Please make sure to check in Windows File Explorer ==› View ==› File name extensions. If you don’t check this option, you will only see:

- nextnano.NEGF (without file extension .exe.) and not
- nextnano.NEGF.exe

Sample input files can be found in the \nextnano.NEGF sample files\ folder. We recommend starting with this example: Open THz_QCL_GaAs_AlGaAs_Fathololoumi_OptExpress2012_10K-FAST.xml and press the run button. This is only a test file that runs fast (typically 10 minutes). For reliable calculations, please use the corresponding file THz_QCL_GaAs_AlGaAs_Fathololoumi_OptExpress2012-MEDIUM.xml which takes more CPU time (~12 hours) than *FAST.xml but produces much more accurate results. If the calculations take too long, your CPU might not be very fast or you do not have sufficient RAM. In that case, please contact us. You need at least 16 GB RAM, 32 GB are better. Please note that you should run the larger input files on a computer having 32 GB RAM. 16 GB might be not enough and require a change in the settings for the number of parallel threads used for the gain calculation. We recommend a Windows PC with 64 GB RAM, and a recent i7 CPU (e.g. a 6-core i7-8700). The whole PC is then around USD 1,000.
Figure 10.4: Template feature of nextnano\textsuperscript{mat} for parameter sweeps.
Figure 10.5: Set the path to the `nextnano.NEGF.exe` executable.
If the software does not run, you might have to specify the `Material_Database.xml` file in `Tools ==> Options ==> Material database`.

We hope you enjoy using `nextnano.NEGF`. Please let us know if you encounter any difficulty or notice any strange simulation results: Contact `nextnano`

Note also these slides on the implemented physics: `http://iqclsw2018.lpa.ens.fr/IMG/pdf/tutorial_iqclsw_2018_grange.pdf`

Finally, the technical documentation for the `nextnano.NEGF` software is available here: `https://www.nextnano.com/nextnano.NEGF/`
11.1 Introduction

Download the nextnano.MSB software

Documentation of the nextnano.MSB software

This manual describes the nextnano.MSB software, an NEGF quantum transport code based on the MSB method. The nextnano.MSB software has been developed to simulate Quantum Cascade Lasers (QCLs) and Resonant Tunneling Diodes (RTDs). The software calculates current-voltage characteristics and gain.

![Figure 11.1.1: Calculated electron and current density of a Quantum Cascade Laser](image)

This software is based on a novel quantum transport method that follows the nonequilibrium Green’s function (NEGF) framework but sidesteps any self-consistent calculation of lesser self-energies by replacing them by a quasi-equilibrium expression. This method generalizes the so-called Büttiker probe model but takes into account all relevant individual scattering mechanisms. It is orders of magnitude more efficient than a fully self-consistent NEGF calculation for realistic devices, yet accurately reproduces the results of the latter method. This method opens the path towards realistic three-dimensional quantum transport calculations.

MSB is the acronym of the Multi-Scattering Büttiker probe model.

nextnanomat software can execute the code and visualize the results.

Which scattering mechanisms are included?

- Longitudinal polar-optical phonon scattering (polar LO phonon scattering).
• Acoustic phonon scattering which includes some kind of interface roughness scattering. The latter is intrinsically included.

Where can I find some background on the implemented physics?

The MSB method is described in the following publications: [GreckOE2015], [GreckIWCE2010], [GreckPhD2012]

Copyright information

The nextnano.MSB software has been developed by Dr. Peter Greck for the nextnano GmbH. It is written in C++.

11.2 Command line arguments

NOTE: nextnano.MSB has been integrated into the nextnano++ executable (since 2021-08).

Below is the old documentation for stand-alone MSB executable.

Command line usage:

nextnano.MSB.exe -inputfile ..\my_QCL_input_file.xml -license "H:\My Documents\nextnano.MSB\License\license.xml" -database Materials.xml -outputdirectory ..\..\output\QCL

nextnano.MSB provides the following options:

- input_file, -inputfile input_file Name of input file to be read in.
  
  Example:
  
  • -inputfile THz_GaAs_QCL.in

- d database_file, -database database_file Name of material parameter database file to be read in (-m also works).
  
  Example:
  
  • -database "E:\nextnano.MSB\Materials_no_Varshni.xml"
-license license_file Name of license file to be read in.
   Example:
   • -license License_nnMSB.lic

-output_directory output_directory Directory name where to output results.
   Example:
   • -outputdirectory "D:\nextnano\nextnano output"

-debug 1 Generate additional debug information.

11.3 Input file syntax

11.3.1 Input file

The structure to be simulated as well as all simulation parameters have to be specified in a text file by the user. In addition to the XML format, the nextnano++-style format with the extension .negf is supported (since 2022 Apr).

Both formats of the sample input files are provided in the installation folder, e.g. THz_QCL_Fathololoumi_OpticsExpress2012.xml and THz_QCL_Fathololoumi_OpticsExpress2012.negf. This manual explains the syntax in the .negf format, but the keywords and hierarchical structure are identical between two formats except for <Variables> (difference described below).

You can convert the existing XML input files to the new format using nextnano.MSB executables later than 2022 Apr. If additional command line option -p (or --parse) is set, the execution stops after the input file and database have been read and parsed. In case XML files were read in, tentative conversions into the new format are output.

11.3.2 General syntax

```plaintext
nextnano.MSB{ # Version = 1.1.0 # This number must be consistent to the version of the executable used.

In the XML format,

```xml
<?xml version="1.0" encoding="utf-8"?>
<nextnano.MSB Version="1.1.0"> <!-- This number must be consistent to the version of the executable used. -->

In XML, Quotation marks are necessary if an equal sign (=) is present, e.g. AlloyX="Barrier".

Comment Section

```plaintext
Header{ # The Header contains a description of the input file.
  Author = "Peter G" # name of the author of this file
  Version = 1.0 # version number of this file
  Content = "See comments in input file."
  # Terahertz quantum cascade lasers operating up to ~200 K with optimized oscillator strength and improved injection tunneling
  # S. Fathololoumi et al.
  # Optics Express 20, 3866 (2012)
  # ...
  # CPU time: ...

(continues on next page)
# Here comes the main part, see below...

## Output

```
Output{
    Directory # Name of output folder for calculated results.
    WriteOutputEveryNthIteration 2 # Determines how frequent output is written.
    MaxNumberOfEigenstates 15 # Determines number of eigenstates written in output.
}
```

If `MaxNumberOfEigenstates` is not present, 25% of all possible eigenstates are written out. The total number of possible eigenstates corresponds to the total number of spatial grip points. More information on output of eigenstates.

```
FormatAsciiEnabled true # Enable output for ASCII files containing data in columns
FormatAsciiExt .dat # Specify the file extension for ASCII files containing data in columns.
FormatGnuPlotEnabled true # Enable GnuPlot output based on text output.
FormatGnuPlotExt .gnuplot # Specify the file extension for GnuPlot output.
FormatVTKPlotEnabled true # Enable VTK output.
FormatAvsEnabled true # Enable AVS output.
FormatAvsExt .avs # Specify the file extension for AVS files.
FormatAvsBinary # Specify AVS output mode
```

Currently, there is no text output included in the code.

```
FormatTextEnabled true # Enable text output.
# (Currently, there is no text output included in the code.)
FormatTextExt .dat # Specify the file extension for text files.
```

## Variables

**New:** In the .negf format, variables must be defined with the dollar sign as in `nextnano++` (see Syntax definition and sample input file in the installation folder).

Iterator feature is realized by the `nextnanomat` feature Template or `nextnanopy` Sweep. This simply generates multiple input files, which can be run in parallel (see Options: Simulation).

```
$Temperature 300 # (DisplayUnit:K) Temperature to sweep.
# (ListOfValues:100,150,175,190,200) (RangeOfValues:From=100,To=200,Step=11)
$Doping 1e16 # (DisplayUnit:1/cm^3) Doping concentration
$Barrier 0.15 # Al concentration of barriers.
```
Below is the documentation of `<Variables>` in the old XML format input file.

In the XML format, one can define variables (Constant) that are used further below in the input file, e.g. Doping = 1e16.

```
<Variables>
  <Constant>
    <Name Comment = "Doping concentration"> Doping </Name>
    <Value Unit = 1/cm^3> 1e16 </Value>
  </Constant>
  <Constant>
    <Name Comment = "Al concentration of barriers."> Barrier </Name>
    <Value Unit = "[0..1]"> 0.15 </Value>
  </Constant>
</Variables>
```

**Iterator Example 1** An Iterator can be used to sweep over variables, e.g. to do calculations for several temperatures. Here, the iterator accepts a certain list of values.

In this example, a calculation for the temperatures Temperature = 100 [K], 150 [K], ..., 200 [K] is performed.

```
<Iterator>
  <Name Comment = "Temperatures to sweep."> Temperature </Name>
  <Values Unit = K> 100, 150, 175, 190, 200 </Values>
</Iterator>
```

**Iterator Example 2** The Iterator also accepts a range of values from an Initial value to a Final value with a certain number of Steps. Here, the Final value and the number of Steps is given, and the step width, i.e. the Delta, is derived.

In this example, a calculation from Temperature = 100 [K] to Temperature = 200 [K] is performed for 11 different temperatures.

```
<Iterator>
  <Name Comment = "Temperatures to sweep."> Temperature </Name>
  <Initial Unit = K> 100 </Initial>
  <Final Unit = K> 200 </Final>
  <Steps Comment = "Note that the value is calculated via Initial+[0..Steps-1]*(Final-Initial)/(Steps-1)" => 11 </Steps>
</Iterator>
```

**Iterator Example 3** The iterator also accepts a range of values starting from an Initial value and performing a certain number of Steps with a step width of Delta. Here, the step width, i.e. the Delta, and the number of sweep Steps is given, and from these, the final value is derived.

In this example, a calculation for 11 different temperatures with a step width of 10 [K], starting from Temperature = 100 [K] is performed.

```
<Iterator>
  <Name Comment = "Temperatures to sweep."> Temperature </Name>
  <Initial Unit = K> 100 </Initial>
  <Delta Unit = K> 10 </Delta>
  <Steps Comment = "Note that the value is calculated via Initial+[0..Steps-1]*Delta"> 11 </Steps>
</Iterator>
```

**Device definition**

Temperature as value
Device{
  Temperature 200 # Unit = K # This is the lattice temperature, not the temperature of the electrons.
}

or temperature as a variable.

Temperature $Temperature # Unit = K # Here, instead of a number, the variable $Temperature is used.

Specify grid This is the grid spacing. It determines the number of grid points in the device. The more grid points, the longer the CPU time. If the grid spacing is modified, then always a slightly different structure is calculated as the barrier height and widths are adjusted to match the grid spacing, see AdjustBandedge. Even if the grid spacing exactly matches the layer widths for all variations of grid spacings, in all cases slightly different structures are calculated as the dispersion relations change with grid spacing. If the grid is too fine, then one cannot calculate any more sufficiently large energies. In this case, the results would not be reliable.

Grid{
  Spacing 0.25 # Unit = nm
}

Orientation and strain

In order to calculate strain related properties, information on the substrate material is required. Strain can be included (yes) or excluded (no). The crystal structure can be Zincblende or Wurtzite. The lattice constants of the substrate are used to calculate strain. The (hkl) Miller indices are needed in order to define the orientation of the substrate on which the heterostructure is grown, i.e. the crystal coordinate system has to be rotated into the simulation coordinate system. The growth direction (simulation axis z) is perpendicular to the substrate (xy plane). The crystal and thus all anisotropic material properties are rotated accordingly.

Crystal Zincblende or Wurtzite

Substrate{
  Material{
    Base # substrate material
  }
}

Orientation{
  z_axis{ # The heterostructure growth direction is along z (simulation direction).
    h 0 # (hkl) are the Miller indices of the plane perpendicular to the z direction.
    k 0
    l 1
  }

  y_axis{ # (hkl) are the Miller indices of the plane perpendicular to the y direction.
    h 0
    k 0
    l 1
  }
}

Strain no # include strain (yes/no)

Energy grid

Here, the energy grid spacing is defined. The energy grid is homogeneous. The Nodes are the number of energy grid points. The more energy grid points, the longer the CPU time. Example: If one has defined an energy Range

---
of 0.3 [eV], then by choosing 601 Nodes, the resulting energy grid spacing is 0.5 [meV]. If the barrier height in the conduction band is 0.5 [eV], then Range should also be around 0.5. The larger the barrier, the higher the Range.

\[
\text{Energy}\{
\begin{array}{l}
\text{Nodes} = 601 \quad \# \text{Comment = "Number of energy grid points."} \\
\text{Range} = 0.3 \quad \# \text{Unit = eV} \quad \# \text{Comment = "Offset is based on conduction band edge, i.e. Input/BandEdge_conduction_input.dat."}
\end{array}
\}
\]

Definition of layers

\[
\# \text{Begin Layers}
\text{Layer}\{
\begin{array}{l}
\text{Thickness} = 4 \quad \# \text{Unit = nm}
\end{array}
\}
\]

or alternatively use conduction band edge as defined by ConductionBandOffset.

\[
\text{Material}\{
\begin{array}{l}
\text{Base} = \text{GaAs}
\end{array}
\}
\]

or (default) calculate conduction band edge by using valence band offset (ValenceBandOffset) and temperature dependent band gap, i.e. \( E_c = E_v + \Delta E_{\text{gap}}(T) \), taking into account the Varshni parameters.

\[
\text{Material}\{
\begin{array}{l}
\text{Base} = \text{GaAs}
\end{array}
\}
\]

It is very useful to define one (or several) periods of a quantum cascade laser with such a marker (BeginCluster). This allows one to define a bias per period very conveniently, i.e. a defined electrostatic potential drop per period. Here, a 4.3 \([\text{nm}]\) wide AlGaAs (Al(\(x\))Ga(1-\(x\))As) layer with an alloy concentration of \(x = 0.15\) is defined. The layer is undoped. Here, two labels, namely Center and QCL, are assigned to a cluster which begins here and ends at Layer EndCluster="Center, QCL".

\[
\# \text{Begin Period}
\text{Layer}\{
\begin{array}{l}
\text{BeginCluster} = \text{"Center, QCL"}
\end{array}
\}
\]

(continues on next page)
Doping 0  # Unit = 1/cm³
}

Layer{   # Here, a 7.9 nm wide GaAs layer is defined. The layer has a doping concentration of 6e16 1/cm³.
    Thickness 7.9  # Unit = nm
    Material{
        Base GaAs
    }
    Probes 1.0
    Doping 6e16  # Unit = 1/cm³
}

Layer{   # Here, a 5.5 nm wide GaAs layer is defined. The layer is undoped.
    Thickness 5.5  # Unit = nm
    Material{
        Base GaAs
    }
    Probes 1.0
    Doping 0  # Unit = 1/cm³
}  # End Cluster

Layer{   # Here, a 4.0 nm wide GaAs layer is defined. The layer has a doping concentration determined by the variable $LeadDoping.
    Thickness 4.0  # Unit = nm  # Here, a 4.0 nm wide GaAs layer is defined. The layer has a doping concentration determined by the variable $LeadDoping.
    Material{
        Base GaAs
    }
    Probes 1.0
    Doping $LeadDoping  # Unit = 1/cm³
}  # End Period

Each Layer{ can have an additional flag:

Layer{ AdjustBandedge yes  # Comment = "yes or no"  # default is yes
}

The ratio of the desired barrier width (e.g. 1.1 [nm]) to the width used in the simulation (e.g. grid spacing 0.5 [nm] => 2*0.5 [nm] = 1.0 [nm]) is added to the barrier height to keep the area of the barrier the same. This approach compensates the loss in accuracy when using a larger grid spacing very well. This applies analogously to the well width. The effect of this flag can be seen in these files in case they are plotted on top of each other:

- Input\BandEdge_conduction_input.dat
- Input\BandEdge_conduction_adjusted.dat

Contacts (leads)

# Begin Lead
Lead{
    Name  # Define a source contact with voltage V = 0.000 V.
    Voltage{
        Initial 0.000  # Unit = V
    }
    Temperature 300  # Unit = K  # Define the temperature of the contact.
        # If not defined, then the device temperature is used

(continues on next page)
\textbf{Lead}\{ 
  \hspace{1em} \texttt{Name} \hspace{2em} # Define a drain contact with voltage V that is \hspace{0em}
  \hspace{1em} varied during the calculation starting from \texttt{V = 0e-3 V}. 
  \hspace{1em} \texttt{Voltage}\{ 
  \hspace{3em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
\}\} 

\textbf{(Check: Probably the terms \textquotesingle Source \textquotesingle and \textquotesingle Drain \textquotesingle are required. This should be checked.)}

In the defined structure, there is a region (here the cluster is called \texttt{Center}) where the given voltage drops, e.g. \texttt{Cluster>Center</Cluster}. One can include the leads into this region, then the whole voltage drops over the whole structure. Typically, for QCL simulations, the given bias voltage refers to one period only, i.e. the bias voltage per period. In this case, the lead regions are not included. Therefore, the actual voltage drop between the left and the right boundary grid points of the whole structure is larger than the specified value of the voltage drop because the structure is larger than the QCL period as the leads have to be taken into account when calculating the length of the structure.

Example: If one applies 50 mV, this difference corresponds to an energy difference of 50 meV of the conduction band edge of the leftmost and rightmost grid point of the structure, or of the leftmost and rightmost grid point of the region where the bias drops. The effective electric field can be calculated as follows:

\[
\text{field} = \frac{\text{applied bias}}{\text{relevant region}}
\]

Depending on the definition in the input file, the relevant region can include the lead regions or not.

\textbf{Option 1} A voltage is specified (which typically corresponds to a bias/period):

\begin{verbatim}
Voltage{ 
  \hspace{2em} \texttt{Cluster} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Initial} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Delta} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Final} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Steps} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
}\}
\end{verbatim}

\textbf{Option 2} As an alternative, instead of specifying a voltage, one can also specify an electric field. Internally, the relevant bias is calculated from the field, and then the calculation starts.

\begin{verbatim}
Voltage{ 
  \hspace{2em} \texttt{SpecifiedByElectricField} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Initial} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Delta} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Final} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
  \hspace{2em} \texttt{Steps} \hspace{1em} \texttt{varied during the calculation starting from \texttt{V = 0e-3 V}}. 
}\}
\end{verbatim}

\textbf{MSB model definition - Multi-scattering Büttiker probe model}

\textbf{MSB single-band (1Band = single-band)}
Piezoelectric and pyroelectric charge densities can be included (yes) or excluded (no) in the Poisson equation.

Define numerical parameters

MaxGreenIts 25 # Comment = "Max. iterations within the G^R cycle."
MaxGreenDelta 3e-7 # Unit = 1/nm/eV # Comment = "Max. change between two cycles to abort iteration."
MaxProbeIts 15 # Comment = "Max. iterations for current conservation calculation. Only used if Core."
MaxProbeNorm 1e-9 # Unit = A/cm^2 # Comment = "Max. absolute leakage current to abort iteration."
MaxPoissonOuterIts 25 # Comment = "Max. outer Poisson iterations where G^R is recalculated."
MaxPoissonInnerIts 15 # Comment = "Max. inner Poisson iterations where the density predictor is used."
MaxPoissonDensityNorm 1e9 # Unit = 1/cm^3 # Comment = "Max. absolute density deviation to abort Poisson iterations."
MaxPoissonCorrectorNorm 1e-4 # Unit = V # Comment = "Max. absolute potential deviation to abort Poisson iterations."

MaxPoissonOuterIts is an important number. Make sure that this number is not exceeded. If it is exceeded, the calculation did not converge! In the log file, the following information is written to inform about the number of Poisson iterations: Poisson iteration 30

Now we define the core properties of the MSB method.

In a ballistic calculation, where scattering is not taken into account, the following parameters are set to zero:

- Probes
- ScatteringStrengthConst
- ScatteringStrengthBP
- ScatteringStrengthMSB

- bulk (default): The initial guess of the electrostatic potential assumes a bulk semiconductor where the resulting density is neutral at the grid point 0 or n, respectively.
- zero: The initial guess of the electrostatic potential is zero.

Note that direct and iterative can lead to different results. They are not equivalent.
• The iterative model is explained in Section 5.2 in the PhD thesis of Peter Greck. Here, for each probe, virtual chemical potentials \( \mu_p(z) \) are calculated. The units are [eV]. Therefore, for each probe a virtual chemical potential exists which leads to a Fermi distribution of each probe \( f(E, \mu_p(z)) \). This virtual chemical potential \( \mu \) refers to the total probe, i.e. AC + LO.

• The direct model is explained in Section 7.2 in the PhD thesis of Peter Greck. Here one assumes that the distribution function \( f_p(z) \) of each probe is a linear combination of the source and drain distributions, \( f_S \) and \( f_D \), respectively, where S means Source, and D means Drain.

\[
    f_p(E) = c_p f_S(E) + (1 - c_p) f_D(E)
\]

The coefficients \( c_p \) for each probe are in the interval [0,1] and are dimensionless. Here, a linear system of equations has to be solved to obtain the coefficients \( c_p \). In contrast to the iterative model, the virtual chemical potentials \( \mu_p \) for the probes cannot be extracted in this case. For the results presented in the PhD thesis of P. Greck, always the direct model was used.

VoltageMode = drop # Comment = "Specify handling of applied bias
→ voltage. Drop-mode ensures that the specified voltage drops along
→ the device. Flat-mode uses Neumann boundary conditions that do not
→ guarantee a complete voltage drop. [drop|flat]"

The total scattering strength that is used for the calculation is the sum of the following three products, i.e. total = Model #1 + Model #2 + Model #3:

```
Probes = *ScatteringStrengthMSB +
Probes = *ScatteringStrengthBP + (This term is usually zero.) ==> These are the “normal” Büttiker probes (see [DattaETMS1995]).
Probes = * ScatteringStrengthConst + (This term is usually zero.) ==> These are the “oldest” type of Büttiker probes (see [DattaETMS1995]).
```

We recommend to only define ScatteringStrengthMSB and set ScatteringStrengthBP and ScatteringStrengthConst to zero.

Model #1 for scattering Here, we can define a scattering strength for both LO, and AC phonon scattering.

```
ScatteringStrengthMSB = 1.0 # Comment = "Novel scattering via
→ MSB. Scattering strength is calculated from material parameters.
→ This parameter tunes the scattering rates from 0.0=disabled over 1.
→ 0=normal to X=amplified."
ScatteringStrengthMSB_AC = 1.0
ScatteringStrengthMSB_LO = 1.0
```

Model #2 for scattering These are the normal Büttiker probes (see [DattaETMS1995]).

```
ScatteringStrengthConst = 0.0 # Unit = eV # Comment =
→ "Additional constant scattering strength for every probe. Typical
→ values are 0.001-0.01 and zero disables this scattering mechanism."
```

Model #3 for scattering These are the oldest type of Büttiker probes (see [DattaETMS1995]).

```
ScatteringStrengthBP = 0.0 # Unit = eV # Comment =
→ "Additional scattering via Büttiker-Probes. Typical values are 0.
→ 0.001-0.01."
```

Gain

```
Gain{
Cluster # Comment = "Specify the
→ cluster where optical gain should be calculated."
PhotonEnergyInitial = 3e-3 # Unit = eV # Comment = "Min. photon
→ energy for gain calculation."
}
```

(continues on next page)
The more PhotonEnergySteps are used, the more ragged the gain curve gets. Decreasing this number, smoother gain curves can be obtained. This is normal. If the photon energy grid spacing is too small, one tries to resolve energies that are not resolved within the calculated Green's functions that are used in the gain calculations, i.e. one tries to calculate several photon energies within one discrete energy step of the Green’s function. An obvious solution would be to increase the energy grid resolution of the Green’s functions. However, it is recommended to rather prefer a coarse photon energy grid as a high photon energy grid resolution does not lead to significantly more insight.

Hints

- One can disable the scattering within the barriers. This can reduce computational time with very minor influence on overall results.

```
Layer{
    Probes = 0.0  # disabled
}
Layer{
    Probes = 1.0  # enabled
}
```

Important: If there are no probes inside the barrier, the iterative calculation of the probes is much more stable while hardly changing the physics because the occupation in the barriers is basically zero. Moreover, this avoids the numerical explosion of the linear system of equations through a division by zero which leads to NaN (not a number) in the .log file.

- How can I do a ballistic calculation? There are three ways to achieve this.
  a) Use

```
Core{
    BallisticCalculation = yes  # Comment = "yes or no"
}
```

b) Set all Probes = 0 to zero, i.e. 0.0.

c) Set all of the following parameters to zero: ScatteringStrengthMSB, ScatteringStrengthBP and ScatteringStrengthConst, i.e. 0.0.

- For a quantum cascade laser simulation it can make sense to simulate 5 or 7 periods and take the middle period as the one where one extracts the results like local density of states, electron density, current density, gain, ...

- If one simulates only one period of a QCL, then one should add a doping layer at the left and right boundary (e.g. of width 4.5 nm). This should not be necessary if 5 periods or more are simulated. In this case, doping could be omitted.
11.4 Material database

The material parameters for zinc blende and wurtzite materials that are used by the nextnano.MSB software are stored in a file called materials.negf. The user can add further materials if needed.

If you run nextnano.MSB via the nextnatomat GUI, you can choose to read in a customized material database as follows:

nextnatomat ==> Tools ==> Options ==> Material database

In the database there are entries for binary compounds like GaAs, AlAs, InP, . . . , as well as for ternary compounds like AlGaAs, InGaAs, . . .

11.4.1 Elements and binary compounds

Note that the material database contains parameters that are not used by the nextnano.MSB code. This is because we unify the database with the nextnano.NEGF software.

```plaintext
# binary compound
Material{
  NameGaAs
  CrystalStructureZincBlende
  ConductionBandOffset 2.979 # Unit = eV
  ValenceBandOffset 1.346 # Unit = eV
  BandGap 1.519 # Unit = eV
  BandGapAlpha 0.5405e-3 # Unit = eV/K
  BandGapBeta 204 # Unit = K
  ElectronMass 0.067 # Unit = m0
  EpsStatic 12.93
  EpsOptic 10.89
  DeformationPotential -9.36 # Unit = eV
  ValenceDefPotHydro 1.16 # Unit = eV
  ValenceDefPotUniaxial_b -2.0 # Unit = eV
  ValenceDefPotShear_d -4.8 # Unit = eV
  MaterialDensity 5.316e3 # Unit = kg/m^3
  VelocityOfSound 4.73e3 # Unit = m/s
  LOPhononEnergy 35e-3 # Unit = eV
  LOPhononWidth 3e-3 # Unit = eV
  TOPhononEnergy 33.86e-3 # Unit = eV
  AcousticPhononEnergy 5e-3 # Unit = eV
  S -2.88 # Unit = eV
  Ep 28.8 # Unit = eV
}
```

(continues on next page)
Elastic_c11 = 122.1 # Unit = GPa
Elastic_c12 = 56.6 # Unit = GPa
Elastic_c44 = 60.0 # Unit = GPa
Piezo_e14 = -0.160 # Unit = C/m^2

ConductionBandOffset
type double
unit [eV]
Energy value that defines the position of the conduction band edges on an absolute energy scale. The zero point of energy is arbitrary. It can be used to define a conduction band offset between two different materials.

ValenceBandOffset
type double
unit [eV]
Energy value that defines the position of the average valence band edge energy \( E_{v,av} \) on an absolute energy scale. The zero point of energy is arbitrary. It can be used to define a valence band offset between two different materials.

average valence band edge energy: 
\[
E_{v,av} = \left( E_{hh} + E_{lh} + E_{so} \right)/3
\]

BandGap
type double
unit [eV]
Band gap at the \( \Gamma \) point given for temperature of \( T = 0 \) K. The code automatically calculates the temperature dependent band gap using the Varshni formula. If the band gap is specified here for another temperature, the Varshni parameters BandGapAlpha and BandGapBeta should be set to zero.

BandGapAlpha
type double
unit [eV/K]
Varshni parameter \( \alpha \) to allow for temperature dependent band gap.

BandGapBeta
type double
unit [K]
Varshni parameter \( \beta \) to allow for temperature dependent band gap.

Note: BandGap, BandGapAlpha, BandGapBeta are not used inside the calculation. They are just needed to output the valence band edge (which is not used either).
**ElectronMass**

- **type**: double
- **unit**: [m0]

Isotropic effective electron mass of the \( \Gamma \) conduction band.

**EpsStatic**

- **type**: double
- **unit**: []

Static dielectric constant, low frequency dielectric constant \( \varepsilon_0 \)

**EpsOptic**

- **type**: double
- **unit**: []

Optical dielectric constant, high frequency dielectric constant :math:`\varepsilon_\infty`

**LOPhononEnergy**

- **type**: double
- **unit**: [eV]

Longitudinal optical (LO) phonon energy \( E_{OP} \).

This parameter must not be set to zero as there will be a division by zero in this case, see p. 44 of PhD thesis of Peter Greck:

\[
N_{OP} = \exp\left(\frac{E_{OP}}{(k_B T)}\right) - 1 \approx \frac{1}{1 - 1} = \text{NaN (not a number)}
\]

\( N_{OP} \) is the phonon distribution and a prefactor of the equation (eq. (7.5)) where the LO phonon scattering strength is calculated, i.e. if \( N_{OP} \ll 1 \), then the LO phonon scattering is rather small.

\( E_{OP} = 35 \text{ meV in GaAs} \)

<table>
<thead>
<tr>
<th>( N_{OP} ) for GaAs</th>
<th>( T ) [K]</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 \cdot 10^{-45}</td>
<td>4</td>
</tr>
<tr>
<td>1.3 \cdot 10^{-5}</td>
<td>30</td>
</tr>
<tr>
<td>0.000297</td>
<td>50</td>
</tr>
<tr>
<td>0.017524</td>
<td>100</td>
</tr>
<tr>
<td>0.151055</td>
<td>200</td>
</tr>
<tr>
<td>0.348148</td>
<td>300</td>
</tr>
</tbody>
</table>

**LOPhononWidth**

- **type**: double
- **unit**: [eV]

This is a numerical value that avoids reducing the coupling strength to a \( \delta \) function:

\( E + E_{OP} \rightarrow E + E_{OP} \pm \Delta E/2 \), where \( \Delta E = \text{LOPhononWidth}. \)

**Note**: The following 4 variables are only relevant for acoustic phonon scattering.
DeformationPotential

:type double
:unit [eV]
Scalar deformation potential
It is used for acoustic phonon scattering.

MaterialDensity

:type double
:unit [kg/m^3]
Material density or mass density

VelocityOfSound

:type double
:unit [m/s]
Sound velocity

AcousticPhononEnergy

:type double
:unit [eV]
Acoustic phonon energy

Note: The following 5 variables are only relevant for strain calculations.

Lattice_a

:type double
:unit [nm]
Lattice constant \( a \)

Elastic_c11

:type double
:unit [GPa]
Elastic constant \( c_{11} \)
Elastic\_c12

\begin{verbatim}
  type double
  unit [GPa]
\end{verbatim}

Elastic constant $c_{12}$

Elastic\_c44

\begin{verbatim}
  type double
  unit [GPa]
\end{verbatim}

Elastic constant $c_{44}$

Piezo\_e14

\begin{verbatim}
  type double
  unit [C/m^2]
\end{verbatim}

Piezoelectric constant $e_{14}$

11.4.2 Ternary compounds

For ternary compounds like Al$_x$Ga$_{1-x}$As, we have to specify bowing parameters. The material parameters in many ternary alloys ($A_B C$) can be approximated in the form of the usual quadratic function

$$T_{ABC} = x B_{AC} + (1-x) B_{BC} - x(1-x) C_{ABC}$$

where $C_{ABC}$ is the bowing parameter.

```plaintext
# ternary compound
Material{
  Name = "In(x)Ga(1-x)As"
  Binary1 = "InAs"
  Binary2 = "GaAs"
  CrystalStructure = Zinic
  ConductionBandOffset = 0 # Unit = eV
  ValenceBandOffset = -0.38 # Unit = eV
  BandGap = 0.477 # Unit = eV
  BandGapAlpha = 0 # Unit = eV/K
  BandGapBeta = 0 # Unit = eV/K
  DeltaSO = 0.15 # Unit = eV
  ElectronMass = 0.0091 # Unit = m0
  EpsStatic = 0 # Unit = eV
  EpsOptic = 0 # Unit = eV
  S = 3.54 # Unit = eV
  Ep = -1.48 # Unit = eV
  kp_8_bands{
    B = 0.0
    L = 0.0
    M = -1.140907266
    N = 0.0
  }
  DeformationPotential = 2.61 # Unit = eV
  ValenceDefPotHydro = 0 # Unit = eV
  ValenceDefPotUniaxial_b = 0 # Unit = eV
  ValenceDefPotShear_d = 0 # Unit = eV
  MaterialDensity = 0 # Unit = kg/m^3
}
```

(continues on next page)
velocityOfSound 0 # Unit = m/s
LOPhononEnergy 0 # Unit = eV
LOPhononWidth 0 # Unit = eV
AcousticPhononEnergy 0 # Unit = eV
Lattice_a 0 # Unit = nm
Elastic_c11 0 # Unit = GPa
Elastic_c12 0 # Unit = GPa
Elastic_c44 0 # Unit = GPa
Piezo_e14 0 # Unit = C/m^2

Note: Currently, the Varshni parameters $\alpha$ (BandGapAlpha) and $\beta$ (BandGapBeta) are interpolated. It is better and more meaningful to interpolate the band gap instead.

11.4.3 Quaternary compounds

Quaternary compounds like $\text{Al}_x\text{In}_y\text{Ga}_{1-x-y}\text{N}$ are implemented as:

```plaintext
# quaternary compounds
Material{
  Name "Al[x]In[y]Ga[1-x-y]N"
  CrystalStructure "Wurtzite"
  Alloy "AlN[x]In[y]Ga[1-x-y]N"
  Binary1 "AlN[x]"
  Binary2 "InN[y]"
  Binary3 "GaN[1-x-y]"
  Ternary_xy "Al[0.5]In[0.5]Ga[0.5]N"
  Ternary_x "Al[0.5]Ga[0.5]N"
  Ternary_y "In[0.5]Ga[0.5]N"
  ConductionBandOffset 0 # Unit = eV
  ValenceBandOffset 0 # Unit = eV
}
```

Final remark It is recommended to use position dependent material parameters, i.e. for parameters like LO phonon energy, deformation potential, sound velocity, material density and acoustic phonon energy. Obviously, the Büttiker probes $B(x)$ depend on position. But in fact, the parameters for the wells are the most important ones. The parameters in the barriers only have a minor influence. One can include them in the calculation but the Büttiker probes in the barriers should not have any significant influence on the final result.

11.5 Simulation output

Output files of the simulation

Note: The output of a simulation can easily exceed 1 GB. Please make sure you have enough disk space available.

All files that have the file extension
- .dat can be plotted with nextnanomat or any other visualization software like e.g. Origin.
- .gnu.plt can be plotted with Gnuplot.
- .fld can be plotted with nextnanomat or AVS/Express.
- .vtr can be plotted with nextnanomat, Paraview or VisIt.
**Note:** Recommendation: Please install Gnuplot. It is then very convenient to plot the results of the nextnano.MSB calculations. Within nextnanomat, one can plot the band profile together with other data using *Keep current graph as overlay.*

### 11.5.1 Input

In this folder, all material and input parameters are contained.

**Material parameters**

- **BandEdge_conduction_input.dat**
  This is the conduction band edge profile that has been specified in the *input file.*
- **BandEdge_conduction_adjusted.dat**
  This is the conduction band edge profile that has been used in the *simulation.*
  The suffix _adjusted indicates that the well and barrier widths, as well as heights, had been adjusted automatically by the program.
  Why do the band edges for *_input.dat and *_adjusted.dat differ? See AdjustBandedge in *Input file* documentation for more information.
  conduction band edge in units of [eV]
  Position [nm] Conduction Band Edge [eV]

- **EffectiveMass.dat**
  electron effective mass in units of [m0]
  Position [nm] Effective Mass [m0]

- **EpsStatic.dat**
  static dielectric constant in units of []
  Position [nm] Relative Static Permittivity []

- **EpsOptic.dat**
  optical dielectric constant in units of []
  Position [nm] Relative Optical Permittivity []

- **MaterialDensity.dat**
  material density or mass density in units of [kg/m^3]
  Position [nm] Material Density [kg/m^3]

- **PhononEnergy_acoustic.dat**
  acoustic phonon energy in units of [eV]
  Position [nm] Acoustic Phonon Energy [eV]

- **PhononEnergy_LO.dat**
  longitudinal optical phonon energy (LO phonon energy) in units of [eV]
  Position [nm] LO Phonon Energy [eV]
• PhononEnergy_LO_width.dat
  width of longitudinal optical phonon energy (LO phonon energy) in units of [eV]
  For an explanation, see Material database.
  Position [nm] LO Phonon Energy Width [eV]

• VelocityOfSound.dat
  sound velocity in units of [m/s]
  Position [nm] Velocity of Sound [m/s]

Input parameters

• AlloyContent.dat
  alloy profile in units of [
  Position [nm] Alloy Content [

• DopingConcentration.dat
  doping concentration in units of [cm^-3].
  It is assumed that all dopants are ionized (fully ionized). An ionization model is not included.
  Position [nm] Doping Concentration [1/cm3]

• ProbeValues.dat
  profile of the Büttiker probes in units of [], value is between 0 and 1
  Position [nm] Probe Values []

11.5.2 Output

BandProfile

• BandEdge_conduction.dat
  conduction band edge in units of [eV]
  Position [nm] Conduction Band Profile [eV]

• ElectrostaticPotential.dat
  electrostatic potential in units of [V]
  Position [nm] Electrostatic Potential [V]

• ElectricField.dat
  electric field in units of [kV/cm]
  Position [nm] Electric Field [kV/cm]
Eigenstates

The data contained in this folder is not used inside the actual MSB algorithm. It is merely a post-processing feature. Once the self-consistently calculated conduction band edge,

\[ E_c(x) = E_{c,0}(x) - e\phi(x) \]

is known, the eigenenergies \( E_i \) and wave functions \( \psi_i(x) \) of the single-band Schrödinger equation are calculated. \( \phi(x) \) is the electrostatic potential which is the solution of the Poisson equation.

\[ H\psi(x) = E\psi(x) \]

The Schrödinger equation is solved three times, i.e. with three different boundary conditions

- **periodic**: \( \psi(x = 0) = \psi(x = L) \)
- **Dirichlet**: \( \psi(x = 0) = \psi(x = L) = 0 \), and
- **Neumann**: \( \frac{d\psi}{dx} = 0 \) at the left \( (x = 0) \) and right \( (x = L) \) boundary.

There are files for the

- **amplitudes** \( \psi_i(x) \) in units of \( \text{[nm}^{-1/2}\text{]} \)
  
  Amplitudes_Dirichlet.dat / *_Neumann.dat / `*`_Periodic.dat

- **amplitudes** \( \psi_i(x) \) **shifted** by their eigenenergies \( E_i \)
  
  Amplitudes_shift_Dirichlet.dat / *_Neumann.dat / `*`_Periodic.dat

- **probability densities** \( \psi_i^2(x) \) in units of \( \text{[nm}^{-1}\text{]} \)
  
  Probabilities_Dirichlet.dat / *_Neumann.dat / `*`_Periodic.dat

- **probability densities** \( \psi_i^2(x) \) **shifted** by their eigenenergies \( E_i \)
  
  Probabilities_shift_Dirichlet.dat / *_Neumann.dat / `*`_Periodic.dat

- **eigenvalues** \( E_i \) in units of \( \text{[eV]} \)
  
  Eigenvalues_Dirichlet.dat / *_Neumann.dat / `*`_Periodic.dat

CarrierDensity

- The position and energy resolved electron density \( n(z, E) \) is contained in this file:
  
  CarrierDensity_energy_resolved.avs.fld (or the corresponding *.gnu.plt/*.dat file)

- The electron density \( n(z) \) is contained in this file:
  
  CarrierDensity.dat / *.gnu.plt

  Position [nm] Density [1/cm^3]

DOS

- **DOS_position_resolved.avs.fld (or the corresponding *.gnu.plt/*.dat file)**
  
  The position and energy resolved local density of states (LDOS) \( \rho(z, E) \) in units of \( \text{[eV}^{-1} \text{nm}^{-1}\text{]} \).

- **DOS.dat / *.gnu.plt**
  
  Energy [eV] DOS [1/eV]

  The density of states (DOS) \( n(E) \).

The density of states is the sum of the DOS due to source, drain and Büttiker probes, i.e.

\[ \text{DOS} = \text{DOS}_\text{Source} + \text{DOS}_\text{Drain} + \text{DOS}_\text{Probes} \]
• DOS_Proses_position_resolved.avs.fld (or the corresponding *.gnu.plt / *.dat file)

The position and energy resolved density of states (LDOS) \( \rho(z, E) \) due to the Büttiker probes only in units of \([\text{eV}\,\text{nm}^{-1}]\).

This DOS is induced by scattering events. Like the lead-connected DOS enters the device through the source or drain contacts, respectively, the probe DOS is due to scattering.

Here we plot the LDOS for the probes, i.e. all probes are summed up, and the LDOS of the probes is determined by the self-energies of the probes. A probe has the scattering strength \( B = B_{\text{AC}} + B_{\text{LO}} \).

From this plot one cannot see if the DOS is due to LO or AC scattering events as both scattering potentials are added to obtain \( B \).

In fact, as one considers the probes for each grid point individually, one could print out the LDOS for each grid point. So each probe grid point produces a probe spectral function \( A_p(z, E) \), e.g. the probe at grid point 5 produces the grid point 5 connected local density of states which is nonzero not only on grid point #5 but everywhere.

Each probe has its own chemical potential \( \mu \), e.g. the probe at grid point 5 has \( \mu_5 \). Then the LDOS of probe 5 \( \rho_5(z, E) \) is occupied everywhere with this chemical potential \( \mu_5 \). In our algorithm, we only have one probe at each grid point having the combined scattering potential \( B = B_{\text{AC}} + B_{\text{LO}} \). In principle, each grid point could have 2 probes, one for AC and one for LO phonon scattering. However, this is not the case in our algorithm so far.

• DOS_Proses.dat/*.gnu.plt

Energy [eV] DOS [1/eV]

The density of states (DOS) \( n(E) \) due to the Büttiker probes only (probe-connected DOS).

• DOS_Lead_Source_position_resolved.fld (or the corresponding *.gnu.plt / *.dat file)

DOS_Lead_Drain_position_resolved.fld (or the corresponding *.gnu.plt / *.dat file)

The position and energy resolved local density of states (LDOS) \( \rho(z, E) \) due to the source and drain contact only in units of \([\text{eV}\,\text{nm}^{-1}]\).

• DOS_Lead_Source.dat/*.gnu.plt

DOS_Lead_Drain.dat/*.gnu.plt

Energy [eV] DOS [1/eV]

The density of states (DOS) \( n(E) \) due to the source and drain contact only (lead-connected DOS).

• DOS_Leads_position_resolved.fld (or the corresponding *.gnu.plt / *.dat file)

The position and energy resolved local density of states (LDOS) \( \rho(z, E) \) due to the drain and source contacts in units of \([\text{eV}\,\text{nm}^{-1}]\).

This corresponds to the sum of

DOS_Lead_Source_position_resolved.fld + DOS_Lead_Drain_position_resolved.fld.

• DOS_Leads.dat/*.gnu.plt

Energy [eV] DOS [1/eV]

The density of states (DOS) \( n(E) \) due to the drain and source contacts (lead-connected DOS). This corresponds to the sum of

DOS_Lead_Source.dat + DOS_Lead_Drain.dat.
**Probes**

- **ProbeLevels.dat**

  This output depends on the probe model used: **ProbeMode**
  
  a) **ProbeMode = iterative** # Comment="Specify method to calculate current conservation."
  
  local Büttiker probe *virtual chemical potentials* $\mu_p$ [eV] related to the occupation of the probes

  Position [nm] Local Probe Levels [eV]

  For zero applied bias, the local probe levels are 0 [eV] which is the same value as the chemical potentials of the source and drain contacts as there is no current flowing. The probe levels indicate the occupation of the scattering states.

  b) **ProbeMode = direct** # Comment="Specify method to calculate current conservation."

  local Büttiker probe *coefficients* $c_p$ [] (dimensionless)

  Position [nm] Local Probe Levels (% of Drain) [0..1]

  Here, the units are dimensionless and the numbers are between 0 and 1.

  0 means 100 % occupation of the probes by the source contact. 1 means 100 % occupation of the probes by the drain contact.

  For zero applied bias, the local probe levels are 0.5, i.e. 50 % occupation due to source and 50 % due to drain contact.

  See also the comments on **ProbeMode** in the documentation of the **Input file**.

  There is only one $B(z, E)$ for which current conservation holds. Once this quantity has been calculated, one cannot distinguish any more between optical and acoustic phonon scattering.

  If the command line argument `-debug 1` is provided, additional output is written to this folder.

- **NumericalPrefactor_MSB_AC.dat**
  
  **NumericalPrefactor_MSB_LO.dat**

  The numerical prefactors for the MSB scattering potentials for acoustic phonon (AC) and LO phonon scattering are given in units of [ ... ]. (Add correct units here.)

  - For LO, the prefactor is given in eq. (7.9) of the PhD thesis of P. Greck. It reads:
    
    $$ B_{OP} \sim \frac{e^2 \epsilon \hbar \omega_{LO}}{32 \pi^2 \epsilon_0} \left( \epsilon_{optic}^{-1} - \epsilon_{static}^{-1} \right) $$

  - For AC, the prefactor is given after eq. (7.8) of the PhD thesis of P. Greck. It reads:
    
    $$ B_{AP} \sim \frac{V_0^2 k_B^2 T}{8\pi M \rho v_s E_{AP}} $$

  The prefactors are independent of applied bias voltage.

- **ScatteringPotential_MSB_AC.dat**
  
  **ScatteringPotential_MSB_LO.dat**

  The scattering potentials for MSB for acoustic phonon (AC) and LO phonon scattering are given in units of [ ... ]. *It is not [nm] as written in the output file.*

  The scattering potential for LO phonons $B_{OP}$ is given in eq. (7.9) of the PhD thesis of P. Greck.

  The scattering potential for acoustic phonons $B_{AP}$ is given after eq. (7.8) of the PhD thesis of P. Greck.

- **ScatteringPotential_MSB_AC_position_resolved.dat**
  
  **ScatteringPotential_MSB_LO_position_resolved.dat**
The position resolved scattering potentials for MSB for acoustic phonon (AC) and LO phonon scattering are given in arbitrary units.

**Gain**

- **gain_energy_resolved.avs.fld** (or the corresponding *.gnu.plt / *.dat file)
  The position and energy resolved optical gain $g(z,E)$ in units of [eV$^{-1}$ cm$^{-1}$].
  Here, energy $E$ is the photon energy.
- **gain_frequency_resolved.avs.fld** (or the corresponding *.gnu.plt / *.dat file)
  The position and frequency resolved optical gain $g(z,\nu)$ in units of [THz$^{-1}$ cm$^{-1}$].
  Here, frequency $\nu$ is the photon frequency.
- **gain_wavelength_resolved.avs.fld** (or the corresponding *.gnu.plt / *.dat file)
  The position and wavelength resolved optical gain $g(z,\lambda)$ in units of [µm$^{-1}$ cm$^{-1}$].
  Here, wavelength $\lambda$ is the photon wavelength.
- **gain_energy.dat / *.gnu.plt**
  The optical gain as a function of photon energy $g(E)$ in units of [cm$^{-1}$].
  Photon Energy [eV] Optical Gain [1/cm]
- **gain_frequency.dat / *.gnu.plt**
  The optical gain as a function of frequency $g(\nu)$ in units of [cm$^{-1}$].
  Photon Frequency [THz] Optical Gain [1/cm]
- **gain_wavelength.dat / *.gnu.plt**
  The optical gain as a function of photon wavelength $g(\lambda)$ in units of [cm$^{-1}$].
  Photon wavelength [µm] Optical Gain [1/cm]

Negative values of the gain correspond to optical absorption.

**Gain-voltage characteristics**

- **GainMaxFrequency-Voltage.dat / *.gnu.plt**
  Source [V] Drain [V] Frequency of Max. Gain [THz]
  0 0 2.41798940e-001
  This file shows the frequency of the maximum value of the gain as a function of voltage. The first two columns contain the source and drain voltages. The third column is the frequency of the maximum gain at this voltage.
- **GainMaxFrequency-Voltage_Source.dat**
  GainMaxFrequency-Voltage_Drain.dat

  These files contain the same as discussed above but here only the source or drain voltages are contained, respectively, i.e. only one column for the voltages instead of two. It is easier to plot the data from one of these files compared to GainMaxFrequency-Voltage.dat.
- **GainMax-Voltage.dat / *.gnu.plt**
  0 0 -1.46451103e+000
This file shows the maximum value of the gain as a function of voltage in units of \([ \text{cm}^{-1} ]\). The first two columns contain the source and drain voltages. The third column is the maximum gain at this voltage. From this file, one can extract the voltage for threshold of gain.

- GainMax-Voltage_Source.dat
- GainMax-Voltage_Drain.dat

These files contain the same as discussed above but here only the source or drain voltages are contained, respectively, i.e. only one column for the voltages instead of two. It is easier to plot the data from one of these files compared to GainMax-Voltage.dat.

Transmission

- Transmission.dat

Transmission \( T(E) \) in units of [eV]

Energy [eV] Transmission (Source->Drain)

Does the transmission have a meaning in the actual calculation? Yes, it adds the ballistic part, i.e. the tunneling from source to drain to the current (compare with Landauer formula (insert reference)), see thesis page 65ff in PhD thesis of Peter Greck (check this).

It has been calculated from the self-consistently obtained conduction band profile. The transmission function is only the coherent ballistic contribution to the current, i.e. the current that goes directly from source to drain. The meaning of this output should be interpreted with care. There is also a noncoherent contribution to the current.

If one does a ballistic calculation then the total current is based on this transmission function (see Landauer formula).

Current density

- current_density_energy_resolved.avs.fld (or the corresponding *.gnu.plt / *.dat file)

  position and energy resolved current density \( j(z,E) \)

- current_density.dat / *.gnu.plt

  current density \( j(z) \)

  Position [nm] Current Density [A/cm^2]

Current-voltage characteristics (I-V curve)

- Current-Voltage.dat / *.gnu.plt

  Source [V] Drain [V] Current [A/cm^2]

  0 0 0.00000000e+000

  This file contains the current through the device (current-voltage or I-V characteristics). The first two columns contain the source and drain voltages. The third column is the current density.

- Current-Voltage_Source.dat
- Current-Voltage_Drain.dat

  These files contain the same as discussed above but here only the source or drain voltages are contained, respectively, i.e. only one column for the voltages instead of two. It is easier to plot the I-V characteristics from one of these files compared to Current-Voltage.dat.
11.6 Log file

Log file of the simulation (*.log)

```
nextnano.MSB v1.0.1(89)  
****************************************************************
****************************************************************
Parsing command line  
License: E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta 9 - my\License\License.xml  
Filename: E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period\THz_QCL_NovelDesignPeterGreck_1period.xml  
Output: E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period  
Materials: E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta 9 - my\nextnano.MSB\Materials.xml  
Materials: E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta 9 - my\nextnano.MSB\Materials_THz_QCL_NovelDesignPeterGreck_no_Varshni.xml  
****************************************************************
Checking license "E:\nextnano\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta 9 - my\License\License.xml"  
License info  
Name: Stefan Birner  
Email: abc@def.com  
Expiration: Wednesday, 2014-12-31  
Features  
MSB-1Band, OutputAvs, OutputGnuPlt.  
****************************************************************
Hardware concurrency: 8 <= Returns the number of hardware thread contexts.  
****************************************************************
Loading "E:\nextnano GmbH\nextnano.MSB\nextnano.MSB - Release beta 9 - my\nextnano.MSB\Materials_THz_QCL_NovelDesignPeterGreck_no_Varshni.xml"  
Database info  
Materials: 12 [#]  
Names: GaAsAlAs, InAs, InSb, GaSb, AlGaAs, AlGaAs_bowing_high, AlGaAs_bowing_low, InGaAs, GaAsSb, InAlAs, InAsSb  
****************************************************************
Loading "E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period\THz_QCL_NovelDesignPeterGreck_1period.xml"  
Header info  
```

Here the information written in the input file within Header is printed.

```
Header{
    ...
}
```

Information on variables and sweeps, such as temperatue sweep $T = 100, 250$.

```
Variables info  
QWDoping = 3e16  
QW_width_1 = 7.35  
Barrier_width_high = 2.1  
BarrierHigh = 0.27  
T = 100, 250  
```

Information on parameter sweep, such as temperature $T = 100$. 

416 Chapter 11. nextnano.MSB
Parameter sweep step 1 of 2

Variables info
  QWDoping = 3e+016
  QW_width_1 = 7.35
  Barrier_width_high = 2.1
  BarrierHigh = 0.27
  T = 100

Output info
  Directory: E:\nextnano output\THz_QCL_NovelDesignPeterGreck_1period

Device info
  Temperature: 100 [K]
  Grid info
    Spacing: 0.5 [nm]

The following information on the layers is only printed if "-debug 1" is specified as command line argument.

Layer info
  Width: 4.5 [nm]
  Doping: 3e-005 [nm^-3]
  AdjustBandedge: yes [yes|no]
  Probes: 100 [%]

Material info
  Base: GaAs

Layer info
  Width: 2.25 [nm]
  Doping: 0 [nm^-3]
  AdjustBandedge: yes [yes|no]
  Probes: 100 [%]

Material info
  Base: AlGaAs
  AlloyX: 27 [%]

Layer info
  Width: 7.25 [nm]
  Doping: 0 [nm^-3]
  AdjustBandedge: yes [yes|no]
  Probes: 100 [%]

Material info
  Base: GaAs

Information on the Source and Drain leads.

Lead info
  Name: Source
  Voltage info
    Value: 0 [V]

Lead info
  Name: Drain
  Voltage info
    Initial: 0 [V]
    Delta: 0.036 [V]
    Steps: 2 [#]
    Cluster: Center

Initialize Device

The following information is only printed if "-debug 1" is specified as command line argument.

Layer
  Width: 4.5 [nm]
  Nodes: 18 => Effective width 4.5nm (Error=0%)

Material GaAs
  Properties
    Mass: 0.067 [m0]

(continues on next page)
Bandedge:
  Offset: -1.519 [eV]
  Gap: 1.519 [eV]
  Alpha: 0 [eV/K]
  Beta: 0 [K]
  EpsStatic: 12.93 [eps_0]
Layer
  Width: 2.25 [nm]

Automatic adjustment of layer width and height.

Nodes: 9 => Effective width 2.25nm (Error=0%)

Error is related to AdjustBandedge feature. Here, grid spacing is 0.25 [nm] which is optimal. Therefore the Error is 0%.

Nodes: 5 => Effective width 2.5nm (Error=11.1111%)

Here, grid spacing is 0.50 [nm] which is too coarse to resolve the layer width. Therefore the Error is larger than 0%.

Material AlGaAs
  Alloy: 27% of AlAs and 73% of GaAs
  Properties
  Mass: 0.08941 [m0]
  Bandedge:
    Offset: -1.6621 [eV]
    Gap: 1.90092 [eV]
    Alpha: 0 [eV/K]
    Beta: 0 [K]
    EpsStatic: 12.1562 [eps_0]

The following information is only printed if `-debug 1` is specified as command line argument.

Material AlGaAs_bowing_high
  Alloy: 27% of AlAs and 73% of GaAs
  Properties
  Mass: 0.08941 [m0]
  Bandedge:
    Offset: -1.6621 [eV]
    Gap: 1.90092 [eV]
    Alpha: 0 [eV/K]
    Beta: 0 [K]
    EpsStatic: 12.1562 [eps_0]
Layer
  Width: 7.25 [nm]
  Nodes: 15 => Effective width 7.5nm (Error=3.44828%)

Log file continues...
MaxGreenIts: 25
MaxGreenDelta: 3e-07
MaxProbeIts: 15
MaxProbeNorm: 1e-009 [A/cm^2]
MaxPoissonOuterIts: 25
MaxPoissonInnerIts: 15
MaxPoissonDensityNorm: 1e+009 [1/cm^3]
MaxPoissonCorrectorNorm: 0.0001 [V]

Core info
IterativeProbes: 0
Flatband: 0

Scattering strength
  Const: 0 [eV]
  BP: 0 [eV]
  MSB: 1 []

Gain info
PhotonEnergyInitial: 0.001 [eV]
PhotonEnergyFinal: 0.03 [eV]
PhotonEnergySteps: 117 [#]
Cluster: Center

Kernel
  Building Hamiltonian
  Calculating

Voltage sweep step starts and MSB self-consistency cycle.

Voltage sweep step 1 of 2
Voltage info
Source: 0 [V]
Drain: 0 [V]

MSB self-consistency cycle
it= 1, norm2=2.478388e+003 [%]
   normI=6.779549e+001 [%]
it= 2, norm2=3.960580e+002 [%]
   normI=2.362588e+001 [%]
...
it= 5, norm2=7.928989e-001 [%]
   normI=3.995536e-002 [%]
Calculating Green's functions
Poisson iteration 0
it= 0, norm2=1.836019e+017 [cm^-3]
   normI=2.943621e+016 [cm^-3]
   charge=17 [%]
it= 1, norm2=1.511046e+018 [cm^-3]
   normI=3.661754e+017 [cm^-3]
   charge=1024 [%]
...
it= 7, norm2=1.771342e+007 [cm^-3]
   normI=4.557251e+006 [cm^-3]
   charge=100 [%]
Corrector: norm2=1.661272e-001 [V]
   normI=1.766968e-002 [V]

Writing output
Device
Density of States
Transmissions
Probes
Carrier Density
Current Density
Consistency checks

(continues on next page)
This DOS value is \( \text{MAX}(\text{ABS}(\text{DOS\_total} - \text{DOS\_Source} - \text{DOS\_Drain} - \text{DOS\_Probes})) \). It must be (numerically) zero.

Current: 0.000000e+000 [A/cm\(^2\)]
Current: nrm2=1.311401e-011 [A/cm\(^2\)]
Current: nrm1=8.209396e-012 [A/cm\(^2\)]
Current: diff=1.244363e-013 [A/cm\(^2\)]

MSB self-consistency cycle
it=1, norm2=4.457296e+002 [%]
normI=2.933419e+001 [%]
...
it=4, norm2=1.054857e-001 [%]
normI=6.648939e-003 [%]
Calculating Green's functions
Poisson iteration 1
it=0, norm2=1.920755e+015 [cm\(^{-3}\)]
normI=7.899763e+014 [cm\(^{-3}\)]
charge=100 [%]
...
it=2, norm2=2.411963e+005 [cm\(^{-3}\)]
normI=1.151118e+005 [cm\(^{-3}\)]
charge=100 [%]
Corrector: norm2=1.587036e-004 [V]
normI=3.504813e-005 [V]
Calculating optical gain
Setup gain calculation
Calculate Green's functions
Calculate gain for photon energies
Write output
Writing output
Device
Density of States
Transmissions
Probes
Carrier Density
Current Density
Consistency checks
DOS: -4.641658e-007 [1/eV/nm]
Current: 0.000000e+000 [A/cm\(^2\)]
Current: nrm2=5.709931e-011 [A/cm\(^2\)]
Current: nrm1=3.783379e-011 [A/cm\(^2\)]
Current: diff=1.383798e-012 [A/cm\(^2\)]

Duration of Voltage Step
Duration in seconds: 16
Duration in hours, minutes, seconds: 0h, 0', 16''
Voltage sweep step 2 of 2
Voltage info
Source: 0 [V]
Drain: 0.0928421 [V]
MSB self-consistency cycle
...
Voltage drop=100 [%]
Writing output
Device
...
it=3, norm2=3.391875e+005 [cm\(^{-3}\)]
normI=9.720889e+004 [cm\(^{-3}\)]
charge=100 [%]
Second Poisson iteration starts. It is related to MaxPoissonOuterIts. Make sure that this number is not exceeded!

Poisson iteration 2
it= 0, norm2=5.254027e+015 [cm^-3]
...  
it= 2, norm2=3.598211e+005 [cm^-3]
   normI=1.012272e+005 [cm^-3]
   charge=100 [%]
Corrector: norm2=7.631612e-005 [V]
   normI=8.710143e-006 [V]
Voltage drop=100 [%]
Calculating optical gain
Setup gain calculation
Calculate Green's functions
Calculate gain for photon energies
Write output
Writing output
Density of States
Transmissions
Probes
Carrier Density
Current Density
Consistency checks
DOS: -7.571542e-007 [1/eV/nm]
Current: 3.251042e+003 [A/cm^2]
Current: nrm2=2.033016e-007 [A/cm^2]
Current: nrmi=4.489969e-008 [A/cm^2]
Current: diff=4.750423e-008 [A/cm^2]
Duration of Voltage Step
Duration in seconds: 31
Duration in hours, minutes, seconds: 0h, 0', 31''
Duration of Parameter Sweep Step
Duration in seconds: 47
Duration in hours, minutes, seconds: 0h, 0', 47''
********************************************************************************

Parameter sweep step for T = 250 [K].

Parameter sweep step 2 of 2
Variables info
   QWDoping = 3e+016
   ...
   T = 250
Write output
   Writing output
   Device
   Density of States
   Transmissions
   Probes
   Carrier Density
   Current Density
   Consistency checks

11.6. Log file
11.7 Tutorials

11.7.1 Transmission coefficient of a double barrier structure

This tutorial is based on the following publication: [BirnerCBR2009]

Input files:

- \texttt{1D\_Transmission\_DoubleBarrier\_CBR\_paper.in} (input file for nextnano\textsuperscript{3} code)
- \texttt{1D\_Transmission\_DoubleBarrier\_CBR\_paper\_MSB.negf} (input file for nextnano.MSB code)
- \texttt{Materials\_1D\_Transmission\_DoubleBarrier\_CBR\_paper\_MSB.negf} (material database for nextnano.MSB code)

This example input file demonstrates how to calculate the transmission coefficient. We use the same structure as outlined in Section 4.4 of [BirnerCBR2009] and compare the results obtained with the MSB method to the results obtained with the CBR method.

Our structure consists of

\begin{verbatim}
GaAs | AlGaAs | GaAs | AlGaAs | GaAs
\end{verbatim}

where the barrier material is indicated in bold.

The following figure shows the conduction band edge profile and the probability density of this quasi-bound resonant state for the case of 10 nm barrier widths calculated with the nextnano\textsuperscript{3} software.

The barrier height is set to 0.1 eV.

The effective mass is set to 0.067 m\textsubscript{0} for both materials.

The widths of the AlGaAs barriers is varied between 2 nm, 4 nm and 6 nm. The nextnanomat feature Template or nextnanopy Sweep can be used to sweep the variables before execution.

\begin{verbatim}
\$BarrierThickness  \# (DisplayUnit:_nm) barrier thicknesses
\end{verbatim}

This simply generates multiple input files, which can be run in parallel (see \textit{Options: Simulation}).

In the XML format, the iterator feature was used:
In this case, single program execution simulates all the cases.

The following figure shows the calculated transmission coefficient $T(E)$ as function of energy for a double barrier structure with varying barrier widths of 2 nm, 4 nm and 10 nm (barrier separation 10 nm). At 25 meV there is a peak where the double barrier becomes transparent, i.e. $T(E) = 1$. This is exactly the energy that matches the resonant state in the well.

Figure 11.7.1.2: Transmission function for double barriers of different barrier thicknesses

The MSB results are very similar to the CBR results (see also Fig. 1 in [BirnerCBR2009]). The conduction band edge minimum is set to 0 eV. Note that the first peak of the 10 nm barrier structure is not well resolved for the MSB calculation in contrast to the CBR calculation. The reason is that the associated confined energy level is very sharp for the 10 nm structure and couples hardly to the source and drain leads. Therefore, the resolution of the
energy grid plays an important role in order to resolve the energy peak.

The following figure shows the local density of states (LDOS), $\rho(z, E_z)$, for the double barrier structure with a barrier thickness of 2 nm. The conduction band edge is indicated by the white line. The barrier height is 0.1 eV. The results look very similar to Fig. 2 in [BirnerCBR2009] calculated with the CBR method.

![Image of LDOS for double barrier with 2 nm barrier thickness](image1)

**Figure 11.7.1.3:** Local Density of states (LDOS) for the double barrier with 2 nm barrier thickness

The resonant state inside the double barrier is very broad with respect to energy because it couples strongly to the leads at the left and right boundaries. This is in contrast to the situation for the 10 nm barriers (not shown) where due to the large barrier widths the resonant state is quasi-bound, i.e. with a very sharp and high density of states at the resonance energy because of the very weak coupling to the contacts. Red (blue) color indicates high (low) density of states.

The following figure shows the calculated density of states $n(E)$ for the double barrier structures.

![Image of DOS for double barriers of different barrier thicknesses](image2)

**Figure 11.7.1.4:** Density of states (DOS) for double barriers of different barrier thicknesses

The first peak in the DOS for the 10 nm barrier structure differs substantially from the other two structures because it is extremely sharp and high. The second peak in the DOS at 87 meV due to the second confined well state is only visible for the 10 nm structure. This is consistent to the transmission coefficient (see Fig. above) which shows a sharp maximum only for the 10 nm structure at this energy.

**Further comments regarding the MSB input file**

In order to avoid solving the Poisson equation for this example we use the following parameters for the number of Poisson iterations.
In the database we have adjusted the material parameters so that we obtain a barrier height of 0.1 eV and a constant effective mass of 0.067 m₀ for both materials.

*Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.*

### 11.7.2 Ballistic current calculation of a GaAs nin resistor

This tutorial is based on the following publications: [BirnerCBR2009], [GreckPhD2012]

The following input files were used:

- **Input files for nextnano³ software**
  - 1D_nin_symmetric_PhDthesis_PeterGreck_cl_zero_bias.in
    classical calculation at zero bias, Fig. 8.1(a) of [GreckPhD2012]
  - 1D_nin_symmetric_PhDthesis_PeterGreck.in
    CBR calculation of Fig. 8.1 and Fig. 8.2 of [GreckPhD2012]
  - 1D_nin_symmetric_CBRpaper.in
    CBR calculation of Chapter 8.5 of [BirnerCBR2009], symmetric doping profile
  - 1D_nin_asymmetric_CBRpaper.in
    CBR calculation of Chapter 8.5 of [BirnerCBR2009], asymmetric doping profile

- **Input files for nextnano.MSB software**
  - nin_resistor_PhDthesis_PeterGreck.xml
    MSB calculation of Fig. 8.1 and Fig. 8.2 of [GreckPhD2012]
  - nin_resistor_CBRpaper_symmetric.xml
    MSB calculation of Chapter 8.5 of [BirnerCBR2009], symmetric doping profile
  - nin_resistor_CBRpaper_asymmetric.xml
    MSB calculation of Chapter 8.5 of [BirnerCBR2009], asymmetric doping profile

These example input files demonstrate how to calculate the ballistic current on a GaAs nin resistor. We use the same structure as outlined in Section 8.1 of [GreckPhD2012] and section 8.5 of [BirnerCBR2009] and compare the results obtained with the MSB method to the results obtained with the *CBR* method.

#### Example 1

The following figure shows the electron density and the conduction band edge profile for a 50 nm GaAs nin structure.

The following input files can be used to reproduce these results.

- **1D_nin_symmetric_PhDthesis_PeterGreck_cl_zero_bias.in** (input file for nextnano³ code)
  classical calculation at zero bias, Fig. 8.1(a) of [GreckPhD2012]

- **1D_nin_symmetric_PhDthesis_PeterGreck.in** (input file for nextnano³ code)
  CBR calculation of Fig. 8.1 and Fig. 8.2 of [GreckPhD2012]
The MSB results are very similar to the CBR results (not shown). Note that the CBR method uses an adaptive energy grid and a probe type as described in the text. Note that all four profiles almost perfectly match.

- nin_resistor_Phdthesis_PeterGreck.xml (input file for nextnano.MSB code)

MSB calculation of Fig. 8.1 and Fig. 8.2 of [GreckPhD2012]

The following results of the CBR input files are consistent to the MSB results for both, zero bias and nonzero bias. This has been checked.

- conduction band edges
- electric field
- electron density and energy resolved electron density
- DOS, position resolved DOS, lead resolved DOS (although the numbers are slightly different but the energy grid is also slightly different, i.e. nonuniform (CBR) vs. uniform (MSB) energy grid)
- Transmission
- IV curve
Example 2

The following input files can be used to reproduce the results of section 8.5 of [BirnerCBR2009].

- **1D_nin_symmetric_CBRpaper.in** (input file for nextnano³ code)
  - CBR calculation of Chapter 8.5 of [BirnerCBR2009], symmetric doping profile
- **1D_nin_asymmetric_CBRpaper.in** (input file for nextnano³ code)
  - CBR calculation of Chapter 8.5 of [BirnerCBR2009], asymmetric doping profile
- **nin_resistor_CBRpaper_symmetric.xml** (input file for nextnano.MSB code)
  - MSB calculation of Chapter 8.5 of [BirnerCBR2009], symmetric doping profile
- **nin_resistor_CBRpaper_asymmetric.xml** (input file for nextnano.MSB code)
  - MSB calculation of Chapter 8.5 of [BirnerCBR2009], asymmetric doping profile

Again, we obtained very good agreement between the CBR and the MSB algorithm.

**Important comment**

Here, we had to use more energy grid points compared to the CBR code (where we used 300) in order to obtain convergence.

