TEM-simulator version 1.3 reference manual

Hans Rullgård and Lars-Göran Öfverstedt

December 9, 2011

1 How to use the program

TEM-simulator is intended to generate simulated transmission electron microscopy data which can be used for testing and evaluating tomographic reconstruction methods. As input, the program requires a specification of the object being imaged, along with a number of parameters.

The program is run by the command

`TEM-simulator` <*input file*>

where <*input file*> is a text file containing parameters controlling the simulation and the names of additional input files. The structure of the input files is described in Section 3.1.

A general help message is displayed by running

```
TEM-simulator -help
```

A brief description of a particular component or input parameter is displayed by running

`TEM-simulator -help` <*component type*>

or

`TEM-simulator -help` <*component type*> <*parameter*>

For example,

```
TEM-simulator -help detector dqe
```

displays a description of the parameter `dqe` of a `detector` component.

1.1 Compiling for Linux or Mac OS X

1. TEM-simulator uses FFTW to compute Fourier transforms. If you do not have FFTW version 3 installed, download the source code from `http://www.fftw.org/download.html` and install it. If you do not know if FFTW is already installed, search the file system for files called `fftw3.h` and `libfftw3.a`, which are the files needed to compile TEM-simulator.

FFTW is most easily compiled and installed by running the three shell commands `./configure`, `make`, and `make install`. The last command copies the required FFTW files to a location where they will automatically be found by the C compiler. This might require administrator privileges. If you can't or don't want to complete this final step, you can instead enter the path to the appropriate directories in the Makefile used to compile TEM-simulator. See comments in the Makefile for more details.

2. Download and unpack the zip file containing the source code for TEM-simulator from `http://tem-simulator.sourceforge.net`.

3. Compile TEM-simulator by running `make` in the `src` directory. If necessary, first change the name of the C compiler in the `Makefile` to the one you want to use.

1.2 Running precompiled executable for Windows

TEM-simulator is delivered with an executable file called `TEM-simulator.exe` compiled using the procedure described in section 1.3 on a 32-bit computer running Windows 7. To use the precompiled executable,follow these steps.

1. Download the precompiled FFTW files for 32-bit Windows from `http://www.fftw.org/install/windows.html`. The zip file is called `fftw-3.2.2.pl1-dll32.zip`. Unpack the zip file and copy the DLL `libfftw3-3.dll` to `C:/Windows/System32` or the corresponding directory on your computer.

2. Download and unpack the zip file containing the source code for TEM-simulator from `http://tem-simulator.sourceforge.net`. The executable `TEM-simulator.exe` is run from command line.

1.3 Compiling for Windows

The following procedure has been used to compile TEM-simulator using Microsoft Visual C++ Express 2010 on a 32-bit machine running Windows 7.

1. Download the appropriate precompiled FFTW files for Windows from `http://www.fftw.org/install/windows.html`. The following instructions have been tried with the files in `fftw-3.2.2.pl1-dll32.zip`. The files needed to compile and run TEM-simulator are called `fftw3.h`, `libfftw3-3.dll`, and `libfftw3-3.def`.

2. Create an "import library" file as described in the FFTW documentation. This is most easily done using the Visual Studio Command Prompt. Open the command prompt from the Tools menu in Visual Studio, go to the FFTW directory, and run the command

   ```
   lib /def:libfftw3-3.def
   ```

   This should create a file called `libfftw3-3.lib`.

3. Copy `libfftw3-3.dll` to `C:/Windows/System32` or the corresponding directory on your computer. Copy `fftw3.h` to the Visual C++ include directory, which is called something like `C:/Program Files/Microsoft Visual Studio 10.0/VC/include/`. Copy `libfftw3-3.lib` to the Visual C++ lib directory, called something similar to `C:/Program Files/Microsoft Visual Studio 10.0/VC/lib/`. Now FFTW is ready for use.

4. Download and unpack the zip file containing the source code for TEM-simulator from `http://tem-simulator.sourceforge.net`.

5. Start Visual Studio and create a new project by selecting New, Project from the File menu. In the New Project dialog, select Win32, Win32 Console Application. Name the project `TEM-simulator`, choose a location, and click OK. In the Application Wizard, click Next, select Console application, select Empty project, deselect Precompiled header, and then click Finish.

6. In the Solution Explorer, right click Source Files, in the menu that opens select Add, Existing item. Select all the source code and header files in the TEM-simulator `src` directory (with extension `.c` and `.h`) and click Add.

7. The FFTW lib file must be added to the dependencies of the project in Visual Studio. In the toolbar, change compile mode from Debug to Release (for best performance of the compiled program). In the Project menu, click Properties. In the dialog that opens, go to Configuration Properties, Linker, Input. Select the line Additional Dependencies, and click Edit in the dropdown menu. In the dialog that opens, write `libfftw3-3.lib` and click OK. (Note that this setting applies only to the current compile mode, so it must be repeated if the compile mode is changed.)

