STEMsalabim Documentation

Release 5.0.0

Jan Oliver Oelerich

Feb 28, 2019

Getting Started

1 What STEMsalabim calculates **1**
- 1.1 Background . . . 1
- 1.2 Probe wave function . . . 1
- 1.3 Multislice Simulation . . . 2
- 1.4 Detector . . . 2

2 Installing STEMsalabim **3**
- 2.1 Requirements . . . 3
- 2.2 Downloading the source code . . . 4
- 2.3 Building STEMsalabim . . . 4

3 Running STEMsalabim **7**
- 3.1 Parallel runs . . . 7
- 3.2 `Si 001` example . . . 8
- 3.3 ssb-mkin and ssb-run . . . 8
- 3.4 NetCDF files . . . 9

4 Visualization of crystals and results **11**
- 4.1 Visualize the supercell with Ovito . . . 11
- 4.2 Generate an ADF STEM image . . . 11
- 4.3 What now? . . . 12

5 General information **13**
- 5.1 Structure of a simulation . . . 13
- 5.2 Hybrid Parallelization model . . . 14

6 Simulation Parameters **15**
- 6.1 Parameter files . . . 15
- 6.2 Command line arguments . . . 21

7 File formats **23**
- 7.1 Crystal file format . . . 23
- 7.2 Output file format . . . 24
- 7.3 Reading NC Files . . . 29

8 Frequently Asked Questions **31**
- 8.1 What about the name STEMsalabim? . . . 31

9 What's new **33**
- 9.1 STEMsalabim 5.0.0 . . . 33
- 9.2 STEMsalabim 4.0.1, 4.0.2 . . . 34
- 9.3 STEMsalabim 4.0 . . . 34

9.4 STEMsalabim 3.1.0, 3.1.1, 3.1.2, 3.1.3, 3.1.4 . . . 34
9.5 STEMsalabim 3.0.1 and 3.0.2 . . . 35
9.6 STEMsalabim 3.0.0 . . . 35
9.7 STEMsalabim 2.0.0 . . . 35
9.8 STEMsalabim 2.0.0-beta2 . . . 35
9.9 STEMsalabim 2.0.0-beta . . . 36
9.10 STEMsalabim 1.0 . . . 36

10 Citing STEMsalabim **37**

11 Research done with STEMsalabim **39**
11.1 2018 . . . 39
11.2 2017 . . . 39

CHAPTER 1

What STEMsalabim calculates

1.1 Background

STEMsalabim simulates the image of a transmission electron microscope (TEM) in scanning mode, i.e., with a convergent electron beam scanning over the sample. (TEM can also be operated using plane wave illumination, but this is (currently) not supported by STEMsalabim.

In scanning TEM (STEM) a narrowly focused beam of high energy electrons is focused onto a very thin layer of some material. Passing through the sample, the incident electrons are scattered by the sample's coulomb potential, which is composed of the atomic cores and the sample's electrons. After leaving the sample the amount and direction of scattering is detected to gain insight into the sample's atomic structure, chemical composition, electric fields, thickness, etc.

Instead of repeating the details of STEM here, we refer the reader to some good literature on the topic:

- Transmission Electron Microscopy by David B. Williams and Barry C. Carter
- Advanced computing in electron microscopy by Earl J. Kirkland

For the details on STEMsalabim's algorithm and implementation, the following paper is instructive:

- STEMsalabim: A high-performance computing cluster friendly code for scanning transmission electron microscopy image simulations of thin specimens

1.2 Probe wave function

The electron probe in the STEM must be modelled as a coherent electronic wave, since diffraction plays an essential role in STEM. In STEM simulations, the focused probe wave function $\Psi_0(x, y)$ is modelled as a complex 2D object, and its propagation in beam direction (here, z) is simulated.

The probe wave function encapsulates all characteristics of the microscope, such as aberrations, aperture, etc. It is calculated via

$$\Psi_0^{x_p,y_p}(x,y) = B\mathcal{F}^{-1}\left(A(k_x,k_y)\exp\left[-i\chi(k_x,k_y)+2\pi i(k_x x_p + k_y y_p)\right]\right) ,$$

with a normalization constant B, the aperture function $A(k_x, k_y)$, and the aberration phase error $\chi(k_x, k_y)$:

$$\chi(k_x,k_y) = \frac{\pi\alpha^2}{\lambda}\left(-\Delta f + C_a\cos(2\phi - 2\phi_a) + \frac{1}{2}C_s\alpha^2 + \frac{1}{3}C_5\alpha^4\right) .$$

The aberration coefficients $C_a, C_s, C_5, \Delta f, \phi_a$ and the aperture radius entering $A(k_x, k_y)$ are parameters of the microscope and can be freely defined in STEMsalabim. (See *Parameter files*)

1.3 Multislice Simulation

While passing through the thin sample, the wave function is scattered by the sample's coulomb potential. In STEMsalabim, the multi-slice algorithm is implemented to model the interaction of the incoming probe wave function with the sample's coulomb potential.

In the multi-slice approximation, the sample is divided into thin slices perpendicular to the beam direction. When the electrons of the probe have much higher energy than those of the specimen, and the slices are thin enough, the interaction between an incoming wave function and one of these slices can be described via

$$\Psi_f(x, y) = t(x, y)\Psi_i(x, y)$$

where $\Psi_i(x, y)$ is the incoming and $\Psi_f(x, y)$ the outgoing WF. The 2D *transmission function* of the slice is $t(x, y)$ depends on the *projected coulomb potential* $v(x, y)$ of the slice:

$$t(x, y) = \exp(-i\sigma v(x, y)$$

with σ being some relativistic interaction parameter. Clearly, the transmission function $t(x, y)$ merely modifies the phase of the wave function and is therefore sometimes called a *weak phase object*.

To model the interaction of the probe WF $\Psi_0(x, y)$ with every slice, it is subsequently multiplied with the transmission function of each single slice. In between the slices, the WF is propagated to the next slice by multiplication

with a Fresnel propagator

$$p(k_x, k_y) = \exp(i\pi(k_x^2 + k_y^2)\lambda dz)$$

with wavelength λ and slice thickness dz. Since the propagator is defined in k-space, an iteration of the multi-slice approximation is calculated as

$$\Psi_{n+1}(x, y) = F^{-1}\left[p(k_x, k_y)F\left[t(x, y)\Psi_n(x, y)\right]\right]$$

where F and F^{-1} are forward and backward 2D Fourier transformations.

1.4 Detector

After the multi-slice simulation, the absolute intensity of the diffracted WF (in k-space) $|\Psi(k_x, k_y)|^2$ is detected. STEMsalabim supports writing the 2D diffractogram directly to the output file, but it can also calculate and write the angular dependency $|\Psi(|k|)|^2$. (See *Parameter files* and *Output file format* for more information.)

CHAPTER 2

Installing STEMsalabim

2.1 Requirements

The following libraries and tools are needed to successfully compile the code:

- A C++11 compiler (such as gcc/g++ or intel compiler suite).
- CMake > 3.3
- NetCDF
- libConfig >= 1.5
- FFTW3 or Intel's MKL
- An MPI implementation (such as OpenMPI)

Note: You may find some of the requirements in the repositories of your Linux distribution, at least the compiler, CMake, and OpenMPI. On Debian or Ubuntu Linux, for example, you can simply run the following command to download and install all the requirements:

```
$ apt-get install build-essential      \
                  cmake                \
                  libconfig++-dev      \
                  libfftw3-dev         \
                  libnetcdf-dev        \
                  libopenmpi-dev       \
                  openmpi-bin
```

Tip: Most of the computing time is spent calculating Fourier transforms, so it is beneficial for STEMsalabim to use optimized FFT libraries. Sometimes, compiling FFTW or MKL on the target machine enables optimizations that are not available in precompiled binaries, so this may be worth a try.

