SMUTHI Documentation

Release 0.7.0

Amos Egel

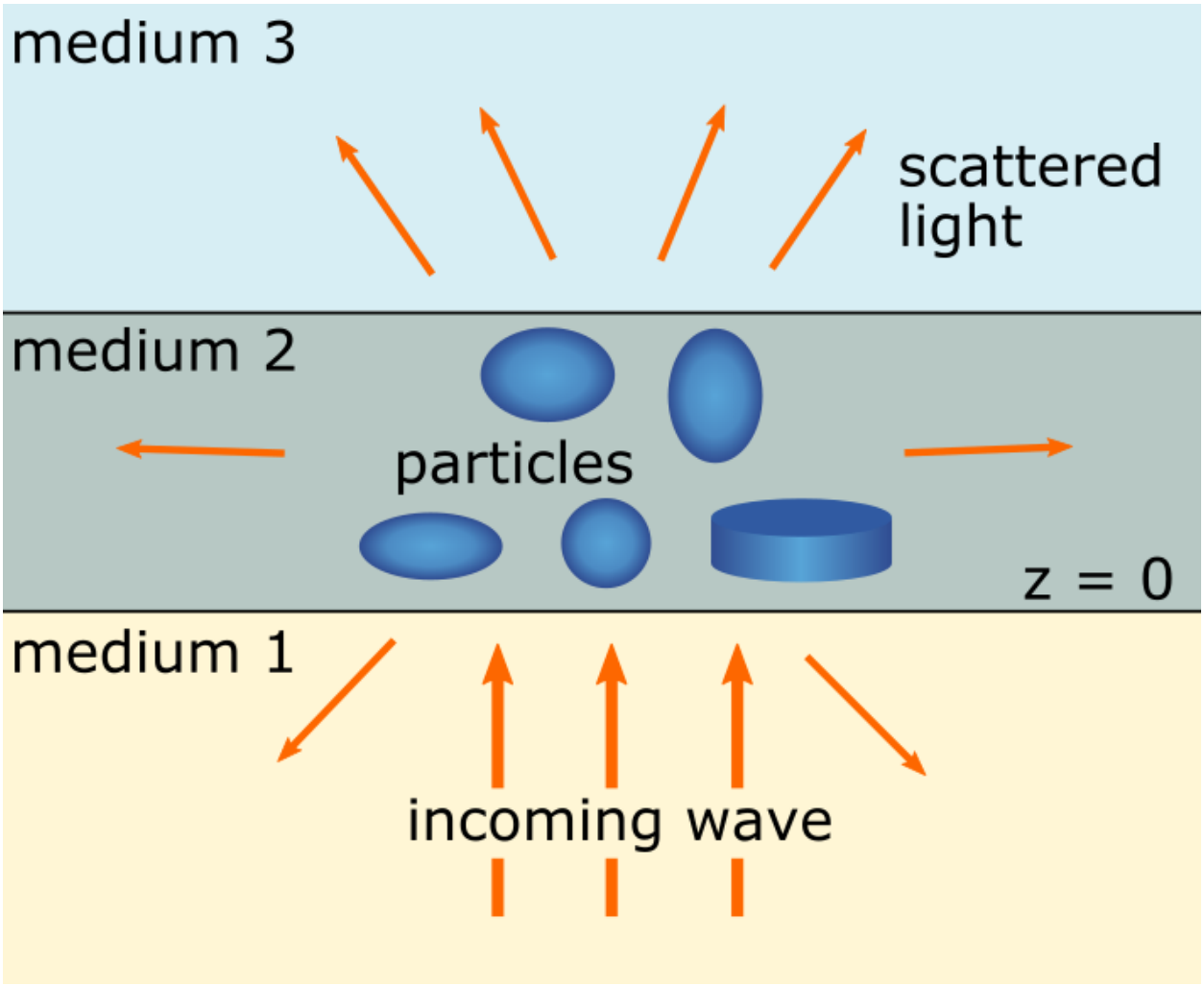
Nov 10, 2017

Contents

CHAPTER 1

About Smuthi

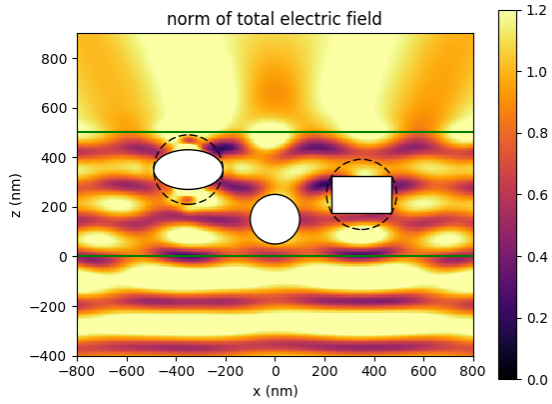
Smuthi stands for ‘scattering by multiple particles in thin-film systems’. The software allows you to solve light scattering problems involving one ore multiple particles near or inside a system of planar layer interfaces. It is based on the T-matrix method for the single particle scattering, and on the scattering-matrix method for the propagation through the layered medium.



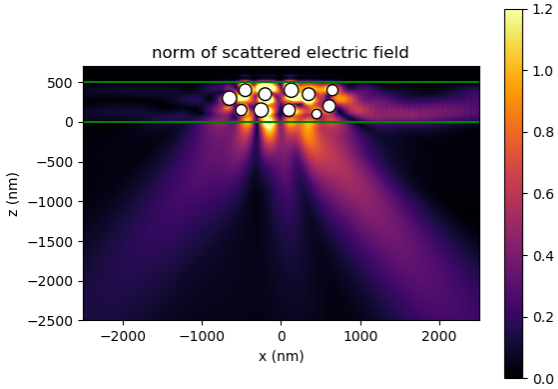
The software solves Maxwell's equations (3D wave optics) in frequency domain (one wavelength per simulation). An arbitrary number of spheres, spheroids and finite cylinders inside an arbitrary system of plane parallel layers can be modelled. For spheres, the T-matrix is given by the Mie-coefficients. For spheroids and finite cylinders, Smuthi calls the NFM-DS, to compute the single particle T-matrix. This is a Fortran software package written by A. Doicu, T. Wriedt and Y. Eremin, based on the "Null-field method with discrete sources".

As the initial excitation, Smuthi supports plane waves, Gaussian beams and single or multiple point dipole sources.

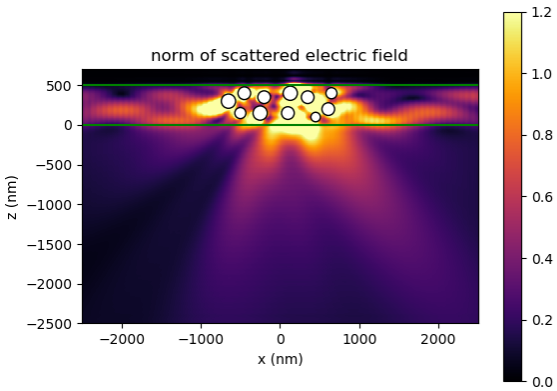
You can compute the 3D electric near field along a cut plane and save it in the form of ascii data files, png images or animations. The dashed circles around the particles are a reminder that inside the circumscribing sphere of the particles, the computed near fields cannot be trusted.



In the above example, the initial field is given by a plane wave incident from below.



The above images show an oblique Gaussian beam hitting a layer with particles under a reflective metal layer. The left image shows the norm of the scattered field, whereas the right image shows an animation of the y-component of the total field. One can see how the scattering couples some of the light into waveguide modes.

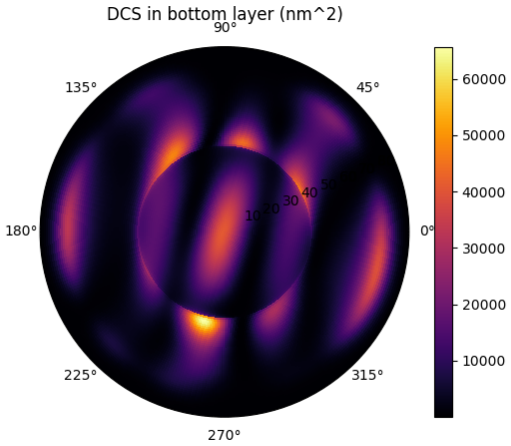


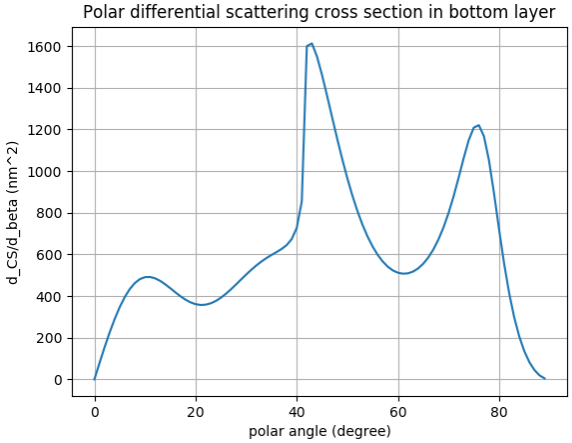
These images show the field from a dipole source between a collection of scattering particles. Again, the left shows the norm of the scattered field, whereas the right shows an animation of the y-component of the total field.

1.1 Far fields

In addition, the far field power flux can be evaluated.

- For Gaussian beams, the far field power can be related to the initial field power to evaluate reflection or transmission figures versus absorption and incoupling into waveguide modes.
- For dipole sources, the outcoupling efficiency can be studied.
- For plane wave incidence, the far field is normalized by the incoming wave's intensity to yield the differential cross section.





The above images show the 2D differential cross section in the bottom layer of the same simulation to which also the first near field images above belong. The cross section is displayed as a polar plot (left) and its azimuthal integral as a function of the polar angle only (right),

$$\mathrm{DCS}_{\mathrm{polar}}(\beta) = \int \mathrm{d}\alpha \, \sin\beta \, \mathrm{DCS}(\beta, \alpha)$$

where (α, β) are the azimuthal and polar angle, respectively.

The sharp feature around 40° in the example relates to total internal reflection at the interface between media 2 and 3.

Further, Smuthi also returns the extinction cross sections for the reflected and the transmitted wave. For the scattering of a plane wave by particles in a homogeneous medium, the extinction cross section is usually defined as the sum of total scattering and absorption cross section.

In Smuthi, we instead use what is usually referred to as the optical theorem to define extinction. That means, the extinction cross section for reflection (transmission) refers to the destructive interference of the scattered signal with the specular reflection (transmission) of the initial wave. It thereby includes absorption in the particles, scattering, and a modified absorption by the layer system, e.g. through incoupling into waveguide modes. If the particles lead to, say, a higher reflection than the bare layer system without particles, the extinction can also be negative.

Acknowledgments and contact information

Smuthi is maintained by Amos Egel. Please contact me for questions, feature requests or if you would like to contribute.

The software is licensed under the MIT license and includes contributions from the following persons:

- Adrian Doicu, Thomas Wriedt and Yuri Eremin through the NFM-DS package, a copy of which is distributed with Smuthi.

Big thanks go to Lorenzo Pattelli for designing the Smuthi logo.

The creation of Smuthi was funded by the DFG through the research project LAMBDA within the priority programme tailored disorder.

CHAPTER 2

Installation

First make sure that Python 3 is installed on your computer. With Linux, this is usually the case. Windows users can install for example Anaconda or WinPython to get a full Python environment.

2.1 Using pip

Under Windows, open a command window and type:

```
pip install smuthi
```

Depending on where pip will install the package, you might need administrator rights for that.