```
Nodes 501 # Comment = Number of energy grid points.
```

We noticed this because the number of outer Poisson iterations exceeded its maximum as can be seen in the .log file.

```
Poisson iteration 60
```

The number of outer Poisson iterations should be increased as follows.

```
MaxPoissonOuterIts 60 # Comment = Max. outer Poisson iterations where G*R is recalculated.
```

**Further comments regarding the MSB input file**

In order to calculate the current ballistically we switched off scattering by using the following flag.

```
BallisticCalculation yes
```

*Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.*

### 11.7.3 Resonant tunneling diode (RTD)

This tutorial is based on the following publications: [GreckPhD2012]

The following input files were used:

- **RTD_PhDthesis_PeterGreck_ballistic.negf** (MSB calculation of Fig. 8.2 of [GreckPhD2012])
- **RTD_PhDthesis_PeterGreck.negf** (MSB calculation of Fig. 8.2 of [GreckPhD2012])

These example input files demonstrate how to calculate the current in a Resonant Tunneling Diode (RTD). In RTD, quantum mechanical effects are essential. We use the same structure as outlined in Section 8.2 of [GreckPhD2012].

We perform two simulations:

- **RTD_PhDthesis_PeterGreck_ballistic.negf** - calculation without scattering (ballistic calculation)
- **RTD_PhDthesis_PeterGreck.negf** - calculation including scattering
The RTD structure consists of 48 nm GaAs.
In the central region, there are two 4 nm wide AlGaAs barriers enclosing a 6 nm wide GaAs well. These two AlGaAs barriers are separated by a 4 nm intrinsic region.
Outside the barriers, the structure is doped with a concentration of $10^{17}$ cm$^{-3}$.
The barrier height was taken to be 0.156 meV.
The temperature is set to 150 K.
The grid spacing is 1.0 nm.
Further material parameters used are:
- effective mass: 0.067 (GaAs) and 0.0795 (Al$_{0.15}$Ga$_{0.85}$As)
- static dielectric constant: 10.89 (GaAs) and 10.4808 (Al$_{0.15}$Ga$_{0.85}$As)
- LO phonon energy: 0.035 eV (GaAs) and 0.0373 (Al$_{0.15}$Ga$_{0.85}$As)
The following figure shows the calculated current-voltage characteristics. It corresponds to Fig. 8.2 of [Greck-PhD2012].

![Graph showing current-voltage characteristics](image)

Figure 11.7.3.1: Fig. 8.2 of PhD thesis of Peter Greck

The relevant lines are gray (Ballistic) for the ballistic calculation and blue (MSB) for the calculation including scattering. The peak current density of the ballistic calculation is reached at about 100 mV where the chemical potential of the source contact is aligned with the lowest quantum well state. However, this result is physically wrong. This is due to the fact that a ballistic calculation neglects any kind of incoherent scattering. In a realistic device the carriers do not ballistically reach the RTD. Instead, a bound state is formed by the triangular shaped potential in front of the RTD and inelastic scattering processes lead to a capture of carriers within this state. Since the energy of this bound state is lower than the chemical potential of the source contact, the alignment with the RTD resonance is realized at higher bias voltages. For the calculated RTD, the peak current density is shifted from 100 mV to 140 mV.

Results
Figure 11.7.3.2: Calculated current-voltage characteristics of the ballistic calculation (gray) and the calculation including scattering (red). The peak current density is at 100 mV for the ballistic calculation but is shifted to 140 mV for the calculation that includes scattering. The file that is plotted is called Current-Voltage_Drain.dat.

The following figures shows the conduction band edges (BandProfile/BandEdge_conduction.dat) together with the

- local density of states $\rho(x, E)$ (DOS/DOS_position_resolved.avs.fld)
- electron density $n(x, E)$ (CarrierDensity/CarrierDensity_energy_resolved.avs.fld)
- current density $j(x, E)$ (CurrentDensity/CurrentDensity_energy_resolved.avs.fld)

at the peak current densities of 100 mV (left figures, ballistic calculation) and 140 mV (right figures, calculation including scattering). Note that the peak current density of the ballistic calculation (100 mV) deviates from the peak current density (140 mV) of the calculation including scattering.

Local density of states $\rho(x, E)$ plots:
Electron density $n(x, E)$ plots:
Current density $j(x, E)$ plots:

**Further comments regarding the MSB input file**

In order to calculate the current ballistically, we switched off the scattering by the following flag.

```
MSB_1Band{
  Core{
    BallisticCalculation = yes
  }
}
```

Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.
Figure 11.7.3.3: For the ballistic calculation (left figure), the quantum well state is aligned energetically with the chemical potential of the left contact (source). For the scattering calculation (right figure), the quantum well state is aligned energetically with the triangular quantum well state left of the barrier.

Figure 11.7.3.4: For the ballistic calculation (left figure), the quantum well state is aligned energetically with the chemical potential of the left contact (source). For the scattering calculation (right figure), the quantum well state is aligned energetically with the triangular quantum well state left of the barrier.
11.7.4 Quantum Well

This tutorial is based on the following nextnano³ tutorial

- Applying the NEGF method to a quantum well

and on the following paper: [KubisNEGF2005]

The following input files were used:

- QW_InGaAs_ballistic.negf (input file for nextnano.MSB code)
- QW_InGaAs_scattering.negf (input file for nextnano.MSB code)
- Materials_QW_InGaAs.negf (material database for nextnano.MSB code)
- 1DNEGF_InGaAs_QW_ballistic_CBR.in (input file for nextnano³ code, CBR method)
- 1DNEGF_InGaAs_QW_ballistic.in (input file for nextnano³ code, NEGF method)
- 1DNEGF_InGaAs_QW_scattering.in (input file for nextnano³ code, NEGF method)
- 1DNEGF_InGaAs_QW_scattering_bias.in (input file for nextnano³ code, NEGF method)

This example input file demonstrates how to calculate the current across a quantum well. It compares the results of a ballistic calculation to the results of a calculation including scattering. Here, input files for both the nextnano.MSB software and for the nextnano³ software are provided so that the MSB algorithm can be benchmarked against the full NEGF algorithm as implemented by T. Kubis. We use the database file called Materials_QW_InGaAs.negf where we adjusted the default material parameters so that they match the publication of [KubisNEGF2005].

Our structure consists of a 12 nm In₀.₁₄Ga₀.₈₆As QW in the center surrounded on each side by GaAs barriers (of width 19 nm each).

GaAs | InGaAs | GaAs

Figure 11.7.3.5: For the ballistic calculation (left figure), the quantum well state is aligned energetically with the chemical potential of the left contact (source). For the scattering calculation (right figure), the quantum well state is aligned energetically with the triangular quantum well state left of the barrier. On the right figure one can nicely see that the carriers that are injected from the left contact (source) scatter into the triangular quantum well state and then tunnel through the double barrier into the right contact (drain).
where the barrier material is indicated in bold.

**Further comments regarding the MSB input file**

- We used 1000 energy grid points. In contrast, the NEGF simulation used much less.
- In the database we have adjusted the material parameters, e.g. for InGaAs we used the effective mass of GaAs, the static and optical dielectric constant of GaAs and the GaAs LO phonon energy for simplicity. We use a conduction band offset of 0.150 eV.

### 11.7.5 AlGaAs/GaAs THz QCL

**Input file:**

- `THz_QCL_Fathololoumi_OpticsExpress2012.xml` (old format)
- `THz_QCL_Fathololoumi_OpticsExpress2012.negf` (since 2022-04)

The tutorial is based on this paper: [FathololoumiOE2012]

This paper describes a THz quantum cascade laser (QCL) with a performance up to 199.5 K. The QCL layout is based on a resonant phonon three well design.

The input file in the XML format (old) uses constants (e.g. `Barrier` for the aluminum content of the barriers) and iterators for temperature sweep. In this case, single program execution simulates all the cases.

The input file in the .negf format (new) uses `nextnano++`-style variables starting with dollar sign (see Syntax definition). Then, the `nextnanomat` feature `Template` or `nextnanopy Sweep` can be used to sweep the variables before execution. This simply generates multiple input files, which can be run in parallel (see Options: Simulation).

Furthermore, in these input files the usage of so-called clusters is demonstrated. A cluster is a set of layers which is specified via `BeginCluster` and `EndCluster`, where the last layer is inclusive. The clusters are used to specify a certain voltage which is extended to the whole device and to define the region in which the optical gain is calculated. This is especially useful if more than one period is calculated.

The input file can be edited by the users to input the structure of a device and some further necessary input parameters. The details of the syntax of the input files is explained in the input file documentation.

Our structure consists of GaAs wells and Al_{0.15}Ga_{0.85}As barriers of width

\[
\begin{align*}
4.0 \text{ nm} & | 4.3 \text{ nm} | 8.9 \text{ nm} | \textbf{2.46 nm} | 8.15 | 4.1 \text{ nm} | 5.5 \text{ nm} + 5.0 \text{ nm} + 5.5 \text{ nm} = 16.0 \text{ nm} | \textbf{4.3 nm} | 4.0 \text{ nm}
\end{align*}
\]

where the barrier material is indicated in **bold** and the doped region in *italics*. The doping concentration is \(6 \cdot 10^{16} \text{ cm}^{-3}\).

The QCL structure consists of the actual QCL period which is embedded between two 4.0 nm wide GaAs regions that serve as the leads at the left and right boundary. These lead layers are also doped. The concentration is \(1 \cdot 10^{16} \text{ cm}^{-3}\).

The QCL design consists of three quantum wells (GaAs) and three barriers (AlGaAs). On the right, just before the lead, we have to add an additional barrier arising from the next QCL period. The total length of the structure as specified above is 56.21 nm. As we are using a 0.5 nm grid, our simulated structure essentially has a length of 56.5 nm. The QCL is designed with respect to an electric field of -12.2 kV/cm. This corresponds to a bias voltage of 68.93 mV for our 56.5 nm structure.

Each layer of the structure is defined within the keyword `<Layer>` where the material name is given, its width and a doping concentration if applicable.

```
Layer{
  Thickness = 5.0 # Unit = nm
  Material{
    Base = AlAs
  }
  Probes = 1.0
}
```

(continues on next page)
An alloy like Al_{0.15}Ga_{0.85}As is specified as follows:

```
Material{
  Base = "Al_{x}Ga_{1-x}As"
  AlloyX = 0.15
}
```

The following figure shows the conduction band profile at -12.2 kV/cm.

The gain has been calculated between 3 and 30 meV in the cluster called Center. The photon energy interval includes of 100 photon energies.

```
Gain{
  Cluster = Center  # Comment = Specify the cluster where optical gain should be calculated.
  PhotonEnergyInitial = 1e-3 # Unit = eV # Comment = Min. photon energy for gain calculation.
  PhotonEnergyFinal = 30e-3 # Unit = eV # Comment = Max. photon energy for gain calculation.
  PhotonEnergySteps = 117 # Unit = # Comment = Number of photon energies for gain calculation.
}
```

The term Center refers to the Center entries in

```
BeginCluster = "Center"
```

and

```
EndCluster = "Center"
```

There is also a Center entry in the Lead called Drain. Here, the applied voltage drops over the layers that are inside the Center region.

a) Specify Center

```
Lead{
  Name = Drain
  Voltage{
    Cluster = Center
  }
}
```

b) or QCL.

```
Lead{
  Name = QCL
  Voltage{
    Cluster = Center
  }
}
```

Both are possible because BeginCluster and EndCluster contain both labels.

In this tutorial, we have enabled, i.e. 1.0, the scattering within the AlGaAs barriers. For each layer we have defined a variable with the name BarrierProbes which we set to 1.0.
$\text{BarrierProbes} = 1.0 \quad \# \text{Enable/disable scattering within barriers. This can reduce computational time with minor influence on overall results.}

Consequently, for each AlGaAs barrier layer we have included the constant $\text{BarrierProbes}$.

\begin{verbatim}
Layer{
  Material{
    Base = "Al\{x\}Ga\{1-x\}As"
    AlloyX = $\text{Barrier}$
  }
  Probes = $\text{BarrierProbes}$
}
\end{verbatim}

The simulation took

\begin{verbatim}
Duration in days, hours, minutes, seconds: 5h 50'34''
\end{verbatim}

on an Intel Core i5-3470S (2.9 GHz) for a 0.5 nm grid spacing.

*Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.*
12.1 Introduction

12.1.1 What is nextnanopy?

nextnanopy is a Python package for nextnano++, nextnano\(^3\), nextnano.NEGF and nextnano.MSB software to automate nextnano simulations and analysis of the results. You can write a Python script to:

- run multiple simulations (multidimensional parameter sweep, many input files, etc.)
- plot multiple figures, overlay data, save as image or PDF
- calculate further quantities using output data
- load polygons from a GDSII file for device geometry input

and anything else possible with the Python language. We maintain the nextnanopy Python package on GitHub as well as on the Python Package Index (PyPI) repository.

12.1.2 How do I install it?

Instructions are on our GitHub site.

Additional to the package installation, you need a valid nextnano license and a configuration file for nextnanopy, if you want to execute a nextnano product.

To easily use your settings and paths already stored in the workflow manager nextnanomat, you can export these settings into a nextnanopy config format. Documented on the following page: Generate nextnanopy Config File.

12.1.3 Where to start?

You can start with the following tutorials: Tutorials. We also provide sample scripts for nextnano++, nextnano\(^3\), nextnano.NEGF and nextnano.MSB software.

12.2 Tutorials

The following links direct you to our tutorials on GitHub. There are Jupyter Notebook (.ipynb) and reStructured-Text (.rst) versions. We recommend opening the Jupyter Notebook (.ipynb) files.

- Tutorial 0 - Set up the configuration
- Tutorial 1 - Execute an input file
- Tutorial 2 - Plotting data files
- Tutorial 3 - Load polygons from a GDSII file
• Tutorial 4 - Sweep to automate the execution
• Tutorial 5 - Navigation in output folder using DataFolder

In addition, we provide sample scripts to run and postprocess nextnano++, nextnano³, nextnano.NEGF and nextnano.MSB simulations.
13.1 1D Tutorials

13.1.1 Basic Tutorial 1 for nextnano++

This tutorial is the first in an introductory series where we will guide you into creating your first nextnano++ input file for performing simulations. The goal of this tutorial is to create a basic 1D structure with one material grown on top of another.

The keyword list that you may find helpful can be downloaded [here](#).

Specifying global() parameters

```plaintext
global{ # required
    simulate1D()
    substrate{ name = "GaAs" } # required
    temperature = 300 # required
    crystal_zb{
        x_hkl = [1, 0, 0]
        y_hkl = [0, 1, 0]
    }
}
```

The above code block specifies the global parameters used in the simulation. simulate1D() is used because this is a 1D structure. We then specify the substrate with substrate{ name = "GaAs"}. GaAs exists in nextnano++’s database, so the default values are used. To change parameters of the compound, refer to the guide [here](#).

We then set the temperature to 300K with temperature{}. By default, this is a temperature dependent bandgap, with the correction calculated with the Varshni formula. To use a temperature independent bandgap, one can specify `temperature_dependent_bandgap = no`.

Using crystal_zb{}, we specify the orientation of the crystal coordinate system with respect to the simulation coordinate system. For zinc blende materials, we specify the (hkl) plane perpendicular to the x and y directions. The z direction is determined automatically here, so only x_hkl and y_hkl are needed.
Specifying the computational grid()

```plaintext
grid{  # required
    xgrid{ line{ pos 0.0 spacing 0.1 }
         line{ pos 50.0 spacing 0.5 } }
}
```

The block for grid() has two functions. It specifies the device size so at least two paremeters are required, and it also specifies the grid over which the computation is run over.

The block above tells us that the device length ranges from $x = 0$ nm to $x = 50$ nm. The $x = 0$ nm point has a spacing of 0.1 nm between grid lines, and the $x = 50$ nm has a spacing of 0.5 nm between points. An exponential grid is interpolated between the two lines as the grid specified is not equidistant. One can view the grid spacing in the visualisation of output, which we will return to later.

Specifying the structure() of the device

```plaintext
structure{ # required
    region{
        binary{ name GAs }
        contact{ name whatever }
        everywhere()
    }
    region{
        binary{ name InAs }
        line{ x [ 20.0, 30.0 ] }
    }
}
```

This section specifies the structure of the device. As a rule, it is important to remember that newly-defined regions overwrite any older region that is overlapped. The first region is specified to be GaAs within binary() and it is given the dummy contact name whatever. Note that this does not specify the type of contact (e.g. Schottky, Fermi, Ohmic etc.) but the name we assign the contact. This is useful when we want to specify parameters of the contact under the grouping contacts().

The second region is specified to be InAs and is only from 20 nm to 30 nm.

Specifying contact() parameters

```plaintext
contacts{
    fermi{ name whatever bias 0.0 }
}
```

As mentioned earlier, we can specify the parameters of the contact using the name we have given it. In this tutorial, we give the whatever contact a bias of 0 V.

Solving parameters

```plaintext
classical{ # required
    Gamma{} HH{} LH{} 
    output_bandedges{ averaged no} # necessary to see anything related to the design
}
```

We indicate that we want the Gamma, heavy hole and light hole bands solved with Gamma{}, HH{}, and LH{} respectively. We also output the bandedges with the command output_bandedges.
Running the simulation

To run the simulation file, we can press F8 on the keyboard, or click the icon.

Viewing output

We can view the output of the simulation under the “Output” tab at the top of nextnano mat. Navigate to the folder bias_00000. Click on bandedges.dat. A figure of the Gamma, LH and HH band edges should be visible. As seen on the x-axis, the bandedge is different from 20nm to 30nm, which is where we specified InAs to be.

We can view the grid points used in the simulation by checking the box “Show grid” in the menu on the left of nextnano mat.

To export the figure as a .plt file, click on the icon at the top right corner.

Then click on bandedges.dat. Hold down shift on the keyboard and click the bandedges of interest. In this tutorial, Gamma [eV], HH [eV] and LH [eV] are chosen from the bottom right panel. Press a on the keyboard or the icon at the top right corner of nextnano mat.

Following which, select icon at the top and choose the option “Create and Open Gnuplot File (*.plt) from Items of Overlay”. A Gnuplot window should pop up. Click the icon and name the file, and save it.

The above figure shows what one should see as output.

Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.
13.1.2 pn junction

Author Stefan Birner

Input files for nextnano³ software

• GaAs_pn_junction_1D_nn3.in
• GaAs_pn_junction_1D_QM_nn3.in
• GaAs_pn_junction_2D_nn3.in
• GaAs_pn_junction_3D_nn3.in
• GaAs_pn_junction_1D_ForwardBias_nn3.in
• GaAs_pn_junction_2D_ForwardBias_nnp.in
• GaAs_pn_junction_3D_ForwardBias_nnp.in

Input files for nextnano++ software

• GaAs_pn_junction_1D_nnp.in
• GaAs_pn_junction_2D_nnp.in
• GaAs_pn_junction_3D_nnp.in
• GaAs_pn_junction_1D_ForwardBias_nnp.in

This tutorial discusses the nextnano++ input file. Identical results can be achieved with the nextnano³ input files listed above. The keyword list that you may find helpful can be downloaded here.

This tutorial aims to reproduce Figure 3.1 (p. 51) of Joachim Piprek’s book “Semiconductor Optoelectronic Devices - Introduction to Physics and Simulation” (Section 3.2 “pn-junctions”)

Doping concentration

The structure consists of 300nm GaAs. At the left and right boundaries, metal contacts are connected to the GaAs semiconductor (i.e. from 0nm to 10nm, and from 310nm to 320nm). The structure is p-type doped from 10nm to 160nm and n-type doped from 160nm and 310nm.

The following figure shows the concentration of donors and acceptors of the pn-junction. In the p-type region between 10nm and 160nm, the number of acceptors, $N_A$ is $0.5 \times 10^{18}$ cm$^{-3}$. In the n-type region between 160nm and 310nm, the number of donors, $N_D$ is $2.0 \times 10^{18}$ cm$^{-3}$.
Carrier concentrations

The equilibrium condition for a pn-junction is achieved by a small transfer of electrons from the n region to the p region, where they recombine with holes. This leads to a depletion region (depletion width = \( w_p + w_n \)), i.e. the region around the pn-junction only has very few free carriers left. The following figure shows the electron and hole densities and the depletion region around the pn-junction at 160 nm. Here, we assumed that all donors and acceptors are fully ionized.

Net charges (space charge)

In the depletion region, a net charge results from the ionized donors \( N_D \) and ionized acceptors \( N_A \). The following figure shows the net charge density of the pn-junction.

Electric field

The slope of the electric field is proportional to the net charge (Poisson equation), thus the extremum of the electric field is expected to be at the pn-junction. In regions without charges, the electric field is zero. The following figure shows the electric field of the pn-junction.

The extremum of the electric field \( F_{\text{max}} \) (at 160nm) can be approximated as follows:

\[
F_{\text{max}} = \frac{-eN_A w_p}{\varepsilon\varepsilon_0} = -6.997 \times 10^{14} \text{V/m}^2 w_p = 387 \text{kV/cm}
\]

\[
= \frac{-eN_D w_n}{\varepsilon\varepsilon_0} = -2.799 \times 10^{15} \text{V/m}^2 w_n = 386 \text{kV/cm}
\]
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e$</td>
<td>$1.6022 \times 10^{-19}$ As</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>12.93 (Dielectric constant of GaAs)</td>
</tr>
<tr>
<td>$\epsilon_0$</td>
<td>$8.854 \times 10^{12}$ As/(Vm)</td>
</tr>
<tr>
<td>$N_A$</td>
<td>$0.5 \times 10^{18}$ cm$^{-3}$</td>
</tr>
<tr>
<td>$N_D$</td>
<td>$2.0 \times 10^{18}$ cm$^{-3}$</td>
</tr>
<tr>
<td>$w_p$</td>
<td>55.3 nm</td>
</tr>
<tr>
<td>$w_n$</td>
<td>13.8 nm</td>
</tr>
</tbody>
</table>

**Electrostatic potential, conduction and valence band edges**

In regions, where the electric field is zero, the electrostatic potential is constant. The electrostatic potential $\phi$ determines the conduction and valence band edges:

- $E_c = E_c^0 - e\phi$
- $E_v = E_v^0 - e\phi$

The following figure shows the conduction and valence band edges, the electrostatic potential and the Fermi level of the pn-junction.

Without external bias (i.e. equilibrium), the Fermi level $E_F$ is constant ($E_F = 0$eV).

The built-in potential $\phi_{bi}$ was calculated by nextnano++ to be equal to 1.426 V. It can be approximated as follows:

$$\phi_{bi} = F_{max}(w_p + w_n)/2$$

Assuming $F_{max} = 387$ kV/cm, this would result in a depletion width: $w_p + w_n = 73.7$ nm
To allow for a constant chemical potential (i.e. constant Fermi level $E_F$), a total potential difference of $-e\phi_{bi}$ is required.

**Quantum mechanical solution**

Using the nextnano³ input file `GaAs_pn_junction_1D_QM_nn3.in`, we can solve the Schroedinger equation for the electrons, light and heavy holes in the single-band approximation over the whole device, rather than classically. We calculate up to 300 eigenvalues for each band. Thus the electron and hole densities are calculated **purely quantum mechanically**. The following figure shows the electron and hole concentrations for the classical and quantum mechanical calculations. For the QM calculations, different boundary conditions were used.

- **Dirichlet** boundary conditions force the wave functions to be zero at the boundaries, thus the density goes to zero at the boundaries which is unphysically.

- **Neumann** boundary conditions lead to unphysically large values at the boundaries.

For the classical calculation, the densities at the boundaries are constant. Nevertheless, in the interesting region around the pn-junction, all four options lead to identical densities.

The following figure shows the band edges of the pn-junction for the four cases:

- Classical calculation
- Quantum mechanical calculation with **Dirichlet** boundary conditions
- Quantum mechanical calculation with **Neumann** boundary conditions
- Quantum mechanical calculation with mixed boundary conditions (this feature is no longer supported)

For all cases the band edges are identical in the area around the pn-junction. Tiny deviations exist at the boundaries of the device.

This figure is a zoom into the right boundary of the conduction band edge. On this scale, the tiny deviations for the different boundary conditions can be clearly seen.
Nonequilibrium

So-called “quasi-Fermi levels” which are different for electrons ($E_F, n$) and holes ($E_F, p$) are used to describe nonequilibrium carrier concentrations.

In equilibrium the quasi-Fermi levels are constant and have the same value for both electrons and holes ($E_{F_n} = E_{F_p} = 0$ eV). The current is proportional to the mobility and the gradient of the quasi-Fermi level $E_F$.

2D/3D Simulations

- $GaAs_{-}pn_{-}junction_{-}2D_{-}nn3.in$ / $*_nnp.in$ - input file for the nextnano$^3$ and nextnano++ software
- $GaAs_{-}pn_{-}junction_{-}3D_{-}nn3.in$ / $*_nnp.in$ - input file for the nextnano$^3$ and nextnano++ software

These input files are for the same pn-junction structure as in the 1D case, but extended into 2D and 3D.

- 2D: rectangle of dimension 320 nm x 200 nm
- 3D: cuboid of dimension 320 nm x 200 nm x 100 nm
Complete input file for nextnano++

```plaintext
#***************************************************************************!
# !
# pn_junction_GaAs_1D_nnp.in
# --------------------------
# !
# This is an input file for nextnano++ to calculate the band edges of a
# simple pn-junction with classical charge densities.
# !
# It's part of the 1D pn-junction tutorial which can be found at:
# !
# https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_pn_junction.htm
# !
# For help on the individual keywords please go to
# !
# https://www.nextnano.com/nextnanoplus/software_documentation/input_file.htm
# !
# nextnano (c) nextnano GmbH
# !
# This input file is (c) Stefan Birner, nextnano GmbH.
# !
# This file is protected by applicable copyright laws. You may use it
# !
# within your research or work group, but you are not allowed to give
# !
# copies to other people without explicit permission.
# !
# !
# Documentation: https://www.nextnano.com/nextnanoplus/
# !
# Support: support@nextnano.com
# !
#***************************************************************************!

## Global

```plaintext
**simulate1D**

```plaintext
**temperature** = 300.0  # Kelvin

**substrate**

```plaintext
**name** = "GaAs"

**crystal_zb**

```plaintext
**x_hkl** = [1, 0, 0]
**y_hkl** = [0, 1, 0]

```plaintext

## Grid

```plaintext
# For consistency reasons, we use the same nonuniform grid spacing as the
# nextnano3 input file.
# However, using jumps in the grid spacing (e.g. at x=100.0 where the grid
# spacing changes abruptly)
# is not a good practice, as numerical errors increase.
# xgrid

```plaintext
**line**

```plaintext
pos = 0.0  spacing = 2.0
pos = 10.0  spacing = 2.0
pos = 10.0  spacing = 1.0
pos = 100.0  spacing = 1.0
pos = 100.0  spacing = 0.5
pos = 140.0  spacing = 0.5
pos = 140.0  spacing = 0.25
pos = 180.0  spacing = 0.25
pos = 180.0  spacing = 0.5
pos = 220.0  spacing = 0.5
pos = 220.0  spacing = 1.0
pos = 310.0  spacing = 1.0
pos = 310.0  spacing = 2.0
```

(continues on next page)
```plaintext
line{ pos 320.0 spacing 2.0 }
}

structure{
  output_region_index{ boxes no }
  output_material_index{ boxes no }
  output_alloy_composition{ boxes no }
  output_impurities{ boxes no }

  region{
    everywhere{}
    binary{ name "GaAs" }
  }
  region{
    line{
      x [0.0, 10.0]
    }
    binary{ name "GaAs" }
    contact { name "s"u"r"c }
  }
  region{
    line{
      x [10.0, 310.0]
    }
    binary{ name "GaAs" }
  }
  region{
    line{
      x [310.0, 320.0]
    }
    binary{ name "GaAs" }
    contact { name "d"r"a"n"c }
  }

  region{
    line{
      x [0.0, 160.0]
      # x = [10.0, 160.0] # doping must not start at 10.0
    }
    doping{
      constant{
        name "p-type"
        conc 0.5e18
      }
    }
  }

  region{
    line{
      # x = [160.0, 310.0] # doping must not end at 310.0
      x [160.0, 320.0]
    }
    doping{
      constant{
      }
    }
  }
}
```

(continues on next page)
13.1.3 Triangular well

In this tutorial we calculate the Schrödinger equation for a triangular well and compare the results with the analytic solution.

The related input files are followings:

- 1DGaAs_triangular_well_nn*.in
Structure

A triangular well consists of a potential with a constant slope that is bound at one side by an infinite barrier.

For \( x < 0 \) nm we have an infinite barrier. In our case it is represented by a huge conduction band offset of 100 eV to avoid any wavefunction penetration into the barrier.

For \( x > 0 \) nm we have a linear potential of \( V(x) = eFx \).

\( V(x) \) describes a charge \( e \) in an electric field \( F \) where the product \( eF \) is assumed to be positive.

Comparison of nextnano++ and the analytic solution

The Schrödinger equation for the transverse component of the electronic wave function has the following form inside the well:

\[
\left[ -\frac{\hbar^2}{2m^*} \frac{d^2}{dx^2} + eFx \right] \psi(x) = E\psi(x)
\]

Usually one applies Dirichlet boundary conditions at \( x = 0 \) nm so that \( \psi(x = 0) = 0 \) in order to represent an infinite barrier, i.e. the high barrier prevents significant penetration of electrons into the barrier region.

In our case, we apply Neumann (or Dirichlet) boundary conditions at \( x = -10 \) nm and \( x = 150 \) nm and let the infinite barrier be represented by the huge conduction band offset of 100 eV. Then, both boundary conditions lead to the same eigenenergies for the lowest eigenstates.

The Schrödinger equation can be simplified by introducing suitable new variables and thus reduces to the Stokes or Airy equation. Its solutions, the so-called Airy functions, are discussed in most textbooks, see for example:


The figure shows the conduction band edge (black line) which is represented by a triangular potential well \( V(x) = eFx \). Also shown are the four lowest energy levels and corresponding wave functions. The electric field that has been applied is \( F = 5 \) [MV/m], i.e. \( 0.05V/10 \) nm. The effective electron mass has the value \( 0.067m_0 \) (GaAs).

One can see that the distance between the energy levels decreases with increasing \( n \) because the quantum well width gets larger for higher energies. Note that in a parabolic well, the energy levels are equally spaced whereas in an infinitely deep square well, the energy level separation increases with increasing energy.
The eigenvalues of the Airy equation can be calculated using the formula:

\[ E_n = c_n \left( \frac{eFh^2}{2m^*} \right)^{1/3} \]

(The units of \( E_n \) in this equation are [J].)

The lowest eigenvalue has the value \( c_1 = 2.338 \).

For large \( n \), \( c_n \) can be approximated by the following equation which can be derived from WKB theory (named after Wentzel, Kramers and Brillouin):

\[ c_n \approx \left[ \frac{3}{2} \pi (n - \frac{1}{4}) \right]^{2/3} \]

The nextnano++ and nextnano3 eigenvalues for the lowest four eigenstates are in very good agreement with the analytic results:

<table>
<thead>
<tr>
<th>( n )</th>
<th>nextnano++ eigenvalue</th>
<th>nextnano3 eigenvalue</th>
<th>calculated eigenvalue</th>
<th>( c_n ) (exact)</th>
<th>( c_n ) (approximated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.05647</td>
<td>0.05644</td>
<td>0.05664</td>
<td>( c_1 = 2.338 )</td>
<td>( c_1 = 2.320251 )</td>
</tr>
<tr>
<td>2</td>
<td>0.09887</td>
<td>0.09882</td>
<td>0.09889</td>
<td>( c_2 )</td>
<td>( c_2 = 4.081810 )</td>
</tr>
<tr>
<td>3</td>
<td>0.13358</td>
<td>0.13351</td>
<td>0.13365</td>
<td>( c_3 )</td>
<td>( c_3 = 5.517164 )</td>
</tr>
<tr>
<td>4</td>
<td>0.16426</td>
<td>0.16416</td>
<td>0.16435</td>
<td>( c_4 )</td>
<td>( c_4 = 6.784455 )</td>
</tr>
</tbody>
</table>

The triangular potential is not symmetric in \( x \), thus the wave functions lack the even or odd symmetry that one obtains for the infinitely deep square well.

The triangular well model is useful because it can be used to approximate the (idealized) triangularlike shape near a heterojunction formed by the discontinuity of the conduction band and an electrostatic field of electrons or remote ionized impurities.

### 13.1.4 Double Quantum Well

This tutorial calculates the energy eigenstates of a double quantum well. This aims to reproduce two figures (Figs. 3.16, 3.17, p. 92) of Paul Harrison’s excellent book “Quantum Wells, Wires and Dots” (Section 3.9 “The Double Quantum Well”), thus the following description is based on the explanations made therein. **We are grateful that the book comes along with a CD so that we were able to look up the relevant material parameters and to check the results for consistency.**

Input files for both the nextnano++ and nextnano3 software are available.

The following input file is used:

* DoubleQuantumWell_6_nm_nn*.in

**Structure: AlGaAs / 6nm GaAs / AlGaAs / 6nm AlGaAs / AlGaAs**

Our symmetric double quantum well consists of two 6 nm GaAs quantum wells, separated by a Al\(_{0.2}\) Ga\(_{0.8}\) As barrier and surrounded by 20 nm Al\(_{0.2}\) Ga\(_{0.8}\) As barriers on each side. We thus have the following layer sequence: 20 nm Al\(_{0.2}\) Ga\(_{0.8}\) As / 6 nm GaAs / Al\(_{0.2}\) Ga\(_{0.8}\) As / 6 nm GaAs / 20 nm Al\(_{0.2}\) Ga\(_{0.8}\) As. (The barriers are printed in bold.)

In this tutorial, we demonstrate the following two examples:

1. we set the thickness of the Al\(_{0.2}\) Ga\(_{0.8}\) As barrier that separates the two quantum wells 4 nm and calculate the lowest two eigenstates.
2. we vary the thickness of the barrier layer from 1 nm to 14 nm fixing the width of the quantum well (6 nm). Then we calculate the lowest two eigenstates for each case and see the barrier-width dependency of their eigenenergies.
We also explain where the relevant output files are in.

**Material Parameters**

The material parameters are given in `database_nn*.in` but we can also redefine them manually in input files. In this tutorial, we redefine parameters so that they are the same as the section 3.9 of Paul Harrison’s book “Quantum Wells, Wires and Dots”.

<table>
<thead>
<tr>
<th>Material Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>conduction band offset ( \text{Al}<em>{0.2} \text{Ga}</em>{0.52} \text{As / GaAs} )</td>
<td>0.167 eV</td>
</tr>
<tr>
<td>conduction band effective mass ( \text{Al}<em>{0.2} \text{Ga}</em>{0.52} \text{As} )</td>
<td>0.084 ( m_0 )</td>
</tr>
<tr>
<td>conduction band effective mass ( \text{GaAs} )</td>
<td>0.067 ( m_0 )</td>
</tr>
</tbody>
</table>

**Results**

1. **barrier width = 4 nm**

- The following figure shows the conduction band edge and wave functions that are confined inside the wells with barrier width = 4 nm.

(Note that the energies were shifted so that the conduction band edge of GaAs equals 0 eV.)

![Double Quantum Well](image1.png)

- The wave functions form a symmetric and an anti-symmetric pair. The symmetric one is lower in energy than the anti-symmetric one. The plot is in excellent agreement with Fig. 3.17 (page 92) of Paul Harrison’s book “Quantum Wells, Wires and Dots”.

- For comparison, the following figure shows for the same structure as above, the square of the wave function rather than \( \psi \) only.

![Double Quantum Well](image2.png)
Output

a. The conduction band edge of the Gamma conduction band can be found here:

- `bias_00000/bandedge_Gamma.dat` (nextnano++)
- `band_structure/cb_Gamma.dat` (nextnano³)

b. This file contains the eigenenergies of the two lowest eigenstates. The units are [eV].

- `bias_00000/Quantum/wf_energy_spectrum_quantum_region_Gamma_0000.dat` (nextnano++)
- `Schroedinger_1band/ev_cb1_sg1_deg1.dat` (nextnano³)

These are the comparison of eigenvalues:

<table>
<thead>
<tr>
<th></th>
<th>nextnano++</th>
<th>nextnano³</th>
<th>Harrison’s book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ground state energy [eV]</td>
<td>0.04920</td>
<td>0.04920</td>
<td>0.04912</td>
</tr>
<tr>
<td>first excited state energy [eV]</td>
<td>0.05779</td>
<td>0.05779</td>
<td>0.05770</td>
</tr>
</tbody>
</table>

c. This file contains the eigenenergies and the wave functions (\(\psi\)):

- `bias_00000/Quantum/wf_amplitudes_shift_quantum_region_Gamma_0000.dat` (nextnano++)
- `Schroedinger_1band/cb1_sg1_deg1_psi_shift.dat` (nextnano³)

This file contains the eigenenergies and the squared wave functions (\(\psi^2\)):

- `bias_00000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat` (nextnano++)
- `Schroedinger_1band/cb1_sg1_deg1_psi_squared_shift.dat` (nextnano³)

The subscript _shift indicates that \(\psi^2\) and \(\psi\) are shifted by the corresponding energy levels.

a. and c. can be used to plot the data as shown in the figures above.

2. barrier width = 1 ~ 14 nm

* Here, we varied the thickness of the Al\(_{0.2}\)Ga\(_{0.8}\) As barrier layer from 1 nm to 14 nm fixing the width of the quantum well (6 nm). We calculated the lowest two eigenstates and show their eigenvalues for each barrier width in the following figure.

- If the separation between the two quantum wells is large, the wells behave as two independent single quantum wells having the identical ground state energies. The interaction between the energy levels localized within each well increases once the distance between the two wells decreases below 10 nm. One state is forced to higher energies and the other to lower energies. (Here, the electron spins align in an “anti-parallel” arrangement in order to satisfy the Pauli exclusion principle.)
• This is analogous to the hydrogen molecule where the formation of a pair of bonding and anti-bonding orbitals occurs once the two hydrogen atoms A and B are brought together.

$$\psi_{\text{bonding}} = \frac{1}{\sqrt{2}} \psi_A + \psi_B \quad \text{(lower energy)}$$

$$\psi_{\text{antibonding}} = \frac{1}{\sqrt{2}} \psi_A - \psi_B \quad \text{(higher energy)}$$

• Again, the plot is in excellent agreement with Fig. 3.16 (page 92) of Paul Harrison’s book “Quantum Wells, Wires and Dots”.

Output

The energy values were taken from the same file as before:

* bias_00000/Quantum/wf_energy_spectrum_quantum_region_Gamma_0000.dat (nextnano++)
  Schredinger_1band/ev_cb1_sg1_deg1.dat (nextnano³)

For example, the values for the 1 nm barrier read:

<table>
<thead>
<tr>
<th></th>
<th>nextnano++</th>
<th>nextnano³</th>
<th>Harrison’s book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ground state energy [eV]</td>
<td>0.03476</td>
<td>0.03476</td>
<td>0.03470</td>
</tr>
<tr>
<td>first excited state energy [eV]</td>
<td>0.07298</td>
<td>0.07298</td>
<td>0.07290</td>
</tr>
</tbody>
</table>

The values for the 14 nm barrier read:

<table>
<thead>
<tr>
<th></th>
<th>nextnano++</th>
<th>nextnano³</th>
<th>Harrison’s book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ground state energy [eV]</td>
<td>0.05332</td>
<td>0.05332</td>
<td>0.05323</td>
</tr>
<tr>
<td>first excited state energy [eV]</td>
<td>0.05338</td>
<td>0.05338</td>
<td>0.05329</td>
</tr>
</tbody>
</table>

Tip: Sweeping

A sweep over the thickness of the Al$_{0.2}$Ga$_{0.8}$As barrier layer, i.e. the variable %QW\_SEPARATION, can easily be done by using nextnanomat’s Template feature. The following screenshot shows how this can be done. Go to “Template”, open input file, select “Range of values”, select “QW\_SEPARATION”, click on “Create input files”, go to “Run and start your simulations.

Another tutorial on coupled quantum wells can be found here.

13.1.5 Simple quantum cascade structure

In this tutorial we simulate a simple quantum cascade structure that has been presented in an article by Capasso et al. (Figures 12 (b) and 16 (b) of [CapassoIEEE1986]).

Input files used in this tutorial are followings:

• IDQCL_simple_nnp.in / *n3.in

These are available in the sample files folder.

We can generate the following picture that is based on Fig. 3 of [BirnerPhotonikInt2008] and [BirnerPhotonik2008].

It shows the conduction band edge profile of an Al$_{0.48}$In$_{0.52}$As/In$_{0.53}$Ga$_{0.47}$As superlattice at an electric field of -89 kV/cm. The single-band effective-mass Schrödinger equation is solved for this band profile. The wave functions ($\psi^2$) of this quantum cascade structure are shown.

The basic idea of such a structure is to depopulate the lowest eigenstate of each quantum well efficiently by bringing it into resonance with the third eigenstate of the next quantum well (resonant tunneling).

The transition second eigenstate $\rightarrow$ lowest eigenstate should be a nonradiative intersubband transition.
13.1. 1D Tutorials
On the other hand, the transition third eigenstate → second eigenstate should be a radiative intersubband transition, i.e. a photon is emitted.

Another important condition for a quantum cascade laser is population inversion, i.e. the occupation of the third eigenstate must be much higher than the occupation of the second eigenstate and lowest eigenstate.

- The input file $1DQCL_{\text{simple.nn}}/\text{nn3.in}$ should be rather intuitive and self-explanatory. Documentation for each keyword and each specifier can be found here: Keywords
- In the nextnano++ sample file, the electric field is applied by specifying the keyword contacts as follows:

```plaintext
contacts{
  charge_neutral{
    name = "leftgate"
    bias = 0.0
  }
  charge_neutral{
    name = "rightgate"
    bias = 1.36081          # corresponds to electric field of $F = -89$ kV/cm
  }
}
```

In the keyword structure, “leftgate” is defined at $x = [-1, 0]$ and “rightgate” is at $x = [152.9, 153.9]$. Thus the electric field applied by this specification is $-1.36081 \text{[V]} / 152.9 \text{[nm]} = -89 \text{[kV/cm]}$

- Alternatively, we can apply a constant electric field by providing a value for the field.

```plaintext
poisson{
  electric_field{ strength = -89e5 } # [V/m]
  output_potential{}
  output_electric_field{}
}
```

### Output

The output files are ASCII files.

### Bandedges

The conduction and valence band edges can be found in the following file:

- $\text{bias}_0000/\text{Quantum/bandedges.dat}$ (nextnano++)
- $\text{band_profile/ch_Gamma.dat}$ (nextnano3, conduction band edge only)

If one plots the conduction band profile, one gets the following figure.

There are six $\text{Al}_{0.48}\text{In}_{0.52}\text{As}$ barriers and five $\text{In}_{0.53}\text{Ga}_{0.47}\text{As}$ wells. The conduction band offset is 0.51 eV.
Eigenvalues

The 40 eigenvalues that were calculated can be found in these files. The units are [eV].

- `bias_0000/Quantum/wf_energy_spectrum_quantum_region_Gamma_0000.dat` (nextnano++)
- `wavefunctions/ev_cb1_sg1_deg1.dat` (nextnano³)

The eigenvalues are also contained in these files, i.e. the eigenvalues for each grid point

- `bias_0000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat` (nextnano++)
- `wavefunctions/cb1_sg1_deg1_psi_squared_shift.dat` (nextnano³)

<table>
<thead>
<tr>
<th>1st column</th>
<th>2nd column</th>
<th>3rd column</th>
<th>...</th>
<th>41st column</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid points in units of [nm]</td>
<td>1st eigenvalue in units of [eV]</td>
<td>2nd eigenvalue in units of [eV]</td>
<td>...</td>
<td>40th eigenvalue in units of [eV]</td>
</tr>
</tbody>
</table>

If one plots these columns (together with the conduction band edge) one obtains the following picture:

**Note:** The figure shows only the following energy levels: 1,2,3,4,5,9,10,12,16,18,20,26,27,30,37

Wavefunctions

The square of the wave functions ($\psi^2$) of the 40 eigenstates can be found in these files.

- `bias_0000/Quantum/wf_probabilities_shift_quantum_region_Gamma_0000.dat` (nextnano++)
- `wavefunctions/cb1_qc1_sg1_deg1_psi_squared_shift.dat` (nextnano³)

<table>
<thead>
<tr>
<th>1st column</th>
<th>...</th>
<th>42nd column</th>
<th>43rd column</th>
<th>...</th>
<th>81st column</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid points in units of [nm]</td>
<td>...</td>
<td>$\psi^2$ of 1st eigenstate</td>
<td>$\psi^2$ of 2nd eigenstate</td>
<td>...</td>
<td>$\psi^2$ of 40st eigenstate</td>
</tr>
</tbody>
</table>
In order to be able to plot the wave functions nicely into the conduction band edge profile, we shift the square of the wave function by its corresponding energy.

If one plots these columns (together with the conduction band edge) one obtains the following picture:

Note: The figure shows only the following wave functions: 1,2,3,4,5,9,10,12,16,18,20,26,27,30,37

Now the lowest eigenstate of each quantum well is in resonance with the third eigenstate of the next quantum well. This leads to the depopulation of the lowest eigenstate of each quantum well.

Photon should be emitted with the radiative intersubband transition $3 \rightarrow 2$ whereas $2 \rightarrow 1$ should be nonradiative intersubband transition.

**Effective masses**

The effective masses that were used for each grid point can be found in these files.

- *Structure/charge_carrier_masses.dat* (nextnano++)
- *material_parameters/conduction_band_masses.dat* (nextnano³)

Note: We need to add the following option into the sample file for nextnano++.

```
output{
    material_parameters{
        charge_carrier_masses{ boxes yes }
    }
}
```

- 1st column: grid points in units of [nm]
other columns:

- nextnano++: effective mass tensor components of Gamma and HH valley in units of [m0]. When we use other valleys for the simulation, then these columns shows the effective mass tensor components in that valleys.

- nextnano³: effective mass tensor components of Gamma, L and X valleys in units of [m0].

These masses have been calculated from the binaries InAs, GaAs and AlAs for the relevant ternaries, including bowing parameters.

**Intersubband matrix elements**

Experienced users might be interested in having a look at the intersubband matrix elements.

We can find the intersubband (or intraband) matrix elements $p_z$, the oscillator strengths and the transition energies by adding the followings into `quantum()` in `IDQCL_simple_nnp.in`:

```plaintext
intraband_matrix_elements{
    Gamma{
        output_matrix_elements = yes
        output_transition_energies = yes
        output_oscillator_strengths = yes
    }
}
```

The relevant output files are

- `bias_0000/Quantum/intraband_matrix_elements_quantum_region_Gamma_100.txt` (nextnano++)
- `bias_0000/Quantum/transition_energies_quantum_region_Gamma_Gamma.txt` (nextnano++)

In the output file of the nextnano³ sample file, we can already have them here:

- `wavefunctions/intraband_pz_cb1_sg1_deg1.txt` (nextnano³)

More information and documentation on these matrix elements is available here:
• Intersubband transitions in InGaAs/AlInAs multiple quantum well systems (nextnano++)
• Intersubband matrix elements (nextnano³)

13.1.6 Cascade solar cell (Tandem solar cell)

In this tutorial, we solve the Poisson equation in an AlGaAs/InGaAs monolithic cascade solar cell (tandem solar cell).

The layout is based on US patent 4,179,70: Cascade solar cell.

See also the following publication for more details

Computer Modeling of a Two-Junction, Monolithic Cascade Solar Cell
M.F. Lamorte, D.H. Abbott

Input files used in this tutorial are followings:
• IDCascadeSolarCell_nnp.in / *nn3.in

Outputs
Band profile

The following figure shows the conduction band edge and the valence band edges (heavy hole, light hole and split-off hole) of this solar cell at zero bias. The built-in potential has been calculated to be 1.83 V.

At the left side (region 1), a graded p-type AlGaAs layer has been used to generate an electric field of 3 kV / cm (= 30 meV / 100 nm). We assumed that all materials are strained with respect to the GaAs substrate, thus the degeneracy of heavy and light hole valence band edges is lifted, especially inside the InGaAs regions.
Band gap

The band gap as a function of distance is shown in the following figure. This data can be found in these files. For `nextnano++`, we need to add `classical { output_bandgap[] }` in the sample file.

- `bias_00000/bandgap.dat` (`nextnano++`)
- `band_structure/BandGap1D.dat` (`nextnano³`)

Electron and hole densities

Here, the electron and hole densities are plotted. This data can be found in these files.

- `bias_00000/density_electron.dat`, `bias_00000/density_hole.dat` (`nextnano++`)
- `densities/density_el.dat`, `densities/density_hl.dat` (`nextnano³`)
The area around the tunnel junction which is in the middle of the device at ~2100 nm is shown in this plot:

The electron and hole densities in the vicinity of the tunnel junction are shown in this graph. Note that the density has been calculated classically (without solving the Schrödinger equation, i.e. without quantum mechanics).
What we can do on a solar cell using nextnano

We have the demonstration of the simulation for GaAs solar cell using nextnano here: *GaAs Solar Cell.*

As we can see in this demonstration, we can calculate the following characteristics by solving the Poisson equation and current equation self-consistently.

- **Current-Voltage characteristics**
  - The dark current can also be calculated.

- **Solar efficiency**
  - We can also see the effect of optical concentration quantitatively.

The data we need to prepare independently for this calculation is:

1. spectral irradiance (solar spectrum)
2. reflectivity at the front surface
3. absorption coefficient

Both nextnano++ and nextnano³ can calculate the generation rate $G(x)$ now. We can also import the data of $G(x)$ directly instead of 2 and 3 above.

The links for all the used data is also specified in this tutorial: *GaAs Solar Cell.*

Please help us to improve our tutorial. Send comments to support [at] nextnano.com.
13.1.7 Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider’s code

In this tutorial we calculate the self-consistent solution of Schrödinger-Poisson equations using nextnano++ and another code provided by Greg Snider (University of Notre Dame). We compare the two results and see the agreement of them.

We also discuss about the basic concept of the Schrödinger-Poisson solution.

The related input files are followings:

- `Greg_Snider_MANUAL_1D_nn*.in`
- `Greg_Snider_MANUAL_ID_analytic_nn*.in`
- `Greg_Snider_MANUAL_2D_nn*.in`
- `Greg_Snider_MANUAL_2D_analytic_nn*.in`

These are available in the sample file folder. The files which have `analytic` in their names use analytic doping function.

We appreciate that Greg Snider provided his code, the manual and the input files free of charge, so that we could use it here as a test case. His 1D Poisson/Schrödinger code can be obtained from this link. This tutorial is based on his manual (1D Poisson Manual.pdf, MANUALEX).

Structure

We simulate a structure consisting of the following materials and doping profile. The additional doping profile based on LSS Theory is explained in the next section.

<table>
<thead>
<tr>
<th>Surface</th>
<th>GaAs</th>
<th>n-type doped (10^{18} \text{cm}^{-3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>z = 0 ~ 15 nm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z = 15 ~ 35 nm</td>
<td>Al_{0.3}Ga_{0.7}As</td>
<td>n-type doped (10^{18} \text{cm}^{-3})</td>
</tr>
<tr>
<td>z = 35 ~ 39.5 nm</td>
<td>Al_{0.3}Ga_{0.7}As</td>
<td></td>
</tr>
<tr>
<td>z = 39.5 ~ 54.5 nm</td>
<td>GaAs</td>
<td>quantum well</td>
</tr>
<tr>
<td>z = 54.5 ~ 105 nm</td>
<td>Al_{0.3}Ga_{0.7}As</td>
<td></td>
</tr>
<tr>
<td>z = 105 ~ 355 nm</td>
<td>Al_{0.3}Ga_{0.7}As</td>
<td>p-type doped (10^{17} \text{cm}^{-3})</td>
</tr>
<tr>
<td>Substrate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The grid resolution is 1 nm with the exception of the 250 nm layer which has a resolution of 5 nm and the material interfaces of the quantum well which has a resolution of 0.5 nm.
- The dopants are assumed to be fully ionized.
- The temperature is 300 K.
- The Schrödinger equation will be solved between 5 nm and 195 nm.

Doping

We consider two further impurity profile resulting from ion implantation using LSS Theory. The donor and acceptor profiles are written out of the file `density_acceptor/acceptor.dat` and look as follows:

<table>
<thead>
<tr>
<th>Implant</th>
<th>dose \text{[cm}^{-2}]</th>
<th>Projected range ( R_p ) [nm]</th>
<th>Projected straggle Delta ( \sigma_p ) [nm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>donor</td>
<td>2 \times 10^{12}</td>
<td>86</td>
<td>44</td>
</tr>
<tr>
<td>acceptor</td>
<td>1 \times 10^{11}</td>
<td>75</td>
<td>20</td>
</tr>
</tbody>
</table>

For further details on the LSS theory (ion implantation) and on the doping profiles, please check the relevant keyword `doping()` for nextnano++ (or `Doping-function` for nextnano³).
Conduction and valence band edges

The following figure shows the conduction and valence band edges as well as the Fermi level (which is constant and has the value of 0 eV) for the structure specified above. These bands are the solutions of the self-consistent Schrödinger-Poisson equation.

Both codes, \texttt{nextnano++} and Greg Snider’s “1D Poisson” lead to the same results.

Electron eigenstates and eigenfunctions

Inside the GaAs quantum well there are three confined electron states. The ground state is below the Fermi level and thus occupied. The following figure shows a zoom of the GaAs Quantum well.

The wave functions as calculated with \texttt{nextnano++} are nearly identical to Greg Snider’s “1D Poisson” code, as well as the energies. However, there are tiny differences which is not too surprising as the conduction band profile is not completely identical.

<table>
<thead>
<tr>
<th>Electron states</th>
<th>\texttt{nextnano++}</th>
<th>\texttt{nextnano}</th>
<th>Greg Snider’s “1D Poisson” code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$ [meV]</td>
<td>-3.1</td>
<td>-3.0</td>
<td>-1.3</td>
</tr>
<tr>
<td>$E_2$ [meV]</td>
<td>43.4</td>
<td>43.5</td>
<td>44.0</td>
</tr>
<tr>
<td>$E_3$ [meV]</td>
<td>117.4</td>
<td>117.5</td>
<td>117.8</td>
</tr>
</tbody>
</table>

Electron and hole densities

The electron and hole densities are depicted in this figure, there is also nice agreement between the two codes.

- The integrated electron density in the GaAs quantum well region is $0.667 \times 10^{12}$ cm$^{-2}$. (Greg Snider’s result: $0.636 \times 10^{12}$ cm$^{-2}$)
- The integrated hole density in the right most Al0.3Ga0.7As region is $1.033 \times 10^{12}$ cm$^{-2}$. (Greg Snider’s result: $1.085 \times 10^{12}$ cm$^{-2}$)

The relevant output files are:

- \texttt{integrated\_density\_electron.dat}
**Conduction and valence band diagram**

Schottky barrier 0.6 V

- GaAs
- Al0.3Ga0.7As

Energy [eV]

Distance [nm]

---

**Conduction band, electron eigenstates and electron wavefunctions (Ψ^2)**

15 nm GaAs quantum well

- Al0.3Ga0.7As

Energy [eV]

Distance [nm]

- 117 meV
- 43 meV
- -3 meV
This tutorial shows very nicely that both codes, nextnano++ and Greg Snider’s “1D Poisson” lead to the same results. Greg Snider’s 1D Poisson/Schrödinger code can be obtained from here: http://www.nd.edu/~gsnider/

2D simulations

• Greg_Snider_MANUAL_2D_*nn*.in

We can also calculate the 2D schrödinger-Poisson equation for the same structure where the y direction has been assumed to be of length 100 nm with periodic boundary conditions.

Self-consistent Schrödinger-Poisson solution

Here we briefly discuss about the basic concept of the method used to get the above results.

In this section, we refer to


Self-consistent calculation of Schrödinger-Poisson equations is one way to treat the manybody effects associated with Coulomb repulsion.

For example, suppose we calculate Schrödinger equation to obtain the energy eigenvalues and eigenstates for a quantum well only one time. If we add a further test electron into the system, the potential that the test electron feels is the band-edge potential plus Coulomb potential which is caused by the original electrons in the system. In most cases, the carrier density in a single quantum well is so high that it is important to take this additional potential into consideration. (6.67*10^{12} \text{ cm}^{-2} for the GaAs quantum well in this tutorial.)

In order to obtain the solution which involves this effect, the potential used in Schrödinger equation for the electrons and the charge distribution which is based on the energy eigenstates from that Schrödinger equation must satisfy Poisson equation. This solution is described as self-consistent, rather like Hartree’s approach to solving many electron atoms.
The process for obtaining self-consistent solution of Schrödinger-Poisson equations is as follows:

1. Solve Schrödinger equation using band-edge potential \( V_{be}(r) \) and obtain the eigenstates of an electron \( \Psi^{el}_{\alpha,E}(r) \) and hole \( \Psi^{hole}_{\beta,E}(r) \). Here \( \alpha \) is the conduction band number, \( \beta \) is the valence band number and \( E \) represents the eigenvalue.

2. Calculate the density distribution of the particles \( n(r) \) using local density of state \( \rho^{el}(r,E) := \sum |\Psi^{el}_{\alpha,E}(r)|^2 \) and \( \rho^{hole}(r,E) := \sum |\Psi^{hole}_{\beta,E}(r)|^2 \) and fermi distribution \( f(E) := \frac{1}{e^{\left(E-E_F\right)/k_B T}+1} \).

3. Solve Poisson equation and obtain the potential distribution \( \phi(r) \) caused by the distributed electrons, holes, and ions.

\[
\nabla \cdot (\epsilon_s(r) \nabla \phi(r)) = -\frac{e}{\epsilon} [n^{hole}(r) - n^{el}(r) + N_D(r) - N_A(r)]
\]

where \( \epsilon_s \) is the dielectric constant, \( N_D(r) \) is the donor concentration and \( N_A(r) \) represents the acceptor concentration.

4. Using the new potential \( V_{new}(r) := -q\phi(r) + V_{be}(r) \) which consists of the result of 3. and band-edge potential, solve Schrödinger equation.

5. Check whether the energy eigenvalues converged or not. Then
   - Yes \( \rightarrow \) End
   - No \( \rightarrow \) Go to 2.

The process is iterated until the energy eigenvalues converge. At last, the potential used in Hamiltonian and one calculated from charge distribution which is from Schrödinger equation will be identical.

### 13.1.8 C-V curve calculation for general structures (Post-processing by python)

nextnano++ can calculate many fundamental quantities like potentials, carrier densities, wavefunctions and so on. By processing the results of nextnano++ using the calculation tools such as python, we can calculate further advanced characteristics required for some specific devices.

C-V curve is one of the example of such characteristics. This curve is used for the analysis of the devices that could have a depletion region such as metal-insulator-semiconductor, p-n junction, MOSFET and so on.

Specifically, the C-V characteristic is obtained by calculating the capacitance as

\[
C = \frac{dQ}{dV}
\]

When the bias sweep and spatial integration are specified in the input file, the electron and hole densities integrated over the region are output in integrated_density_electron.dat and integrated_density_hole.dat with respect to each bias. The C-V curve can be calculated by taking a derivative of the Q-V curve that is obtained from these data file.

(For the details of bias sweep and spatial integration, please refer to the input file of the tutorial in Example.)

In this tutorial we provide python scripts that calculate and plot the C-V curve. They are applied to our MIS tutorial here, but they can also be applied for the general structures that output integrated_density_electron.dat and integrated_density_hole.dat. The second script uses our postprocessing tool nextnanopy.

- **Post-processing without nextnanopy**
- **Post-processing with nextnanopy (under construction)**
Solve Schrödinger equation for band-edge potential

Calculate the density distribution of electrons / holes

Solve Poisson equation

Add resulting potential to band-edge

Solve Schrödinger equation for new potential

Has energy converged?

No

Yes

End

Schematics of the self-consistent iteration
Post-processing without nextnanopy

- CapacitanceBySplines_2021_Nov.py

This script first calculates the Q-V curve interpolating the total integrated charges obtained from the data files and calculates the C-V curve from that Q-V curve.

Example command:

```
python C:\Users\naoki.mitsui\Documents\CapacitanceBySplines_2021_Nov\ →CapacitanceBySplines_2021_Nov.py -o C:\Users\naoki.mitsui\Documents\nextnano\ →Output\ →p
```

The commandline options are followings:

- `-o`: Path of the output folder where `integrated_density_hole.dat` and `integrated_density_electron.dat` are stored follows. (required)
- `-p`: if present in the command line, the total integrated charge and interpolated C-V curves will be plotted using Matplotlib (optional)
- `-b1`: Substring of the contact that will be used as reference follows. When not specified the first common contact of both integrated_density files will be used. (optional)
- `-b2`: Substring of a second contact that will be used as reference follows. The final C-V will be calculated as function of the voltage given by bias1 - bias 2. If bias1 was not specified, bias2 will be ignored. (optional)

Example

Here we have a MIS tutorial: “Capacitance-Voltage curve of a “metal”-insulator-semiconductor (MIS) structure”.

After running the npn input file of this tutorial `MIS_CV_1nmSiO2_1D_nnp.in`, we can find `integrated_density_electron/hole.dat` in the output folder.

By executing `CapacitanceBySplines_2021_Nov.py` in the following command,

```
python C:\Users\naoki.mitsui\Documents\CapacitanceBySplines_2021_Nov\ →CapacitanceBySplines_2021_Nov.py -o C:\Users\naoki.mitsui\Documents\nextnano\ →Output\MIS_CV_1nmSiO2_1D_nnp\ →p
```

we get the Q-V curve and C-V curve as follows.

Post-processing with nextnanopy

*This part will be added soon.*

If you have any question, please contact to support [at] nextnano.com.
Figure 13.1.8.1: Q-V characteristics obtained by postprocessing the result of *MIS_CV_InmSiO2_1D_nnp.in* by *CapacitanceBySplines_2021_Nov.py*. Linear and cubic interpolation are done to the output data.

Figure 13.1.8.2: C-V characteristics obtained by postprocessing the result of *MIS_CV_InmSiO2_1D_nnp.in* by *CapacitanceBySplines_2021_Nov.py*. 

13.1. 1D Tutorials
13.1.9 1D Transmission (CBR)

The transmission function tutorial based on the CBR method for several 1D potential energy profiles are described here.

13.1.10 1D InGaAs Multi-quantum well laser diode

Section author: Takuma Sato

In this tutorial, we simulate optical emission of a 1D InGaAs multi-quantum well laser diode grown on InP substrate. The blue region, seen in Figure 13.1.10.1, is the separate confinement heterostructure (SCH), which forms an optical waveguide in the transverse direction to confine the emitted light (red arrow). The multi-quantum wells and SCH are clad by InP on both sides. A voltage bias is applied to the gray edges.

Figure 13.1.10.1: Structure overview

Input files

- LaserDiode_InGaAs_1D_cl_nnp.in
- LaserDiode_InGaAs_1D_qm_nnp.in

References


Table of contents
• **Current equation**
• **Recombination of carriers and emission spectrum**
• **Input file**
• **Results**
  – Band structure
  – Energy eigenstates and eigenvalues
  – Charge densities
  – Emission spectra and absorption coefficient
  – Current and internal quantum efficiency

### Current equation

The properties of optoelectronic devices are governed by Poisson equation, Schroedinger equation, drift-diffusion and continuity equations. We denote by $n$ and $p$ the carrier number density per unit volume. The continuity equations in the presence of creation (generation, $G$) or annihilation (recombination, $R$) of electron-hole pairs read

\[
- \frac{\partial n}{\partial t} + \nabla \cdot (-e j_n(x)) = -e (G(x) - R(x)),
\]

\[
\frac{\partial p}{\partial t} + \nabla \cdot e j_p(x) = e (G(x) - R(x)),
\]

where the current is proportional to the gradient of quasi Fermi levels $E_{F,n/p}(x)$

\[
\begin{align*}
  j_n(x) &= -\mu_n(x)n(x)\nabla E_{F,n}(x), \\
  j_p(x) &= \mu_p(x)p(x)\nabla E_{F,p}(x).
\end{align*}
\]

Here the charge current has the unit of (area)$^{-1}$ (time)$^{-1}$. $\mu_n/p$ are the mobilities of each carrier. In nextnano++, $\mu_n/p$ are determined using the mobility model specified in the input file under currents`. Hereafter we consider stationary solutions and set $\dot{n} = \dot{p} = 0$. The governing equations then reduce to

\[
\begin{align*}
  \nabla \cdot \mu_n(x)n(x)\nabla E_{F,n}(x) &= -(G(x) - R(x)), \\
  \nabla \cdot \mu_p(x)p(x)\nabla E_{F,p}(x) &= G(x) - R(x),
\end{align*}
\]

which we call **current equation** (generation $G = 0$ in the present case). nextnano++ solves this equation and Poisson equation self-consistently when one specifies it in the input file as:

```plaintext
run{
  current_poisson()
}
```
Recombination of carriers and emission spectrum

The generation/recombination rate, $R(x)$, originates from several physical processes. In nextnano++, the following mechanisms are implemented (cf. recombination):

- **Schockley-Read-Hall recombination** $R_{SRH}$ – carrier trapping by impurities.
- **Auger recombination** $R_{Auger}$ – a collision between two carriers results in the excitation of one and the recombination of the other with a carrier of opposite charge.
- **radiative recombination** $R_{rad}$ – emission/absorption of a photon.

Each mechanism can be turned on and off in the input file.

**Radiative recombination** describes the recombination of electron-hole pairs at a position $x$ by emitting a photon and is given by

$$ R_{rad}^{spont}(x, E) = C(x) \int dE_h \int dE_e n(x, E_e)p(x, E_h)\delta(E_e - E_h - E), \quad (13.1.10.4) $$

where $C(x)$ [cm$^{-3}$s$^{-1}$] is the (material-dependent) radiative recombination parameter which is proportional to the one specified in the database (Radiative recombination) and $n(x, E), p(x, E)$ [cm$^{-3}$eV$^{-1}$] are the charge densities as a function of energy and position.

In nextnano++, this radiative recombination whose rate is calculated as above is regarded as spontaneous emission. On the other hand, the net amount of the stimulated emission rate is given by:

$$ R_{rad,net}^{stim}(x, E) = \left(1 - e^{-\frac{E - (E_F - E_F^0)}{k_B T}}\right) R_{rad}^{spont}(x, E) \quad (13.1.10.5) $$

This is consistent with eq.(9.2.39) in [ChuangOpto1995]. We note that here it is assumed that photon modes occupied by one photon each, i.e. takes into account neither energy-dependent photon density of states nor Bose-Einstein distribution.

Since the radiative recombination process involves no phonons, this transition is vertical and therefore this contribution is only relevant for semiconductors with a direct band gap such as GaAs.

**Absorption coefficient** is calculated from $R_{rad,net}^{stim}(E)$ as

$$ \alpha(E) = \frac{\pi^{-3/2}h^2c^2 R_{rad,net}^{stim}(E)}{n_r^2 E^2 V} \quad (13.1.10.6) $$

where $n_r$ is the refractive index and $V$ is the total volume of the device. The unit is [cm$^{-1}$]. In case of 1D simulation, calculated $R_{rad,net}^{stim}(E)$ has the unit [cm$^{-2}$s$^{-1}$eV$^{-1}$] and is divided by the total length instead of the volume. This formula is consistent with eq (9.2.25) in [ChuangOpto1995].

**Input file**

In the beginning of the input file, we define several variables for the structure and parameters for simulation. The variable definitions are shown in Figure 13.1.10.2.

Charge density as a function of position $n(x)$ is always calculated by default. On the other hand, charge density as a function of energy $n(E), p(E)$, charge density as a function of both position and energy $n(x, E), p(x, E)$ and emission spectrum are calculated when the followings are specified (see classical for details):

```plaintext
classical{
  energy_distribution{
    carrier densities as a function of energy
    min = -1.5
  }
  # Calculation of ...
  # Integrate from ...
```

(continues on next page)
Figure 13.1.10.2: The definition of variables. The gray regions are contacts of 1nm thickness. $\texttt{NUMBER_OF_WELLS}$ determines the repetition of quantum wells. The program automatically sweeps the bias voltage starting from $\texttt{BIAS_START}$ until $\texttt{BIAS_END}$, at intervals of $\texttt{BIAS_STEPS}$.

(continued from previous page)

max \[ 0.5 \] energy_resolution \[ 0.005 \] only_quantum_regions \[ yes \] 

\begin{verbatim}
energy_resolved_density{
    min \[ -1.5 \] max \[ 0.5 \] energy_resolution \[ 0.005 \] only_quantum_regions \[ yes \]
}
\end{verbatim}

emission_spectrum{
    output_spectra{
        emission \[ yes \] gain \[ yes \] absorption \[ yes \] stimulated_emission \[ yes \]
        spectra_over_energy \[ yes \] spectra_over_wavelength \[ yes \] spectra_over_frequency \[ yes \] spectra_over_wavenumber \[ yes \]
        photon_spectra \[ yes \] power_spectra \[ yes \]
    }
    output_photon_density \[ yes \] output_power_density \[ yes \]
}

(continues on previous page)
The mobility model and recombination models for the current equation are specified in `currents()` as

```plaintext
currents{
  mobility_model = minimos
  recombination_model{
    SRH = yes  # 'yes' or 'no'
    Auger = yes  # 'yes' or 'no'
    radiative = yes  # 'yes' or 'no'
  }
}
```

The `run` flag specifies which equations to solve. This is the main difference between LaserDiode_*_qm_nnp.in and LaserDiode_*_cl_nnp.in.