2.2 Downloading the source code

We recommend you download the latest stable release (5.0.0) from the Releases page. If you want the latest features and/or bugfixes, you can also clone the repository using

```
$ git clone https://gitlab.com/STRL/STEMsalabim.git
$ git checkout devel # only if you want the devel code.
```

2.3 Building STEMsalabim

Extract the code archive to some folder on your hard drive, e.g.

```
$ cd /tmp
$ tar xzf stemsalabim-VERSION.tar.gz
```

Then, create a build directory and run CMake to generate the build configuration:

```
$ mkdir /tmp/stemsalabim-build
$ cd /tmp/stemsalabim-build
$ cmake ../stemsalabim-VERSION
```

Please refer to the CMake documentation for instructions how to specify library paths and other environment variables, in case the above commands failed. When your libraries exist at non-standard places in your file system, you can specify the search paths as follows:

```
$ cmake ../stemsalabim-VERSION                                          \
    -DFFTW_ROOT=/my/custom/fftw/                                        \
    -DLIBCONFIG_ROOT=/my/custom/libconfig/                              \
    -DNETCDF_INCLUDE_DIR=/my/custom/netcdf/include                      \
    -DCMAKE_INSTALL_PREFIX=/usr/local                                   \
    -DCMAKE_EXE_LINKER_FLAGS='-Wl,-rpath,/my/custom/lib64:/my/custom/lib' \
    -DCMAKE_CXX_COMPILER=/usr/bin/g++
```

In the above example, some custom library paths are specified and the program's run-time search path is modified. If cmake doesn't detect the correct compiler automatically, you can specify it with `-DCMAKE_CXX_COMPILER=`.

Having generated the necessary build files with CMake, simply compile the program using `make` and move it to the install location with `make install`:

```
$ make -j8          # use 8 cores for compilation
$ make install      # move the binaries and libraries to the INSTALL_PREFIX path
```

You are now ready to execute your first simulation.

2.3.1 Building with Intel MKL, Intel compiler (and Intel MPI)

It is possible to use the Intel® Parallel Studio for compilation, which includes the Intel® Math Kernel Library (MKL) that STEMsalabim can use for discrete fourier transforms instead of FFTW3. If the Intel® MPI Library is also available, it can be used for MPI communication.

Note: We have tested compiling and running STEMsalabim only with Parallel Studio 2017 so far.

STEMsalabim's CMake files try to find the necessary libraries themselves, when the folling conditions are true:

1. Either the environment variable `MKLROOT` is set to a valid install location of the MKL, or the CMake variable `MKL_ROOT` (pointing at the same location) is specified.

2. The CMake variable `GCCDIR` points to the install directory of a C++11 compatible GCC compiler. This is important, because the `libstdc++` from a GCC install is required for the Intel compilers to use modern C++ features.

For example, let's say the Intel suite is installed in `/opt/intel` and we have GCC 6.3 installed in `/opt/gcc-6.3`, then CMake could be invoked like this:

```
$ export PATH=$PATH:/opt/intel/... # mpicxx and icpc should be in the path!
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/gcc-6.3/lib64 \
     cmake ../source                                  \
        -DMKL_ROOT=/opt/intel                         \
        -DCMAKE_CXX_COMPILER=icpc                     \
        -DGCCDIR=/opt/gcc-6.3                         \
        -D... more CMAKE arguments as described above.
```

Depending on how your environment variables are set, you may be able to skip the `LD_LIBRARY_PATH=..` part. When STEMsalabim is executed, you may again need to specify the library path of the `libstdc++`, using

```
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/gcc-6.3/lib64 mpirun -np ... /path/to/
↪stemsalabim -p ...
```

Note: Some fiddling with paths and environment variables is probably necessary. It may help to know basic CMake syntax and have a look at `/path/to/stemsalabim/cmake/FindMKL.cmake` if CMake is unable to find something.

CHAPTER 3

Running STEMsalabim

STEMsalabim is executed on the command line and configured via input configuration files in libConfig syntax. To learn about the structure of the configuration files, please read *Parameter files*.

Note: Some of configuration parameters can be changed via command line parameters, which are described in *Command line arguments*.

STEMsalabim supports both threaded (shared memory) and MPI (distributed memory) parallelization. For most efficient resource usage we recommend a hybrid approach, where one MPI task is run per node that spawns a bunch of threads to parallelize the work within the node. (See *Hybrid Parallelization model* for more information on how STEMsalabim can be parallelized.)

3.1 Parallel runs

3.1.1 Thread-only parallelization

You can execute STEMsalabim on a single multi-core computer as follows:

```
$ stemsalabim --params=./my_config_file.cfg --num-threads=32
```

This will run the simulation configured in `my_config_file.cfg` on 32 cores, of which 31 are used as workers.

3.1.2 MPI only parallelization

For pure MPI parallelization without spawning additional threads, STEMsalabim must be called via `mpirun` or `mpiexec`, depending on the MPI implementation available on your machine:

```
$ mpirun -n 32 stemsalabim --params=./my_config_file.cfg --num-threads=1 --package-
↪size=10
```

This command will run the simulation in parallel on 32 MPI processors without spawning additional threads.

Note: We chose a *work package size* ten times the number of threads on each MPI processor (which is 1 here). This is so that each thread calculates (on average) ten pixels until results are communicated via the network. This reduces management overhead but increases the amount of data sent via the network.

3.1.3 Hybrid parallelization

Hybrid parallelization is the recommended mode to run STEMsalabim.

For hybrid parallelization, make sure that on each node only a single MPI process is spawned and that there is no CPU pinning active, i.e., STEMsalabim needs to be able to spawn threads on different cores.

For example, if we wanted to run a simulation in parallel on 32 machines using OpenMPI and on each machine use 16 cores, we would run

```
$ mpirun -n 32 --bind-to none --map-by ppr:1:node:pe=16 \
    stemsalabim                                         \
        --params=./my_config_file.cfg                   \
        --num-threads=16                                \
        --package-size=160
```

The options `--bind-to none --map-by ppr:1:node:pe=16` tell OpenMPI not to bind the process to anything and to reserve 16 threads for each instance. Please refer to the manual of your MPI implementation to figure out how start a hybrid parallelization run. On computing clusters, node and/or socket topology may affect performance, so it is wise to consult your cluster admin team.

3.2 `Si 001` example

In the source code archive you find an `examples/Si_001` folder that contains a simple example that you can execute to get started. The file `Si_001.xyz` describes a 2x2x36 unit cell Si sample. Please see *Crystal file format* for the format description.

In the file `Si_001.cfg` we find the simulation configuration / parameters. The file contains all available parameters, regardless of whether they are set to their default value. We recommend to always specify a complete set of simulation parameters in the configuration files.

You can now run the simulation:

```
$ /path/to/stemsalabim --params Si_001.cfg --num-threads=8
```

After the simulation finished (about 3 hours on an Intel i7 CPU with 8 cores) you can analyze the results found in `Si_001.nc`. Please see the next page (*Visualization of crystals and results*) for details.

3.3 ssb-mkin and ssb-run

Along with the main `stemsalabim` binary, the `ssb-mkin` and `ssb-run` tools are also compiled and put into your `bin/` directory.

`ssb-run` can be used to start a STEMsalabim simulation from an existing NetCDF file. Results in the file are discarded and all required parameters are read from the file. Most importantly, the generated atomic displacements for all the frozen phonon configurations are read from the file, so that starting from an NetCDF file `ssb-run` should always produce the *exact same results*.

`ssb-mkin` is the complementary tool to `ssb-run`, generating an *input* NetCDF file from a parameter file (see *Parameter files*) and a crystal file (see *Crystal file format*). The output of `ssb-mkin` is identical to the output of `stemsalabim`, except that it doesn't contain any results.

```
$ /path/to/ssb-mkin --params Si_001.cfg --output-file Si_001.nc
$ /path/to/ssb-run --params Si_001.nc --num-threads=8
```

The above two commands are identical to the example in *Si 001 example*.