Under Ubuntu, type:

```
sudo pip3 install smuthi
```

2.2 Installing manually

Alternatively, you can download the Smuthi project folder manually from here or git fork https://gitlab.com/AmosEgel/smuthi.git. Open a command prompt and change directory to the Smuthi project folder. Then, enter (Windows):

```
python setup.py install
```

or (Ubuntu):

```
sudo python3 setup.py install
```

If you plan to edit the Smuthi code, install in develop mode by (Windows):

```
python setup.py develop
```

or (Ubuntu):

```
python3 setup.py develop
```

2.3 NFM-DS

When you run a Smuthi simulation (containing non-spherical particles) for the first time after installation, you will be asked to enter a path where it will install the NFM-DS Fortran package. This automatically created folder should not be removed or modified afterwards. Otherwise, the simulation of non-spherical particles becomes impossible and you might need to re-install Smuthi.

CHAPTER 3

Running a simulation

There are two different ways to call smuthi:

- From the command line with an input file. No programming skills are required.
- From a Python script. This option is more flexible regarding how to run and evaluate the simulations.

3.1 Run from command line

SMUTHI is executed from the command line together with one argument, specifying the input file that contains all parameters of the configuration to be simulated.

To execute SMUTHI, open a command window (shell or Win Python Command Prompt) and type:

```
smuthi path/to/input.dat
```

If `smuthi` is called without an argument, it uses an `example_input.dat`. The output should look like this:

```
C:\>smuthi
Please type a path where NFM-DS will be installed!
NFMDS

Reading c:\pycharmprojects\smuthi\smuthi\data\example_input.dat

**********************************
    SMUTHI version 0.2.2
**********************************

Compute initial field coefficients ... done.
Compute T-matrices ... done.
Compute direct particle coupling matrix ... done.
Compute layer system mediated particle coupling matrix ... done.
Solve linear system ... done.
Post processing ...
--------------------------------------------------------------------------
Cross sections:
Scattering into bottom layer (diffuse reflection):   61179.4958595  nm^2
Scattering into top layer (diffuse transmission):    34853.9421017  nm^2
Total scattering cross section:                      96033.4379612  nm^2
Bottom layer extinction (extinction of reflection):  29186.8096812  nm^2
Top layer extinction (extinction of transmission):   95824.3714375  nm^2
Total extinction cross section:                      125011.181119  nm^2
--------------------------------------------------------------------------
done.
```

3.1.1 The input file

The input file uses the YAML format. Download an example file `example_input.dat` and play around with its entries to get a quick start.

For a detailed explanation of the specified parameters, see the *section on input files*.

3.2 Running simulations as Python scripts

In the SMUTHI project folder, you find a script called `run_smuthi_as_script.py`. You can also download it from here by clicking on the above filename.

Edit and run that script to get a quick start. For details, see the section on running SMUTHI from scripts.

CHAPTER 4

Input files

4.1 Parameters specified in the input file

In the following, the parameters which can be specified in the input file are listed:

4.1.1 Units

Declare here the units in which you want to specify lengths and angles. The length unit has no influence on the calculations and can be chosen arbitrarily. This field is mainly there to remind the user that all lengths have to be specified in consistent units. In addition, it is used for the axis annotation of output plots. The angle units can be 'degree', otherwise radians are assumed.

length unit: nm

angle unit: degree

4.1.2 Vacuum wavelength

The vacuum wavelength λ of the electromagnetic field, in the specified length unit:

```
vacuum wavelength: 550
```

4.1.3 Layer system

Define the background geometry of the layered medium. A layer system consists of N layers, counted from bottom to top. Each layer is characterized by its thickness as well as its (real) refractive index n and extinction coefficient k (the latter is equivalent to the imaginary part of the complex refractive index $\tilde{n} = n+jk$). Provide the thickness information in the form of $[d_0, d_1, ..., d_N]$, where d_i is the thickness of the i-th layer. As the outermost layers are infinitely thick, specify them with a thickness of 0. Analogously, provide the refractive indices and extinction coefficients in the form of $[n_0, ..., n_N]$ and $[k_0, ..., k_N]$.

For example, the following entry:

```
layer system:
- thicknesses: [0, 500, 0]
  refractive indices: [1.5, 2.1, 1]
  extinction coefficients: [0, 0.01, 0]
```

would specify a single film of thickness 500, consisting of a material with complex refractive index $n_1 = 2.1 + 0.01j$, located on top of a substrate with refractive index $n_0 = 1.5$, and below air/vacuum (refractive index $n_2 = 1$).

4.1.4 Scattering particles

The ensemble of scattering particles inside the layered medium.

For spherical particles, specify `shape: sphere`, the radius, refractive index, extinction coefficient and the `[x, y, z]` coordinates of the particle position.

For spheroids, specify `shape: spheroid`, the half axes along (*half axis c*) and transverse (*half axis a*) to the axis of revolution, refractive index, extinction coefficient and the `[x, y, z]` coordinates of the particle position, as well as the Euler angles defining the rotation of the axis of revolution relative to the z axis (currently rotations other than *[0, 0, 0]* are not implemented).

For finite cylinders, specify `shape: finite cylinder`, the cylinder height, cylinder radius, refractive index, extinction coefficient and the `[x, y, z]` coordinates of the particle position, as well as the Euler angles defining the rotation of the axis of revolution relative to the z axis (currently rotations other than *[0, 0, 0]* are not implemented).

The coordinate system is such that the interface between the first two layers defines the plane $z = 0$.

In addition, specify `l_max` and `m_max`, which refer to the maximal multipole degree and order used for the spherical wave expansion of that particle's scattered field. These parameters should be chosen with reference to the desired accuracy and to the particle size parameter and refractive index contrast, see for example https://arxiv.org/ftp/arxiv/papers/1202/1202.5904.pdf A larger value leads to higher accuracy, but also to longer computation time. `l_max` is a positive integer and `m_max` is a non-negative integer and not greater than `l_max`.

In the case of non-spherical particles, you can also specify a structure `NFM-DS settings` with the fields `use discrete sources` (default is `True`), `nint` (default is `200`) and `nrank: 8` (default is `l_max + 2`). These parameters specify the calculation of the T-matrix using the NFM-DS module. For further information about the meaning of these parameters, see the NFM-DS documentation.

The parameters for the scattering particles can be listed directly in the input file, in the following format:

```
scattering particles:
- shape: sphere
  radius: 100
  refractive index: 2.4
  extinction coefficient: 0.05
  position: [0, 100, 150]
  l_max: 3
  m_max: 3
- shape: finite cylinder
  cylinder radius: 120
  cylinder height: 150
  refractive index: 2.7
  extinction coefficient: 0
  position: [350, -100, 250]
  euler angles: [0, 0, 0]
  l_max: 4
  m_max: 4
  NFM-DS settings:
```

```
    use discrete sources: true
    nint: 200
    nrank: 8
- shape: spheroid
  semi axis c: 80
  semi axis a: 140
  refractive index: 2.5
  extinction coefficient: 0.05
  position: [-350, 50, 350]
  euler angles: [0, 0, 0]
  l_max: 3
  m_max: 3
  NFM-DS settings:
    use discrete sources: true
    nint: 200
    nrank: 8
```

Alternatively, the scattering particles can be specified in a separate file, which needs to be located in the SMUTHI project folder. This is more convenient for large particle numbers. In that case, specify the filename of the particles parameters file, for example:

```
scattering particles: particle_specs.dat
```

The format of the particle specifications file is described below, see *The particle specifications file*.

4.1.5 Initial field

Currently, plane waves and beams with Gaussian transverse cross-section are implemented, as well as single or multiple electric point dipole sources.

For plane waves, specify the initial field in the following format:

```
initial field:
  type: plane wave
  polar angle: 0
  azimuthal angle: 0
  polarization: TE
  amplitude: 1
  reference point: [0, 0, 0]
```

For polarization, select either `TE` or `TM`.

The electric field of the plane wave in the layer from which it comes then reads

$$\mathbf{E}_{\mathrm{init}}(\mathbf{r}) = A \exp(\mathrm{j}\mathbf{k} \cdot (\mathbf{r} - \mathbf{r_0}))\hat{\mathbf{e}}_j,$$

where A is the amplitude, j is the imaginary unit,

$$\mathbf{k} = \frac{2\pi n_{\mathrm{init}}}{\lambda} \begin{pmatrix} \sin(\beta)\cos(\alpha) \\ \sin(\beta)\sin(\alpha) \\ \cos(\beta) \end{pmatrix}$$

is the wave vector in the layer from which the plane wave comes, n_{init} is the refractive index in that layer (must be real), (β, α) are the polar and azimuthal angle of the plane wave, $\mathbf{r_0}$ is the reference point and $\hat{\mathbf{e}}_j$ is the unit vector pointing into the α-direction for `TE` polarization and into the in the β-direction for `TM` polarization.

If the polar angle is in the range $0 \leq \beta 90°$, the k-vector has a positive z-component and consequently, the plane wave is incident from the bottom side. If the polar angle is in the range $90° \beta \leq 180°$, then the plane wave is incident from the top.

For Gaussian beams, specify the input in this format:

```
initial field:
  type: Gaussian beam
  polar angle: 0
  azimuthal angle: 0
  polarization: TE
  amplitude: 1
  focus point: [0, 0, 0]
  beam waist: 1000
```

The Gaussian beam amplitude corresponds to the electric field value at the focus point. The beam waist parameter describes the transverse width of the beam near the focus point.

More precisely, the beam is designed to fulfill

$$\mathbf{E}(\mathbf{r}) = \exp\left[\frac{(x-x_G)^2 + (y-y_G)^2}{w^2}\right] \mathbf{A}_G$$

for $z = z_G$, where (x_G, y_G, z_G) are the coordinates of the focus point, and w is the beam waist parameter and $\mathbf{A}_G$ is the amplitude vector given by the amplitude parameter and the polarization.

For a single electric point dipole source, use an input of the format:

```
initial field:
  type: dipole source
  position: [100, 10, 350]
  dipole moment: [3e7, 3e7, 0]
```

The dipole moment vector μ specifies the amplitude and the orientation of the dipole oscillation. It corresponds to a current density of

$$\mathbf{j}(\mathbf{r}) = -j\omega\mu\delta(\mathbf{r} - \mathbf{r}_D),$$

where $\mathbf{r}_D$ is the dipole position.

For multiple point dipole sources, specify the parameters in this format:

```
initial field:
  type: dipole collection
  dipoles:
  - position: [150, -100, 90]
    dipole moment: [1.5e7, 1.5e7, 0]
  - position: [-100, 100, 290]
    dipole moment: [0, 1.5e7, 1.5e7]
```

4.1.6 Numerical parameters

The radial wavevector component of a plane wave expansion is defined by a sequence `n_effective` in the complex plane, where `n_effective = k_parallel / omega` refers to the effective refractive index of the partial wave:

```
n_effective resolution: 1e-3

max n_effective: 3

n_effective imaginary deflection: 5e-2
```

'n_effective resolution' determines the sampling of the expansion/contour, where n_effective = k_parallel / omega refers to the effective refractive index of the partial wave (default=1e-2). A smaller value leads to more precise results and to a longer computation time. 'max n_effective' specifies where the expansion is truncated. It should be chosen somewhere above the maximal refractive index of the layers (default=max(refractive indices)+1). 'n_effective imaginary deflection' determines how much the contour is deflected into the lower complex half plane to avoid the vicinity of waveguide or branch point singularities (default=5e-2).