```plaintext
# qm
run{
  strain{}  # solves the strain equation
  current_poisson{}  # solves the coupled current
  and Poisson equations self-consistently
  output_log = yes
  iterations = 1000
  alpha_fermi = 0.5
  quantum_current_poisson{}  # solves the Schroedinger
  and Poisson (and current) equations self-consistently
  iterations = 1000
  alpha_fermi = 0.9
  residual = 1e6
  residual_fermi = 1e-8
  output_log = yes
}
# cl
run{
  strain{}  # solves the strain equation
  current_poisson{}  # solves the coupled current
  and Poisson equations self-consistently
  output_log = yes
  iterations = 1000
  alpha_fermi = 0.7
  residual_fermi = 1e-10
}
```

In this case nextnano++ first solves the strain equation from the crystal orientation to decide the polarization charges (piezoelectric effect) and shifted bandedges. Then the program solves the coupled current-Poisson-Schroedinger equations in a self-consistent way (input file: LaserDiode_InGaAs_1D_qm_nnp.in). For the classical calculation (LaserDiode_InGaAs_1D_cl_nnp.in), `quantum_current_poisson()` is commented out to restrict the calculation to the current-Poisson equations only.
Results

Band structure

The band structure and emission power spectrum of the system are stored in `bandedges.dat`. Figure 13.1.10.3 shows the case for the bias 0.2 V. Here the quasi Fermi level of electrons is lower than the quantum wells.

![Bandstructure](image)

Figure 13.1.10.3: Bandstructure of the laser diode system for a low bias of 0.2 V.

For the bias 0.8 V (Figure 13.1.10.4), in contrast, it lies above the red line, allowing electrons to flow into the quantum wells. An electron trapped in the quantum wells is likely to recombine with a hole in the valence band, emitting a photon. In the input file `\Optical\emission_photon_density.dat`, one can see that the photons are emitted from this active region (not shown). Figure 13.1.10.12 shows the emission spectrum in this case. When the bias is too small, e.g. Figure 13.1.10.3, the intensity is much smaller, as can be seen in Figure 13.1.10.16.

Energy eigenstates and eigenvalues

In the input file `LaserDiode_InGaAs_1D_qm_nnp.in`, the single-band Schroedinger equation is coupled to the current-Poisson equation and solved self-consistently. The wave functions of electrons and holes along with eigenvalues are written in `\Quantum\probabilities_shift_quantum_region_Gamma_0000.dat` and `\Quantum\probabilities_shift_quantum_region_HH_0000.dat` (Figure 13.1.10.5 and Figure 13.1.10.6). The light hole and split-off states are out of the quantum wells and not of our interest here.
Figure 13.1.10.4: Bandstructure for a high bias 0.8 V. Electrons flowing from the left and holes from the right recombine in the active zone (multi-quantum well structure).

Figure 13.1.10.5: Probability distribution $|\psi(x)|^2$ of the lowest localized modes of electrons and holes for the bandstructure Figure 13.1.10.3. Horizontal lines are the corresponding eigenenergies.
Figure 13.1.10: Eigenvalues of the Gamma-band up to 5th and heavy-hole-band states up to 13th in relation to bandedges. The Eigenvalues above these are higher than the barrier energy of the quantum wells. The Gamma band has single “miniband”, whereas the heavy-hole band has three. The 1st heavy-hole miniband consists of the 1st-5th eigenvalues, the 2nd heavy-hole miniband consists of the 6th-11th eigenvalues and the 3rd consists of the 12th and 13th eigenvalues.

**Charge densities**

We can find the energy-resolved charge density \( n(x, E) \) and \( p(x, E) \) in the output `electron_density_vs_energy.fld` and `hole_density_vs_energy.fld`. The following figures represent \( n(x, E) \) and \( p(x, E) \) [cm\(^{-3}\)eV\(^{-1}\)] with respect to the band edges and quasi-fermi levels at bias 0.2, 0.4, 0.6, 0.8 and 1.0 V. We can see that the carrier densities around the quantum wells increase as the bias increases.

**Note:** These graphs are generated by `nextnanopy`.

We also have the charge densities integrated over the device \( n(E) \), \( p(E) \) [cm\(^{-2}\)eV\(^{-1}\)] and energy \( n(x) \), \( p(x) \) [cm\(^{-3}\)].

\( n(E) \) and \( p(E) \) with and without quantum calculation shows different features due to the discretization of energy levels in quantum wells. This is shown in `integrated_densities_vs_energy.dat`.

Figure 13.1.10.12 illustrates the population inversion in stationary (quasi-equilibrium) state of the device under bias. Solid and dashed lines are for quantum and classical calculations, respectively. The black arrows mark the relevant energies of the structure 4 at bias of 0.8 V. The hole density is shown in Figure 13.1.10.13 with higher resolution.

The energy resolution in Figure 13.1.10.13 has been increased by a factor of 10 from Figure 13.1.10.12.

**Note:** Although these charge densities either with variable \( E \) or \( x \) are both obtained by integrating \( n(x, E) \) and \( p(x, E) \) over the corresponding variable, these are independently calculated in `nn++` simulation. Hence it is possible to turn off the calculation only for \( n(x, E) \) and \( p(x, E) \) calculating the integrated charge densities. In this
Figure 13.1.10.7: Energy-resolved electron and hole density, Gamma conduction bandedge, HH valence bandedge and quasi-Fermi levels at bias 0.2 V in quantum calculation.

Figure 13.1.10.8: Energy-resolved electron and hole density, Gamma conduction bandedge, HH valence bandedge and quasi-Fermi levels at bias 0.4 V in quantum calculation.
Figure 13.1.10.9: Energy-resolved electron and hole density, Gamma conduction bandedge, HH valence bandedge and quasi-Fermi levels at bias 0.6 V in quantum calculation.

Figure 13.1.10.10: Energy-resolved electron and hole density, Gamma conduction bandedge, HH valence bandedge and quasi-Fermi levels at bias 0.8 V in quantum calculation.
Figure 13.1.10.11: Energy-resolved electron and hole density, Gamma conduction band-edge, HH valence band-edge and quasi-Fermi levels at bias 1.0 V in quantum calculation.

Figure 13.1.10.12: Electron (red) and hole (blue) densities integrated over the device as a function of energy.
Emission spectra and absorption coefficient

The spontaneous and stimulated emission spectra are written in \Optical\emission_spectrum_photons.dat and \Optical\stim_emission_spectrum_photons.dat, respectively (Figure 13.1.10.14). The peak is at around 0.7-0.8eV, which is consistent with the charge distribution in Figure 13.1.10.12. The stimulated emission does not occur above the quasi Fermi level separation, $E_{F_n} - E_{F_p}$.

The formulas used for the calculation in the source code are specified above: \textit{Recombination of carriers and emission spectrum.}

The absorption coefficient is calculated as

$$\alpha(E) = \frac{\pi^2 \hbar^3 e^2}{n_r^2 E^2} R_{\text{rad,net}}(E)$$

where $n_r$ is the refractive index and $V$ is the total volume of the device. The unit is [cm$^{-1}$]. In case of 1D simulation, calculated $R_{\text{rad,net}}(E)$ has the unit [cm$^{-2}$s$^{-1}$eV$^{-1}$] and is divided by the total length instead of the volume. This formula is consistent with eq (9.2.25) in [ChuangOpto1995].

The absorption coefficient $\alpha(E)$ and gain (coefficient) $g(E)$ are essentially the same quantity with opposite signs,

$$\alpha(E) = -g(E)$$

These are by definition independent of the initial photon population. \textbf{Please note that the gain spectrum in nextnano++ is cut off where it is negative.} For details, see \textit{classical}. 

The spectrum changes its sign at the energy $E_{F_n} - E_{F_p}$, that is, the separation of the quasi Fermi levels. According to the output bandedges.dat, this value is -0.0001-(-0.7702)=0.7701eV. The following result has been calculated classically. We also get qualitatively consistent results from quantum mechanical simulation.
Figure 13.1.10.14: Emission spectrum of the laser diode for the bias 0.8 V.

Figure 13.1.10.15: Classically calculated absorption coefficient and gain spectrum. The sign of the spectrum switches at the energy corresponding to the quasi Fermi-level separation in the active region.
Current and internal quantum efficiency

The output file `IV_characteristics.dat` contains right- and left-contact current in unit of \( \text{Acm}^{-2} \). In the present case, the right-contact current is hole current, whereas the left-contact current is electron current. In Figure 13.1.10.15, we compare the hole current and photocurrent.

![IV Characteristics Graph]

Figure 13.1.10.16: Charge current and photocurrent as a function of bias voltage (IV characteristics).

Figure 13.1.10.16 clearly shows the consequence of the difference in bandstructures Figure 13.1.10.3 and Figure 13.1.10.4. The holes and electrons recombine in the multi-quantum well layers, emitting one photon per electron-hole pair. The efficiency of conversion from charge current into photocurrent is called the **internal quantum efficiency**

\[
\eta = \frac{I_{\text{photon}}}{I_{\text{charge}}}.
\]

This quantity is written in `internal_quantum_efficiency.dat` and shown in Figure 16.

Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.

13.1.11 UV LED: Quantitative evaluation of the effectiveness of EBL

Section author: Naoki Mitsui

We investigate how the electron blocking layer (EBL) improves the characteristics of UV LEDs using `nextnano++`. Current-Poisson equation and semi-classical calculation of optical properties \((\text{classical/})\) in `nextnano++` enables us to quantitatively analyze the effect of this structure.

We refer to the structure used to obtain Fig. 28 in the following paper:

- H. Hirayama
  
  *Quaternary InAlGaN-based high-efficiency ultraviolet light-emitting diodes*,
  

Sample input file
Figure 13.1.10.17: Conversion efficiency of the InGaAs laser diode.

- 1D_DUV_LED_HirayamaJAP2005_EBL_nnp.in

Table of contents

- Structure
- Scheme
- Results
  - Current-voltage characteristics
  - Bandedges
  - Current Density
  - Internal quantum efficiency
- What can we do further?
Structure

The simulation region consists of the following structure:

- n-Al\textsubscript{0.18}Ga\textsubscript{0.82}N layer
- 3-layer MQW based on InAlGaN
- Al\textsubscript{x}Ga\textsubscript{1-x}N EBL (Al content = 0.18, 0.24, 0.28)
- p-Al\textsubscript{0.18}Ga\textsubscript{0.82}N layer

Each layer has the following thickness and doping concentration:

<table>
<thead>
<tr>
<th>Material</th>
<th>Thickness</th>
<th>Doping</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-Al\textsubscript{0.18}Ga\textsubscript{0.82}N</td>
<td>100nm</td>
<td>8 × 10\textsuperscript{18} [cm\textsuperscript{-3}] (donor)</td>
</tr>
<tr>
<td>In\textsubscript{0.02}Al\textsubscript{0.09}Ga\textsubscript{0.89}N</td>
<td>well: 2.5nm, barrier: 15nm</td>
<td>0 [cm\textsuperscript{-3}]</td>
</tr>
<tr>
<td>In\textsubscript{0.02}Al\textsubscript{0.22}Ga\textsubscript{0.76}N 3-layer MQW</td>
<td>Barrier: 2.5nm, well: 15nm</td>
<td>0 [cm\textsuperscript{-3}] for x=0.28, 0.24, 0.18</td>
</tr>
<tr>
<td>Al\textsubscript{x}Ga\textsubscript{1-x}N EBL with x=0.28, 0.24, 0.18</td>
<td>10nm</td>
<td>2 × 10\textsuperscript{19} [cm\textsuperscript{-3}] (acceptor)</td>
</tr>
<tr>
<td>p-Al\textsubscript{0.18}Ga\textsubscript{0.82}N</td>
<td>100nm</td>
<td>2 × 10\textsuperscript{19} [cm\textsuperscript{-3}] (acceptor)</td>
</tr>
</tbody>
</table>

Al content x=0.18 in the EBL is used for the structure without EBL, while x=0.24 and 0.28 are for the structure with EBL in different barrier height.

Donor and acceptor ionization energies are defined as 0.030 eV and 0.158 eV where Si and Mg are in mind, respectively.

![Band profiles](image)

Figure 13.1.11.1: The bandedges and Fermi levels for the structure with higher EBL (x=0.28, bias=4.00V, total current density=1.67×10\textsuperscript{5} A/cm\textsuperscript{2})
Scheme

We can specify which simulation or equations would be solved on `run/` section in your input file.

In `1D_DUV_LED_HirayamaJAP2005_EBL_nnp.in` it is described as

```plaintext
run{
    strain{}
    current_poisson()
}
```

Then `nextnano++` solves the current equation and Poisson equation self-consistently after solving strain equation.

After the Current-Poisson equation is converged, optoelectronic characteristics are calculated according to the specification in the section `classical/`.

For further details, please see *General scheme of the optical device analysis*.

Results

Current-voltage characteristics

Here we show the current-voltage characteristics for the total current density $I_{\text{total}}$ measured at p-contact and photocurrent density $I_{\text{photo}}$, which is defined as (13.6.3.6). $I_{\text{photo}}$ represents the amount of electrical current consumed by the radiative recombination in the total current $I_{\text{total}}$. Please note that the scales of the y-axis in these graphs are different in 10 times.

We can observe that the smaller $I_{\text{total}}$ is, the higher the EBL barrier is. On the other hand, at the applied bias of 4.0V, the bigger $I_{\text{photo}}$ is, the higher the EBL barrier is. We can say that the larger proportion of the total current consists of the photocurrent in the higher EBL structure, which results in the larger IQE.

![Current-voltage characteristics](image)

Figure 13.1.11.2: *(Left)* The relationship between the p-contact current density and bias voltage. *(Right)* The relationship between the photocurrent $I_{\text{photo}}$ and bias voltage.
Bandedges

The following figures show the bandedge profiles and the quasi-Fermi levels for the higher EBL (top) and no EBL (bottom) structure where the total current densities are almost the same around $1.70 \times 10^5 \text{ A/cm}^2$. The applied bias is 4.00 V for the left graph and is 3.90 V for the right graph.

Figure 13.1.1.3: The bandedges and Fermi levels for the structure with EBL (x=0.28, bias=4.00V, total current density=1.67×10^5 A/cm^2)

Current Density

The following figure show the current density profiles for the higher EBL (top, x=0.28), lower EBL (middle, x=0.24), and no EBL (bottom, x=0.18) structure where the total current densities are almost the same around $1.70 \times 10^5 \text{ A/cm}^2$.

We can see that the amount of electron current and hole current becomes closer as the EBL height is increased, while the electron current is dominant without EBL. It can be also confirmed that the current overflow is suppressed by the EBL.

Charge carrier densities

The figures showed below are the electron and hole densities around the MQW region for the structure with higher EBL and without EBL (left, x=0.28 and right, x=0.18) for almost the same current density around $1.70 \times 10^5 \text{ A/cm}^2$. The introduction of EBL at 167nm-177nm reduces the electron density in the p-AlGaN region.
Figure 13.1.11.4: The bandedges and Fermi levels for the structure without EBL \((x=0.18,\ \text{bias}=3.90V,\ \text{total current density}=1.68 \times 10^5 \ \text{A/cm}^2)\)

**Power of light emission**

Here we show the relationship between optical power defined in (13.6.3.11) and current density of p-contact for each structure.

**Internal quantum efficiency**

In nextnano++, the internal quantum efficiency \(\eta_{\text{IQE}}\) is calculated as

\[
\eta_{\text{IQE}} = \eta_{\text{VQE}} \cdot \eta_{\text{IE}} = \frac{I_{\text{photo}}}{I_{\text{total}}} \tag{13.1.11.1}
\]

where \(I_{\text{photo}}\) is the photo-current consumed by the radiative recombination and \(I_{\text{total}}\) is the current injected in total.

This quantity shows the improvement by the introduction of higher EBL as follows:

nextnano++ also outputs the volume quantum efficiency \(\eta_{\text{VQE}}\), also known as radiative efficiency, which represents the proportion of the radiative recombination rate to the total recombination rate. This quantity is calculated as

\[
\eta_{\text{VQE}} = \frac{R_{\text{rad,net}} + R_{\text{fixed}}}{R_{\text{total}}} \tag{13.1.11.2}
\]

and also shows the improvement by the introduction of EBL:

The IQE can be decomposed like (13.1.11.1) into this volume QE and the injection efficiency \(\eta_{\text{IE}}\), which represents the proportion of the current consumed by the total recombination (radiative + nonradiative) to the total injected current.

Thus using the results of \(\eta_{\text{IQE}}\) and \(\eta_{\text{VQE}}\) above, we can also get this \(\eta_{\text{IE}}\):
Figure 13.1.11.5: The current density profile for the structures with higher EBL (top, 4.00 V, $1.67 \times 10^5$ A/cm$^2$), lower EBL (middle, 3.94 V, $1.74 \times 10^5$ A/cm$^2$), and no EBL (bottom, 3.90 V, $1.68 \times 10^5$ A/cm$^2$).
Figure 13.1.11.6: The electron and hole densities calculated in the structures with higher EBL (left, 4.00 V, $1.67 \times 10^5$ A/cm$^2$) and no EBL (right, 3.90 V, $1.68 \times 10^5$ A/cm$^2$).

Figure 13.1.11.7: Current vs. power of light emission
Figure 13.1.11.8: Current and internal quantum efficiency (IQE).

Figure 13.1.11.9: Current and volume quantum efficiency (radiative efficiency).
What can we do further?

The effect of EBL on the optoelectronic characteristics has been estimated quantitatively using the semiclassical calculation in nextnano++. We can also optimize the Al content of EBL or the thickness by sweeping the corresponding parameters, for example. Our open source python package nextnanopy is a strong tool for this purpose.

The graphs shown in this tutorial are also generated by a python script using nextnanopy.

Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.

13.1.12 UV LED: Quantitative evaluation of the effectiveness of superlattice structure in p-region

Section author: Naoki Mitsui

In the recent UV-LEDs based on AlGaN, the superlattice (SL) structure is introduced into the p-type layer in order to enhance the acceptor ionization, which results in the improvement of the hole conductivity. We investigate how this structure improves the characteristics of UV LEDs using nextnano++.

First, the hole concentration in a p-type AlGaN/GaN SL structure is calculated using Schrödinger-Poisson solver and the enhancement of the acceptor ionization is quantitatively examined. This part is based on the following literatures:
Second, the SL structure is introduced into the p-region of LED structure with InAlGaN MQW and Current-Poisson equation is solved. Then the IQE result is compared to the LED structure with the bulk p-region.

The structure used in this part is based on the following literature:

- **H. Hirayama**
  
  *Quaternary InAlGaN-based high-efficiency ultraviolet light-emitting diodes,*
  

**Sample input files**

- 1D_UV_LED_KozodoyAPL1999_nnp.in
- 1D_DUV_LED_HirayamaJAP2005_SL_nnp.in

**Table of contents**

- **Hole density estimation**
  - Structure
    - Bandedges
  - Scheme
    - Schrödinger-Poisson equation
    - Ionization of dopant
  - Results
    - Spatially averaged hole density
    - Hole density/Ionized acceptor density distribution
- **IQE estimation**
  - Structure
    - Bandedges
  - Scheme
  - IQE result
- **What can we do further?**
Hole density estimation

Structure

The simulation region consists of the following structure:

<table>
<thead>
<tr>
<th>Material</th>
<th>Thickness</th>
<th>Doping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al$<em>{0.2}$Ga$</em>{0.8}$N/GaN 8-layer MQW</td>
<td>$L = L_{\text{well}} = L_{\text{barrier}}$</td>
<td>$5.0 \times 10^{19}$ [cm$^{-3}$]</td>
</tr>
</tbody>
</table>

The simulation is swept over the well and barrier thickness $L$ from 1 nm to 10 nm.

Bandedges

The following figure shows the bandedge profile and the Fermi energy for the SL structure with $L = 4.0$ nm.

![Bandedge profile and Fermi level](image)

**Figure 13.1.12.1: The bandedge profile and the Fermi level**

The bandedge tilting is due to the piezo- and pyro-electricity, which actually enhances the acceptor ionization as can be seen later.

Scheme

Schrödinger-Poisson equation

We can specify which simulation or equations would be solved on the `run` section in your input file. In `1D_UV_LED_KozodoyAPL1999_nnp.in` it is described as
run{
    strain()
    poisson()
    quantum_poisson()
}

Then nextnano++ solves the strain equation and self-consistent Schrödinger-Poisson equation. The resulting electrostatic potential \( \phi(x) \), electron density \( n(x) \), and hole density \( p(x) \) should satisfy both Poisson equation and the carrier density calculation based on Schrödinger equation. For further detailed discussion, please refer to *General scheme of the optical device analysis*.

### Ionization of dopant

The ionized donor and acceptor densities, \( N_D^+, N_A^- \) are calculated as

\[
N_D^+(x) = \sum_{i \in \text{Donors}} \frac{N_{D,i}(x)}{1 + g_D,i \exp((E_{F,n}(x) - E_{D,i}(x))/k_B T)}
\]

\[
N_A^-(x) = \sum_{i \in \text{Acceptors}} \frac{N_{A,i}(x)}{1 + g_A,i \exp((E_{F,p}(x) - E_{A,i}(x))/k_B T)}
\]

where the summation is over all different donor or acceptors, \( N_D, N_A \) are the doping concentrations, \( g_D, g_A \) are the degeneracy factors (\( g_D = 2 \) and \( g_A = 4 \) for shallow impurities), and \( E_D, E_A \) are the energies of the neutral donor and acceptor impurities, respectively.

These energies \( E_D, E_A \) are determined by the ionization energies \( E_{D,i}^{\text{ion}}, E_{A,i}^{\text{ion}} \), the bulk conduction and valence bandedges (including shifts due to strain) and the electrostatic potential as

\[
E_D(x) = E_{c,0}(x) - e\phi(x) - E_{D,i}^{\text{ion}}(x),
\]

\[
E_A(x) = E_{v,0}(x) - e\phi(x) + E_{A,i}^{\text{ion}}(x).
\]

The parameters can be specified in the input file as follows:

- **Doping concentrations** \( N_D, N_A \) are specified at `structure{ region{ doping{ } } }` like

```plaintext
structure{
    region{
        doping{
            #constant{
                # name = "donor_impurity"
                # conc = 2.0e18 # cm^-3
            } #
            constant{
                name = "acceptor_impurity"
                conc = 5.0e19 # cm^-3
            }
        }
    }
}
```

- **The degeneracy factors** \( g_D, g_A \) and ionization energies \( E_{D,i}^{\text{ion}}, E_{A,i}^{\text{ion}} \) are specified at `impurities{}` like

```plaintext
impurities{
    donor{
        name = "donor_impurity" # Si
        energy = 0.030 # ionization energy measured from the
                        conduction bandedge. (fully ionized when -1000)
    }
}```

(continues on next page)
Results

Spatially averaged hole density

Here we show the relation between $L = L_{\text{well}} = L_{\text{barrier}}$ and the spatially averaged hole densities. The orange line is the result of Poisson equation ignoring the polarization fields, the blue line is the result of Poisson equation including the polarization fields, and the green line is the result of Schrödinger-Poisson equation including the polarization fields.

The corresponding hole density for bulk Al$_{0.2}$Ga$_{0.8}$N with the same acceptor concentration $5.0 \times 10^{19} \text{ [cm}^{-3}\text{]}$ has been calculated as around $0.43 \times 10^{18} \text{ [cm}^{-3}\text{]}$, so the hole density is improved in any case.

What we can also see is that the polarization field further enhances the acceptor ionization, while the quantization effect reduces it as $L$ becomes smaller.

Figure 13.1.12.2: Barrier and well width $L$ and spatially averaged hole densities.
Hole density/ionized acceptor density distribution

Here we see the spatial distribution of the hole density and ionized acceptor density. We can confirm that the holes generated by the ionization of the acceptors in the barrier layers are accumulated into the well layers.

![Hole density distribution](image)

Figure 13.1.12.3: Hole density distribution calculated at $L = 4.0$ nm by Schrödinger-Poisson equation including the polarization fields. The valence bandedges are also displayed.

**IQE estimation**

**Structure**

The simulation region consists of the following structure:

<table>
<thead>
<tr>
<th>Material</th>
<th>Thickness</th>
<th>Doping</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-Al$<em>{0.18}$Ga$</em>{0.82}$N</td>
<td>100nm</td>
<td>$8 \times 10^{18}$ [cm$^{-3}$] (donor)</td>
</tr>
<tr>
<td>In$<em>{0.02}$Al$</em>{0.09}$Ga$_{0.89}$N</td>
<td>3-layer MQW</td>
<td>well: 2.5nm, barrier: 15nm</td>
</tr>
<tr>
<td>Al$<em>{0.24}$Ga$</em>{0.76}$N/Al$<em>{0.17}$Ga$</em>{0.83}$N 8-layer MQW</td>
<td>well: 4.0nm, barrier: 2 $\times$ 10$^{19}$ [cm$^{-3}$] (acceptor)</td>
<td></td>
</tr>
<tr>
<td>$p$-Al$<em>{0.17}$Ga$</em>{0.83}$N as a p-contact layer</td>
<td>20nm</td>
<td>2 $\times$ 10$^{19}$ [cm$^{-3}$] (acceptor)</td>
</tr>
</tbody>
</table>

The simulation result of this structure is compared with the structure where the p-region consists of bulk Al$_{0.20}$Ga$_{0.80}$N.

The electron blocking layer is not included here.
Figure 13.1.12.4: Ionized acceptor density distribution calculated at $L = 4.0$ nm by Schrödinger-Poisson equation including the polarization fields. The valence bandedges are also displayed.

**Bandedges**

The following figures show the bandedge profiles and the Fermi energies for the structures with (top) and without (bottom) SL. The width of the SL wells and barriers is set to $L = 4.0$ nm.

**Scheme**

The corresponding `run{}` section is described as

```plaintext
run{
  strain()
  current_poisson()
}
```

Then `nextnano++` solves the current equation and Poisson equation self-consistently after solving strain equation. After the Current-Poisson equation has been converged, optoelectronic characteristics are calculated according to the specification in the section `classical{}`.

For further details, please see *General scheme of the optical device analysis*. 
Figure 13.1.12.5: The bandedges and Fermi levels for the structure with SL (bias=4.00 V, total current density=\(2.67 \times 10^5\) A/cm\(^2\))

Figure 13.1.12.6: The bandedges and Fermi levels for the structure with bulk p-region (bias=3.97 V, total current density=\(2.71 \times 10^5\) A/cm\(^2\))
IQE result

The calculated IQEs with respect to the applied bias (left) and current density (right) are shown here. We can see that the IQE for the structure with SL, which is slightly smaller than that of bulk at the bias around 3.4 V, becomes superior to bulk for larger biases.

![Bias voltage and IQE](image1.png) ![Current density (p-contact) and IQE](image2.png)

Figure 13.1.12.7: Left: Applied bias and IQE. Right: Current density at p-contact and IQE.

What can we do further?

By sweeping the simulation over the corresponding parameters, we can optimize the device structures such as $L$, number of SLs, or the Al content of the SL region, for example. Our open source python package nextnanopy is a powerful tool for this purpose.

The graphs shown in this tutorial are also generated by a python script using nextnanopy.

Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.

13.1.13 1D Optics: Optical absorption for interband and intersubband transitions

Section author: Takuma Sato
In this tutorial we illustrate the `optics` module to demonstrate what `nextnano++` can simulate for optoelectronic devices. This module performs a detailed calculation to **optical absorption phenomena**, using **8 (or 6) band k · p models**. If you are interested in

- the background physics of this module and how to write the input file, go to *Principle and nextnano++ implementation*.
- the simulation results for intersubband transitions, go to *1D tutorial for intersubband transitions: Quantum well infrared photodetector*.
- the simulation results for interband transitions, go to *1D tutorial for interband transitions: Frankenberger*.
- optical absorption in 2D devices, *(under construction)*
- optical absorption in broken-gap structures, *(under construction)*

This algorithm is implemented based on the following diploma thesis:


For the physics of optical transition in semiconductors and its application, we refer to


### Principle and nextnano++ implementation

#### k⊥ space

In the k.p analysis of one- (or two-) dimensional structures we have a projection of the Bloch wave vector along translation-invariant directions. We denote them as \( \mathbf{k}_\parallel = k_y \hat{y} + k_z \hat{z} \) (1D) and \( \mathbf{k}_\parallel = k_z \hat{z} \) (2D). Under envelope function approximation the k.p model yields the following equation to determine the confined states in structured directions

\[
\sum_{\mu=1}^{8} H_{\nu\mu} \psi_{\nu}(\mathbf{r}_\perp) = E_{\nu}(\mathbf{k}_\parallel) \psi_{\nu}(\mathbf{r}_\perp) \quad (\nu = 1, \ldots, 8),
\]

where the Greek indices label the k.p bands and \( m \) denotes eigenvalues, \( \mathbf{r}_\perp = x \hat{x} \) in 1D and \( \mathbf{r}_\perp = x \hat{x} + y \hat{y} \) in 2D. \( H_{\nu\mu} \) is the \( 8 \times 8 \) matrix whose elements are given by the k.p parameters in the database. \( f_{m,\nu}(\mathbf{r}_\perp) \) are the envelopes in the structured directions. The full wave function is given at each \( \mathbf{k}_\parallel \) as

\[
\psi_{\nu}(\mathbf{k}_\parallel, \mathbf{r}) = \sum_{\mu=1}^{8} F_{m,\nu}(\mathbf{k}_\parallel, \mathbf{r}) u_{\mu}(\mathbf{r}) = \sum_{\mu=1}^{8} e^{i\mathbf{k}_\parallel \cdot \mathbf{r}_\perp} \sqrt{A} f_{m,\nu}(\mathbf{r}_\perp) u_{\mu}(\mathbf{r}),
\]

where \( u_{\mu}(\mathbf{r}) \) is the Bloch function of the band \( \mu \) at \( \mathbf{k} = 0 \) and \( A = \int d\mathbf{r}_\parallel \). In general, both the conduction band (\( \Gamma \)) and valence bands contribute to this full wave function. The spinor composition is exported to Quantum\/spinor_composition. After solving this “Schrödinger” equation, the wave function is integrated over a limited region in \( \mathbf{k}_\parallel \) space to obtain the charge density, which is used in the quantum-current-Poisson iteration. The region is specified under `quantum/` as

```plaintext
continue on next page
```

(continues on next page)
Note: When \texttt{force\_k0\_subspace=yes} in \texttt{quantum()} or \texttt{optics()}, the Schrödinger equations at non-zero \(k\)-points are solved in the subspace of the eigenfunctions obtained by the Schrödinger equation at \(k\|=0\). This approximation largely improves the computational speed. In case you are planning to use this approximation for final results, please make sure to check whether the resulting loss of accuracy in density is acceptable (\texttt{quantum()})) or the loss in optical spectra is acceptable (\texttt{optics()}).

### Optical absorption spectrum

When 1) Schrödinger equation is solved with \(k.p\) method, 2) \texttt{optics()} flag is present and 3) the specifier \texttt{optics()} is present under \texttt{run()} flag, \texttt{nextnano++} calculates the absorption spectrum.

\[
\alpha(\tilde{\epsilon}, \omega) = \frac{\pi e^2}{n_s \epsilon_0 m_0^* \omega} \frac{1}{V} \sum_{n>m} \sum_{k_{\parallel}} |\tilde{\epsilon} \cdot \tilde{r}_{nm}(k_{\parallel})|^2 (f_m - f_n) \delta(E_n - E_m - \hbar \omega),
\]

(13.1.13.3) where the first sum runs over bands that fulfill \(E_n > E_m\), and \(f_m(k_{\parallel}) = [1 + e^{(E_m(k_{\parallel}) - E_F)/k_B T}]^{-1}\) is the occupation of eigenstate \(m\). When \texttt{optics( occupation\_ignore=yes )} (default is no), the program assumes

\[
\begin{cases}
  f_m(k_{\parallel}) = 0 & \text{if } m \in \text{conduction band} \\
  f_m(k_{\parallel}) = 1 & \text{if } m \in \text{valence band}
\end{cases}
\]

The light polarization \(\tilde{\epsilon}\) and refractive index \(n_s\) are specified in the input file. The refractive index is in general frequency-dependent, but we assume it to be constant and equal to the substrate value.

502 Chapter 13. \texttt{nextnano++} Tutorials
The core of the optical transition is the optical matrix elements \( \vec{\pi}_{nm}(k_\parallel) \) because the kinematic momentum operator \( \vec{\pi} = (\pi_x, \pi_y, \pi_z) \) couples linearly to the vector potential that accounts for the electromagnetic field. Here \( \vec{\pi} \) is the sum of the conventional momentum operator \( p \) and the contribution of spin-orbit interaction. The optical matrix elements are calculated as

\[
\vec{\pi}_{nm}(k_\parallel) = \langle n | \vec{\pi} | m \rangle = \int dr \left( F_{n1}^* \cdots F_{n8}^* \right) \left( \begin{array}{c} F_{m1} \\ \vdots \\ F_{m8} \end{array} \right),
\]

where the \( 8 \times 8 \) matrix representation of the momentum operator, \( \vec{\pi}_{kp8}^{\mu} \), has been derived using the Hellmann-Feynman theorem extended to the 8 band \( k.p \) model up to first order in \( k \) [Eißfeller]. For the analysis of the absorption spectrum, nextnano++ also prints out some fractions of the absorption coefficient formula in the output folder, namely

1. occupation (if \text{output\_occupations=\text{yes}}) \text{Optics\_occupation\_\_\_\_.dat} \( f_m(k_\parallel) \)
2. eigenvalue dispersion (if \text{output\_energies=\text{yes}}) \text{Optics\_energy\_disp\_\_\_.dat} \( E_m(k_\parallel) \)
3. transition intensity (if \text{output\_transitions=\text{yes}}) \text{Optics\_transition\_disp\_\_\_.dat} \( T_{nm}(\vec{\epsilon}, k_\parallel) = \frac{2}{m_0} \vec{\epsilon} \cdot \vec{\pi}_{nm}(k_\parallel) \)
4. imaginary part of the dielectric function for each transition (if \text{output\_spectra\{output\_components=\text{yes}\}}) \text{Optics\_imepsilon\_\_\_\_.dat} \( \text{Im} \varepsilon_{nm}(\vec{\epsilon}, \omega) \)
5. total imaginary part of the dielectric function \( \text{Optics\_imepsilon\_\_\_\_.dat} \text{Im} \varepsilon(\vec{\epsilon}, \omega) = \sum_{n>m} \text{Im} \varepsilon_{nm}(\vec{\epsilon}, \omega) \)
6. total absorption coefficient spectrum \( \text{Optics\_absorption\_\_\_\_.dat} \alpha(\vec{\epsilon}, \omega) = \sum_{n>m} \alpha_{nm}(\vec{\epsilon}, \omega) = \sum_{n>m} \omega_{ns} \frac{\text{Im} \varepsilon_{nm}(\vec{\epsilon}, \omega)}{m_0} \)

The following part of the input specifies how much transitions to be taken into account. The setting for \text{k\_integration{}} is explained in the next section.
Parameters in \texttt{k\_integration\{} (for fine tuning)\

Parameters in \texttt{k\_integration\{} in \texttt{optics\{} flag (hereafter \(r_{\text{opt}}, N_{\text{opt}}, N'_{\text{opt}}\)) specify the size and resolution of the \texttt{k\_parallel} space integration in absorption calculation, \(\sum_{k_{||}}\). This should not be confused with the specifier \texttt{k\_integration\{} in \texttt{quantum\{} flag used for quantum mechanical charge density integration (hereafter \(r_{q}, N_{q}, N'_{q}\), see Figure 13.1.13.1).

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure13_1_13_1}
\caption{Calculation algorithm of optical absorption and its relation to the parameters in \texttt{k\_integration\{}. \(r_{q}, N_{q}, N'_{q}\) and \(r_{\text{opt}}, N_{\text{opt}}, N'_{\text{opt}}\) are specified in \texttt{quantum\{} and \texttt{optics\{} respectively. To do; the energy dispersion is interpolated with \(N'_{q}\) or \(N'_{\text{opt}}\)?}
\end{figure}

First we discuss the parameters \(r_{\text{opt}}\) and \(N_{\text{opt}}\). The size of \texttt{k\_parallel} space may affect the validity of simulation results. It also determines the simulation load. Here are some hints to determine the appropriate parameter sets:

- In undoped systems, integrating up to \(|k_{\parallel}|\) that gives in-plane kinetic energy \(\hbar^{2}k_{\parallel}^{2}/2m\) corresponding to \(2k_{\parallel}T\) or \(3k_{\parallel}T\) should be sufficient. Usually \(r_{\text{opt}} = 0.3\) is sufficiently large to include all occupied states. In doped systems, it depends on the Fermi energy.

- To see the range of occupied states in \texttt{k\_parallel} space, run a simulation and look at the output \texttt{\Output\occupation\_\~.dat}. We recommend to check the box “Show grid” on the left panel in Output tab of \texttt{nextnanomat} (see also \texttt{Output}). This shows the occupation \(f_{m}(k_{\parallel})\) as a function of \(k_{\parallel}\). Let us consider 1D simulation and suppose you got the following:
where \((r_{\text{opt}}, N_{\text{opt}}) = (0.3, 8)\). The horizontal- and vertical axes are \(k_y\) and \(k_z\), respectively. The area \(|k_{yz}| \leq r_{\text{opt}}\) is shown with the kll-space gridding (thin white lines). The number of kll points in one direction is \(2N_{\text{opt}} + 3\). The occupation profile is not smooth and you might want a higher resolution by increasing the parameter \((r_{\text{opt}}, N_{\text{opt}}) \rightarrow (0.3, 60)\):

The occupation becomes smooth, but at the same time this significantly increases the number of k points (in 1D simulation, the number of k points) \(\propto (r_{\text{opt}}N_{\text{opt}})^2\). Noting that the black region, where occupation is zero, does not contribute to the absorption, you can “zoom in” to the colored region by decreasing \(r_{\text{opt}}\) and \(N_{\text{opt}}\) in such a way that the ratio \(r_{\text{opt}}/N_{\text{opt}}\) remains constant. This will cut down the irrelevant region without changing the resolution. For example, if you set \((r_{\text{opt}}, N_{\text{opt}}) = (0.05, 10)\), you obtain

and this should be sufficient for the kll-space integration.

After tuning the parameters \(r_{\text{opt}}, N_{\text{opt}}\), we can further optimize the setting regarding to the interpolation. The number of subpoints \(N'_{\text{opt}}\) determines at how many kll points the transition intensity should be interpolated. Increasing \(N'_{\text{opt}}\) gives \(E_m(k_{ll})\) of higher resolution and makes the absorption spectrum smooth. Figure 13.1.13.2 shows that this parameter improves the absorption coefficient spectrum.

*To do: investigate* spin_degeneracy=yes/no and dipole_approximation = yes/no
The effect of the parameter $N'_{\text{opt}}$ specified in `options{k_integration{}}` on absorption spectrum output `/Optics\absorption`. Larger $N'_{\text{opt}}$ smoothens the $k_{\parallel}$-dependence of the integrand, which leads to a smoother spectrum.

**1D tutorial for intersubband transitions: Quantum well infrared photodetector**

In the following we apply the formalism to several devices. As a first example, we simulate the absorption spectrum of an AlGaAs/GaAs quantum well infrared photodetector (QWIP). The QWIP is based on photoconductivity due to intersubband excitation.

**Input files**

- `QWIP_singleQW_GaAs_AlGaAs_nnp.in`
- `QWIP_singleQW_InAs_AlSb_nnp.in`
- `QWIP(Gunapala_Levine_Chad-J.Appl.Phys70-1991)_nnp.in`

The first example uses the same parameters used in


while the third example is based on

GaAs/AlGaAs single QW - band structure, eigenstates and absorption

We first illustrate the first example \texttt{QWIP\_singleQW\_GaAs\_AlGaAs\_nnp.in}. In this example, we simulate optical absorption in single quantum well structure. The following input is required for self-consistent quantum-current-Poisson simulation:

```plaintext
quantum{
  region{
    name "optical_active"
    no_density no
    kp_8band{
      num_electrons $OptNumE
      num_holes $OptNumH
    }
  }
}
poisson{}
current{}
run{
  strain{}  # strain calculation
  current_poisson{}
  quantum_current_poisson{}
  optics{}  # absorption calculation
}
```

The specifier \texttt{no\_density=no} lets the program calculate quantum mechanical charge density (default). Current-Poisson equation takes over this value. The bandstructure and wave functions are shown in Figure 13.1.13.3 and Figure 13.1.13.4, respectively.

![Bandstructure](image)

**Figure 13.1.13.3**: Single quantum well structure \texttt{\_bandedges.dat}. The bias voltage between two contacts is set to 2mV.

The output folder \texttt{\_Optics} contains simulation results for optical absorption. Let us first check the occupation \( f_m(k||) \) used in the absorption calculation. When comparing the results \texttt{\_Optics\_occupation}, please mind
Figure 13.1.13.4: Probability distribution $|\psi(x)|^2$ of the confined states at $k_{\parallel} = 0$ (\Quantum\probabilities_shift_optical_active). The wave functions here are the solution to the 8-band k.p model. The energy separation is $\Delta E=0.06960-(-0.05589)=0.1255$[eV] according to the output data. The electron Fermi energy lies between two bound states.

The absorption coefficient for TE ($\vec{\epsilon} = \hat{y}$) and TM ($\vec{\epsilon} = \hat{x}$) light polarization is shown in Figure 13.1.13.7. The energy grid spacing here is $\textit{ENERGY\_RESOLUTION}=0.5$meV. For single-band models the peak becomes very sharp unless one introduces phenomenological broadening function such as Lorentzian. In k.p calculation, in contrast, peaks gets broadened because the transition energies, $E_n(k_{\parallel}) - E_m(k_{\parallel})$, depends on $k_{\parallel}$. One can confirm this by comparing the output \Optics\energy_disp_~.dat for states $m=1$ and 2 (not shown). In intersubband transitions the transition energies can be concave downward in $k_{\parallel}$ space, i.e., $E_n(k_{\parallel}) - E_m(k_{\parallel}) \propto -k^2$, depending on the masses. In the present case the absorption spectrum has a tail in the region $\hbar \omega < \Delta E$.

The optical transitions between conduction band states (intersubband transitions) in response to TE-polarized light is only allowed when eigenstates have finite spinor components in valence bands. In the present case its large bandgap and small confinement leads to small band-mixing, rendering TE absorption orders of magnitude smaller than TM polarization (Figure 13.1.13.7). As seen in the output \Quantum\spinor_composition_~.dat, eigenstates contain approximately 98% contribution from conduction band and 2% from valence band.
Figure 13.1.13.5: Occupation of the first (m=1) bound states as a function of $k_{||}$.

Figure 13.1.13.6: Occupation of the second (m=2) bound states as a function of $k_{\perp}$. 

Figure 13.1.13.7: Absorption coefficient in \Optics\absorption_-.dat as a function of photon energy, for TE and TM. Black arrow points the energy separation $\Delta E$. The broadening of the spectrum is due to the $k_l$-dependence of wave functions and corresponding eigenvalues.

**InAs/AlSb single QW - small bandgap & large confinement**

In the second example **QWIP_singleQW_InAs_AlSb_nnp.in**, single quantum well is narrower and the bandgap is smaller than the first example. The small bandgap and large confinement of the wave function (Figure 13.1.13.8) leads to large band mixing. In fact, the output \Quantum\spinor_composition_-.dat shows that the ground states in Figure Figure 13.1.13.8 consists of 80.7% of conduction band and 19.3% of valence band contribution.

**Periodic case**

In the third example **QWIP(Gunapala_Levine_Chad-J.Appl.Phys70-1991)_nnp.in**, we set the bias to zero and impose the periodic boundary condition. The GaAs/Al$_x$Ga$_{1-x}$As superlattice structure induces miniband states below the barriers, enabling bound-to-continuum absorptions of sub-eV photons. This $\mu$m-wavelength photodetector works without electron tunneling through the barriers, thereby improving the detectivity [Gunapala]. The band structure bandedges.dat and wave functions \Quantum\probabilities_shift.dat are shown in Figure 13.1.13.10. We have continuum states above the barriers as well as bound states in the superlattice (miniband).

The absorption coefficient is exported to \Optics\absorption. The indices in the filename *_kp8_TE_m_n.dat refer to the transition from state m to state n. The files without indices contain the total absorption (sum over all transitions). The total absorption coefficient for TE and TM polarization looks like this: The peak positions do not depend on polarization, while the peak height is much larger for TM polarization compared to the one for TE. Looking at the absorption spectrum for each transition, we identify which transition contributes to which peak (Figure 13.1.13.12).

Let us look at the eigenvalue and occupation of each state to confirm this result. The eigenvalues of the bound-
Figure 13.1.13.8: Confined states at $k_{\parallel} = 0$ (\texttt{Quantum\probabilities\_shift\_optical\_active}) in a narrower and deeper quantum well. The blue line marks the electron Fermi energy (0eV).

Figure 13.1.13.9: Absorption coefficient as a function of photon energy for TE and TM. TE absorption becomes relevant compared to Figure 13.1.13.7 because of the large band-mixing. Note that TE spectrum here is multiplied by a factor of 100, instead of 1000 in Figure 13.1.13.7.
Figure 13.1.13.10: Gamma band profile and probability distribution of the bound miniband states and continuum states above the top of the barriers.

Figure 13.1.13.11: Absorption coefficient as a function of photon energy for TE ($\vec{e} = \hat{y}$) and TM ($\vec{e} = \hat{x}$) polarization. TE spectrum is magnified by factor of 1000. We observe that TM absorption is much larger than TE, while the peak positions are the same.
and continuum states are written in the output \Quantum\probabilities_shift.dat or \Quantum\energy_spectrum.

**Note:** quantum{} uses spin-resolved index for the eigenstates, so there are 80 states in total. In optics{}, however, two spin-degenerate states are summed up and there are only 40 states. This number (1 to 40) is used in the \Optics output filenames. For the consistency, we use the latter notation throughout. *(To do: examine the specifier spin_degeneracy)*

Based on the indices in **Figure 13.1.13.12**, we identify the first four peaks to the following four different transitions (**Figure 13.1.13.13**). We have confirmed that the peak energies in **Figure 13.1.13.12** are consistent to the energy separation of the corresponding states.

Lastly we check the occupation (Fermi-Dirac distribution) $f_m(k_{\parallel})$. In the output \Optics\eigenvaluespectrum (**Figure 13.1.13.14**), occupation at $k_{\parallel}=0$ of $m$-th state, $f_m(k_{\parallel} = 0)$, is plotted at corresponding eigenvalues $E_m$. The function takes the maximum value at the origin $k_{\parallel} = 0$. In the present system, $f_1(0) = 0.087, f_2(0) = 0.077, \ldots, f_{10}(0) = 0.0148$ for the bound states, whereas $f_m(0) < 10^{-4}$ for continuum states ($m \geq 11$). Therefore the initial states in **Figure 13.1.13.13** are well occupied and the final states are mostly empty. This enables optical absorption via bound-to-continuum excitation of electrons, thereby realizing a quantum well photodetector with high detectivity.
Figure 13.1.13.13: Eigenenergies of relevant bound- and continuum states. Many other transitions have little contribution due to the shape of the wave functions and/or occupation of the states. When we calculate for wider energy range, i.e. increase the parameter $\text{ENERGY\_MAX}$, there will be many more peaks that are attributed to higher energy transitions.

Figure 13.1.13.14: Occupation of eigenstates showing a noticeable difference for bound (m=1-10) and continuum (m=11,\ldots) states.
1D tutorial for interband transitions: Frankenberger

Input files

- AlGaAs_QW_Frankenberger_Simple_nnp.in
- AlGaAs_QW_Frankenberger_Simple_nnp_fast.in
- AlGaAs_QW_Frankenberger_Doping_schottky07_nnp.in
- AlGaAs_QW_Frankenberger_Doping_schottky07_nnp_fast.in

These files are located in the sample files folder. The fast examples reduce the computation load by limiting exact solution only to $k = 0$ point and computing all other $k$ points in the basis of the $k = 0$ wavefunctions (force_k0_subspace; see quantum/ and optics/ documentations).

Optical absorption and interband transitions

In the input file AlGaAs_QW_Frankenberger_Simple_nnp.in, we consider a single quantum well structure:

![Bound states](image)

Figure 13.1.13.15: The conduction bandedge profile (bandedges.dat) and wave functions of the bound states (\Quantum\probabilities_shift).

The program solves the 8-band k.p model coupled to the Poisson equation to find the eigenstates and compute the absorption coefficient. Figure Figure 13.1.13.16 shows the absorption coefficient spectrum for circularly polarized light ($\vec{\epsilon} = \hat{y} - i \hat{z}$). In contrast to QWIP examples above, peaks have long tails toward higher energy. This is because the transition energies $E_n(k_i) - E_m(k_f)$ in interband transitions are concave upward $\sim +k^2$ (here we do not consider Type 2 semiconductors).

The steps of this absorption spectrum are associated with the following interband transitions:

Note: In the end of the log file, you find the message “Integration reliable up to $-eV$”. This tells you up to which energy the absorption spectrum is reliable. Since we only consider the vicinity of the origin $k_\parallel = 0$, the
Figure 13.1.13.16: Absorption coefficient of circularly polarized lights. Numbers “m-n” denote each transition $m \rightarrow n$. The first four transitions are sketched in Figure 13.1.13.17.

The reliable energy interval is bound from above by the energy difference of the initial and final states at the edge of the $k_{||}$-space considered. The upper limit $d$ [eV] is given by

$$d = \min_{k_{||} \in \Omega^*_{\text{edge}}} |E_n(k_{||}) - E_m(k_{||})|$$

where $\Omega^*$ is the region in $k_{||}$-space specified in \texttt{optics{} region{ k_integration{} } } with parameters $r_{opt}$ and $N_{opt}$. In the present case $d=3.2$ eV, while the calculation is safely performed for the interval $[1.4, 1.7]$ (eV). This message appears only when interband transitions are computed, i.e. when \texttt{interband=yes} and \texttt{intraband=no} in \texttt{optics{} flag}.

### Doping and Schottkey barrier

In the second input file \texttt{AlGaAs_QW_Frankenberger_Doping_schottky07_nnp.in}, we consider the following structure:

Figure 13.1.13.20 compares the results for different settings for occupation $f_m(k_{||})$. When \texttt{optics{} occupation_ignore=yes }, valence bands and conduction bands are considered to be fully occupied and fully empty, respectively. When the actual occupation of eigenstates are taken into account, in contrast, optical transitions to conduction band states just above the Fermi energy are prohibited because of the thermal distribution of electrons.
Figure 13.1.13.17: Eigenvalues (black) and transitions from valence-band to conduction band bound states (arrows) which are responsible for the first four steps in Figure 13.1.13.16. Here spin-degenerate states are counted as one state (eigenstate numbering in optics{}).
Figure 13.1.13.18: The bandstructure and eigen functions used for optics calculation. The Fermi level is at 0eV.

Figure 13.1.13.19: Absorption coefficient of circularly polarized lights. Numbers “m-n” denote each transition $m \rightarrow n$. 
Figure 13.1.13.20: Absorption coefficient for different settings of occupation. The red curve is identical to the total absorption in Figure 13.1.13.19. When occupation_ignore=no, absorption of low energy photons is suppressed due to the occupation of lowest conduction band states (also see Figure 13.1.13.18).

13.1.14 Intersubband absorption of an infinite quantum well

This tutorial calculates the intersubband absorption of a GaAs quantum well with infinite barriers.

Input files for both the nextnano++ and nextnano3 software are available.

The following input file was used:

- 1D_IntersubbandAbsorption_InfiniteWell_GaAs_Chuang_sg_nn3.in (single-band effective mass approximation)

This tutorial aims to reproduce the example discussed on p. 376f of Section 9.6.2 Intersubband Absorption Spectrum of [ChuangOpto1995].

Structure

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>unit</th>
<th>[ChuangOpto1995]</th>
<th>nextnano</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantum well width</td>
<td>L</td>
<td>nm</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>barrier height</td>
<td>E_b</td>
<td>eV</td>
<td>infinite quantum well model 1000</td>
<td>1000</td>
</tr>
<tr>
<td>effective electron mass</td>
<td>m_e</td>
<td>m_0</td>
<td>0.0665</td>
<td>0.0665</td>
</tr>
<tr>
<td>refractive index</td>
<td>n_r</td>
<td></td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>doping concentration (n-type)</td>
<td>N_D</td>
<td>cm⁻³</td>
<td>1 × 10¹⁸</td>
<td>1 × 10¹⁸</td>
</tr>
<tr>
<td>linewidth (FWHM)</td>
<td>Γ</td>
<td>meV</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>temperature</td>
<td>T</td>
<td>K</td>
<td>300</td>
<td>300</td>
</tr>
</tbody>
</table>

[ChuangOpto1995] models the infinite quantum well using the analytical solution while we are using a numerical model with a barrier height of 1000 eV.
Results

[ChuangOpto1995] uses the analytical infinite quantum well model and calculates the energy levels, and the intersubband dipole moment exactly. Our calculated transition energies differ by 3 meV which is acceptable as we use a finite grid spacing of 0.05 nm. Our calculated dipole moment is also reasonable. More difficult are the densities. In our calculation we solve the Schrödinger-Poisson equation self-consistently. For that reason, the quantum well bottom is not entirely flat but slightly bended. At T = 300 K, the second subband shows a small density which is larger than in the model of [ChuangOpto1995]. The difference in subband densities leads to a slight deviation for the peak of the absorption curve because the occupation of the second level N₂ decreases the absorption. Nevertheless, the agreement is reasonable.

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>unit</th>
<th>[ChuangOpto1995]</th>
<th>nextnano</th>
</tr>
</thead>
<tbody>
<tr>
<td>energy level</td>
<td>E₁</td>
<td>meV</td>
<td>56.5 (exact)</td>
<td></td>
</tr>
<tr>
<td>energy level</td>
<td>E₂</td>
<td>meV</td>
<td>226 (exact)</td>
<td></td>
</tr>
<tr>
<td>transition energy</td>
<td>E₂₁</td>
<td>meV</td>
<td>169.5 (exact)</td>
<td></td>
</tr>
<tr>
<td>dipole moment</td>
<td>x₂₁</td>
<td>nm</td>
<td>-1.8 (exact)</td>
<td>-1.82</td>
</tr>
<tr>
<td>Eₚ - E₁</td>
<td>eV</td>
<td></td>
<td>78</td>
<td>28.2</td>
</tr>
<tr>
<td>subband density</td>
<td>N₁</td>
<td>cm⁻²</td>
<td>7.19 · 10¹¹</td>
<td>9.92 · 10¹¹</td>
</tr>
<tr>
<td>subband density</td>
<td>N₂</td>
<td>cm⁻²</td>
<td>3 · 10⁹</td>
<td></td>
</tr>
<tr>
<td>peak in absorption</td>
<td>αₚeₜₜ</td>
<td>cm⁻¹</td>
<td>1.015 · 10⁴</td>
<td>0.986 · 10⁴</td>
</tr>
</tbody>
</table>

The following figures show the

- lowest eigenstates (probability densities) of the infinite quantum well
- absorption spectra \( \alpha(\omega) \) in units of cm⁻¹
- position dependent absorption spectra \( \alpha(\omega, x) \) in units of cm⁻¹

The peak in the absorption spectra occurs at the transition energy \( E₂₁ \).

Then we perform two parameter sweeps:

- We vary the quantum well width (Variable: $QuantumWellWidth$).
- We vary the doping concentration (Variable: $DopingConcentration$).

Results and explanations for the sweeps can be found further below.

— Begin —

**Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnano**

The following documentation and figures were generated automatically using nextnano.

The following Python script was used: intersubband_InfiniteQW_nextnano3.py

The following figures have been generated using the nextnano³ software. Self-consistent Schrödinger-Poisson calculations have been performed for an infinite quantum well.

A single-band effective mass approach has been used, i.e. not \( k \cdot p \).

The absorption has been calculated assuming a parabolic energy dispersion \( E(k) \).

**Infinite Quantum Well** (QuantumWellWidth = 10 nm)

**Infinite Quantum Well** (QuantumWellWidth = 13 nm)

**Infinite Quantum Well** (QuantumWellWidth = 16 nm)

**Infinite Quantum Well** (QuantumWellWidth = 19 nm)

**Parameter sweep: Well width**
Figure 13.1.14.1: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 10 nm)

Figure 13.1.14.2: Calculated absorption $\alpha(E)$ of an infinite quantum well (QuantumWellWidth = 10 nm)
Figure 13.1.14.3: Calculated position resolved absorption $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 10 nm)

Figure 13.1.14.4: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 13 nm)
Figure 13.1.14.5: Calculated position resolved absorption $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 13 nm)

Figure 13.1.14.6: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 16 nm)
Figure 13.1.14.7: Calculated position resolved absorption $\alpha(x, E)$ of an infinite quantum well (QuantumWellWidth = 16 nm)

Figure 13.1.14.8: Conduction band edge, Fermi level and confined electron states of an infinite quantum well (QuantumWellWidth = 19 nm)
Figure 13.1.14.9: Calculated position resolved absorption $\alpha(x, E)$ of an infinite quantum well (QuantumWell-Width = 19 nm)

Figure 13.1.14.10 shows the absorption for different quantum well widths (Variable: $QuantumWellWidth$). The larger the well, the closer the energy level spacings. Therefore the peak occurs at smaller energies. The larger wells show absorption also for transitions other than $E_{21}$.

**Parameter sweep: Doping concentration**

Figure 13.1.14.11 shows the absorption for different doping concentrations (Variable: $DopingConcentration$). The peak absorption coefficient increases with the doping concentration $N_D$.

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

— End —

### 13.1.15 Intersubband transitions in InGaAs/AlInAs multiple quantum well systems

This tutorial calculates the eigenstates of a single, double and triple quantum wells. It compares the energy levels and wave functions of the single-band effective mass approximation with the 8-band $k \cdot p$ model. Finally, the intersubband absorption is calculated.

Input files for both the nextnano++ and nextnano^3 software are available.

The following input files were used:

- Single Quantum Well
  - 1DSirtoriPRB1994_OneWell_sg_self-consistent_nn*.in (single-band effective mass approximation)
Figure 13.1.14.10: Calculated absorption $\alpha(E)$ of an infinite quantum well for different well widths.

Figure 13.1.14.11: Calculated absorption $\alpha(E)$ of an infinite quantum well for different doping concentrations.
This tutorial aims to reproduce Figs. 4 and 5 of [SirtoriPRB1994].

This tutorial nicely demonstrates that for the ground state energy the single-band effective mass approximation is sufficient whereas for the higher lying states a nonparabolic model, like the 8-band $k \cdot p$ approximation, is necessary. This is important for e.g. quantum cascade lasers where higher lying states have a dominant role.

**Layer sequence**

We investigate three structures:

a) a single quantum well  
b) two coupled quantum wells  
c) three coupled quantum wells

**Material parameters**

We use In$_{0.53}$Ga$_{0.47}$As as the quantum well material and Al$_{0.48}$In$_{0.52}$As as the barrier material. Both materials are lattice matched to the substrate material InP. Thus we assume that the InGaAs and AlInAs layers are unstrained with respect to the InP substrate. The publication [SirtoriPRB1994] lists the following material parameters:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conduction band offset</td>
<td>0.510 eV</td>
</tr>
<tr>
<td>Conduction band effective mass</td>
<td>0.072 $m_0$</td>
</tr>
<tr>
<td>Conduction band effective mass</td>
<td>0.043 $m_0$</td>
</tr>
</tbody>
</table>

The temperature is set to 10 Kelvin.
Method

Single-band effective mass approximation

Because our structure is doped, we have to solve the single-band Schrödinger-Poisson equation self-consistently. The doping is such that the electron ground state is below the Fermi level and all other states are far away from the Fermi level, i.e. only the ground state is occupied and contributes to the charge density.

For nextnano++ we use:

```plaintext
# '0' solve Schroedinger equation only
# '1' solve Schroedinger and Poisson equations self-consistently
$SELF_CONSISTENT = 1

run{
  !IF($SELF_CONSISTENT)
    poisson{}
    quantum_poisson{ iterations = 50 } # Schrödinger-Poisson
  !ELSE
    quantum{}
    # Schrödinger only
  !ENDIF
}

quantum {
  region{
    Gamma{
      num_ev = 3 # 3 eigenstates
    }
  }
}
```

For nextnano3 we use:

```plaintext
# 'QUANTUM_ONLY': solve Schroedinger and Poisson equations self-
# consistently
# 'SELF_CONSISTENT': solve Schroedinger equation only
%QUANTUM_ONLY = F A L A S E
%SELF_CONSISTENT = T R U E

$simulation -flow -control
!IF %QUANTUM_ONLY flow-scheme = 3 # Schrödinger only
!IF %SELF_CONSISTENT flow-scheme = 2 # Schrödinger-Poisson
!IF %QUANTUM_ONLY raw-potential-in = yes
!IF %SELF_CONSISTENT raw-potential-in = no

$quantum -mode -control
  name = e f f e c t i v e - m a s s
  # single-band
  num_ev = 3 # 3 eigenstates
```

Note: Single-band eigenstates are two-fold spin degenerate.

The Fermi level is always equal to 0 eV in our simulations and the band profile is shifted accordingly to meet this requirement.

8-band k.p approximation

Old version of this tutorial:

Because both, the single-band and the 8-band $k \cdot p$ ground state energy and the corresponding wave functions are almost identical, we can read in the self-consistently calculated electrostatic potential of the single-band approximation and calculate for this potential the 8-band $k \cdot p$ eigenstates and wave functions for $k_{||} = 0$. 

13.1. 1D Tutorials 529
For nextnano$^3$ we use:

```plaintext
$simulation$
  !IF %QUANTUM_ONLY flow-scheme = 3  # Schrödinger only
  !IF %QUANTUM_ONLY raw-directory-in = raw_data/
  !IF %QUANTUM_ONLY raw-potential-in = yes

$quantum$
  name  8x8  # 8-band k.p
  band  6  # 6 eigenstates
```

**Note:** One k · p eigenstate for each spin component.

**New version of this tutorial:**

We provide input files for:

- a) self-consistent single-band Schrödinger equation (because the structure is doped)
- b) single-band Schrödinger equation (without self-consistency)
- c) 8-band k · p single-band Schrödinger equation (without self-consistency)

For a), although the structure is doped, the band bending is very small. Thus we omit for the single-band / k · p comparison in b) and c) the self-consistent cycle.

**Results**

**Single quantum well**

Figure 13.1.15.1 shows the lowest two electron eigenstates for an In$_{0.53}$Ga$_{0.47}$As / Al$_{0.48}$In$_{0.52}$As quantum well structure calculated with single-band effective mass approximation and with a nonparabolic 8-band k · p model.

The energies (and square of the wave functions $\psi^2$) for the ground state are identical in both models but the second eigenstate differs substantially. Clearly the single-band model leads to an energy which is far too high for the upper state.

Our calculated value for the intersubband transition energy $E_{12}$ of 255 meV compares well with both, the calculated value of [SirtoriPRB1994] (258 meV) and their measured value (compare with absorption curve in Fig. 4 of [SirtoriPRB1994]).

The calculated intersubband dipole moments are:

- $z_{12} = 1.55$ nm (single-band)

For comparison: $z_{12} = 1.53$ nm (exp.), $z_{12} = 1.48$ nm (th.) ([SirtoriPRB1994])

The influence of doping on the eigenenergies is negligible (smaller than 1 meV).

**Two coupled quantum wells**

Figure 13.1.15.2 shows the lowest three electron eigenstates for an In$_{0.53}$Ga$_{0.47}$As / Al$_{0.48}$In$_{0.52}$As double quantum well structure calculated with single-band effective mass approximation and with a nonparabolic 8-band k · p model.

The energies (and square of the wave functions $\psi^2$) for the ground state are very similar in both models but the second and especially the third eigenstate differ substantially. Clearly the single-band model leads to energies which are far too high for the higher lying states.