The intermediate `Si_001.nc` file is small (as it contains no results) and contains everything required to start a simulation. It is therefore well-suited for backing up or sending around. In addition, the `--stored-potentials` parameter can be used.

3.3.1 –stored-potentials

Both `ssb-mkin` and `ssb-run` accept the `--stored-potentials` command line parameter. When it is specified, the scattering Coulomb potentials are calculated already while running `ssb-mkin` and written to the NetCDF file in `AMBER/slice_potentials`. `ssb-run` then reads the potentials from the file and starts the multi-slice simulation without recalculating the potentials.

This is useful for inspecting and modifying potentials prior to the simulation.

3.4 NetCDF files

STEMsalabim writes its results and a bunch of information about the simulation to in NetCDF binary format. NetCDF is a hierarchical storage format for storing multi-dimensional data. It is (most of the times) based on HDF5.

Please see *Output file format* for more information how to read / write NetCDF files.

CHAPTER 4

Visualization of crystals and results

Now that our simulation finished successfully, we can continue with visualizing the results.

4.1 Visualize the supercell with Ovito

The STEMsalabim output files (somewhat) comply with the AMBER specifications to visualize the specimen structure. However, all the dimensions and variables of AMBER live within the NetCDF group `/AMBER` in the NC file. This means that STEMsalabim output files **will not be compatible to visualization programs requiring the pure AMBER specs!**

However, the authors of the excellent cross-platform Ovito software, which is a visualization program for atomic structures (and much more), have added support for the `/AMBER` sub-group, so that STEMsalabim NetCDF result files can be visualized seamlessly in Ovito.

What you will see is the atomic structure of the input specimen. In addition to the positional coordinates of each atom, you find the mean square displacement (`msd`), the slice ID and coordinate (`slice` and `slice_coordinates`), the equilibrium coordinates (`lattice_coordinates`), elements (`elements`) and atomic radii (`radii`) as variables. Each frozen lattice configuration is one `frame` in the AMBER specs, so you can see the atoms wiggling around if you use the frame slider of Ovito.

4.2 Generate an ADF STEM image

In the `examples/Si_001` folder you will find the two files `make_haadf.m` and `make_haadf.py`. Both extract an HAADF image from the result NetCDF file. Please have a look at the code to get an idea of how to work with the NetCDF result files.

- MATLAB® uses HDF5 for its `.mat` format for a couple of versions now, and is therefore perfectly capable of reading STEMsalabim result files. For quick analysis and image generation is a great tool.
- Python with the NetCDF4 module is also a great tool to analyze and visualize STEMsalabim result files, especially combined with numerical libraries such as numpy or pandas.

4.3 What now?

You have now completed your first simulation and looked at some of its results. In order to use STEMsalabim for your research you should dig deeper into this documentation by reading the documentation links on the left.

CHAPTER 5

General information

Warning: This documentation is not really complete (yet).

Throughout this documentation we assume that you are familiar with the theoretical background behind the scanning transmission electron microscope (STEM) to some degree. Also, we assume that you have some knowledge about the UNIX/Linux command line and parallelized computation. STEMsalabim is currently not intended to be run on a desktop computer. While that is possible and works, the main purpose of the program is to be used in a highly parallelized multi-computer environment.

We took great care of making STEMsalabim easy to install. You can find instructions at *Installing STEMsalabim*. However, if you run into technical problems you should seek help from an administrator of your computer cluster first.

5.1 Structure of a simulation

There essence of STEMsalabim is to model the interaction of a focused electron beam with a bunch of atoms, typically in the form of a crystalline sample. Given the necessary input files, the simulation crunches numbers for some time, after which all of the calculated results can be found in the output file. Please refer to *Running STEMsalabim* for notes how to start a simulation.

5.1.1 Input files

All information about the specimen are listed in the *Crystal file format*, which is one of the two required input files for STEMsalabim. It contains each atom's species (element), coordinates, and mean square displacement as it appears in the Debye-Waller factors.

In addition, you need to supply a *Parameter files* for each simulation, containing information about the microscope, detector, and all required simulation parameters. All these parameters are given in a specific syntax in the *Parameter files* that are always required for starting a STEMsalabim* simulation.

5.1.2 Output files

The complete output of a STEMsalabim simulation is written to a NetCDF file. NetCDF is a binary, hierarchical file format for scientific data, based on HDF5. NetCDF/HDF5 allow us to compress the output data and store it in machine-readable, organized format while still only having to deal with a single output file.

You can read more about the output file structure at *Output file format*.

5.2 Hybrid Parallelization model

STEMsalabim simulations is parallelized both via POSIX threads and via message passing interface (MPI). A typical simulation will use both schemes at the same time: MPI is used for communication between the computing nodes, and threads are used for intra-node parallelization, the usual multi-cpu/multi-core structure.

Hint: A high performance computation cluster is an array of many (equal) computing *nodes*. Typical highly-parallelized software uses more than one of the nodes for parallel computations. There is usually no memory that is shared between the nodes, so all information required for the management of parallel computing needs to be explicitely communicated between the processes on the different machines. The quasi-standard for that is the message passing interface (MPI).

Let us assume a simulation that runs on M computers and each of them spawns N threads.

Depending on the simulation parameters chosen, STEMsalabim may need to loop through multiple frozen phonon configurations and values of the probe defocus. The same simulation (with differently displaced atoms and different probe defocus) is therefore typically run multiple times. There are three parallelization schemes implemented in STEMsalabim:

- When $M == 1$, i.e., no MPI parallelization is used, all pixels (probe positions) are distributed among the N threads and calculated in parallel.
- Each MPI processor calculates *all* pixels (probe positions) of its own frozen phonon / defocus configuration, i.e., M configurations are calculated in parallel. Each of the M calculations splits its pixels between N threads (each thread calculates one pixel at a time).

  This scheme makes sense when the total number of configurations (*probe.num_defoci* $\times$ *frozen_phonon.number_configurations*) is much larger than or divisible by M.
- A single configuration is calculated at a time, and all the pixels are split between all $M \times N$ threads. In order to reduce the required MPI communication around, only the main thread of each of the M MPI processors communicates with the master thread. The master thread sends a *work package* containing some number of probe pixels to be calculated to an MPI process, which then carries out all the calculations in parallel on its N threads. When a work package is finished, it requests another work package from the master MPI process until there is no work left. In parallel, the worker threads of the MPI process with rank 0 also work on emptying the work queue.

In MPI mode, each MPI process writes results to its own temporary file, and after each frozen lattice configuration the results are merged. Merging is carried out sequentially by each individual MPI processor, to avoid race conditions. The parameter `output.tmp_dir` (see *Parameter files*) should be set to a directory that is local to each MPI processor (e.g., `/tmp`).

Note: Within one MPI processor, the threads can share their memory. As the main memory consumption comes from storing the weak phase objects of the slices in the multi-slice simulation, which don't change during the actual simulation, this greatly reduces memory usage as compared to MPI only parallelization. You should therefore always aim for hybrid parallelization!

CHAPTER 6

Simulation Parameters

A STEMsalabim simulation is mainly configured via *parameter files*, with a few exceptions where configuration options may be overriden by *command line arguments*.

6.1 Parameter files

The configuration file that STEMsalabim expects for the command line parameter `--params` is formatted using the simple JSON-like syntax of libConfig syntax. Below all available parameters are tabulated. Each section (`block` in the libConfig syntax) is described separately below.