In addition, specify the resolution (in angle units) of the azimuthal angle coordinate of plane wave expansions, as well as polar and azimuthal angle coordinates of far field evaluations:

```
angular resolution: 1
```

4.1.7 Solution strategy

Choose a solver that is used for the solution of the linear system. Currently, `LU` (default) for LU-factorization and `gmres` for an interative GMRES solver are possible input. In general, the iterative solver is recommended for large particle numbers:

```
solver type: LU
```

If an iterative solver is chosen, the following setting determines at what relative accuracy the solver terminates:

```
solver tolerance: 1e-4
```

If the following parameter is set to true (default), the coupling matrix is stored explicitly. This is recommended for small particle numbers, whereas for large particle numbers, it leads to large memory consumption:

```
store coupling matrix: true
```

If the coupling matrix is not stored, matrix-vector producs are evaluated by recomputing the coupling coefficients on the fly during each step of the iterative solver. In that case, the computation time can be drastically reduced by computing the coupling coefficients through interpolation from a lookup table. For that purpose, set the following parameter to a posive value (that is the spatial resolution of the lookup table in length units):

```
coupling matrix lookup resolution: 0
```

Note:

- currently only applicable with GMRES solver and when coupling matrix NOT stored
- only applicable if all particles are in the same layer
- if NOT all particles share the same height (same position z-coordinate), and the particles are distributed over a large volume, the lookup can have very large memory footprint. In that case, consider a coarser resolution in combination with cubic interpolation (see below) to compensate the precision loss.

For the interpoloation from the lookup table, you can choose between `linear` (default, faster) and `cubic` (more preicse):

```
interpolation order: linear
```

Set the following parameter to 'true' to benefint from greatly accelerated calculations using the graphics processing unit. Requires a CUDA-enabled NVIDIA GPU, a suitable version of the CUDA toolkit and the PyCuda package installed:

```
enable GPU: false
```

4.1.8 Post processing

Define here, what output you want to generate. Currently, the following tasks can be defined for the post processing phase:

- Evaluation of the far field. If the initial field is a plane wave, the far field is interpreted in terms of the differential scattering cross section and the extinction cross section. For the case of an initial Gaussian beam, the far field denotes the radiative intensity, and relative reflectivity as well as transmittivity figures are displayed in the terminal. You can export images and raw data in ascii format.
- Evaluation of the electrical near field. You can export images, animations and raw data regarding field components or the field modulus.

Write for example:

```
post processing:
- task: evaluate far field
  show plots: true
  save plots: true
  save data: false
- task: evaluate near field
  show plots: true
  save plots: true
  save animations: true
  save data: false
  quantities to plot: [E_y, norm(E), E_scat_y, norm(E_scat), E_init_y, norm(E_init)]
  xmin: -800
  xmax: 800
  zmin: -400
  zmax: 900
  spatial resolution: 50
  interpolation spatial resolution: 5
  maximal field strength: 1.2
```

The `show plots`, `save plots` and `save data` flags deterimine, if the respective output is plotted, if the plots are saved and if the raw data is exported to ascii files.

In the `evaluate near field` task, the `save animations` flags deterimines, if the near field figures are exported as gif animations.

The `quantities to plot` are a list of strings that can be: `E_x`, `E_y`, `E_z` or `norm(E)` for the x-, y- and z-component or the norm of the total electric field, `E_scat_x`, `E_scat_y`, `E_scat_z` or `norm(E_scat)` for the x-, y- and z-component or the norm of the scattered electric field, or `E_init_x`, `E_init_y`, `E_init_z` or `norm(E_init)` for the x-, y- and z-component or the norm of the initial electric field.

To specify the plane where the near field is computed, provide `xmin`, `xmax`, `ymin`, `ymax`, `zmin` and `zmax`. If any of these is not given, it is assumed to be 0. For exactly one of the coordinates x, y or z the min and max value should be identical, e.g. `ymin` = `ymax` as in the above example. In that case, the field would be plotted in the xz-plane.

`spatial resolution` determines, how fine the grid of points is, where the near field is computed. As `xmin` etc., this parameter is specified in length units. If `interpolation spatial resolution` is specified, the near field will be interpolated to that finer value to allow for smoother looking field plots without the long computing time of a fine grained actual field evaluation.

With `maximal field strength`, you can set the color scale of the field plots to a fixed maximum.

4.1.9 Further settings for the generation of output data

The path to the output folder can be specified as:

```
output folder: smuthi_output
```

This folder will be created and in it a subfolder with a timestamp that contains all file output of the simulation.

Finally, if:

```
save simulation: true
```

is specified, the simulation object will be saved as a binary data file from which it can be reimported at a later time.

4.2 The particle specifications file

The file containing the particle specifications needs to be written in the following format:

```
# spheres
# x, y, z, radius, refractive index, exctinction coefficient, l_max, m_max
0       100     150     100     2.4     0.05    3       3
...     ...     ...     ...     ...     ...     ...     ...

# cylinders
# x, y, z, cylinder radius, cylinder height, refractive index, exctinction␣
↪coefficient, l_max, m_max
250     -100    250     120     150     2.7     0       4       4
...     ...     ...     ...     ...     ...     ...     ...     ...

# spheroids
# x, y, z, semi-axis c, semi-axis a, refractive index, exctinction coefficient, l_max,
↪ m_max
-250    0       350     80      140     2.5     0.05    3       3
...     ...     ...     ...     ...     ...     ...     ...     ...
```

An examplary particle specifiacations can be downloaded from `here`.

Back to *main page*

CHAPTER 5

API

Smuthi is a Python package with the following modules and sub-packages.

5.1 smuthi.coordinates module

smuthi.coordinates.**angular_frequency**(*vacuum_wavelength*)
Angular frequency $\omega = 2\pi c/\lambda$

Parameters **vacuum_wavelength** (*float*) – Vacuum wavelength in length unit

Returns Angular frequency in the units of c=1 (time units=length units). This is at the same time the vacuum wavenumber.

smuthi.coordinates.**complex_contour**(*vacuum_wavelength*, *neff_waypoints*, *neff_resolution*)

smuthi.coordinates.**k_z**(*k_parallel=None*, *n_effective=None*, *k=None*, *omega=None*, *vacuum_wavelength=None*, *refractive_index=None*)
z-component $k_z = \sqrt{k^2 - \kappa^2}$ of the wavevector. The branch cut is defined such that the imaginary part is not negative. Not all of the arguments need to be specified.

Parameters

- **k_parallel** (*numpy ndarray*) – In-plane wavenumber κ (inverse length)
- **n_effective** (*numpy ndarray*) – Effective refractive index n_{eff}
- **k** (*float*) – Wavenumber (inverse length)
- **omega** (*float*) – Angular frequency ω or vacuum wavenumber (inverse length, c=1)
- **vacuum_wavelength** (*float*) – Vacuum wavelength λ (length)
- **refractive_index** (*complex*) – Refractive index n_i of material

Returns z-component k_z of wavenumber with non-negative imaginary part (inverse length)

smuthi.coordinates.**set_default_k_parallel**(*vacuum_wavelength*, *neff_waypoints=None*, *neff_resolution=0.01*, *neff_max=None*, *neff_imag=0.05*)

5.2 smuthi.cuda_sources module

smuthi.cuda_sources.**enable_gpu**(*enable=True*)

Sets the use_gpu flag to enable/disable the use of CUDA kernels.

Parameters **enable** (*bool*) – Set use_gpu flag to this value (default=True).

5.3 smuthi.field_expansion module

Classes and functions to manage the expansion of the electric field in plane wave and spherical wave basis sets.

class smuthi.field_expansion.**FarField**(*polar_angles='default'*, *azimuthal_angles='default'*, *signal_type='intensity'*)

Represent the far field intensity of an electromagnetic field.

$$P = \sum_{j=1}^{2} \iint \mathrm{d}^2\Omega \, I_{\Omega,j}(\beta, \alpha),$$

where P is the radiative power, j indicates the polarization and $\mathrm{d}^2\Omega = \mathrm{d}\alpha \sin\beta \mathrm{d}\beta$ denotes the infinitesimal solid angle.

Parameters

- **polar_angles** (*numpy.ndarray*) – Polar angles (default: from 0 to 180 degree in steps of 1 degree)
- **azimuthal_angles** (*numpy.ndarray*) – Azimuthal angles (default: from 0 to 360 degree in steps of 1 degree)
- **signal_type** (*str*) – Type of the signal (e.g., 'intensity' for power flux far fields).

alpha_grid()

Returns Meshgrid with α values.

append(*other*)

Combine two FarField objects with disjoint angular ranges. The other far field is appended to this one.

Parameters **other** (FarField) – far field to append to this one.

azimuthal_integral()

Far field as a function of polar angle only.

$$P = \sum_{j=1}^{2} \int \mathrm{d}\beta \, I_{\beta,j}(\beta),$$

with

$$I_{\beta,j}(\beta) = \int \mathrm{d}\alpha \, \sin\beta I_j(\beta, \alpha),$$

Returns $I_{\beta,j}(\beta)$ as numpy ndarray. First index is polarization, second is polar angle.

beta_grid()

Returns Meshgrid with β values.

bottom()
Split far field into top and bottom part.

Returns FarField object with only the intensity for bottom hemisphere ($\beta \geq \pi/2$)

export(*output_directory='.'*, *tag='far_field'*)
Export far field information to text file in ASCII format.

Parameters

- **output_directory** (*str*) – Path to folder where to store data.
- **tag** (*str*) – Keyword to use in the naming of data files, allowing to assign them to this object.

integral()
Integrate intensity to obtain total power P.

Returns P_j as numpy 1D-array with length 2, the index referring to polarization.

top()
Split far field into top and bottom part.

Returns FarField object with only the intensity for top hemisphere ($\beta \leq \pi/2$)

class smuthi.field_expansion.**FieldExpansion**
Base class for field expansions.

diverging(*x*, *y*, *z*)
Test if points are in domain where expansion could diverge. Virtual method to be overwritten in child classes.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside divergence domain.

electric_field(*x*, *y*, *z*)
Evaluate electric field. Virtual method to be overwritten in child classes.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns Tuple of (E_x, E_y, E_z) numpy.ndarray objects with the Cartesian coordinates of complex electric field.

valid(*x*, *y*, *z*)
Test if points are in definition range of the expansion. Virtual method to be overwritten in child classes.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points

- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside definition domain.

class smuthi.field_expansion.**PiecewiseFieldExpansion**

Manage a field that is expanded in different ways for different domains, i.e., an expansion of the kind

$$\mathbf{E}(\mathbf{r}) = \sum_i \mathbf{E}_i(\mathbf{r}),$$

where

$$\mathbf{E}_i(\mathbf{r}) = \begin{cases} \tilde{\mathbf{E}}_i(\mathbf{r}) & \text{if } \mathbf{r} \in D_i \\ 0 & \text{else} \end{cases}$$

and $\tilde{\mathbf{E}_\mathbf{i}}(\mathbf{r})$ is either a plane wave expansion or a spherical wave expansion, and D_i is its domain of validity.

compatible(*other*)

Returns always true, because any field expansion can be added to a piecewise field expansion.

diverging(*x*, *y*, *z*)

Test if points are in domain where expansion could diverge.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside divergence domain.

electric_field(*x*, *y*, *z*)

Evaluate electric field.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns Tuple of (E_x, E_y, E_z) numpy.ndarray objects with the Cartesian coordinates of complex electric field.

valid(*x*, *y*, *z*)

Test if points are in definition range of the expansion.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside definition domain.

class smuthi.field_expansion.**PlaneWaveExpansion**(*k*, *k_parallel='default'*, *azimuthal_angles='default'*, *kind=None*, *reference_point=None*, *lower_z=-inf*, *upper_z=inf*)

A class to manage plane wave expansions of the form

$$\mathbf{E}(\mathbf{r}) = \sum_{j=1}^{2} \iint \mathrm{d}^2\mathbf{k}_\parallel \, g_j(\kappa,\alpha) \mathbf{\Phi}_j^{\pm}(\kappa,\alpha;\mathbf{r}-\mathbf{r}_i)$$

for $\mathbf{r}$ located in a layer defined by $z \in [z_{min}, z_{max}]$ and $\mathrm{d}^2\mathbf{k}_\parallel = \kappa \, \mathrm{d}\alpha \, \mathrm{d}\kappa$.