Our calculated values for the intersubband transition energies $E_{12} = 150$ meV and $E_{13} = 267$ meV compare well with both, the calculated values of [SirtoriPRB1994] (150 meV and 271 meV) and their measured values (compare with absorption curve in Fig. 5 (a) of [SirtoriPRB1994]).
Figure 13.1.15.1: Conduction band edge, Fermi level and confined electron states of a quantum well

Figure 13.1.15.2: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells
The calculated intersubband dipole moments are:

- \( z_{12} = 1.61 \) nm (single-band)
- \( z_{13} = 1.11 \) nm (single-band)

For comparison: \( z_{12} = 1.64 \) nm (exp.), \( z_{12} = 1.65 \) nm (th.) \([\text{SirtoriPRB1994}]\)

The influence of doping on the eigenenergies is almost negligible (between 0 and 2 meV).

### Three coupled quantum wells

Figure 13.1.15.3 shows the lowest four electron eigenstates for an In\(_{0.53}\)Ga\(_{0.47}\)As / Al\(_{0.48}\)In\(_{0.52}\)As triple quantum well structure calculated with single-band effective mass approximation and with a nonparabolic 8-band \( k \cdot p \) model.

The energies (and square of the wave functions \( \psi^2 \)) for the ground state are similar in both models but the second and especially the third and forth eigenstates differ substantially. Clearly the single-band model leads to energies which are far too high for the higher lying states.

Our calculated values for the intersubband transition energies \( E_{12} = 118 \) meV, \( E_{13} = 261 \) and \( E_{14} = 370 \) meV compare well with both, the calculated values of \([\text{SirtoriPRB1994}]\) (116 meV, 257 meV and 368 meV) and their measured values (compare with absorption curve in Fig. 5 (b) of \([\text{SirtoriPRB1994}]\)).

---

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnano

The following documentation and figures were generated automatically using nextnano.

The following Python script was used: intersubband_MQW_nextnano3.py
The following figures have been generated using the nextnano³ software. Self-consistent Schroedinger-Poisson calculations have been performed for three different structures:

- Single Quantum Well
- Two coupled Quantum Wells
- Three coupled Quantum Wells

The single-band effective mass and the 8-band $k \cdot p$ results are compared to each other. In both cases the wave functions and the quantum density are calculated self-consistently. The $k \cdot p$ quantum density has been calculated taking into account the solution at different $k\parallel$ vectors.

The absorption has been calculated using a simple model assuming a parabolic energy dispersion. The dipole moment $z_{ij} = \langle i | z | j \rangle$ has been evaluated only at $k\parallel = 0$. The subband density is used to calculate the absorption. For the $k \cdot p$ calculation, the density was calculated taking into account a nonparabolic energy dispersion, i.e. including all relevant $k\parallel$ vectors.

**Quantum Well (single-band)**

![Quantum Well (single-band)](image)

Figure 13.1.15.4: Conduction band edge, Fermi level and confined electron states of a quantum well

**Quantum Well (k.p)**

**Two Coupled Quantum Wells (single-band)**

**Two Coupled Quantum Wells (k.p)**

**Three Coupled Quantum Wells (single-band)**

**Three Coupled Quantum Wells (k.p)**

---

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

---

13.1. 1D Tutorials 533
Figure 13.1.15.5: Calculated position resolved absorption $\alpha(x, E)$ of a quantum well

Figure 13.1.15.6: Conduction band edge, Fermi level and confined electron states of a quantum well
Figure 13.1.15.7: Calculated position resolved absorption $\alpha(x, E)$ of a quantum well

Figure 13.1.15.8: Calculated absorption $\alpha(E)$ of a quantum well
Figure 13.1.15.9: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

Figure 13.1.15.10: Calculated position resolved absorption $\alpha(x, E)$ of two coupled quantum wells
Figure 13.1.15.11: Conduction band edge, Fermi level and confined electron states of two coupled quantum wells

Figure 13.1.15.12: Calculated position resolved absorption $\alpha(x, E)$ of two coupled quantum wells
Figure 13.1.15.13: Calculated absorption $\alpha(E)$ of two coupled quantum wells

Figure 13.1.15.14: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells
Figure 13.1.15.15: Calculated position resolved absorption $\alpha(x, E)$ of three coupled quantum wells

Figure 13.1.15.16: Conduction band edge, Fermi level and confined electron states of three coupled quantum wells
Figure 13.1.15.17: Calculated position resolved absorption $\alpha(x, E)$ of three coupled quantum wells

Figure 13.1.15.18: Calculated absorption $\alpha(E)$ of three coupled quantum wells
13.1.16 Piezoelectricity in wurtzite

nextnano++ and nextnano³ can simulate growth orientation dependence of the piezoelectric effect in heterostructures. Following A.E. Romanov et al., Journal of Applied Physics 100, 023522 (2006), we consider InₓGa₁₋ₓN and AlₓGa₁₋ₓN thin layers pseudomorphically grown on GaN substrates. The c-axis of the substrate GaN is inclined by an angle θ with respect to the interface of the heterostructure.

The layer is assumed to be very thin compared to substrate so that the strain is approximately homogeneous in all direction (pseudomorphic), and the ternary alloys mimic the orientation of crystallography direction. The layer material deforms such that the lattice translation vector of each layer has a common projection onto the interface.

The strain in a crystal induces piezoelectric polarization, which contributes as an additional component to the total charge density profile. The important consequence of their analysis is that the piezoelectric polarization normal to the interface becomes zero at a nontrivial angle. The piezoelectric charge in a heterostructure in general results in an additional offset between electron and hole spatial probability distribution, thereby reducing the overlap of their wavefunctions in real space. The small overlap of electron and hole leads to an inefficient radiative recombination, i.e. lower efficiency of optoelectronic devices. The work by Romanov et al. paved the way to device optimization by the growth direction of the crystal.

An introduction for the strain calculation is described here: Introduction to strain calculation

Table of contents

- Specify crystal orientation
- Parameter sweep of the angle using Template: Sweep over the variable theta
- Strain
- Piezoelectric effect (first-order)
- Postprocessing for polarization
- Alloy content dependence
- AlGaN
- Piezoelectric effect (second-order)

References


The corresponding input files are located in the nextnano++ sample files folder:

- Romanov_InGaN_theta_nnp.in
- Romanov_AlGaN_theta_nnp.in
- Romanov_InGaN_theta_nnp_2nd.in
- Romanov_InGaN_theta_nnp3.in
- Romanov_InGaN_theta_nnp3_2nd.in
Specify crystal orientation

nextnano software treats the rotation of crystal orientation by the **Miller-Bravais indices** in the input file. The setup of our system is as follows: the x-axis of the simulation coordinate system (hereafter \(x'\)-axis) is taken to the normal vector of the interface. The z-axis of the simulation system (\(z'\)) is normal to the (-1 2 -1 0) plane of the crystal, i.e. it is along \(a_2\) direction in Figure 13.1.16.1. The rotation axis indicated with red line is along \(z'\)-axis, and the interface is shown as the blue plane. The inclination angle \(\theta\) is defined as the angle between the c-axis (0001) and the normal vector of the blue plane, which is \(x'\)-axis.

Then the crystal orientation is specified in nextnano++ input file as

```plaintext
crystal_wz{
  x_hkl = [1, 0, l(\theta)] # x axis perpendicular to (hkl) plane = (hkil) plane
  z_hkl = [-1, 2, 0] # z axis perpendicular to (hkl) plane = (hkil) plane
}
```

where \(l(\theta)\) is an integer determined by the inclination angle. This statement means *the \(x'\)-axis is normal to the (1 0 -1 \(l(\theta)\)) plane of the crystal, whereas \(z'\)-axis is normal to the (-1 2 -1 0) plane.* (Note that nextnano++ does not require the third entry, i.e. the letter \(i\), in Miller-Bravais notation (hkil) because \(i=-(h+k)\).)

The index \(l(\theta)\) is deduced from a simple geometry consideration. Figure 13.1.16.2 shows the cross-section of a wurtzite lattice that is perpendicular to the rotation axis in Figure 13.1.16.1.

- When \(\theta = 0\), the interface is normal to the (0001) plane, i.e. \(x'\)-axis is normal to the (0001) plane.
- When \(\theta = 90\) degree, the \(x'\)-axis should be normal to the (1 0 -1 0) plane of the crystal.
- When \(0 < \theta < 90\) degree, definition of the index is \(l(\theta) := \frac{\theta}{\pi}\) and the following relation holds
Figure 13.1.16.2: Cross-section of the wurtzite lattice. The dashed blue line indicates the x'-direction, which is normal to the interface (solid blue line).

\[ d = \frac{\sqrt{3}}{2} a \tan \theta. \]

From these equations we find

\[ l(\theta) = \frac{2c}{\sqrt{3}a \tan \theta}. \]

The plane to be determined can be then taken as

\[(hkl) = (\sin \theta \ 0 \ -\sin \theta \ \frac{2c}{\sqrt{3}a} \ \cos \theta)\]

We note that the expression in the third case includes the other two special cases. To approximate the direction with integer entries, we multiply 100 and take the floor function:

\[
\text{$\gamma = \frac{c_{\text{InGaN}}}{a_{\text{InGaN}}}$ # c/a ratio
# ideal c/a ratio in wurtzite is SQRT(8/3)=1.63299
$h$ = (100, 0, 0)
$l$ = (100, 0, 0)
x_hkl = [$h, 0, l] # x axis perpendicular to (hkl) plane = (hkl) plane
\]

Since nextnano\(^3\) does not support variables in the Miller-Bravais indices, we explicitly give the indices in the input file using statements like:

![IF %ORIENTATION40 hkil-x-direction = 64 0 -64 143.](image)
Parameter sweep of the angle using Template: Sweep over the variable theta

- Input file: Romanov_InGaN_theta_nnp.in

One can make use of ‘Template’ feature of nextnanomat to sweep the angle $\theta$ and obtain crystal orientation dependence of several physical quantities. Here, calculation is performed for every 5 degrees.

We obtain the angle dependence using ‘Postprocessing’ feature. Here, we collect the strain tensor components $\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \varepsilon_{xy}, \varepsilon_{xz}$ and $\varepsilon_{yz}$ that are in columns 2, 3, 4, 5, 6, 7 of the file strain_simulation.dat.

- Select file containing values for the strain tensor components strain_simulation.dat by clicking on the folder icon below Postprocessing.
- Select 1 for the Maximum number of values lines.
- Select 2 for the Number of relevant columns. (To do: Improve nextnanomat to include all columns.)
- Click on Create file with combined data to generate file theta_strain_simulation_Column2.dat.
- Select 3 for the Number of relevant column.
- Click on Create file with combined data to generate file theta_strain_simulation_Column3.dat.
- Select 4 for the Number of relevant column.
- Click on Create file with combined data to generate file theta_strain_simulation_Column4.dat.
- Select 5 for the Number of relevant column.
- Click on Create file with combined data to generate file theta_strain_simulation_Column5.dat.
- Select 6 for the Number of relevant column.
- Click on Create file with combined data to generate file theta_strain_simulation_Column6.dat.
- The postprocessing results are contained in the folder <name_of_input_file>_postprocessing.
- Finally, the plotted results of the postprocessing file can be exported to gnuplot. Add all columns to the Overlay, and then click on: Create and Open Gnuplot (*.plt) from Items of Overlay.
Strain

Figure 13.1.16.3 and Figure 13.1.16.4 are the strain tensor elements as a function of inclination angle $\theta$, with respect to simulation and crystal coordinate systems, respectively. One can confirm that they reproduce correctly Figure 5 and 6 in [Romanov2006]. Please note that Romanov takes $z'$-axis as growth direction, while we take $x'$-axis. Therefore $x'$- and $z'$-axes are interchanged from [Romanov2006].

![Diagram of strain tensor components](image)

Figure 13.1.16.3: Elastic strain tensor components as a function of c-axis inclination angle $\theta$ in simulation coordinate system.

Piezoelectric effect (first-order)

The piezoelectric effect is at first instance described by a linear response against strain. In crystal coordinate system,

$$ P^{(1)}_\mu = \sum_{j=1}^{6} e_{\mu j} \epsilon_j, $$

where $\mu = 1, 2, 3$ and the strain tensor is expressed in six-dimensional Voigt notation

$$ \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{pmatrix} = \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{yz} \\ 2\epsilon_{xz} \\ 2\epsilon_{xy} \end{pmatrix}. $$

Please note that the indices $x, y, z$ without prime refer to the axes of the crystal coordinate system. The superscript $(1)$ indicates first-order piezoeffect. For the symmetry of the wurtzite structure, only three parameters remain in
Figure 13.1.16.4: Elastic strain tensor components as a function of c-axis inclination angle $\theta$ in crystal coordinate system.

The piezoelectric coefficient tensor $e_{ij}$

\[
\begin{pmatrix}
 p_x^{(1)} \\
p_y^{(1)} \\
p_z^{(1)}
\end{pmatrix} =
\begin{pmatrix}
 0 & 0 & 0 & e_{15} & 0 \\
 0 & 0 & 0 & e_{15} & 0 \\
e_{31} & e_{31} & e_{33} & 0 & 0
\end{pmatrix} =
\begin{pmatrix}
 \epsilon_{xx} \\
 \epsilon_{yy} \\
 \epsilon_{zz} \\
 2\epsilon_{yz} \\
 2\epsilon_{zx} \epsilon_{31}(\epsilon_{xx} + \epsilon_{yy}) + \epsilon_{33}\epsilon_{zz}
\end{pmatrix},
\]

cf. Eq. (4) in [Schulz2015]. Note that Eq. (14) in [Romanov2006] misses the factor 2 for off-diagonal elements of the strain tensor. These equations are implemented in both the nextnano++ and nextnano3 software with corresponding material parameters in the respective database. The following flags export the strain tensor components and piezoelectric polarization vector in crystal and simulation coordinate systems (cf. nextnano++ and nextnano3 documentation). The piezoelectric polarization vector with respect to the simulation coordinate system can be found in the file Strain\piezoelectric_polarization_vector_simulation.dat.
For consistency, we have used the same material parameters as [Romanov2006], i.e. we have overwritten our default material parameters of the database with the values specified in the input file.

Analytical expression is derived as follows [Schulz2015]. Since we are interested in the polarization normal to the interface, it is useful to switch to the simulation coordinate system \((x', y', z')\). This can be done by transforming the polarization vector and the strain tensor to the simulation system,

\[
P_{\mu'}^{(1)} = \left( R P^{(1)} \right)_{\mu'} = \sum_{\mu=1}^{3} R_{\mu'\mu} P_{\mu}^{(1)}, \quad \epsilon_{\mu'\nu'} = (R \epsilon R^{-1})_{\mu'\nu'} = \sum_{\mu, \nu=1}^{3} R_{\mu'\mu} R_{\nu'\nu} \epsilon_{\mu\nu},
\]

where the \(3 \times 3\) rotation matrix \(R\) accounts for a rotation of angle \(\theta\) and we have used the fact that the rotation matrix is orthogonal: \((R^{-1})_{\mu'\nu'} = R_{\mu\nu}\). Prime denotes the axes in simulation coordinate system. These equations can be expressed in vector form as

\[
\begin{pmatrix}
    p_{x'}^{(1)} \\
    p_{y'}^{(1)} \\
    p_{z'}^{(1)}
\end{pmatrix}
= R^{-1}(\theta) \begin{pmatrix}
    p_{x}^{(1)} \\
    p_{y}^{(1)} \\
    p_{z}^{(1)}
\end{pmatrix},
\quad
\begin{pmatrix}
    \epsilon_{xx} \\
    \epsilon_{yy} \\
    \epsilon_{zz}
\end{pmatrix}
= S^{-1}(\theta) \begin{pmatrix}
    \epsilon_{x'x'} \\
    \epsilon_{y'y'} \\
    \epsilon_{z'z'}
\end{pmatrix}
\]

where \(S(\theta)\) is a \(6 \times 6\) matrix. The second transformation is given in Eq. (13) in [Romanov2006]. From equations above, we obtain the first-order piezoelectric effect in the simulation coordinate system

\[
\begin{pmatrix}
    p_{x'}^{(1)} \\
    p_{y'}^{(1)} \\
    p_{z'}^{(1)}
\end{pmatrix}
= R(\theta) \begin{pmatrix}
    0 & 0 & e_{15} & 0 & 0 & 0 \\
    0 & 0 & 0 & e_{15} & 0 & 0 \\
    e_{31} & e_{31} & e_{33} & 0 & 0 & 0
\end{pmatrix}
S^{-1}(\theta) \begin{pmatrix}
    \epsilon_{x'x'} \\
    \epsilon_{y'y'} \\
    \epsilon_{z'z'}
\end{pmatrix}
\]

The \(z'\)-component is explicitly

\[
p_{z'}^{(1)} = e_{31} \cos \theta \epsilon_{x'x'} + \left( e_{31} \cos^2 \theta + \frac{e_{33} - 2e_{15}}{2} \sin \theta \sin 2\theta \right) \epsilon_{y'y'} + \left( e_{31} + 2e_{15} \right) \frac{\sin \theta \sin 2\theta + e_{33} \cos^2 \theta}{2} \epsilon_{z'z'} + \left( e_{31} - e_{33} \right) \cos \theta \sin 2\theta + 2e_{15} \sin \theta \cos 2\theta \epsilon_{y'z'}.
\]

Note that the corresponding analytical expression Eq. (18) in [Romanov2006] misses the factor 2 in front of \(e_{15}\) in the 2nd, 3rd and 4th line, and contains a typo in the 3rd line, i.e. \(e_{33}\) has to be \(e_{31}\) in the first term. Our expression is consistent to eq. (5) in [Schulz2015]. Figure 13.1.16.5 compares the results of the nextnano software with the results of [Romanov2006] and [Schulz2015], respectively. The analytical results in Figure 13.1.16.5 are the plot of the equation above, with an interchange of \(x'\)- and \(z'\)-axes.

From the results in Figure 13.1.16.5 we can see that the piezoelectric polarization vanishes at an intermediate angle around 38 degree and that it is maximized when the inclination angle is zero.
Figure 13.1.16.5: Piezoelectric polarization as a function of inclination angle. The gray dotted curve contains a typo $\varepsilon_{33} \leftrightarrow \varepsilon_{31}$ and misses the factor 2. When the first typo is fixed, the gray solid curve is obtained and looks to be consistent with Figure 7(a) in [Romanov2006]. With the factor 2 the result becomes the black curve. nextnano$^3$ and nextnano++ reproduce the black curve and are thus consistent to [Schulz2015].
Postprocessing for polarization

We obtain the angle dependence using ‘Postprocessing’ feature. Here, we collect the polarization components $P_z$ that is in column 1 of the file polarization_vector_piezoelectric_simulation.dat.

- Select file containing values for the piezoelectric components polarization_vector_piezoelectric_simulation.dat by clicking on the folder icon below Postprocessing.
- Select 2 for the Number of relevant column.
- Select 1 for the Maximum number of values lines.
- Click on Create file with combined data to generate file theta_polarization_vector_piezoelectric_simulation.dat.
- The postprocessing results are contained in the folder <name_of_input_file>_postprocessing.
- Finally, the plotted results of the postprocessing file can be exported to gnuplot. Add all columns to the Overlay, and then click on: Create and Open Gnuplot (*.plt) from Items of Overlay

Alloy content dependence

One can also sweep the alloy content $x$. The following results correspond to Figure 7(a) in [Romanov2006].

One can see that the zero point is universal for different alloy contents. The zero point is different compared to [Romanov2006] as he misses the factor of 2 for the strain tensor component. As can be seen in Figure 13.1.16.5 shown above, this mistake is not relevant for 0 and 90 degrees.

![In$_x$Ga$_{1-x}$N: Piezoelectric polarization](image)

Figure 13.1.16.6: Alloy content dependence of the piezoelectric polarization for In$_x$Ga$_{1-x}$N/GaN structure. In$_x$Ga$_{1-x}$N is under biaxial compressive strain with respect to GaN.
**AlGaN**

- Input file: Romanov_AlGaN_theta_nnp.in

Similarly, piezoelectric polarization of Al$_x$Ga$_{1-x}$N/GaN structure is calculated and shown in Figure 13.1.16.7. This result corresponds to Figure 8(a) in [Romanov2006]. The piezoelectric effect vanishes at around 38 degree in this case as well. Again, the zero point is different compared to [Romanov2006] as he misses the factor of 2 for the strain tensor component. As can be seen in Figure 13.1.16.5 shown above, this mistake is not relevant for 0 and 90 degrees.

![Al$_x$Ga$_{1-x}$N: Piezoelectric polarization](image)

Figure 13.1.16.7: Alloy content dependence of the piezoelectric polarization for Al$_x$Ga$_{1-x}$N/GaN structure. Al$_x$Ga$_{1-x}$N is under biaxial tensile strain with respect to GaN.

The sign of the piezoelectric polarization in Figure 13.1.16.7 is opposite to the case of InGaN/GaN composition (Figure 13.1.16.6). This is due to the fact that the lattice constants of InN, GaN and AlN obey the following relation

$$a_{InN} > a_{GaN} > a_{AlN}$$

(also for $c$). Since we take GaN as a substitute, In$_x$Ga$_{1-x}$N layer is subject to compressive strain, whereas Al$_x$Ga$_{1-x}$N is under tensile strain [Romanov2006].
Piezoelectric effect (second-order)

- Input file: *Romanov_InGaN_theta_nnp_2nd.in*

Optimization of optoelectronic device design requires an accurate and detailed knowledge of the growth-direction dependence of the built-in electric field. Recently, the second order piezoelectric effect has been reported to be relevant for wurtzite III-N materials, namely GaN, AlN and InN. This potentially affects the electronic and optical properties of the devices. The piezoelectric polarization is generalized in crystal coordinate as [Patra2017]

\[
P_{\mu}^{pz} = \sum_{j=1}^{6} \epsilon_{\mu j} \epsilon_{j} + \frac{1}{2} \sum_{j,k=1}^{6} B_{\mu j k} \epsilon_{j} \epsilon_{k} + \cdots,
\]

where \(\epsilon_{\mu j}\) and \(B_{\mu j k}\) are first- and second-order piezoelectric coefficients, respectively. For binary wurtzite structure, one can show that \(B_{\mu j k}\) has 8 independent components \(B_{311}, B_{312}, B_{313}, B_{333}, B_{115}, B_{125}, B_{135}, B_{344}\).

The explicit expression of the second-order term is given in Eq. (3) in [Patra2017], which is also implemented in nextnano++ and nextnano3.

One can turn on the second-order contribution in nextnano++ as

```plaintext
# nextnano++
strain{
  second_order_piezo = yes  # default: no
}
```

and in nextnano3,

```plaintext
! nextnano3
$numeric
  second_order = 2nd-order  ! [no/2nd-order]
$end_numeric
```

Figure 13.1.16.8 shows the results of the nextnano software. While the second-order contribution becomes negligible between the orientation (10\(-\)13) and (10\(-\)12), and also between 85 and 95 degrees, it enhancing the piezo effect up to 14% in other directions. This figure can be qualitatively compared to Figure 1(c) in [Patra2017], but note that they consider binary InN/GaN structure there while we are using In\(_{0.2}\)Ga\(_{0.8}\)N/GaN. The pink curve is different from the one in Figure 13.1.16.5 because we employed the material parameters used in [Patra2017]. nextnano3 also produces a consistent result (input file: *Romanov_InGaN_theta_nn3_2nd.in*). Within nextnano3 one can also use different formulae for the second-order effect, cf. nextnano3 documentation.

Please help us to improve our tutorial! Should you have any questions and comments, please send to support [at] nextnano.com.

### 13.1.17 GaAs Solar Cell

Here we demonstrate that solar cells can be simulated using nextnano. The self-consistent solutions to the Poisson equation coupled with current (drift-diffusion) equation give the figure of merit of solar cells that consists of arbitrary materials. Current-Voltage (I-V) curves and corresponding power and solar cell efficiency as a function of bias voltage are exported to the output folder.

**Note:** This tutorial is based on this one, which mainly focuses on nextnano3.
Second-order piezoelectricity. The second-order term enhances the piezoelectric polarization. The nextnano result (yellow) is consistent to the nextnano++ result (blue). Interface planes are indicated at corresponding angles.

### Input files

- 1DGaAs_SolarCell_nnp.in
- 1DGaAs_SolarCell_nnp_import_generation.in
- 1DGaAs_SolarCell_nn3.in
- 1DGaAs_SolarCell_nn3_import_generation.in

Here the numerics parameters are optimized for convergence of the calculation in the bias range of interest. Please pay attention to the convergence of the calculation when you change device geometry etc.

In the simulation of input files 1DGaAs_SolarCell_nnp.in and 1DGaAs_SolarCell_nn3.in, the following data are used to calculate generation rate $G(E, x)$ internally:

- Absorption coefficient $\alpha(E)$
- Reflectivity $R(E)$
- Solar spectral irradiance

In 1DGaAs_SolarCell_nnp.in (nextnano++), these data are already specified in database_optional.in for some materials. For example, you can use these by specifying irradiation{} as follows:

```plaintext
classical{
  
  irradiation{
    
    illumination{
      direction_x 1
    }

    database_spectrum{

    }

    
  }

  
}
```

(continues on next page)
If you want to use the materials that are not in the database or rewrite the database, you can specify the new data in `database()` as you want.

In `1DGaAs_SolarCell_nn3.in` (nextnano³), you need to prepare the above three data as external files to calculate generation rate $G(E, x)$ internally. We have the external data files, whose data are identical with those used in `1DGaAs_SolarCell_nnp.in`, for nextnano³. Please send an email to support [at] nextnano.com if you want them.

You can also import the data of generation rate itself. In the simulation of `1DGaAs_SolarCell_nnp_import_generation.in` and `1DGaAs_SolarCell_nn3_import_generation.in`, the following output file of nextnano³ must be read in.

- `/optics/GenerationRateLight_vs_Position_sun1.dat`

This data file is also in the sample file folder.

### Reference


### Structure

Figure 13.1.17.1 shows the bandedges and quasi Fermi levels of the device. The device structure is as follows:

- 0-30nm $\text{Al}_{0.8}\text{Ga}_{0.2}\text{As}$ Window layer
- 30-530nm p-doped GaAs
- 530-3530nm n-doped GaAs
- 3530-3630nm n-doped GaAs back surface field layer

Strain is not calculated in this example.

The left side of the device (x=0nm) is illuminated by the sun. As shown in Figure 13.1.17.6, mobile electrons and holes are created mainly in the p-layer. Electrons then flow to the right because of the AlGaAs ternary barrier.
Figure 13.1.17.1: Bandedges and quasi-Fermi levels of the solar cell at zero bias \textit{bias\_000000/bandedges.dat (nextnano++)} / \textit{band\_structure/BandEdges.dat (nextnano\textsuperscript{3})}

(0-30nm), and holes to the left. The back of the cell (3530-3630nm) is doped with 10 times larger concentration, so that it prevents the minority carrier (hole) from leaking to the right contact. Since the current from p-layer to n-layer is defined to be positive, the photo-induced current has negative sign.

**Simulation procedure**

The workflow of the simulation is summarized in Figure 13.1.17.2. To obtain the figures shown in this tutorial,

1. Specify in the input file the three data, namely (1) spectral irradiance (solar spectrum), (2) reflectivity at the front surface and (3) absorption coefficient. (Referring the database or rewriting the database)

2. Run the nextnano++ software, and all of your nextnano++ results are in your output folder! Generation rate \( G(E, x) \) is internally calculated before the current-Poisson iteration starts. The efficiency-voltage curve is generated as a final result.

3. **If you already have generation rate profile as a .dat file, you** can either import it into nextnano++ or in nextnano\textsuperscript{3}.
1. Solar spectrum

The sun emits light with a range of wavelengths ranging from the ultraviolet, visible to infrared region. The extraterrestrial solar spectrum resembles the spectrum of a black body at $T_{\text{sun}} = 5760K$ [Nelson Chapter 2]:

$$
\frac{2\pi \sin^2 \theta_{\text{sun}}}{\hbar c^2} \frac{E^2}{e^{E/k_B T_{\text{sun}}} - 1},
$$

where $E$ is the photon energy and $\theta_{\text{sun}} = 1.44 \times 10^{-3}[\text{rad}]$ when measured from the earth. The solar light travels from the sun to the earth, and then from the outer space to our solar cell devices, during which the spectrum attenuates and changes its shape. The standard solar spectrum assumed in solar cell analysis is called AM1.5G (AM = air mass), which takes into account the attenuation of the intensity and illumination from all angles (rather than direct from the sun) due to scattering in the atmosphere. The spectral photon flux, i.e. the spectrum of the number of incident photons per area per time, is denoted by $\phi(E)$ [$\text{m}^{-2}\text{s}^{-1}\text{eV}^{-1}$]. The spectral irradiance, namely the spectrum of the amount of energy supplied per area per time, is given by $L(E) = E\phi(E)$ with the unit of [$\text{Wm}^{-2}\text{eV}^{-1}$]. We have taken the AM1.5G spectral irradiance data from this website (Figure 13.1.17.3). If you have space applications in mind, please use the extraterrestrial spectrum, namely air mass zero (AM0).

The power of incident light

$$
P_{\text{sun}} = \int_0^\infty L(E) dE = 1000\ \text{Wm}^{-2},
$$

is solely determined by the condition of the sun and the atmosphere of the earth (for AM0 $P_{\text{sun}} = 1353\ \text{Wm}^{-2}$). The ultimate challenge of solar cell research is to achieve the most efficient conversion of this energy input into electric power $P_{\text{out}}$ [$\text{Wm}^{-2}$]. The figure of merit is therefore defined as $\eta = \frac{P_{\text{out}}}{P_{\text{sun}}}$.
nextnano Manual, Release latest

1. Generation rate (internal calculation)

(If you already have available data for generation rate, you can skip this section.)

When the sun light illuminates the device, some of the photons are reflected at the front surface (air-semiconductor interface) and the rest enters the device. This effect is taken into account by considering the reflectivity of \( \text{Al}_{0.8}\text{Ga}_{0.2}\text{As} \). Through the absorption of one photon, a pair of mobile electron and hole is created, while the photon flux attenuates exponentially with respect to the penetration depth. The generation rate thus depends not only on the incident photon flux \( \phi(E) \) but also on the absorption coefficient \( \alpha(E) \) of the material and the reflectivity \( R(E) \) at the surface (Figure 13.1.17.2):

\[
G(E, x) = N \phi(E) \cdot (1 - R(E)) \cdot \alpha(E) e^{-\alpha(E)x},
\]

where “the number of suns” \( N \) is multiplied to the photon flux \( \phi(E) \) to take into account the concentration of sunlight. The corresponding keyword is `concentration` (nextnano++) / `number-of-suns` (nextnano3).

In the sample input file for nextnano++, predefined value is used for \( L(\lambda) \). \( \alpha(\lambda) \) and \( R(\lambda) \) are defined in database{}. In keyword `classical{ irradiation{ illumination{ direction_x = 1

# nextnano++
classical{
  illumination{
    direction_x = 1

(continues on next page)
In the input file for nextnano3, $\alpha(\lambda)$, $R(\lambda)$, $L(\lambda)$ and $N$ are imported as specified in keyword $optical-absorption$.

If no reflectivity data is specified, perfect interface (zero reflection) is assumed. The reflectivity data of Al$_{0.8}$Ga$_{0.2}$As used in this simulation (optical_reflectivity in database_nnp_optional.in for nextnano++, Reflectivity_A10.80Ga0.20As.dat for nextnano3) have been generated through the Fresnel formula for perpendicular incident light

$$R(\lambda) = |r(\lambda)|^2 = \left| \frac{1 - [n(\lambda) + iK(\lambda)]}{1 + [n(\lambda) + iK(\lambda)]} \right|^2$$

where the refractive index $n$ and extinction coefficient $K$ of GaAs and AlAs are taken from here. To obtain the values of ternary Al$_{0.8}$Ga$_{0.2}$As, we performed linear interpolation.

If you consider a textured surface to reduce surface light reflection, please prepare the corresponding reflectivity data and import to the nextnano simulation. For nextnano++, $\alpha(E), R(E)$ and $L(E)$ are stored in the output folder Irradiation with file names absorption_spectrum_eV.dat, reflectivity_spectrum_eV.dat and illumination_spectrum_eV.dat, respectively. For nextnano3, they are stored in the output folder optics with file names AbsorptionCoefficient.dat, Reflectivity.dat and SolarSpectralIrradiance.dat, respectively.

The resulting generation rate is shown in Figure 13.1.17.4, Figure 13.1.17.5 and Figure 13.1.17.6.
Figure 13.1.17.4: Generation rate as a function of position and energy optics/GenerationRateLight_vs_Position_and_Energy_2Dplot_sun1.dat (nextnano³) in units of $10^{18}$ cm$^{-3}$ eV$^{-1}$ s$^{-1}$. The corresponding file for nextnano++ is Irradiation/photo_generation_energy_resolved.fld. This quantity is internally calculated using the absorption coefficient, reflectivity of the front surface and solar spectrum AM1.5G (Figure 13.1.17.3). Photons at around 3V are largely absorbed near the front surface due to a large absorption coefficient, which can be seen in the output optics/AbsorptionCoefficient_eV.dat/Irradiation/absorption_spectrum_eV.dat (not shown). Photons with lower energy, in contrast, travel a longer distance in the device.
Figure 13.1.17.5: Generation rate as a function of energy \textit{optics/GenerationRateLight_vs_Energy_sun1.dat} (nextnano\textsuperscript{3}), red curve). The corresponding file for nextnano++ is \textit{Irradiation/photo_generation_integrated.dat}. Also shown is the result for zero reflection at the front surface (blue curve). Obviously, the generation rate becomes larger when the reflection at the front surface is neglected. One can also clearly see, by comparing with Figure 13.1.17.4, that the low energy photons below the bandgap cannot contribute to the carrier generation. For this reason the bandgap of semiconductors affects the solar cell efficiency and is discussed in the context of the Shockley-Queisser efficiency limit.
Figure 13.1.17.6: Generation rate as a function of position $\text{optics/GenerationRateLight_vs_Position_sun1.dat}$ (nextnano$^{3}$). The corresponding file for (nextnano++) is $\text{Irradiation/photogeneration.dat}$. This data is obtained by integrating Figure 13.1.17.4 over energy $E$. When the photon flux travels through the device, the intensity diminishes exponentially, leading to the exponential decrease in generation rate. Most of the carrier generation occurs within 500 nm from the front surface, i.e. within the p-layer (30–530 nm).

1. Generation rate (import)

If the generation rate data $G(x) = \int G(E, x) dE$ (Figure 13.1.17.6) is available from literature or publications, you can import the .dat file without worrying about the above mentioned calculation. The data must contain position [nm] in the first column and generation rate $[10^{18}\text{cm}^{-3}\text{s}^{-1}]$ in the second. In the sample file $\text{1DGaAs_SolarCell_nnp_import_generation.in}$ (nextnano++) and $\text{1DGaAs_SolarCell_nn3_import_generation.in}$ (nextnano$^{3}$), we import the data generated from $\text{1DGaAs_SolarCell_nn3.in}$.

```plaintext
# nextnano++
structure{
    region{
        everywhere{
            generation{
                import{ import_from "GenImportProfile" }
            }
        }
    }
}

import{
    file{
        name "GenImportProfile"
        filename "..."
        format DAT
        scale 1e18 # import data is multiplied by this scaling factor,
    }
}
```

---

560 Chapter 13. nextnano++ Tutorials
4. Current-Voltage characteristics

The calculated or imported generation rate contributes to the right-hand side of the coupled current equations for electrons and holes,

\[-e \frac{\partial n}{\partial t} + \nabla \cdot j_n = -e(G - R),\]
\[e \frac{\partial p}{\partial t} + \nabla \cdot j_p = e(G - R),\]

where $G$ and $R$ are the (position-dependent) generation and recombination rates for electron-hole pairs. Here the charge current density $j_{n,p}$ has a dimension of (charge)(area)$^{-1}$(time)$^{-1}$ and the generation rate has (volume)$^{-1}$(time)$^{-1}$. The recombination rate is the sum of three different processes $R = R_{\text{rad}} + R_{\text{Auger}} + R_{\text{SRH}}$. See our Laser diode tutorial, [Nelson] or other literature for details. By solving this current equation and the Poisson equation self-consistently, the program obtains the current density at each bias step. The resulting I-V curve is shown in Figure 13.1.17.7 and Figure 13.1.17.8. For comparison, the dark current has been simulated by setting

The dark current in the present device behaves like in a diode under forward bias. When the sun illuminates the device, electrons and holes are created and current flows in the reverse direction.

If you change the device geometry or materials and the I-V curve is no longer reasonable, it is likely that the numerical calculation did not converge. Please check the .log file. For the convergence of the current-Poisson equation, you might need to change the settings under run{} (nextnano++) or $\texttt{numeric-control}$ (nextnano3) keywords. If you need help, feel free to contact us at support [at] nextnano.com.
Figure 13.1.17.7: I-V characteristics of the solar cell *IV_characteristics.dat* (*nextnano++*). In the bias regime 0-1 V the system works as a solar cell.

Figure 13.1.17.8: I-V characteristics of the solar cell *currentIV_characteristics_new.dat* (*nextnano*³). In the bias regime 0-1 V the system works as a solar cell.
5. Solar efficiency

From the I-V curve the solar cell power density $P_{out} = -IV$ and the efficiency $\eta = \frac{P_{out}}{P_{in}}$ are calculated. For the present device under 1 sun, the maximum efficiency of 15.8% (nextnano++) / 17.0% (nextnano3) is achieved at the bias 0.9 V (Figure 13.1.17.9, Figure 13.1.17.10 red). The theoretical limit for GaAs (bandgap 1.42 eV at $T = 300 \, K$) is around 30% under the AM1.5 condition without concentration [Sze].

![Solar cell efficiency graph](image)

Figure 13.1.17.9: Solar cell efficiency $\eta$ for no sunlight concentration (red) and 100-sun concentration (blue) by nextnano++. The data is contained in solar_cell_efficiency.dat.

The maximum efficiency of the present device increases to 21.6% (nextnano++) / 22.3% (nextnano3) for 100-sun concentration, mainly due to the increase in open circuit voltage (Figure 13.1.17.9, Figure 13.1.17.10 blue). This means one cell operating under 100 suns can produce the same power output as $100P_{max} \times 0.216 = 137$ cells under 1 sun. Optical concentration reduces the total cost of solar cells since concentrator materials are usually less expensive than the ones for solar cells [Sze].

The .log file and the file solar_cell_info.txt (nextnano3) contain additional properties of the solar cell.
Figure 13.1.17.10: Solar cell efficiency $\eta$ for no sunlight concentration (red) and 100-sun concentration (blue) by nextnano$^3$. The data is contained in current/solar_cell_efficiency.dat.

The convergence of the simulation is sensitive to the device settings such as the number of suns. If the convergence fails in your original device, please consider changing the settings in run{} (nextnano++) or $\$\text{numeric-control}$ (nextnano$^3$).

- Please help us to improve our tutorial. Should you have any questions or comments, please send them to support [at] nextnano.com.

13.1.18 Multiple quantum wells and finite superlattices

Author: Brandon Loke

This tutorial simulates a real layered structure with a finite number of quantum wells. The transition between a finite superlattice and a multiple quantum well system is also observed. This tutorial aims to reproduce the figures in Paul Harrison’s book “Quantum Wells, Wires and Dots” (Section 3.10, “Multiple Quantum Wells and Finite Superlattices”)

The inputfile used for this tutorial is

- Superlattice_N Wells_mnp.in

The corresponding Jupyter Notebook for this tutorial can be found over here: MQW_Superlattices.ipynb.
Structure

The structure consists of $N$ repeats of 4nm GaAs wells and 4nm Ga$_{0.8}$Al$_{0.2}$As quantum wells. This superlattice structure is sandwiched between 20nm Ga$_{0.8}$Al$_{0.2}$As barriers.

We first define key variables, such as the well width, the right and left wall width, and the number of wells.

```plaintext
# Global constants
$TEMPERATURE = 300  # Temperature

# Structure
$WELL_WIDTH = 10.0  # Width of the quantum well
$BARRIER_WIDTH = 10.0  # Width of the barrier
$NUMBER_OF_WELLS = 4  # number of quantum wells
$SUPERLATTICE_WIDTH = $NUMBER_OF_WELLS * ($BARRIER_WIDTH + $WELL_WIDTH) - $BARRIER_WIDTH

# Materials and doping
$ALLOY_X = 0.8
```

Following this, we are able to generate the structure of the GaAs/Ga$_{0.8}$Al$_{0.2}$As superlattice under `structure{}`. The keywords `array_x{}` duplicate the structure in the x-direction to give us the number of wells required.

```plaintext
region{  # LEFT WALL
    line{  x [[$LEFT_BARRIER_WIDTH, 0]]
    }
    ternary_constant{
        name "Ga$_{x}$Al$_{1-x}$As"  # Ga0.8Al0.2As
        alloy_x $ALLOY_X
    }
}

# SUPERLATTICE

region{  # Barrier
    array_x{
        shift $BARRIER_WIDTH $WELL_WIDTH
        max $NUMBER_OF_WELLS 1
    }
    line{  x [0, $BARRIER_WIDTH]
    }
    binary{
```
name "GaAs"
}

region# Quantum well
array_x{
  shift $BARRIER_WIDTH $WELL_WIDTH
  max $NUMBER_OF_WELLS
}
line{
  x [$BARRIER_WIDTH, $BARRIER_WIDTH + $WELL_WIDTH]
}
ternary_constant{
  name "Ga$(x)Al(1-x)As"
  alloy_x $ALLOY_X
}

region# RIGHT WALL
line{
  x [$SUPERLATTICE_WIDTH, $SUPERLATTICE_WIDTH + $RIGHT_BARRIER_WIDTH]
}
ternary_constant{
  name "Ga$(x)Al(1-x)As"
  alloy_x $ALLOY_X
}

Simulation Settings

Under quantum{}, we specify

quantum {
  region{
    name "quantum_region"
    x [$LEFT_BARRIER_WIDTH, $SUPERLATTICE_WIDTH]
    $RIGHT_BARRIER_WIDTH ]
    # Schroedinger equation is solved only in region of LEFT WALL + SUPERLATTICE + RIGHT WALL
    boundary{
      # x = dirichlet # Dirichlet boundary
      x $LEFT_BARRIER_WIDTH $SUPERLATTICE_WIDTH $RIGHT_BARRIER_WIDTH ]
      # condition for the Schroedinger equation, psi = 0
      x $SUPERLATTICE_WIDTH $RIGHT_BARRIER_WIDTH ]
      # Neumann boundary
      # condition for the Schroedinger equation, dpsi/dx = 0
    }
    Gamma{
      num_ev 70
      # 70 eigenvalues have to be calculated
    }
    HH{
      num_ev 250
      # 150 eigenvalues have to be calculated
    }
    LH{
      num_ev 70
      # 70 eigenvalues have to be calculated
    }
    SO{
We want to obtain the energies and the amplitudes of the wavefunctions outputted.

**Ground state energies**

After generating the input file, we are able to run the simulation for a variable number of quantum wells using the variable sweep functionality in nextnanomat. One can go to “Template” on the tabs at the top, under “Sweep”, select the variable of interest and the range or list of values to iterate over. Click on “Create input file” at the bottom and run the simulations in the “Simulation” tab.

The following graphs were generated with *nextnanopy*. The reference potential energy used in Harrison’s book and nextnano++ is different. Thus, postprocessing was done in Python to match the reference energy levels.

![Ground state energies](image)

Figure 13.1.18.1: Ground state energies plotted as a function of $N$. Convergence at higher number of wells is observed.
Wavefunction in a superlattice

The wavefunctions can also be plotted. The first example in Harrison’s book has the following parameters:

- 10 wells
- 4nm Ga$_{0.8}$Al$_{0.2}$As barrier
- 4nm Ga$_{0.8}$Al$_{0.2}$As quantum well width
- 20nm left and right Ga$_{0.8}$Al$_{0.2}$As walls

![Finite Superlattice](image)

Figure 13.1.18.2: The wavefunction for a superlattice system

This figure is in agreement with Harrison’s result. It is observed that the system functions as a superlattice as the wavefunction in each well overlaps with the wavefunction of the adjacent wells.

Wavefunction in a multiple quantum well system

Harrison’s final figure uses the following parameters:

- 4 wells
- 10nm Ga$_{0.6}$Al$_{0.4}$As barriers
- 10nm Ga$_{0.6}$Al$_{0.4}$As quantum wells
- 10nm Ga$_{0.6}$Al$_{0.4}$As left and right walls

This figure is also in good agreement with Harrison’s results. It is observed from the figure that this system functions as a multiple quantum well because the wavefunction reaches zero between the wells.

Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.
Figure 13.1.18.3: The wavefunction for a multiple quantum well system
13.1.19 InAs / In$_{0.4}$Ga$_{0.6}$Sb superlattice dispersion with 8-band k.p (type-II band alignment)

Authors: Stefan Birner, Michael Povolotskyi

The input files used this tutorial are:

- IDInAs_InGaSb_k_zero_nnp.in
- IDInAs_InGaSb_k_parallel_k_superlattice_nnp.in
- IDInAs_InGaSb_k_parallel_k_superlattice_nnp.in
- IDInAs_InGaSb_SL_k_parallel_k_superlattice_nnp.in

This tutorial aims to reproduce Fig. 2(a) of “Long wavelength InAs/InGaSb infrared detectors: Optimization of carrier lifetimes” by Grein and Young.

Conduction and valence band edges

The input file used is IDInAs_InGaSb_k_zero_nnp.in. The heterostructure is a superlattice with 3.98 nm InAs and 1.5 nm In$_{0.4}$Ga$_{0.6}$Sb, where both constituents are strained with respect to the GaSb substrate.

The structure has a type-II band alignment, i.e. the electrons are confined in the InAs layer, whereas the holes are confined in the In$_{0.4}$Ga$_{0.6}$Sb layer.

The In$_{0.4}$Ga$_{0.6}$Sb layer is strained pseudomorphically with respect to the GaSb substrate, leading to a compressive strain (-2.5%) which splits the degeneracy of the heavy and light hole band edges in this layer. Thus, the heavy hole band edge lies above the light hole band edge.

The InAs layer is also strained pseudomorphically with respect to the GaSb substrate, and is thus under slight biaxial tension (+0.6 %). The splitting of the hole band edges is the opposite as in InGaSb, i.e. the light hole band edge is above the heavy hole band edge.

The following figure shows the electron and hole band edges.

Note that the origin of the energy scale is set to the GaSb valence band edge energy.
Electron and hole wavefunction for $k_{||} = 0$

We simulate one period only (i.e. from 0 nm to 5.48 nm) and solve the Schrödinger equation with periodic boundary conditions to mimic an infinite superlattice.

The following figure shows the conduction band edge and the heavy, light and split-off hole valence band edges in this superlattice structure together with the electron ($c_1$), heavy hole (hh1) and light hole (lh1) energies and wave functions ($\psi^2$), calculated within 8-band $k\cdot p$ theory.

One can clearly see that the electron state ($c_1$) is confined in the InAs layer (right part of the figure), whereas the heavy (hh1) and light hole (lh1) states are confined in the In$_{0.4}$Ga$_{0.6}$Sb layer (left part of the figure).

![Figure showing electron and hole wavefunctions](image)

We used the same material parameters as given in the above cited paper by Grein et al., apart from the $k\cdot p$ parameters.

Electron and hole energies for $k_{||} \neq 0$

The inputfile 1DInAs_InGaSb_k_parallel_nnp.in is used. The following figure shows the E($k_{||}$) dispersion of the electron ground state and the two highest hole states along two different directions in ($k_x, k_y$) space.

![Figure showing electron and hole dispersions](image)

This data is contained in this file: Schroedinger_kp/par1D_disp_01_00_11_hl_8x8kp_ev_min001_ev_max010.dat. Note that the band gap is not determined by the bandgap of one individual layer. It is determined by the electron ground state in the InAs layer, and the hole ground state in the InGaSb layer. This means more freedom for band gap engineering.
**Electron and hole energies for \( k_z \neq 0 \)**

The input file used is `1DInAs_InGaSb_k_superlattice_nnp.in`

The right part of the following figure shows the \( E(k_z) \) superlattice dispersion of the electron ground state and the two highest hole states. \( k_z \) is the superlattice vector between 0 and 1 \( \frac{2\pi}{L} \) where \( L = 5.48 \text{ nm} \) is the length of one superlattice period. \( (1 \frac{2\pi}{L} = 0.05731/\text{Å}) \)

This data is contained in this file: `Schroedinger_kp/8x8kp_dispSL_hl_qc001_evmin001_evmax016.dat`

The left part of the figure shows the \( E(k_||) \) dispersion along [10], i.e. from \((k_z, k_y) = (0,0)\) to \((k_z, k_y) = (-0.1,0)\) which is shown in the figure above already.

![Graph](image)

One can clearly see that these heterostructure bands are highly nonparabolic.

**3D plot**

Using `1DInAs_InGaSb_SL_k_parallel_superlattice.in`, one can generate a 3D plot of the energy dispersion \( E(k_x, k_y, k_{SL}) \) for each eigenvalue. The files are called `dispersion_k_parallel_k_SL_ev001.fld` where `ev001` indicates eigenvalue number 1.

*Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.*

### 13.1.20 InAs / GaSb broken gap quantum well (BGQW) (type-II band alignment)

**Author:** Stefan Birner

**Input files required:**

- `1DInAs_GaSb_BGQW_k_zero_nnp.in`
- `1DInAs_GaSb_BGQW_k_parallel_nnp.in`
- `1DInAs_GaSb_BGQW_k_parallel_nnp_01.in`
- `1DInAs_GaSb_BGQW_k_parallel_nnp_11.in`

This tutorial aims to reproduce Figs. 1, 2(a), 2(b) and 3 of *Hybridization of electron, light-hole, and heavy-hole states in InAs/GaSb quantum wells*

Material parameters used are taken from *Optical transitions in broken gap heterostructures*.

The heterostructure is a **broken gap quantum well (BGQW)** with 15 nm InAs and 10 nm GaSb, sandwiched between two 10 nm AlSb layers. Note that this heterostructure is asymmetric.

To be consistent with the above cited papers, strain is not included into the calculations although this would be possible. The structure has a **type-II** band alignment, i.e. the electrons are confined in the InAs layer, whereas the holes are confined in the GaSb layer. Depending on the width of the InAs and/or GaSb layers, things can be even
more complicated because the hole states can hybridize with the electron states, making it difficult to distinguish between electron-like and hole-like states. Another difficulty arises because the lowest electron states might be located below the highest hole states. This requires a new algorithm to occupy the states according to a suitable Fermi level.

The following figure shows the electron and hole band edges of the BGQW structure.

- \texttt{band\_structure/cb1D\_001.dat} (Gamma conduction band edge) in units of [eV]
- \texttt{band\_structure/vb1D\_001.dat} (heavy hole valence band edge) in units of [eV]
- \texttt{band\_structure/vb1D\_002.dat} (light hole valence band edge) in units of [eV]
- \texttt{band\_structure/vb1D\_003.dat} (split-off hole valence band edge) in units of [eV]

The origin of the energy scale is set to the InAs conduction band edge energy. The heavy hole and light hole band edges are degenerate because we neglect the effects of strain to be consistent with the above cited papers.

**Results**

The input file used here is 1DInAs_GaSb_BGQW\_k\_zero\_nnp.in. The following figure shows the conduction band edge and the heavy/light hole valence band edges in this BGQW structure together with the electron (e1, e2), heavy hole (hh1, hh2, hh3) and light hole (lh1) energies and wave functions ($\psi^2$), calculated within 8-band $k$,$p$ theory at the zone center, i.e. at $k_{||}$: $= 0$.

One can clearly see that the electron state (e1, e2) are confined in the InAs layer (left part of the figure), whereas the heavy (hh1, hh2, hh3) and light hole (lh1) states are confined in the GaSb layer (right part of the figure). One can see a slight hybridization of the e1 and lh1 states, i.e. these states are mixed states whereas the heavy hole states (hh1, h2, hh3) are not mixed and thus confined in the GaSb layer.

We use the data files

- \texttt{Schroedinger\_kp/kp\_8x8psi\_squared\_qc001\_el\_kpar0001\_1D\_dir.dat}, which contains $\psi^2$
- \texttt{Schroedinger\_kp/kp\_8x8psi\_squared\_qc001\_el\_kpar0001\_1D\_dir\_shift.dat}, which contains $\psi^2 + E_i$

The latter file contains the square of the wave functions (for par0001, i.e $k_{||} = 0$, i.e. $k_x = k_y = 0$), shifted by their energies, so that one can nicely plot the conduction and valence band edges together with the square of the wave functions.
The energies of the eigenstates are in units of [eV] and are contained in the file `Schroedinger_kp/kp_8x8eigenvalues_qc001_el_kpar0001_1D_dir.dat`

The input file `1DInAs_GaSb_BGQW_k_parallel.in` was used for the following results. The following figure shows the $E(k_{\parallel})$ dispersion of the electron and hole states along the [10] direction and along the [11] direction in ($k_x, k_y$) space. The [01] direction has the same dispersion due to symmetry arguments.

In this input file, the energy levels and wave functions for 24 $k_{\parallel}$ points along a line from ($k_x, k_y$) = (0, 0) to ($k_x, k_y$) = (0, $k_y$) have been calculated.

`Schroedinger_kp/kpar1D_disp_01_00el_8x8kp_ev_min001_ev_max020.dat` contains the $k_{\parallel}$ dispersion from [00] to [01] because in the input file, it is specified that

```
dispersion{
  path{
    name "kpar_01_00_10"
    point{ k [0.0, 0.0, 1.0] }
    point{ k [0.0, 0.0, 0.0] }
    point{ k [0.0, 1.0, 0.0] }
    spacing $number_k_parallel
  }
  path{
    name "kpar_10_00_11"
    point{ k [0.0, 1.0, 0.0] }
    point{ k [0.0, 0.0, 0.0] }
    point{ k [0.0, 1.0, 1.0] }
    spacing $number_k_parallel
  }

  output_dispersions{}
  output_masses{}
}
```

The first column contains the $k_{\parallel}$ value, the other columns contain the eigenvalues for each $k_{\parallel}$ value: $E_n(k_{\parallel}) = E_n(k_x, k_y) = E_n(0, k_y)$. Here, $n = 1, \ldots, 20$. ($\ldots$ ev_min 001** ev_max **020$\ldots$) Note that for this particular example, the eigenvalues have to be sorted manually if you want to connect the energy values, i.e. to include lines (“lines are a guide to the eye”).

The black lines are the results of the `nextnano++` software, the red dots are the results of the `nextnano`3 software.

![Subband dispersion along [10] and [11] directions](image)

At an in-plane wave vector of 0.014 1/Å, strong intermixing between the e1 and the lh1 states occurs. In contrast to the wave functions at $k_{\parallel} = 0$, where the e1 and lh1 wave functions are nearly purely electron- or hole like, the wave functions at $k_{\parallel} = (0, 0.014) = (0.014, 0)$ are a mixture of electron and light hole wave functions. Compare with Fig. 4 of the A. Zakharova et al.

In asymmetric quantum wells, the double spin degeneracy is lifted at finite values of $k_{\parallel}$ because of spin-orbit interaction. This is the reason why we have two different dispersions $E(k_{\parallel})$ for “spin up” and “spin down” states. This also means that the wave functions at finite $k_{\parallel}$ are different for “spin up” and “spin down” states.
The file `Schroedinger_kp/kp_8x8k_parallel_qc001_el1D_dir.dat` tells us which number of \( \mathbf{k}_|| \) vector corresponds to \((k_x, k_y)\).

| \( \mathbf{k}_{\parallel}\) number | \( k_x [1/\text{nm}] \) | \( k_y [1/\text{nm}] \) | \( \rightarrow \) | \( \mathbf{k}_|| = (k_x,k_y) = (0,0) [1/\text{nm}] \) |
|-----------------------------|-----------------|-----------------|----------------|------------------|
| 1                           | 0.000000E+000   | 0.000000E+000   | \( \rightarrow \) | \( \mathbf{k}_|| = (k_x,k_y) = (0,0) [1/\text{nm}] \) |
| 29                          | 0.000000E+000   | 1.400000E+000   | \( \rightarrow \) | \( \mathbf{k}_|| = (k_x,k_y) = (0,0.14) [1/\text{nm}] \) |
| 1326                        | 1.000000E+000   | 1.000000E+000   | \( \rightarrow \) | \( \mathbf{k}_|| = (k_x,k_y) = (1.0,1.0) [1/\text{nm}] \) |

In the following figure, we plot the square of the wave functions for \( \mathbf{k}_|| = (0,0.14) \text{ nm}^{-1} \). The corresponding label of our \( \mathbf{k}_|| \) numbering is 29. Note that this labeling depends on the \( \mathbf{k}_|| \) space resolution, i.e. the number of \( \mathbf{k}_|| \) points that have been specified in the input file: num-kp-parallel = 10000.

The wavefunctions \( (\psi^2 + E) \) are contained in the file `Schroedinger_kp/kp_8x8psi_squared_qc001_hl_kpar00029_1D_dir_shift.dat`.

![Wavefunction plot](image)

The electron states (\( e_1 \)) couple strongly with the light hole states (lh1). This is expected from the energy dispersion plot because at 0.14 nm\(^{-1}\) a strong anticrossing is present for these states. One can also clearly see that for spin up and spin down states, different energy levels and different probability densities exist. This is in contrast to the states at \( k_|| = 0 \) which are two-fold spin degenerate as shown in the figure further above. Our results are similar to Fig. 4 of Zakharova’s paper.

*Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.*

### 13.1.21 Interband tunneling current in a highly-doped nitride heterojunction

Author: Takuma Sato

**Introduction**

We compute interband tunneling current through a highly-doped heterojunction by nextnano++ simulation and Python postprocessing. We follow the methods in the following publication, using less approximations wherever possible:

*Jean-Yves Duboz and Borge Vinter. Theoretical estimation of tunnel currents in hetero-junctions: The special case of nitride tunnel junctions*

This tutorial uses the Python script in the `nextnanopy` repository

- `nextnanopy/templates/InterbandTunneling_Duboz2019_nnp.py`

  to automate the simulation of the nextnano++ input file (in the sample input file folder of nextnano installation)

- `InterbandTunneling_Duboz2019_nnp.in`

  and post-calculation of interband tunneling current.

For the details of the formulation and the algorithm in the Python script, see

- `nextnanopy/templates/InterbandTunneling_Duboz2019_formulation.pdf`
Results

The structure is an AlGaN/GaN p-i-n junction with 2 nm GaN interlayer.

The energy overlap between the hole states and electron states increases as the bias, leading to larger tunnel current.

The Python script calculates dipole matrix elements from the simulation results:
from which we obtain the tunnel current as a function of bias:

Please help us to improve our tutorial. Should you have any questions or comments, create a ticket here.

13.2 2D Tutorials

13.2.1 2D MOSFET

This tutorial is described here.

13.2.2 Electron wave functions in a cylindrical well (2D Quantum Corral)

In this tutorial we demonstrate 2D simulation of a cylindrical quantum well. We will see the electron eigenstates and their degeneracy.

Input files used in this tutorial are the followings:

- 2DQuantumCorral_n3.in / *_nnp.in
Bandedge diagram at bias -0.7V

Position-dependent material parameters

Dipole matrix element $<Z|z|S>$
Figure 13.1.21.1: Interband tunneling current in a nitride p-i-n junction. Following the paper, backward bias is taken to be positive in this plot.
**Structure**

- A cylindrical InAs quantum well (diameter 80 nm) is surrounded by a cylindrical GaAs barrier (20 nm) which is surrounded by air. The whole sample is 160 nm x 160 nm.
- We assume infinite GaAs barriers. This can be achieved by a circular quantum cluster with Dirichlet boundary conditions, i.e. the wave function is forced to be zero in the GaAs barrier.
- The electron mass of InAs is assumed to be isotropic and parabolic ($m_e = 0.026m_0$).
- Strain is not taken into account.
Simulation outcome

Electron wave functions

The size of the quantum cluster is a circle of diameter $2a = 80$ nm.

The following figures shows the square of the electron wave functions (i.e. $\psi^2$) of the corresponding eigenstates. They were calculated within the effective-mass approximation (single-band) on a rectangular finite-differences grid.

- 1st eigenstate, $(n, l) = (1, 0)$

- 2nd eigenstate, $(n, l) = (1, 1)$

- 3rd eigenstate, $(n, l) = (1, -1)$
• 4th eigenstate, \((n, l) = (1, 2)\)

• 5th eigenstate, \((n, l) = (1, -2)\)

• 6th eigenstate, \((n, l) = (2, 0)\)

• 15th eigenstate, \((n, l) = (3, 0)\)

• 20th eigenstate, \((n, l) = (1, 6)\)

• 22th eigenstate, \((n, l) = (3, 1)\)
The parameters of the quantum corral are the followings:

- radius: $a = 40$ nm
- $m_e = 0.026m_0$
- $V(r) = 0$ for $r < a$
- $V(r) = \infty$ for $r > a$

The analytical solution of the eigenstates of this quantum well is:

$$\psi_{n,l}(r, \theta) \propto J_l \left( \frac{j_{l,n} r}{a} \right) \left[ A \cos(l \theta) + B \sin(l \theta) \right]$$  \hspace{1cm} (13.2.2.1)

where

- $J_l(x)$ is the Bessel function of the first kind (We cite them for $l = 0, 1, 2$ below.)
- $j_{l,n}$ is its zero point i.e. $J_l(j_{l,n}) = 0$ and $n = 1, 2, ...
- A, B are constant
- $l = 0, \pm 1, \pm 2, ...

The corresponding eigenenergies are:

$$E_{nl} = \frac{\hbar^2 j_{l,n}^2}{2m_e a^2}$$

The Quantum number $n$ comes from the boundary condition $\psi(a, \theta) = 0$. The requirement that $\psi$ has the same value at $\theta = 0$ and $2\pi$ leads to the quantum number $l$. In the above figures of the eigenstates, we can know them through the following relations:

- (the number of zero points in the radial direction) = $n$
- (the number of zero points in the circumferential direction)/2 = $|l|$

### Energy spectrum

The following figure shows the energy spectrum of the quantum corral. (The zero of energy corresponds to the InAs conduction band edge.)

The two-fold degeneracies of the states

- $(2, 3), (4, 5), (7, 8), (9, 10), (11, 12), (13, 14), (16, 17), (18, 19), (20, 21), (22, 23), (24, 25), (26, 27), (28, 29), (31, 32), (33, 34), (35, 36), (37, 38), (39, 40)$

corresponds to $|l| \geq 1$. On the other hand, the non-degenerate energy eigenvalues corresponds to $l = 0$

The analytical energy values are: $E_{nl} = \frac{\hbar^2 j_{l,n}^2}{2m_e a^2}$.

There is a formula to approximate $j_{l,n}$: $j_{l,n} = (n + \frac{1}{2}|l| - \frac{1}{4})\pi$ which is accurate as $n \to \infty$.

Here we describe the comparison between the analytical values, approximate values, nextnano++ results and nextnano3 results.
Figure 13.2.2.1: Bessel functions of the first kind for $l = 0, 1, 2$ generated by scipy.
Further details about the analytical solution of the cylindrical quantum well with infinite barriers can be found in:

The Physics of Low-Dimensional Semiconductors - An Introduction
John H. Davies

Please help us to improve our tutorial. Send comments to support [at] nextnano.com.

13.2.3 2D Optics: Intersubband absorption spectrum of a GaAs cylindrical quantum wire

Section author: Naoki Mitsui

This tutorial calculates the optical absorption spectrum of a GaAs cylindrical quantum wire with infinite barriers. We will see which output file we should refer to in order to understand the absorption spectrum.

Also, the formula used for the absorption calculation is presented. For the detailed scheme of the calculation of the optical matrix elements or absorption spectrum, please see our 1D optics tutorial: 1D Optics: Optical absorption for interband and intersubband transitions

|   | $|n, l|$ | $j_{l,n}$ | $E_{n,l}$ (approx.) [eV] | $E_{n,l}$ (nextnano++) [eV] | $E_{n,l}$ (nextnano³) [eV] |
|---|---|---|---|---|---|
| 1st | [1, 0] | 2.405 | 0.75π ≈ 2.356 | 0.00530 | 0.00508 | 0.00511 |
| 2nd | [1, 1] | 3.832 | 1.25π ≈ 3.926 | 0.01345 | 0.01412 | 0.01094 |
| 3rd | [1, -1] | 3.832 | 1.25π ≈ 3.926 | 0.01345 | 0.01412 | 0.01094 |
| 4th | [1, 2] | 5.136 | 1.75π ≈ 5.497 | 0.02416 | 0.02768 | 0.02320 |
| 5th | [1, -2] | 5.136 | 1.75π ≈ 5.497 | 0.02416 | 0.02768 | 0.02329 |
| 6th | [2, 0] | 5.520 | 1.75π ≈ 5.497 | 0.02791 | 0.02767 | 0.02685 |
| 7th | [2, 1] | 7.016 | 2.25π ≈ 7.067 | 0.04508 | 0.04574 | 0.03584 |

• Structure
• Scheme
• Results
  - Absorption
  - Eigenvalues, transition energies, and occupations
  - Transition intensity (Momentum matrix element)
  - Eigenstates
Input file:
- 2Dcircular_infinite_wire_GaAs_intra_nnp.in

**Structure**

![Graph showing Gamma bandedge and slice of the bandedge along x = 0.]

Figure 13.2.3.1: Left: Conduction bandedge for a cylindrical quantum wire. Right: Slice of the bandedge along x = 0.

The above figures show the Gamma bandedge of the circular GaAs region and the barrier region. We model the infinite barrier by assigning 100 eV for the bandedge of AlAs barrier region from database section. Please see the input file for the details.