6.1.1 `application`

The `application` block contains general settings regarding the simulation.

```
application: {
    random_seed = 0;              # the random seed. 0 -> generate
}
```

application.random_seed *unsigned int [default: '0']*

> In the frozen lattice approximation, the atoms are randomly dislocated from their equilibrium position. The random seed for that can be specified here. If set to 0, a random seed is generated by the program. For reproduction of previous results, the seed can be set to a specific value.

6.1.2 `simulation`

Some general settings specific to this simulation.

```
simulation: {
    title = "benchmarksmall";     # title of the simulation
    bandwidth_limiting = true;    # bandwidth limit the wave functions?
    normalize_always = false;     # normalize wave functions after each slice?
    output_file = "out.nc";       # output file name
    tmp_dir = "/tmp/";            # directory for temporary files.
```

(continues on next page)

(continued from previous page)

```
    output_compress = "false";    # compress output data. This reduces file sizes,␣
↪but increases IO times.
}
```

simulation.title *string [default: ‘’]*

The title of the simulation, as saved in the output NC file.

simulation.bandwidth_limiting *boolean [default: ‘true‘]*

Enforce cylindrical symmetry on all wave functions/operations. This results in all wave functions to be clipped significantly in momentum space. However, the benefit of reduced artefacts due to periodic boundary conditions is usually more important.

simulation.normalize_always *boolean [default: ‘false‘]*

Renormalize the electronic wave function after each slice of the multi-slice approximation, so that lost intensitiy due to limited k space grid is compensated for. Usually, this is not necessary, so the default is `false`.

simulation.output_file *string [default: ‘’]*

Path to the result NetCDF file. Please read *Output file format* to learn about its format. Relative paths are interpreted relative to the working directory of the simulation.

simulation.tmp_dir *string [default: ‘folder of ouput file‘]*

When MPI parallelized, all MPI processors will write their results to temporary binary files in this directory, and only at the end of each frozen lattice configuration, the results are merged. We recommend using a local directory here (local to each MPI processor) so that access is fast. By default, the directory of the output file is used, which typically is **not local** but on a network file system. This can have a profound effect when CBED results are collected, i.e., output becomes large.

simulation.output_compress *boolean [default: ‘false‘]*

The written output data can be compressed by HDF5. This obviously takes more I/O time, but reduces output file sizes.

6.1.3 probe

Parameters of the STEM probe.

```
probe: {
    c5 = 5e6;                    # Fifth order spherical aberrations coefficient.␣
↪in nm
    cs = 2e3;                    # Third order spherical aberrations coefficient.␣
↪in nm
    defocus = -2.0;              # defocus value in nm
    fwhm_defoci = 6.0;           # FWHM of the defocus distribution when␣
↪simulating a defocus series for Cc.
    num_defoci = 1;              # number of the defoci when simulating a defocus␣
↪series for Cc. Should be odd.
    astigmatism_ca = 0;          # Two-fold astigmatism. in nm
    astigmatism_angle = 0;       # Two-fold astigmatism angle. in mrad
    min_apert = 0.0;             # Minimum numerical aperture of the objective. in␣
↪mrad
    max_apert = 24.0;            # Maximum numerical aperture of the objective. in␣
↪mrad
    beam_energy = 200.0;         # Electron beam energy. in kV
    scan_density = 40;           # The sampling density for the electron probe␣
↪scanning. in 1/nm
}
```

probe.c5 *number (nm) [default: '5e7']*

Fifth order spherical aberrations coefficient.

probe.cs *number (nm) [default: '2e4']*

Third order spherical aberrations coefficient.

probe.defocus *number (nm) [default: '2.0']*

Probe defocus.

probe.fwhm_defoci *number (nm) [default: '6.0']*

STEMsalabim can calculate a defocus series to model chromatic aberrations. In that case, this parameter is the full-width-half-maximum of the normal distribution of defocus spread in nm.

probe.num_defoci *number [default: '1']*

STEMsalabim can calculate a defocus series to model chromatic aberrations. In that case, this parameter is the number of defoci calculated.

probe.astigmatism_ca *number (nm) [default: '0']*

Two-fold astigmatism.

probe.astigmatism_angle *number (mrad) [default: '0']*

Two-fold astigmatism angle.

probe.min_apert *number (mrad) [default: '0']*

Minimum numerical aperture of the objective.

probe.max_apert *number (mrad) [default: '24']*

Maximum numerical aperture of the objective.

probe.beam_energy *number (keV) [default: '200']*

Electron beam energy.

probe.scan_density *number (1/nm) [default: '40.0']*

The sampling density for the electron probe scanning. This number multiplied by the supercell size in x and y direction gives the number of positions, i.e., scan points, that are simulated.

6.1.4 specimen

Settings for the specimen and potentials.

```
specimen: {
    max_potential_radius = 0.3;   # potential cut-off radius. in nm
    crystal_file = "input.xyz";   # xyz file with columns [Element, x, y, z, MSD]
}
```

specimen.max_potential_radius *number (nm) [default: '0.3']*

Distance, after which the atomic potentials are cut off during generation of the slices' transmission functions.

specimen.crystal_file *string [default: '']*

Path to the file containing the atomic coordinates. Please see *Crystal file format* to learn about its format. Relative paths are interpreted relative to the working directory of the simulation.

6.1.5 grating

Settings that describe the multi-slice algorithm, i.e., the density of the discretization and the slice thickness.

```
grating: {
    density = 360.0;              # The density for the real space and fourier␣
↪space grids. in 1/nm
    slice_thickness = 0.2715;     # Multi-slice slice thickness. in nm
}
```

grating.density *number (1/nm) [default: '360.0']*

> **The density for the real space and fourier space grids. This number multiplied by the supercell size in** x and y direction gives the minimal number of sampling grid points for the calculation. The actual grid size used for the simulation may be bigger than that, as an efficient size for the fourier transforms is chosen. This also determines the maximum angle $\alpha = k\lambda$ that is described by the k-space grids.

grating.slice_thickness *number (nm) [default: '0.2715']*

> The thickness of the slices in the multi-slice algorithm.

6.1.6 adf

Settings for collection of ADF data.

```
adf: {
    enabled = true;               # enable calculation and collection of ADF␣
↪intensities
    x = (0.0, 1.0);               # [min, max] where min and max are in relative␣
↪units
    y = (0.0, 1.0);               # [min, max] where min and max are in relative␣
↪units
    detector_min_angle = 1.0;     # inner detector angle in mrad
    detector_max_angle = 300.0;   # outer detector angle in mrad
    detector_num_angles = 300;    # number of bins of the ADF detector.
    detector_interval_exponent = 1.0;      # possibility to set non-linear detector␣
↪bins.
    save_slices_every = 0;        # save only every n slices. 0 -> only the sample␣
↪bottom is saved.
    average_configurations = true; # average the frozen phonon configurations in␣
↪the output file
    average_defoci = true;        # average the defoci in the output file
}
```

adf.enabled *boolean [default: 'true']*

> Enable or disable calculation of ADF intensities completely.

adf.x *array [default: '[0.0,1.0]']*

> The relative coordinates, between which the supercell is scanned by the electron probe. When `[0.0, 1.0]`, the whole `x` width of the supercell is scanned.

adf.y *array [default: '[0.0,1.0]']*

> The relative coordinates, between which the supercell is scanned by the electron probe. When `[0.0, 1.0]`, the whole `y` width of the supercell is scanned.

adf.detector_min_angle *number (mrad) [default: '0.0']*

> Inner ADF detector angle in mrad.

adf.detector_max_angle *number (mrad) [default: '300.0']*

> Outer ADF detector angle in mrad.

adf.detector_num_angles *number [default: '301']*

Number of ADF detector angle bins.

adf.detector_interval_exponent *number [default: '1.0']*

A non-linear grid spacing of the detector grid can be chosen to increase the sampling at smaller angles. The `ith` grid point between θ_{min} and θ_{max} is calculated by $\theta_i = \theta_{min} + (i/N)^p(\theta_{max} - \theta_{min})$, where p is the interval exponent and N is the number of grid points as given by **detector.angles**. When set to `1.0`, the grid is linear.