The double integral runs over $\alpha \in [0, 2\pi]$ and $\kappa \in [0, \kappa_{\max}]$. Further, $\mathbf{\Phi}_j^{\pm}$ are the PVWFs, see `plane_vector_wave_function()`.

Internally, the expansion coefficients $g_{ij}^{\pm}(\kappa,\alpha)$ are stored as a 3-dimensional numpy ndarray.

If the attributes k_parallel and azimuthal_angles have only a single entry, a discrete distribution is assumed:

$$g_j^{\pm}(\kappa,\alpha) \sim \delta^2(\mathbf{k}_\parallel - \mathbf{k}_{\parallel,0})$$

Parameters

- **k** (*float*) – wavenumber in layer where expansion is valid
- **k_parallel** (*numpy ndarray*) – array of in-plane wavenumbers (can be float or complex) If 'default', use smuthi.coordinates.default_k_parallel
- **azimuthal_angles** (*numpy ndarray*) – α, from 0 to 2π If 'default', use smuthi.coordinates.default_azimuthal_angles
- **kind** (*str*) – 'upgoing' for g^+ and 'downgoing' for g^- type expansions
- **reference_point** (*list or tuple*) – [x, y, z]-coordinates of point relative to which the plane waves are defined.
- **lower_z** (*float*) – the expansion is valid on and above that z-coordinate
- **upper_z** (*float*) – the expansion is valid below that z-coordinate

coefficients
numpy ndarray – coefficients[j, k, l] contains $g_j^{\pm}(\kappa_k, \alpha_l)$

azimuthal_angle_grid()
Meshgrid of azimuthal_angles with respect to n_effective

compatible(*other*)
Check if two plane wave expansions are compatible in the sense that they can be added coefficient-wise

Parameters **other** (FieldExpansion) – expansion object to add to this object

Returns bool (true if compatible, false else)

diverging(*x*, *y*, *z*)
Test if points are in domain where expansion could diverge.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside divergence domain.

electric_field(*x*, *y*, *z*)
Evaluate electric field.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns Tuple of (E_x, E_y, E_z) numpy.ndarray objects with the Cartesian coordinates of complex electric field.

k_parallel_grid()
Meshgrid of n_effective with respect to azimuthal_angles

k_z()

k_z_grid()

valid(*x*, *y*, *z*)
Test if points are in definition range of the expansion.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside definition domain.

class smuthi.field_expansion.**SphericalWaveExpansion**(*k*, *l_max*, *m_max=None*, *kind=None*, *reference_point=None*, *lower_z=-inf*, *upper_z=inf*, *inner_r=0*, *outer_r=inf*)

A class to manage spherical wave expansions of the form

$$\mathbf{E}(\mathbf{r}) = \sum_{\tau=1}^{2} \sum_{l=1}^{\infty} \sum_{m=-l}^{l} a_{\tau lm} \mathbf{\Psi}^{(\nu)}_{\tau lm}(\mathbf{r} - \mathbf{r}_i)$$

for $\mathbf{r}$ located in a layer defined by $z \in [z_{min}, z_{max}]$ and where $\mathbf{\Psi}^{(\nu)}_{\tau lm}$ are the SVWFs, see *smuthi.vector_wave_functions.spherical_vector_wave_function()*.

Internally, the expansion coefficients $a_{\tau lm}$ are stored as a 1-dimensional array running over a multi index n subsumming over the SVWF indices (τ, l, m). The mapping from the SVWF indices to the multi index is organized by the function *multi_to_single_index()*.

Parameters

- **k** (*float*) – wavenumber in layer where expansion is valid
- **l_max** (*int*) – maximal multipole degree $l_{\max} \geq 1$ where to truncate the expansion.
- **m_max** (*int*) – maximal multipole order $0 \leq m_{\max} \leq l_{\max}$ where to truncate the expansion.
- **kind** (*str*) – 'regular' for $\nu = 1$ or 'outgoing' for $\nu = 3$
- **reference_point** (*list or tuple*) – [x, y, z]-coordinates of point relative to which the spherical waves are considered (e.g., particle center).
- **lower_z** (*float*) – the expansion is valid on and above that z-coordinate
- **upper_z** (*float*) – the expansion is valid below that z-coordinate
- **inner_r** (*float*) – radius inside which the expansion diverges (e.g. circumscribing sphere of particle)

- **outer_r** (*float*) – radius outside which the expansion diverges

coefficients
numpy ndarray – expansion coefficients $a_{\tau lm}$ ordered by multi index n

coefficients_tlm(*tau*, *l*, *m*)
SWE coefficient for given (tau, l, m)

Parameters

- **tau** (*int*) – SVWF polarization (0 for spherical TE, 1 for spherical TM)
- **l** (*int*) – SVWF degree
- **m** (*int*) – SVWF order

Returns SWE coefficient

compatible(*other*)
Check if two spherical wave expansions are compatible in the sense that they can be added coefficient-wise

Parameters **other** (FieldExpansion) – expansion object to add to this object

Returns bool (true if compatible, false else)

diverging(*x*, *y*, *z*)
Test if points are in domain where expansion could diverge.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside divergence domain.

electric_field(*x*, *y*, *z*)
Evaluate electric field.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns Tuple of (E_x, E_y, E_z) numpy.ndarray objects with the Cartesian coordinates of complex electric field.

valid(*x*, *y*, *z*)
Test if points are in definition range of the expansion.

Parameters

- **x** (*numpy.ndarray*) – x-coordinates of query points
- **y** (*numpy.ndarray*) – y-coordinates of query points
- **z** (*numpy.ndarray*) – z-coordinates of query points

Returns numpy.ndarray of bool datatype indicating if points are inside definition domain.

smuthi.field_expansion.**blocksize**(*l_max*, *m_max*)
Number of coefficients in outgoing or regular spherical wave expansion for a single particle.

Parameters

- **l_max** (*int*) – Maximal multipole degree
- **m_max** (*int*) – Maximal multipole order

Returns Number of indices for one particle, which is the maximal index plus 1.

smuthi.field_expansion.**multi_to_single_index**(*tau*, *l*, *m*, *l_max*, *m_max*)

Unique single index for the totality of indices characterizing a svwf expansion coefficient.

The mapping follows the scheme:

single index	spherical wave expansion indices		
n	τ	l	m
1	1	1	-1
2	1	1	0
3	1	1	1
4	1	2	-2
5	1	2	-1
6	1	2	0
...	...	...	...
...	1	l_max	m_max
...	2	1	-1
...	...	...	...

Parameters

- **tau** (*int*) – Polarization index :math:‘tau‘(0=spherical TE, 1=spherical TM)
- **l** (*int*) – Degree l (1, ..., lmax)
- **m** (*int*) – Order m (-min(l,mmax),...,min(l,mmax))
- **l_max** (*int*) – Maximal multipole degree
- **m_max** (*int*) – Maximal multipole order

Returns single index (int) subsuming (τ, l, m)

smuthi.field_expansion.**pwe_to_ff_conversion**(*vacuum_wavelength*, *plane_wave_expansion*)

Compute the far field of a plane wave expansion object.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength in length units.
- **plane_wave_expansion** (smuthi.field_expansion.PlaneWaveExpansion) – Plane wave expansion to convert into far field object.

Returns A smuthi.field_evaluation.FarField object containing the far field intensity.

smuthi.field_expansion.**pwe_to_swe_conversion**(*pwe*, *l_max*, *m_max*, *reference_point*)

Convert plane wave expansion object to a spherical wave expansion object.

Parameters

- **pwe** (PlaneWaveExpansion) – Plane wave expansion to be converted
- **l_max** (*int*) – Maximal multipole degree of spherical wave expansion
- **m_max** (*int*) – Maximal multipole order of spherical wave expansion
- **reference_point** (*list*) – Coordinates of reference point in the format [x, y, z]

Returns SphericalWaveExpansion object.

smuthi.field_expansion.**swe_to_pwe_conversion**(*swe*, *k_parallel='default'*, *azimuthal_angles='default'*, *layer_system=None*, *layer_number=None*, *layer_system_mediated=False*)

Convert SphericalWaveExpansion object to a PlaneWaveExpansion object.

Parameters

- **swe** (SphericalWaveExpansion) – Spherical wave expansion to be converted
- **k_parallel** (*numpy array or str*) – In-plane wavenumbers for the pwe object. If 'default', use smuthi.coordinates.default_k_parallel
- **azimuthal_angles** (*numpy array or str*) – Azimuthal angles for the pwe object If 'default', use smuthi.coordinates.default_azimuthal_angles
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium in which the origin of the SWE is located
- **layer_number** (*int*) – Layer number in which the PWE should be valid.
- **layer_system_mediated** (*bool*) – If True, the PWE refers to the layer system response of the SWE, otherwise it is the direct transform.

Returns Tuple of two PlaneWaveExpansion objects, first upgoing, second downgoing.

5.4 smuthi.graphical_output module

smuthi.graphical_output.**plot_layer_interfaces**(*dim1min*, *dim1max*, *layer_system*)

Add lines to plot to display layer system interfaces

Parameters

- **dim1min** (*float*) – From what x-value plot line
- **dim1max** (*float*) – To what x-value plot line
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium

smuthi.graphical_output.**plot_particles**(*xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax*, *particle_list*, *max_particle_distance*)

Add circles, ellipses and rectangles to plot to display spheres, spheroids and cylinders.

Parameters

- **xmin** (*float*) – Minimal x-value of plot
- **xmax** (*float*) – Maximal x-value of plot
- **ymin** (*float*) – Minimal y-value of plot
- **ymax** (*float*) – Maximal y-value of plot
- **zmin** (*float*) – Minimal z-value of plot
- **zmax** (*float*) – Maximal z-value of plot
- **particle_list** (*list*) – List of smuthi.particles.Particle objects
- **max_particle_distance** (*float*) – Plot only particles that ar not further away from image plane

smuthi.graphical_output.**show_far_field**(*far_field*, *save_plots*, *show_plots*, *save_data=False*, *tag='far_field'*, *outputdir='.'*, *flip_downward=True*, *split=True*)

Display and export the far field.

Parameters

- **far_field** (smuthi.field_expansion.FarField) – far field object to show and export
- **save_plots** (*bool*) – save images if true
- **show_plots** (*bool*) – display plots if true
- **save_data** (*bool*) – export data in ascii format if true
- **tag** (*str*) – name to attribute files
- **outputdir** (*str*) – path to the directory where data to be stored
- **flip_downward** (*bool*) – represent downward directions as 0-90 deg instead of 90-180 if true
- **split** (*bool*) – show two different plots for upward and downward directions if true

smuthi.graphical_output.**show_near_field**(*quantities_to_plot=None*, *save_plots=False*, *show_plots=True*, *save_animations=False*, *save_data=False*, *outputdir='.'*, *xmin=0*, *xmax=0*, *ymin=0*, *ymax=0*, *zmin=0*, *zmax=0*, *resolution=25*, *interpolate=None*, *k_parallel='default'*, *azimuthal_angles='default'*, *simulation=None*, *max_field=None*, *max_particle_distance=inf*)

Plot the electric near field along a plane. To plot along the xy-plane, specify zmin=zmax and so on.