The parameters used in this simulation are as follows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>Value [unit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantum wire radius</td>
<td>$R$</td>
<td>5 [nm]</td>
</tr>
<tr>
<td>barrier height</td>
<td>$E_b$</td>
<td>92 [eV]</td>
</tr>
<tr>
<td>effective electron mass</td>
<td>$m_e$</td>
<td>0.0665</td>
</tr>
<tr>
<td>refractive index</td>
<td>$n_r$</td>
<td>3.3</td>
</tr>
<tr>
<td>doping concentration (n-type)</td>
<td>$N_D$</td>
<td>$5 \cdot 10^{18}$ [cm$^{-3}$]</td>
</tr>
<tr>
<td>linewidth (FWHM)</td>
<td>$\Gamma$</td>
<td>0.01 [eV]</td>
</tr>
<tr>
<td>temperature</td>
<td>$T$</td>
<td>300 [K]</td>
</tr>
</tbody>
</table>
## Scheme

The `run()` section is specified as follows:

```plaintext
tenextnano Manual, Release latest

Scheme

The \texttt{run()} section is specified as follows:

```plaintext
run{
    poisson{}
    quantum{}
    optics{}
}
```

Then the simulation follows these steps:

1. Poisson equation is solved with the setting specified in the \texttt{poisson{}} section.
2. “Schrödinger” equation is solved with the setting specified in the \texttt{quantum{}} section.
3. “Schrödinger” equation is solved again with the setting specified in the \texttt{optics{}} section and optical properties are calculated.

\textbf{Note:}

- If \texttt{quantum_poisson{}} is specified instead of \texttt{quantum{}}, Poisson and Schrödinger equations are solved self-consistently.
- \texttt{optics{}} requires that kp8 model is used in the quantum region specified in \texttt{quantum{}}.
- In this tutorial the kp parameters are adjusted so that the conduction and valence bands are decoupled from each other. Thus the single-band Schrödinger equations are solved effectively by the kp solver.

The optical absorption accompanied by the excitation of charge carriers (state $n \rightarrow m$) in a condensed matter is calculated on the basis of Fermi’s golden rule \cite{ChuangOpto1995} in the dimension of (length)$^{-1}$:

\[
\alpha(\varepsilon, \omega) = \frac{\pi \varepsilon^2}{n_s \epsilon_0^2 m_0^2 c^2} \frac{1}{V} \sum_{n \geq m \rightarrow k_z} |\bar{\varepsilon} \cdot \bar{\pi}_{nm}(k_z)|^2 (f_m(k_z) - f_n(k_z)) \mathcal{L}(E_n(k_z) - E_m(k_z) - \hbar \omega),
\]  

(13.2.3.1)

where

- \(k_z\) is the Bloch wave vector along translation-invariant directions. In 2D simulation this is 1D vector.
- \(E_n(k_z)\) is the energy of eigenstate \(n\). The first sum runs over the pair of states where \(E_n(k_z) > E_m(k_z)\).
- \(f_n(k_z)\) is the occupation of eigenstate \(n\).
- \(\bar{\varepsilon}\) is the optical polarization vector defined in \texttt{optics{} region{}}.
- \(\bar{\pi} = \bar{p} + \frac{1}{3m_0 c^2} (\sigma \times \nabla V)\) where \(\bar{p}\) is the canonical momentum operator and \(\frac{1}{4m_0 c^2} (\sigma \times \nabla V)\) is the contribution of spin-orbit interaction.
- \(\bar{\pi}_{nm}(k_z) = \langle n | \bar{\pi} | m \rangle\).
- we call \(\bar{\varepsilon} \cdot \bar{\pi}_{nm}(k_z)\) as the optical matrix elements.
- \(\mathcal{L}(E_n(k_z) - E_m(k_z) - \hbar \omega)\) is the energy broadening function.

- When \texttt{energy_broadening_lorentzian} is specified in \texttt{optics{} region{}}:
  \[
  \mathcal{L}(E_n - E_m - \hbar \omega) = \frac{1}{\pi} \left( \frac{\Gamma/2}{E_n - E_m - \hbar \omega + \Gamma/2} \right),
  \]
  where \(\Gamma\) is the FWHM defined by \texttt{energy_broadening_lorentzian}.

- When \texttt{energy_broadening_gaussian} is specified in \texttt{optics{} region{}}:
  \[
  \mathcal{L}(E_n - E_m - \hbar \omega) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left( - \frac{(E_n - E_m - \hbar \omega)^2}{2\sigma^2} \right)
  \]
  where \texttt{energy_broadening_lorentzian} defines the FWHM \(\Gamma = 2\sqrt{\ln 2} \cdot \sigma\).
When neither `energy_broadening_lorentzian` nor `energy_broadening_gaussian` is specified in `optics{ region{ } }`, $\mathcal{L}$ is replace by the delta function $\delta(E_n - E_m - \hbar \omega)$.

It is also possible to include both Lorentzian and Gaussian broadening (Voigt profile).

The detailed calculation scheme of the optical matrix elements $\vec{\varepsilon} \cdot \vec{\pi}_{nm}(k_z)$ and the absorption spectrum $\alpha$ is described in *1D Optics: Optical absorption for interband and intersubband transitions*.

**Results**

**Absorption**

![Absorption Spectrum](image)

Figure 13.2.3.2: Calculated absorption spectrum $\alpha(\vec{\varepsilon}, E)$ for $\vec{\varepsilon} = \hat{x}, \hat{y}, \hat{z}$.

Figure 13.2.3.2 shows the calculated $\alpha(\vec{\varepsilon}, E)$ specified in `
Optics\absorption_.~.dat` for each polarization $x$, $y$, and $z$. The absorptions for $x$- and $y$-polarization, which are identical due to the rotational symmetry in $x$-$y$ plane, have two peaks at around 0.2 eV (P1) and 0.95 eV (P2). $\alpha(\vec{\varepsilon}, E) = 0$ for $z$-polarization, which is characteristic for intersubband transition. These results can be understood from the output data explained below.

**Note:** $\alpha(\vec{\varepsilon}, E)$ for $z$-polarization is generally non-zero in the calculation through k.p model. This is because the eigenstates above the conduction band edge can have the component of valence band Bloch functions and vice versa (band-mixing).

$\alpha(\hat{z}, E) = 0$ in Figure 13.2.3.2 is reasonable since the single-band model is emulated in this tutorial.
**Eigenvalues, transition energies, and occupations**

![Energy spectrum graph](image)

**Figure 13.2.3.3**: Calculated energy spectrum and Fermi energy (=0 eV).

**Figure 13.2.3.3** shows the calculated energy eigenvalues at \( k_z = 0 \) specified in `Quantum\energy_spectrum_.dat`.

Please note that the output in `Quantum` counts the eigenstates with different spins individually when k.p model is used, while they are counted jointly in `Optics`.

The only states below the Fermi energy are the ground states (no. 1 and 2). Comparing the excitation energy of other upper states to \( k_B T \approx 0.026 \text{ eV at } T = 300 \text{ K} \), we can expect the occupation probability of each excited state is almost 0 and the optical transition will occur only from the ground states in this case.

We can see the peak energy of P1 in **Figure 13.2.3.2** corresponds to the transition energy from the ground states (no. 1 and 2) to the 1st excited states (no. 3, 4, 5, and 6). Also the peak energy of P2 corresponds to the transition energy from the ground states to 5th excited states (no. 17, 18, 19, and 20).

The occupation probabilities for each state can be checked from `Optics\occupation_disp_.dat` as a function of the 1D Bloch wave vector \( k_z \):

As we expected above, the ground state is well occupied for small \( k_z \) and the occupation of the 1st excited state is almost 0.

**Note**: The eigenstates with different spins are counted individually in `Quantum` when k.p model is used, while they are counted jointly in `Optics`.

For example, the two ground states counted as no. 1 and 2 in **Figure 13.2.3.3** due to spin are put together as one eigenstate in `Optics`. Thus `Optics\occupation_disp~kp8_1.dat` shows the occupation of the ground state and `Optics\occupation Disp~kp8_2.dat` and `Optics\occupation Disp~kp8_3.dat` show the 1st excited state in this case.

From the above data of eigenvalues and occupations, we could see which pair of states contributes to each peak in the absorption spectrum **Figure 13.2.3.2**. In order to understand the magnitude of the peaks and why some pairs of states don’t appear as peaks, we will see the output data for \( |\vec{\epsilon} \cdot \vec{\pi}_{nm}(k_z)|^2 \) next.
Transition intensity (Momentum matrix element)

One of the key element for the calculation of optical absorption is the transition intensity

\[ T_{nm}(\vec{\epsilon}, k_z) = \frac{2}{m_0} |\vec{\epsilon} \cdot \pi_{nm}(k_z)|^2 \]  (13.2.3.2)

which has the dimension of energy [eV].

The intensity at \( k_z = 0 \) \( (T_{nm}(\vec{\epsilon}, k_z = 0)) \) for each pair of states \((n,m)\) is specified in \( \text{Optics\_Transitions\_~.txt} \). These intensities whose “From” states are the ground state are shown here (x-polarization). We can also check the transition energy of each pair of states.

<table>
<thead>
<tr>
<th>Energy[eV] From</th>
<th>To</th>
<th>Intensity_k0[eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.19824</td>
<td>1</td>
<td>2.77912</td>
</tr>
<tr>
<td>3.80277e-08</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0.19824</td>
<td>1</td>
<td>2.9137</td>
</tr>
<tr>
<td>3.62712e-08</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0.775938</td>
<td>1</td>
<td>8.37435e-06</td>
</tr>
<tr>
<td>0.00322418</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>0.775938</td>
<td>1</td>
<td>6.88813e-06</td>
</tr>
<tr>
<td>0.00391985</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0.964304</td>
<td>1</td>
<td>0.368533</td>
</tr>
<tr>
<td>5.89532e-08</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0.964304</td>
<td>10</td>
<td>0.427067</td>
</tr>
</tbody>
</table>

We can explain the large P1 (~0.198 eV) and small P2 (~0.964 eV) by the large and small transition intensities in these output data. Also we can see the transitions from 1 to 4, 5, 6, 7 are almost zero and these pairs of states don’t contribute to the absorption (transitions from 1 to 4, 5 are omitted here since Intensity_k0 are too small).
There is also the output files that specify the k-dispersion of the transition intensities for each light polarization in *Optics\transition\_disp\_\~\_.dat*.

### Eigenstates

The probability distribution of eigenfunctions $|\psi(r)|^2$ is output in *Quantum\probabilities\_\~\_.vtr*. The amplitude of the envelope function on each Bloch function $|S\rangle, |X\rangle, |Y\rangle, |Z\rangle$ can be found in *Quantum\amplitudes\_\~\_SXYZ.vtr*.

The analytical expression of the eigenfunctions for the cylindrical quantum wire is shown as eq. (13.2.2.1) in this tutorial: *Electron wave functions in a cylindrical well (2D Quantum Corral)*. According to this analytical solution, the eigenfunction has 2 quantum numbers: $n$ for radial direction and $l$ for circumferential direction.

Here the amplitudes of eigenfunctions calculated by single-band model are shown. We can see the optical transition from ground state ($n = 1, l = 0$) occurs only to the states with $l = \pm 1$.

---

**Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.**

### 13.2.4 2D Optics: Interband absorption spectrum of a GaAs cylindrical quantum wire

*Section author: Naoki Mitsui (simulation), Brandon Loke (write-up and visualisation)*

This tutorial calculates the optical spectrum of a GaAs cylindrical quantum wire with infinite barriers. The formulas used to calculate absorption will be highlighted and a brief explanation of the output files will be given.

For the detailed scheme of the calculation of the optical matrix elements or absorption spectrum, please see our 1D optics tutorial: *1D Optics: Optical absorption for interband and intersubband transitions*. For the corresponding tutorial for the intraband absorption, please see *2D Optics: Intersubband absorption spectrum of a GaAs cylindrical quantum wire*. Input file:

- *2Dcircular\_infinite\_wire\_GaAs\_inter\_nnp.in*

**Note:** Figures in this tutorial will be generated with *nextnanopy*.

The corresponding Jupyter Notebook used to generate the figures in this tutorial can be found here at *2DInterb\_bandQuantum\_Cylinder.ipynb*.

### Structure

![Figure 13.2.4.1: Left: Conduction band edge for cylindrical quantum wire. Right: Slice of the band edge along $x = 0$.](image)
Figure 13.2.3.5: Wave function of the ground state. \((n, l) = (1, 0)\)

Figure 13.2.3.6: Wave function of the 1st excited state. \((n, l) = (1, \pm 1)\)

Figure 13.2.3.7: Wave function of the 2nd excited state. \((n, l) = (1, \pm 2)\)

Figure 13.2.3.8: Wave function of the 3rd excited state. \((n, l) = (2, 0)\)

Figure 13.2.3.9: Wave function of the 4th excited state. \((n, l) = (1, \pm 3)\)

Figure 13.2.3.10: Wave function of the 5th excited state. \((n, l) = (2, \pm 1)\)
model the infinite barrier by assigning 100 eV for the bandedge of AlAs barrier region from database{} section. Please see the input file for the details.

The parameters used in this simulation are as follows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>Value [unit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantum wire radius</td>
<td>$R$</td>
<td>5 [nm]</td>
</tr>
<tr>
<td>barrier height</td>
<td>$E_b$</td>
<td>92 [eV]</td>
</tr>
<tr>
<td>effective electron mass</td>
<td>$m_e$</td>
<td>0.0665</td>
</tr>
<tr>
<td>refractive index</td>
<td>$n_r$</td>
<td>3.3</td>
</tr>
<tr>
<td>doping concentration (n-type)</td>
<td>$N_D$</td>
<td>$5 \cdot 10^{18}$ [cm$^{-3}$]</td>
</tr>
<tr>
<td>linewidth (FWHM)</td>
<td>$\Gamma$</td>
<td>0.01 [eV]</td>
</tr>
<tr>
<td>temperature</td>
<td>$T$</td>
<td>300 [K]</td>
</tr>
</tbody>
</table>

The run{} section is specified as follows:

```plaintext
run{
    poisson{}
    quantum{}
    optics{}
}
```

Then the simulation follows these steps:

1. Poisson equation is solved with the setting specified in the poisson{} section.
2. “Schrödinger” equation is solved with the setting specified in the quantum{} section.
3. “Schrödinger” equation is solved again with the setting specified in the optics{} section and optical properties are calculated.

**Note:**

- If quantum_poisson() is specified instead of quantum{}, Poisson and Schrödinger equations are solved self-consistently.
- optics{} requires that kp8 model is used in the quantum region specified in quantum{}.
- In this tutorial the kp parameters are adjusted so that the conduction and valence bands are decoupled from each other. Thus the single-band Schrödinger equations are solved effectively by the kp solver.

The optical absorption accompanied by the excitation of charge carriers (state $n \rightarrow m$) in a condensed matter is calculated on the basis of Fermi’s golden rule [ChuangOpto1995] in the dimension of (length)$^1$:

$$
\alpha(\vec{r}, \omega) = \frac{\pi e^2}{n_x \epsilon_0 m_0^2 \omega} \frac{1}{V} \sum_{n,m} \sum_{k_z} | \vec{c} \cdot \vec{\pi}_{nm}(k_z) |^2 (f_m(k_z) - f_n(k_z)) \mathcal{L}(E_n(k_z) - E_m(k_z) - \hbar \omega),
$$

(13.2.4.1)
where

• $k_z$ is the Bloch wave vector along translation-invariant directions. In 2D simulation this is 1D vector.

• $E_n(k_z)$ is the energy of eigenstate $n$. The first sum runs over the pair of states where $E_n(k_z) > E_m(k_z)$.

• $f_n(k_z)$ is the occupation of eigenstate $n$.

• $\vec{\epsilon}$ is the optical polarization vector defined in optics{region{}}.

• $\vec{\pi} = \vec{p} + \frac{1}{4m_0c^2}(\sigma \times \nabla V)$ where $\vec{p}$ is the canonical momentum operator and $\frac{1}{4m_0c^2}(\sigma \times \nabla V)$ is the contribution of spin-orbit interaction.

• $\vec{\pi}_{nm}(k_z) = \langle n | \vec{\pi} | m \rangle$.

• $\vec{\epsilon} \cdot \vec{\pi}_{nm}(k_z)$ is known as the optical matrix elements.

• $\mathcal{L}(E_n(k_z) - E_m(k_z) - \hbar \omega)$ is the energy broadening function.

  - When energy_broadening_lorentzian is specified in optics{region{}},
    $$\mathcal{L}(E_n - E_m - \hbar \omega) = \frac{1}{\pi (E_n - E_m - \hbar \omega + \Gamma/2)^2}$$
    where $\Gamma$ is the FWHM defined by energy_broadening_lorentzian.

  - When energy_broadening_gaussian is specified in optics{region{}},
    $$\mathcal{L}(E_n - E_m - \hbar \omega) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(E_n - E_m - \hbar \omega)^2}{2\sigma^2}\right)$$
    where energy_broadening_lorentzian defines the FWMH $\Gamma = 2\sqrt{\ln 2} \cdot \sigma$

  - When neither energy_broadening_lorentzian nor energy_broadening_gaussian is specified in optics{region{}} , $\mathcal{L}$ is replace by the delta function $\delta(E_n - E_m - \hbar \omega)$.

  - It is also possible to include both Lorentzian and Gaussian broadening (Voigt profile).

The detailed calculation scheme of the optical matrix elements $\vec{\epsilon} \cdot \vec{\pi}_{nm}(k_z)$ and the absorption spectrum $\alpha$ is described in 1D Optics: Optical absorption for interband and intersubband transitions.

### Results

#### Absorption

Figure 13.2.4.2 shows the calculated $\alpha(\vec{\epsilon}, E)$ specified in \\'Optics\absorption_{~}.dat\' for each polarization x, y, and z. The absorptions for the x- and y-polarisation are identical due to the rotational symmetry of the quantum cylinder in the x-y plane. It is observed that there are peaks at 1.675 eV (P1), 1.806 eV (P2) and 2.005 eV (P3).

**Note:** $\alpha(\vec{\epsilon}, E)$ for z-polarization is generally non-zero in the calculation through k.p model. This is because the eigenstates above the conduction bandedge can have the component of valence band Bloch functions and vice versa (band-mixing).
Figure 13.2.4.2: Calculated absorption spectrum $\alpha(\vec{\epsilon}, E)$ for $\vec{\epsilon} = \hat{x}, \hat{y}, \hat{z}$.

**Eigenvalues, transition energies, and occupations**

Figure 13.2.4.3 shows the calculated energy eigenvalues at $k_z = 0$ specified in \Quantum\energy_spectrum_.dat.

Please note that the output in Quantum\ counts the eigenstates with different spins individually when k.p model is used, while they are counted jointly in Optics\.

The valence band states lie below the Fermi level (0 eV). The minimum hole energy is indicated in Figure 13.2.4.3 with the purple line. It can be seen through a comparison with Figure 13.2.4.2 that the peak in absorption at P1 corresponds to the transition energy from the minimum hole energy level to the first conduction band state (number 31, 32). Similarly, the peak at P2 corresponds to the transition energy between the minimum hole energy state and the second conduction band state (number 33-36).

The occupation probabilities for each state can be checked from \Optics\occupation_disp_.dat as a function of the 1D Bloch wave vector $k_z$:

In the above figure, the occupation probabilities are plotted for the 1st and the 16th excited state. The 16th excited state corresponds to the lowest conduction band level.

**Note:** The eigenstates with different spins are counted individually in Quantum\ when k.p model is used, while they are counted jointly in Optics\.

For example, the two ground states in the conduction band counted as no.31 and 32 in Figure 13.2.4.3 due to spin are put together as one eigenstate in Optics\.

Thus \Optics\occupation_disp_.kp8_16.dat shows the occupation of the ground state in the conduction band and \Optics\occupation_disp_.kp8_2.dat and \Optics\occupation_disp_.kp8_17.dat show the 1st excited state in the conduction band (number 33 & 34) in Figure 13.2.4.3.

At $T = 300$K, $k_B T \simeq 0.026$ eV , which is insufficient energy to excite electron carriers to the upper conduction band states.

From the above data of eigenvalues and occupations, we could see which pair of states contributes to each peak in the absorption spectrum Figure 13.2.4.2. In order to understand the magnitude of the peaks and why some pairs of states don’t appear as peaks, we will see the output data for $|\vec{\epsilon} \cdot \vec{\eta}_{nm}(k_z)|^2$ next.
Figure 13.2.4.3: Calculated energy spectrum and the minimum hole energy.

Figure 13.2.4.4: Calculated occupation probabilities for the ground state and 16th excited state as a function of $k_z$. 
Transition intensity (Momentum matrix element)

An important part of the calculation of optical absorption is the transition intensity:

\[ T_{nm}(\vec{r}, k_z) = \frac{2}{m_0} |\vec{r} \cdot \vec{\pi}_{nm}(k_z)|^2 \]  

(13.2.4.2)

which has dimensions of energy [eV].

The intensity at \( k_z = 0 \) \( (T_{nm}(\vec{r}, k_z = 0)) \) for each pair of states \((n, m)\) is specified in Optics\transitions\_\~.txt. These intensities whose “From” states are the ground state are shown here (x-polarization). We can also check the transition energy of each pair of states.

<table>
<thead>
<tr>
<th>Energy [eV]</th>
<th>From</th>
<th>To</th>
<th>Intensity_k0 [eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00196</td>
<td>10</td>
<td>19</td>
<td>5.9913</td>
</tr>
<tr>
<td>2.00394</td>
<td>10</td>
<td>20</td>
<td>1.79227</td>
</tr>
<tr>
<td>1.67437</td>
<td>13</td>
<td>16</td>
<td>19.9021</td>
</tr>
<tr>
<td>1.80179</td>
<td>14</td>
<td>17</td>
<td>6.25494</td>
</tr>
<tr>
<td>1.85897e-09</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Above are the transitions of interest. The other transitions are omitted for brevity. The “From” and “To” states tell us which band the transition belongs to. Using this information, we can identify which peaks (P1, P2, P3) correspond to transitions between which bands. This is marked in Figure 13.2.4.3.

There are also the output files that specify the k-dispersion of the transition intensities for each light polarization in Optics\transition\_disp\_\~.dat.

Eigenstates

The probability distributions of the eigenfunctions \(|\psi(r)|^2\) can be found in Quantum\probabilities\_\~.vtr. The amplitude of the envelope function on each Bloch function \(|S\rangle, |X\rangle, |Y\rangle, |Z\rangle\) can be found in Quantum\amplitudes\_\~_SXYZ.vtr.

The analytical expression of the eigenfunctions for the cylindrical quantum wire is shown as eq. (13.2.2.1) in this tutorial: Electron wave functions in a cylindrical well (2D Quantum Corral). According to this analytical solution, the eigenfunction has 2 quantum numbers: \(n\) for radial direction and \(l\) for circumferential direction.

Here the amplitudes of eigenfunctions calculated by single-band model are shown. We can see the optical transition from ground state \((n = 1, l = 0)\) occurs only to the states with \(l = \pm 1\). The file used for this plot is amplitudes\_quantum_region_Gamma_00000.vtr in the single band calculation.

Wave functions of the energy eigenstates calculated by the single-band model.

Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.

13.2.5 Electron wave functions of a 2D slice of a Triple Gate MOSFET

In this tutorial we demonstrate the 2D simulation of a Triple Gate MOSFET. We solve the two-dimensional Schrödinger and Poisson equations self-consistently for a 2D slice. We would see the difference between the electron densities calculated quantum mechanically and classically.

The relevant input files are as follows:

- 2DSi_TGMOS_2Dcut_atGate_cl.in / *_nnp.in
- 2DSi_TGMOS_2Dcut_atGate_qm.in / *_nnp.in
- 2DSi_TGMOS_2Dcut_atGate_qm_iso.in / *_nnp.in
Figure 13.2.4.5: Wave function of the ground state. \((n, l) = (1, 0)\)

Figure 13.2.4.6: Wave function of the first excited state. \((n, l) = (1, \pm 1)\)
Figure 13.2.4.7: Wave function of the second excited state. \((n, l) = (1, \pm 2)\)

Figure 13.2.4.8: Wave function of the third excited state. \((n, l) = (2, 0)\)
Figure 13.2.4.9: Wave function of the fourth excited state. \((n, l) = (1, \pm 3)\)

Figure 13.2.4.10: Wave function of the fifth excited state. \((n, l) = (2, \pm 1)\)
If you want to obtain the input files that are used within this tutorial, please contact support [at] nextnano.com.

The values and graphs described in this tutorial are the result of nextnano³.

2D Simulation

Structure

A Triple Gate MOSFET is a nanowire if the dimensions along the x and y directions are only a few nanometers, thus quantization effects have to be taken into account. The structure considered is as follows:

• The Si channel has a rectangular shape with a width of 5 nm and a height of 5 nm.
• The Si channel is surrounded by SiO₂ (thickness 1.5 nm).

The Si/SiO₂ nanowire is surrounded by a Gate (at the left and right side, and at the top).

The following schematic shows a 2D slice of a 3D Triple Gate MOSFET.

Simulation Details

In this tutorial we will only simulate this 2D slice and not the whole 3D structure.

We apply a voltage of 0.5 V to the Gates and solve the two-dimensional Schrödinger and Poisson equations self-consistently (including the SiO₂ region).

There are six equivalent conduction band minima in silicon (Delta valleys). Since the constant energy surfaces are ellipsoids, the mass tensor has the following two kinds of effective masses:

• The longitudinal mass is \( m_\perp = 0.916 m_0 \).
• The transversal mass is \( m_\parallel = 0.190 m_0 \) (2 directions).

Therefore we need to solve three 2D Schrödinger equations with different effective mass tensor orientations.

Our Schrödinger equations are numbered \( X1, X2, X3 \) in nextnano++ or \( \text{deg1}, \text{deg2}, \text{deg3} \) in nextnano³.

• \( \text{X1/deg1}: \) \( m_{xx} = m_t = 0.916 m_0, \ m_{yy} = m_t = 0.190 m_0 \)
• \( \text{X2/deg2}: \) \( m_{xx} = m_t = 0.190 m_0, \ m_{yy} = m_t = 0.916 m_0 \)
• \( \text{X3/deg3}: \) \( m_{xx} = m_{yy} = m_t = 0.190 m_0 \)

The potential \( E_c(x, y) \) that enters the Schrödinger equation is the same in these three cases.

Note: The cases a) and b) are not identical (i.e. degenerate) because the potential is not symmetric with respect to changing x and y coordinates.

The following keyword and specifier can be used to output the effective mass tensors \( (1/m_{ij}) \).

```plaintext
# nextnano++
output{
  material_parameters{
    charge_carrier_masses{
      (continues on next page)
    }
  }
}
```
Figure 13.2.5.1: 2D slice of a 3D Triple Gate MOSFET

Figure 13.2.5.2: constant energy surface of Si conduction band
Results

Electron wave functions $|\psi^2|$

- $2DSi_TGMOS_2Dcut_atGate_gnm_nnp.in, *_nn3.in$

The lowest eigenstates for the cases a), b) and c) are the following:

- $X1/deg1$: a) $m_{xx} = m_t = 0.916m_0, m_{yy} = m_t = 0.190m_0$

Figure 13.2.5.3: $E_{1,X1} = -26$ meV, $E_{2,X1} = -1$ meV, $E_{3,X1} = 77$ meV

Here, the heavier mass is along the x direction, and the lighter mass along the y direction. The energy spacing between the two lowest subbands is about 24 meV. The eigenvalues are contained in $bias_00000/Quantum/energy_spectrum_quantum_region_X1_00000.dat/Schroedinger_1band/ev2D_cb003_qc001_sg001_deg001_dir_Kx001_Ky001_Kz001.dat$.

- $X2/deg2$: b) $m_{xx} = m_t = 0.190m_0, m_{yy} = m_t = 0.916m_0$

Here, the lighter mass is along the x direction, and the heavier mass along the y direction. The energy spacing between the two lowest subbands is about 35 meV. The eigenvalues are contained in $bias_00000/Quantum/energy_spectrum_quantum_region_X2_00000.dat/Schroedinger_1band/ev2D_cb003_qc001_sg001_deg002_dir_Kx001_Ky001_Kz001.dat$.

- $X3/deg3$: c) $m_{xx} = m_{yy} = m_t = 0.190m_0$

These eigenvalues have the lighter mass in the x and y directions. Consequently, their energies are much higher than in the other two Schrödinger equations. The energy spacings between the lowest subbands is of the order 140-150 meV. The eigenvalues are contained in $bias_00000/Quantum/energy_spectrum_quantum_region_X3_00000.dat/Schroedinger_1band/ev2D_cb003_qc001_sg001_deg003_dir_Kx001_Ky001_Kz001.dat$.

(Compare the wave functions and the energies with the isotropic case as discussed further below.)
Figure 13.2.5.4: $E_{1,X_2} = -28 \text{ meV}$, $E_{2,X_2} = 6 \text{ meV}$, $E_{3,X_2} = 82 \text{ meV}$

Figure 13.2.5.5: $E_{1,X_3} = 16 \text{ meV}$, $E_{2,X_3} = 167 \text{ meV}$, $E_{3,X_3} = 173 \text{ meV}$
Electron density

The resulting electron density has the following shape, see Figure 13.2.5.6:

The units are \(1 \times 10^{18}\) cm\(^{-3}\). The density has been calculated by occupying the eigenstates with respect to the Fermi level which is at 0 eV. Note that the quantum mechanical density is close to zero near the Si/SiO\(_2\) interfaces because the wave functions tend to zero at the SiO\(_2\) barriers.

![Figure 13.2.5.6: electron density](image)

Figure 13.2.5.6: electron density

Figure 13.2.5.7 shows the same quantum mechanical electron density together with two slices through the conduction band edges. The units are in eV and the conduction band offset between SiO\(_2\) and Si is 3.1 eV. At the gates, the conduction band edge is set to -0.5 eV, representing the applied bias of 0.5 eV. One can clearly see that for silicon in the middle of the nanowire the conduction band has its highest value and its lowest value close at the Si/SiO\(_2\) interface.

If one had neglected the effect of quantum confinement, then the resulting classical electron density would have peaks near the Si/SiO\(_2\) interfaces as is shown in Figure 13.2.5.8.

- 2DSi_TGMOS_2Dcut_atGate_cl_nnp.in, _mm3.in
Figure 13.2.5.7: electron density and slices through the conduction band edges
Figure 13.2.5.8: classical electron density calculated by 2DSi_TGMOS_2Dcut_atGate_cl.in
Obviously, a realistic calculation of such transistors cannot be based on classical densities. The full 2D (or better 3D) Schrödinger equations have to be solved. The IV characteristics of such a quantum-mechanically calculated Triple Gate MOSFET transistor will be discussed in another tutorial.

**Isotropic electron masses**

Very often, for simplicity, an isotropic electron mass for the Schrödinger equation is assumed. E.g. the DOS (density of states) electron mass of Si in the Delta minima can be calculated as follows:

\[ m_{e,DOS}^* = (m_lm_t^2)^{1/3} = (0.9160.19^2)^{1/3}m_0 = 0.321m_0 \]

In this case, only one Schrödinger equation has to be solved (in contrast to three equations as described above). The wave functions and energies in this case are:

- \( m_{xx} = m_{yy} = m_{DOS} = 0.321m_0 \)

The wave functions \(|\psi|^2\) look very similar as in the case of “X3/deg3: c)” (see above) where the masses are isotropic in the (x,y) plane but here, the energy spacings between different subbands are smaller (around 90-100 meV) because the DOS mass is larger than the transversal masses.

The eigenvalues are contained in bias_00000/Quantum/energy_spectrum_quantum_region_X3_00000.dat/Schroedinger_1band/ev2D.dat.
3D simulation of the Triple Gate MOSFET

The following figures show the results of the self-consistent 3D Schrödinger-Poisson solution of this Triple Gate structure (Si channel length = 25 nm, source region length = 10 nm, drain region length = 10 nm, constant doping profile in source and drain region with a doping concentration of $1 \times 10^{20}$ cm$^{-3}$ (fully ionized).

![Figure showing the 3D structure of a triple gate MOSFET and electron density through a 2D slice.]

Figure 13.2.5.11: The whole 3D structure of this triple gate MOSFET and electron density through a 2D slice.

The plots show the isosurfaces of the electron densities along 2D slices through the Triple Gate MOSFET. Figure 13.2.5.12 and Figure 13.2.5.13 also show 1D slices of the conduction band profiles and 1D slices of the electron densities in the middle of the device.

The classical densities would look similar to the classical densities of the 2D calculations shown above.

13.2.6 Landau levels of a bulk GaAs sample in a magnetic field

In this tutorial, we study the electron energy levels of a bulk GaAs sample that is subject to a magnetic field.

The input files are the followings:

- 2DBulkGaAs_LandauLevels.nn3.in / _nnp.in

Simulation details

The magnetic field is oriented along the z direction. The motion in the z direction is not influenced by the magnetic field and is thus that of a free particle with energies and wave functions given by:

$$E_z = \frac{\hbar^2 k_z^2}{2m^*}$$

$$\psi(z) = \exp(\pm ik_z z)$$

For that reason, we do not include the z direction into our simulation domain, and thus only simulate in the (x,y) plane (two-dimensional simulation).
Figure 13.2.5.12: closed channel, $V_{SD} = 0.0\,\text{V}$, $V_{SG} = 0.0\,\text{V}$
Figure 13.2.5.13: open channel, $V_{SD} = 0.0\text{V}$, $V_{SG} = 0.5\text{V}$
This plane has the size of 300nm × 300nm and consists of GaAs. At the domain boundaries we employ Dirichlet boundary conditions to the Schrödinger equation, i.e. infinite barriers.

We calculate the eigenstates for different magnetic field strengths (1 T, 2 T, 3 T).

Magnetic length and cyclotron frequency A useful quantity is the magnetic length (or Landau magnetic length) which is defined as:

\[ l_B = \left( \frac{\hbar}{m^*_e \omega_c} \right)^{1/2} = \left( \frac{\hbar}{|e|B} \right)^{1/2} \]

It is independent of the mass of the particle and depends only on the magnetic field strength:

- 1 T: \( l_B = 25.6556 \text{ nm} \)
- 2 T: \( l_B = 18.1413 \text{ nm} \)
- 3 T: \( l_B = 14.8123 \text{ nm} \)

In the above formula, \( \omega_c \) is the cyclotron frequency:

\[ \omega_c = \frac{|e|B}{m^*_e} \]

Thus for the electrons in GaAs, where \( m^*_e = 0.067 m_0 \), it holds for the different magnetic field strengths:

- 1 T: \( \hbar \omega_c = 1.7279 \text{ meV} \)
- 2 T: \( \hbar \omega_c = 3.4558 \text{ meV} \)
- 3 T: \( \hbar \omega_c = 5.1836 \text{ meV} \)
Results

The calculated energy spectra for different magnetic fields (1 T, 2 T, 3 T) are as follows:

Landau levels

The Landau levels are analytically given by

\[ E_n = \left( n - \frac{1}{2} \right) \hbar \omega_c \]

where \( n = 1, 2, 3, ... \)
The number of states for each Landau level can be calculated as follows (see P.Y. Yu, M. Cardona, Fundamentals of Semiconductors, p. 536, 3rd ed.):

\[
N = L_x L_y \frac{|e|B}{\hbar} = \frac{L_x L_y}{2\pi l_B^2}
\]

where \(L_x\) and \(L_y\) are the lengths in the x and y directions (300 nm in this example) and \(l_B\) is the magnetic length. Here we ignore spin.

- \(N(1\ \text{T}) = 21.76 \sim 22\) states per Landau level (in the figure above: 42)
- \(N(2\ \text{T}) = 43.52 \sim 44\) states per Landau level (in the figure above: 86)
- \(N(3\ \text{T}) = 65.29 \sim 65\) states per Landau level (in the figure above: 130)

When \texttt{magnetic\_field()}\ is specified, \texttt{nextnano++} caculates the Schrödinger-Pauli equation, which takes into account the spin. Since the interaction energy between the spin and magnetic field is small compared to the separation of Landau levels, the number of states per Landau level calculated by \texttt{nextnano++} is almost double of the analytical result that ignores the spin.

\textbf{Energy eigenvalues}

For the calculations, we used the symmetric gauge \(A = -\frac{1}{2}r \times B = \frac{1}{2}B \times r\) leading to the following energies (see J.H. Davies, The Physics of Low-Dimensional Semiconductors, p. 222):

\[
E_{n,l} = \left( n + \frac{1}{2} l + \frac{1}{2} |l| - \frac{1}{2} \right) \hbar \omega_c
\]

One can see that all states having a negative value of \(l\) are degenerate with the states with \(l = 0\), i.e. the allowed energies are independent of \(l\) if \(l < 0\) (for the same \(n\)). The energies increase if \(l\) increases (for \(l > 0\) and for the same \(n\)).

\textit{Please help us to improve our tutorial. Send comments to support [at] nextnano.com.}
13.2.7 Fock-Darwin states of a 2D parabolic, isotropic potential in a magnetic field

- 2D parabolic confinement with $\hbar \omega_0 = 4 \text{ meV}$
  - Results
- 2D parabolic confinement with $\hbar \omega_0 = 3 \text{ meV}$ - Fock-Darwin spectrum
  - Results

In this tutorial, we study the electron energy levels of a two-dimensional parabolic confinement potential that is subject to a magnetic field. Such a potential can be constructed by surrounding GaAs with an Al$_x$Ga$_{1-x}$As alloy that has a parabolic alloy profile in the (x,y) plane.

The magnetic field $\mathbf{B}$ is oriented along the z direction. $\mathbf{B}$ is the rotation of the vector potential $\mathbf{A}$ so, in this case, we can always take the z-component of the vector potential as 0. Thus the motion in the z direction is not influenced by the magnetic field and that of a free particle with energies and wave functions given by:

$$E_z = \frac{\hbar^2 k_z^2}{2m_z^*}$$

$$\psi(z) = \exp(\pm ik_z z)$$

For that reason, we do not include the z direction into our simulation domain, and thus only simulate in the (x,y) plane (two-dimensional simulation).

This tutorial consists of two parts. First we benchmark the nextnano++ code to the numerical calculation in

M. Governale, C. Ungarelli
Gauge-invariant grid discretization of the Schrödinger equation

Second we reproduce some of the figures of

L.P. Kouwenhoven, D.G. Austing, S. Tarucha
Few-electron quantum dots

The input files are the followings:

- 2DGaAs_BiParabolicQW_4meV_GovernalePRB1998.in/*_nnp.in
- 2DGaAs_BiParabolicQW_3meV_FockDarwin.in/*_nnp.in

The figures provided in this tutorial is the results of nextnano++ input files.

**Note:** When magnetic_field is specified in a 2D or 3D simulation of nextnano++, the Pauli equation, in which both spin eigenfunctions are taken into consideration, is calculated instead of Schrödinger equation. On the other hand, the effect of spin is not considered in nextnano++. Thus the number of eigenstates calculated in this nextnano++ input file is two times of the one in the nextnano++ input file. Since the splitting of energy levels due to the spin is small compared to the difference of energy levels, we call the two states splitted from $i$th eigenstates as “$i$th eigenstates with up-spin” or “$i$th eigenstates with down-spin”.
2D parabolic confinement with $\hbar \omega_0 = 4$ meV

We want to benchmark the nextnano++ code to the numerical calculation in

M. Governale, C. Ungarelli
Gauge-invariant grid discretization of the Schrödinger equation

Input file 2DGaAs_BiParabolicQW_4meV_GovernalePRB1998.in/*_nnp.in aims to reproduce the figures of eigenvalues, ground state and 14th excited state probability densities, and ground state energy as a function of magnetic field magnitude (Fig.1, 2, 3 and 4 of the paper).

The GaAs sample extends in the x and y directions (i.e. this is a two-dimensional simulation) and has the size of 240 nm x 240 nm. At the domain boundaries we employ Dirichlet boundary conditions to the Schrödinger equation, i.e. infinite barriers. The grid is chosen to be rectangular with a grid spacing of 2.4 nm, in agreement with Governale’s paper.

A two-dimensional parabolic confinement potential is constructed by surrounding GaAs with an Al$_x$Ga$_{1-x}$As alloy that has a parabolic alloy profile in the (x,y) plane. This is chosen so that the electron ground state has the energy: $E_1 = \hbar \omega_0 = 4$ meV (without magnetic field).

The magnetic field is oriented along the z direction, i.e. it is perpendicular to the simulation plane which is oriented in the (x,y) plane). (In nextnano++, the direction is automatically set to the direction perpendicular to the simulation plane.) We calculate the eigenstates for different magnetic field strengths (1 T, 2 T, ... , 20 T), i.e. we make use of the magnetic field sweep. Since nextnano++ doesn’t have this feature for magnetic_field so far, please use the “Template” feature of nextnanomat (See the last section of Double Quantum Well).

In nextnano++, the magnetic field sweeping can be specified in the input file:

```
global{
  magnetic_field{
    strength $STRENGTH
    #direction = [,,] # We must not specify this in 1D or 2D simulation
  }
}
```

Magnetic length and cyclotron frequency A useful quantity is the magnetic length (or Landau magnetic
length) which is defined as:

\[ l_B = \left( \frac{\hbar}{m_e^* \omega_c} \right)^{1/2} = \left( \frac{\hbar}{|e| B} \right)^{1/2} \]

It is independent of the mass of the particle and depends only on the magnetic field strength:

- 1 T: \( l_B = 25.6556 \text{ nm} \)
- 2 T: \( l_B = 18.1413 \text{ nm} \)
- 3 T: \( l_B = 14.8123 \text{ nm} \)
- …
- 20 T: \( l_B = 5.7368 \text{ nm} \)

The electron effective mass in GaAs is \( m_e^* = 0.067 m_0 \). We assume this value for the effective mass in the whole region (i.e. also inside the AlGaAs alloy). In the above formula, \( \omega_c \) is the cyclotron frequency:

\[ \omega_c = \frac{|e| B}{m_e^*} \]

Thus for the electrons in GaAs, where \( m_e^* = 0.067 m_0 \), it holds for the different magnetic field strengths:

- 1 T: \( \hbar \omega_c = 1.7279 \text{ meV} \)
- 2 T: \( \hbar \omega_c = 3.4558 \text{ meV} \)
- 3 T: \( \hbar \omega_c = 5.1836 \text{ meV} \)
- …
- 20 T: \( \hbar \omega_c = 34.5575 \text{ meV} \)

Results

**Lowest 15 eigenvalues** The following figure shows the lowest fifteen eigenvalues for a magnetic field magnitude of \( B = 10 \text{ T} \). It is in perfect agreement with Fig. 1 of Governale’s paper. The ground state has the energy \( E_{1\uparrow} = 9.38 \text{ meV} \) and \( E_{1\downarrow} = 9.55 \) (at \( B = 10 \text{ T} \)). The spin-split energy, \( \frac{eB}{2m_e^*} \) is 0.174 meV, is calculated from our result as 0.174 meV which is constant in all of the pair of spin states.
Probability densities ($\psi^2$) The following figure shows the probability density of the ground state with up-spin ($\psi^2$) for a magnetic field magnitude of $B = 10$ T. It is in perfect agreement with Fig. 2(a) of Governale’s paper. The ground state has the energy $E_{1,\uparrow} = 9.38 \text{ meV}$ and $E_{1,\downarrow} = 9.55$ (at $B = 10$ T) in nextnano++. The corresponding eigenvalue calculated in nextnano$^3$ is $E_1 = 9.44 \text{ meV}$.

The left, vertical axis shows $\psi^2$ in units of nm$^{-2}$ (the peak value is 0.00267 nm$^{-2}$).

In the same figure, the parabolic conduction band edge confinement potential is also shown. The above axis shows the colormap of the conduction band edge values. In the middle of the sample the conduction band edge is 0 eV, and at the boundary region, the conduction band edge has the value 0.1014 eV.

The following figure shows the probability density ($\psi^2$) of the 14th excited state (up-spin) (i.e. $E_{15,\uparrow}$) for a magnetic field magnitude of $B = 10$ T. It is in perfect agreement with Fig. 3(a) of Governale’s paper. 14th excited states have the energy $E_{15,\uparrow} = 21.71$ and $E_{15,\downarrow} = 21.88$ meV (at $B = 10$ T). The corresponding eigenvalue calculated in nextnano$^3$ is $E_{15} = 21.72$ meV. The left, vertical axis shows $\psi^2$ in units of nm$^{-2}$ (the peak value is 0.000283 nm$^{-2}$).

In the same figure, parabolic conduction band edge confinement potential is also shown. The above axis shows the colormap of the conduction band edge values. In the middle of the sample the conduction band edge is 0 eV, and at the boundary region, the conduction band edge has the value 0.1014 eV.

Ground state energy vs. magnetic field magnitude The following figure shows the ground state energy as a function of magnetic field magnitude. It is in perfect agreement with Fig.4 of Governale’s paper. The ground state has the energy $E_{1} = 4.04 \text{ meV}$ (spin-degenerated) in nextnano++ and $E_1 = 4.01 \text{ meV}$ in nextnano$^3$ at $B = 0$ T.
The following figure shows the magnetic field strength dependence of the spin-split energy \( E_{1,\downarrow} - E_{1,\uparrow} \). The formula of the split energy in the Pauli equation is \( \frac{eB}{2m^*} \). We can see the proportionality is reproduced in our calculation. The factor is calculated as 0.0174 [meV/T].

**2D parabolic confinement with \( h\omega_0 = 3 \text{ meV} \) - Fock-Darwin spectrum**

Next we reproduce some of the figures of

L.P. Kouwenhoven, D.G. Austing, S. Tarucha

Few-electron quantum dots


Input file 2DGaAs_BiParabolicQW_3meV_FockDarwin.in/*.nnp.in aims to reproduce the figures of the eigenvalues as a function of magnetic field magnitude and the probability densities of some of eigenstates (Figs. 5(a) and 6(a) (which are analytical results) of the paper).

The GaAs sample extends in the x and y directions (i.e. this is a two-dimensional simulation) and has the size of 600 nm x 600 nm. At the domain boundaries we employ Dirichlet boundary conditions to the Schrödinger equation, i.e. infinite barriers.

A two-dimensional parabolic confinement potential is constructed by surrounding GaAs with an Al\(_x\)Ga\(_{1-x}\)As alloy that has a parabolic alloy profile in the (x,y) plane. This is chosen so that the electron ground state has the energy: \( E_1 = h\omega_0 = 3 \text{ meV} \) (without magnetic field) in agreement to the paper.

The eigenvalues of a two-dimensional parabolic potential that is subject to a magnetic field can be solved analytically. The spectrum of the resulting eigenstates is known as the **Fock-Darwin states** (1928):

\[
E_{n,l} = (2n + |l| + 1)\hbar[\frac{1}{4}\omega_c^2 + \frac{1}{4}\omega_0^2]^{1/2} - \frac{1}{2}\hbar\omega_c \quad \text{for} \quad n = 0, 1, 2, 3, \ldots \text{and} \quad l = 0, \pm 1, \pm 2, \ldots
\]

Note that the last term is \( \omega_c \) and not \( \omega_0 \) as in Kouwenhoven’s paper. \( \omega_c = \frac{|e|B}{m^*c} = \) cyclotron frequency, as described before.)
Each of these states is two-fold spin-degenerate. A magnetic field lifts this degeneracy (Zeeman splitting). This effect is taking into account only in the input file of nextnano++ but this splitting is small compared to the scale of $E_{n,l}$.

The degeneracy of the eigenvalues for zero magnetic field is as follows:

- the ground state is not degenerate
- the second state is two-fold degenerate
- the third state is three-fold degenerate
- the forth state is four-fold degenerate
- ...

Applying a magnetic field, these degeneracies are lifted as the following figure.

**Results**

**Fock-Darwin spectrum** The following figure shows the calculated Fock-Darwin spectrum, i.e. the eigenstates as a function of magnetic field magnitude. The figure is in excellent agreement with Fig. 5(a) of Kouwenhoven’s paper.

*Figure showing the calculated Fock-Darwin spectrum.*

**Probability densities ($\psi^2$)** The following figure show the probability densities ($\psi^2$) of some of these eigenstates for a magnetic field of $B = 0.05$ T. All of them are the up-spin states. The label of the colorbar shows the actual number of each eigenstates specified in the data file. For example, 5th state in this figure has the label “Psi^2_9[nm^-9]”.

The figures are in excellent agreement with Fig. 6(a) of Kouwenhoven’s paper.

The parabolic conduction band edges are also shown.

Fock-Darwin spectrum in a very high magnetic fields
Figure 13.2.7.1: left: \((n, l) = (0, 0)\) (1st), right: \((n, l) = (0, 1)\) (2nd)

Figure 13.2.7.2: left: \((n, l) = (0, 2)\) (4th), right: \((n, l) = (1, 0)\) (5th)

Figure 13.2.7.3: left: \((n, l) = (2, 0)\) (13th), right: \((n, l) = (2, 2)\) (18th)
The following figure shows the magnetic field dependence of the lowest 30 eigen values (0~4T) and lowest 60 eigenvalues (4~70T). We can see that eventually all states are becoming degenerate Landau levels for very high magnetic fields. The reason is that the electrons are confined only by the magnetic field and not any longer by the parabolic conduction band edge.

The red line shows the fan of the lowest Landau level at $\frac{1}{2}\hbar\omega_c$. The higher lying states (not shown) will collect in the second, third, ..., and higher Landau fans (not shown).

The left part of the figure (black region) contains exactly the same Fock-Darwin spectrum that has been shown in the figure further above (from 0 T to 3.5 T).

### 13.2.8 Fock-Darwin states of a 2D parabolic, anisotropic (elliptical) potential in a magnetic field

- **1D parabolic confinement along the x direction with $\hbar\omega_0 = 4.6$ meV (1D simulation)**
- **1D parabolic confinement along the y direction with $\hbar\omega_0 = 6.1$ meV (1D simulation)**
- **2D parabolic, anisotropic (elliptical) confinement with $\hbar\omega_x = 4.6$ meV and $\hbar\omega_y = 6.1$ meV - Fock-Darwin-like spectrum (2D simulation)**

In this tutorial we study the electron energy levels of a two-dimensional parabolic, anisotropic (elliptical) confinement potential that is subject to a magnetic field. Such a potential can be constructed by surrounding GaAs with an Al$_x$Ga$_{1-x}$As alloy that has a parabolic alloy profile in the (x,y) plane.
It is a good idea to get familiar with the results of a 2D parabolic and isotropic confinement beforehand: *Fock-Darwin states of a 2D parabolic potential in a magnetic field*

The input files used in this tutorial are the followings:

- `1DGaAs_ParabolicQW_infinite_4_6meV.in`
- `1DGaAs_ParabolicQW_infinite_6_1meV.in`
- `2DGaAs_BiParabolicEllipticQD_Austing.in/*_nnp.in`

First, it is necessary to study the energy states of a 1D parabolic confinement.

**1D parabolic confinement along the x direction with \( h\omega_0 = 4.6 \) meV (1D simulation)**

- `1DGaAs_ParabolicQW_infinite_4_6meV.in`

For similar results and a discussion, we refer to this tutorial: Parabolic Quantum Well (GaAs / AlAs)

**1D parabolic confinement along the y direction with \( h\omega_0 = 6.1 \) meV (1D simulation)**

- `1DGaAs_ParabolicQW_infinite_6_1meV.in`

For similar results and a discussion, we refer to this tutorial: Parabolic Quantum Well (GaAs / AlAs)

**2D parabolic, anisotropic (elliptical) confinement with \( h\omega_x = 4.6 \) meV and \( h\omega_y = 6.1 \) meV - Fock-Darwin-like spectrum (2D simulation)**

- `2DGaAs_BiParabolicEllipticQD_Austing.in/*_nnp.in`

The electron effective mass in GaAs is \( m_e^* = 0.067m_0 \). We assume this value for the effective mass in the whole region (i.e. also inside the AlGaAs alloy).

**Ground state wave function (\( \psi^2 \))**

The following figure shows the parabolic, anisotropic (elliptical) conduction band edge confinement potential, as well as the ground state wave function \((\psi^2)\) at \( B = 0 \) T calculated in nextnano++. In the middle of the sample the conduction band edge is at 0 eV and at the boundary region the conduction band edge has the value 0.84 eV. The radii of the ellipse are 300 nm along the x axis and 226 nm along the y axis. The parabolic confinement along the x direction is: \( h\omega_x = 4.6 \) meV

The parabolic confinement along the y direction is: \( h\omega_y = 6.1 \) meV

Thus the ellipticity is roughly 4/3.

**Fock-Darwin spectrum**

At zero magnetic field, the eigenvalues for such a system are given by:

\[
E_{n_x,n_y} = (n_x + \frac{1}{2})h\omega_x + (n_y + \frac{1}{2})h\omega_y = n + \frac{1}{2}|l| - \frac{1}{2}|l|n_y = n + \frac{1}{2}|l| + \frac{1}{2}|l|
\]

for \( n = 0, 1, 2, 3, \ldots \) \( l = 0, \pm 1, \pm 2, \ldots \)

where \( n \) is a radial quantum number, \( l \) an angular momentum quantum number, \( \omega_x \) and \( \omega_y \) oscillator frequencies.

The eigenvalue spectrum of a 2D parabolic and isotropic potential shows a shell-like structure: Energy levels of an “artificial atom” - 2D harmonic potential . For the anisotropic elliptical potential, this degeneracy at $B = 0$ T is lifted.

The following figure shows the calculated Fock-Darwin-like spectrum, i.e. the eigenstates as a function of magnetic field magnitude. This is the result of nextnano$^3$ and each of these states is two-fold spin-degenerate. However, a magnetic field lifts this degeneracy (Zeeman splitting) but this effect is not taking into account in this tutorial.

Such a spectrum can be related to experimental transport measurements which give insight into the single-particle energy spectrum of a quantum dot.

The rectangles in the above figure are related to the figures of the following publications:

cyan rectangle: Fig.2 of

Two-level anti-crossings high up in the single-particle energy spectrum of a quantum dot
C. Payette, D.G. Austing, G. Yu, J.A. Gupta, S.V. Nair, B. Partoens, S. Amaha, S. Tarucha

green rectangle and red rectangle: Fig.2(b) and Fig.3(a) of

Probing by transport the single-particle energy spectrum up to high energy of one quantum dot with the ground state of an adjacent weakly coupled quantum dot
D.G. Austing, G. Yu, C. Payette, J.A. Gupta, M. Korkusinski, G.C. Aers
physica status solidi (a), 508 (2007)
Fock-Darwin spectrum of a 2D parabolic, anisotropic (elliptical) potential in a magnetic field.
It is interesting to note that there are exact crossings in the calculated spectrum whereas the experiment reveals anti-crossings. In the first reference, this difference on crossings is regarded as a clue to investigate the deviations between the confining potential of realistic dots used in the experiment and the idealistic parabolic potential assumed in the calculation.

13.2.9 Flying Qubit

In this tutorial, we discuss multi-terminal electron transport in various nanodevices. As an example, we focus on so-called Electron flying qubits, which are solid-state counterparts of the quantum optics devices. Basic building blocks of these qubits are the following semiconductor bases nanodevices:

- Tunneling-coupled wires, TCW - the electronic counterpart of the optical beam splitter, see Figure 13.2.9.1;
- Aharonov-Bohm rings, ABR - the electronic counterpart of the optical interferometer, or their analogy realized in asymetrically gated regions, see the central region of Figure 13.2.9.2;
- Circuits containing these elements connected in a series, see Figure 13.2.9.2.

Left rectangular regions with numbers 1 and 2 are incoming leads, where the electron can be injected into the nanodevice. We will assume that it is injected into the first lead. Right rectangular regions with numbers 3 and 4 are outgoing leads, where the electron can be detected after propagating through the entire nanodevice. The functionality of the electron flying qubits request a reflectionless propagation of the electron. If the electron is reflected and returns to one of the incoming leads, a part of the quantum information is lost. The important task of numerical simulations is to identify regimes where reflection is reduced as much as possible.

The interior part of the nanodevices is assumed to be made from 2D GaAs-based semiconductor and includes regions with different electrostatic potentials and applied gate voltages. Those govern the energy profile through which the electron propagates. Colors reflect the strength of the electrostatic potential in different parts of the device, ranging from 0 eV (dark blue color) up to \( \gg 1 \) eV (dark red color).

Let us first discuss transport in TCW. The horizontal line in Figure 13.2.9.1 shows a potential barrier separating two paths, along which the electron can move towards the outgoing leads. Red parts of the barrier are impenetrable for the electron while the electron can tunnel through the light blue segment. The latter is precisely the region where the quantum interference between the upper- and lower- path takes place. Having experienced the interference, the electron wave-function is split between the upper- and lower- path. As a result, there is a finite probability to detect the electron in one of the outgoing leads, either 3 or 4. This probability depends both on the electron energy and on the parameters of the nanodevice, including the height of the tunneling barrier. The nextnano software allows one to calculate the transmission from the lead, No.1 to the leads 3 and 4. The theoretical background involves the numerical solution of the Schroedinger equation and the Contact Block Reduction method.
Figure 13.2.9.2: An example of the geometry of a circuit containing two TCWs and one ABR. The circuit is also connected to four terminals.

Figure 13.2.9.3: Transmission

\[
E_z = \frac{\hbar^2 k_z^2}{2m_e^*} \\
\psi(z) = \exp(\pm ik_z z)
\]

13.3 3D Tutorials

13.3.1 Transmission through a 3D nanowire (CBR)

Author: Takuma Sato, nextnano GmbH

We apply the Contact Block Reduction (CBR) method to a simple GaAs nanowire of cuboidal shape. The corresponding tutorial for nextnano is [here](#).

System

We consider a GaAs cuboidal tube of dimensions 10 nm×10 nm×20 nm. Two leads of 10 nm×10 nm each are attached to the edge of the device. The grid spacing is 1 nm in all directions. The effective electron mass is assumed to be constant throughout the device and equal to 0.067 \( m_0 \).

Input file

To simulate 3D (or 2D) system with CBR method in nextnano++ correctly, the quantum regions have to be appropriately specified in the input file.

```plaintext
quantum{
    region{
        name "lead_1"
        x [-6,6]
        y [-6,6]
        z [-0.1,0.1]
        boundary x=dirichlet y=dirichlet z=cbr
        Gamma{ num_ev $num_eigenstates_device }
    }
}
```
The perpendicular directions, i.e. x- and y-directions, of the system are elongated by one grid due to the treatment of edge points in nextnano++. Since the simulation is three dimensional, the lead region specified here has to be two dimensional. The number ±0.1 is chosen to be smaller than the grid spacing, so that the region “lead_1” becomes a 2D sheet (Note: this is slightly different in nextnano³ input). CBR boundary condition has to be imposed in the propagation direction, i.e. z-direction, whereas Dirichlet boundary condition is set for perpendicular directions.

```plaintext

cbr{
  name = "device"
  lead( name = "lead_1" )
  lead( name = "lead_2" )
  delta_energy = $delta_energy
  abs_min_energy = $E_min
  abs_max_energy = $E_max
}
```

Here we specify the device region and leads attached to the device. The program calculates transmission through the region “device”, from “lead_1” to “lead_2”. The resolution, minimum and maximum of the energy axis can be also tuned here.

**CBR efficiency assessment**

The biggest advantage of the CBR method is that it can correctly predict the spectrum without calculating all eigenmodes of the 3D device. That means that, for low energies, one can significantly reduce the simulation load for the calculation of transmission spectrum Birner2009. To demonstrate it we perform three different simulations, sweeping the number of modes considered in the calculation. In the input file, the variable $CBR_case switches the number of eigenmodes.

```plaintext
$CBR_case = 1  # (ListOfValues:1,2,3)
$CBR_light = {($CBR_case-1)
$CBR_medium = {($CBR_case-2)
$CBR_heavy = {($CBR_case-3)
#if $CBR_light $num_eigenstates_device = 200          # 5.6% of all device
  -modes
#if $CBR_light $num_eigenstates_lead = 30            # 17.8% of all lead
  -modes
#if $CBR_medium $num_eigenstates_device = 400        # 11.3% of all device
  -modes
#if $CBR_medium $num_eigenstates_lead = 50           # 30.0% of all lead
  -modes
#if $CBR Heavy $num_eigenstates_device = 600         # 16.9% of all device
  -modes
#if $CBR Heavy $num_eigenstates_lead = 80            # 47.3% of all lead
  -modes
```

Figure 13.3.1.1 shows the calculated transmission coefficient as a function of energy. The result of nextnano³ is shown for reference. Arrows indicate the cutoff energies, namely the eigenenergy of the highest device mode considered in each simulation. The transmission coefficient drops when the energy exceeds the cutoff value. In the low energy, however, it is sufficient to calculate only a part of all eigenfunctions of the device Hamiltonian. Lower cutoff energy means lower dimension of matrices and vectors in the simulation, e.g. Eq.(36) in Birner2009, which reduces the calculation load. For example, a simulation performed at nextnano office took

- 42 sec for $CBR_case=1 (black)
- 3 min 14 sec for $CBR_case=2 (blue)
- 11min 17 sec for $CBR_case=3 (red)
Figure 13.3.1.1: Transmission coefficient of a GaAs 3D nanowire simulated with three different CBR parameters. \texttt{nextnano}$^3$ result is shown for reference. Arrows indicate the cutoff energies, namely the eigenenergy of the highest device eigenmode considered in each simulation.
Lead modes

The step-like increase of the transmission coefficient is attributed to the discrete energy levels of the lead modes. Let us have a close look at the first few steps. We can see that $T(E)$ increases by integers.

![Figure 13.3.1.2: Zoom into the first few steps of $T(E)$. The transmission increases by integer at the eigenenergies of the lead.](image)

The lead mode probability distribution $|\psi(x, y)|^2$ and corresponding eigenvalues are exported to the following files:

```
~\Quantum\wf_probabilities_lead_1_Gamma_0000.fld  ~\Quantum\wf_energy_spectrum_lead_1_Gamma_0000.dat
```

To see the energy eigenvalues, it is convenient to switch to Show Output File as Text (marked yellow).

Once the energy reaches 76 meV, the first lead mode energy is reached and then this mode transmits perfectly, giving a transmission of 1.

As can be seen from `\Quantum\wf_probabilities_lead_1_Gamma_0000.fld`, the second and third lead mode states are degenerate due to the symmetry of the lead cross-section. Thus they have the same energy 190 meV. Consequently, the spectrum increases by 2 at the energy of 190 meV. In this fashion, the step-like behaviour of the transmission coefficient is explained by lead eigenmodes.
Figure 13.3.1.3: The probability distribution $|\psi(x, y)|^2$ of the 2nd lead mode.
13.3.2 3D Quantum Dot Molecule

In this tutorial, we study two coupled quantum dots (QDs), i.e. two “artificial atoms” that form an “artificial molecule”. The two QDs are asymmetric and differ with respect to their height (4 nm and 6 nm).

With no electric field, the groundstates of both electron and hole are localized at the larger QD. By applying the electric field and increasing its strength, the hole groundstate becomes bonding state and then localizes at the smaller QD. At the same time the electron groundstate is still localized at the larger QD because of the weaker coupling between the two QDs due to the higher barrier height. We will see this leads to the change from an direct exciton to indirect exciton.

The relevant input files are as followings:

• 3DQD_molecule_cuboid_asymmetric_nn3.in / _nnp.in

Some of the material parameters that are used in this tutorial are based on the paper of

M. Grundmann, D. Bimberg
Formation of quantum dots in twofold cleaved edge overgrowth

i.e. they are the same as in the 3D CEO QD tutorial (apart from the effective masses).

Simulation

This simulation has the following features:

• We keep things simple by using cuboidal shaped GaAs QDs surrounded by Al$_{0.35}$Ga$_{0.65}$As barriers, i.e. we neglect strain and piezoelectric effects which is reasonable as the two materials GaAs and Al$_{0.35}$Ga$_{0.65}$As have pretty similar lattice constants.

• We also neglect the wetting layers and excitonic effects.

• In order to keep the CPU time to a minimum, we do not use the k.p approximation, i.e. we use for both electrons and the heavy hole a single-band effective mass approximation for the Schrödinger equation (parabolic and isotropic effective mass tensor). Nevertheless, this is sufficient to show some basic quantum physical effects of this QD molecule.

• We use different electron and hole masses in the barrier and well material, respectively.

• The left QD has the dimensions 10 nm x 10 nm x 4 nm (smaller dot). The right QD has the dimensions 10 nm x 10 nm x 6 nm (larger dot).

• The two QDs are separated by a 2 nm Al$_{0.35}$Ga$_{0.65}$As barrier.

• The grid resolution is 0.5 nm (rectangular tensor grid). This leads to a 3D Schrödinger matrix of dimension 50,225.

• We apply Dirichlet boundary conditions to the Schrödinger equation, i.e. the wave functions are allowed to penetrate the following distances into the barrier material (on each side): - along the x and y directions: 4 nm - along the z direction: 4.5 nm

• We vary the electric field along the growth direction (z axis) in steps of -2.5 kV/cm, i.e. from 0 kV/cm to -40 kV/cm.
Results

Electron and heavy hole ground states

The following figures show the square of the electron (left side) and heavy hole (right side) wave functions (isovolumes of 25% of $\psi^2$) for different applied electric fields (0 kV/cm, -17.5 kV/cm, -40 kV/cm). A slice through the conduction and valence band edges along the growth direction through the center of the QDs is also shown.

- Zero electric field

![Figure 13.3.2.1: electron ground state for zero electric field](image)

At zero applied electric field, both electron and heavy hole are located in the larger dot and form a **direct (bright) exciton**

- Electric field of -17.5 kV/cm

At an electric field of -17.5 kV/cm, the electron is still located in the larger dot on the right side, whereas the heavy hole forms a bonding (Figure 13.3.2.4) and an antibonding state (not shown) and is thus located in both wells (strong coupling). The exciton that is formed is something in between a direct and an indirect exciton.

- Electric field of -40 kV/cm

At an electric field of -40 kV/cm, the electron is still located in the larger dot on the right side, whereas the heavy hole ground state is now located in the left QD. An indirect (dark) exciton is formed. The exciton is called dark because the electron-hole overlap is much smaller and thus its oscillator strength (probability of optical transition) is much weaker (see Figure 13.3.2.15 below on spatial electron-hole overlap integrals).
Figure 13.3.2.2: heavy hole ground state for zero electric field

Figure 13.3.2.3: electron ground state for an electric field of -17.5 kV/cm
Figure 13.3.2.4: heavy hole ground state for an electric field of -17.5 kV/cm

Figure 13.3.2.5: electron ground state for an electric field of -40 kV/cm
Figure 13.3.2.6: heavy hole ground state for an electric field of -40 kV/cm

**Electron and heavy hole energies**

Figure 13.3.2.7 shows the electron energies of the ground state (e1) and the first excited electron state (e2) of the QD molecule. The ground state (e1) is always located in the larger QD (right side) whereas the first excited electron state (e2) is always located in the smaller QD (left side). The third and the forth eigenstate (e3, e4) are degenerate (not shown) because our QD molecule has a symmetry with respect to the x and y coordinates. They are always located in the right QD.

Figure 13.3.2.8 shows the heavy hole energies of the ground state (h1) and the excited hole states (h2, h3, h4, h5) of the QD molecule. In contrast to the electrons, the hole coupling between the two QDs is much stronger due to the smaller barrier height. At -17.5 kV/cm anticrossing between the states occur due to the formation of bonding and antibonding states (see Figure 13.3.2.9 to Figure 13.3.2.13 of the hole wave functions further below).
Figure 13.3.2.7: Electron energies

Figure 13.3.2.8: Heavy hole energies
Bonding and antibonding heavy hole state at anticrossing point

The following figures show the square of the lowest five hole wave functions (isovolumes of 2% of $\psi^2$) at an electric field of -17.5 kV/cm.

Figure 13.3.2.9: heavy hole ($h^1$) ground state at -17.5 kV/cm (bonding)

Electron-hole transition energies

The following figure shows the five lowest electron-hole transition energies of the QD molecule as a function of electric field. For fields smaller than -17.5 kV/cm a direct (bright) exciton is the ground state (both electron and hole wave function are located in the larger QD (right side), whereas for fields larger than -17.5 kV/cm an indirect (dark) exciton is the ground state where the electron is located in the larger QD (right side) and the hole is located in the smaller QD (left side). Therefore, the nature of the QD molecule ground state changes from direct to indirect.
Figure 13.3.2.10: heavy hole (h2) first excited state at -17.5 kV/cm (antibonding)

Figure 13.3.2.11: heavy hole (h3) state at -17.5 kV/cm (bonding)
Figure 13.3.2.12: heavy hole (h4) state at -17.5 kV/cm (bonding)

Figure 13.3.2.13: heavy hole (h5) state at -17.5 kV/cm (antibonding)
Electron-hole overlap

To understand the strength of the optical transitions we have to evaluate the matrix elements of the envelope functions, i.e. the spatial overlap integral over the electron and hole wave functions.

\[ \int \psi_{el,i}^* (x) \psi_{hl,j} (x) dx \]

Figure 13.3.2.15: Spatial electron-hole overlap integrals
13.3.3 3D Optics: absorption spectrum of a GaAs spherical quantum dot

Section author: Naoki Mitsui

This tutorial calculates the optical absorption spectrum of a GaAs spherical quantum dot with infinite barriers. We will see which output file we should refer to in order to understand the absorption spectrum.

Also, the formula used for the absorption calculation is presented. For the detailed scheme of the calculation of the optical matrix elements and absorption spectrum, please see our 1D optics tutorial: 1D Optics: Optical absorption for interband and intersubband transitions

Input file:

- 3Dspherical_infinite_dot_GaAs_intra_nnp.in

Structure

![Image](image.png)

Figure 13.3.3.1: Left: GaAs region as a spherical quantum dot. Right: Slice of the Gamma bandedge along $z = 0$. 

- Structure
- Scheme
- Results
  - Absorption
  - Eigenvalues, transition energies, and occupations
  - Transition intensity (Momentum matrix element)
  - Eigenstates
We model the infinite barrier by assigning 100 eV for the bandedge of AlAs barrier region from database{} section. Please see the input file for the details.

The parameters used in this simulation are as follows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>Value [unit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantum dot radius</td>
<td>( R )</td>
<td>5 [nm]</td>
</tr>
<tr>
<td>barrier height</td>
<td>( E_b )</td>
<td>92 [eV]</td>
</tr>
<tr>
<td>effective electron mass</td>
<td>( m_e )</td>
<td>0.0665</td>
</tr>
<tr>
<td>refractive index</td>
<td>( n_r )</td>
<td>3.3</td>
</tr>
<tr>
<td>doping concentration (n-type)</td>
<td>( N_D )</td>
<td>( 8 \cdot 10^{18} ) [cm(^{-3})]</td>
</tr>
<tr>
<td>linewidth (FWHM)</td>
<td>( \Gamma )</td>
<td>0.01 [eV]</td>
</tr>
<tr>
<td>temperature</td>
<td>( T )</td>
<td>300 [K]</td>
</tr>
</tbody>
</table>

**Scheme**

The run{} section is specified as follows:

```plaintext
run{
  poisson()
  quantum()
  optics()
}
```

Then the simulation follows these steps:

1. Poisson equation is solved with the setting specified in the poisson{} section.
2. “Schrödinger” equation is solved with the setting specified in the quantum{} section.
3. “Schrödinger” equation is solved again with the setting specified in the optics{} section and optical properties are calculated.

**Note:**

- If quantum_poisson{} is specified instead of quantum{}, Poisson and Schrödinger equations are solved self-consistently.
- optics{} requires that kp8 model is used in the quantum region specified in quantum{}.
- In this tutorial the kp parameters are adjusted so that the conduction and valence bands are decoupled from each other. Thus the single-band Schrödinger equations are solved effectively by the kp solver.

The optical absorption accompanied by the excitation of charge carriers (state \( n \rightarrow m \)) in a condensed matter is calculated on the basis of Fermi’s golden rule [ChuangOpto1995] in the dimension of (length\(^{-1}\)):

\[
\alpha(\tilde{\epsilon}, \omega) = \frac{\pi e^2}{n_s c \epsilon_0 m_0^2 \omega} \sum_{n > m} |\tilde{\epsilon} \cdot \tilde{\Pi}_{nm}|^2 (f_m - f_n) \mathcal{L}(E_n - E_m - \hbar \omega),
\]

where

- \( E_n \) is the energy of eigenstate \( n \). The first sum runs over the pair of states where \( E_n > E_m \).
- \( f_n \) is the occupation of eigenstate \( n \).
- \( \tilde{\epsilon} \) is the optical polarization vector defined in optics{} region{}.
\[ \vec{\pi} = \vec{\rho} + \frac{1}{4m_0c^2}(\sigma \times \nabla V) \]

where \( \vec{\rho} \) is the canonical momentum operator and \( \frac{1}{4m_0c^2}(\sigma \times \nabla V) \) is the contribution of spin-orbit interaction.

- \( \vec{\pi}_{nm} = \langle n | \vec{\pi} | m \rangle \).

- we call \( \vec{\epsilon} \cdot \vec{\pi}_{nm} \) as the optical matrix elements.

- \( \mathcal{L}(E_n - E_m - \hbar \omega) \) is the energy broadening function:
  - When energy\_broadening\_lorentzian is specified in optics\{ region\},
    \[ \mathcal{L}(E_n - E_m - \hbar \omega) = \frac{1}{\pi} \frac{\Gamma/2}{(E_n - E_m - \hbar \omega)^2 + (\Gamma/2)^2} \]
    where \( \Gamma \) is the FWHM defined by energy\_broadening\_lorentzian.
  - When energy\_broadening\_gaussian is specified in optics\{ region\},
    \[ \mathcal{L}(E_n - E_m - \hbar \omega) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(E_n - E_m - \hbar \omega)^2}{2\sigma^2}\right) \]
    where energy\_broadening\_lorentzian defines the FWHM \( \Gamma = 2\sqrt{\ln 2} \cdot \sigma \)
  - When neither energy\_broadening\_lorentzian nor energy\_broadening\_gaussian is specified in optics\{ region\}, \( \mathcal{L} \) is replace by the delta function \( \delta(E_n - E_m - \hbar \omega) \).

It is also possible to include both Lorentzian and Gaussian broadening (Voigt profile).

The detailed calculation scheme of the optical matrix elements \( \vec{\epsilon} \cdot \vec{\pi}_{nm} \) is described in 1D Optics: Optical absorption for interband and intersubband transitions. In 3D simulation we don’t have the k-summation like 1D and 2D cases.

Results

Absorption

Figure 13.3.3.2 shows the calculated \( \alpha(\vec{\epsilon}, E) \) specified in \texttt{Optics\_{absorption}\.dat} for x-polarization. The absorptions for y- and z-polarization are identical to this graph due to the rotational symmetry. We have one peak at around 0.23 eV (P1). These results can be understood from the output data explained below.

Note: When we use the realistic k.p parameters, \( \alpha(\vec{\epsilon}, E) \) for each polarization would no more be identical in general. This is because the eigenstates above the conduction band edge can have the component of valence band Bloch functions (band-mixing).

They are identical in this tutorial since the single-band model is emulated.
Figure 13.3.3.2: Calculated absorption spectrum $\alpha(\vec{c}, E)$ for $\vec{c} = \hat{x}$.

**Eigenvalues, transition energies, and occupations**

Figure 13.3.3.3 shows the calculated energy eigenvalues specified in `Quantum\energy_spectrum_.~.dat`.

Please note that the output in `Quantum` counts the eigenstates with different spins individually when k.p model is used, while they are counted jointly in `Optics`.

Comparing the excitation energy of other upper states to $k_B T \approx 0.026$ eV at $T = 300$ K, we can expect the occupation probability of each excited state is almost 0 and only the ground states have the non-zero occupation. Thus the optical transition will occur only from the ground states in this case.

We can see the peak energy of P1 in Figure 13.3.3.2 corresponds to the transition energy from the ground states (no. 1 and 2) to the 1st excited states (no. 3,4,5,6,7 and 8).

**Note:** The eigenstates with different spins are counted individually in `Quantum` when k.p model is used, while they are counted jointly in `Optics`.

For example, the two ground states counted as no.1 and 2 in Figure 13.3.3.3 due to spin are put together as one eigenstate in `Optics`.

From the above data of eigenvalues, we could see which pair of states contributes to the peak in the absorption spectrum Figure 13.3.3.2. In order to understand why some pairs of states don’t appear as peaks, we will see the output data for $|\vec{c} \cdot \vec{p}_{nm}|^2$ next.
Transition intensity (Momentum matrix element)

One of the key elements for the calculation of optical absorption is the transition intensity

\[ T_{nm}(\vec{\epsilon}) = \frac{2}{m_0} |\vec{\epsilon} \cdot \vec{\pi}_{nm}|^2 \]

which has the dimension of energy [eV].

The intensity \( T_{nm}(\vec{\epsilon}) \) for each pair of states \((n, m)\) is specified in *Optics\_transitions~.txt*. These intensities whose “From” states are the ground state are shown here for \( x\)-polarization. We can also check the transition energy of each pair of states.

<table>
<thead>
<tr>
<th>Energy [eV]</th>
<th>From</th>
<th>To</th>
<th>Radiative Rate [s]</th>
<th>Intensity_k0 [eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.233098</td>
<td>1</td>
<td>2</td>
<td>2.02882</td>
<td>4.43013e-08</td>
</tr>
<tr>
<td>0.233098</td>
<td>1</td>
<td>3</td>
<td>2.42777</td>
<td>3.70214e-08</td>
</tr>
<tr>
<td>0.233098</td>
<td>1</td>
<td>4</td>
<td>2.30413</td>
<td>3.90079e-08</td>
</tr>
</tbody>
</table>

The transitions from 1 to 5~10 are zero and these pairs of states don’t contribute to the absorption (They are omitted here since Intensity_k0 are too small).
Eigenstates

The probability distribution of eigenfunctions $|\psi(\mathbf{r})|^2$ is output in `Quantum/probabilities_.vtr`. The amplitude of the envelope function on each Bloch function $|S\rangle, |X\rangle, |Y\rangle, |Z\rangle$ can be found in `Quantum/amplitudes_~.SXYZ.vtr`. Here the probability distribution of eigenfunctions calculated by single-band model are shown.

Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.

13.4 Introduction to strain calculation

Here we introduce the theoretical background of the strain and stress calculation in nextnano++. At first we will describe the definition of a strain tensor $\varepsilon$ and stress tensor $\sigma$ and then describe the basis of strain tensor calculation in nextnano++. A strain tensor is used to calculate the shifts and splittings of band-edge energies and piezoelectric charges.

The detailed explanation for the syntax in `strain{}` is here: `strain{}`.

Related tutorials

- Piezoelectricity in wurtzite
- Wurtzite
- kp dispersion in bulk GaAs (strained / unstrained)
- kp dispersion in bulk unstrained, compressively and tensilely strained GaN (wurtzite)
- kp energy dispersion of an unstrained GaN QW embedded between strained AlGaN layers
- Energy levels in a pyramidal shaped InAs/GaAs quantum dot including strain and piezoelectric fields

Table of contents

- Strain tensor $\varepsilon$
- Stress tensor $\sigma$
- Strain and stress calculation
  - In general
  - In nextnano++

References

Figure 13.3.3.4: $|\psi|^2$ of the ground state. (s orbital, not degenerated.)

Figure 13.3.3.5: $|\psi|^2$ of the 1st excited state. (3 times degenerated, p orbital)
13.4.1 Strain tensor $\varepsilon$

The calculation of strain effects in nextnano++ is based on linear continuum elasticity theory, in which a crystal can be described by a field of material points with coordinates $x$. A distortion of the crystal shifts any point to a new position $x' = x'(x)$. A field of displacement vectors $u$ is defined as the deviation between the new position and the original position:

$$u(x) := x'(x) - x$$

A strain tensor $\varepsilon$ is defined using this displacement vector:

$$\varepsilon_{ij} := \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (i,j = 1,2,3)$$

Strain is dimensionless. The diagonal elements of this strain tensor $\varepsilon_{ii}$ represents the length changes per unit length in $x_i$-direction as described in Figure 13.4.1.2.

The off-diagonal elements $\varepsilon_{ij}(i \neq j)$ arise due to shear deformations of the crystal. Figure 13.4.1.3 shows the deformation of an infinitesimal rectangle in $x_1x_2$ plane. We can see $\frac{\partial u_2}{\partial x_1} = \frac{u_2(x_1, x_2+\Delta x_2) - u_2(x_1, x_2)}{\Delta x_2} = \sin \alpha \simeq \alpha$ and $\frac{\partial u_1}{\partial x_2} = \frac{u_1(x_1, x_2+\Delta x_2) - u_1(x_1, x_2)}{\Delta x_2} = \sin \beta \simeq \beta$. In these angle changes, $\frac{\alpha + \beta}{2}$ corresponds to a pure solid-body rotation and $\frac{\alpha - \beta}{2}$ measures the shear strain.

By definition strain tensor $\varepsilon$ is symmetric (i.e. $\varepsilon_{ij} = \varepsilon_{ji}$) so the number of components that must be specified is actually 6. Voigt notation is the useful convention in which these 6 independent components are written in form of a $6 \times 1$ matrix for short. This notation reads:

$$11 \rightarrow 1, 22 \rightarrow 2, 33 \rightarrow 3, 23 \rightarrow 4, 31 \rightarrow 5, 12 \rightarrow 6$$

and

$$\begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{pmatrix} = \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \\ 2\varepsilon_{12} \end{pmatrix}$$
Figure 13.4.1.2: Deformation of a dilatable string in an unstrained (top) and strained state (bottom). We can see the diagonal element $\varepsilon_{ii} = \frac{\partial u_i}{\partial x_i}$ represents the length changes per unit length in $x_i$-direction.

Figure 13.4.1.3: Deformation of an infinitesimal rectangle in a strained state.
13.4.2 Stress tensor \( \sigma \)

A stress tensor component \( \sigma_{ij} \) represents the force towards \( x_j \)-direction acting on infinitesimal area that is perpendicular to \( x_i \)-direction. Its unit is the same with pressure (\([\text{Pa}] = [\text{N/m}^2]\)).

![Image](image.png)

Figure 13.4.2.1: The components of stress tensor \( \sigma \).

In linear approximation, this stress tensor is related to the strain tensor \( \varepsilon \) by means of Hook’s law:

\[
\sigma_{ij} = \sum_{kl} C_{ijkl} \varepsilon_{kl}
\]

where \( C_{ijkl} \) is the component of elasticity stiffness tensor, which is the forth-order tensor comprising \( 3^4 = 81 \) components. It’s dimension is the same with stress tensor components and defined as \([\text{GPa}]\) in nextnano++. In Voigt notation, \( C \) is the form of a \( 6 \times 6 \) matrix by putting \( C_{ijkl} = C_{mn} \ (i, j, k, l = 1, 2, 3, \ m, n = 1, \ldots 6) \). Then the Hook’s law reads

\[
\begin{pmatrix}
\sigma_1 \\
\sigma_2 \\
\sigma_3 \\
\sigma_4 \\
\sigma_5 \\
\sigma_6
\end{pmatrix} =
\begin{pmatrix}
C_{11} & C_{12} & 0 & 0 & 0 & 0 \\
C_{13} & C_{14} & 0 & 0 & 0 & 0 \\
C_{15} & C_{16} & 0 & 0 & 0 & 0 \\
C_{21} & C_{22} & 0 & 0 & 0 & 0 \\
C_{23} & C_{24} & 0 & 0 & 0 & 0 \\
C_{25} & C_{26} & 0 & 0 & 0 & 0 \\
C_{31} & C_{32} & 0 & 0 & 0 & 0 \\
C_{33} & C_{34} & 0 & 0 & 0 & 0 \\
C_{35} & C_{36} & 0 & 0 & 0 & 0 \\
C_{41} & C_{42} & 0 & 0 & 0 & 0 \\
C_{43} & C_{44} & 0 & 0 & 0 & 0 \\
C_{45} & C_{46} & 0 & 0 & 0 & 0 \\
C_{51} & C_{52} & 0 & 0 & 0 & 0 \\
C_{53} & C_{54} & 0 & 0 & 0 & 0 \\
C_{55} & C_{56} & 0 & 0 & 0 & 0 \\
C_{61} & C_{62} & 0 & 0 & 0 & 0 \\
C_{63} & C_{64} & 0 & 0 & 0 & 0 \\
C_{65} & C_{66} & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\varepsilon_1 \\
\varepsilon_2 \\
\varepsilon_3 \\
\varepsilon_4 \\
\varepsilon_5 \\
\varepsilon_6
\end{pmatrix}
\]

For many crystal structures with high symmetry, many of these coefficients are 0 and some are related to others.
The elasticity tensor of zincblende and wurtzite crystals are given by

\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{12} & C_{11} \\
C_{12} & C_{11} \\
C_{12} & C_{12} \\
\end{pmatrix}
\]

with \( C_{66} = \frac{1}{2}(C_{11} - C_{22}) \) in wurtzite.

These constants are defined in `database_nnp.in`. You can also overwrite these values in your input file.

- For zinc-blend materials, for example:

  ```
  database{
    binary_zb{
      name = GaAs
      valence = III_V

      elastic_consts{
        c11 = 122.1 # [GPa] elastic constants
        c12 = 56.6 # 1 * 1011 dyn/cm2 = 10 GPa
        c44 = 60.0 # The elastic constants are needed for the calculation of the strain in heterostructures.
      }
    }
  }
  ```

- For wurtzite materials, for example:
13.4.3 Strain and stress calculation

Next we will describe how the strain tensor $\varepsilon$ and stress tensor $\sigma$ are determined in general. Then the two types of calculation implemented in nextnano++ are introduced briefly.

In general

The principle of conservation of linear momentum results in the following equations of stress tensor components for $i = 1, 2, 3$:

$$\sum_{j=1}^{3} \frac{\partial \sigma_{ji}}{\partial x_j} + f_i = 0$$

where $f$ is the body force such as gravity. When the boundary conditions are specified, the field of displacement vector $u$, by which the stress tensor components $\sigma_{ij}$ are eventually written, is determined according to these simultaneous differential equations. Then the strain tensor $\varepsilon$ and stress tensor $\sigma$ are also determined from $u$.

| Note: The principle of conservation of angular momentum, on the other hand, results in the symmetricity of stress tensor: $\sigma_{ij} = \sigma_{ji}$.

The field of displacement vector which satisfies the above balance equations and boundary conditions also minimizes the total potential energy $U + V_E$ where $U$ is the elastic strain energy and $V_E$ is the potential energy associated with the body force $f$. This is so called minimum total potential energy principle.

In the linear approximation regime, the elastic energy stored in the whole body is:

$$U = \frac{1}{2} \int_V C_{ijkl} \varepsilon_{ij} \varepsilon_{kl} \, dV$$

When the body force $f$ is assumed to be zero throughout the system, solving the above differential equations is equivalent to find the strain tensor that minimizes this elastic energy $U$. 13.4. Introduction to strain calculation
In nextnano++

There are two kinds of calculation of strain, `pseudomorphic_strain()` and `minimized_strain()`, in nextnano++. In both of implementations pseudomorphic layer is assumed as the boundary condition between the substrate and the layer grown on this substrate. The substrate is assumed to be so thick that the in-plane lattice constants of the layer is matched to that of substrate. Also, the body force \( f \) is assumed to be 0 throughout the structure.

In this assumption, the analytic expressions for strain tensor that satisfies the aforementioned stress balance equations (i.e. that minimizes the elastic energy) can be found for 1D structures. This analytic solution is implemented on `pseudomorphic_strain()`. This feature also works in 2D or 3D but the user must be sure that the model makes sense from a physical point of view (i.e. the 2D/3D structure should consist of different layers along the growth direction whereas the layers must be homogenous along the two perpendicular directions).

On the other hand, `minimized_strain()` calculates the strain tensor by minimizing the elastic energy mentioned before. This can also be used for 1D simulations. In this case, the results will be equivalent to the analytical model `pseudomorphic_strain()`.

The detailed explanation for the syntax in `strain()` is here: `strain()`. Please refer to T.Andauer’s PhD thesis for more details about these topics.

Please help us to improve our tutorial! Should you have any questions and comments, please send to support [at] nextnano.com.

13.5 Workshop

A few nextnano workshops were held in the past using the material presented below. The following tutorials are suited to learn more about the nextnano software.

- pn junction
- Double Quantum Well
- Multiple quantum wells and finite superlattices
- Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider’s code
- InGaAs Multi-Quantum Well Laser diode
- Optical interband transitions in a quantum well - Matrix elements and selection rules
- Wurtzite
- Two-dimensional electron gas in an AlGaN/GaN field effect transistor
- \( \mathbf{k} \cdot \mathbf{p} \) dispersion in bulk GaAs (strained / unstrained)
- \( \mathbf{k} \cdot \mathbf{p} \) dispersion in bulk unstrained ZnS, CdS, CdSe and ZnO (wurtzite)
- \( \mathbf{k} \cdot \mathbf{p} \) dispersion in bulk unstrained, compressively and tensilely strained GaN (wurtzite)
- \( \mathbf{k} \cdot \mathbf{p} \) energy dispersion of holes in a quantum well
- \( \mathbf{k} \cdot \mathbf{p} \) energy dispersion of an unstrained GaN QW embedded between strained AlGaN layers
- Capacitance-Voltage curve of a metal–insulator–semiconductor (MIS) structure
- Core-shell nanowire
13.5.1 pn junction

*recommended*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_pn_junction.htm

\[ \text{pn\_junction\_GaAs\_1D*\.in (nextnano++ & nextnano)} \]
\[ \text{pn\_junction\_GaAs\_2D*\.in (nextnano++ & nextnano)} \]

**Summary**  This example is rather simple.

**Model used**  1D; Poisson equation

**Task**  Try to reproduce the results in the figures. Run and plot the 2D files.

**Challenge**  To plot two different graphs, i.e. data files, simultaneously. Export the graph containing two different graphs to gnuplot. Export a 2D file to gnuplot.

**Duration**  10 minutes

13.5.2 Double Quantum Well

*recommended*

*Double Quantum Well*

\[ \text{DoubleQuantumWell\_*.in (nextnano++ & nextnano)} \]

**Summary**  This example teaches quantum physics: bonding and anti-bonding wavefunctions.

**Model used**  1D; Schrödinger equation

**Task**  Try to reproduce the results in the figures.

**Challenge**  Perform a parameter sweep using nextnanomat ’s Template feature and do a post-processing using nextnanomat to reproduce the figure *Eigenvalues vs. barrier width*. Export the graph to gnuplot.

**Duration**  20 minutes

13.5.3 Multiple quantum wells and finite superlattices

*MQW and finite superlattices*

\[ \text{Superlattice\_N\_wells\_nnp.in (nextnano++)} \]

**Summary**  Covers sweeping variables. The transition between a finite superlattice and a multiple quantum well system is observed by changing the number of wells in the system.

**Model used**  1D; Schrödinger-Poisson

**Task**  Try to reproduce the structure of the system and the results in the figures
Challenge Perform a parameter sweep with nextnanomat or nextnanopy. If the latter is chosen, perform post-processing using nextnanopy to match Harrison’s figures.

Duration 30 minutes

13.5.4 Schrödinger-Poisson - A comparison to the tutorial file of Greg Snider’s code

recommended

Schrödinger-Poisson

Greg_Snider_MANUAL_*\.in (nextnano++ & nextnano³)

Summary This example is relatively easy.

Model used 1D; self-consistent Schrödinger-Poisson

Task Try to reproduce the results in the figures.

Challenge To plot two different graphs, i.e. data files, simultaneously. Understand where you find eigenvalues and wavefunctions. Export the graph containing two different graphs to gnuplot.

Duration 20 minutes

13.5.5 InGaAs Multi-Quantum Well Laser diode

very important

Laser diode

LaserDiode_InGaAs_1D_cl_nnp.in (nextnano++)
LaserDiode_InGaAs_1D_qm_nnp.in (nextnano++)

Summary This example teaches how to apply a bias and solve the coupled system of Schrödinger, Poisson and Current equations.

Model used 1D; selfconsistent Schrödinger–Poisson–current

Task Try to reproduce some of the figures.

Challenge To understand how a laser works. Plot the recombination rates and the classical emission spectrum. Do a parameter sweep, e.g. number of wells, doping, alloy content and see how the classical emission spectrum changes. Plot the wavefunctions. Compare the classical density vs. the quantum density for the same bias, e.g. for 1 V where the quantum wells contain a significant density. Plot the convergence log files on a logarithmic scale. Plot the current densities, IV curve, doping profile, . . .

Duration 40 minutes
13.5.6 Optical interband transitions in a quantum well - Matrix elements and selection rules

optional
https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_OpticalTransitions.htm

1DQW_interband_matrixelements_*.in (nextnano++ & nextnano³)

Summary  This example teaches how to calculate the spatial overlap of electron and hole wavefunctions and their transition energy.
Model used  1D; Schrödinger equation
Task  Try to reproduce the results in the figures.
Challenge  Compare the infinite vs. finite quantum well to understand selection rules.
Duration  10 minutes

13.5.7 Wurtzite

optional
https://www.nextnano.com/nextnano3/tutorial/1Dtutorial11.htm

wurtzite*.in (nextnano++ & nextnano³)

Summary  This example is relatively easy and discusses strain, piezo any pyroelectricity.
Model used  1D; strain and Poisson equation
Task  Try to reproduce the results in the figures.
Challenge  To understand the peculiarities of wurzite.
Duration  20 minutes

13.5.8 Two-dimensional electron gas in an AlGaN/GaN field effect transistor

optional
https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_AlGaN_GaN_FET.htm

Jogai_AlGaNGaN_FET_JAP2003*.in (nextnano++ & nextnano³)

Summary  This example is nice. Try to reproduce some of the figures.
Model used  1D; strain, selfconsistent Schrödinger–Poisson
Task  Try to reproduce the results in the figures.
Challenge  Can you do a parameter sweep using nextnanoMat 's Template feature and plot the 2DEG density vs. AlₓGa₁₋ₓN thickness?
Duration  20 minutes
13.5.9 \( \mathbf{k} \cdot \mathbf{p} \) dispersion in bulk GaAs (strained / unstrained)

*advanced*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial13.htm

bulk\_kp\_dispersion\_GaAs*.in (nextnano++ & nextnano³)

**Summary** This example teaches the \( \mathbf{k} \cdot \mathbf{p} \) band structure.

**Model used** bulk; strain, \( \mathbf{k} \cdot \mathbf{p} \)

**Task** Try to reproduce some of the figures: Plot the single-band, the 6-band dispersion and the 8-band \( E(k) \) dispersion in the same plot. Do the same for the strained case to see how strain alters the band structure.

**Challenge** To understand the \( \mathbf{k} \cdot \mathbf{p} \) method.

**Duration** 15 minutes

13.5.10 \( \mathbf{k} \cdot \mathbf{p} \) dispersion in bulk unstrained ZnS, CdS, CdSe and ZnO (wurtzite)

*advanced*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_bulk_6x6kp_dispersion_IIVI.htm

bulk\_6x6kp\_dispersion\_ZnO*.in (nextnano++ & nextnano³)

**Summary** This example teaches the \( \mathbf{k} \cdot \mathbf{p} \) valence band structure for wurtzite materials.

**Model used** bulk; strain, \( \mathbf{k} \cdot \mathbf{p} \)

**Task** Try to reproduce some of the figures.

**Challenge** To understand the \( \mathbf{k} \cdot \mathbf{p} \) method for wurtzite.

**Duration** 5 minutes

13.5.11 \( \mathbf{k} \cdot \mathbf{p} \) dispersion in bulk unstrained, compressively and tensilely strained GaN (wurtzite)

*advanced*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_strained_GaN_dispersion.htm

bulk\_kp\_dispersion\_GaN\_unstrained*.in (nextnano³)

**Summary** This example teaches the \( \mathbf{k} \cdot \mathbf{p} \) valence band structure for wurtzite materials.

**Model used** bulk; strain, \( \mathbf{k} \cdot \mathbf{p} \)

**Task** Try to reproduce some of the figures.

**Challenge** To understand the \( \mathbf{k} \cdot \mathbf{p} \) method for wurtzite.

**Duration** 10 minutes
13.5.12  \( k \cdot p \) energy dispersion of holes in a quantum well

*advanced*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial8.htm

1Dwell_GaAs_AlAs_*.in (nextnano++ & nextnano³)

**Summary**  This example teaches the \( k \cdot p \) model: \( E(k_z) \) dispersion.

**Model used**  1D; \( k \cdot p \) Schrödinger equation

**Task**  Try to reproduce some of the figures.

**Challenge**  To understand the \( k \cdot p \) features for heterostructures.

**Duration**  20 minutes

13.5.13  \( k \cdot p \) energy dispersion of an unstrained GaN QW embedded between strained AlGaN layers

*advanced*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_GaN_AlGaN_QW_dispersion.htm

1DGaN_AlGaN_QW_k_zero_*.in (nextnano++ & nextnano³)

**Summary**  This example teaches the \( k \cdot p \) model: \( E(k_z) \) dispersion.

**Model used**  1D; \( k \cdot p \) Schrödinger equation

**Task**  Try to reproduce some of the figures.

**Challenge**  To understand the \( k \cdot p \) features for heterostructures.

**Duration**  20 minutes

13.5.14 Capacitance-Voltage curve of a metal–insulator–semiconductor (MIS) structure

*advanced*

https://www.nextnano.com/nextnano3/tutorial/1Dtutorial_MIS_CV.htm

1DMIS_CV_Fermi_*.in (nextnano++ & nextnano³)

**Summary**  This example teaches how to apply a bias without solving the current equation.

**Model used**  1D; selfconsistent Schrödinger–Poisson

**Task**  Try to reproduce some of the figures.

**Challenge**  To understand how to integrate charge carrier densities in specific regions.

**Duration**  20 minutes
13.5.15 Core-shell nanowire

recommened

Schrödinger equation of a two-dimensional core-shell structure

Hexagonal 2DEG - Two-dimensional electron gas in a delta-doped hexagonal shaped GaAs/AlGaAs nanowire heterostructure

https://www.nextnano.com/nextnano3/tutorial/2Dtutorial_core_shell_circle_hexagon.htm

2DGaAs_AlgAs_*_.in (nextnano++ & nextnano³)
2D_Hexagonal_Nanowire_2DEG*_.in (nextnano++ & nextnano³)

Summary This example teaches how to perform a 2D simulation.

Model used 2D; Schrödinger equation, selfconsistent Schrödinger–Poisson

Task Try to reproduce some of the figures.

Challenge To understand how to visualize 2D results and how to export them to gnuplot. Plot the geometry together with the electron density of the modulation doped core-shell nanowire in one graph.

Duration 15 minutes

13.5.16 Quantum dot molecule

advanced

QD molecule

3DQD_molecule_cuboid_asymmetric_*.in (nextnano++ & nextnano³)

Summary This example teaches how to apply an electric field in a 3D simulation.

Model used 3D; Schrödinger equation

Task Try to reproduce some of the figures.

Challenge To understand how to visualize 3D results and how to export them to Paraview.

Duration 10 minutes

13.5.17 Energy levels in a pyramidal shaped InAs/GaAs quantum dot including strain and piezoelectric fields

advanced

https://www.nextnano.com/nextnano3/tutorial/3Dtutorial_QD_pyramid.htm

3DInAsGaAsQDPyramid_PryorPRB1998_10nm_*.in (nextnano++ & nextnano³)

Summary This example teaches how to calculate and plot strain, piezoelectric charge densities and wavefunctions in a 3D simulation.

Model used 3D; strain equation, Poisson equation, Schrödinger equation

Task Try to reproduce some of the figures.
**Challenge**  To understand how to visualize 3D results and how to export them to Paraview.

**Duration**  20 minutes

13.5.18 Single-electron transistor - laterally defined quantum dot

*advanced*

https://www.nextnano.com/nextnano3/tutorial/3Dtutorial_SET_lateral_QD.htm

SET_Scholze_IEEE2000_*.in (nextnano++ & nextnano³)

**Summary**  This example teaches how a gate geometry depletes the 2DEG density locally.

**Model used**  1D, 3D; Poisson equation, Schrödinger equation, selfconsistent Schrödinger–Poisson

**Task**  Try to reproduce some of the figures.

**Challenge**  To understand how to visualize 3D results and how to export them to Paraview.

**Duration**  15 minutes

---

**Note:**  Search through

- the other input files in the installation folder,
- the list of tutorials on the website, https://www.nextnano.com/nextnano3/tutorial/tutorial.htm

and simulate the topics that are of interest for you.

---

**Important nextnanomat features that you should learn**

- Tree View vs. List View
- Parameter sweeps using Template
- Parameter sweep post-processing using Template
- Postprocessing using nextnanopy
- Multi-Parameter sweeps using Template (beta)
- Batch list
- Exporting 2D slices of 3D data
- Exporting 1D slices of 2D/3D data
- Export to .vtr
- Exporting 3D data to Paraview
- Open files with notepad++, Origin, …
- Plotting several graphs (Overlay)
- Plotting several graphs in 2D/3D (Overlay)
- Zoom feature
- SOFT_KILL feature
- Display options
- Tools Options
- HTCondor Cloud Computing
• In principle you can run also your own software with nextnanomat. nextnanomat can run any executable either locally or on HTCondor.

• How to access online documentation and how it is structured.

• Which features are you missing?

If you have any comments or suggestions regarding this workshop material, please send your feedback to support [at] nextnano.com. We really appreciate your feedback on errors, broken links, typos, …

13.6 General scheme of the optical device analysis

Here we summarize the models and equations that is used for the optical device analysis in nextnano++.  

Table of contents

• Related tutorials
  
• Determination of carrier densities and current densities
    – Quantum mechanical calculation of charge carrier densities
      * Multi-band model (*k · p model)
      * Single-band model
    – Classical calculation of charge carrier densities
    – Poisson equation
      * Poisson equation
      * Ionized donor/acceptor densities
      * Piezoelectric and pyroelectric charge densities
    – Current equation
      * Current equation
      * Recombination/Generation
  
• Optoelectronic characteristics based on the semi-classical model

• Optoelectronic characteristics based on the quantum model

• References

13.6.1 Related tutorials

• Semi-classical model
  – GaAs Solar Cell
  – 1D InGaAs Multi-quantum well laser diode
  – UV LED: Quantitative evaluation of the effectiveness of EBL
  – UV LED: Quantitative evaluation of the effectiveness of superlattice structure in p-region

• Quantum model
  – 1D Optics: Optical absorption for interband and intersubband transitions
  – tutorial_gain_InGaAs_QW
  – tutorial_gan_gain
13.6.2 Determination of carrier densities and current densities

Quantum mechanical calculation of charge carrier densities

Multi-band model (k ⋅ p model)

Once the \( \mu \)-th component envelope function of the \( j \)-th eigenstate of electron \((l = c)\) or hole \((l = v)\) in the \( i \)-th band is obtained as \( (F_\mu)_l^{i,j}(x) \) from the multi-band Schrödinger equation, the probability distribution of this \( j \)-th eigenstate reads

\[
p_i^{l,j}(x) = \sum_{\mu} \left| (F_\mu)_l^{i,j}(x) \right|^2. \tag{13.6.2.1}
\]

where we are assuming 3D structure so far.

Then the quantum mechanical carrier densities for 3D structure are defined from these probability densities, energy eigenvalues \( E_{c,j} \) and \( E_{v,j} \), position-dependent quasi-Fermi levels \( E_{F,n}(x) \) and \( E_{F,p}(x) \) as

\[
n(x) = \sum_{i \in CB} g_c^i \sum_j p_{c,j}^i(x) f \left( \left[ E_{c,j}^i - E_{F,n}(x) \right]/kT \right) \tag{13.6.2.2}
\]
\[
p(x) = \sum_{i \in VB} g_v^i \sum_j p_{v,j}^i(x) f \left( \left[ -E_{v,j}^i + E_{F,n}(x) \right]/kT \right) \tag{13.6.2.3}
\]

where \( f(E) \) is the Fermi-Dirac distribution at temperature \( T \). \( g_c^i \) and \( g_v^i \) represent the possible spin and valley degeneracies.

When the simulation is over 1D structure, the wavefunction can be separated into the plane wave specified with the lattice wave vector \( k_\parallel \) in the lateral 2D direction and the quantized wavefunction in the growth direction, which has the \( k_\perp \)-dependency. Then the charge carrier density is obtained by the following integral over \( k_\parallel \):

\[
n(x) = \sum_{i \in CB} g_c^i \sum_j \frac{1}{(2\pi)^2} \int_{\Omega_{BZ}} d^2k_\parallel p_{c,j}^i(x, k_\parallel) f \left( \left[ E_{c,j}^i(k_\parallel) - E_{F,n}(x) \right]/kT \right) \tag{13.6.2.4}
\]
\[
p(x) = \sum_{i \in VB} g_v^i \sum_j \frac{1}{(2\pi)^2} \int_{\Omega_{BZ}} d^2k_\parallel p_{v,j}^i(x, k_\parallel) f \left( \left[ -E_{v,j}^i(k_\parallel) + E_{F,n}(x) \right]/kT \right) \tag{13.6.2.5}
\]

Here the integration is over the two-dimensional Brillouin zone \( \Omega_{BZ} \).

Similarly, the charge carrier densities for 2D structure is calculated by the integral over the 1-dimensional Brillouin zone as

\[
n(x) = \sum_{i \in CB} g_c^i \sum_j \frac{1}{2\pi} \int_{\Omega_{BZ}} dk_\parallel p_{c,j}^i(x, k) f \left( \left[ E_{c,j}^i(k) - E_{F,n}(x) \right]/kT \right) \tag{13.6.2.6}
\]
\[
p(x) = \sum_{i \in VB} g_v^i \sum_j \frac{1}{2\pi} \int_{\Omega_{BZ}} dk_\parallel p_{v,j}^i(x, k) f \left( \left[ -E_{v,j}^i(k) + E_{F,p}(x) \right]/kT \right) \tag{13.6.2.7}
\]
Single-band model

Things are simpler.

When the single-band Schrödinger equation is set to be solved, the envelope function of the \( j \)-th eigenstate has only one component \( F_{i,j}(x) \). Also, the \( k \)-integration in (13.6.2.4) to (13.6.2.7) can be done analytically due to the parabolic dispersion according to the effective mass tensor \( m^\ast_i \) and \( m_i^\ast \).

Thanks to this simplicity the quantum mechanical charge carrier densities for \( d \)-dimentional simulation can be written up by the following expression:

\[
\begin{align*}
n(x) &= \sum_{i\in CB} g^i_c \left( \frac{m_{\text{dos},c} k T}{2 \pi \hbar^2} \right)^{(3-d)/2} \sum_j p^{i,j}_c(x) F_{(1-d)/2} \left( [E^i_{c,j} - E_{F,n}(x)]/k T \right) \\
p(x) &= \sum_{i\in VB} g^i_v \left( \frac{m_{\text{dos},v} k T}{2 \pi \hbar^2} \right)^{(3-d)/2} \sum_j p^{i,j}_v(x) F_{(1-d)/2} \left( [-E^i_{v,j} + E_{F,p}(x)]/k T \right)
\end{align*}
\]

(13.6.2.8) (13.6.2.9)

TODO: The sign in the fermi-dirac integral might be opposite. check the source code.

Here \( F_n(E) \) denotes the Fermi-Dirac integral of order \( n \) and \( m^\ast_i \) is so-called density-of-states mass defined as

\[
m^\ast_i = (\det m^\ast_i) \quad \lambda = e, h
\]

(13.6.2.10)

where \( m^\ast_i \) describes the \( 2 \times 2 \) or \( 1 \times 1 \) submatrix of the effective mass tensor \( m_i^\ast \) in the direction of \( k_i \).

In any cases, the carrier densities are dependent on the electrostatic potential \( \phi(x) \) through the wave function, which is obtained from the \( \phi \)-dependent Hamiltonian \( H(\phi) \). Thus we can also write them as \( n(x, \phi) \) and \( p(x, \phi) \), which enters into the non-linear Poisson equation introduced later.

Moreover, when the current equation is included in the calculation scheme, seeing the carrier densities as \( n(x, \phi, E_{F,n}) \) and \( p(x, \phi, E_{F,p}) \) makes it easy to understand what the self-consistent calculation is actually doing.

Classical calculation of charge carrier densities

Things are much more simpler.

When any kind of Schrödinger equation is not solved, the charge carrier densities are estimated from the position-dependent conduction and valence bandedges \( E^c_i(x) \) and \( E^v_i(x) \), quasi-Fermi levels, and the electrostatic potential \( \phi(x) \) in the context of Thomas-Fermi approximation.

These classical charge carrier densities are calculated as

\[
\begin{align*}
n(x) &= \sum_{i\in CB} N^c_i(T) F_{1/2} \left( [-E^c_i(x) + e\phi(x) + E_{F,n}(x)]/k T \right) \\
p(x) &= \sum_{i\in VB} N^v_i(T) F_{1/2} \left( [E^v_i(x) - e\phi(x) - E_{F,p}(x)]/k T \right)
\end{align*}
\]

(13.6.2.11) (13.6.2.12)

Here \( N^c_i(T) \) and \( N^v_i(T) \) are the equivalent density of states at the conduction and valence bandedges, which are given by

\[
N^\lambda_i(T) = g^i_\lambda \left( \frac{m^\ast_{\text{dos},\lambda} k T}{2 \pi \hbar} \right)^{2/3} \quad (\lambda, \lambda) = (v,h), \text{ or } (c,e).
\]

(13.6.2.13)

Here \( m^\ast_i \) is the density-of-mass for \( d=3 \) defined in (13.6.2.10).

This calculation of carrier densities is much faster than the quantum mechanical calculation, but the quantum effect such as energy quantization, carrier leakage into the barrier, etc. cannot be taken into account.

Also in this case, the carrier densities can be written as \( n(x, \phi) \) and \( p(x, \phi) \), which enters into the non-linear Poisson equation introduced next.

Moreover, when the current equation is included in the calculation scheme, seeing the carrier densities as \( n(x, \phi, E_{F,n}) \) and \( p(x, \phi, E_{F,p}) \) makes it easy to understand what the self-consistent calculation is actually doing.
Poisson equation

This equation governs the relation between the electrostatic potential $\phi(x)$ and total charge density distribution $\rho(x, \phi)$ as follows:

$$-\nabla \cdot [\varepsilon_0 \varepsilon_r(x) \nabla \phi(x)] = \rho(x, \phi)$$

(13.6.2.14)

where $\varepsilon_0$ is the vacuum permittivity, $\varepsilon_r$ is the material dependent static dielectric constant. And the total charge density distribution consists of the densities of ionized donors $N_D^+$, ionized acceptors $N_A^-$, piezoelectric and pyroelectric charge $\rho_{pz}$ and $\rho_{py}$, besides the carrier densities $n(x, \phi)$ and $p(x, \phi)$, which are calculated either classically or quantum mechanically:

$$\rho(x, \phi) = e[-n(x, \phi) + p(x, \phi) + N_D^+(x) - N_A^-(x) + \rho_{pz}(x) + \rho_{py}(x)]$$

(13.6.2.15)

When the Schrödinger-Poisson equation is solved, i.e. quantum_poisson{} is specified in run{} section, the carrier densities defined in either multi-band model or single-band model are substituted into this Poisson equation and the carrier densities are calculated once again.

This cycle is continued until the carrier densities satisfies the convergence criteria, which can be tuned by the users from run{} section. The final result of $n(x, \phi), p(x, \phi)$ and $\phi(x)$ must satisfy both Schrödinger and Poisson equations, or we can say that the Schrödinger equation and Poisson equation are self-consistent with respect to the resulting carrier densities and electrostatic potential.

On the other hand, when only the Poisson equation is solved, i.e. only poisson{} is specified run{} section, the carrier densities are calculated according to (13.6.2.11) and (13.6.2.12) instead. We can say in other words that the Schrödinger equation and Poisson equation are self-consistent with respect to the resulting carrier densities and electrostatic potential.

Ionized donor/acceptor densities

The densities of ionized impurities are calculated in the context of Thomas-Fermi approximation with these formula:

$$N_D^+(x) = \sum_{i \in \text{Donors}} \frac{N_{D,i}(x)}{1 + g_{D,i} \exp(\frac{(E_{F,n}(x) - E_{D,i}(x))/k_BT)}$$

(13.6.2.16)

$$N_A^-(x) = \sum_{i \in \text{Acceptors}} \frac{N_{A,i}(x)}{1 + g_{A,i} \exp(\frac{(E_{F,p}(x) - E_{A,i}(x))/k_BT)}$$

(13.6.2.17)

where the summation is over all different donor or acceptors, $N_D, N_A$ are the doping concentrations, $g_D, g_A$ are the degeneracy factors ($g_D = 2$ and $g_A = 4$ for shallow impurities), and $E_D, E_A$ are the energies of the neutral donor and acceptor impurities, respectively.

These energies of neutral impurities $E_{D,i}, E_{A,i}$ are determined by the ionization energies $E_{D,i}^{\text{ion}}, E_{A,i}^{\text{ion}}$, the bulk conduction and valence bandedges (including shifts due to strain) and the electrostatic potential.

$$E_{D,i}(x) = E_c(x) - e\phi(x) - E_{D,i}^{\text{ion}}(x)$$

(13.6.2.18)

$$E_{A,i}(x) = E_v(x) - e\phi(x) + E_{A,i}^{\text{ion}}(x)$$

(13.6.2.19)
Piezoelectric and pyroelectric charge densities

$\rho_{pz}$ and $\rho_{py}$ are calculated according to the result of strain equation. (TO be updated)

Current equation

The continuity equations in the presence of creation (generation, $G$ ) or annihilation (recombination, $R$ ) of electron-hole pairs read

\[
-e \frac{\partial n}{\partial t} + \nabla \cdot (-e j_n(x)) = -e(G(x) - R(x)),
\]

\[
e \frac{\partial p}{\partial t} + \nabla \cdot e j_p(x) = e(G(x) - R(x)),
\]

where the current is proportional to the gradient of quasi Fermi levels $E_{F,n/p}(x)$

\[
\begin{align*}
j_n(x) &= -\mu_n(x)n(x)\nabla E_{F,n}(x), \\
j_p(x) &= \mu_p(x)p(x)\nabla E_{F,p}(x).
\end{align*}
\]

Hereafter we consider stationary solutions and set $\dot{n} = \dot{p} = 0$. The governing equations then reduce to

\[
\begin{align*}
\nabla \cdot \mu_n(x)n(x)\nabla E_{F,n}(x) &= -(G(x) - R(x)), \\
\nabla \cdot \mu_p(x)p(x)\nabla E_{F,p}(x) &= G(x) - R(x),
\end{align*}
\]

which we call current equation.

We can also say that the current equation governs the relationship between the carrier densities $n(x)$, $p(x)$ and quasi Fermi levels $E_{F,n/p}(x)$.

nextnano++ solves this equation and Poisson equation (and also Schrödinger equation) self-consistently.

In their solution, the corresponding calculation of the carrier densities $\{n(x, \phi, E_{F,n}), p(x, \phi, E_{F,p})\}$ and Poisson equation are firstly iterated for a given quasi-Fermi levels until the carrier densities converge. Then the resulting carrier densities are substituted into the current equation and the quasi-Fermi levels are updated. This whole cycle is iterated until the quasi-Fermi levels satisfy the convergence criteria, which can be tuned by the users from run{ current_poisson[] } or run{ quantum_current_poisson[] }.

Recombination/Generation

The recombination mechanisms that nextnano++ takes into account for the right-hand-side of (13.6.2.20) are

- Shockley-Read-Hall (SRH) recombination
- Auger recombination
- Radiative recombination
- “fixed (applied)”
The equations and parameters used for the three recombination mechanisms on the top are explained here: recombination[].

The last one “fixed (applied)” is the contribution defined from structure generation[] and classical irradiation[]. These typically represent generation instead of recombination and used for the simulation of the devices under irradiation such as solar cells or CCDs. (For example, see nextnano++ tutorial GaAs Solar Cell.)

13.6.3 Optoelectronic characteristics based on the semi-classical model

According to the specification in the section classical[], nextnano++ can calculate optoelectronic characteristics of the arbitrary structure by means of the so-called semi-classical model.

In this model, various quantities are calculated from the spontaneous emission rate, which is obtained at each position \(x\) for the photons with each energy \(E\) based on the energy-resolved carrier densities \(n(x, E)\) and \(p(x, E)\) obtained in the forgoing simulation.

- **Spontaneous emission rate**

\[
R_{\text{rad}}^{\text{spont}}(x, E) = C(x) \int dE_n \int dE_p n(x, E_n)p(x, E_p)\delta(E_n - E_p - E) .
\]  

(13.6.3.1)

Here \(C(x) [\text{cm}^3\text{s}^{-1}]\) is the (material-dependent) radiative recombination parameter which is proportional to the one specified in the database (Radiative recombination).

Then the other optical characteristics like stimulated emission rate, absorption/gain spectrum, and the imaginary part of the dielectric constant are calculated according to this \(R_{\text{rad}}^{\text{spont}}(x, E)\).

- **Stimulated emission rate**

Stimulated emission rate is calculated here as the net emission rate containing both the generation by the stimulated absorption and the recombinant by the spontaneous and stimulated emission according to the following equation:

\[
R_{\text{rad,net}}^{\text{stim}}(x, E) = R_{\text{rad}}^{\text{spont}}(x, E) - \frac{E}{h} \int dE R_{\text{rad}}^{\text{rad}}(x, E) .
\]  

(13.6.3.2)

The reference equation is eq.(9.2.39) of [ChuangOpto1995].

\(R_{\text{rad}}^{\text{spont}}(E)\) and \(R_{\text{rad,net}}^{\text{stim}}(E)\) are output on Optical/emission_spectrum_photons_.dat and stim_emission_photons_.dat as the integral of the above two quantities over \(x\), i.e.

\[
R_{\text{rad}}^{\text{spont}}(E) = \int dE R_{\text{rad}}^{\text{spont}}(x, E), \quad R_{\text{rad,net}}^{\text{stim}}(E) = \int dE R_{\text{rad,net}}^{\text{stim}}(x, E)
\]  

(13.6.3.3)

On the other hand, \(R_{\text{rad,net}}^{\text{stim}}(x)\) is obtained as the integral of \(R_{\text{rad,net}}^{\text{stim}}(x, E)\) over the photon energy, which is written as “radiative” in the output file recombination.dat, i.e.

\[
R_{\text{rad,net}}^{\text{stim}}(x) = \int dE R_{\text{rad,net}}^{\text{stim}}(x, E).
\]  

(13.6.3.4)

**Note:** Precisely speaking, \(R_{\text{rad,net}}^{\text{stim}}(x)\) is not directly integrated from \(R_{\text{rad,net}}^{\text{stim}}(x, E)\) but calculated according to the equation

\[
R_{\text{rad,net}}^{\text{stim}}(x) = C(x)n(x)p(x)\left(1 - e^{-\frac{E_{p_{\text{rad}}}}{k_{\text{B}}}T}\right).
\]

This is meanwhile equivalent to (13.6.3.4).

- **Generation by the irradiation (fixed(applied))**

There is another radiative recombination rate output on recombination.dat called “fixed(applied)”, which should be always negative. This is the contribution of the generation specified from structure generation[] and classical irradiation[]. When we don’t specify either of them, this recombination rate is always 0.

\[
R_{\text{fixed}}(x) = -(G(x) \text{ specified from structure})
- (\int dE G(E, x) \text{ calculated according to the configuration in classical}).
\]  

(13.6.3.5)
This is mostly used for the analysis of the absorbing devices such as solar cells or CCDs.

- **photocurrent**

Then the *photocurrent* $I_{\text{photo}}$ is calculated as the summation of the integration of these “radiative” and “fixed”:

$$I_{\text{photo}} = e \cdot \left( \int \! dx \, R_{\text{stim,net}}(x) + \int \! dx \, R_{\text{fixed}}(x) \right)$$  \hspace{1cm} (13.6.3.6)

- **internal quantum efficiency**

is calculated as

$$\eta_{\text{IQE}} = \frac{I_{\text{photo}}}{I_{\text{total}}}$$  \hspace{1cm} (13.6.3.7)

where $I_{\text{total}}$ is the total injected current consisted of both electron and hole currents.

- **volume quantum efficiency**

, which is also called as *radiative quantum efficiency*, is calculated as

$$\eta_{\text{VQE}} = \frac{R_{\text{stim,net}} + R_{\text{fixed}}}{R_{\text{total}}}$$  \hspace{1cm} (13.6.3.8)

where $R_{\text{total}} = R_{\text{stim,net}} + R_{\text{fixed}} + R_{\text{Auger}} + R_{\text{SRH}}$ is the total recombination rate including both radiative and non-radiative recombination.

Both $\eta_{\text{IQE}}$ and $\eta_{\text{VQE}}$ agree if the electrons and holes injected into the active region are fully consumed up by the recombination there. However, if they are not consumed up, $e \cdot R_{\text{total}} < I_{\text{charge}}$ and this results in $\eta_{\text{IQE1}} > \eta_{\text{IQE2}}$.

**Note:** If you have any comments on the terminologies and definitions of these quantities, please send to support [at] nextnano.com.

Moreover, the electrical power and optical power are calculated and output in *power.dat*:

- **Power**

$$\sum_i V_{\text{i-th contact}} \cdot I_{\text{i-th contact}}$$  \hspace{1cm} (13.6.3.9)

- **Absorbed-power**

$$\int dE \! dx \, E \cdot G(E, x)$$  \hspace{1cm} (13.6.3.10)

where $G(E, x)$ is the generation rate calculated according to the configuration in *classical*.

- **Emitted-power**

$$\int dE \! dx \, E \cdot R_{\text{rad,sp}}^\text{em}(E, x)$$  \hspace{1cm} (13.6.3.11)

### 13.6.4 Optoelectronic characteristics based on the quantum model

*nextnano++* has another important calculation scheme of optical properties, which is specified in the section *optics*. Here *nextnano++* calculates them using the Fermi’s golden rule (time-dependent perturbation theory) with 8 band k.p model.

These quantities are now supported

- Optical absorption
- Real/imaginary part of the dielectric constant
- Refractive index
- Optical gain as a negative part of optical absorption
- Spontaneous emission rate
- Transition intensity (optical matrix element)

For further detail about this section, please see *1D Optics: Optical absorption for interband and intersubband transitions*.

### 13.6.5 References

- [ZiboldPhD2007]
  Semiconductor based quantum information devices: Theory and simulations
  T. Zibold

- [BirnerPhD2011]
  Modeling of semiconductor nanostructures and semiconductor–electrolyte interfaces
  S. Birner
  ISBN 978-3-941650-35-0

- [ChuangOpto1995]
  Physics of Optoelectronic Devices
  S. L. Chuang

*Please help us to improve our tutorial. Should you have any questions or comments, please send to support [at] nextnano.com.*
14.1 Quantum Confined Stark Effect (QCSE)

Author: Stefan Birner

The input file used is:

- `1DQuantumConfinedStarkEffect.in`

This tutorial aims to reproduce Figure 3.22 (p. 96) of Paul Harrison’s excellent book “Quantum Wells, Wires and Dots” (Section 3.12 “Quantum Confined Stark Effect”), thus the following description is based on the explanations made therein. We are grateful that the book comes along with a CD so that we are able to look up the relevant material parameters and to check the results for consistency.

14.1.1 Single quantum well: 20 nm AlGaAs / 6 nm GaAs / 20 nm AlGaAs

Our structure consists of a 6 nm GaAs quantum well that is surrounded by 20 nm Al\textsubscript{0.2}Ga\textsubscript{0.8}As barriers on each side. We thus have the following sequence: 20 nm Al\textsubscript{0.2}Ga\textsubscript{0.8}As / 6 nm GaAs / 20 nm Al\textsubscript{0.2}Ga\textsubscript{0.8}As. The barriers are printed in bold.

The figure shows the conduction band edge and the square of the ground state electron wave function ($\psi^2$) that is confined inside the well for two cases:

- No applied electric field (0 kV/cm)
- Applied electric field (-70 kV/cm)

In the case of an applied electric field, the wave function moves to the right and its ground state energy decreases slightly. The reason is that a charged particle prefers to move to areas of lower potential in order to lower the total energy. (Note that the energies were shifted so that the conduction band edge of GaAs equals 0 eV) The origin of the electric field is chosen automatically to be the center of the well. This makes it possible to compare energies by varying the applied electric field as shown in this tutorial. For holes, the wave function would move to the left (not shown here) thus making it possible to produce a space charge or a polarization of the charge carriers.
14.1.2 Technical Details

We use

\begin{verbatim}
$simulation
flow-scheme 21 ! apply constant electric field
\end{verbatim}

and

\begin{verbatim}
$numeric
zr-potential yes
\end{verbatim}

This flow scheme includes the following:

- We calculate the strain (if any)
- We calculate the piezo and pyroelectric charges (if any)
- **We do not solve Poisson’s equation** (This is the difference with \texttt{flow\_scheme = 20})
- We apply the electric field
- We calculate the eigenstates and wave functions by solving Schrödinger’s equation (either single-band or \texttt{k.p})

Note that in this case, this is not a self-consistent calculation of the Poisson-Schrödinger equation.

14.1.3 Output

a. The \textbf{conduction} band edge of the Gamma conduction band can be found at \texttt{band\_structure / cb1D_Gamma.dat}

b. This file contains the \textbf{eigenenergies} of the ground state. The units are in \textbf{[eV]} \texttt{Schroedinger\_1band / ev1D_cb001_qc001_sg001_deg001_dir.dat}

For example, the values for zero applied electric field read:

<table>
<thead>
<tr>
<th>num_ev</th>
<th>eigenvalue [eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextnano</td>
<td>1</td>
</tr>
<tr>
<td>Paul Harrison’s book</td>
<td>1</td>
</tr>
</tbody>
</table>
c. This file contains the **eigenenergies** and the **squared wave functions** ($\psi^2$): `Schroedinger_1band / cb001_qc001_sg001deg001_dir_psi_squared.dat`

d. This file contains the **eigenenergies** and the **wave functions** ($\psi$): `Schroedinger_1band / cb001_qc001_sg001deg001_dir_psi.dat`

a) and b) can be used to plot the data as shown in the figure above.

### 14.1.4 Varying the electric field

There are two ways to vary the applied electric field from 0 kV/cm to -70 kV/cm.

#### Possibility 1

We perform several individual calculations and vary the strength of the electric field by specifying its value for each individual calculation.

```plaintext
$electric
  electric-field-strength = -5d5 ! in units of [V/m] - Here: -5 kV/cm
  electric-field-strength = -10d5 ! in units of [V/m] - Here: -10 kV/cm
  electric-field-strength = -15d5 ! in units of [V/m] - Here: -15 kV/cm
  electric-field-strength = -20d5 ! in units of [V/m] - Here: -20 kV/cm
  electric-field-strength = -25d5 ! in units of [V/m] - Here: -25 kV/cm
  electric-field-strength = -30d5 ! in units of [V/m] - Here: -30 kV/cm
  electric-field-strength = -40d5 ! in units of [V/m] - Here: -40 kV/cm
  electric-field-strength = -50d5 ! in units of [V/m] - Here: -50 kV/cm
  electric-field-strength = -60d5 ! in units of [V/m] - Here: -60 kV/cm
  electric-field-strength = -70d5 ! in units of [V/m] - Here: -70 kV/cm
$end_electric
```

#### Possibility 2

An **alternative** (and much more user friendly approach) would be the usage of an “electric field sweep”. Here, only one calculation is necessary. The variation of the electric field strength is done automatically.