Note: A fundamental lower limit for the grid spacing is given by the **grating.density** density. Finer sampling than the grating density will result in odd artefacts!

---

adf.save_slices_every *integer [default: '1']*

The ADF intensity is calculated and stored after every slice that is a multiple of this parameter. The default value of `1` results in every slice to be saved, higher numbers skip slices. The value `0` corresponds to the intensity only being saved after the last slice, i.e., at the sample bottom.

adf.average_configurations *boolean [default: 'true']*

Whether or not to perform in-place averaging over frozen lattice configurations during writing to the output file.

adf.average_defoci *boolean [default: 'true']*

Whether or not to perform in-place (weighted) averaging over defoci during writing to the output file.

6.1.7 cbed

Settings for collection of CBEDs.

Note: Storing CBEDs will increase the output file size drastically. Moreover, the information contained in the CBEDs may be redundant to the ADF data.

```
cbed: {
    enabled = true;                 # enable calculation and collection of CBED␣
↪intensities
    x = (0.0, 1.0);                 # [min, max] where min and max are in relative␣
↪units
    y = (0.0, 1.0);                 # [min, max] where min and max are in relative␣
↪units
    size = [128, 128];              # When provided, this parameter determines the␣
↪size of CBEDs saved
                                    # to the output file. The CBEDs are resized␣
↪using bilinear interpolation.
    save_slices_every = 0;          # save only every n slices. 0 -> only the␣
↪sample bottom is saved.
    average_configurations = true; # average the frozen phonon configurations in␣
↪the output file
    average_defoci = true;          # average the defoci in the output file
}
```

cbed.enabled *boolean [default: 'false']*

Enable or disable calculation of CBED intensities completely.

cbed.x *array [default: '[0.0,1.0]']*

The relative `x` coordinates, between which CBED images are to be saved. When `[0.0, 1.0]`, the whole `y` width of the supercell is scanned.

cbed.y *array [default: '[0.0,1.0]']*

The relative y coordinates, between which CBED images are to be saved. When `[0.0, 1.0]`, the whole `y` width of the supercell is scanned.

cbed.size *array [default: '[0.0,0.0]']*

Width and height of the CBEDs. Saving CBEDs can become very disk-space intensive, especially when the `k` grids are huge (as required for big samples). When this parameter is given, one can choose the size of the CBEDs that are saved to the output file. The images are reduced using bilinear interpolation. The total intensity is preserved. Leave the parameter out or set any direction to 0 to use the original size.

cbed.save_slices_every *integer [default: '1']*

The STEM intensity is calculated and stored after every slice that is a multiple of this parameter. The default value of `1` results in every slice to be saved, higher numbers skip slices. The value `0` corresponds to the intensity only being saved after the last slice, i.e., at the sample bottom.

cbed.average_configurations *boolean [default: 'true']*

Whether or not to perform in-place averaging over frozen lattice configurations during writing to the output file.

cbed.average_defoci *boolean [default: 'true']*

Whether or not to perform in-place (weighted) averaging over defoci during writing to the output file.

6.1.8 `frozen_phonon`

Settings for the `frozen_phonon` algorithm to simulate TDS.

```
frozen_phonon: {
    enabled = true;              # enable or disable the frozen phonon feature
    number_configurations = 15;  # Number of frozen phonon configurations to␣
↪calculate
    fixed_slicing = true;        # When this is true, the z coordinate is not␣
↪varied during phonon vibrations.
}
```

frozen_phonon.enabled *boolean [default: 'true']*

Whether diffuse thermal scattering via frozen phonon approximation should be enabled.

frozen_phonon.number_configurations *integer [default: '1']*

Number of frozen phonon configurations to calculate.

frozen_phonon.fixed_slicing *boolean [default: 'true']*

When true, the `z` coordinates (beam direction) of the atoms is not varied, resulting in fixed slicing between subsequent frozen phonon configurations.

6.1.9 `plasmon_scattering`

Settings for the `single plasmon scattering`.

```
plasmon_scattering: {
    enabled = true;              # enable or disable the feature
    simple_mode = true;                 # No energy resolution, only E = 0 and 0 <␣
↪E < max_energy
    max_energy = 10;                    # max energy of the energy grid considered␣
↪for plasmon energy transfer in eV
    energy_grid_density = 10;           # density of the energy grid in 1/eV
    mean_free_path = 120;               # mean free path of a plasmon in nm
```

(continues on next page)

(continued from previous page)

```
    plasmon_energy = 16.7;              # plasmon energy in eV
    plasmon_fwhm = 3.7;                 # plasmon energy FWHM in eV
}
```

plasmon_scattering.enabled *boolean [default: 'false']*

Whether diffuse thermal scattering via frozen phonon approximation should be enabled.

plasmon_scattering.simple_mode *integer [default: 'true']*

Number of frozen phonon configurations to calculate.

plasmon_scattering.max_energy *float (eV) [default: '10']*

Max energy of the plasmon energy grid

plasmon_scattering.energy_grid_density *float (1/eV) [default: '10']*

Density of the plasmon energy grid

plasmon_scattering.mean_free_path *float (nm) [default: '120']*

Mean free path of the plasmons in the material.

plasmon_scattering.plasmon_energy *float (eV) [default: '16.7']*

Characteristic energy of the material's plasmons.

plasmon_scattering.plasmon_fwhm *float (eV) [default: '3.7']*

Full width half maximum of the plasmon peak of the spectrum.

6.2 Command line arguments

6.2.1 stemsalabim

–help, -h *flag*

Display a help message with a brief description of available command line parameters.

–params, -p *string (required)*

Path to the configuration file as explained above in *Parameter files* files.

–num-threads *integer [default: '1']*

Number of threads per MPI core. Note, that STEMsalabim will do nothing if only parallelized via threads and `--num-threads=1`, as thread `0` of the master MPI process does not participate in the calculation. See *Hybrid Parallelization model* for details.

–package-size *integer [default: '10']*

The number of tasks that are sent to an MPI process. This should scale with the number of threads each MPI process spawns. A good value is $10\times$ the value of **–num_threads**.

–tmp-dir *string*

Override the value of the **output.tmp_dir** setting.

–output-file, -o *string*

Override the value of the **output.output_file** setting.

–crystal-file, -c *string*

Override the value of the **specimen.crystal_file** setting.

6.2.2 ssb-mkin

–help, -h *flag*

Display a help message with a brief description of available command line parameters.

–params, -p *string (required)*

Path to the configuration file as explained above in *Parameter files* files.

–num-threads *integer [default: '1']*

Number of threads per MPI core. Note, that STEMsalabim will do nothing if only parallelized via threads and `--num-threads=1`, as thread `0` of the master MPI process does not participate in the calculation. See *Hybrid Parallelization model* for details.

–output-file, -o *string*

Override the value of the **output.output_file** setting.

–crystal-file, -c *string*

Override the value of the **specimen.crystal_file** setting.

–stored-potentials *flag*

When set, `ssb-mkin` calculates the slice coulomb potentials and stores them in the output file. When `ssb-run` is also called with `--stored-potentials`, the potentials are read from the file instead of being recalculated.

6.2.3 ssb-run

–help, -h *flag*

Display a help message with a brief description of available command line parameters.

–params, -p *string (required)*

Path to the configuration file as explained above in *Parameter files* files.

–num-threads *integer [default: '1']*

Number of threads per MPI core. Note, that STEMsalabim will do nothing if only parallelized via threads and `--num-threads=1`, as thread `0` of the master MPI process does not participate in the calculation. See *Hybrid Parallelization model* for details.