Parameters

- **quantities_to_plot** – List of strings that specify what to plot. Select from 'E_x', 'E_y', 'E_z', 'norm(E)' The list may contain one or more of the following strings:

  'E_x' real part of x-component of complex total electric field 'E_y' real part of y-component of complex total electric field 'E_z' real part of z-component of complex total electric field 'norm(E)' norm of complex total electric field

  'E_scat_x' real part of x-component of complex scattered electric field 'E_scat_y' real part of y-component of complex scattered electric field 'E_scat_z' real part of z-component of complex scattered electric field 'norm(E_scat)' norm of complex scattered electric field

  'E_init_x' real part of x-component of complex initial electric field 'E_init_y' real part of y-component of complex initial electric field 'E_init_z' real part of z-component of complex initial electric field 'norm(E_init)' norm of complex initial electric field
- **save_plots** (*logical*) – If True, plots are exported to file.
- **show_plots** (*logical*) – If True, plots are shown
- **save_animations** (*logical*) – If True, animated gif-images are exported
- **save_data** (*logical*) – If True, raw data are exported to file.
- **outputdir** (*str*) – Path to directory where to save the export files
- **xmin** (*float*) – Plot from that x (length unit)
- **xmax** (*float*) – Plot up to that x (length unit)

- **ymin** (*float*) – Plot from that y (length unit)
- **ymax** (*float*) – Plot up to that y (length unit)
- **zmin** (*float*) – Plot from that z (length unit)
- **zmax** (*float*) – Plot up to that z (length unit)
- **resolution** (*float*) – Compute the field with that spatial resolution (length unit)
- **interpolate** (*float*) – Use spline interpolation with that resolution to plot a smooth field (length unit)
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumbers for the plane wave expansion if 'default', use smuthi.coordinates.default_k_parallel
- **azimuthal_angles** (*numpy.ndarray or str*) – azimuthal angles for the plane wave expansion if 'default', use smuthi.coordinates.default_azimuthal_angles
- **simulation** (smuthi.simulation.Simulation) – Simulation object
- **max_field** (*float*) – If specified, truncate the color scale of the field plots at that value.
- **max_particle_distance** (*float*) – Show particles that are closer than that distance to the image plane (length unit, default = inf).

5.5 smuthi.initial_field module

class smuthi.initial_field.**DipoleCollection**(*vacuum_wavelength*)

append(*dipole*)

dissipated_power(*particle_list*, *layer_system*)

Compute the power that the dipole collection feeds into the system.

It is computed according to

$$P = \sum_i P_{0,i} + \frac{\omega}{2}\mathrm{Im}(\mu_i^* \cdot \mathbf{E}_i(\mathbf{r}_i))$$

where $P_{0,i}$ is the power that the i-th dipole would feed into an infinte homogeneous medium with the same refractive index as the layer that contains that dipole, $\mathbf{r}_i$ is the location of the i-th dipole, ω is the angular frequency, μ_i is the dipole moment and $\mathbf{E}_i$ includes the reflections of the dipole field from the layer interfaces, as well as the scattered field from all particles and the fields from all other dipoles.

Parameters

- **particle_list** (*list of smuthi.particles.Particle objects*) – scattering particles
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium

Returns dissipated power of each dipole (list of floats)

electric_field(*x*, *y*, *z*, *layer_system*)

Evaluate the complex electric field of the dipole collection.

Parameters

- **x** (*array like*) – Array of x-values where to evaluate the field (length unit)
- **y** (*array like*) – Array of y-values where to evaluate the field (length unit)

- **z** (*array like*) – Array of z-values where to evaluate the field (length unit)
- **layer_system** (*smuthi.layer.LayerSystem*) – Stratified medium

Returns Tuple (E_x, E_y, E_z) of electric field values

piecewise_field_expansion(*layer_system*)
Compute a piecewise field expansion of the dipole collection..

Parameters **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium

Returns smuthi.field_expansion.PiecewiseWaveExpansion object

plane_wave_expansion(*layer_system*, *i*, *k_parallel_array=None*, *azimuthal_angles_array=None*)
Plane wave expansion of the dipole collection's field.

Parameters

- **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium
- **i** (*int*) – layer number in which to evaluate the expansion
- **k_parallel_array** (*numpy.ndarray*) – in-plane wavenumber array for the expansion
- **azimuthal_angles_array** (*numpy.ndarray*) – azimuthal angles for the expansion

Returns tuple of to smuthi.field_expansion.PlaneWaveExpansion objects, one for upgoing and one for downgoing component

spherical_wave_expansion(*particle*, *layer_system*)

class smuthi.initial_field.**DipoleSource**(*vacuum_wavelength*, *dipole_moment*, *position*, *k_parallel='default'*, *azimuthal_angles='default'*)
Class for the representation of a single point dipole source.

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength (length units)
- **dipole_moment** (*list or tuple*) – (x, y, z)-coordinates of dipole moment vector
- **position** (*list or tuple*) – (x, y, z)-coordinates of dipole position
- **k_parallel** (*numpy.ndarray or str*) – In-plane wavenumber. If 'default', use smuthi.coordinates.default_k_parallel
- **azimuthal_angles** (*numpy.ndarray or str*) – Azimuthal angles for plane wave expansions If 'default', use smuthi.coordinates.default_azimuthal_angles

check_dissipated_power_homogeneous_background(*layer_system*)

current()
The current density takes the form

$$\mathbf{j}(\mathbf{r}) = \delta(\mathbf{r} - \mathbf{r}_D)\mathbf{j}_D,$$

where $\mathbf{j}_D = -j\omega\mu$, $\mathbf{r}_D$ is the location of the dipole, ω is the angular frequency and μ is the dipole moment. For further details, see 'Principles of nano optics' by Novotny and Hecht.

Returns List of [x, y, z]-components of current density vector $\mathbf{j}_D$

dissipated_power(*particle_list*, *layer_system*)
Compute the power that the dipole feeds into the system.

It is computed according to

$$P = P_0 + \frac{\omega}{2}\mathrm{Im}(\mu^* \cdot \mathbf{E}(\mathbf{r}_D))$$

where P_0 is the power that the dipole would feed into an infinte homogeneous medium with the same refractive index as the layer that contains the dipole, $\mathbf{r}_D$ is the location of the dipole, ω is the angular frequency, μ is the dipole moment and $\mathbf{E}$ includes the reflections of the dipole field from the layer interfaces, as well as the scattered field from all particles.

Parameters

- **particle_list** (*list of smuthi.particles.Particle objects*) – scattering particles
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium

Returns dissipated power as float

dissipated_power_homogeneous_background(*layer_system*)
Compute the power that the dipole would radiate in an infinite homogeneous medium of the same refractive index as the layer that contains the dipole.

$$P_0 = \frac{|\mu| k \omega^3}{12\pi}$$

where :math: P = P_0 + frac{omega}{2} mathrm{Im} (mathbf{mu}^* cdot mathbf{E}(mathbf{r}_D))

Parameters **layer_system** (smuthi.layers.LayerSystem) – stratified medium

Returns power (float)

electric_field(*x*, *y*, *z*, *layer_system*, *include_direct_field=True*)
Evaluate the complex electric field of the dipole source.

Parameters

- **x** (*array like*) – Array of x-values where to evaluate the field (length unit)
- **y** (*array like*) – Array of y-values where to evaluate the field (length unit)
- **z** (*array like*) – Array of z-values where to evaluate the field (length unit)
- **layer_system** (*smuthi.layer.LayerSystem*) – Stratified medium
- **include_direct_field** (*bool*) – if True (default), the direct dipole field is included. otherwise, only the layer response of the dipole field is returned.

Returns Tuple (E_x, E_y, E_z) of electric field values

outgoing_spherical_wave_expansion(*layer_system*)
The dipole field as an expansion in spherical vector wave functions.

Parameters **layer_system** (smuthi.layers.LayerSystem) – stratified medium

Returns outgoing smuthi.field_expansion.SphericalWaveExpansion object

piecewise_field_expansion(*layer_system*, *include_direct_field=True*)
Compute a piecewise field expansion of the dipole field.

Parameters

- **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium
- **include_direct_field** (*bool*) – if True (default), the direct dipole field is included. otherwise, only the layer response of the dipole field is returned.

Returns smuthi.field_expansion.PiecewiseWaveExpansion object

plane_wave_expansion(*layer_system*, *i*, *k_parallel_array=None*, *azimuthal_angles_array=None*)
Plane wave expansion of the dipole field.

Parameters

- **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium
- **i** (*int*) – layer number in which to evaluate the expansion
- **k_parallel_array** (*numpy.ndarray*) – in-plane wavenumber array for the expansion. if none specified, self.k_parallel_array is used
- **azimuthal_angles_array** (*numpy.ndarray*) – azimuthal angles for the expansion. if none specified, self.azimuthal_angles_array is used

Returns tuple of to smuthi.field_expansion.PlaneWaveExpansion objects, one for upgoing and one for downgoing component

spherical_wave_expansion(*particle*, *layer_system*)
Regular spherical wave expansion of the wave including layer system response, at the locations of the particles.

Parameters

- **particle** (smuthi.particles.Particle) – particle relative to which the swe is computed
- **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium

Returns regular smuthi.field_expansion.SphericalWaveExpansion object

class smuthi.initial_field.**GaussianBeam**(*vacuum_wavelength*, *polar_angle*, *azimuthal_angle*, *polarization*, *beam_waist*, *k_parallel_array='default'*, *azimuthal_angles_array='default'*, *amplitude=1*, *reference_point=None*)
Class for the representation of a Gaussian beam as initial field.

initial_intensity(*layer_system*)
Evaluate the incoming intensity of the initial field.

Parameters **layer_system** (smuthi.layers.LayerSystem) – Stratified medium

Returns A smuthi.field_evaluation.FarField object holding the initial intensity information.

plane_wave_expansion(*layer_system*, *i*, *k_parallel_array=None*, *azimuthal_angles_array=None*)
Plane wave expansion of the Gaussian beam.

Parameters

- **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium
- **i** (*int*) – layer number in which to evaluate the expansion
- **k_parallel_array** (*numpy.ndarray*) – in-plane wavenumber array for the expansion. if none specified, self.k_parallel_array is used
- **azimuthal_angles_array** (*numpy.ndarray*) – azimuthal angles for the expansion. if none specified, self.azimuthal_angles_array is used

Returns tuple of to smuthi.field_expansion.PlaneWaveExpansion objects, one for upgoing and one for downgoing component

propagated_far_field(*layer_system*)
Evaluate the far field intensity of the reflected / transmitted initial field.

Parameters **layer_system** (smuthi.layers.LayerSystem) – Stratified medium

Returns A tuple of smuthi.field_evaluation.FarField objects, one for forward (i.e., into the top hemisphere) and one for backward propagation (bottom hemisphere).

class smuthi.initial_field.**InitialField**(*vacuum_wavelength*)
Base class for initial field classes

angular_frequency()
Angular frequency.

Returns Angular frequency (float) according to the vacuum wavelength in units of c=1.

piecewise_field_expansion(*layer_system*)
Virtual method to be overwritten.

plane_wave_expansion(*layer_system*, *i*)
Virtual method to be overwritten.

spherical_wave_expansion(*particle*, *layer_system*)
Virtual method to be overwritten.

class smuthi.initial_field.**InitialPropagatingWave**(*vacuum_wavelength*, *polar_angle*, *azimuthal_angle*, *polarization*, *amplitude=1*, *reference_point=None*)
Base class for plane waves and Gaussian beams

Parameters

- **vacuum_wavelength** (*float*) –
- **polar_angle** (*float*) – polar propagation angle (0 means, parallel to z-axis)
- **azimuthal_angle** (*float*) – azimuthal propagation angle (0 means, in x-z plane)
- **polarization** (*int*) – 0 for TE/s, 1 for TM/p
- **amplitude** (*float or complex*) – Electric field amplitude
- **reference_point** (*list*) – Location where electric field of incoming wave equals amplitude

electric_field(*x*, *y*, *z*, *layer_system*)
Evaluate the complex electric field corresponding to the wave.