8. The project should now be ready to compile. Choose Build Solution in the Debug menu. If everything works correctly, the executable `TEM-simulator.exe` will be in the directory `TEM-simulator/Release/` under the location chosen when setting up the project.

2 Changes from previous version

1. Added a parameter `tilt_mode` in the `geometry` component, to enable a simple kind of single particle simulations.

2. Fixed a bug which prevented the program from reading input files with Windows style line endings (carriage return + line feed).

3 Input and output files

3.1 The input file

All parameters controlling the simulation are specified in a text file which will be called the input file and whose structure is here described.

Lines in the input file are of three types: *comments*, *component headings*, and *parameter assignments*. In addition, the input file can containg empty lines (containing only space characters) which are ignored.

A comment is any line starting with the character `#` and is ignored by the program. A component heading is a line starting with the character `=`, followed by an arbitrary number of `=` and space characters (which are ignored) and a string specifying the type of component. The type of the component can optionally be followed by a string delimited by space or `=`, which specifies the name of the component. (The name is used to identify the `particle` component used in a `particleset`.)

Possible component types are `simulation`, `detector`, `electronbeam`, `geometry`, `optics`, `particle`, `particleset`, `sample` and `volume`. Details on the different types of components are given below. Additionally, the component heading `end` can be given, which indicates the end of input. A component heading of any other type and the parameter assignments following it will be ignored. The input file can contain more than one component of the types `detector`, `particle`, `particleset`, and `volume`, while the other types of components should occur only once.

A parameter assignment should be of the form

<*parameter*> `=` <*value*>

Parameters are of four different types:

Boolean The value must be `yes` or `no`.

Integer The value should be an integer written in decimal form.

Real The value should be a real number written in decimal form.

String The value can be an arbitrary string of characters. If the string contains spaces, it must be surrounded by double quotes (`"`) in the input file.

Parameter names and values of string parameters are case sensitive. Only lower case letters and underscores are currently used in parameter names. The parameters which can be used in each component type are listed in Section 4 below.

Some of the numerical input parameters have a range of allowed values. Values outside this range are immeditely rejected, since they are almost certainly unintentional. This does not imply that the software will produce realistic results for all input values within the allowed range.

3.2 The log file

The log file starts out by listing all components and parameters used in the simulation in the same format as the input file. The log file can in fact be used as input file in another simulation. Following the input listing, comments are added to the log file as major steps of the simulation are completed.

3.3 Tabular text files

Many of the input and output data used by TEM-simulator have the form of a table or matrix of numerical values. Such input and output are represented by text files with the following format.

A comment line starting with a `#` character can occur at any place in the file and is ignored. The first non-comment line, should contain two positive integers m and n, separated by space, which are the number of rows and columns in the table. The following m non-comment lines should each contain n real numbers separated by space. Any content in the file following these lines is ignored.

3.4 MRC files and raw map files

Potential maps of individual particles, maps of the entire sample, and the simulated detector images are stored in binary files. These files can be MRC files of mode 1 (2-byte integer data) or mode 2 (4-byte floating point data) or a raw 4-byte floating point format, which is equivalent to MRC mrc 2, but without the file header.

The recommended format is MRC mode 2. With files of format MRC mode 1 there is a risk of integer overflow leading to loss of data, and with raw floating point files the user must manually keep up with the size of the map or the images the data in the file represents, since there is no file header that can store such information.

For information on the MRC file format, see `http://bio3d.colorado.edu/imod/doc/mrc_format.txt`.

3.5 Other files

Besides the file formats listed above, TEM-simulator can read data from PDB files, which specify molecular structures by listing atomic coordinates. For information on the PDB file format, see `http://www.wwpdb.org/docs.html`.

4 Simulation components

Below the functionality of each simulation component is described.

4.1 Simulation

These are general parameters controlling what kind of simulations will be performed.

Parameter: `generate_micrographs`
Type: boolean
Default: `no`
Description: If this parameter has the value `yes`, a series of simulated micrographs will be generated based on the input data. With this option, components of types

`detector`, `electronbeam`, `geometry`, `optics`, and `sample` must be provided. If the simulation is to contain any molecules, at least one component of type `particle` and one of type `particleset` must also be given. Components of type `volume` are ignored.

Parameter: `generate_volumes`
Type: boolean
Default: `no`
Description: If this parameter has the value `yes`, one or several volume maps of the sample will be produced. This option requires an component of type `sample`, one or more components of type `volume` and optionally one or more components of type `particle` and `particleset`. (If the `particle` components involve the computation of an imaginary potential from PDB data, a component of type `electronbeam` is also needed.)

Parameter: `generate_particle_maps`
Type: boolean
Default: `no`
Description: If this parameter has the value `yes`, a map of each particle type specified in the input file will be generated. This will also happen automatically in the process of simulating micrographs or generating volume maps. The most common use of this option is to generate a potential map based on a PDB file, and do nothing more. This option only uses components of type `particle`.