–package-size *integer [default: '10']*

The number of tasks that are sent to an MPI process. This should scale with the number of threads each MPI process spawns. A good value is $10\times$ the value of **–num_threads**.

–tmp-dir *string*

Override the value of the **output.tmp_dir** setting.

–output-file, -o *string*

Override the value of the **output.output_file** setting.

–stored-potentials *flag*

When set, `ssb-run` reads the slice coulomb potentials from the input NetCDF file. They must have been calculated via `ssb-mkin --stored-potentials ...`.

CHAPTER 7

File formats

A STEMsalabim simulation is set-up via **input files** and its results are stored in an **output file**. The file for configuring a simulation is described in detail at *Parameter files*. Here, we describe the format of the **crystal file**, i.e., the atomic information about the specimen, and the **output file**, in which the results are stored.

7.1 Crystal file format

The crystal file is expected to be in XYZ format.

1. The **first line** contains the number of atoms.
2. The **second line** contains the cell dimension in x,y,z direction as floating point numbers in units of `nm`, separated by a space. Optionally, it can contain the x, y, z dimensions in the .exyz format:

   ```
   Lattice="lx 0.0 0.0 0.0 ly 0.0 0.0 0.0 lz"
   ```

3. The atomic information is given from the **third line onwards**, with each line corresponding to a single atom. Each line must have *exactly 5 columns*:
   - The atomic species as elemental abbreviation (e.g., `Ga` for gallium)
   - the x,y,z coordinates as floating point numbers in units of `nm`
   - the mean square displacement for the frozen lattice dislocations in units of nm^2.
   - (**optional**) The id of the slice this atom belongs to. This can be used to do custom slicing.

Below is a very brief, artificial example (without custom slicing):

```
5
1.0 2.0 10.0
Ga   0.0   0.0    0.0    1e-5
P    0.2   0.1    0.0    2e-5
Ga   0.0   0.0    1.0    1e-5
P    1.2   0.1    0.0    2e-5
O    1.0   2.0   10.0    0.0
```

Note: Atomic coordinates outside of the cell are periodically wrapped in `x` and `y` and clipped to the simulation box in `z` direction!

7.2 Output file format

All results of a STEMsalabim simulation are written to a binary NetCDF file. The NetCDF format is based on the Hierarchical Data Format and there are libraries to read the data for many programming languages.

The structure of NetCDF files can be inspected using the handy tool `ncdump -h YOUR_FILE.nc` (don't forget the `-h` parameter, otherwise the whole content of the file is dumped!). Here is the output of an example run:

```
netcdf out {

group: AMBER {
  dimensions:
    atom = 164140 ;
    elements = 1 ;
    spatial = 3 ;
    cell_spatial = 3 ;
    cell_angular = 3 ;
    label = 6 ;
    frame = 10 ;
    slices = 142 ;
    grid_x = 490 ;
    grid_y = 490 ;
  variables:
    char spatial(spatial) ;
    char cell_spatial(cell_spatial) ;
    char cell_angular(cell_angular, label) ;
    float coordinates(frame, atom, spatial) ;
        coordinates:unit = "nanometer" ;
    float lattice_coordinates(frame, atom, spatial) ;
    float cell_lengths(frame, cell_spatial) ;
        cell_lengths:unit = "nanometer" ;
    float cell_angles(frame, cell_angular) ;
        cell_angles:unit = "degree" ;
    float radius(frame, atom) ;
        radius:unit = "nanometer" ;
    float msd(frame, atom) ;
    int slice(frame, atom) ;
    float slice_coordinates(slices) ;
    short element(frame, atom) ;
    float system_lengths(cell_spatial) ;
    float system_angles(cell_spatial) ;
    char atom_types(elements, label) ;

  // group attributes:
        :Conventions = "AMBER" ;
        :ConventionVersion = "1.0" ;
        :program = "STEMsalabim" ;
        :programVersion = "5.0.0b" ;
        :title = "sim" ;
  } // group AMBER

group: runtime {

  // group attributes:
        :programVersionMajor = "5" ;
```

(continues on next page)

(continued from previous page)

```
        :programVersionMinor = "0" ;
        :programVersionPatch = "0b" ;
        :gitCommit = "f1dcc606c9a78b12fc3afda9496f638992b591bf" ;
        :title = "sim" ;
        :UUID = "8dce768e-f1d6-4876-bb20-c301e3e323f8" ;
        :time_start = "2019-02-12 13:25:43" ;
        :time_stop = "2019-02-13 00:06:05" ;
  } // group runtime

group: params {
  dimensions:
    defocus = 1 ;
    plasmon_energies = 51 ;
  variables:
    float defocus(defocus) ;
    float defocus_weights(defocus) ;
    float plasmon_energies(plasmon_energies) ;

  // group attributes:
        :program_arguments = "--params=inp.cfg --num-threads=4 --tmp-dir=/local --
↪output-file=out.nc" ;
        :config_file_contents = "..." ;

  group: application {

    // group attributes:
            :random_seed = 967613772U ;
    } // group application

  group: simulation {

    // group attributes:
            :title = "sim" ;
            :normalize_always = 0US ;
            :bandwidth_limiting = 1US ;
            :output_file = "out.nc" ;
            :output_compress = 0US ;
    } // group simulation

  group: probe {

    // group attributes:
            :c5 = 5000000. ;
            :cs = 2000. ;
            :astigmatism_ca = 0. ;
            :defocus = -0. ;
            :fwhm_defoci = 6. ;
            :num_defoci = 1U ;
            :astigmatism_angle = 0. ;
            :min_apert = 0. ;
            :max_apert = 15.07 ;
            :beam_energy = 200. ;
            :scan_density = 40. ;
    } // group probe

  group: specimen {

    // group attributes:
            :max_potential_radius = 0.3 ;
            :crystal_file = "Si_110_10x10x200_300K.xyz" ;
    } // group specimen
```

(continues on next page)

(continued from previous page)

```
  group: grating {

    // group attributes:
            :density = 90. ;
            :nx = 490U ;
            :ny = 490U ;
            :slice_thickness = 0.76806 ;
    } // group grating

  group: adf {

    // group attributes:
            :enabled = 1US ;
            :x = 0.5, 0.6 ;
            :y = 0.5, 0.6 ;
            :detector_min_angle = 0. ;
            :detector_max_angle = 150. ;
            :detector_num_angles = 151U ;
            :detector_interval_exponent = 1.f ;
            :average_configurations = 1US ;
            :average_defoci = 1US ;
            :save_slices_every = 10U ;
    } // group adf

  group: cbed {

    // group attributes:
            :enabled = 1US ;
            :x = 0.5, 0.6 ;
            :y = 0.5, 0.6 ;
            :size = 0U, 0U ;
            :average_configurations = 1US ;
            :average_defoci = 0US ;
            :save_slices_every = 101U ;
    } // group cbed

  group: frozen_phonon {

    // group attributes:
            :number_configurations = 10U ;
            :fixed_slicing = 1US ;
            :enabled = 1US ;
    } // group frozen_phonon

  group: plasmon_scattering {

    // group attributes:
            :enabled = 1US ;
            :simple_mode = 0US ;
            :plural_scattering = 0US ;
            :max_energy = 25.f ;
            :energy_grid_density = 2.f ;
            :mean_free_path = 128.f ;
            :plasmon_energy = 16.9f ;
            :plasmon_fwhm = 4.f ;
    } // group plasmon_scattering
  } // group params

group: adf {
  dimensions:
```

(continues on next page)