Parameters

- **x** (*array like*) – Array of x-values where to evaluate the field (length unit)
- **y** (*array like*) – Array of y-values where to evaluate the field (length unit)
- **z** (*array like*) – Array of z-values where to evaluate the field (length unit)
- **layer_system** (*smuthi.layer.LayerSystem*) – Stratified medium

Returns Tuple (E_x, E_y, E_z) of electric field values

piecewise_field_expansion(*layer_system*)
Compute a piecewise field expansion of the initial field.

Parameters **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium

Returns smuthi.field_expansion.PiecewiseWaveExpansion object

spherical_wave_expansion(*particle*, *layer_system*)
Regular spherical wave expansion of the wave including layer system response, at the locations of the particles.

Parameters

- **particle** (smuthi.particles.Particle) – particle relative to which the swe is computed
- **layer_system** (*smuthi.layer.LayerSystem*) – stratified medium

Returns regular smuthi.field_expansion.SphericalWaveExpansion object

class smuthi.initial_field.**PlaneWave**(*vacuum_wavelength*, *polar_angle*, *azimuthal_angle*, *polarization*, *amplitude=1*, *reference_point=None*)
Class for the representation of a plane wave as initial field.

Parameters

- **vacuum_wavelength** (*float*) –
- **polar_angle** (*float*) – polar angle of k-vector (0 means, k is parallel to z-axis)
- **azimuthal_angle** (*float*) – azimuthal angle of k-vector (0 means, k is in x-z plane)
- **polarization** (*int*) – 0 for TE/s, 1 for TM/p
- **amplitude** (*float or complex*) – Plane wave amplitude at reference point
- **reference_point** (*list*) – Location where electric field of incoming wave equals amplitude

plane_wave_expansion(*layer_system*, *i*)
Plane wave expansion for the plane wave including its layer system response. As it already is a plane wave, the plane wave expansion is somehow trivial (containing only one partial wave, i.e., a discrete plane wave expansion).

Parameters

- **layer_system** (smuthi.layers.LayerSystem) – Layer system object
- **i** (*int*) – layer number in which the plane wave expansion is valid

Returns Tuple of smuthi.field_expansion.PlaneWaveExpansion objects. The first element is an upgoing PWE, whereas the second element is a downgoing PWE.

5.6 smuthi.layers module

Provide class for the representation of planar layer systems.

class smuthi.layers.**LayerSystem**(*thicknesses=None*, *refractive_indices=None*)
Stack of planar layers.

Parameters

- **thicknesses** (*list*) – layer thicknesses, first and last are semi inf and set to 0 (length unit)
- **refractive_indices** (*list*) – complex refractive indices in the form n+jk

layer_number(*z*)
Return number of layer that contains point [0,0,z]

If z is on the interface, the higher layer number is selected.

Parameters **z** (*float*) – z-coordinate of query point (length unit)

Returns number of layer containing z

lower_zlimit(*i*)
Return the z-coordinate of lower boundary

The coordinate system is defined such that z=0 corresponds to the interface between layer 0 and layer 1.

Parameters **i** (*int*) – index of layer in question (must be between 0 and number_of_layers-1)

Returns z-coordinate of lower boundary

number_of_layers()
Return total number of layers

Returns number of layers

reference_z(*i*)
Return the anchor point's z-coordinate.

The coordinate system is defined such that z=0 corresponds to the interface between layer 0 and layer 1.

Parameters **i** (*int*) – index of layer in question (must be between 0 and number_of_layers-1)

Returns anchor point's z-coordinate

response(*pwe*, *from_layer*, *to_layer*)
Evaluate the layer system response to an electromagnetic excitation inside the layer system.

Parameters

- **pwe** (*tuple or* smuthi.field_expansion.PlaneWaveExpansion) – Either specify a PlaneWaveExpansion object that that represents the electromagnetic excitation, or a tuple of two PlaneWaveExpansion objects representing the upwards- and downwards propagating partial waves of the excitation.
- **from_layer** (*int*) – Layer number in which the excitation is located
- **to_layer** (*int*) – Layer number in which the layer response is to be evaluated

Returns Tuple (pwe_up, pwe_sown) of PlaneWaveExpansion objects representing the layer system response to the excitation.

upper_zlimit(*i*)
Return the z-coordinate of upper boundary.

The coordinate system is defined such that z=0 corresponds to the interface between layer 0 and layer 1.

Parameters **i** (*int*) – index of layer in question (must be between 0 and number_of_layers-1)

Returns z-coordinate of upper boundary

wavenumber(*layer_number*, *vacuum_wavelength*)

Parameters

- **layer_number** (*int*) – number of layer in question
- **vacuum_wavelength** (*float*) – vacuum wavelength

Returns wavenumber in that layer as float

smuthi.layers.**fresnel_r**(*pol*, *kz1*, *kz2*, *n1*, *n2*)
Fresnel reflection coefficient.

Parameters

- **pol** (*int*) – polarization (0=TE, 1=TM)
- **kz1** (*float or array*) – incoming wave's z-wavenumber (k*cos(alpha1))
- **kz2** (*float or array*) – transmitted wave's z-wavenumber (k*cos(alpha2))
- **n1** (*float or complex*) – first medium's complex refractive index (n+ik)
- **n2** (*float or complex*) – second medium's complex refractive index (n+ik)

Returns Complex Fresnel reflection coefficient (float or array)

smuthi.layers.**fresnel_t**(*pol*, *kz1*, *kz2*, *n1*, *n2*)
Fresnel transmission coefficient.

Parameters

- **pol** (*int*) – polarization (0=TE, 1=TM)
- **kz1** (*float or array*) – incoming wave's z-wavenumber (k*cos(alpha1))
- **kz2** (*float or array*) – transmitted wave's z-wavenumber (k*cos(alpha2))
- **n1** (*float or complex*) – first medium's complex refractive index (n+ik)
- **n2** (*float or complex*) – second medium's complex refractive index (n+ik)

Returns Complex Fresnel transmission coefficient (float or array)

smuthi.layers.**interface_transition_matrix**(*pol*, *kz1*, *kz2*, *n1*, *n2*)
Interface transition matrix to be used in the Transfer matrix algorithm.

Parameters

- **pol** (*int*) – polarization (0=TE, 1=TM)
- **kz1** (*float or array*) – incoming wave's z-wavenumber (k*cos(alpha1))
- **kz2** (*float or array*) – transmitted wave's z-wavenumber (k*cos(alpha2))
- **n1** (*float or complex*) – first medium's complex refractive index (n+ik)
- **n2** (*float or complex*) – second medium's complex refractive index (n+ik)

Returns Interface transition matrix as 2x2 numpy array or as 2x2 mpmath.matrix

smuthi.layers.**layer_propagation_matrix**(*kz*, *d*)
Layer propagation matrix to be used in the Transfer matrix algorithm.

Parameters

- **kz** (*float or complex*) – z-wavenumber (k*cos(alpha))
- **d** (*float*) – thickness of layer

Returns Layer propagation matrix as 2x2 numpy array or as 2x2 mpmath.matrix

smuthi.layers.**layersystem_scattering_matrix**(*pol*, *layer_d*, *layer_n*, *kpar*, *omega*)
Scattering matrix of a planarly layered medium.

Parameters

- **pol** (*int*) – polarization(0=TE, 1=TM)
- **layer_d** (*list*) – layer thicknesses

- **layer_n** (*list*) – complex layer refractive indices
- **kpar** (*float*) – in-plane wavenumber
- **omega** (*float*) – angular frequency in units of c=1: omega=2*pi/lambda

Returns Scattering matrix as 2x2 numpy array or as 2x2 mpmath.matrix

smuthi.layers.**layersystem_transfer_matrix**(*pol*, *layer_d*, *layer_n*, *kpar*, *omega*)
Transfer matrix of a planarly layered medium.

Parameters

- **pol** (*int*) – polarization(0=TE, 1=TM)
- **layer_d** (*list*) – layer thicknesses
- **layer_n** (*list*) – complex layer refractive indices
- **kpar** (*float*) – in-plane wavenumber
- **omega** (*float*) – angular frequency in units of c=1: omega=2*pi/lambda

Returns Transfer matrix as 2x2 numpy array or as 2x2 mpmath.matrix

smuthi.layers.**matrix_inverse**(*m*)

Parameters **m** (*mpmath.matrix or numpy.ndarray*) – matrix to invert

Returns inverse of m with same data type as m1 and m2

smuthi.layers.**matrix_product**(*m1*, *m2*)

Parameters

- **m1** (*mpmath.matrix or numpy.ndarray*) – first matrix
- **m2** (*mpmath.matrix or numpy.ndarray*) – second matrix

Returns matrix product m1 * m2 with same data type as m1 and m2

smuthi.layers.**set_precision**(*prec=None*)
Set the numerical precision of the layer system response. You can use this to evaluate the layer response of unstable systems, for example in the case of evanescent waves in very thick layers. Calculations take longer time if the precision is set to a value other than None (default).

Parameters **prec** (*None or int*) – If None, calculations are done using standard double precision. If int, that many decimal digits are considered in the calculations, using the mpmath package.

5.7 smuthi.linear_system module

class smuthi.linear_system.**CouplingMatrixExplicit**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*)
Class for an explicit representation of the coupling matrix. Recommended for small particle numbers.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength in length units
- **particle_list** (*list*) – List of smuthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium

- **k_parallell** (*numpy.ndarray or str*) – In-plane wavenumber. If 'default', use smuthi.coordinates.default_k_parallel

class smuthi.linear_system.**CouplingMatrixRadialLookup**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*)

Base class for radial lookup based coupling matrix either on CPU or on GPU (CUDA).

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength in length units
- **particle_list** (*list*) – list of sumthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **resolution** (*float or None*) – spatial resolution of the lookup in the radial direction

class smuthi.linear_system.**CouplingMatrixRadialLookupCPU**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*, *interpolator_kind='linear'*)

Class for radial lookup based coupling matrix running on CPU. This is used when no suitable GPU device is detected or when PyCuda is not installed.

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength in length units
- **particle_list** (*list*) – list of sumthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **resolution** (*float or None*) – spatial resolution of the lookup in the radial direction
- **kind** (*str*) – interpolation order, e.g. 'linear' or 'cubic'

class smuthi.linear_system.**CouplingMatrixRadialLookupCUDA**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*, *cuda_blocksize=128*, *interpolator_kind='linear'*)

Radial lookup based coupling matrix either on GPU (CUDA).

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength in length units
- **particle_list** (*list*) – list of sumthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **resolution** (*float or None*) – spatial resolution of the lookup in the radial direction

- **cuda_blocksize** (*int*) – threads per block when calling CUDA kernel

class smuthi.linear_system.**CouplingMatrixVolumeLookup**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*)

Base class for 3D lookup based coupling matrix either on CPU or on GPU (CUDA).

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength in length units
- **particle_list** (*list*) – list of sumthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **resolution** (*float or None*) – spatial resolution of the lookup in the radial direction

class smuthi.linear_system.**CouplingMatrixVolumeLookupCPU**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*, *interpolator_kind='cubic'*)

Class for 3D lookup based coupling matrix running on CPU. This is used when no suitable GPU device is detected or when PyCuda is not installed.

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength in length units
- **particle_list** (*list*) – list of sumthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **resolution** (*float or None*) – spatial resolution of the lookup in the radial direction
- **interpolator_kind** (*str*) – 'linear' or 'cubic' interpolation

class smuthi.linear_system.**CouplingMatrixVolumeLookupCUDA**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*, *cuda_blocksize=128*, *interpolator_kind='linear'*)

Class for 3D lookup based coupling matrix running on GPU.