Parameter: `log_file`
Type: string
Use: Optional
Description: This parameter gives the name of a log file which in which the actions of the program are recorded. If the file does not already exists it is created. If the file exists, the old contents are deleted. If the parameter is not provided, a unique log file name is generated based on the time when the program is started. In this case no existing file will be overwritten by the log file.

Parameter: `rand_seed`
Type: integer
Use: Optional
Description: If this parameter is proveided, it is used to seed the random number generator. If it is not provided, the random number generator is seeded by the current time.

4.2 Sample

These parameters specify the general shape of the sample which is simulated. The sample is simulated as a circular hole in a support film of uniform thickness. The hole is filled with ice, whose surfaces can be flat or concave. The sample is assumed to be symmetric about the central plane. The geometry is completely determined by specifying the diameter of the hole, the thickness of the ice at the center of the hole and at the edges, and the coordinates of the center of the hole.

Parameter: `diameter`
Type: real
Use: Required
Allowed values: Any nonnegative real number.
Description: The diameter of the sample in nm.

Parameter: paramthickness_center and `thickness_edge`
Type: real
Use: Required
Allowed values: Any nonnegative real number.
Description: The thickness of the sample at the center and at the edge in nm. `thickness_edge` is also the thickness of the surrounding support.

Parameter: `offset_x` and `offset_y`
Type: real
Allowed values: Any real number.
Default: 0
Description: The coordinates of the center of the sample.

4.3 Particle

These parameters specify the particles (for example macromolecules) which are present in the sample. Each `particle` component specifies a single particle. Multiple copies of this particle are placed in the sample using a `particleset` component. Up to 20 `particle` components can be given, for specifying different kinds of particles in the sample.

4.3.1 General parameters for the particle model

Parameter: `source`
Type: string
Use: Required
Allowed values: `map`, `pdb`, or `random`
Description: The type of source from which the particle potential is obtained. `map` means that the potential is read from a map file, `pdb` that the potential is computed from a PDB file, and `random` that a random potential is generated. The last option is useful for adding structural noise to a simulation.

Parameter: `use_imag_pot`
Type: boolean
Default: `yes`
Description: If this parameter has the value `yes`, a separate map of an imaginary potential is used in the simulation. The imaginary potential can be read from a map file, computed from a pdb file or generated randomly depending on the value of `source`. If `use_imag_pot` has the value `no`, an imaginary potential which is directly proportional to the real potential will be used, as determined by `famp`.

Parameter: `famp`
Type: real
Use: Required if `use_imag_pot` has value `no`, not used otherwise.
Allowed values: Real numbers in the range 0 to 1.
Description: A multiplicative factor used to compute an imaginary potential, which produces amplitude contrast, from the real electrostatic potential.

Parameter: `use_defocus_corr`
Type: boolean
Default: `no`
Description: If this parameter has the value `yes`, a correction is applied to correct for the varying defocus within a slice of the sample. Using this option increases the computational cost, and the difference is usually quite marginal.

Parameter: `voxel_size`
Type: real
Use: Required unless `source` is `map` and `map_file_format` is `mrc` or `mrc-int`. In the latter case it is optional.
Allowed values: Real numbers in the range 0.01 to 100.
Description: The size of voxels in the particle map in nm.

Parameter: `nx`, `ny`, and `nz`
Type: integer
Use: Required if `source` is `map` and `map_file_format` is `raw`, or if `source` is `random`. Optional if `source` is `pdb`. Not used in other cases.
Allowed values: Integers in the range 1 to 10^3
Description: The number of voxels in the particle map along the x, y, and z axes. These are needed to correctly interpret a raw map file, and to generate a random map. If the parameters are defined when the map is computed from a PDB file, they will be used to determine the number of voxels, otherwise the smallest box in which the structure will fit is used.

4.3.2 Parameters related to map files.

Parameter: `map_file_re_in`
Type: string
Use: Required if `source` has the value `map`, not used otherwise.
Description: The name of a file from which the potential map is read.

Parameter: `map_file_im_in`
Type: string
Use: Required if `source` has the value `map` and `use_imag_pot` has value `yes`, not used otherwise.
Description: The name of a file from which the imaginary part of the potential is read.

Parameter: `map_file_re_out`

Type: string
Use: Optional if `source` has value `pdb` or `random`, not used otherwise.
Description: The name of a file which the generated potential map is written to.

Parameter: `map_file_im_out`
Type: string
Use: Optional if `source` has value `pdb` or `random`, not used otherwise.
Description: The name of a file which the generated potential map is written to.

Parameter: `map_file_format`
Type: string
Allowed values: `mrc`, `mrc-int`, or `raw`
Default: `mrc`
Description: Determines the file format of input and output map files. For output files, `mrc` is MRC format with 4 byte floating point data, `mrc-int` is MRC format with 2 byte integer data, and `raw` is 4 byte floating point data and no header. For input files, `mrc` and `mrc-int` are equivalent, the MRC data format is determined by the `mode` field in the header. Supported formats are the same as for output files, that is 4 byte floating point and 2 byte integer. `raw` is the same as for output files.