```plaintext
$electric
  electric-field-strength = -5d5 ! in units of [V/m] - Here: 0 kV/cm
  electric-field-strength = -10d5 ! in units of [V/m] - Here: -5 kV/cm
  electric-field-strength = -15d5 ! in units of [V/m] - Here: -10 kV/cm
  electric-field-strength = -20d5 ! in units of [V/m] - Here: -15 kV/cm
  electric-field-strength = -25d5 ! in units of [V/m] - Here: -20 kV/cm
  electric-field-strength = -30d5 ! in units of [V/m] - Here: -25 kV/cm
  electric-field-strength = -40d5 ! in units of [V/m] - Here: -30 kV/cm
  electric-field-strength = -50d5 ! in units of [V/m] - Here: -40 kV/cm
  electric-field-strength = -60d5 ! in units of [V/m] - Here: -50 kV/cm
  electric-field-strength = -70d5 ! in units of [V/m] - Here: -60 kV/cm
  electric-field-strength = -80d5 ! in units of [V/m] - Here: -70 kV/cm
$end_electric
```

Here, the electric field is varied from 0 kV/cm to -70 kV/cm, in steps of -5 kV/cm. The output of the eigenvalues is then contained in `Schroedinger_1band / electric_ev1D_cb001_qc001_sg001_deg001_dir_Kx001_Ky001_Kz001.dat`

The first column contains the strength of the electric field in units of [kV/cm]. The second column contains the 1\textsuperscript{st} eigenvalue for the specified electric field in units of [eV]. The third column contains the 2\textsuperscript{nd} eigenvalue for the specified electric field in units of [eV].

The following figure shows the ground state energy of the 6 nm quantum well as a function of the applied electric field strength $F$. The calculated energies can be represented by a parabolic fit. Over the range of electric fields
investigated, the ground state energy can be represented by the parabola:

\[ E_1(F) = E_1(0) - 0.000365 F^2 \]

where \( E_1(0) \) refers to the ground state energy at zero electric field (in units of meV). Here, the electric field strength \( F \) is given in units of kV/cm. (Note: Paul Harrison’s value is 0.00036)

This suppression of the confined energy level by an electric field is called “Quantum Confined Stark Effect (QCSE)”. The data that were plotted here are contained in this file, Schroedinger_1band\electric[kV/cm]_ev1D_cb1_qcl_sg1_degl_dir.dat. The energy levels are contained as a function of the sweep variable “electric field”.

The plot is (almost) in agreement with Fig. 3.22 (p. 96) of Paul Harrison’s book “Quantum Wells, Wires and Dots”. The energies differ slightly although we used:

- identical effective masses
- identical conduction band offset
- identical grid resolution (0.1 nm)

The only difference is that we use multiple points at the heterointerfaces but this should not explain the difference. The answer lies probably in a tiny inconsistency in the book. On page 96 it says: “where \( E_1(0) \) refers to the ground-state energy (53.310 meV) at zero field” However, the value that was used in Fig. 3.22 is \( E_1(0) = 53.26045 \) meV. (This value can be found on the CD that accompanies the book.) The nextnano\(^3\) value is 53.287 meV which lies in the middle between these two values.

Output

The energy values were taken from the file Schroedinger_1band / ev1D_cb001_qc001_sg001_deg001_dir_Kx001_Ky001_Kz001.dat.

For example, the values for zero applied electric field read:

<table>
<thead>
<tr>
<th>num_ev</th>
<th>eigenvalue [eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextnano</td>
<td>0.053287</td>
</tr>
<tr>
<td>Paul Harrison’s book</td>
<td>0.05326045 (or 0.053310)</td>
</tr>
</tbody>
</table>

The values for an applied electric field of -70 kV/cm read:

<table>
<thead>
<tr>
<th>num_ev</th>
<th>eigenvalue [eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextnano</td>
<td>0.051497</td>
</tr>
<tr>
<td>Paul Harrison’s book</td>
<td>0.051472</td>
</tr>
</tbody>
</table>
14.2 Exciton energy in quantum wells - Tutorial

This tutorial aims to reproduce figures 6.4 (p. 196) and 6.5 (p. 197) of Paul Harrison’s excellent book *Quantum Wells, Wires and Dots* (Section 6.5 "The two-dimensional and three-dimensional limits") (*HarrisonQWWD2005*), thus the following description is based on the explanations made therein.

*We are grateful that the book comes along with a CD so that we were able to look up the relevant material parameters and to check the results for consistency.*

The following input file was used:
- 1DExcitonCdTe_QW.in (input files for nextnano\textsuperscript{3} software)

CdTe quantum well with infinite barriers

In order to correlate the calculated optical transition energies of a 1D quantum well to experimental data, one has to include *exciton* (electron-hole pair) corrections. In this tutorial we study the exciton correction of the electron ground state to the heavy hole ground state (e1-hh1).

### 14.2.1 Bulk

The 3D bulk exciton binding energy can be calculated analytically

\[
E_{\text{ex},b} = -\frac{\mu e^4}{32\pi\hbar^2}\varepsilon r \varepsilon_0^2 = -\frac{\mu}{m_0 c^2} \cdot 13.61 \text{ eV},
\]

where \(\mu\) is the reduced mass of the electron–hole pair, \(1/\mu = 1/m_e + 1/m_h\).

- GaAs: \(1/\mu = 1/0.067 + 1/0.5 \Rightarrow \mu = 0.0591\)
- CdTe: \(1/\mu = 1/0.096 + 1/0.6 \Rightarrow \mu = 0.0828\)

- \(h\) is Planck’s constant divided by \(2\pi\)
- \(e\) is the electron charge
- \(\varepsilon\) is the dielectric constant (GaAs: 12.93, CdTe: 10.6)
- \(\varepsilon_0\) is the vacuum permittivity
- \(m_0\) is the rest mass of the electron and
- 13.61 eV is the Rydberg energy.

In GaAs, the 3D bulk exciton binding energy is equal to -4.8 meV with a Bohr radius of \(\lambda = 11.6\) nm. In CdTe it is equal to -10.0 meV with a Bohr radius of \(\lambda = 6.8\) nm. Thus the energy of the exciton, i.e. the band gap transition, reads:

- GaAs: \(E_{\text{ex}} = E_{\text{gap}} + E_{\text{ex},b} = 1.519 \text{ eV} - 0.005 \text{ eV} = 1.514 \text{ eV}\)
- CdTe: \(E_{\text{ex}} = E_{\text{gap}} + E_{\text{ex},b} = 1.606 \text{ eV} - 0.010 \text{ eV} = 1.596 \text{ eV}\)

### 14.2.2 Quantum well (type-I)

A 1D quantum well for a type-I structure has two exciton limits for the ground state transition (e1-hh1):

- infinitely thin quantum well (2D limit)
  \(E_{\text{ex,QW}} = 4E_{\text{ex}}, \lambda_{\text{ex,QW}} = \lambda_{\text{ex}}/2\)
- infinitely thick quantum well (3D bulk exciton limit)
  \(E_{\text{ex,QW}} = E_{\text{ex}}, \lambda_{\text{ex,QW}} = \lambda_{\text{ex}}\)

Between these limits, the exciton correction which depends on the well width has to be calculated numerically, not only for the ground state but also for excited states (e.g. e2-hh2, e1-lh1).
14.2.3 CdTe quantum well with infinite barriers

In this tutorial we study the exciton binding energy of CdTe quantum wells (with infinite barriers) as a function of well width.

The material parameters used are the following ($\text{binary- zb-default}$):

```
!---------------------------------------------------------!
! Here we are overwriting the database entries for CdTe. !
!---------------------------------------------------------!
$\text{binary}$
$\text{type}$
$\text{numbers}$
$\text{masses}$
$\text{[m0]}$
$\text{masses}$
$\text{[m0]}$
$\text{constants}$
```

We chose infinite barriers, in order to be able to compare the nextnano calculations with standard textbook results, originally published by [BastardPRB1982], namely the exciton binding energy of a type-I quantum well (in units of the 3D bulk exciton energy $E_x$, also called effective Rydberg energy) as a function of well width (in units of the 3D bulk exciton Bohr radius $\lambda_x$).

**Template**

The following screenshot shows how to use the Template feature of nextnanomat in order to calculate the exciton binding energy as a function of the quantum well width.

![Parameter sweep screenshot](image)

Figure 14.2.3.1: Parameter sweep: QuantumWellWidth
Open input file in Template tab.

Select List of values, select variable QuantumWellWidth. The corresponding list of values are loaded from the template input file.

Click on Create input files to create an input file for each quantum well width.

Switch to Simulation tab and start the batch list of jobs.

Results

The following figure shows the exciton binding energy in an infinitely deep quantum well as a function of well width. Both quantities are given in terms of the effective Rydberg energy and the Bohr radius for a 3D exciton in the same material.

Figure 14.2.3.2: Exciton energy as a function of quantum well width

Our numerical approach is the following:

The exciton binding energy is minimized with respect to the variational parameter \( \lambda \). We use a separable wave function:

\[
\psi(r) = \sqrt{\frac{2}{\pi \lambda}} \exp\left(-\frac{r}{\lambda}\right)
\]

see e.g. p. 562, Eq. (13.4.27), Section 13.4.3 Variational Method for Exciton Problem in [ChuangOpto1995] or [BastardPRB1982].

Thus the 3D limit is not reproduced correctly in our approach (not shown in the figure). To obtain the 3D limit, a nonseparable wave function has to be used, \( \psi(r, z_e, z_h) \).

Figure 14.2.3.3: Exciton binding energy in an infinitely deep quantum well

14.2. Exciton energy in quantum wells - Tutorial
The following figure shows the exciton binding energy in an infinitely deep CdTe quantum well as a function of well width. The nextnano\textsuperscript{3} results are in nice agreement with the Fig. 6.4 of [HarrisonQWWD2005] although we use a simpler trial wave function with only one variational parameter.

![Exciton Bohr radius energy in an infinitely deep quantum well](image)

**Figure 14.2.3.4:** Exciton Bohr radius energy in an infinitely deep quantum well

In order to calculate the exciton correction, the following flags have to be used:

```
$numeric
c ontrol
simulation-dimension = 1
c alculate exciton = yes ! to switch on exciton correction
e xciton-electron number = 1 ! electron ground state
e xciton-hole number = 1 ! hole ground state
```

The output of the exciton binding energies can be found in this file: Schroedinger_lband/exciton_energy1D.dat

The output for the 5 nm CdTe QW looks like this:

```
<table>
<thead>
<tr>
<th>lambda [nm]</th>
<th>exciton energy [meV]</th>
<th>exciton energy [Rydberg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.338893904E+001</td>
<td>-0.158496790E+002</td>
<td>0.158160603E+001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.421888329E+001</td>
<td>-0.215591082E+002</td>
<td>0.215133793E+001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.553296169E+001</td>
<td>-0.232757580E+002</td>
<td>0.232263879E+001</td>
</tr>
</tbody>
</table>
```

Calculated lambda and exciton energy:

```
<table>
<thead>
<tr>
<th>lambda [nm]</th>
<th>exciton energy [meV]</th>
<th>exciton energy [Rydberg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.546379967E+001</td>
<td>-0.232817837E+002</td>
<td>0.232324009E+001</td>
</tr>
</tbody>
</table>
```

The last iteration yields -23.28 meV for the exciton binding energy. \( \lambda \) is the variational parameter \( \lambda \) which is equivalent to the exciton Bohr radius in units of [nm].
14.3 Optical interband absorption in a quantum well including excitonic effects

This tutorial calculates the optical interband absorption in a quantum well including excitonic effects.

There is a separate tutorial that discusses the calculation of the exciton binding energy and exciton Bohr radius of an infinite quantum well: *Exciton energy in quantum wells - Tutorial*

In this tutorial we calculate the optical absorption of a 10 nm GaAs quantum well. The purpose is to calculate the absorption spectrum for a simple model and model that includes *excitonic effects* on the absorption spectrum.

The absorption has been calculated using a simple model assuming a parabolic energy dispersion. In order to keep things simple, i.e. to be able to compare our results with analytical formula, we used the same effective mass for electrons and holes \((m_e = m_h = 0.065 m_0)\).

The excitonic binding energy \(E_b\) has been calculated to be -9.5 meV. Therefore, the optical absorption spectrum that includes excitonic contributions starts at an energy roughly 10 meV below than band gap. The exciton Bohr radius \(\lambda\) was found to be 13.1 nm.

For Lorentzian broadening we use a linewidth of FWHM = 6 meV, and for Gaussian broadening we use FWHM = 10 meV. The FWHM(Voigt) depends in a complicated way on FWHM(Lorentzian) and FWHM(Gaussian).

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
<th>unit</th>
<th>analytical calculation</th>
<th>nextnano</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantum well width</td>
<td>(L)</td>
<td>nm</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>barrier height</td>
<td>(E_b)</td>
<td>eV</td>
<td>infinite quantum well model</td>
<td>1000</td>
</tr>
<tr>
<td>effective electron mass</td>
<td>(m_e)</td>
<td>(m_0)</td>
<td>0.0665</td>
<td>0.0665</td>
</tr>
<tr>
<td>effective hole mass</td>
<td>(m_h)</td>
<td>(m_0)</td>
<td>0.0665</td>
<td>0.0665</td>
</tr>
<tr>
<td>refractive index</td>
<td>(n_r)</td>
<td></td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>linewidth (FWHM) Lorentzian</td>
<td>(\Gamma_L)</td>
<td>meV</td>
<td>n/a</td>
<td>6</td>
</tr>
<tr>
<td>linewidth (FWHM) Gaussian</td>
<td>(\Gamma_G)</td>
<td>meV</td>
<td>n/a</td>
<td>10</td>
</tr>
<tr>
<td>temperature</td>
<td>(T)</td>
<td>K</td>
<td>300</td>
<td>300</td>
</tr>
</tbody>
</table>

The Coulomb enhancement factor is given by \(S_{2D} = \frac{\exp(\pi/\sqrt{\Delta})}{\cosh(\pi/\sqrt{\Delta})}\), where \(\Delta\) is the total excess kinetic energy of the electron–hole pair normalized to \(E_b/4\) [LeverJLT2010].

We observe two major contributions to the absorption:

- A distinct peak a few meV (corresponding to the exciton binding energy \(E_b\)) lower than the absorption edge (band gap). This is the signature of the bound exciton.
- Sommerfeld enhancement: In the continuum part of the absorption, the absorption is scaled via the Coulomb enhancement factor \(S_{2D}\).

---

**Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy**

The following documentation and figures were generated automatically using nextnanopy.

The following Python script was used: `1D_InterbandAbsorption_InfiniteWell_Exciton_nextnano3.py`.

The following figures have been generated using the nextnano³ software.

**Infinite QW (single-band)**

Optical absorption of bulk and of a quantum well
Figure 14.3.1: Conduction and valence band edges, Fermi level, electron and holes states of a quantum well.

Figure 14.3.2: The optical absorption of bulk GaAs and a 10 nm infinite quantum well.
Figure 14.3.3: The optical absorption spectrum of a 10 nm infinite quantum well consisting of GaAs. The absorption spectrum has been calculated with and without excitonic contributions to the spectrum.

**Optical absorption of a 10 nm quantum well**

**Optical absorption of a 10 nm quantum well using different broadening functions**

**Optical absorption of a 10 nm quantum well showing the different contributions to the excitonic absorption**

We acknowledge funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 101017194 (SiPho-G).

Automatic documentation: Running simulations, generating figures and reStructured Text (*.rst) using nextnanopy

— End —

## 14.4 Mobility in two-dimensional electron gases (2DEGs)

Author: Stefan Birner

The input files used in this tutorial are:

- 1DInSb_mobility_ShaoFig3.in
- 1DGaAs_mobility_WalukiewiczFig2.in
- 1DGaAs_mobility_WalukiewiczFig3.in
- 1DInGaAs_mobility_WalukiewiczFig8.in
- 1DGaN_mobility_WalukiewiczFig4.in
- 1DGaN_mobility_WalukiewiczFig5.in
Figure 14.3.4: The optical absorption spectrum of a 10 nm infinite quantum well consisting of GaAs. The absorption spectrum has been calculated with Lorentzian, Gaussian and Voigt broadening function, and without excitonic contributions to the spectrum.

Figure 14.3.5: The optical absorption spectrum of a 10 nm infinite quantum well consisting of GaAs showing the different excitonic contributions. The absorption spectrum has been calculated with a Voigt broadening functions, and without excitonic contributions to the spectrum. In order to see the contributions of the Coulomb enhancement factor and the lineshape peak, each of these contributions can be switched off.
14.4.1 Table of Contents

14.4.2 Mobility in delta-doped InSb quantum wells

This tutorial is based on the following paper:


Our implementation is based on the equations that are given in this paper (with the exception of Eq. (3.5) where we added a factor of $\frac{1}{4\pi}$ because of SI units).

We calculate the mobility in a 40 nm InSb quantum well that is surrounded by and strained with respect to $\text{Al}_{0.15}\text{In}_{0.85}\text{Sb}$ barriers.

At $z = -20$ nm, there is a delta-doping layer with a sheet doping density of $1 \times 10^{12} \text{cm}^{-2}$. The delta-doping layer is separated from the InSb QW by a 40 nm $\text{Al}_{0.15}\text{In}_{0.85}\text{Sb}$ spacer layer.

<table>
<thead>
<tr>
<th>quantum width</th>
<th>40.0 [nm]</th>
<th>[Shao] Fig. 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta width</td>
<td>40.0 [nm]</td>
<td>[Shao] Fig. 3</td>
</tr>
<tr>
<td>Density</td>
<td>1e12 [cm$^{-2}$]</td>
<td>[Shao] Fig. 3</td>
</tr>
</tbody>
</table>

We calculate all properties for different temperatures. This can be done as follows:

```
$global
    temperature      1.0 ! start value $T = 1$ [K]
    active           yes ! 'yes' / 'no'
    size             10.0 ! increase temperature each time by $T = 10$ [K]
    steps            31 ! increase temperature 31 times
    step              1 ! output all data for every temperature
$end_global
```

All output files are labelled with an index, starting from zero, that refers to each individual temperature sweep: `.../..._ind000....dat`. Here, the index runs from 0 (000) to 30 (030), i.e. 31 output files for each property in total.

We note that band gaps and lattice constants depend on temperature. This is taken into account automatically for each temperature sweep. The following figure shows the conduction band edge, the Fermi level and the square of the lowest two electron wave functions at $T = 1$ K.
To plot such a figure, the following output files are needed:

- **band_structure/cb1D_001_ind000.dat**
  - 1st column: distance [nm]
  - 2nd column: conduction band edge at the Gamma point [eV]
  - Note: The index ‘ind000’ refers to the temperature sweep. Here, index 0 means T = 1 K.

- **current/fermi1Del_ind000.dat**
  - 1st column: distance [nm]
  - 2nd column: Fermi level of the electrons [eV]
  - In this tutorial, the Fermi level is always equal to 0 eV.

- **Schroedinger_1band/cb001_ind000_sgl_deg1.dat**
  - 1st column: distance [nm]
  - n columns: n energies of the eigenstates [eV]
  - n columns: n squares of the wave functions ($\psi^2$) [eV]
  - In the figure, we plotted the columns for $\psi^2$ of the two lowest states: $\psi_1^2$, $\psi_2^2$

The following figure shows the sheet electron density as a function of temperature. We considered the two lowest subbands for calculating the 2DEG density (the spin degeneracy of the subbands is included):

Our results differ from the results of Fig. 3(b) of the Shao’s paper. (Some obvious discrepancies are the conduction band offset (We used ~0.15 eV whereas Shao used ~0.25 eV.) and the Schottky barrier height.)
To plot such a figure, the following output file was used:

- Monte_Carlo/mobility_TemperatureSweep.dat
  - 1st column: temperature [K]
  - last column: electron sheet density of the lowest subband(s) [m^-2]

The following figure shows the calculated 2DEG mobility as a function of temperature. The relevant data can be found in this file:

- Monte_Carlo/mobility_TemperatureSweep.dat
  - 1st column: temperature [K]
  - 2nd column: total mobility [m^2/Vs]
  - 3rd column: mobility due to ionized impurity scattering [m^2/Vs]
  - 4th column: mobility due to background impurity scattering [m^2/Vs]
  - 5th column: mobility due to deformation potential acoustic phonon scattering [m^2/Vs]
  - 6th column: mobility due to polar optic (LO) phonon scattering [m^2/Vs]
We included the following scattering mechanisms:

- Ionized impurity scattering: yes! [Shao] (including remote and background impurity scattering)
- Polar optic phonon scattering: yes! [Shao]
- All other scattering: no! [Shao]

We now discuss the agreement/disagreement compared to Fig. 3(a) of the [Shao] paper.

- The mobility due to acoustic phonon scattering is in excellent agreement.
- The mobility due to polar optic LO phonon scattering is in excellent agreement if one takes into account that [Shao] forgot to include the factor of 1/(4π) due to SI units.
- The mobility due to ionized and background impurity scattering differs significantly. It seems that the disagreement is not only due to the different sheet density that has been used. We used the following value:

  \[
  \text{concentration} = 5 \times 10^{15} \text{ cm}^{-2} \]  

The following InSb material parameters have been used:

- masses: \( 0.0135 \ 0.0135 \ 0.0135 \) ! [m0] [Shao]
- \( \varepsilon(0) \) constants: \( 16.82 \ 16.82 \ 16.82 \) ! [Shao]
- \( \varepsilon(\infty) \) constants: \( 15.7 \) ! [Shao]
- \( \text{phonon energy} \): \( 0.025 \) ! [eV] [Shao] (optical)
14.4.3 Mobility in doped GaAs quantum wells

The following input files were used:

- `1DGaAs_mobility_WalukiewiczFig2.in` (Experiment of Hiyamizu et al.)
- `1DGaAs_mobility_WalukiewiczFig3.in` (Experiment of DiLorenzo et al.)

Here, we test our algorithm to results on GaAs 2DEGs of another publication. We note that our algorithm is suitable for delta-doped 2DEGs but the GaAs examples are not delta-doped.

This section is based on the following paper:

[Walukiewicz]
Electron mobility in modulation-doped heterostructures
W. Walukiewicz, H.E. Ruda, J. Lagowski, H.C. Gatos

The experimental data is based on:

[Walukiewicz, Fig. 2]: [Hiyamizu]
Improved Electron Mobility Higher than 10^6 cm^2/Vs in Selectively Doped GaAs/N-AlGaAs Heterostructures grown by MBE
S. Hiyamizu, J. Saito, K. Nanbu

[Walukiewicz, Fig. 3]: [DiLorenzo]
Material and device considerations for selectively doped heterojunction transistors

Conduction band profile and wave functions

[Walukiewicz, Fig. 2]: [Hiyamizu]
![20 nm Al$_{0.3}$Ga$_{0.7}$As spacer](#)

!20 nm Al$_{0.3}$Ga$_{0.7}$As spacer

[ Walukiewicz, Fig. 3]: [DiLorenzo]

![23 nm Al$_{0.3}$Ga$_{0.7}$As spacer](#)

!23 nm Al$_{0.3}$Ga$_{0.7}$As spacer

[ Walukiewicz, Fig. 3]: [DiLorenzo]
2DEG sheet density

[Walukiewicz, Fig. 2]: [Hiyamizu]

[Walukiewicz, Fig. 3]: [DiLorenzo]

Mobility

[Walukiewicz, Fig. 2]: [Hiyamizu]
GaAs 2DEG mobility

- \( \mu_{\text{background-impurity}} \)
- \( \mu_{\text{ionized-impurity}} \)
- \( \mu_{\text{alloy}} \)
- \( \mu_{\text{acoustic}} \)
- \( \mu_{\text{polar-optic}} \)

\( \mu_{\text{total}} \)

20 nm Al\(_{0.3}\)Ga\(_{0.7}\)As spacer

**Temperature (K)**

**Mobility (m\(^2\)/Vs)**

- Concentration: \( 9 \times 10^{13} \) cm\(^{-3}\)
- Density: \( 3.5 \times 10^{11} \) cm\(^{-2}\) (to fit experiment)
- Density: \( 1.948344 \times 10^{11} \) cm\(^{-2}\)

\([\text{Walukiewicz, Fig. 2}]: [\text{DiLorenzo}]\)

\([\text{Walukiewicz, Fig. 3}]: [\text{DiLorenzo}]\)
**Differences with respect to Walukiewicz [Hiyamizu] paper**

Walukiewicz used a 2DEG density of \(3 \times 10^{11} \text{cm}^{-2}\).

Here we used 0.067 as this gives better agreement to the mobility at higher temperatures and this is the usually accepted material parameter for GaAs.

Alloy scattering is relevant for the part of the wave function that penetrates into the AlGaAs barrier.

The squares are experimental values of Fig. 5 in:

Improved Electron Mobility Higher than 106 cm\(^2\)/Vs in Selectively Doped GaAs/N-AlGaAs Heterostructures Grown by MBE

S. Hiyamizu et al.
The following GaAs material parameters have been used:

\[
\begin{align*}
\text{epsilon(0)} &= 12.9 \\
\text{epsilon(infinity)} &= 10.9 \\
\text{omega_e} &= 0.036 \\
\text{mass} &= 0.076 \\
\text{density} &= 5.318 \times 10^{11} \\
\text{masses} &= 0.076 \\
\text{quantum well width} &= 20.0 \\
\text{potential} &= 7.0 \\
\end{align*}
\]

Into the equation for the deformation potential acoustic phonon scattering, the quantum well width is an input parameter. We used a value of 13 nm which corresponds roughly to the extension of the ground state wave function inside the “triangular” QW.

Differences with respect to Walukiewicz [DiLorenzo] paper

In Walukiewicz’s paper, the background impurity scattering is dominating the ionized impurity scattering. We found the opposite.

Walukiewicz used a 2DEG density of \(2 \times 10^{11} \text{ cm}^{-2}\)

\[
\begin{align*}
\text{masses} &= 0.076 \\
\text{potential} &= 7.0 \\
\text{quantum well width} &= 20.0 \\
\end{align*}
\]

14.4.4 Mobility in doped InGaAs quantum wells

The input file used is:

- IDInGaAs_mobility_WalukiewiczFig8.in (Experiment of Kastalsky et al.)

Here, we test our algorithm to results on InGaAs 2DEGs of another publication. We note that our algorithm is suitable for delta-doped 2DEGs but the InGaAs examples are not delta-doped.

This tutorial is based on the following paper:

[Walukiewicz]
Electron mobility in modulation-doped heterostructures
W. Walukiewicz, H.E. Ruda, J. Lagowski, H.C. Gatos

The experimental data is based on:

[Walukiewicz, Fig. 8]: [Kastalsky]
Two-dimensional electron gas at a molecular beam epitaxial-grown, selectively doped, In0.53Ga0.47As-In0.48Al0.52As interface

[Kastalsky] and [Walukiewicz] have used In0.52Al0.48As whereas we used In0.52Al0.48As which is lattice matched to InP and In0.53Al0.47As.

The conduction band profile is shown in the following figure. Here, two subbands ($\psi_1$, $\psi_2$) are occupied although our implementation of calculating the mobility is only applicable to one occupied subband. Note that the 2DEG is located in an alloy, i.e. InGaAs. Thus we expect that alloy scattering has a significant effect on the total mobility.

The following figure shows the subband density of the first subband and of the first two subbands as a function of temperature. Inside the mobility algorithm, only the density of the first subband has been considered.

The following figure shows the mobility as a function of temperature. At temperatures below 100 K, the total mobility is dominated by alloy scattering.
Our results for the mobility are in reasonable agreement with Fig. 8 of the paper of [Walukiewicz].

The following parameters were used:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>scattering</td>
<td>yes (including remote and</td>
</tr>
<tr>
<td>background ionized impurity scattering</td>
<td></td>
</tr>
<tr>
<td>ion-scattering</td>
<td>yes</td>
</tr>
<tr>
<td>alloy-scattering</td>
<td>yes</td>
</tr>
<tr>
<td>quantum well width</td>
<td>15.0 ! [nm] 15 nm seems to be a</td>
</tr>
<tr>
<td>reasonable approximation for the triangular well</td>
<td></td>
</tr>
<tr>
<td>height width</td>
<td>8.0 ! [nm]</td>
</tr>
<tr>
<td>ionized-impurity concentration</td>
<td>1.0e16 ! [cm−3]</td>
</tr>
<tr>
<td>density</td>
<td>1.0e12 ! [cm−2]</td>
</tr>
<tr>
<td>subbands</td>
<td>1</td>
</tr>
<tr>
<td>potential</td>
<td>0.60 ! [eV] InGaAs bulk value [J.R.</td>
</tr>
<tr>
<td>material parameters</td>
<td></td>
</tr>
<tr>
<td>density</td>
<td>5.5025e3 ! [kg/m^3] InGaAs</td>
</tr>
<tr>
<td>velocity</td>
<td>4.753e3 ! [m/s] In0.53Ga0.47As[111]</td>
</tr>
<tr>
<td>potential</td>
<td>0.60 ! [eV] InGaAs</td>
</tr>
</tbody>
</table>

---

InGaAs bulk value [J.R. Hayes et al. (1982)]
14.4.5 Mobility in doped GaN quantum wells

The input files used are:

- 1DGaN_mobility_WalukiewiczFig4.in
- 1DGaN_mobility_WalukiewiczFig5.in

Here, we test our algorithm to results on GaN 2DEGs of another publication. We note that our algorithm is suitable for delta-doped 2DEGs but the GaN examples are not delta-doped.

This tutorial is based on the following paper:

[WalukiewiczGaN]
Electron mobility in AlxGa1-xN/GaN heterostructures
L. Hsu, W. Walukiewicz

Note: To be consistent with the paper of Walukiewicz, no (!) piezo- and pyroelectricity is included.

The following GaN material parameters have been used:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>masses</td>
<td>0.21 0.21 0.21</td>
<td>[m0]</td>
</tr>
<tr>
<td>epsilon(0) constants</td>
<td>9.5 9.5 9.5</td>
<td>[WalukiewiczGaN]</td>
</tr>
<tr>
<td>epsilon(infinity) constants</td>
<td>5.35 5.35 5.35</td>
<td>[WalukiewiczGaN]</td>
</tr>
<tr>
<td>energy</td>
<td>0.0905 0.0905 0.0905</td>
<td>[eV]</td>
</tr>
<tr>
<td>density</td>
<td>6.1e3</td>
<td>[kg/m^3]</td>
</tr>
<tr>
<td>velocity</td>
<td>6.6e3</td>
<td>[m/s]</td>
</tr>
<tr>
<td>potential</td>
<td>8.5</td>
<td>[eV]</td>
</tr>
<tr>
<td>potential</td>
<td>2.3</td>
<td>[eV] conduction band offset GaN/AlN</td>
</tr>
</tbody>
</table>

Here, alloy scattering is only relevant for the part of the wave function that penetrates into the AlGaN barrier.
Conduction band profile and wave functions [WalukiewiczGaN]

[WalukiewiczGaN] Fig. 4

![Conduction Band Profile with 20 nm Al_{0.15}Ga_{0.85}N Spacer](image)

- Fermi level
- Piezo and pyroelectricity is not included!

[WalukiewiczGaN] Fig. 5

![Conduction Band Profile without Spacer](image)

- Fermi level
- Piezo and pyroelectricity is not included!
2DEG sheet density

[WalukiewiczGaN] Fig. 4
Into the equation for the deformation potential acoustic phonon scattering, the quantum well width is an input parameter. We used values which correspond roughly to the extension of the ground state wave function inside the “triangular” QW.

[WalukiewiczGaN] used a 2DEG density of $6.2 \times 10^{15}$ m$^{-2}$.

Quantum well width $= 15.0 \pm 15$ [nm]

[WalukiewiczGaN] Fig. 5

[WalukiewiczGaN] used a 2DEG density of $1.59 \times 10^{16}$ m$^{-2}$.

Quantum well width $= 10.0 \pm 10$ [nm]
Mobility

[WalukiewiczGaN] Fig. 4

![GaN 2DEG mobility graph](image)

Fig. 4: $7 \times 10^{17}$ cm$^{-3} \rightarrow 7 \times 10^{17} \text{cm}^{-3} \times \frac{2}{3}$

[WalukiewiczGaN] Fig. 5

![GaN 2DEG mobility graph](image)
Final remark: In principle, the results of these GaN 2DEGs are not reliable as piezo- and pyroelectricity have to be included. Our results disagree quantitatively with the results of [WalukiewiczGaN]. However, it is not clear, which material parameters he used for the conduction band offset and the alloy scattering.

Further hints

If two remote doping regions should be taken into account, one can input an array of values.

| Concentration | 4e15 \text{ cm}^{-3} |
| Density       | 1e12 \text{ cm}^{-2} |

14.5 Efficient method for the calculation of ballistic quantum transport - The CBR method (2D example)

In this tutorial we apply the Contact Block Reduction (CBR) method to a Aharonov-Bohm-type structure with a large barrier in the middle of the device.

The following two nextnano\textsuperscript{3} input files are based on the paper [MamaluyCBR2003]

- 2D_CBR_MamaluySabathilJAP2003.in
- 2D_CBR_MamaluySabathilJAP2003_holes.in

the latter of which simulates holes instead of electrons.

![Figure 14.5.1: Schematic sketch of the device showing two possible paths of the electrons](image)

- The device consists of three leads that are called ‘source’, ‘gate’ and ‘drain’ in this example.
- In the middle of the device a potential barrier of two-dimensional Gaussian shape effectively expels the electrons from the center.
- In the upper part of the device, a thin tunneling double barrier is present.
- The device dimensions are 20 nm x 20 nm.
A detailed description of the device can be found in Section V of [MamaluyCBR2003].

• The effective electron mass is assumed to be constant throughout the device and equal to $0.3 m_0$.
• The device region consists of $41 \times 41 = 1681$ grid points, which is equivalent to a grid spacing of 0.5 nm. This means that the device Hamiltonian is a matrix of size $1681 \times 1681$.
• The tunneling barriers have a width of 1 nm each and are separated by 3 nm.
• The maximum height of the Gaussian barrier is $E_{c,0} = 1$ eV, the height of the double barriers is 0.4 eV.
• The conduction band profile is given by $E_c = E_{c,0} \exp\left[-(x^2 + y^2)/a^2\right]$ where $x$ and $y$ are with respect to the center of the device, and $a = 5$ nm.
• The conduction band profile is achieved by using an appropriate ternary material having a 2D Gaussian alloy profile.
• The lower gate is 6 nm long, all other leads are 20 nm long.

The following figure shows the calculated transmission coefficients of the various lead combinations $T_{12}$, $T_{23}$ and $T_{13}$. For the thick lines 18 % (303 of 1681) of all eigenvectors were used whereas for the thin lines only 7 % (118 of 1681) had to be calculated, i.e. one does not have to calculate all eigenvalues of the device Hamiltonian which grossly reduces CPU time. A small percentage of eigenvalues suffices for $T(E)$ in relevant energy range of interest.
14.5. Efficient method for the calculation of ballistic quantum transport - The CBR method (2D example)
The transmission coefficient can be found in this file:  CBR_data1/transmission2D_cb_sg001_ind000_CBR.dat

The nextnano\textsuperscript{3} results differ slightly from the [MamaluyCBR2003] paper. Reason: The potential energy profile in the device and in the leads is not exactly identical, as well as the dimensions of the barriers. Therefore the eigenenergies and wave functions in the device, and in the leads differ slightly which explains the small deviations.

The eigenstates # 16 is a resonance state of the lower transmission path.
- 1st resonance: # 16: 0.123 eV
- Its square of the wave function is shown below.

The eigenstates # 26 and # 29 are resonance states of the double barrier.
- 1st resonance:
  - # 26: 0.182 eV (delocalized)
  - # 29: 0.196 eV (more localized)
- 2nd resonance:
  - # 55: 0.322 eV (delocalized)
  - # 57: 0.333 eV (more localized)
The following figure shows the conduction band profile together with the square of the wave function of the 26th eigenstate. One can clearly see that it is a resonance state of the double barrier and corresponds to the second peak in the blue transmission curve $T_{13}$ from source to drain around 180 meV.

14.5.1 Lead modes

The following two figures show the lead modes of the gate, and the source (which is identical to the drain). In the transmission curve $T_{12}(E) = T_{23}(E)$, the transmission shows a step-like behavior which is related to the energies of lead no. 2 (‘gate’).

The lead modes (eigenvalues, psi, $\psi^2$, band edge profile, ...) can be found in these files:

CBR_data1/modes_lead00*_sg001_*_.dat
14.5.2 Technical details

Definition of contacts

For each contact (lead), a quantum cluster ("lead quantum cluster") has to be defined because in each lead, a one-dimensional Schrödinger equation has to be solved which gives us the lead modes (i.e. energies and eigenvectors of the leads). In addition, a quantum cluster is required for the device itself ("main quantum cluster").

![Diagram of lead modes](image)

```plaintext
$quantum$

$number$ 1 ! 'device'
$geometry$ rectangle
$priority$ 1 !
x-coordinates (0) 20 ! [nm] width of 'device' = 20
y-coordinates (0) 20 ! [nm] length of 'device' = 20

$number$ 2 ! 'source'
$geometry$ rectangle
$priority$ 2 !
x-coordinates (0) -0.5 ! [nm] (including 2 gridpoints along this direction)
y-coordinates (0) 20 ! [nm] length of 'lead 1' = 20

$number$ 3 ! 'gate'
$geometry$ rectangle
```

(continues on next page)
For each quantum cluster, the number of eigenstates to be calculated and its boundary conditions have to be specified.

For the main quantum cluster it holds: For each grid point in the main quantum cluster it is checked if it is at the boundary and if it is in contact to a lead. If it is at the boundary, and if it is in contact to a lead, a Neumann boundary condition is set. If it is at the boundary, and if it is not in contact to a lead, a Dirichlet boundary condition is set.

14.5. Efficient method for the calculation of ballistic quantum transport - The CBR method (example)
For each energy $E$ (num-energy-steps = 100) where the transmission coefficient $T(E)$ has to be calculated, a matrix of size 95 x 95 has to be inverted. The size of 95 is determined by the sum of the number of grid points in each lead that are in contact to the device.

- Lead 1 (Source): 41 grid points
- Lead 2 (Gate): 13 grid points
- Lead 3 (Drain): 41 grid points
  - in total: 95 grid points
  
  The total CPU time for calculation of the transmission $T(E)$ in this example was about 30 seconds for 303 eigenstates.

For further information, please study this section: SCBR-current.
14.6 Transmission through a 3D nanowire (3D example)

In this tutorial we apply the Contact Block Reduction (CBR) method to a simple GaAs nanowire of cuboidal shape, using the input files

- 3D_CBR_nanowire_10x10x20.in
- 3D_CBR_nanowire_10x10x20_holes.in

the latter of which is for holes instead of electrons.

The corresponding tutorial for nextnano++ can be found [here](#).

![Schematic sketch of the 3D device showing the GaAs region that is placed between two contacts (red and green leads)](image)

- The device consists of two leads of 10 nm x 10 nm each. Each lead has a total of 121 grid points (11 x 11 grid points).
- The leads are described as quantum regions, i.e. in each lead (quantum region) a two-dimensional Schrödinger equation has to be solved to obtain the eigenenergies and wave functions of the lead modes.
- The device dimensions are 10 nm x 10 nm x 20 nm.
- The grid spacing is 1 nm in all directions.
- The effective electron mass is assumed to be constant throughout the device and equal to 0.067 \( m_0 \).
- The device region consists of 11 x 11 x 21 = 2541 grid points, which is equivalent to a grid spacing of 1.0 nm. This means that the device Hamiltonian is a matrix of size 2541 x 2541.
- The conduction band profile is constant and set to \( E_c = 0 \) eV.

The following figure shows the calculated transmission coefficient as a function of energy between the leads 1 and 2.

- For the blue lines 23.6 % (600 of 2541) of all eigenvectors were used whereas for the red lines only 15.7 % (400 of 2541) had to be calculated, i.e. one does not have to calculate all eigenvalues of the device Hamiltonian which grossly reduces CPU time.
- For the black lines 7.9 % (200 of 2541) of all eigenvectors were used.

A small percentage of eigenvalues suffices for \( T(E) \) in relevant energy range of interest. Note that the transmission drops significantly once the cutoff energy of the highest eigenvector taken into account is reached.

The transmission coefficient can be found in this file: CBR_data1/transmission3D_cb_sg001_CBR.dat
In the following, the same data as above is shown again as a zoom into the energy range 0 eV - 0.5 eV. The colored figures show the wave function amplitude of the lowest energy lead modes.

They can be found in these files:

- CBR_data1/2Dmodes_lead001_sg001_psi_ev001.dat
- ...
- CBR_data1/2Dmodes_lead001_sg001_psi_ev080.dat
- CBR_data1/2Dmodes_lead002_sg001_psi_ev001.dat
- ...
- CBR_data1/2Dmodes_lead002_sg001_psi_ev080.dat

• Once the energy reaches 78 meV, the first lead mode energy is reached and then this mode transmits perfectly, giving a transmission of 1.

• The second and third lead mode states are degenerate due to the symmetry of the lead cross-section, thus they have the same energy (191 meV). Consequently, once the energy of 191 meV is reached, the transmission increases by 2.

• The total transmission is now equal to 3 as all lead modes transmit perfectly.
• The energy of the 4th lead mode is at 305 meV.
• The degeneracy of the 5th and 6th mode is accidental. They have the same energy.

As one can clearly see, in this low energy limit, it is sufficient to calculate only a few percent of all eigenfunctions of the device Hamiltonian. For the leads, in all cases 80 eigenstates have been calculated.

14.6.1 Density of states

Using

\[ \text{DOS } \text{yes } \]

the CPU time increases but information like the density of states (DOS) can be obtained. This is shown in the next figure where the same transmission data as above is plotted but this time including the density of states. Again, the colors indicate taking into account 200, 400 or 600 eigenvectors of the decoupled system (closed system).

![Density of states graph](image)

14.6.2 Technical details

Definition of contacts

For each contact (lead), a quantum cluster (“lead quantum cluster”) has to be defined because in each lead, a two-dimensional Schrödinger equation has to be solved which gives us the lead modes (i.e. energies and eigenvectors of the leads). In addition, a quantum cluster is required for the device itself (“main quantum cluster”).

```
$quantum
regiogn number = 1 ! 'device'
regiogn geometry = bud
regiogn priority = !
x coordinates = 0, 10 nm ! [nm] width of 'device' along x
y coordinates = 0, 10 nm ! [nm] width of 'device' along y
```

(continues on next page)
For each quantum cluster, the number of eigenstates to be calculated.

```
!---------------------------------------------------!
$quantum_modell
!---------------------------------------------------!
|cluster_numbers| 1 |
!---------------------------------------------------!
|number of eigenvalues per band | 200 | ! corresponds to 7.87 % of 2541
!---------------------------------------------------!
|number of eigenvalues per band | 400 | ! corresponds to 15.74 % of 2541
!---------------------------------------------------!
|number of eigenvalues per band | 600 | ! corresponds to 23.61 % of 2541
!---------------------------------------------------!
lead 1 = source: number of modes per lead = 80  maximum number of relevant quantum grid points in lead 1 = 121
!---------------------------------------------------!
|cluster_numbers| 2 |
!---------------------------------------------------!
|number of eigenvalues per band | 80 | ! calculate 80 lead modes, corresponds to 66.1 % of 121
!---------------------------------------------------!
lead 2 = source: number of modes per lead = 80  maximum number of relevant quantum grid points in lead 1 = 121
!---------------------------------------------------!
|cluster_numbers| 3 |
!---------------------------------------------------!
|number of eigenvalues per band | 80 | ! calculate 80 lead modes, corresponds to 66.1 % of 121
!---------------------------------------------------!
$end_quantum
```

```
712 Chapter 14. nextnano3 Tutorials
```
Loop over energy (parallelization)

For each energy $E$ (num-energy-steps = 250) where the transmission coefficient $T(E)$ has to be calculated, a matrix of size $160 \times 160$ has to be inverted.

The size of $160$ is determined by the sum over the number of lead modes taken into account for each lead.

The upper limit would be the number of grid points in each lead that are in contact to the device, i.e. in this example where each lead has $11 \times 11 = 121$ grid points, the maximum size of the matrix to be inverted could be $242 = 121 + 121$.

- Lead 1 (Source): 121 grid points (80 lead modes taken into account)
- Lead 2 (Drain): 121 grid points (80 lead modes taken into account)
  - in total: 242 grid points (but only 80 modes are taken into account for each lead: 160 = 80 + 80)
  - The total CPU time for calculation of the transmission $T(E)$ in this example was less than a minute for 200 eigenstates, 80 lead modes in each lead, and 250 energy steps.

This loop can be executed in parallel to improve computational performance.

For further information, please study this section: $CBR$-current.

For 14.6. Transmission through a 3D nanowire (3D example)
14.7 Poisson–Boltzmann equation: The Gouy–Chapman solution

The following nextnano input files were used:
- 1DGouyChapman_template.in
- 2DGouyChapman_template.in
- 3DGouyChapman_template.in

We solve the Poisson–Boltzmann equation for a monovalent salt, i.e. NaCl (Na⁺ Cl⁻). For this particular case, our numerical solution of the Poisson–Boltzmann equation can be compared to the analytical one-dimensional Gouy–Chapman solution for a monovalent and symmetric salt.

The temperature is set to 298.15 K = 25°C, the static dielectric constant of water is set to 78.

The electrolyte region (0 nm - 200 nm) contains the following ions ($electrolyte-ion-content$):

```plaintext
! The electrolyte (NaCl) contains two types of ions:
! 1) 100 mM singly charged cations (Na⁺) <- 100 mM NaCl
! 2) 100 mM singly charged anions (Cl⁻) <- 100 mM NaCl

$electrolyte-ion-content

<table>
<thead>
<tr>
<th>ion-number</th>
<th>ion-valency</th>
<th>ion-concentration</th>
<th>ion-region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>100e-3</td>
<td>0.0 200.0</td>
</tr>
<tr>
<td>2</td>
<td>-1.0</td>
<td>100e-3</td>
<td>0.0 200.0</td>
</tr>
</tbody>
</table>
```

We vary the NaCl concentration ($ion-concentration$) from 0.1 mM to 1 M.
- 0.1 mM
- 1 mM
- 10 mM
- 0.1 M
- 1 M

We assume an interface charge density ($interface-states$) between the oxide and the electrolyte of -0.2 C/m² = -124.83 · 10¹² cm⁻².

```plaintext
$interface-states

<table>
<thead>
<tr>
<th>interface-number</th>
<th>interface-type</th>
<th>interface-density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>contact-electrolyte</td>
<td>-124.830193e12</td>
</tr>
<tr>
<td>2</td>
<td>contact-electrolyte</td>
<td>-124.830193e12</td>
</tr>
</tbody>
</table>
```

The pH value ($electrolyte$) is 7, i.e. neutral.
The following figure shows the electrostatic potential for different salt concentrations (0.1 mM, 1 mM, 10 mM, 0.1 M and 1 M) at a fixed surface charge of -0.2 C/m². The potential at the surface at 0 nm, that arises due to the fixed surface charge density that is in contact with the electrolyte, is screened by the ions in the solution and the resulting distribution of the ions depends on the value of the spatially varying electrostatic potential.

![Electrostatic Potential Diagram](image)

**Figure 14.7.1:** Electrostatic potential for different salt concentrations. The interface is at 0 nm.

The Debye screening lengths (DebyeScreeningLength.txt) are indicated by the squares and the values are:

<table>
<thead>
<tr>
<th>NaCl salt concentration</th>
<th>Debye screening length (nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 M</td>
<td>0.303</td>
</tr>
<tr>
<td>0.1 M</td>
<td>0.959</td>
</tr>
<tr>
<td>10 mM</td>
<td>3.032</td>
</tr>
<tr>
<td>1 mM</td>
<td>9.589</td>
</tr>
<tr>
<td>0.1 mM</td>
<td>30.308</td>
</tr>
</tbody>
</table>

**Note:** For 0.1 mM, the concentrations of the H₃O⁺ and OH⁻ ions slightly influence the Debye screening length (last two digits) because in the nextnano³ simulations these ions are always present and their concentrations depend on the pH value.

For a definition of the Debye screening length, have a look here: **Electrolyte**

The following figure shows the Debye screening length for a monovalent salt such as NaCl as a function of the salt concentration.

For a monovalent salt the nominal value of the salt concentration is equal to the ionic strength which is a measure for the screening of charges in a solution.

The surface potential can be found in this file: InterfacePotentialDensity_vs_pH1D.dat

It reads for a salt concentration of 0.1 M NaCl:

<table>
<thead>
<tr>
<th>pH value</th>
<th>interface potential [V]</th>
<th>interface density (1\times10^{12} [e/cm^2])</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.000000</td>
<td>-0.123478240580906</td>
<td>-124.830193000000</td>
</tr>
</tbody>
</table>

The following figure shows the ion distribution for a 0.1 M NaCl electrolyte. The multiples of the Debye screening lengths ($\kappa^{-1} = 0.959$ nm) are indicated by the vertical lines. The negative surface charge is screened by the positive Na$^+$ ions whereas the negatively charged Cl$^-$ ions are repelled from the surface. At about 5 nm both ions reach their equilibrium concentration of 0.1 M.
14.7.1 Linearization of the Gouy–Chapman model

In this approximation which is only valid for high salt concentrations or small surface charges, the surface charge and the surface potential can be related through the basic capacitor equation

$$\sigma_s = \phi_s C_{DL}$$

where $C_{DL}$ is the capacitance per unit area of the electric double layer.

The following figure shows the surface potential at the solid/electrolyte interface as a function of interface charge for a monovalent salt such as NaCl at different salt concentrations calculated with the Poisson–Boltzmann equation (symbols). The solid lines are the solutions of the analytical Grahame equation for a monovalent salt which relates surface potential to surface charge. The dotted lines are the solutions of the Debye–Hückel approximation for a monovalent salt which relates surface potential to surface charge.

It can be clearly seen that only for high salt concentrations or small surface charges the linearization is valid.

![Figure 14.7.1.1: Surface potential vs. surface charge for different salt concentrations](image)

The linearization of the Poisson–Boltzmann equation is called *Debye–Hückel approximation*. It can be switched on using:

```
! electrolyte-equation = Poisson-Boltzmann
--equation (default)

! electrolyte-equation = Debye-Huckel
--approximation
```

14.7.2 Capacitance

The numerical Poisson–Boltzmann calculations for the capacitance (using the same data as in the previous figure) are shown next. The capacitance increases rapidly for higher potentials but at very small surface potentials, the capacitance is equal to the approximation of parallel plate capacitor model of the electric double layer. This is expected because in the limit of low potentials, the solution of the Poisson-Boltzmann equation must converge to the solution of the Debye–Hückel equation.

![Capacitance vs. surface potential](image)

Figure 14.7.2.1: Capacitance vs. surface potential

14.8 Empirical tight-binding $sp^3s^*$ band structure of GaAs, GaP, AlAs, InAs, C (diamond) and Si

The input files to be used are:

- `1D_TightBinding_bulk_GaAs.in`
- `1D_TightBinding_bulk_GaAs_so.in`
- `1D_TightBinding_bulk_Al0.3Ga0.7As.in`
- `1D_TightBinding_bulk_GaP.in`
- `1D_TightBinding_bulk_GaP_so.in`
- `1D_TightBinding_bulk_AlAs.in`
- `1D_TightBinding_bulk_AlAs_so.in`
- `1D_TightBinding_bulk_C.in`
- `1D_TightBinding_bulk_Si.in`
- `1D_TightBinding_bulk_Ge.in`
- `1D_TightBinding_bulk_InAs_so.in`
- `1D_TightBinding_bulk_AlSb_so.in`
- `1D_TightBinding_bulk_InSb_so.in`
14.8.1 Empirical tight-binding $sp^3s^*$ band structure of GaAs and GaP

The empirical tight-binding model that is used here is based on the $sp^3s^*$ Hamiltonian, i.e. the 10 x 10 matrix given in Table (A) of [VoglJPCS1983].

In addition, we include spin-orbit coupling leading to a 20 x 20 matrix. The additional terms arising due to spin-orbit coupling are given for instance on p. R5 of [CarloSST2003].

We note that nowadays much better theoretical methods are available for calculating the band structure of bulk materials. However, for educational purposes, the chosen $sp^3s^*$ method should be sufficient.

In this tutorial, we calculate the bulk band structure of

- GaAs, GaP and AlAs without spin-orbit coupling using the parameters of [VoglJPCS1983] at T = 0 K
- GaAs, GaP and AlAs including spin-orbit coupling using the parameters of [KlimeckSM2000] at T = 300 K

14.8.2 Input

The values for the tight binding parametrization have to be specified in the input file:

```plaintext
$numeric

!------------------------------------------------------------------------------
! Tight-binding parameters for GaAs (values of [Klimeck]). The units are [eV].
!------------------------------------------------------------------------------
!tight-binding-parameters = -3.53284d0 ! Esa (GaAs)
0.27772  ! Epa
-8.11499  ! Esc
4.57341  ! Epc
12.33930  ! Es_a
4.31241  ! Es_c
-6.87653  ! Vss
1.33572  ! Vxx
5.07596  ! Vxy
0        ! Vs_s_ 2.85929  ! Vsa_pc
11.09774  ! Vsc_pa
6.31619  ! Vs_a_pc
5.02335  ! Vs_c_pa
0.32703  ! 0.12000 ! Delta_so_a Delta_so_c

! Note: a = anion, c = cation
! s_ = s*
```

For more information about the meaning of these parameters, refer to the above cited references.

14.8.3 Output

The output of the calculated tight-binding band structure can be found in the following file: TightBinding/BandStructure.dat

The first column contains the number of the grid point in the Brillouin zone. These grid points run

- from L point to Gamma point (along Lambda)
- from Gamma point to X point (along Delta)
- from X point to the U, K points
- from U,K points to Gamma point (along Sigma)
The next columns are the eigenvalues of the tight-binding Hamiltonian in units of \([\text{eV}]\) for each grid point in \(k = (k_x, k_y, k_z)\) space.

The file `TightBinding/BandStructure_without_so.dat` contains the tight-binding band structure without spin-orbit coupling.

The file `TightBinding/k_vectors.dat` contains for each point the information to which \(k\) point it belongs to.

| no. | \(k_x\)  | \(k_y\)  | \(k_z\)  | \(|k|\)  | \(k_x [2\pi/a]\) | \(k_y [2\pi/a]\) | \(k_z [2\pi/a]\) | \(|k| [2\pi/a]\) |
|-----|-----------|-----------|-----------|---------|----------------|----------------|----------------|----------------|
| 1   | 0.314159E+01 | 0.314159E+01 | 0.314159E+01 | 0.544140E+01 | 0.500000E+00 | 0.500000E+00 | 0.500000E+00 | 0.866025E+00 |

Note: Currently the units of \(k_x\), \(k_y\) and \(k_z\) do not take into account the lattice constant \(a\). This should be modified. The values for \(k_x\), \(k_y\) and \(k_z\) in units of \([2\pi/a]\) are correct, however. Another improvement would be to calculate and output the three-dimensional energy dispersion \(E(k_x, k_y, k_z)\) and two-dimensional slices \(E(k_x, k_z, 0)\) through the three-dimensional energy dispersion \(E(k_x, k_y, k_z)\) for a constant value of \(k_z\), e.g. \(k_z = 0\).

### 14.8.4 Results

**GaAs without spin-orbit coupling** from `1D_TightBinding_bulk_GaAs.in`

![Tight-binding band structure of GaAs](image)

**sp\(^3\)s\(^*\) parameters of Vogl et al.**

**without spin-orbit coupling**

The calculated band structure is in excellent agreement with Fig. 11(d) of [VoglJPCS1983]. The conduction band minimum is at the Gamma point (direct band gap). Because spin-orbit coupling is not included in the Hamiltonian, the \(sp^3s^*\) empirical tight-binding parameters were taken from [VoglJPCS1983] at \(T = 0\) K.

**GaAs including spin-orbit coupling** from `1D_TightBinding_bulk_GaAs_so.in`
The calculated band structure is in excellent agreement with Fig. 1 of [KlimeckSM2000]. The conduction band minimum is at the Gamma point (direct band gap). Spin-orbit coupling lifts the degeneracy of heavy/light hole and split-off hole at the Gamma point. Heavy and light hole are still degenerate at the Gamma point. The $sp^3s^*$ empirical tight-binding parameters were taken from [KlimeckSM2000] at $T = 300$ K.

**GaP without spin-orbit coupling** from 1D_TightBinding_bulk_GaP.in
The calculated band structure is in excellent agreement with Fig. 2 of [VoglJPCS1983]. The conduction band minimum is calculated to be at the X point (indirect band gap). Because spin-orbit coupling is not included in the Hamiltonian, heavy, light and the split-off hole are degenerate at the Gamma point, i.e at \( \mathbf{k} = (k_x, k_y, k_z) = 0 \). The \( \text{sp}^3\text{s}^* \) empirical tight-binding parameters were taken from [VoglJPCS1983] at \( T = 0 \) K.

**GaP including spin-orbit coupling** from 1D_TightBinding_bulk_GaP_so.in
The calculated band structure is in excellent agreement with Fig. 1 of [KlimeckSM2000]. The conduction band minimum is in the vicinity of the X point at the Delta line (indirect band gap), so-called camel’s back. Spin-orbit coupling lifts the degeneracy of heavy/light hole and split-off hole at the Gamma point. Heavy and light hole are still degenerate at the Gamma point. The $sp^3s^*$ empirical tight-binding parameters were taken from [KlimeckSM2000] at $T = 300$ K.

AIAs without spin-orbit coupling from 1D_TightBinding_bulk_AlAs.in
InAs including spin-orbit coupling from 1D_TightBinding_bulk_InAs_so.in

C (diamond) without spin-orbit coupling from 1D_TightBinding_bulk_C.in

Si (silicon) without spin-orbit coupling from 1D_TightBinding_bulk_Si.in

The k space resolution, i.e. the number of grid points on the axis of these plots can be adjusted. This can be done with:
$\text{tighten}$

```plaintext
\text{tighten} \quad \text{no}
\text{directory} \quad \text{TightBinding}
\text{points} \quad 50 \quad ! \text{This number corresponds to 50 k-}
\text{points between the Gamma point and the X point} \quad ! \text{The number of k points along}
\text{the other directions are scaled accordingly.}
```

Author: Stefan Birner, Reinhard Scholz

### 14.9 Tight-binding band structure of graphene

The input file used is:

- `1D_TightBinding_graphene.in`

#### 14.9.1 Nearest-neighbor tight-binding approximation

In this tutorial we calculate the bulk band structure of graphene which is a two-dimensional crystal (i.e., a monolayer of graphite) using a standard tight-binding approach. For more details, see for example the article [SaitoS2001](#).

The following figure shows the conduction band $\pi^*$ (upper part) and valence band $\pi$ (lower part) of graphene along special high-symmetry directions in the two-dimensional hexagonal Brillouin zone ($\mathbf{k}$ space).

The high symmetry points that are used in this graph (from left to right) are:

- **K**: $\mathbf{k} = (k_x, k_y) = (0, \frac{2}{3}) \times \frac{2\pi}{a}$
- **Gamma**: $\mathbf{k} = (k_x, k_y) = (0, 0)$
- **M**: $\mathbf{k} = (k_x, k_y) = (\frac{1}{2}, 0) \times \frac{2\pi}{a}$
- **K’**: $\mathbf{k} = (k_x, k_y) = (\frac{1}{3}, \frac{1}{3}) \times \frac{2\pi}{a}$
Two lines correspond to the case where the \( s_0 \) parameter is set to zero, i.e. in that case the dispersion of both \( \pi^* \) and \( \pi \) is the same (apart from the sign, i.e. they are symmetric with respect to the Fermi level \( E_F = 0 \text{ eV} \)). In this case the splitting energy at Gamma is three times as large as at the M point:

- splitting at Gamma: \( 6 \gamma_0 \) (for \( s_0 = 0 \))
- splitting at M: \( 2 \gamma_0 \) (for \( s_0 = 0 \))

Two lines correspond to the case where \( s_0 = 0.129 \). \( \pi^* \) and \( \pi \) are then nonsymmetric and are close to calculations from first principles and experimental data.

The data points are contained in the following file: `TightBinding/BandStructureGraphene.dat`. The first column contains integers which refer to the x axis (i.e. numbering of \( k \) points), the second column contains the eigenvalue of \( \pi^* \) in units of \([\text{eV}]\) (conduction band), the third column contains the eigenvalue of \( \pi \) in units of \([\text{eV}]\) (valence band). The file `TightBinding/k_vectors.dat` contains information about which integer corresponds to which \( k_x \) and \( k_y \) value.

The general formula for these lines read:

\[
E_{\pi,\pi^*} = \frac{[E_{2p} \pm \Gamma_0 w(k)]}{1 \pm s_0 w(k)}
\]

The parameters can be specified in the input file:

```plaintext
$numeric

-3.013 \text{ [eV]} E_{2p}: \text{site energy of the } 2pz \text{ atomic orbital (orbital energy)}

-3.013 \text{ [eV]} \gamma_0: \text{C-C transfer energy (usually it holds: } -3 \text{ eV} < \gamma_0 < -2.5 \text{ eV)}

0.129 \text{ []} s_0 = 0.129: \text{denotes the overlap of the electronic wave function on adjacent sites}
```

(continues on next page)
Then there are two lines that are linear around the K point (k.p approximation or linear expansion). Their linear dispersion is independent of the parameter $s_0$. Thus for small values of $k$ (i.e. with respect to the K point), the energy dispersion can be approximated by a linear dispersion relation.

$$E(k) = E_{2p} \pm hV_F |k| = E_{2p} \pm 3^{1/2} \Gamma_0 \frac{k a}{2}$$

where

- $a$ is the lattice constant of graphene ($a = 0.24612 \text{ nm}$)
- the Fermi velocity of the charge carriers is given by $V_F = 3^{1/2} \Gamma_0 \frac{a}{2\pi} = 0.98 \times 106 \text{ m s}^{-1} \simeq 0.003c$
- $c$ is the velocity of light

These data points are contained in this file: TightBinding/BandStructureGraphene_kp.dat

At the K point, the band gap is zero.

The following figure shows the energy dispersion $E(k_x, k_y)$ of graphene for $s_0 = 0.129$. At the K points, the band gap is zero. The point in the middle is the Gamma point. $k_x$ is from $[-\frac{2}{3}, \frac{2}{3}]\frac{2\pi}{a}$, the same holds for $k_y$.

These data points are contained in the files:

- TightBinding/BandStructureGraphene_cb.vtr
- TightBinding/BandStructureGraphene_vb.vtr

The figure has been generated using the AVS/Express software.
The following figure shows the contour plot of the energy dispersion $E(k_x, k_y)$ of graphene for $s_0 = 0$ for the conduction band. (Note that the valence band dispersion is identical, apart from the sign, for $s_0 = 0$.) These data points are contained in the file: TightBinding/BandStructureGraphene_cb.vtr

The following figure shows the contour plot of the energy dispersion $E(k_x, k_y)$ of graphene for $s_0 = 0.129$ for the conduction band. These data points are contained in the file: TightBinding/BandStructureGraphene_cb.vtr

The following figure shows the contour plot of the energy dispersion $E(k_x, k_y)$ of graphene for $s_0 = 0.129$ for the valence band. These data points are contained in the file: TightBinding/BandStructureGraphene_vb.vtr

The following figure is the nextnanomat screenshot for the conduction band dispersion $E(k_x, k_y)$. The six dark areas correspond to the Dirac points in graphene.
14.9. Tight-binding band structure of graphene
s = 0.129
14.9.2 Output Options

$\textbf{output}$

In this case, this refers to the bulk 2D tight-binding energy dispersion $E(k_x,k_y)$.
If yes, then the two-dimensional energy dispersion $E(k_x,k_y)$ is written out.

$tighten$

This parameter defines the resolution of the $E(k_x,k_y)$ energy dispersion.
E.g. if 'number-of-k-points = 100' then the $k_x$ gridding has a total of
$[-100,...,0,...,100]$ grid points,
i.e. a total of $2 \times 100 + 1 = 201$ grid points along the $k_x$ direction.
The same applies for $k_y$ direction.
14.9.3 Third-nearest-neighbor tight-binding approximation

The following figure shows the band structure of graphene. All lines are identical to the ones shown already above with the exception of the blue lines which is the third-nearest-neighbor tight-binding approximation.

The third-nearest-neighbor tight-binding approximation is described in ReichPR2002

The following parameters have been used for the third-nearest neighbor approximation, i.e. 7 parameters:

- Parameters $t_{ijg}$
- orbital (orbital energy) $-0.28$ eV
- $\gamma_0$: C-C transfer energy (nearest-neighbor, nn) $-2.97$ eV
- $s_0$: denotes the overlap of the electronic wave function on adjacent sites (nn) $0.073$
- $\gamma_1$: (2nd-neighbor, nn) $0.018$
- $\gamma_2$: (3rd-neighbor, nn) $0.026$
- $E_{2p}$: site energy of the 2pz atomic orbital $-2.97$ eV
- $\gamma_0$: gamma0: C-C transfer energy (nearest-neighbor, nn) $0.073$
- $s_0$: denotes the overlap of the electronic wave function on adjacent sites (nn) $0.073$
- $\gamma_1$: (2nd-neighbor, nn) $0.018$
- $\gamma_2$: (3rd-neighbor, nn) $0.026$
- $E_{2p}$: site energy of the 2pz atomic orbital $-2.97$ eV
- $\gamma_0$: gamma0: C-C transfer energy (nearest-neighbor, nn) $0.073$
- $s_0$: denotes the overlap of the electronic wave function on adjacent sites (nn) $0.073$
- $\gamma_1$: (2nd-neighbor, nn) $0.018$
- $\gamma_2$: (3rd-neighbor, nn) $0.026$
- $E_{2p}$: site energy of the 2pz atomic orbital $-2.97$ eV
- $\gamma_0$: gamma0: C-C transfer energy (nearest-neighbor, nn) $0.073$
- $s_0$: denotes the overlap of the electronic wave function on adjacent sites (nn) $0.073$
- $\gamma_1$: (2nd-neighbor, nn) $0.018$
- $\gamma_2$: (3rd-neighbor, nn) $0.026$

The parameters are taken from Reich et al. Note that the band gap at the K point is not exactly zero when using these parameters.)

14.9.4 More Options

Inside the code several options, i.e. several algorithms, exist to setup the tight-binding Hamiltonian:
Special option for third-nearest-neighbor tight-binding approximation in graphene:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>no (default)</td>
<td>use 7 parameters ($E_2p$, $\gamma$)</td>
</tr>
<tr>
<td>$\gamma_0$, $\gamma_1$, $\gamma_2$, $s_0$, $s_1$, $s_2$</td>
<td>use 6 parameters ($\gamma_0$, $\gamma_1$, $\gamma_2$, $s_0$, $s_1$, $s_2$)</td>
</tr>
<tr>
<td>$\gamma_0$, $\gamma_2$, $s_0$, $s_1$, $s_2$</td>
<td>use 6 parameters ($E_2p$, $\gamma_2$, $s_0$, $s_1$, $s_2$)</td>
</tr>
<tr>
<td>$\gamma_0$, $\gamma_1$, $s_0$, $s_1$, $s_2$</td>
<td>use 6 parameters ($E_2p$, $\gamma_1$, $s_0$, $s_1$, $s_2$)</td>
</tr>
</tbody>
</table>

$E_2p$ or $\gamma_1$ can be calculated internally in order to force $E(K) = 0$ eV where $K$ is the K point in the Brillouin zone. In that case, only 6 parameters are used (although 7 parameters have to be present in the input file). The parameter to be calculated is simply ignored inside the code.

The k space resolution, i.e. the number of grid points on the axis of these plots can be adjusted.

$tighten$

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$no$</td>
</tr>
<tr>
<td>$gamma0$, $gamma1$, $gamma2$, $s0$, $s1$, $s2$</td>
</tr>
<tr>
<td>$gamma0$, $gamma2$, $s0$, $s1$, $s2$</td>
</tr>
<tr>
<td>$gamma0$, $gamma1$, $s0$, $s1$, $s2$</td>
</tr>
</tbody>
</table>

This corresponds to 50 k points between the Gamma point and the M point. The number of k points along the other directions are scaled correspondingly.
Here we describe several example input files.

- **THz QCLs**
  - GaAs/AlGaAs
  - InGaAs/AlGaSb
- **Mid-IR QCLs**
  - InGaAs/AlInAs
  - GaAs/AlGaAs
- **RTDs**

### 15.1 THz QCLs

#### 15.1.1 GaAs/AlGaAs

- **THz QCL - Fatholoumi (2012)**
  Terahertz quantum cascade lasers operating up to ~200 K with optimized oscillator strength and improved injection tunneling
  Optics Express 20, 3866 (2012)

- **GaAs/Al0.15Ga0.85As terahertz quantum cascade lasers with double-phonon resonant depopulation operating up to 172 K**

- **Influence of doping on the performance of terahertz quantum-cascade lasers**

- **1.9 THz quantum-cascade lasers with one-well injector**
  S. Kumar, B. S. Williams, Q. Hu

- **Far-infrared (\(\lambda \approx 87\mu m\)) bound-to-continuum quantum-cascade lasers operating up to 90 K**
  G. Scalari, L. Ajili, J. Faist, H. Beere, E. Linfield, D. Ritchie, G. Davies
• Broadband THz lasing from a photon-phonon quantum cascade structure  
  G. Scalari, M. I. Amanti, C. Walther, R. Terazzi, M. Beck, J. Faist  
  Optics Express 18, 8043 (2010)

15.1.2 InGaAs/AlGaSb

• High performance InGaAs/GaAsSb terahertz quantum cascade lasers operating up to 142 K  

15.2 Mid-IR QCLs

15.2.1 InGaAs/AlInAs

• Mid-IR QCL - Yu Slivken Razeghi  
  Injector doping level-dependent continuous-wave operation of InP-based QCLs at $\lambda = 7.3\mu m$ above room temperature  
  J. S. Yu, S. Slivken, M. Razeghi  
  Semiconductor Science and Technology 25, 125015 (2010)

15.2.2 GaAs/AlGaAs

• 300 K operation of a GaAs-based quantum-cascade laser at $\lambda \approx 9\mu m$  
  H. Page, C. Becker, A. Robertson, G. Glastre, V. Ortiz, C. Sirtori  

15.3 RTDs

15.3.1 AlGaAs/GaAs RTD

This tutorial describes the nextnano.NEGF simulation of AlGaAs/GaAs resonant tunneling diode (RTD).  
Sample input files are available from the sample file folder:  
• RTD_Example_withScattering.xml  
• RTD_Example_ballistic.xml  
The first input file takes the scattering mechanisms into consideration, while the second input file ignores them.

Figure 15.3.1.1: Top: Local density of states calculated from RTD_Example_withScattering.xml. The structure consists of GaAs and two AlGaAs barriers. Down: Current-voltage characteristics calculated from RTD_Example_withScattering.xml.
Simulation input

Definition of materials

At first, the materials used in the structure have to be defined. Each material is referred by an alias, which is here “well” for GaAs and “barrier” for AlGaAs.

```xml
<Materials>
  <Material>
    <Name>GaAs</Name>
    <Alias>well</Alias>
    <Effective_mass_from_kp_parameters>yes</Effective_mass_from_kp_parameters>
  </Material>

  <Material>
    <Name>Al(0.4)Ga(0.6)As</Name>
    <Alloy_Composition>0.4</Alloy_Composition>
    <Alias>barrier</Alias>
    <Effective_mass_from_kp_parameters>yes</Effective_mass_from_kp_parameters>
  </Material>
</Materials>
```

It is specified that the effective mass is calculated from the k.p parameters.
Also, the effective mass is assumed as energy dependent and the model used for the calculation of effective mass is the one for single-band model.

For the details of the syntaxes of this block, please refer to this page: Syntax of the input file -> Material definition and parameters. The models used for the calculation of the effective mass are described here: Electronic Band Structure.

**Definition of layers**

Next, alternating layers consisting of barrier and well have to be specified. In this tutorial, the thickness of each layer is $5.0/2.0/10.0/2.0/5.0$ (nm) where AlGaAs barrier layer is in bold fonts.