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength in length units
- **particle_list** (*list*) – list of sumthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **resolution** (*float or None*) – spatial resolution of the lookup in the radial direction
- **cuda_blocksize** (*int*) – threads per block for cuda call

- **interpolator_kind** (*str*) – 'linear' (default) or 'cubic' interpolation

class smuthi.linear_system.**LinearSystem**(*particle_list*, *initial_field*, *layer_system*, *k_parallel='default'*, *solver_type='LU'*, *solver_tolerance=0.0001*, *store_coupling_matrix=True*, *coupling_matrix_lookup_resolution=None*, *interpolator_kind='cubic'*, *cuda_blocksize=128*)

Manage the assembly and solution of the linear system of equations.

Parameters

- **particle_list** (*list*) – List of smuthi.particles.Particle objects
- **initial_field** (smuthi.initial_field.InitialField) – Initial field object
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber. If 'default', use smuthi.coord.default_k_parallel
- **solver_type** (*str*) – What solver to use? Options: 'LU' for LU factorization, 'gmres' for GMRES iterative solver
- **store_coupling_matrix** (*bool*) – If True (default), the coupling matrix is stored. Otherwise it is recomputed on the fly during each iteration of the solver.
- **coupling_matrix_lookup_resolution** (*float or None*) – If type float, compute particle coupling by interpolation of a lookup table with that spacial resolution. A smaller number implies higher accuracy and memory footprint. If None (default), don't use a lookup table but compute the coupling directly. This is more suitable for a small particle number.
- **interpolator_kind** (*str*) – interpolation order to be used, e.g. 'linear' or 'cubic'. This argument is ignored if coupling_matrix_lookup_resolution is None. In general, cubic interpolation is more accurate but a bit slower than linear.

solve()

Compute scattered field coefficients and store them in the particles' spherical wave expansion objects.

class smuthi.linear_system.**MasterMatrix**(*t_matrix*, *coupling_matrix*)

Represent the master matrix $M = 1 - TW$ as a linear operator.

Parameters

- **t_matrix** (*SystemTMatrix*) – T-matrix object
- **coupling_matrix** (*CouplingMatrix*) – Coupling matrix object

class smuthi.linear_system.**SystemMatrix**(*particle_list*)

A system matrix is an abstract linear operator that operates on a system coefficient vector, i.e. a vector $c = c^i_{\tau,l,m}$, where (τ, l, m) are the multipole indices and i indicates the particle number.

index(*i*, *tau*, *l*, *m*)

Parameters

- **i** (*int*) – particle number
- **tau** (*int*) – spherical polarization index
- **l** (*int*) – multipole degree
- **m** (*int*) – multipole order

Returns Position in a system vector that corresponds to the (τ, l, m) coefficient of the i-th particle.

index_block(*i*)

Parameters **i** (*int*) – number of particle

Returns indices that correspond to the coefficients for that particle

class smuthi.linear_system.**TMatrix**(*particle_list*)
Collect the particle T-matrices in a global lienear operator.

Parameters **particle_list** (*list*) – List of smuthi.particles.Particle objects containing a t_matrix attribute.

right_hand_side()
The right hand side of the linear system is given by $\sum_{\tau lm} T^i_{\tau lm} a^i_{\tau lm}$

Returns right hand side as a complex numpy.ndarray

5.8 smuthi.memoizing module

Provide functionality to store intermediate results in lookup tables (memoize)

class smuthi.memoizing.**Memoize**(*fn*)
To be used as a decorator for functions that are memoized.

5.9 smuthi.particle_coupling module

Routines for multiple scattering. The first half of the module contains functions to explicitly compute the coupling matrix entries. The second half of the module contains functions for the preparation of lookup tables that are used to approximate the coupling matrices by interoplation.

smuthi.particle_coupling.**direct_coupling_block**(*vacuum_wavelength*, *receiving_particle*, *emitting_particle*, *layer_system*)
Direct particle coupling matrix W for two particles. This routine is explicit, but slow.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength λ (length unit)
- **receiving_particle** (smuthi.particles.Particle) – Particle that receives the scattered field
- **emitting_particle** (smuthi.particles.Particle) – Particle that emits the scattered field
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium in which the coupling takes place

Returns Direct coupling matrix block as numpy array.

smuthi.particle_coupling.**direct_coupling_matrix**(*vacuum_wavelength*, *particle_list*, *layer_system*)
Return the direct particle coupling matrix W for a particle collection in a layered medium.

Parameters

- **vacuum_wavelength** (*float*) – Wavelength in length unit

- **(list of smuthi.particles.Particle obejcts** (*particle_list*) – Scattering particles
- **layer_system** (smuthi.layers.LayerSystem) – The stratified medium

Returns Ensemble coupling matrix as numpy array.

smuthi.particle_coupling.**layer_mediated_coupling_block**(*vacuum_wavelength*, *receiving_particle*, *emitting_particle*, *layer_system*, *k_parallel='default'*, *show_integrand=False*)

Layer-system mediated particle coupling matrix W^R for two particles. This routine is explicit, but slow.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength λ (length unit)
- **receiving_particle** (smuthi.particles.Particle) – Particle that receives the scattered field
- **emitting_particle** (smuthi.particles.Particle) – Particle that emits the scattered field
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium in which the coupling takes place
- **k_parallel** (*numpy ndarray*) – In-plane wavenumbers for Sommerfeld integral If 'default', use smuthi.coordinates.default_k_parallel
- **show_integrand** (*bool*) – If True, the norm of the integrand is plotted.

Returns Layer mediated coupling matrix block as numpy array.

smuthi.particle_coupling.**layer_mediated_coupling_matrix**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*)

Layer system mediated particle coupling matrix W^R for a particle collection in a layered medium.

Parameters

- **vacuum_wavelength** (*float*) – Wavelength in length unit
- **(list of smuthi.particles.Particle obejcts** (*particle_list*) – Scattering particles
- **layer_system** (smuthi.layers.LayerSystem) – The stratified medium
- **k_parallel** (*numpy.ndarray or str*) – In-plane wavenumber for Sommerfeld integrals. If 'default', smuthi.coordinates.default_k_parallel

Returns Ensemble coupling matrix as numpy array.

smuthi.particle_coupling.**radial_coupling_lookup_table**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*, *enable_cuda=False*)

Prepare Sommerfeld integral lookup table to allow for a fast calculation of the coupling matrix by interpolation. This function is called when all particles are on the same z-position.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength in length units

- **particle_list** (*list*) – List of particle objects
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium
- **k_parallel** (*numpy.ndarray or str*) – In-plane wavenumber for Sommerfeld integrals. If 'default', smuthi.coordinates.default_k_parallel
- **resolution** (*float*) – Spatial resolution of lookup table in length units. (default: vacuum_wavelength / 100) Smaller means more accurate but higher memory footprint

Returns

lookup_table (ndarray): Coupling lookup, indices are [rho, n1, n2]. rho_array (ndarray): Values for the radial distance considered for the lookup (starting from negative

numbers to allow for simpler cubic interpolation without distinction of cases at rho=0)

Return type (tuple) tuple containing

smuthi.particle_coupling.**size_format**(*b*)

smuthi.particle_coupling.**volumetric_coupling_lookup_table**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *resolution=None*)

Prepare Sommerfeld integral lookup table to allow for a fast calculation of the coupling matrix by interpolation. This function is called when not all particles are on the same z-position.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength in length units
- **particle_list** (*list*) – List of particle objects
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium
- **k_parallel** (*numpy.ndarray or str*) – In-plane wavenumber for Sommerfeld integrals. If 'default', smuthi.coordinates.default_k_parallel
- **resolution** (*float*) – Spatial resolution of lookup table in length units. (default: vacuum_wavelength / 100) Smaller means more accurate but higher memory footprint

Returns

tuple containing:

w_pl (ndarray): Coupling lookup for z1 + z2, indices are [rho, z, n1, n2]. Includes layer mediated coupling. w_mn (ndarray): Coupling lookup for z1 + z2, indices are [rho, z, n1, n2]. Includes layer mediated and

direct coupling.

rho_array (ndarray): Values for the radial distance considered for the lookup (starting from negative numbers to allow for simpler cubic interpolation without distinction of cases for lookup edges

sz_array (ndarray): Values for the sum of z-coordinates (z1 + z2) considered for the lookup dz_array (ndarray): Values for the difference of z-coordinates (z1 - z2) considered for the lookup

Return type (tuple)

5.10 smuthi.particles module

Provide class for the representation of scattering particles.

class smuthi.particles.**FiniteCylinder**(*position=None*, *euler_angles=None*, *refractive_index=(1+0j)*, *cylinder_radius=1*, *cylinder_height=1*, *l_max=None*, *m_max=None*, *t_matrix_method=None*)

Particle subclass for finite cylinders.

Parameters

- **position** (*list*) – Particle position in the format [x, y, z] (length unit)
- **refractive_index** (*complex*) – Complex refractive index of particle
- **cylinder_radius** (*float*) – Radius of cylinder (length unit)
- **cylinder_height** (*float*) – Height of cylinder, in z-direction if not rotated (length unit)
- **l_max** (*int*) – Maximal multipole degree used for the spherical wave expansion of incoming and scattered field
- **m_max** (*int*) – Maximal multipole order used for the spherical wave expansion of incoming and scattered field

class smuthi.particles.**Particle**(*position=None*, *euler_angles=None*, *refractive_index=(1+0j)*, *l_max=None*, *m_max=None*)

Base class for scattering particles.

Parameters

- **position** (*list*) – Particle position in the format [x, y, z] (length unit)
- **euler_angles** (*list*) – Particle Euler angles in the format [alpha, beta, gamma]
- **refractive_index** (*complex*) – Complex refractive index of particle
- **l_max** (*int*) – Maximal multipole degree used for the spherical wave expansion of incoming and scattered field
- **m_max** (*int*) – Maximal multipole order used for the spherical wave expansion of incoming and scattered field

class smuthi.particles.**Sphere**(*position=None*, *refractive_index=(1+0j)*, *radius=1*, *l_max=None*, *m_max=None*)

Particle subclass for spheres.

Parameters

- **position** (*list*) – Particle position in the format [x, y, z] (length unit)
- **refractive_index** (*complex*) – Complex refractive index of particle
- **radius** (*float*) – Particle radius (length unit)
- **l_max** (*int*) – Maximal multipole degree used for the spherical wave expansion of incoming and scattered field
- **m_max** (*int*) – Maximal multipole order used for the spherical wave expansion of incoming and scattered field
- **t_matrix_method** (*dict*) – Dictionary containing the parameters for the algorithm to compute the T-matrix

class smuthi.particles.**Spheroid**(*position=None*, *euler_angles=None*, *refractive_index=(1+0j)*, *semi_axis_c=1*, *semi_axis_a=1*, *l_max=None*, *m_max=None*, *t_matrix_method=None*)

Particle subclass for spheroids.

Parameters

- **position** (*list*) – Particle position in the format [x, y, z] (length unit)
- **refractive_index** (*complex*) – Complex refractive index of particle
- **semi_axis_c** (*float*) – Spheroid half axis in direction of axis of revolution (z-axis if not rotated)
- **semi_axis_a** (*float*) – Spheroid half axis in lateral direction (x- and y-axis if not rotated)
- **l_max** (*int*) – Maximal multipole degree used for the spherical wave expansion of incoming and scattered field
- **m_max** (*int*) – Maximal multipole order used for the spherical wave expansion of incoming and scattered field
- **t_matrix_method** (*dict*) – Dictionary containing the parameters for the algorithm to compute the T-matrix

5.11 smuthi.post_processing module

Manage all post processing tasks after solving the linear system.

class smuthi.post_processing.**PostProcessing**

run(*simulation*)

Run tasks for post processing.