Parameter: `map_file_byte_order`
Type: string
Allowed values: `be`, `le`, or `native`
Default: `native`
Description: The byte order (big endian or little endian) used in input and output map files. `native` is the native byte order of the computer running the program. When reading data from MRC files the type is determined by the header, in all other cases the byte order is determined by this parameter.

Parameter: `map_axis_order`
Type: string
Allowed values: `xyz`, `xzy`, `yxz`, `yzx`, `zxy`, or `zyx`.
Default: `xyz`
Description: The order in which voxel values are stored in input and output map files. The first variable in the string is the fastest varying.

4.3.3 Parameters for input from PDB files

Parameter: `pdb_file_in`
Type: string
Use: Required if `source` is `pdb`, not used otherwise.
Description: The name of the PDB file to read input from.

Parameter: `pdb_transf_file_in`
Type: string
Use: Optional if `source` is `pdb`, not used otherwise.
Description: The name of a tabular text file specifying a set of geometric transfor-

mations to apply to the structure in the PDB file. If the parameter is not defined, the coordinates in the PDB file are used directly. The table in the file should have 4 or 6 columns. If it has 4 colums, groups of 3 rows are interpreted as a 3×4 matrix, specifying a linear mapping and a translation. If it has 6 columns, the first three entries in each row are interpreted as a translation vector, and the following three entries as Euler angles for rotation around the z axis, then around the x axis, and again around the z axis. Translations are given in Åunits, to comply with the conventions for PDB files, and angles are in degrees.

Parameter: `add_hydrogen`
Type: boolean
Default: `yes`
Description: If the parameter has the value `yes`, hydrogen atoms, which are normally not specified in PDB files, will be added at appropriate positions in the amino acid chain. The parameter should be set to `no`if a PDB file containing records for hydrogen atoms is used as input.

4.3.4 Parameters for generating random maps

Parameter: `contrast_re` and `contrast_im`
Type: real
Use: `contrast_re` is required if `source` is `random`. `contrast_im` is required if `source` is `random` and `use_imag_pot` is `yes`.
Allowed values: Any real number.
Description: The overall contrast level of the real and imaginary part of a random map.

Parameter: `smoothness`
Type: real
Use: Required if `source` is `random`.
Allowed values: Real numbers in the range 0 to 10.
Description: The smoothness of a random map, higher values mean more smooth maps.

Parameter: `make_positive`
Type: boolean
Use: Required if `source` is `random`.
Description: If the parameter is `yes`, the random map will only have positive values, if the parameter is `no`, both positive and negative values will be randomly used. (If the parameter is `yes`and `contrast_re` or `contrast_im` is negative, only negative values will be used in the corresponding map.)

4.4 Particleset

Specifies a set of identical particles with different postitions and orientations. The positions and orientations can be read from a file or generated randomly. The particle

itself is specified in a `particle` component. Up to 20 `particleset` components can be given, for specifying different kinds of particles in the sample. If no `particleset` input is given, an empty sample is simulated.

Parameter: `particle_type`
Type: string
Use: Required
Description: The name of the `particle` component which specifies a single particle in the set.

Parameter: `particle_coords`
Type: string
Use: Required
Allowed values: `file`, `random`, or `grid`
Description: Specifies how the particle coordinates should be generated. `file` means that positions and orientations are read from a text file. `random` means that positions and orientations are randomly generated. `grid` means that the positions are on a regular grid, and the orientations are random.

Parameter: `gen_rand_positions`
Type: boolean
Use: Obsolete
Description: The parameter is included only for backward compatibility, and should be replaced by `particle_coords`. The value `yes`is equivalent to `particle_coords = random` and the value `no`is equivalent to `particle_coords = file`.

4.4.1 Parameters controlling random placement of particles

Parameter: `where`
Type: string
Use: Required if `particle_coords` has the value `random`, not used otherwise.
Allowed values: `volume`, `surface`, `top`, or `bottom`.
Description: Possible placement of particles with random coordinates. `volume` is the entire sample volume, `surface` is the top and bottom surfaces, `top` and `bottom` are only the top surface and only the bottom surface.

Parameter: `num_particles`
Type: integer
Use: Optional.
Allowed values: Integers in the range 0 to 10^6.
Description: The number of particles to use in the sample. If the parameter is defined it determines the number of random coordinates to generate, or the number of records from `coord_file_in` to use. If the parameter is not defined the number of particles is determined by `particle_conc` or `occupancy` (if `particle_coords` is `random`) or the entire file is used (if `particle_coords` is `file`).

Parameter: `particle_conc`

Type: real
Use: Optional if `particle_coords` is `random` and `num_particles` is not defined. Not used otherwise.
Allowed values: Real numbers in the range 0 to 10^6.
Description: This parameter determines the number of particles in the sample if `gen_rand_positions` is `yes`and `num_particles` is not defined. It specifies the number of particles per μm^3 (if `where` is `volume`) or per μm^2 (if `where` is `surface`, `top` or `bottom`). If neither `num_particles` or `particle_conc` is defined, the number of particles is determined by `occupancy`.