```xml
<Superlattice>
  <Layer>
    <Material>well</Material>
    <Thickness unit="nm">5.0</Thickness>
  </Layer>
  <Layer>
    <Material>barrier</Material>
    <Thickness unit="nm">2.0</Thickness>
  </Layer>
  <Layer>
    <Material>well</Material>
    <Thickness unit="nm">10.0</Thickness>
  </Layer>
  <Layer>
    <Material>barrier</Material>
    <Thickness unit="nm">2.0</Thickness>
  </Layer>
  <Layer>
    <Material>well</Material>
    <Thickness unit="nm">5.0</Thickness>
  </Layer>
  <Analysis_Separator>
    <Separator_Position>6.0</Separator_Position>
    <!-- collector barrier -->
  </Analysis_Separator>
  <Analysis_Separator>
    <Separator_Position>18.0</Separator_Position>
    <!-- injector barrier -->
  </Analysis_Separator>
</Superlattice>
```

The resulting conduction bandedge profile can be found in the file called `Conduction_BandEdge.dat`. This file includes the (small) band bending due to the electrostatic potential. At a bias voltage of 0 mV, it looks as follows:
Figure 15.3.1.2: Conduction band edge at a zero bias voltage. This is obtained from RTD_Example_withScattering.xml.

Contacts (open boundary condition)

In order to simulate a system with open boundary conditions (instead of the default field-periodic boundary condition), contacts have to be defined by adding a <Contacts> section into the input file.

In RTD_Example_withScattering.xml this section is specified as follows:

```xml
<Contacts>
    <DensityLeft unit="cm^-3">1e18</DensityLeft>
    <DensityRight unit="cm^-3">1e18</DensityRight>
    <MaterialLeft>well</MaterialLeft>
    <MaterialRight>well</MaterialRight>
    <Broadening unit="meV">10.0</Broadening>
    <Ballistic>no</Ballistic>
</Contacts>
```

The carrier densities in the left and right contact have to been defined using the <DensityLeft> and <DensityRight> commands, as shown above. The unit is \([\text{cm}^{-3}]\).

The material of the left and right contacts needs to be defined by the command <MaterialLeft> and <MaterialRight>. The string value has to be an alias defined in the <Materials> section.

A broadening energy can be defined by the command <Broadening>. Indeed, scattering is not accounted in the contact, so that this command allows a phenomenological broadening of the density of states in the contact.

The command <Ballistic> can be used to calculate ballistic transport between the contacts (i.e. no scattering process considered) if its value is set to yes. This is the case of RTD_Example_ballistic.xml.

**Note:** In the current version (2020-11-19), only single band calculations are supported for open boundary conditions.
Scattering

The detailed description of the syntaxes for each scattering mechanism is described here: Syntax of the input file -> Scattering processes.

The following scattering mechanisms are included in this tutorial.

- Interface roughness scattering

```xml
<Interface_Roughness>
  <Amplitude_in_Z unit="nm">0.1</Amplitude_in_Z>
  <InterfaceAutoCorrelationType>0</InterfaceAutoCorrelationType> <!-- Correlation type: 0=Exponential, 1=Gaussian -->
  <Correlation_Length_in_XY unit="nm">8</Correlation_Length_in_XY>
</Interface_Roughness>
```

- Acoustic phonon scattering

```xml
<!-- Acoustic phonons -->
<Acoustic_Phonon_Scattering>no</Acoustic_Phonon_Scattering> <!-- Comment: Acoustic phonons are in general not efficient - can be neglected in most cases -->
<AcousticPhonon_Scattering_EnergyMax unit="meV">3.0</AcousticPhonon_Scattering_EnergyMax> <!-- Maximum acoustic phonon energy -->
```

- Charged impurity scattering

```xml
<!-- Charged impurities -->
  <!-- Effective temperature of the electrons involved in electrostatic screening: -->
  <!-- 3 models available -->
  <!-- model #1: Teff = T + Toffset * exp(-T/Toffset) with Toffset specified as Temperature_Offset_parameter -->
  <!-- model #2: self-consistent calculation (requires several iterations of the all-calculation). The accuracy specified by Accuracy_Self_consistent_Electron_Temperature -->
  <!-- model #3: Teff is directly specified by Electron_Temperature_for_Screening -->
  <Model_Temperature_for_Screening>1</Model_Temperature_for_Screening> <!-- integer 1,2 or 3 is required accordingly to the desired model -->
  <Temperature_Offset_parameter>150</Temperature_Offset_parameter> <!-- integer only such as Teff = T + Toffset * exp(-T/Toffset) -->
  <Accuracy_Self_consistent_Electron_Temperature>0.05</Accuracy_Self_consistent_Electron_Temperature> <!-- for model #2 only: self-consistent calculation until the effective temperature convergences below the desired accuracy -->
  <Electron_Temperature_for_Screening>200</Electron_Temperature_for_Screening> <!-- for model #3 only: the effective temperature is directly specified -->
  <ImpurityScattering_Strength>1</ImpurityScattering_Strength> <!-- 1.0 is the normal physical calculation. Other values may be used for testing the importance of impurity scattering. -->
```

- Electron-electron scattering

```xml
<Electron_Electron_Scattering>yes</Electron_Electron_Scattering>
```

- Alloy scattering

```xml
<Alloy_scattering>yes</Alloy_scattering>
```

(Advanced) In this sample input file, the Coulomb scatterers (ionized impurities and other charge carriers) are assumed to be homogeneously distributed in order to speed up the calculation.
<Homogeneous_Coulomb>yes</Homogeneous_Coulomb>

**Note:** LO-phonon scattering is applied by default as all materials have intrinsic parameters controlling the electron-phonon interaction. We can modify this default behavior by this command.

### Poisson equation

<Poisson>yes</Poisson>

If yes, the Poisson equation is included in the program flow. Then the electrostatic mean-field interactions (electron-electron and electron-impurities interactions) can be taken into account.

### Lateral motion

<Lateral_motion>
  <Material_for_lateral_motion>well</Material_for_lateral_motion>
  <!-- Lateral energy spacing -->
  <Value unit="meV">10</Value>
</Lateral_motion>

In this sample file, the parameters for the lateral motion (i.e. the two-dimensional free motion in the directions perpendicular to the growth axis) are taken from the material of well, i.e. GaAs.

Also, the discretization energy for this lateral motion is specified as 10 meV.

### Simulation output

#### Electron eigen states

The electron eigenstates calculated for the whole region biased for each voltage is written in Wannier-Stark/WannierStark_states.dat. These states are used as the basis states of the Green’s function.

**Note:** There is another output WannierStark/TightBinding_states.dat that describes the electron eigenstates confined in each GaAs region. These eigenstates are calculated from the Schrödinger equations for each region separated by <Analysis_Separator> in <Superlattice> section.

For the detailed explanation, please refer to [here](#).
Local density of states

The following figures show the local density of states (LDOS) for the bias of 100 mV written in 2D_plots/DOS_energy_resolved.vtr. Please note that the scaling of the colormap is different in the two figures. The gnuplot file which generates the gif animation is also available in the top directory as Animation_DensityOfStates.plt.

The LDOS tells us where and at which energy electronic states are available that the charge carriers can occupy. The LDOS is shown for $k_{||} = 0$, i.e. there are also electronic states available for $k_{||} \neq 0$. But they are not shown in this plot because then the picture could not show the minimum energy of each subband, which is at $k_{||} = 0$, so nicely.

Note: The gnuplot files that generate the above figures are DOS_energy_resolved.plt. The file explorer in the nextnanomat doesn’t show .plt file so please access through your default file explorer.

Electron density

The following figures show the energy resolved electron density $n(x, E)$ for the bias of 100 mV written in 2D_plots/CarrierDensity_energy_resolved.vtr. Please note that the scaling of the colormap is different in the two figures. The gnuplot file which generates the gif animation is also available in the top directory as Animation_CarrierDensity.plt.

The electron density is obtained from occupying the LDOS (for both $k_{||} = 0$ and $k_{||} \neq 0$) with charge carriers. It is a nonequilibrium occupation that is not described by a Fermi distribution.

Figure 15.3.1.3: Electron eigenstates for the whole region at the bias of 100 mV.
Figure 15.3.1.4: Local density of states calculated with scattering at the bias of 100 mV.

Figure 15.3.1.5: Local density of states calculated with the ballistic condition at the bias of 100 mV.
Figure 15.3.1.6: Electron density calculated with scattering at the bias of 100 mV.

Figure 15.3.1.7: Electron density calculated with the ballistic condition at the bias of 100 mV.
Current density

The following figures show the energy resolved current density $j(x,E)$ for the bias of 100 mV written in 2D_plots/CurrentDensity_energy_resolved.vtr. Please note that the scaling of the colormap is different in the two figures. The gnuplot file which generates the gif animation is also available as Animation_CurrentDensity.plt.

![Current Density Graph]

Figure 15.3.1.8: Current density calculated with scattering at the bias of 100 mV.

Current-voltage characteristics

The following figure shows the current-voltage characteristics calculated both with and without scattering. These results are taken from Current_vs_Voltage.dat.
Figure 15.3.1.9: Current density calculated with the ballistic condition at the bias of 100 mV.

Figure 15.3.1.10: Current-voltage calculated with and without scattering.
You can use HTCondor to run the nextnano software on your local computer infrastructure (“on-premise”). Essentially, the nextnanomat software submits the job either locally or on the “HTCondor” cluster. In both cases, the results of the calculations are located on your local computer.

This feature is only supported with our new license system.

16.1 HTCondor on nextnanomat

The following shows a screenshot from nextnanomat. 6 computers are connected to the HTCondor pool called e25nn. 120 slots are configured, 44 are currently available. Computers 2, 3, 4 and 6 are selected to accept jobs. Computers 2 and 6 are currently not available as they are in use.

16.2 Recommended Installation Process

Download HTCondor installer from HTCondor.

1. On the webpage, click on Download and go to Current Stable Release of UW Madison (as of September 24 2020, HTCondor 8.8.10).

2. We recommend the file for Windows in Native Packages. The filenames look similar to this one:
   - condor-8.8.10-513586-Windows-x64.msi (Version 8.8.10)

3. Select the file, agree to the license agreement and download the .msi file. When you download it, you can optionally enter your name, email address and institution and subscribe to the HTCondor newsletter.

Install HTCondor.

1. Start installer

2. Click Next and then accept License Agreement

3. Then there are two options. There will be one special computer that manages all HTCondor jobs (Central Manager), and other computers as submit/execute nodes. If there is no Central Manager yet, we have to create a new pool.
   1. If you are on the Central Manager, choose Create a new HTCondor Pool and fill in the name of the Pool, e.g. nextnanoHTCondorPool. This is a unique name for your pool of machines.
   2. If you are not the Central Manager, choose Join an existing HTCondor Pool and fill in the hostname of the central manager, e.g. computername where nextnanoHTCondorPool has been created.

4. Tic Submit jobs to HTCondorPool and choose Always run jobs and never suspend them. (Alternative: If you do not want other people to run jobs on your machine at all, select Do not run jobs on this machine or if you do not want other people to run jobs on your machine while you are working, select When keyboard has been idle for 15 minutes.. You can of course modify these settings later.)
Figure 16.1.1: Screenshot taken from nextnano\text{mat} with integrated HTCondor feature.
5. Fill in your domain name (Example: Your Windows domain, e.g. yourcompanyname.com (without www).) All PCs of your network should get the same domain name, this does not necessarily have to be your Windows domain.

6. Hostname of SMTP Server and email address of administrator (not needed currently, leave it blank)

7. Path to Java Virtual Machine (not needed currently, leave it blank)

8. Host with Read access: *

9. Host with Write access: ${CONDOR_HOST}, ${IP_ADDRESS}, *.yourdomainname.com, 192.168.178.*, (Replace *.cs.wisc.edu with your domain name and add your local IP subnet e.g. 192.168.178.*). On Windows you can find your IP subnet by opening the Command Prompt cmd.exe and typing in ipconfig.

10. Host with Administrator access * (or $(IP_ADDRESS))

11. Enable VM Universe No

12. Choose an installation directory and press next (e.g. C:\condor\). The directory Program Files is problematic due to write permissions, so we do not recommend using it.

13. Press Install and type in the Administrator password of your PC. (You need Administrator rights.)

14. Once installed, please restart the computer. Then your new pool or pool member should be up and running.

A few more setups

1. To be able to submit jobs from nextnanomat to HTCondor, you have to store your credentials once. Open a command shell and type the following command: condor_store_cred add

   • Enter your password and you are ready to submit your first HTCondor job.

   • If this does not work, try to enter condor_store_cred add -debug for more output information on the error.

2. Please make sure that nextnanomat has successfully found the HTCondor pool. In nextnanomat go to Tools -> Options -> Cloud computing. If everything is correctly set up, you will find the “HTCondor” section highlighted with green color, and the available computers show up in “Cluster”. If this is not the case, maybe you have not installed HTCondor on the computer where you are running nextnanomat. Please also check that the HTCondor installation path is correctly set within nextnanomat, e.g. the default path C:\condor might not be the one where you installed HTCondor.

### 16.2.1 Summary of settings (Example)

<table>
<thead>
<tr>
<th>Hostname (for HTCondor pool): computername.yourcompanyname.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy: &quot;Always run jobs&quot;</td>
</tr>
<tr>
<td>Accounting domain: yourcompanyname.com</td>
</tr>
<tr>
<td>Read access: *</td>
</tr>
<tr>
<td>Administrator: $(IP_ADDRESS)</td>
</tr>
</tbody>
</table>

### 16.2.2 Config file

You can find your HTCondor config settings in the file C:\condor\condor_config. Let’s look at an example below.

- Your company is called Simpson.
- Your Windows domain is called simpson.com.
- Your HTCondor pool shall have the name TheSimpsonsCondorPool.
- The HTCondor host that manages the HTCondor jobs has the computer name homer.simpson.com.
- Your computer is called lisa.simpson.com.
- The computers in your network have the IP range 192.168.188.* (or 2001:db8:2042::* in IPv6)

| RELEASE_DIR = C:\condor |
| LOCAL_CONFIG_FILE = $(LOCAL_DIR)\condor_config.local |
| REQUIRE_LOCAL_CONFIG_FILE = FALSE |
| LOCAL_CONFIG_DIR = $(LOCAL_DIR)\config |
| use SECURITY : HOST_BASED |
| #CONDOR_HOST: $(FULL_HOSTNAME) # on computer called homer |
| CONDOR_HOST: homer # on computer called lisa |
| COLLECTOR_NAME = TheSimpsonsCondorPool # only on computer called homer |
| #UID_DOMAIN = # empty if you do not have a domain |
| UID_DOMAIN = simpson.com |
| SOFT_UID_DOMAIN=TRUE # entry is missing if you do not have a domain |
| FILESYSTEM_DOMAIN = simpson.com # entry is missing if you do not have a domain |
| CONDOR_ADMIN = |
| ALLOW_READ = * |
| ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS), *.simpson.com, 192.168.188.*, 2001:db8:2042::* |
| ALLOW_ADMINISTRATOR = $(IP_ADDRESS) |
| #use POLICY : ALWAYS_RUN_JOBS |
| WANT_VACATE = FALSE |
| WANT_SUSPEND = TRUE |
| #DAEMON_LIST = MASTER SCHEDD COLLECTOR NEGOTIATOR STARTD # on computer called homer |
| #DAEMON_LIST = MASTER SCHEDD STARTD KBDD # on computer called lisa |
| if keyboard idle 15 minutes option was chosen |
| DAEMON_LIST = MASTER SCHEDD STARTD # on computer called lisa |

16.2.3 Configuring a pool without a domain

It is also possible to set up a HTCondor pool with computers that are not on a domain by using the NO_DNS option in the config file. This also provides an easy way to setup a mixed pool with Windows and Linux machines. Below are example configurations for central manager and submit/execute nodes for Windows and Linux, respectively. The user passwords on Windows machines need to be stored by condor_store_cred add command as above, on Linux machines no password is needed.

Central Manager on Windows:

| RELEASE_DIR = C:\condor |
| LOCAL_DIR = $(RELEASE_DIR) |
| LOCAL_CONFIG_FILE = $(LOCAL_DIR)\condor_config.local |
| REQUIRE_LOCAL_CONFIG_FILE = FALSE |
| use SECURITY : HOST_BASED |
| CONDOR_HOST = $(FULL_HOSTNAME) |
| COLLECTOR_NAME = EXAMPLE_POOL |
| NO_DNS = True |
| DEFAULT_DOMAIN_NAME = MY_DOMAIN |
| ALLOW_CONFIG = $(IP_ADDRESS) |
| ALLOW_ADMINISTRATOR = $(CONDOR_HOST), $(IP_ADDRESS), condor_pool@$(UID_DOMAIN), |
| #EXAMPLE_POOL |
| ALLOW_READ = $(IP_ADDRESS)/8, $(CONDOR_HOST) |
| ALLOW_WRITE = $(IP_ADDRESS)/8, $(CONDOR_HOST) |
| ALLOW_NEGOTIATOR = $(CONDOR_HOST), condor_pool@$(UID_DOMAIN) |

(continues on next page)
ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS)
DAEMON_LIST = MASTER STARTD SCHEDD NEGOTIATOR COLLECTOR

use POLICY : ALWAYS_RUN_JOBS
WANT_VACATE = FALSE
WANT_SUSPEND = TRUE

Submit/Execute Node on Windows:

RELEASE_DIR = C:\condor
LOCAL_DIR = $(RELEASE_DIR)
LOCAL_CONFIG_FILE = $(LOCAL_DIR)\condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = FALSE

use SECURITY : HOST_BASED

CONDOR_HOST = 192.168.188.xy #substitute with the ip address of the central manager
COLLECTOR_HOST = $(CONDOR_HOST)

NO_DNS = True
DEFAULT_DOMAIN_NAME = MY_DOMAIN

ALLOW_CONFIG = $(IP_ADDRESS), $(CONDOR_HOST)
ALLOW_ADMINISTRATOR = $(CONDOR_HOST), $(IP_ADDRESS), condor_pool@(UID_DOMAIN),...
ALLOW_READ = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_WRITE = $(IP_ADDRESS)/8, $(CONDOR_HOST)

ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8, $(CONDOR_HOST)
ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS),...
DAEMON_LIST = MASTER STARTD SCHEDD

use POLICY : ALWAYS_RUN_JOBS
WANT_VACATE = FALSE
WANT_SUSPEND = TRUE

Central Manager on Linux:

RELEASE_DIR = /usr
LOCAL_DIR = /var
LOCAL_CONFIG_FILE = /etc/condor/condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = false
LOCAL_CONFIG_DIR = /etc/condor/config.d

## Pathnames
RUN = $(LOCAL_DIR)/run/condor
LOG = $(LOCAL_DIR)/log/condor
LOCK = $(LOCAL_DIR)/lock/condor
SPOOL = $(LOCAL_DIR)/spool/condor
EXECUTE = $(LOCAL_DIR)/lib/condor/execute
BIN = $(RELEASE_DIR)/bin
LIB = $(RELEASE_DIR)/lib/condor

16.2. Recommended Installation Process 751
INCLUDE = $(RELEASE_DIR)/include/condor
SBIN = $(RELEASE_DIR)/sbin
LIBEXEC = $(RELEASE_DIR)/lib/condor/libexec
SHARE = $(RELEASE_DIR)/share/condor
GANGLIA_LIB64_PATH = /lib,/usr/lib,/usr/local/lib
PROCD_ADDRESS = $(RUN)/procd_pipe

use SECURITY : HOST_BASED

CONDOR_HOST = $(FULL_HOSTNAME)
COLLECTOR_NAME = EXAMPLE_POOL

NO_DNS = True
DEFAULT_DOMAIN_NAME = MY_DOMAIN

ALLOW_CONFIG = $(IP_ADDRESS)
ALLOW_ADMINISTRATOR = $(IP_ADDRESS), $(CONDOR_HOST), condor_pool@$(UID_DOMAIN),
→ EXAMPLE_POOL
ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS)/8
ALLOW_READ = $(CONDOR_HOST), $(IP_ADDRESS)/8
ALLOW_NEGOTIATOR = $(CONDOR_HOST), condor_pool@$(UID_DOMAIN)
ALLOW_ADVERTISE_MASTER = $(IP_ADDRESS)/8
ALLOW_ADVERTISE_STARTD = $(IP_ADDRESS)/8
ALLOW_ADVERTISE_SCHEDD = $(IP_ADDRESS)/8
ALLOW_DAEMON = condor_pool@$(UID_DOMAIN), condor@$(UID_DOMAIN), $(IP_ADDRESS)
DAEMON_LIST = MASTER, STARTD, SCHEDD, NEGOTIATOR, COLLECTOR

# Don't phone home
CONDOR_DEVELOPERS = NONE
CONDOR_DEVELOPERS_COLLECTOR = NONE

SSH_TO_JOB_SSHD_CONFIG_TEMPLATE = /etc/condor/condor_ssh_to_job_sshd_config_.template

Submit/Execute Node on Linux:

RELEASE_DIR = /usr
LOCAL_DIR = /var
LOCAL_CONFIG_FILE = /etc/condor/condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = false
LOCAL_CONFIG_DIR = /etc/condor/config.d

## Pathnames
RUN = $(LOCAL_DIR)/run/condor
LOG = $(LOCAL_DIR)/log/condor
LOCK = $(LOCAL_DIR)/lock/condor
SPOOL = $(LOCAL_DIR)/spool/condor
EXECUTE = $(LOCAL_DIR)/lib/condor/execute
BIN = $(RELEASE_DIR)/bin
LIB = $(RELEASE_DIR)/lib/condor
INCLUDE = $(RELEASE_DIR)/include/condor
SBIN = $(RELEASE_DIR)/sbin
LIBEXEC = $(RELEASE_DIR)/lib/condor/libexec
SHARE = $(RELEASE_DIR)/share/condor
GANGLIA_LIB64_PATH = /lib,/usr/lib,/usr/local/lib
PROCD_ADDRESS = $(RUN)/procd_pipe

(continues on next page)
16.3 Submitting jobs to HTCondor pool with nextnanomat

Submit job

1. Add a job to the Batch list in the Run tab.
2. Click on the Run in HTCondor Cluster button (button with triangle and network).

Show information on HTCondor cluster

1. Click on Show Additional Info for Cluster Simulation.
2. Press the Refresh button on the right.
3. The results of the condor_status command are shown, i.e. the number of compute slots are displayed.
4. You can select another HTCondor command such as condor_q to show the status of your submitted jobs, i.e. select condor_q, and then press the Refresh button.
   • You can type in any command in the line System command:, e.g. dir.
   • The button Open Documentation opens the online documentation (this website).

Results of HTCondor simulations

• Once your HTCondor jobs are finished, the results are automatically copied back to your simulation output folder <nextnano simulation output folder><name of input file>.
• For debugging purposes regarding the HTCondor job, you can analyze the generated log file, <input file name>.log.
16.4 Useful HTCondor commands for the Command Prompt

- `condor_submit <filename>.sub` Submit a job to the pool.
- `condor_q` Shows current state of own jobs in the queue.
  - `condor_q -nobatch -global -allusers` Shows state of all jobs in the cluster. Of all users.
  - `condor_q -goodput -global -allusers` Shows state and occupied CPU of all jobs in the cluster.
  - `condor_q -allusers -global -analyze` Detailed information for every job in the cluster.
  - `condor_q -global -allusers -hold` Shows why jobs are in hold state.
- `condor_status` Shows state of all available resources.
- `condor_status -long` Shows state of all available resources and many other information.
- `condor_status -debug` Shows state of all available resources and some additional information, e.g. WARNING: Saw slow DNS query, which may impact entire system: getaddrinfo(<Computername>) took 11.083566 seconds.
- `condor_rm` Remove jobs from a queue:
  - `condor_rm -all` Removes all jobs from a queue.
  - `condor_rm <cluster>.<id>` Removes jobs on cluster <cluster> with id <id> (It seems <cluster> can be omitted, and id is the JOB_IDS number.)
- `condor_release -all` If any jobs are in state hold, use this command to restart them.
- `condor_restart` Restart all HTCondor daemons/services after changes in config file.
- `condor_version` Returns the version number of HTCondor
- `condor_store_cred query` Returns info about the credentials stored for HTCondor jobs
- `condor_history` Lists the recently submitted jobs. If for a specific job ID the status has the value ST=C, then this job has been completed (C) successfully.
- `condor_status -master` returns Name, HTCondor Version, CPU and Memory of central manager
- **Open Command Prompt cmd.exe as Administrator. Type in:** `net start condor` This has the same effect as restarting your computer, i.e. the networking service condor is started. This is useful if you have changed your local condor_config file.

16.5 HTCondor Pool - Managing Slots

- Each PC runs one `condor_startd` daemon. By default, the `condor_startd` will automatically divide the machine into slots, placing one core in each slot. E.g. a 6-core computer with hyperthreading has 12 logical processors. Alternatively, the number of cores (or logical processors) can be distributed to the slots as follows.

```plaintext
SLOT_TYPE_1 = cpus=4
SLOT_TYPE_2 = cpus=4
SLOT_TYPE_3 = cpus=2
SLOT_TYPE_4 = cpus=1
SLOT_TYPE_5 = cpus=1
SLOT_TYPE_1_PARTITIONABLE = TRUE
SLOT_TYPE_2_PARTITIONABLE = TRUE
SLOT_TYPE_3_PARTITIONABLE = TRUE
SLOT_TYPE_4_PARTITIONABLE = TRUE
SLOT_TYPE_5_PARTITIONABLE = TRUE
```

(continues on next page)
NUM_SLOTS_TYPE_1 = 1
NUM_SLOTS_TYPE_2 = 1
NUM_SLOTS_TYPE_3 = 1
NUM_SLOTS_TYPE_4 = 1
NUM_SLOTS_TYPE_5 = 1

- **PartitionableSlot**: For SMP (symmetric multiprocessing) machines, a boolean value identifying that this slot may be partitioned.
- **DynamicSlot**: For SMP machines that allow dynamic partitioning of a slot, this boolean value identifies that this dynamic slot may be partitioned.
- **SlotID**: For SMP machines, the integer that identifies the slot.
- A useful command might be: `condor_status -af Name TotalCpus DynamicSlot PartitionableSlot SlotID`. It returns the requested properties of each slot:
  - Name
  - TotalCpus
  - DynamicSlot
  - PartitionableSlot
  - SlotID

### 16.5.1 Dynamic slots

In our pool we have chosen dynamic partitioning which gives full flexibility. For instance, a quad-core CPU that is dynamically partitioned can accept

- 4 single-threaded jobs (`request_cpus = 1`)
- 2 jobs with 2 threads each (`request_cpus = 2`)
- 2 jobs of which one is single-threaded (`request_cpus = 1`) and the other uses 3 threads (`request_cpus = 3`)
- 1 job with 4 threads (`request_cpus = 4`).

```plaintext
# Dynamic partitioning
# We use HTCondors dynamic partitioning mechanism.
# Each PC has one partitionable whole machine slot.
# (It seems that hyperthreading is not taken into account.)
NUM_SLOTS = 1
NUM_SLOTS_TYPE_1 = 1
SLOT_TYPE_1 = 100%
SLOT_TYPE_1_PARTITIONABLE = true
SlotWeight = Cpus
```
16.6 Machine states

A machine is in any of the following 6 states. The most important one are Owner, Unclaimed, Claimed.

- **Owner**: The machine is being used by the machine owner, and/or is not available to run HTCondor jobs. When the machine first starts up, it begins in this state.
- **Unclaimed**: The machine is available to run HTCondor jobs, but it is not currently doing so.
- **Claimed**: The machine is available to run jobs, and it has been matched by the negotiator with a specific schedd. That schedd just has not yet claimed this machine. In this state, the machine is unavailable for further matches.
- **Preempting**: The machine was claimed by a schedd, but is now preempting that claim for one of the following reasons.
  - The owner of the machine came back
  - Another user with higher priority has jobs waiting to run.
  - Another request that this resource would rather serve was found.
  - "Backfill": (not relevant for us)

16.7 Machine activities

Each machine state can have different activities. The machine state Claimed can have one out of these four activities.

- Idle:
- Busy:
- Suspended:
- Retiring:

16.8 Configuration options for the Central Manager computer

With this option in the `condor.config` file on the central manager, one can set a policy that the jobs are spread out over several machines rather than filling all slots of one computer before filling the slots of the other computers.

```plaintext
##------nn: SPREAD JOBS BREADTH-FIRST OVER SERVERS
##-- Jobs are "spread out" as much as possible,
## so that each machine is running the fewest number of jobs.
NEGOTIATOR_PRE_JOB_RANK = isUndefined(RemoteOwner) * (~ SlotId)
```

16.9 FAQ

**Q**: I submitted a job to HTCondor, but nothing happens. `nextnano` says “transmitted”.

**A**: It could be that `nextnano` does not have read in all required settings. You can try to type in the command line `condor_restart`. Please make sure that you entered your credentials using `condor_store_cred add -debug`. You should then start `nextnano` again.

**Q**: I submitted a job to HTCondor, but the Batch line of `nextnano` is stuck with `preparing`. What is wrong?

**A1**: Did you store your credentials after the installation of HTCondor? If not, enter `condor_store_cred add` into the command prompt to add your password, see above (Recommended Installation Process).
A2: Did you change your password recently? If yes you have to reenter your credentials for HTCondor. Enter `condor_store_cred add` into the command prompt to add your password, see above (Recommended Installation Process). If this does not work, try to enter `condor_store_cred add -debug` for more output information on the error.

Q: I specified target machines in Tools - Options. Afterwards every submitted job to HTCondor is stuck with transmitting. What is wrong?

A: The value for `UID_DOMAIN` within the condor_config file needs to be the same for every computer of your cluster. (You can easily test it in a command prompt with `condor_status -af uiddomain` If it's not the same value, no matching computer will be found and the job won’t be transmitted successfully.

### 16.10 Problems with HTCondor

#### 16.10.1 Error: communication error

If you receive the following error when you type in `condor_status`

```
C:\Users\<your user name>\>condor_status
Error: communication error
CEDAR:6001:Failed to connect to <123.456.789.123>
```

you can check whether the computer associated with this IP address is your HTCondor computer using the following command.

```
nslookup 123.456.789.123
```

It is also a good idea to type in

```
nslookup
```

This will return the name of the Default Server that resolves DNS names. If it is not the expected computer, you can open a Command Prompt as `Administrator` and type in `ipconfig /flushdns` to flush the DNS Resolver Cache.

```
C:\Users\<your user name>\>ipconfig /flushdns
```

If the DNS address cannot be resolved correctly it could be related to a VPN connection that has configured a different default server for Domain Name to IP address mapping. E.g. if your Windows Domain is called contoso.com (which is only visible within your own network and your own HTCondor pool) but your DNS is resolved to www.contoso.com (which might be outside your local HTCondor pool).

#### 16.10.2 Error: `condor_store_cred add` failed with Operation failed. Make sure your ALLOW_WRITE setting include this host.

Solution: Edit `condor_config` file and add host, i.e. local computer name (here: nn-delta).

```
ALLOW_WRITE = $(CONDOR_HOST), $(IP_ADDRESS)
```
16.10.3 Error? Check the Log files

If you encounter any strange errors, you can find some hints in the history or Log files generated by HTCondor. You can find them here:

C:\condor\spool
  • history
C:\condor\log
  • CollectorLog
  • MasterLog
  • MatchLog
  • NegotiatorLog
  • ProcLog
  • SchedLog
  • ShadowLog
  • SharedPortLog
  • StarterLog
  • StartLog

More details can be found here: Logging in HTCondor

16.11 Known bugs

• HTCondor < 8.9.5 works with all nextnano executables
• HTCondor >= 8.9.5 works with nextnano executables newer than 2020-Jan

16.12 Run your custom executable on HTCondor with nextnanomat

You can even run your own executable with nextnanomat locally or on HTCondor! We tested the following programs:

• HelloWorld.exe
• Quantum ESPRESSO (pw.exe)
• ABINIT (abinit.exe)

16.12.1 Input file identifier

An input file identifier is a special string in the input file that signals to nextnanomat whether the input file is an input file for the nextnano++, nextnano3, nextnano.NEGF or nextnano.MSB software, or for a custom executable.
### 16.12.2 Settings for Hello World (HW)

In nextnano\textsuperscript{mat}, we need the following settings:

- **Path to executable file**: e.g. D:\HW\HelloWorld.exe
- **Input file identifier**: e.g. HelloWorld
- **Working directory**: Select ‘Simulation output folder’
- **HTCondor**: Output folder and files (transfer\_output\_files = ...):

Open input file input\_file\_for\_HelloWorld.in (or any other input file that contains the string HelloWorld) and run the simulation either locally or on HTCondor.

### 16.12.3 Settings for Quantum ESPRESSO (QE)

Our folder structure is

- D:\QE\inputfile\My\_QE\_inputfile\_in (QE input file)
- D:\QE\input\pseudo\C\_UPF (pseudopotential file for atom species ‘C’ as specified in input file)
- D:\QE\exe\pw.exe (QE executable file)
- D:\QE\exe\*.dll (all dll files needed by pw.exe)
- D:\QE\working\_directory\QE\_nextnanomat\_HTCondor\_bat (batch file)

In nextnano\textsuperscript{mat}, we need the following settings:

- **Path to executable file**: e.g. D:\QE\working\_directory\QE\_nextnanomat\_HTCondor\_bat
- **Path to folder with additional files**: D:\QE\%
- **Input file identifier**: e.g. &control
- **Working directory**: Select ‘Simulation output folder’
- **HTCondor**: Output folder and files (transfer\_output\_files = ...):
- **(Additional arguments passed to the executable: $INPUTFILE)**

The batch file (*.bat) contains the following content:

```
  .\exe\pw.exe -in .\inputfile\My\_QE\_inputfile\_in
```

This means that relative to the working directory, pw.exe is started, and the specified input file is read in. In this input file, the following quantities are specified:

- **C\_UPF**: name of pseudopotential file
- **./input/pseudo/**: path to pseudopotential file C\_UPF

Open input file My\_QE\_inputfile\_in and run the simulation either locally or on HTCondor.

Things that could be improved:

- Write all files into output folder created by nextnano\textsuperscript{mat}. In particular, the folder output/ should be moved.
- condor\_exec.exe is deleted (better: do not copy it back)
- all *.dll files should be deleted (better: do not copy them back)
- Don’t copy back *.exe and *.dll files (both HTCondor and local)
16.12.4 Settings for ABINIT

Our folder structure is

- D:\abinit\inputfile\t30.in (ABINIT input file)
- D:\abinit\input\* (input files needed by ABINIT)
- D:\abinit\exe\abinit.exe (ABINIT executable file)
- D:\abinit\exe\*.dll (all dll files needed by abinit.exe)
- D:\abinit\working_directory\abinit_nextnanomat.bat (batch file)
- Path to executable file: e.g. D:\abinit\working_directory\abinit_nextnanomat.bat
- Path to folder with additional files: D:\abinit\inputfile\t30.in
- Input file identifier: e.g. acell
- Working directory: Select ‘Simulation output folder’
- HTCondor: Output folder and files (transfer_output_files = ...): .
- Additional arguments passed to the executable: (empty)

The batch file (*.bat) contains the following content:

```
.exe\abinit.exe < .\input\ab_nextnanomat_HTCondor.files
```

This means that relative to the working directory, abinit.exe is started, and the specified input file is read in. In this input file, the following quantities are specified:

- .\inputfile\t30.in: name of input file
- .\input\14si.pspnc:

Open input file t30.in and run the simulation either locally or on HTCondor.

Notes

- condor_exec.exe is deleted (better: do not copy it back)
- all *.dll files should be deleted (better: do not copy them back)
- Don’t copy back *.exe and *.dll files (both HTCondor and local)
CHAPTER
SEVENTEEN

FREQUENTLY ASKED QUESTIONS (FAQ)

17.1 General

- Copyright Statement
- Are there any video tutorials available?
- Hardware requirements for nextnano? I want to buy a new computer. What shall I buy?
- How shall I cite the nextnano software in publications?

17.1.1 Copyright Statement

Copyright

17.1.2 Are there any video tutorials available?

Yes, there are! Check out our Sway presentation and our YouTube channel.

17.1.3 Hardware requirements for nextnano? I want to buy a new computer. What shall I buy?

The nextnano software even runs on Laptops. Therefore, for most typical simulations, you don’t have to buy a special computer. Still, you might have CPU-intensive calculations which require more horsepower. In this case, the following configurations or their equivalent will give excellent performance for about 1.000 € or less:

**CPU+RAM** Intel i7-8700, 6 cores, 3.2 GHz (Coffee Lake) with 32 GB DDR4-2666 RAM, or Intel i7-7700, 4 cores, 3.6 GHz (Kaby Lake) with 32 GB DDR4-2400 RAM.

Note: These CPUs can be bought as “boxed” (including a default cooler) or “tray” (without default cooler). However, both CPUs are too noisy with the default cooler in the boxed variant when running simulations. We recommend buying a large additional CPU cooler.

**Main board** Any compatible motherboard will work, please select according to your extensibility needs, required peripheral ports, etc. The integrated graphics on the mainboard usually suffices for office use and development. But be prepared to add a dedicated graphics card for CAD, multi-monitor setups and the likes.

**Storage** E.g. a 500 GB SSD for the OS and programs, together with a 2 TB HD for simulation data.

**Power supply** E.g. 500 W to allow future upgrades such as a dedicated graphics card.
Computer case  Select according to available space, future extensibility, and aesthetic desire. Silent cases are recommended for reducing noise. Please make sure that there is enough clearance to fit your CPU cooler and large additional components such as graphics cards inside.

Recommended BIOS settings (if available)

- CPU: Hyperthreading  Enabled (accelerates compilation)
- CPU: VT-d  Enabled (accelerates virtualization)
- CPU: Hardware prefetch  Enabled (accelerates linear memory accesses)
- RAM: XMP enabled (if disabled, RAM much slower than maximum supported speed, e.g. 2166 instead of maximum supported 2666 (DDR4-2666))

(For the experts: If you are planning to have your PC assembled from components, please use tools such as prime95, Intel Extreme Tuning Utility, and/or Intel Linpack to test system stability and adequate cooling under extended heavy load.)

Example: Our latest computers have the following configurations

i7-8700 (purchased 2018-Oct: ~1.000 EUR, purchased 2018-Dec: ~900 EUR)

- CPU: Intel i7-8700, 6 cores, 3.2 GHz (Coffee Lake) (a large additional CPU cooler is recommended, see comments above)
- CPU cooler, e.g. be quiet! Pure Rock
- RAM: 32 GB (DDR4-2666), e.g. Corsair Vengeance
- Storage: 500 GB SDD + 3 TB HDD
- Motherboard with integrated graphics processing unit, e.g. ASRock Z370M-ITX/ac (CPU socket 1151)
- Power supply: e.g. be quiet! Pure Power 10-CM or 11, 500 W
- Computer case: Midi Tower, e.g. Zalman R1 or be quiet! Pure Base 600

i7-7700 (purchased 2018-Jan, ~900 EUR)

- CPU: Intel i7-7700, 4 cores, 3.6 GHz (Kaby Lake) (a large additional CPU cooler is recommended, see comments above)
- CPU cooler, e.g. be quiet! Pure Rock Slim
- RAM: 32 GB (DDR-2400), e.g. Corsair Vengeance
- Storage: 256 GB SDD + 2 TB HDD
- Motherboard with integrated graphics processing unit, e.g. ASRock Z270M-ITX/ac
- Power supply: e.g. Pure Power 10, 500 W ATX 2
- Computer case: Midi Tower, e.g. Zalman R1

17.1.4 How shall I cite the nextnano software in publications?

You can cite any of the following papers:

- nextnano: General Purpose 3-D Simulations
  S. Birner, T. Zibold, T. Andlauer, T. Kubis, M. Sabathil, A. Trellakis, P. Vogl

- The 3D nanometer device project nextnano: Concepts, methods, results
  A. Trellakis, T. Zibold, T. Andlauer, S. Birner, R. K. Smith, R. Morschl, P. Vogl
For simulations including electrolytes, you should cite:

- Theoretical model for the detection of charged proteins with a silicon-on-insulator sensor
  
  S. Birner, C. Uhl, M. Bayer, P. Vogl
  

For simulations that use the Contact Block Reduction method (CBR) (ballistic transport), you should cite any of the following papers:

- Efficient method for the calculation of ballistic quantum transport
  
  D. Mamaluy, M. Sabathil, P. Vogl
  

- Ballistic quantum transport using the contact block reduction (CBR) method - An introduction
  
  S. Birner, C. Schindler, P. Greck, M. Sabathil, P. Vogl
  

**nextnano.MSB software:** For simulations that use the multi-scattering Büttiker (MSB) probe model (NEGF), you should cite:

- Efficient method for the calculation of dissipative quantum transport in quantum cascade lasers
  
  P. Greck, S. Birner, B. Huber, P. Vogl
  
  Optics Express **23**, 6587

**nextnano.NEGF software:** For simulations that use the NEGF method, you should cite:

- Contrasting influence of charged impurities on transport and gain in terahertz quantum cascade lasers
  
  T. Grange
  

For simulations that use the NEGF algorithm included in the nextnano$^3$ software, you should cite any of these publications:

- Modeling techniques for quantum cascade lasers
  
  C. Jirauschek, T. Kubis
  

- Theory of non-equilibrium quantum transport and energy dissipation in terahertz quantum cascade lasers
  
  T. Kubis, C. Yeh, P. Vogl, A. Benz, G. Fasching, C. Deutsch
  

There might be further papers in the literature that are more suited to be cited in certain cases.
17.2 Simulation

**nextnano Product related**

- Which features have been implemented recently?
- What is the difference between nextnano³ and nextnano++?
- Can I convert nextnano³ input files into nextnano++ input files?
- How can I track how much memory is used during the simulations?
- Can I pass additional command line arguments to the executable?
- How can I speed up my calculations with respect to CPU time?
- Can I take advantage of parallelization of the nextnano software on multi-core CPUs?
- Dirichlet vs. Neumann boundary conditions
- Quasi-Fermi level
- I don’t understand the $k \cdot p$ parameters
- Can I add new materials to the database?

**GUI nextnanomat related**

- How do I produce 1D slices through the 2D plots in the GUI?
- Is there a way to produce a 1D (or 2D) plot of some result, for example the probability density with the conduction band edge superimposed?
- In other words, can the GUI show multiple plots at once?
- What is the difference between “List view” and “Tree view”?  

17.2.1 nextnano Product related

**Which features have been implemented recently?**

- nn++: Added periodic repetition of quantum regions
- nmm: New colormaps
- nmm: Arrow showing difference value between two selected graphs
- nmm: HTCondor support for mixed Linux/Windows pool
- nn++: Added input files for GaN bulk k.p dispersion tutorial (has already existed for nn3)
- nn++: Added input files for self-consistent k.p multi-QW intersubband tutorial (has already existed for nn3)
- nn3: Bulk band structure with 14-band and 30-band k.p
- nn++: Added electron injection (e.g. by electron beam) into structure definition
- nn++: Integration of nextnano.MSB into nextnano++, incl. nextnano.MSB sample files into installer
- nn++: Output reflection components of CBR transmissions
- nn3: Improvements for CBR
- nn++/nn3: New 2D CBR input files (QPC)
- nn3: Improvement of 2DEG mobility input files (e.g. GaN)
- nn++/nn3: Synonyms in material database (e.g. Al(x)In(x)As and In(x)Al(1-x)As)
• nn++: Added input file for tutorial: “QCSE” (existed already for nn3)
• nn++/nn3: Input files for tutorial: “Intersubband absorption in infinite QW”
• nn3: Input files for intersubband absorption in Ge/SiGe QWs
• nn++: Input files for absorption in Ge/SiGe QWs
• nn++: Input files for absorption in Ge
• nn++: Calculation of reflection and extinction coefficient
• nn++: Gaussian and Lorentzian broadening for optical absorption
• nn++: Improvements for optical absorption tutorial (k.p)
• nn++: Improvements for optical absorption (k.p)
• nn++: Improvements for k.p (speed: k=0 subspace expansion)
• nn++/nn3: Added more tutorial input files to samples folder
• nn++: New UVC LED AlGaN/GaN input files
• nn++: Improvements for intersubband absorption (k.p)
• nn3: Improvements for intersubband absorption (single-band, simple k.p model, k.p)
• nnpy: Intersubband absorption script
• nn++: Solar cell features, irradiation
• nn++: Added quaternaries and quinternaries to database; AlScN, AlYN, . . .
• nn3: Grid lines can be independent of region object boundaries (has always been the case for nn++)
• nn++/nn3: Added XML support to input files
• nn++/nn3: New region objects: circle/sphere
• nn++/nn3: New region objects: triangle, polygonal_prism, regular_prism, hexagonal_prism, polygonal_pyramid, regular_pyramid, hexagonal_pyramid
• nnpy: New Python tool: nextnanopy
• nn++: Array of different biases is allowed in addition to bias sweep using steps
• nn++: Output of emission spectrum for LEDs based on classical or quantum density
• nn++: Output of energy resolved density n(E) and n(x,E)
• nn++: Improved convergence and speed for current calculations
• nn++: More intuitive setting in run{}
• nn++/nn3: 2nd order piezo constants and 2nd order piezo tutorial
• nn3: Bulk band structure with pseudopotential method
• nn3: Bulk band structure with tight-binding method (improvements)
• nn++/nn3: Piezo/pyro tutorial for arbitrary (hkil) in wurtzite
• nn++: MOSFET tutorial
What is the difference between nextnano\(^3\) and nextnano++?

**The short answer is:** Use the nextnano++ software. This is the software where we put on most effort in improving it by adding new features. The nextnano\(^3\) software is mainly distributed for historical reasons.

nextnano\(^3\) is written in Fortran. It has been developed at the Walter Schottky Institute from 1999 to 2010. nextnano++ is written in C++. It has been developed at the Walter Schottky Institute from 2004 to 2008. The software packages have been written by different people that were working at the Walter Schottky Institute of the Technische Universität München in the Theoretical Semiconductor Physics group of Prof. Peter Vogl.

Essentially, both software packages cover the same physics and methods, namely

- the strain equation
- the Poisson equation
- the Schrödinger equation
- the drift-diffusion current equation.

Additionally, the nextnano\(^3\) software includes

- solar cells
- electrolytes
- graphene
- tight-binding (for bulk and one-dimensional superlattices)
- the self-consistent CBR method (1D, 2D and 3D)
- the NEGF method (1D only) which is particularly suited for quantum cascade lasers.

In contrast to nextnano++, nextnano\(^3\) is not able to

- treat the magnetic field within the \(\mathbf{k} \cdot \mathbf{p}\) approach
- calculate the g tensor
- include quaternary materials.

nextnano++ is much faster for drift-diffusion calculations and for 2D/3D simulations. For instance, if you work on LEDs, MOSFETs or Quantum Dots, nextnano++ is much better suited.

There are some applications where it is irrelevant which software to use. In this case we recommend to use both. This has the advantage that the results of one software can be compared to the results of the other software in order to gain more confidence in them. For some applications, one software package should be preferred. Please contact <support[@]nextnano.com> in order to find out which software to choose for your particular application.

**nextnano\(^3\) syntax**

- "$" character for the keywords: $regions ... $end_regions
- "%" character for the variables: %QuantumWellWidth = 5.0
- "!" character for comments: ! This is a comment. (# is supported by both nextnano++ and nextnano\(^3\) as a comment sign.)

**nextnano++ syntax**

- "{}" brackets for the keywords: structure{ ... }
- "$" character for the variables: $QuantumWellWidth = 5.0
- "#" character for comments: # This is a comment. (# is also supported by nextnano\(^3\)).
Can I convert nextnano³ input files into nextnano++ input files?

Within nextnanomat, there is an experimental feature to convert a nextnano³ input file to a nextnano++ input file. How to use this automatic conversion:

In the menu select Tools ==> Convert nextnano³ input file to nextnano++

When you use the automatic conversion of nextnano³ input file into nextnano++, you will find that it probably does not work completely.

If you save and run the nextnano++ input file that has been converted and that has the suffix _nnp.in, very likely some errors appear indicating which line(s) to change. Then some manual adjustments are needed, but the rough structure should help a lot for the conversion.

How can I track how much memory is used during the simulations?

Tools ==> Options ==> Expert settings ==> Include memory usage in log file (units: MB)

Can I pass additional command line arguments to the executable?

Yes, this is possible. Go to Tools ==> Options ==> Expert settings ==> Command line

For nextnano³, one could use for instance: -database "D:\My folder\nextnano3\Syntax\my_database_nn3.in

-threads 4

How can I speed up my calculations with respect to CPU time?

The most obvious way is to reduce the number of grid points you are using. For instance, for the following pn junction simulation, a grid spacing of 1 nm was used (gray lines in Figure 17.2.1.1). If one is using a coarse grid of only 10 nm, the calculated values (squares in Figure 17.2.1.1) agree very well with the calculated values of the thin lines.

The difference in CPU time comes from the fact that for the 10 nm resolution the dimension of the matrix that is used for discretizing the Poisson equation is 30, while in the case for the 1 nm grid spacing it has the dimension 300. The proper choice of an optimal grid spacing is very relevant for 2D and 3D simulations, as can be seen in the following.

1D simulation (length of sample: x = 300 nm)

- 1 nm grid spacing: dimension of Poisson matrix: \( N = 300 \)
- 10 nm grid spacing: dimension of Poisson matrix: \( N = 30 \)

2D simulation (length of sample: x = 300 nm, y = 300 nm)

- 1 nm grid spacing: dimension of Poisson matrix: \( N = 300 \cdot 300 = 90,000 \)
- 10 nm grid spacing: dimension of Poisson matrix: \( N = 30 \cdot 30 = 900 \)

3D simulation (length of sample: x = 300 nm, y = 300 nm, z = 300 nm)

- 1 nm grid spacing: dimension of Poisson matrix: \( N = 300 \cdot 300 \cdot 300 = 27,000,000 \)
- 10 nm grid spacing: dimension of Poisson matrix: \( N = 30 \cdot 30 \cdot 30 = 27,000 \)

If a quantum mechanical simulation is performed, the numerical effort of eigenvalue solvers increases with the number of grid points \( N \) with order \( O(N^2) \).
Figure 17.2.1.1: Hole (blue) and electron (red) densities of the pn junction in units of $10^{18} \text{cm}^3$. The gray lines are from simulations using a 1 nm grid spacing. The squares are from a simulation that uses only a 10 nm grid resolution. Note that the center coordinate of this plot is x=160 nm. The depletion width for the holes is around wp50 nm, for the electrons it is wn10 nm which is of the order of the grid spacing. Even in this case, the calculated electron density is reasonably accurate.

Can I take advantage of parallelization of the nextnano software on multi-core CPUs?

The short answer is: Some numerical routines are parallelized which is done automatically. These are the numerical routines, e.g. for calculating the eigenvalues with a LAPACK solver (which itself uses BLAS).

The long answer is: The nextnano software includes the Intel® Math Kernel Library (MKL). MKL includes the BLAS and LAPACK library routines for numerical operations. The MKL dynamically changes the number of threads.

- nextnano++ - uses MKL (parallel version) The executables that are compiled with the Intel and Microsoft compilers use MKL (parallel version). The executable that is compiled with the GNU compiler (gcc/gfortran) uses the nonparallelized version of the BLAS and LAPACK source codes available from netlib webpage.
- nextnano - uses MKL (parallel version) The executables that are compiled with the Intel and NAG (64-bit) compilers use MKL (parallel version). The executables that are compiled with the GNU compiler (gfortran) and NAG (32-bit) use the nonparallelized version of the BLAS and LAPACK source codes available from netlib webpage. There is a nextnano executable available that uses OpenMP parallelization for
  - CBR (parallelization with respect to energy grid)
  - NEGF (parallelization with respect to energy grid and further loops) number-of-MKL-threads = 8
  - Calculation of eigenstates for each $k$ (1D and 2D simulations)
  - Matrix-vector products of numerical routines Note: Not all operations are thread-safe, e.g. one cannot combine $k$ parallelization with the ARPACK eigenvalue solver. Only for this executable, the flag number-of-parallel-threads = 4 has an effect. The NEGF keyword also supports number-of-MKL-threads = 4 (0 means dynamic with is recommended) and MKL-set-dynamic = yes
The NEGF algorithms (nextnano.NEGF, nextnano.MSB, CBR) include matrix-matrix operations which are well parallelized within the BLAS routines.

If e.g. 4 nextnano simulations are running in parallel on a quad-core CPU, i.e. 4 nextnano executables are running simultaneously and each of them is using calls to the parallelized MKL library simultaneously, the total performance might be slower compared to running these simulations one after the other. In this case using a nextnano executable compiled with the serial version of the Intel MKL could be faster.

In fact, it strongly depends on your nextnano application (e.g. 1D vs. 3D simulation, LAPACK vs. ARPACK eigenvalue solver, ...) if you benefit from parallelization or not. In general, the best parallelization can be obtained if you run several nextnano simulations in parallel. For instance, you could do parameter sweeps (e.g. sweep over quantum well width) using nextnanomat’s Template feature, i.e. if you run 4 simulations simultaneously on a quad-core CPU, e.g. for 4 different quantum well widths.

**Dirichlet vs. Neumann boundary conditions**

There are three different boundary conditions that we use:

- **periodic**: \( \psi(x = 0) = \psi(x = L) \)
- **Dirichlet**: \( \psi(x = 0) = \psi(x = L) = 0 \), and
- **Neumann**: \( \frac{d\psi}{dx} \) = const at the left \((x = 0)\) and right \((x = L)\) boundary. Typically, const = 0.

By specifying both Dirichlet and Neumann boundary conditions, the system would be over-determined.

**Quasi-Fermi level**

So-called quasi-Fermi levels which are different for electrons \( E_{F,n} \) and holes \( E_{F,p} \) are used to describe nonequilibrium carrier concentrations. In equilibrium the quasi-Fermi levels are constant and have the same value for both electrons and holes, \( E_{F,n} = E_{F,p} = 0 \) eV. The electron current is proportional to the electron mobility \( \mu_n(x) \), carrier density \( n(x) \) and the gradient of the quasi-Fermi level of the carriers, \( \nabla E_{F,n}(x) \), and analogously for the holes.

**I don’t understand the \( k \cdot p \) parameters**

In the literature, there are two different notations used:

- **Dresselhaus–Kip–Kittel (DKK):** \( L, M, N^+, N^- \) (zinc blende); \( L_1, L_2, M_1, M_2, M_3, N_1^+, N_1^-, N_2^+, N_2^- \) (wurtzite)
- **Luttinger parameters:** \( \gamma_1, \gamma_2, \gamma_3, \kappa \) (zinc blende); Rashba–Sheka–Pikus (RSP) parameters \( A_1, A_2, A_3, A_4, A_5, A_6, A_7 \) (wurtzite)

They are equivalent and can be converted into each other.

Some authors only use 3 parameters \( L, M, N \) (or \( \gamma_1, \gamma_2, \gamma_3 \)) which is fine for bulk semiconductors without magnetic field but not for heterostructures because the latter require 4 parameters, i.e. \( N^+, N^- \) (instead of \( N \) only) or \( \kappa \). If these parameters are not known, they can be approximated.

There are different \( k \cdot p \) parameters for

- 6-band \( k \cdot p \) and
- 8-band \( k \cdot p \).
The 8-band $k \cdot p$ parameters can be calculated from the 6-band parameters taking into account the temperature dependent band gap $E_{\text{gap}}$ and the Kane parameter $E_P$ (zinc blende). For wurtzite the parameters are $E_{\text{gap}}$ and the Kane parameters $E_{P1}, E_{P2}$.

The 8-band Hamiltonian also needs the conduction band mass parameter $S$ (zinc blende) or $S_{1,2}$ (wurtzite). They can be calculated from the conduction band effective mass $m_c$, the band gap $E_{\text{gap}}$, the spin-orbit split-off energy $\Delta_{so}$ and the Kane parameter $E_P$ (zinc blende). For wurtzite the parameters are $m_{c,\parallel}, m_{c,\perp}, E_{\text{gap}}, \Delta_{so}$, the crystal-field split-off energy $\Delta_{cr}$ and the Kane parameters $E_{P1}, E_{P2}$.

Finally there is the inversion asymmetry parameter $B$ for zinc blende. For wurtzite there are $B_1, B_2, B_3$.

For more details on these equations, please refer to Section 3.1 The multi-band $k \cdot p$ Schrödinger equation in the PhD thesis of S. Birner.

**Spurious solutions** Some people rescale the 8-band $k \cdot p$ in order to avoid spurious solutions. The 8-band $k \cdot p$ parameters can be calculated from the 6-band parameters taking into account the band gap $E_{\text{gap}}$, the spin-orbit split-off energy $\Delta_{so}$ and the Kane parameter $E_P$ (zinc blende). For wurtzite the parameters are $E_{\text{gap}}$, the spin-orbit split-off energy $\Delta_{so}$ and the Kane parameters $E_{P1}, E_{P2}$.

For more details, please refer to Section 3.2 Spurious solutions in the PhD thesis of S. Birner.

**Specific implementation nextnano++** See section kp_8band() in quantum().

**Specific implementation nextnano**

- See section Choice of $k \cdot p$ parameters in Snumeric-control.
- See section $k \cdot p$ parameters in Which material parameters are used?.
- See section Luttinger-parameters in Sbinary-zb-default.

**Can I add new materials to the database?**

Sure!

**Option 1** The material parameters are contained in ASCII text files. You can find them in the installation folder:

nextnano$^3$ software:

```
C:\Program Files (x86)\nextnano\nextnano++\Syntax\database_nnp.in
```

database_nnp.in in nextnano++ software: C:\Program Files (x86)\nextnano\<date>\nextnano3\Syntax\database_nn3.in These files can be edited with any text editor such as notepad++ (available free of charge).

More information on how to add materials to the database:

- nextnano++ software can be found https://www.nextnano.com/manual/nextnanoplus/database/index.html,

It is best if you search for a material such as “GaSb” and then simply use “Copy & Paste” to reproduce all relevant entries and then you rename “GaSb” to something like “GaSb_test”. Finally, you adjust the necessary material parameters that you need. In most cases, you don’t have to replace all material parameters. It is only necessary to replace the ones that you need in the simulation.

It is a good idea to save the new database to a new location such as

```
C:\Users\<user name>\Documents\nextnano\My Database\database_nnp_GaSb_modified.in
```
You can then read in the new nextnano++ (or nextnano$^3$) database specifying the location within the Tools Options of nextnanomat.
Tools => Options... => Material database => nextnano++/nextnano³
database file:

Option 2 A quicker way is the following. You can overwrite certain material parameters in the input
file rather than entirely defining new materials. For instance if you need “HfO2”, you could use
the material “SiO2” and just change the static dielectric constant and conduction and valence
band edges or any other relevant parameters that you need. So basically, you are using the
material “SiO2” a modified static dielectric constant and band edges.

More information on how to add materials to the

• nextnano++ software can be found on https://www.nextnano.com/manual/nexnanoplus/
  input_file/keywords/database.html,

• nextnano³ software can be found on https://www.nextnano.com/manual/nextnano3/
  input_file/keywords/binary-zb-default.html.

Please note that we treat all materials to be either of the crystal structure

• zinc blende (including diamond type) or
• wurtzite.

17.2.2 GUI nextnanomat related

How do I produce 1D slices through the 2D plots in the GUI?

1. Visualize horizontal and vertical slices:
   Menu View => Show Horizontal and Vertical Slices (or click on appropriate button)

2. Export horizontal or vertical slices:
   Right-click on 2D graph, Export horizontal/vertical slice as data (.dat) or as
   image (.png).

Is there a way to produce a 1D (or 2D) plot of some result, for example the probability density
with the conduction band edge superimposed?

In other words, can the GUI show multiple plots at once?

Yes, this is possible. Multiple output files can be shown at once, see Overlay feature documentation.

What is the difference between “List view” and “Tree view”?

In/For the Output tab of nextnanomat, you can switch between these options using: Tools =>
Options => View => Output folder browser

List view is the default and shows a list of the top level output folders. Tree view additionally
offers the possibility of showing the subfolders within the output folder panel.

If you want to sort your output folders by date, List view is better suited.
17.3 Licensing

• Which types of licenses exist?
• How many people can use the software simultaneously?
• Which license do I need for Cluster computing

17.3.1 Which types of licenses exist?

There are three types of licenses:

• University license
• Government institution license
• Company license

17.3.2 How many people can use the software simultaneously?

• A University license and a Government institution license are issued to a particular research group (e.g. a professor or group leader) and can be used by all group members simultaneously.

• A Company license applies to a single and named user or a single-PC.

The university and government institution licenses can be used on several computers simultaneously. A university license includes 15 computers, a government institution license includes 10 computers.

The license is an annual license. After the license has expired, no further simulations can be done. Visualization of previous results of calculations is still possible.

If a user has a valid license, this license can also be installed on a private computer.

17.3.3 Which license do I need for Cluster computing

Any computer that has the free HTCondor software installed can be connected to a licensed computer. There is no need to activate a license for these computers or install nextnano on them. They can be used to execute a batch list of nextnano simulations.

17.4 Error messages

• ERROR when loading input file “File format is not valid”
• Error while starting simulation (“The specified executable is not a valid application for this OS platform.”)
• No Dirichlet points for Fermi levels found
• Quantum-Current-Poisson fails to converge
17.4.1 ERROR when loading input file “File format is not valid”

A beta feature in a recent Windows 10 version (1803) causes the following error message when starting nextnanomat:

You can disable this feature in the Region’s settings of Windows (screenshot see Figure 17.4.1.2).

1. Select Additional date, time, & regional setting
2. Then Change date, time, or number formats
3. Go the the Administrative Tab
4. Click Change system locale...
5. And finally you can un-select that Beta-Feature.

17.4.2 Error while starting simulation (“The specified executable is not a valid application for this OS platform.”)

If you get this error, you are executing an executable that is not suited for this operating system. There are two possible reasons:

- You are executing a 64-bit version on a 32-bit operating system. Solution: Specify the location of the 32-bit nextnano++ or nextnano3 executables. Tools == Options... == Simulation == nextnano executable file == 32bit.exe
- You are executing the Intel or Microsoft 64-bit executable that has not been compiled for your rather old CPU. Our 64-bit executables use more advanced CPU instructions compared to our 32-bit executables. Solution: Use the 32-bit nextnano++ executable that has been compiled with the Microsoft compiler. Tools ==
Figure 17.4.1.2: How to change Windows 10 region settings.
Options... ==> Simulation ==> nextnano++ executable file ==> nextnano++_Microsoft_32bit.exe

Sometimes the displayed error message does not contain any content apart from ERROR.

You can share information about your CPU with us using the menu Help ==> Generate System Snapshot for Support and then send us the generated file nextnano_SystemSnapshot.txt.

17.4.3 No Dirichlet points for Fermi levels found

**Question** I get the error message:

```
ERROR: No dirichlet points for Fermi levels were found!
Terminating program !
```

**Comment:** Meanwhile, this error message is different. Once an update has been released, change the FAQ here.

**Answer** The program does not know at which energy the Fermi level should be located. A solution is this:

```plaintext
currents{
    region{
        everywhere{}
        binary { name = "GaAs" }
        contact{ name = "dummy" }
    }

    contacts{
        fermi{ name = "dummy" bias = 0.0 }
        # only needed to define reference energy
    }
}
```

This defines the Fermi level to be at 0 [eV] in the whole device.

17.4.4 Quantum-Current-Poisson fails to converge

**Question** My Quantum-Current-Poisson calculation fails to converge. How can I avoid this problem?

**Answer** The most common errors are the following:

a) The maximum number of iterations has been chosen too small (run{} ==> outer_iteration{} ==> iterations) and the calculation just needs some more iterations.

b) The number of electron or hole eigenstates has been chosen too small (e.g. quantum{} ==> region{} ==> Gamma{} ==> num_ev). Check the occupations in the files wf_spectrum"name quantum region"_"name quantum model".dat. If the occupation does not drop from the ground state to the highest excited state by several orders of magnitude, you need to calculate more states (= more possibilities to fill in charges).

c) The underrelaxation parameter has been chosen too large (run{} ==> outer_iteration{} ==> alpha_current). Check whether the solution oscillates (residuals do not drop continuously but increase in some iterations). Try to decrease the underrelaxation parameter in order to damp the oscillations.

d) The minimum charge density in the current equation has been chosen too small (currents{} ==> minimum_density). Try to increase the minimum charge density to improve conditioning of the current equation. In fact, the current equation is solved with the charge density = max( real density, minimum_density).
REFERENCES

• [AndlauerPhD2009]
  Optoelectronic and spin-related properties of semiconductor nanostructures in magnetic fields
  T. Andlauer
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann,
  and P. Vogl, eds.), Vol. 105, Verein zur Förderung des Walter Schottky Instituts der Technischen
  Universität München e.V., München, 157 pp. (2009)

• [BastardPRB1982]
  Exciton binding energy in quantum wells
  G. Bastard, E. E. Mendez, L. L. Chang, and L. Esaki

• [Beynon1988]
  A Macintosh Hypercard stack for calculation of thermodynamically-corrected buffer recipes
  R. J. Beynon

• [Beynon1996]
  Buffer solutions: The basics
  R. J. Beynon, and J. S. Easterby
  Oxford University Press (1996)

• [BirnerPhotonikInt2008]
  Simulation of quantum cascade lasers - optimizing laser performance
  S. Birner, T. Kubis, and P. Vogl
  Photonik international 2, 60 (2008)

• [BirnerPhotonik2008]
  Simulation zur Optimierung von Quantenkaskadenlasern
  S. Birner, T. Kubis, and P. Vogl
  Photonik 1, 44 (2008)

• [BirnerCBR2009]
  Ballistic Quantum Transport using the Contact Block Reduction (CBR) Method - An introduction
  S. Birner, C. Schindler, P. Greck, M. Sabathil, and P. Vogl
  DOI 10.1007/s10825-009-0293-z

• [BirnerPhD2011]
  Modeling of semiconductor nanostructures and semiconductor–electrolyte interfaces
  S. Birner
  Selected Topics of Semiconductor Physics and Technology (G. Abstreiter, M.-C. Amann, M. Stutzmann,
  and P. Vogl, eds.), Vol. 135, Verein zur Förderung des Walter Schottky Instituts der Technischen
  Universität München e.V., München, 239 pp. (2011)
ISBN 978-3-941650-35-0

- [CapassoIEEE1986]
  Resonant Tunneling Through Double Barriers, Perpendicular Quantum Transport Phenomena in Superlattices, and Their Device Applications
  F. Capasso, K. Mohammed, and A. Y. Cho
  DOI 10.1109/JQE.1986.1073171

- [ChuangOpto1995]
  Physics of Optoelectronic Devices
  S. L. Chuang

- [ChuangIEEE1996]
  Optical gain of strained wurtzite GaN quantum-well lasers
  S. L. Chuang
  doi: 10.1109/3.538786

- [DattaETMS1995]
  Electronic Transport in Mesoscopic Systems
  S. Datta
  DOI 10.1017/CBO9780511805776

- [DarwishIEEE1997]
  An Improved Electron and Hole Mobility Model for General Purpose Device Simulation
  M. N. Darwish, J. L. Lentz, M. R. Pinto, P. M. Zeitzoff, T. J. Krutsick, and H. H. Vuong

- [Duboz2019]
  Theoretical estimation of tunnel currents in hetero-junctions: The special case of nitride tunnel junctions
  J-Y. Duboz and B. Vinter

- [Eissfeller2008]
  Linear Optical Response of Semiconductor Nanodevices
  T. Eißfeller

- [FaistQCL2013]
  Quantum Cascade Lasers
  J. Faist
  Oxford University Press (2013)

- [FathololoumiOE2012]
  Terahertz quantum cascade lasers operating up to ~200 K with optimized oscillator strength and improved injection tunneling
  Optics Express 20, 3866 (2012)
  DOI 10.1364/OE.20.003866

- [HackenbuehnerPhD2002]
  Elektronische Struktur von Halbleiter-Nanobauelementen im thermodynamischen Nichtgleichgewicht
  S. Hackenbuehner

- [GreckPhD2012]
  Efficient calculation of dissipative quantum transport properties in semiconductor nanostructures
  P. Greck
  ISBN 978-3-941650-53-4

- [GreckOE2015]
  Efficient Method for the Calculation of Dissipative Quantum Transport in Quantum Cascade Lasers
  P. Greck, S. Birner, B. Huber, and P. Vogl
  Optics Express 23, 6587–6600 (2015)
  DOI 10.1364/OE.23.006587

- [GreckIWCE2010]
  The nonequilibrium Green’s functions method and descendants: ways to avoid and to go
  P. Greck, C. Schindler, T. Kubis, and P. Vogl
  Abstracts of 14th International Workshop on Computational Electronics (IWCE), Pisa, Italy, 145 (2010)
  DOI 10.1109/IWCE.2010.5677996

- [HarrisonQWWD2005]
  Quantum Wells, Wires and Dots
  P. Harrison
  DOI 10.1017/CBO9780511805776

- [JancuPRB1998]
  Empirical spd* tight-binding calculation for cubic semiconductors: General method and material parameters
  J.-M. Jancu, R. Scholz, F. Beltram, F. Bassani

- [Klaassen1992]
  An Improved Electron and Hole Mobility Model for General Purpose Device Simulation - I. Model equations and concentration dependence
  D. B. M. Klaassen
  Solid-State Electronics 35, 953 (1992)

- [KubisPhD2009]
  Quantum transport in semiconductor nanostructures
  T. C. Kubis

- [KubisNEGF2005]
  Self-consistent quantum transport theory of carrier capture in heterostructures
  T. Kubis, A. Trellakis, and P. Vogl
  DOI 10.1007/978-3-540-36588-4_84
• [LeverJLT2010]
  Design of Ge–SiGe Quantum-Confined Stark Effect Electroabsorption Heterostructures for CMOS Compatible Photonics
  L. Lever, Z. Ikonic, A. Valavanis, J. D. Cooper, and R. W. Kelsall

• [LombardiIEEE1988]
  A physically based mobility model for numerical simulation of nonplanar devices
  C. Lombardi, S. Manzini, A. Saporito, and M. Vanzi

• [MamaluyCBR2003]
  Efficient method for the calculation of ballistic quantum transport
  D. Mamaluy, M. Sabathil, and P. Vogl
  DOI 10.1063/1.1560567

• [SabathilPhD2005]
  Opto-electronic and quantum transport properties of semiconductor nanostructures
  M. Sabathil

• [Sarma2002]
  Realistic tight-binding model for the electronic structure of II-VI semiconductors
  S. Sapra, N. Shanthi, D. D. Sarma

• [SirtoriPRB1994]
  Nonparabolicity and a sum rule associated with bound-to-bound and bound-to-continuum intersubband transitions in quantum wells
  C. Sirtori, F. Capasso, J. Faist, and S. Scandolo
  DOI 10.1103/PhysRevB.50.8663

• [VurgaftmanJAP2001]
  Band parameters for III-V compound semiconductors and their alloys
  I. Vurgaftman and J. R. Meyer
  Journal of Applied Physics 89, 5815 (2001)

• [VurgaftmanJAP2003]
  Band parameters for nitrogen-containing semiconductors
  I. Vurgaftman and J. R. Meyer

• [ZiboldPhD2007]
  Semiconductor based quantum information devices: Theory and simulations
  T. Zibold

• [VoglJPCS1983]
  A Semi-Empricial Tight-Binding Theory of the Electronic Structure of Semiconductors
  P. Vogl, H. P. Hjalmarsen, and J. D. Dow
  doi: 10.1016/0022-3697(83)90064-1
• [CarloSST2003]
  Microscopic theory of nanostructured semiconductor devices: beyond the envelope=function approximation
  A. D. Carlo
  Semiconductor Science and Technology, 18(1), R1-31
  doi: 10.1088/0268-1242/18/1/201

• [KlimeckSM2000]
  sp3s* Tight-binding parameters for transport simulations in compound semiconductors
  G. Klimeck, R. C. Bowen, T. B. Boykin, T. A. Cwik
  Superlattices and Microstructures, Vol. 27, No. 5, pp. 519-524

• [SaitoS2001]
  Optical Properties and Raman Spectroscopy of Carbon Nanotubes
  M.S. Dresselhaus, G. Dresselhaus, P. Avouris (Eds.)
  Topics in Applied Physics, Vol. 80, Springer (2001)
  doi: 10.1007/3-540-39947-X_9

• [ReichPR2002]
  Tight-binding description of graphene
  S. Reich, J. Maultzsch, C. Thomsen, P. Ordejon
  doi: 10.1103/PhysRevB.66.035412

• [PiprekE2003]
  Semiconductor Optoelectronic Devices: Introduction to Physics and Simulation
  J. Piprek

• [GreinAIP1995]
  Long wavelength InAs/InGaSb infrared detectors: Optimization of carrier lifetimes
  C. H. Grein, P. M. Young

• [ZakharovaPR2001]
  Hybridization of electron, light-hole, and heavy-hole states in InAs/GaSb quantum wells
  A. Zakharova, S. T. Yen, K. A. Chao
  Physical Review B 64, 235332 (2001)

• [HalvorsenPR2000]
  Optical transitions in broken gap heterostructures
  E. Halvorsen, Y. Galperin, K. A. Chao