(continued from previous page)

```
    adf_position_x = 22 ;
    adf_position_y = 22 ;
    adf_detector_angle = 151 ;
    adf_defocus = 1 ;
    adf_phonon = 1 ;
    adf_slice = 15 ;
    coordinate_dim = 2 ;
    adf_plasmon_energies = 51 ;
  variables:
    float adf_intensities(adf_defocus, adf_position_x, adf_position_y, adf_phonon,␣
↪adf_slice, adf_plasmon_energies, adf_detector_angle) ;
    float center_of_mass(adf_defocus, adf_position_x, adf_position_y, adf_phonon,␣
↪adf_slice, adf_plasmon_energies, coordinate_dim) ;
    double adf_probe_x_grid(adf_position_x) ;
    double adf_probe_y_grid(adf_position_y) ;
    double adf_detector_grid(adf_detector_angle) ;
    double adf_slice_coords(adf_slice) ;
  } // group adf

group: cbed {
  dimensions:
    cbed_position_x = 22 ;
    cbed_position_y = 22 ;
    cbed_k_x = 327 ;
    cbed_k_y = 327 ;
    cbed_defocus = 1 ;
    cbed_phonon = 1 ;
    cbed_slice = 2 ;
    cbed_plasmon_energies = 51 ;
  variables:
    float cbed_intensities(cbed_defocus, cbed_position_x, cbed_position_y, cbed_
↪phonon, cbed_slice, cbed_plasmon_energies, cbed_k_x, cbed_k_y) ;
    double cbed_probe_x_grid(cbed_position_x) ;
    double cbed_probe_y_grid(cbed_position_y) ;
    double cbed_x_grid(cbed_k_x) ;
    double cbed_y_grid(cbed_k_y) ;
    double cbed_slice_coords(cbed_slice) ;
  } // group cbed
}
```

The structure of NetCDF files is hierarchical and organized in groups. The following groups are written by STEMsalabim:

7.2.1 AMBER

This group contains the atomic coordinates, species, displacements, radii, etc. for the complete crystal for each single calculated frozen lattice configuration, as well as for each calculated defocus value. The AMBER group content is compatible with the AMBER specifications. A STEMsalabim NetCDF file can be opened seamlessly with the Ovito crystal viewer.

Attributes	
Conventions	String “AMBER” (required for AMBER)
ConventionVersion	Version of the AMBER spec.
program	Program name (“STEMsalabim”)
programVersion	STEMsalabim’s version
title	Simulation title (Param `simulation.title`)
Dimensions	
atom	Number of atoms

Continued on next page

Table 1 – continued from previous page

elements	Number of different elements
spatial	Number of spatial dimensions (3)
cell_spatial	Number of spatial dimensions (3)
cell_angular	Number of spatial dimensions (3)
label	Character String for element names (6)
frame	Number of frozen phonon configurations * number of defoci
slices	Number of slices in the multi-slice approximation
grid_x	Number of simulation grid points in x direction
grid_y	Number of simulation grid points in y direction
Variables	
spatial	Names of the spatial dimensions (“x,y,z”)
cell_spatial	Names of the spatial cell parameters (“a,b,c”)
cell_angular	Names of the cell angles (“alpha,beta,gamma”)
coordinates	Coordinates of the atoms `[nm]`
lattice_coordinates	Equilibrium coordinates of the atoms (i.e. lattice positions without displacements) `[nm]`
cell_lengths	Cell lengths (Same for each frame) `[nm]`
cell_angles	Cell angles (Same for each frame, always “90, 90, 90”) `[degree]`
radius	Radii of each atom `[nm]`
msd	Mean square displacement of each atom `[nm^2]`
slice	Slice id of each atom
slice_coordinates	z-Coordinate of each slice `[nm]`
element	Element id of each atom (see `atom_types`)
system_lengths	Cell lengths `[nm]`
system_angles	Cell angles `[degree]`
atom_types	Description of atom types

7.2.2 runtime

Attributes	
programVersionMajor	Major version of STEMsalabim
programVersionMinor	Minor version of STEMsalabim
programVersionPatch	Patch version of STEMsalabim
gitCommit	Commit hash of the git commit of STEMsalabim
title	Simulation title (Param `simulation.title`)
UUID	Automatically generated universally unique identifier of this run.
time_start	Start time of the simulation run
time_stop	Finish time of the simulation run

7.2.3 params

Note: The `params` group contains subgroups with attributes that correspond exactly to the simulation parameters as written, except

- `/params/application/random_seed` is set to the generated random seed
- `/params/grating/nx` and `/params/grating/ny` contain the simulation grid size used.

Attributes	
program_arguments	CLI arguments of this run
config_file_contents	Contents of the parameter file of this run.
Dimensions	
defocus	Number of defocus (param `probe.num_defoci`)
Variables	
defocus	The values of the defoci of the defocus series
defocus_weights	The weights for defocus averaging corresponding to each defocus

7.2.4 adf

Dimensions	
adf_position_x	Number of probe positions in x direction
adf_position_y	Number of probe positions in y direction
adf_detector_angle	Number of stored detector angle bins
adf_defocus	Number of stored defoci (1 when averaged over defoci)
adf_phonon	Number of stored frozen phonon configuration (1 when averaged over configurations)
adf_slice	Number of stored slices
coordinate_dim	x,y coordinate dimension (2)
Variables	
adf_intensities	ADF intensities of each pixel `[fraction of beam]`
center_of_mass	Center of mass of each pixel `[mrad]`
adf_probe_x_grid	Position vector of the probe in x direction `[nm]`
adf_probe_y_grid	Position vector of the probe in y direction `[nm]`
adf_detector_grid	Lower angles of the detector bins `[mrad]`
adf_slice_coords	Coordinates of the stored slices `[nm]`

7.2.5 cbed

Dimensions	
cbed_position_x	Number of probe positions in x direction
cbed_position_y	Number of probe positions in y direction
cbed_k_x	Number of k grid in k_x direction
cbed_k_y	Number of k grid in k_y direction
cbed_defocus	Number of stored defoci (1 when averaged over defoci)
cbed_phonon	Number of stored frozen phonon configuration (1 when averaged over configurations)
cbed_slice	Number of stored slices
Variables	
cbed_intensities	cbed intensities of each pixel `[fraction of beam]`
cbed_probe_x_grid	Position vector of the probe in x direction `[nm]`
cbed_probe_y_grid	Position vector of the probe in y direction `[nm]`
cbed_x_grid	Angles of k_x grid `[mrad]`
cbed_y_grid	Angles of k_y grid `[mrad]`
cbed_slice_coords	Coordinates of the stored slices `[nm]`

7.3 Reading NC Files

For a detailed view of the structure, we suggest using the ncdump utility: `ncdump -h some_results_file.nc`. As the underlying file format of NetCDF is HDF5, you may use any other HDF5 viewer to have a look at the results.

There are NetCDF bindings for most popular programming languages.

1. In MATLAB, we recommend using `h5read()` and the other HDF5 functions.
2. For Python, use the netCDF4 module.

CHAPTER 8

Frequently Asked Questions

8.1 What about the name STEMsalabim?

STEMsalabim stems from the word *Simsalabim* which is known in Germany as what a magician says when casting spells (see here for a german dictionary entry). Furthermore, the phrase appears in the famous German children's song Auf einem Baum ein Kuckuck, where the phrase to our knowledge doesn't really have any meaning. (The song is about a cuckoo sitting on a tree and being shot by a huntsman...) Most germans will recognize the word. There is no contextual relation between STEM and the word Simsalabim.

When we first started writing the code we came up with it as a working title and we then simply agreed on keeping it.

CHAPTER 9

What's new

9.1 STEMsalabim 5.0.0

February 28th, 2019

IMPORTANT

The parameters *application.verbose* and *simulation.skip_simulation* are deprecated now. The groups *adf/adf_intensities*, *cbed/cbed_intensities*, and *adf/center_of_mass* now have a dimension for energy loss. It is usually *1* unless plasmon scattering feature is used.