Parameter: `occupancy`
Type: real
Use: Required if `particle_coords` is `random` and neither `num_particles` or `particle_conc` is defined. Not used otherwise.
Allowed values: Real numbers in the range 0 to 100.
Description: This parameter determines the number of particles in the sample if neither `num_particles` or `particle_conc` is defined. It specifies the fraction of the volume (if `where` is `volume`) or the surface area (if `where` is `surface`, `top` or `bottom`) which is occupied by particles. Even if the value is greater than 1, the entire volume or surface may not be occupied since particles might overlap.

4.4.2 Parameters controlling placement of particles on a regular grid

Parameter: `nx`, `ny`, and `nz`
Type: integer
Use: Required if `particle_coords` is `grid`, not used otherwise.
Allowed values: Integers in the range 1 to 10^3.
Description: Specifies the number of particles if `particle_coords` is `grid`. The particles are placed in a regular `nx` $\times$ `ny` $\times$ `nz` grid centered at the origin.

Parameter: `particle_dist`
Type: real
Use: Required if `particle_coords` is `grid`, not used otherwise.
Allowed values: Any positive real value.
Description: The distance between the centers of adjacent particles in the grid.

4.4.3 Parameters specifying input and output files

Parameter: `coord_file_in`
Type: string
Use: Required if `particle_coords` is `file`, not used otherwise.
Description: The name of a tabular text file to read particle coordinates from. The table should have 6 columns. The first three columns are x, y, and z coordinates of the particle center. The following three columns are Euler angles for rotation around the z axis, then around the x axis, and again around the z axis. Coordinates are in nmunits, and angles in degrees.

Parameter: `coord_file_out`
Type: string
Use: Optional if `particle_coords` is `random` or `grid`, not used otherwise.
Description: The name of a text file to which randomly generated particle coordinates are written. The format of the file is the same as for `coord_file_in`

4.5 Geometry

The geometry component controls how the sample is moved from image to image in a tilt series. The movement can be a simple rotation around a tilt axis, but much more general tilt geometries can be accomplished by reading information from a file. It is also possible to impose random errors on the tilt geometry. A geometry component is required for simulation of micrographs.

In general, the orientation of the sample in a given image in the tilt series is controlled by three angles, ψ, θ, and ϕ. Starting from a position where the coordinate systems of the microscope and the sample coincide, the sample is rotated by the angle ψ around the z axis, then by the angle θ around the x axis, and finally by the angle ϕ around the z axis again. Here, the z axis is the optical axis of the microscope. The angles ψ, θ, and ϕ can either be read from the file specified by parameter `geom_file_in`, or computed from the three parameters `tilt_axis`, `theta_start`, and `theta_incr`.

After this rotation, the sample can optionally be subjected to further rotations and translations considered as geometry errors. This rotation is specified by three angles α, τ, and ρ. The interpretation is as follows. First the sample is rotated by the angle τ around an axis in the xy plane. The angle between this axis and the x axis is α. Next the sample is rotated by the angle ρ around the z axis. Finally, the sample is translated by a vector $(x_0, y_0, 0)$ in the xy plane. The angles α, τ, and ρ and the distances x_0 and y_0 can either be read from a file or randomly generated. When the angles are randomly generated, τ and ρ are normally distributed with given standard deviation, while α is uniformly distributed in the range $[0, \pi]$. The translations x_0 and y_0 are normally distributed with given standard deviation.

The tilt geometry can be compactly expressed as a relation between two coordinate systems, (x, y, z) which is fixed relative to the optical system and detector of the microscope, and (x', y', z') which is fixed relative to the sample. The relation between the coordinates is

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = A_{\text{tilt}} \times A_{\text{err}} \times \begin{pmatrix} x - x_0 \\ y - y_0 \\ z \end{pmatrix} \tag{1}$$

where the matrix A_{tilt} is defined by

$$A_{\text{tilt}} = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \times \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

and A_{err} is defined by

$$A_{\text{err}} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\tau & \sin\tau \\ 0 & -\sin\tau & \cos\tau \end{pmatrix} \times \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \cos\rho & \sin\rho & 0 \\ -\sin\rho & \cos\rho & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

The following parameters can be used in the geometry component.

Parameter: `gen_tilt_data`
Type: boolean
Use: Required
Description: If this parameter has the value `yes`, tilt angles will be computed based on `tilt_axis`, `theta_start`, and `theta_incr`. Otherwise the tilt angles are read from a file specified by `data_file_in`. In the former case the tilt series will consist of rotations around a single axis, with constant angular increments. If the tilt angles are read from a file much more general tilt series are possible.

Parameter: `ntilts`
Type: integer
Use: Required if `gen_tilt_data` has value `yes`, otherwise it is optional.
Allowed values: Integers in the range 1 to 10^6.
Description: The number of images in the tilt series. If the tilt angles are read from a file and the parameter is not provided, the number of images is determined from the file.

Parameter: `tilt_axis`
Type: real
Use: Only used if `gen_tilt_data` has value `yes`.
Allowed values: Any real number
Default: 0
Description: The angle in degrees between the x axis and the tilt axis. This means that $\phi = -\psi =$ `tilt_axis` for every image in the tilt series.

Parameter: `theta_start`
Type: real
Use: Required if `gen_tilt_data` has value `yes`, not used otherwise.
Allowed values: Any real number
Description: The rotation angle θ in degrees around the tilt axis for the first image in the tilt series.

Parameter: `theta_incr`
Type: real
Use: Required if `gen_tilt_data` has value `yes`, not used otherwise.
Allowed values: Any real number
Description: The increment in degrees of the rotation angle θ around the tilt axis for each successive image in the tilt series.

Parameter: `tilt_mode`
Type: string
Allowed values: `tiltseries` or `single_particle`
Default: `tiltseries`
Description: Controls how the tilt geometry is applied to the sample. `tiltseries` means that rotations are applied to the entire sample and the particle positions. `single_particle` means that the sample and particle positions stay fixed, while individual particles are rotated around their center.

Parameter: `geom_file_in`
Type: string
Use: Required if `gen_tilt_data` has value `no`, not used otherwise.
Description: The name of a tabular text file to read tilt angles from. The table should have 3 columns, which specify Euler angles ψ, θ, and ϕ for rotation around the z axis, then around the x axis, and again around the z axis. The number of rows should be at least `ntilts`, if that parameter is specified. Angles are in degrees.

Parameter: `geom_file_out`
Type: string
Use: Optional if `gen_tilt_data` has value `yes`, not used otherwise.
Description: The name of a tabular text file to which tilt angles are written. The format is the same as for `geom_file_in`

Parameter: `geom_errors`
Type: string
Use: Required
Allowed values: `none`, `random`, or `file`
Description: If the parameter has the value `none` there are no geometric errors. If the parameter has the value `random`, alignment errors are generated randomly. Finally, if the parameter has the value `file`, alignment errors are read from a file specified by `error_file_in`. The transformations specified by these parameters are applied after the tilt angle transformations.

Parameter: `tilt_err`
Type: real
Use: Used only if `errors` has the value `random`.
Allowed values: Any postitive real number.
Default: 0
Description: `tilt_err` is the standard deviation of the randomly generated angle τ, measured in degrees.

Parameter: `align_err_rot`
Type: real
Use: Used only if `errors` has the value `random`.
Allowed values: Any positive real number.
Default: 0
Description: `align_err_rot` is the standard deviation measured in degrees of the randomly generated angle ρ.

Parameter: `align_err_tr`
Type: real
Use: Used only if `errors` has the value `random`.
Allowed values: Any positive real number.
Default: 0
Description: `align_err_tr` is the standard deviation measured in nmof a random translation of the projected image. This means that x_0 and y_0 each have standard deviation `align_err_tr`$/\sqrt{2}$.

Parameter: `error_file_in`
Type: string
Use: Required if `errors` has the value `file`, not used otherwise.
Description: The name of a tabular text file to read geometry errors from. The table should have 5 columns, and at least n rows, where n is the number of tilts, defined either by `ntilts` or the number of rows in `geom_file_in`. Each row of the table should contain the angles ρ, α, and τ, measured in degrees, followed by the translations x_0 and y_0 measured in nm, for the corresponding image.

Parameter: `error_file_out`
Type: string
Use: Optional if `errors` has the value `file`, not used otherwise.
Description: The name of a tabular text file to which geometry errors are written. The format is the same as for `error_file_in`

4.6 Electronbeam

The electronbeam component controls the properties of the electron beam. An electronbeam component is required for simulation of micrographs.

Parameter: `acc_voltage`
Type: real
Use: Required
Allowed values: Real numbers in the range 1 to 10^4.
Description: The acceleration voltage of the microscope in kV. (The model used in the simulation is unlikely to produce realistic results for the whole range of possible values.)

Parameter: `energy_spread`
Type: real
Use: Required
Allowed values: Any positive real number.
Description: The energy spread of the electron beam in V.

Parameter: `gen_dose`
Type: boolean
Use: Required
Description: If the parameter has the value `yes`, the electron dose of each image is

computed based on `total_dose` and the number of images or `dose_per_im`, optionally with random variations determined by `dose_sd`. Otherwise the electron dose for each image is read from the file specified by `dose_file_in`.

Parameter: `total_dose` and `dose_per_im`
Type: real
Use: One of the parameters is requred if `gen_rand_dose` has value `yes`, not used otherwise.
Allowed values: Any positive real number.
Description: The total electron dose in all images, or the electron dose in each image, in electrons per nm^2. If both parameters are given, the dose is determined by `total_dose`.