9.1.1 Highlights

- Speed improvements by increasing the grid sizes to match efficient FFT sizes. Note, that this may result in a higher simulation grid density than specified in *grating.density* parameter!
- Alternative parallelization scheme, see *Hybrid Parallelization model*. When appropriate, different MPI procs now calculate different frozen phonon configurations / defoci in parallel. This reduces the required amount of communication between the processors.
- Automatic calculation of *center of mass* of the CBEDs for all ADF points. The COMs are calculated when *adf.enabled* = *true* and stored in the NC file next to *adf/adf_intensities* in *adf/center_of_mass*. Unit is mrad.
- New executables *ssb-mkin* and *ssb-run*. The former prepares an **input** NC file from which the latter can run the simulation. This has multiple advantages. See *Structure of a simulation* for more information.
- Single plasmon scattering.

9.1.2 Other changes

- Removed *application.verbose* parameter.
- Removed *simulation.skip_simulation*.
- Ability to disable thermal displacements via *frozen_phonon.enable* = *false* parameter.
- Fixed a serious bug with the integrated defocus averaging.
- Input XYZ files can now contain more than one space or TAB character for column separation.

- Removed Doxygen documentation and doc string comments.
- Default FFTW planning is now *FFTW_MEASURE*. This improves startup times of the simulation slightly.
- Changed the chunking of the *adf/adf_intensities* and *cbed/cbed_intensities* variables for faster write speed.
- Added *AMBER/slice_coordinates* variable to the output file, that contains the z coordinate of the upper boundary of each slice in nm.
- Removed HTTP reporting and CURL dependency.
- Significant code refactoring and some minor bugs fixed.
- Improved documentation.

9.2 STEMsalabim 4.0.1, 4.0.2

March 23rd, 2018 March 21st, 2018

- Bug fixes
- Changed chunking of the ADF variable

9.3 STEMsalabim 4.0

March 9th, 2018

IMPORTANT

I'm releasing this version as 4.0.0, but neither the input nor output files changed. The parameter *precision* has become deprecated and there is a parameter *tmp-dir*. Please see the documentation.

- Removed option for double precision. When requested, this may be re-introduced, but it slowed down compilation times and made the code significantly more complicated. The multislice algorithm with all its approximations, including the scattering factor parametrization, is not precise enough to make the difference between single and double precision significant.
- Improved the Wave class, so that some important parts can now be vectorized by the compiler.
- Introduced some more caches, so that performance could greatly be improved. STEMsalabim should now be about twice as fast as before.
- Results of the MPI processors are now written to temporary files and merged after each configuration is finished. This removes many MPI calls which tended to slow down the simulation. See the *–tmp-dir* parameter.
- Moved the *Element*, *Atom*, and *Scattering* classes to their own (isolated) library *libatomic*. This is easier to maintain.
- Simplified MPI communication by getting rid of serialization of C++ objects into char arrays. This is too error-prone anyway.
- Added compatibility with the Intel parallel studio (Compilers, MKL for FFTs, Intel MPI). Tested with Intel 17 only.
- Some minor fixes and improvements.

9.4 STEMsalabim 3.1.0, 3.1.1, 3.1.2, 3.1.3, 3.1.4

February 23nd, 2018

- Added GPL-3 License

- Moved all the code to Gitlab
- Moved documentation to readthedocs.org
- Added Gitlab CI

9.5 STEMsalabim 3.0.1 and 3.0.2

February 22nd, 2018

- Fixed a few bugs
- Improved the CMake files for better build process

9.6 STEMsalabim 3.0.0

January 3rd, 2018

- Reworked input/output file format.
- Reworked CBED storing. Blank areas due to bandwidth limiting are now removed.
- Changes to the configuration, mainly to defocus series.
- Compression can be switched on and off via config file now.
- Prepared the project for adding a Python API in the future.
- Added tapering to smoothen the atomic potential at the edges as explained in I. Lobato, et al, Ultramicroscopy 168, 17 (2016).
- Added analysis scripts for Python and MATLAB to the Si 001 example.

9.7 STEMsalabim 2.0.0

August 1st, 2017

- Changed Documentation generator to Sphinx
- Introduced a lot of memory management to prevent memory fragmentation bugs
- split STEMsalabim into a core library and binaries to ease creation of tools
- Added diagnostics output with –print-diagnostics
- Code cleanup and commenting

9.8 STEMsalabim 2.0.0-beta2

April 20th, 2017

- Added possibility to also save CBEDs, i.e., the kx/ky resolved intensities in reciprocal space.
- Improved documentation.
- Switched to NetCDF C API. Dependency on NetCDF C++ is dropped.
- Switched to distributed (parallel) writing of the NC files, which is required for the CBED feature. This requires NetCDF C and HDF5 to be compiled with MPI support.

9.9 STEMsalabim 2.0.0-beta

March 27th, 2017

- Lots of code refactoring and cleanup
- Added Doxygen doc strings
- Added Markdown documentation and `make doc` target to build this website.
- Refined the output file structure
- Added HTTP reporting feature
- Added `fixed_slicing` option to fix each atom's slice througout the simulation
- Got rid of the boost libraries to ease compilation and installation

9.10 STEMsalabim 1.0

November 18th, 2016

- Initial release.

CHAPTER 10

Citing STEMsalabim

A technical paper introducing STEMsalabim is published in Ultramicroscopy journal [1].

If you use our program or its results, please cite us. You may use the following bibTeX entry:

```
@article{Oelerich2017,
    title = "STEMsalabim: A high-performance computing cluster friendly code for scanning transmission electron microscopy image simulations of thin specimens",
    journal = "Ultramicroscopy",
    volume = "177",
    number = "",
    pages = "91 - 96",
    year = "2017",
    note = "",
    issn = "0304-3991",
    doi = "http://dx.doi.org/10.1016/j.ultramic.2017.03.010",
    url = "http://www.sciencedirect.com/science/article/pii/S030439911630300X",
    author = "Jan Oliver Oelerich and Lennart Duschek and Jürgen Belz and Andreas Beyer and Sergei D. Baranovskii and Kerstin Volz",
    keywords = "Multislice simulations",
    keywords = "Electron scattering factors",
    keywords = "MPI",
    keywords = "Phonons"
}
```

[1]: http://dx.doi.org/10.1016/j.ultramic.2017.03.010

CHAPTER 11

Research done with STEMsalabim

11.1 2018

- Composition determination of multinary III/V semiconductors via STEM HAADF multislice simulations L. Duschek, A. Beyer, J. O. Oelerich, K. Volz

11.2 2017

- Surface relaxation of strained Ga(P,As)/GaP heterostructures investigated by HAADF STEM A. Beyer, L. Duschek, J. Belz, J. O. Oelerich, K. Jandieri, K. Volz
- Atomic structure of 'W'-type quantum well heterostructures investigated by aberration-corrected STEM P. Kuekelhan, A. Beyer, C. Fuchs, M.J. Weseloh, S.W. Koch, W. Stolz, K. Volz
- Influence of surface relaxation of strained layers on atomic resolution ADF imaging A. Beyer, L. Duschek, J. Belz, J. O. Oelerich, K. Jandieri, K. Volz
- Local Bi ordering in MOVPE grown Ga(As,Bi) investigated by high resolution scanning transmission electron microscopy A. Beyer, N. Knaub, P. Rosenow, K. Jandieri, P. Ludewig, L. Bannow, S. W.Koch, R. Tonner, K. Volz
- Influence of surface relaxation of strained layers on atomic resolution ADF imaging A. Beyer, L. Duschek, J. Belz, J. O. Oelerich, K. Jandieri, K. Volz
- STEMsalabim: A high-performance computing cluster friendly code for scanning transmission electron microscopy image simulations of thin specimens J. O. Oelerich, L. Duschek, J. Belz, A. Beyer, S. D. Baranovskii, K. Volz