Parameter: `dose_sd`
Type: real
Use: Used only if `gen_dose` has value `yes`.
Allowed values: Real numbers between 0 and 1.
Default: 0
Description: The standard deviation of the electron dose in each image, relative to the average dose per image. The dose in each image is computed by multiplying the average dose by a Gaussian random variable with mean 1 and standard deviation `dose_sd`. If this becomes a negative number, the dose in the image is set to 0.

Parameter: `dose_file_in`
Type: string
Use: Required if `gen_rand_dose` has value `no`, not used otherwise.
Description: Name of a tabular text file from which the electron dose to use in each image is read. The table should have one column, and at least as many rows as the number of tilts.

Parameter: `dose_file_out`
Type: string
Use: Optional if `gen_rand_dose` has value `yes`, not used otherwise.
Description: If the parameter is provided, it gives the name of a tabular text file to which the electron dose used in each image is written.

4.7 Optics

The optics component specifies the characteristics of the optical system in the microscope. Most of the optical parameters stay the same in all the images of a tilt series, but the defocus can be varied from image to image. An optics component is required for simulation of micrographs.

Parameter: `magnification`
Type: real
Use: Required
Allowed values: Any positive real number greater than 1.

Description: The magnification of the simulated microscope.

Parameter: `gen_defocus`
Type: boolean
Use: Required
Description: If this parameter has the value `yes`, a defocus value for each image is generated as a random value with specified mean and standard deviation. (The standard deviation can be set to 0, and there is then nothing random about the defocus.) Otherwise, the defocus of each image is read from a file specified by `defocus_file_in`.

Parameter: `defocus_nominal`
Type: real
Use: Required if `gen_defocus` has value `yes`, not used otherwise.
Allowed values: Any real number.
Description: The nominal defocus value measured in µm.

Parameter: `defocus_syst_error`
Type: real
Use: Used only if `gen_defocus` has value `yes`.
Allowed values: Any positive real number.
Default: 0
Description: The standard deviation of a systematic error added to the nominal defocus, measured in µm. The same error is added to the defocus of every image.

Parameter: `defocus_nonsyst_error`
Type: real
Use: Used only if `gen_defocus` has value `yes`.
Allowed values: Any positive real number.
Default: 0
Description: The standard deviation of a nonsystematic error added to the nominal defocus and the systematic error, measured in µm. A new value of the error is computed for every image.

Parameter: `cs` and `cc`
Type: real
Use: Required
Allowed values: Any real number.
Description: The spherical aberration and chromatic aberration of the optical system in mm.

Parameter: `aperture`
Type: real
Use: Required
Allowed values: Any positive real number.
Description: The diameter in µmof the aperture in the back focal plane.

Parameter: `focal_length`
Type: real

Use: Required
Allowed values: Any positive real number.
Description: The focal length in mmof the primary lens.

Parameter: `cond_ap_angle`
Type: real
Use: Required
Allowed values: Any positive real number.
Description: The aperture angle in mradof the beam furnished by the condenser lens.

Parameter: `defocus_file_in`
Type: string
Use: Required if `gen_defocus` has value `no`, not used otherwise.
Description: The name of a tabular text file to read defocus values from. The table should have one column, and at least as many rows as the number of tilts. The defocus values are in µmunits, with positive values for underfocus.

Parameter: `defocus_file_out`
Type: string
Use: Optional if `gen_defocus` has value `yes`, not used otherwise.
Description: The name of a tabular text file to which randomly generated defocus values are written.

4.8 Detector

These parameters specify the properties of the detector. Up to 10 detector components can be specified. This can be used, for example, to generate both realistic, noisy images, and noise-free images for reference in a single simulation.

The most important characteristics of a detector are the detector quantum efficiency (DQE) and the modulation transfer function (MTF). The DQE is a number between 0 and 1 which specifies how efficiently the quanta (in this case electrons) of the signal being measured are detected. The MTF is a function which specifies how well a given spatial frequency is transferred from the scintillator to the detector plane. The MTF is specified by 7 parameters, a, b, c, α, $beta$, p, and q, and is of the form

$$\mathrm{MTF}(\xi) = aP_p(1+\alpha|\xi|^2) + bP_q(1+\beta|\xi|^2) + c$$

where

$$P_n(s) = \frac{2^n}{2s^n + 2^n - 2}.$$

By adjusting these parameters, it should be possible to approximate sufficiently well the MTF of most real detectors.

Parameter: `det_pix_x` and `det_pix_y`
Type: integer
Use: Required

Allowed values: Integers in the range 1 to 10^4.
Description: The number of pixels on the detector along the x and y axes.

Parameter: `padding`
Type: integer
Allowed values: Integers in the range 0 to 10^3.
Default: 20
Description: The number of pixels used as padding around the edges of the detector in internal computations.

Parameter: `pixel_size`
Type: real
Use: Required
Allowed values: Real numbers in the range 10^-2 to 10^3.
Description: The physical size of the detector pixels in μm.

Parameter: `gain`
Type: real
Use: Required
Allowed values: Any positive real value.
Description: The detector gain, that is the average number of counts produced by each electron.

Parameter: `use_quantization`
Type: boolean
Use: Required
Description: If this parameter has the value `yes`, electron wave is quantized as discrete electrons, which is the primary source of detector noise. This is what happens in any real detector. Setting the parameter to `no`, produces practically noise-free data. While this is certainly unrealistic, it may be useful for testing and debugging.

Parameter: `dqe`
Type: real
Use: Required if `use_quantization` has the value `yes`.
Allowed values: Real numbers in the range 10^{-2} to 1.
Description: The detector quantum efficiency

Parameter: `mtf_a`, `mtf_b`, and `mtf_c`
Type: real
Use: Required
Allowed values: Real numbers in the range 0 to 1.
Description: Three of the seven parameters specifying the shape of the detector modulation transfer function (MTF).

Parameter: `mtf_alpha` and `mtf_beta`
Type: real
Use: Required
Allowed values: Any positive real number.

Description: Twi of the seven parameters specifying the shape of the detector modulation transfer function (MTF).

Parameter: `mtf_p` and `mtf_q`
Type: real
Allowed values: Real numbers in the range 0 to 10.
Default: 1
Description: Two of the seven parameters specifying the shape of the detector modulation transfer function (MTF). In most cases it should be possible to have these parameters equal to the default value 1.

Parameter: `image_file_out`
Type: string
Use: Required
Description: Name of the file to which simulated images are recorded. If the file does not already exists it is created. If the file exists, the old contents are deleted.

Parameter: `image_file_format`
Type: string
Allowed values: `mrc`, `mrc-int`, or `raw`.
Default: `mrc`
Description: The file format used to record the simulated images. `mrc` produces an MRC file with 4-byte floating point data. `mrc-int` produces an MRC file with 2-byte integer data. `raw` produces a file with no header and 4-byte floating point data.

Parameter: `image_axis_order`
Type: string
Allowed values: `xy` or `yx`.
Default: `xy`
Description: The order in which image pixels are recorded in the output file. The fastest variable is the first one in the string.

Parameter: `image_file_byte_order`
Type: string
Allowed values: `native`, `be`, or `le`.
Default: `native`
Description: The byte order used in the output file. `be` is big endian, `le` is little endian, and `native` is the byte order of the machine running the program.

4.9 Volume

The `volume` component plays no role in the actual simulation of micrographs. It is used to export to a file a map of the entire sample or a subvolume. This map can be used as a reference, for example when evaluating reconstructions made from the tilt series. Up to 20 `volume` components can be given in the input file.

Parameter: `voxel_size`

Type: real
Use: Required
Allowed values: Real numbers in the range 0.01 to 100.
Description: The size of voxels in the volume map in nm.

Parameter: `nx`, `ny`, and `nz`
Type: integer
Use: Required
Allowed values: Integers in the range 1 to 10^3
Description: The number of voxels in the volume map along the x, y, and z axes.

Parameter: `offset_x`, `offset_y`, and `offset_z`
Type: real
Allowed values: Any real number.
Default: 0
Description: The x, y, and z coordinates of the center of the volume map.

Parameter: `map_file_re_out`
Type: string
Use: Required
Description: The name of a file which the real potential map is written to.

Parameter: `map_file_im_out`
Type: string
Use: Optional.
Description: The name of a file which the imaginary potential map is written to. If the parameter is not defined, imaginary potential is not written to file.

Parameter: `map_file_format`
Type: string
Allowed values: `mrc`, `mrc-int`, or `raw`
Default: `mrc`
Description: Determines the file format of output map files. `mrc` is MRC format with 4 byte floating point data, `mrc-int` is MRC format with 2 byte integer data, and `raw` is 4 byte floating point data and no header.

Parameter: `map_file_byte_order`
Type: string
Allowed values: `be`, `le`, or `native`
Default: `native`
Description: The byte order (big endian or little endian) used output map files. `native` is the native byte order of the computer running the program.

Parameter: `map_axis_order`
Type: string
Allowed values: `xyz`, `xzy`, `yxz`, `yzx`, `zxy`, or `zyx`.
Default: `xyz`
Description: The order in which voxel values are stored in output map files. The

first variable in the string is the fastest varying.

5 Conditions of use

TEM-simulator is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

TEM-simulator is distributed in the hope that it will be useful, but **without any warranty**; without even the implied warranty of **merchandibility** or **fitness for a particular purpose**. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with TEM-simulator. If not, see `http://www.gnu.org/licenses/`.

6 Contributors

TEM-simulator is the result of a collaboration between Bertil Daneholt, Sergej Masich, and Lars-Göran Öfverstedt at the Department of Cell and Molecular Biology, Karolinska Institute in Stockholm, Ozan Öktem at Sidec AB and KTH Royal Institute of Technology in Stockholm, and Hans Rullgård at the Department of Mathematics, Stockholm University.