Parameters **simulation** (smuthi.simulation.Simulation) – simulation object containing input and solution of the problem

smuthi.post_processing.**evaluate_cross_section**(*polar_angles='default'*, *azimuthal_angles='default'*, *initial_field=None*, *particle_list=None*, *layer_system=None*, *outputdir=None*, *show_plots=None*, *save_plots=None*, *save_data=None*, *length_unit=None*)

Compute differential scattering cross section as well as extinction cross sections.

Parameters

- **polar_angles** (*numpy.ndarray or str*) – array of polar angles for differential cross section. if 'default', use smuthi.coordinates.default_polar_angles
- **azimuthal_angles** (*numpy.ndarray or str*) – array of azimuthal angles for differential cross section if 'default', use smuthi.coordinates.default_azimuthal_angles
- **initial_field** (*smuthi.initial.InitialField*) – initial field object
- **particle_list** (*list*) – list of smuthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **outputdir** (*str*) – path to folder where to store data and figures

- **show_plots** (*bool*) – show plots if true
- **save_plots** (*bool*) – export plots to disc if true
- **save_data** (*bool*) – save raw data to disc if true
- **length_unit** (*str*) – length unit used in simulation, e.g. 'nm'

Returns A tuple of a scattering cross section object and an extinction cross section dictionary.

5.12 smuthi.read_input module

smuthi.read_input.**read_input_yaml**(*filename*)

Parse input file

Parameters **filename** (*str*) – relative path and filename of input file

Returns smuthi.simulation.Simulation object containing the params of the input file

5.13 smuthi.scattered_field module

Manage post processing steps to evaluate the scattered near and far field

smuthi.scattered_field.**extinction_cross_section**(*initial_field*, *particle_list*, *layer_system*)

Evaluate and display the differential scattering cross section as a function of solid angle.

Parameters

- **initial_field** (smuthi.initial_field.PlaneWave) – Plane wave object
- **particle_list** (*list*) – List of smuthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – Representing the stratified medium

Returns

Dictionary with following entries

- 'forward': Extinction in the positinve z-direction (top layer)
- 'backward': Extinction in the negative z-direction (bottom layer)

smuthi.scattered_field.**scattered_far_field**(*vacuum_wavelength*, *particle_list*, *layer_system*, *polar_angles='default'*, *azimuthal_angles='default'*)

Evaluate the scattered far field.

Parameters

- **vacuum_wavelength** (*float*) – in length units
- **particle_list** (*list*) – list of smuthi.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – represents the stratified medium
- **polar_angles** (*numpy.ndarray or str*) – polar angles values (radian). if 'default', use smuthi.coordinates.default_polar_angles

- **azimuthal_angles** (*numpy.ndarray or str*) – azimuthal angle values (radian) if 'default', use smuthi.coordinates.default_azimuthal_angles

Returns A FarField object of the scattered field.

smuthi.scattered_field.**scattered_field_piecewise_expansion**(*vacuum_wavelength*, *particle_list*, *layer_system*, *k_parallel='default'*, *azimuthal_angles='default'*)

Compute a piecewise field expansion of the scattered field.

Parameters

- **vacuum_wavelength** (*float*) – vacuum wavelength
- **particle_list** (*list*) – list of smuthi.particles.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumbers array. if 'default', use smuthi.coordinates.default_k_parallel
- **azimuthal_angles** (*numpy.ndarray or str*) – azimuthal angles array if 'default', use smuthi.coordinates.default_azimuthal_angles

Returns scattered field as smuthi.field_expansion.PiecewiseFieldExpansion object

smuthi.scattered_field.**scattered_field_pwe**(*vacuum_wavelength*, *particle_list*, *layer_system*, *layer_number*, *k_parallel='default'*, *azimuthal_angles='default'*, *include_direct=True*, *include_layer_response=True*)

Calculate the plane wave expansion of the scattered field of a set of particles.

Parameters

- **vacuum_wavelength** (*float*) – Vacuum wavelength (length unit)
- **particle_list** (*list*) – List of Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – Stratified medium
- **layer_number** (*int*) – Layer number in which the plane wave expansion should be valid
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumbers array. if 'default', use smuthi.coordinates.default_k_parallel
- **azimuthal_angles** (*numpy.ndarray or str*) – azimuthal angles array if 'default', use smuthi.coordinates.default_azimuthal_angles
- **include_direct** (*bool*) – If True, include the direct scattered field
- **include_layer_response** (*bool*) – If True, include the layer system response

Returns A tuple of PlaneWaveExpansion objects for upgoing and downgoing waves.

smuthi.scattered_field.**scattering_cross_section**(*initial_field*, *particle_list*, *layer_system*, *polar_angles='default'*, *azimuthal_angles='default'*)

Evaluate and display the differential scattering cross section as a function of solid angle.

Parameters

- **initial_field** (*smuthi.initial.PlaneWave*) – Initial Plane wave
- **particle_list** (*list*) – scattering particles
- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **polar_angles** (*numpy.ndarray or str*) – polar angles values (radian). if 'default', use smuthi.coordinates.default_polar_angles
- **azimuthal_angles** (*numpy.ndarray or str*) – azimuthal angle values (radian) if 'default', use smuthi.coordinates.default_azimuthal_angles

Returns A tuple of FarField objects, one for forward scattering (i.e., into the top hemisphere) and one for backward scattering (bottom hemisphere).

smuthi.scattered_field.**total_far_field**(*initial_field*, *particle_list*, *layer_system*, *polar_angles='default'*, *azimuthal_angles='default'*)

Evaluate the total far field, the initial far field and the scattered far field. Cannot be used if initial field is a plane wave.

Parameters

- **initial_field** (smuthi.initial_field.InitialField) – represents the initial field
- **particle_list** (*list*) – list of smuthi.Particle objects
- **layer_system** (smuthi.layers.LayerSystem) – represents the stratified medium
- **polar_angles** (*numpy.ndarray or str*) – polar angles values (radian). if 'default', use smuthi.coordinates.default_polar_angles
- **azimuthal_angles** (*numpy.ndarray or str*) – azimuthal angle values (radian) if 'default', use smuthi.coordinates.default_azimuthal_angles

Returns A tuple of three FarField objects for total, initial and scattered far field. Mind that the scattered far field has no physical meaning and is for illustration purposes only.

5.14 smuthi.simulation module

Provide class to manage a simulation.

class smuthi.simulation.**Logger**(*log_filename*)

Allows to prompt messages both to terminal and to log file simultaneously.

flush()

write(*message*)

class smuthi.simulation.**Simulation**(*layer_system=None*, *particle_list=None*, *initial_field=None*, *post_processing=None*, *k_parallel='default'*, *solver_type='LU'*, *solver_tolerance=0.0001*, *store_coupling_matrix=True*, *coupling_matrix_lookup_resolution=None*, *coupling_matrix_interpolator_kind='linear'*, *length_unit='length unit'*, *input_file=None*, *output_dir='smuthi_output'*, *save_after_run=False*)

Central class to manage a simulation.

Parameters

- **layer_system** (smuthi.layers.LayerSystem) – stratified medium
- **particle_list** (*list*) – list of smuthi.particles.Particle objects
- **initial_field** (smuthi.initial_field.InitialField) – initial field object
- **post_processing** (smuthi.post_processing.PostProcessing) – object managing post processing tasks
- **k_parallel** (*numpy.ndarray or str*) – in-plane wavenumber for Sommerfeld integrals. if 'default', keep what is in smuthi.coordinates.default_k_parallel
- **solver_type** (*str*) – What solver type to use? Options: 'LU' for LU factorization, 'gmres' for GMRES iterative solver
- **coupling_matrix_lookup_resolution** (*float or None*) – If type float, compute particle coupling by interpolation of a lookup table with that spacial resolution. If None (default), don't use a lookup table but compute the coupling directly. This is more suitable for a small particle number.
- **coupling_matrix_interpolator_kind** (*str*) – Set to 'linear' (default) or 'cubic' interpolation of the lookup table.
- **store_coupling_matrix** (*bool*) – If True (default), the coupling matrix is stored. Otherwise it is recomputed on the fly during each iteration of the solver.
- **length_unit** (*str*) – what is the physical length unit? has no influence on the computations
- **input_file** (*str*) – path and filename of input file (for logging purposes)
- **output_dir** (*str*) – path to folder where to export data
- **save_after_run** (*bool*) – if true, the simulation object is exported to disc when over

print_simulation_header()

run()
Start the simulation.

save(*filename=None*)
Export simulation object to disc.

Parameters **filename** (*str*) – path and file name where to store data

5.15 smuthi.spherical_functions module

smuthi.spherical_functions.**dx_xh**(*n*, *x*)
Derivative of $xh_n(x)$, where $h_n(x)$ is the spherical Hankel function.

Parameters

- **n** (*int*) – (n>0): Order of spherical Bessel function
- **x** (*array, complex or float*) – Argument for spherical Hankel function

Returns Derivative $\partial_x(xh_n(x))$ as array.

smuthi.spherical_functions.**dx_xj**(*n*, *x*)
Derivative of $xj_n(x)$, where $j_n(x)$ is the spherical Bessel function.

Parameters

- **n** (*int*) – (n>0): Order of spherical Bessel function

- **x** (*array, complex or float*) – Argument for spherical Bessel function

Returns Derivative $\partial_x(xj_n(x))$ as array.

smuthi.spherical_functions.**legendre_normalized**(*ct*, *st*, *lmax*)
Return the normalized associated Legendre function $P_l^m(\cos\theta)$ and the angular functions $\pi_l^m(\cos\theta)$ and $\tau_l^m(\cos\theta)$, as defined in A. Doicu, T. Wriedt, and Y. A. Eremin: "Light Scattering by Systems of Particles", Springer-Verlag, 2006. Two arguments (ct and st) are passed such that the function is valid for general complex arguments, while the branch cuts are defined by the user already in the definition of st.

Parameters

- **ct** (*ndarray*) – cosine of theta (or kz/k)
- **st** (*ndarray*) – sine of theta (or kp/k), need to have same dimension as ct, and st**2+ct**2=1 is assumed
- **lmax** (*int*) – maximal multipole order

Returns

- list plm[l][m] contains $P_l^m(\cos\theta)$. The entries of the list have same dimension as ct (and st)
- list pilm[l][m] contains $\pi_l^m(\cos\theta)$.
- list taulm[l][m] contains $\tau_l^m(\cos\theta)$.

smuthi.spherical_functions.**spherical_hankel**(*n*, *x*)

5.16 smuthi.t_matrix module

smuthi.t_matrix.**mie_coefficient**(*tau*, *l*, *k_medium*, *k_particle*, *radius*)
Return the Mie coefficients of a sphere.

Parameters

- **integer** (*l*) – spherical polarization, 0 for spherical TE and 1 for spherical TM
- **integer** – l=1,... multipole degree (polar quantum number)
- **floator complex** (*k_particle*) – wavenumber in surrounding medium (inverse length unit)
- **floator complex** – wavenumber inside sphere (inverse length unit)
- **float** (*radius*) – radius of sphere (length unit)

Returns Mie coefficients as complex

smuthi.t_matrix.**rotate_t_matrix**(*t*, *euler_angles*)
Placeholder for a proper T-matrix rotation routine

smuthi.t_matrix.**t_matrix**(*vacuum_wavelength*, *n_medium*, *particle*)
Return the T-matrix of a particle.

Parameters

- **vacuum_wavelength** (*float*) –
- **n_medium** (*float or complex*) – Refractive index of surrounding medium
- **particle** (smuthi.particles.Particle) – Particle object

Returns T-matrix as ndarray

smuthi.t_matrix.**t_matrix_sphere**(*k_medium*, *k_particle*, *radius*, *l_max*, *m_max*)

T-matrix of a spherical scattering object.

Parameters

- **k_medium** (*float or complex*) – Wavenumber in surrounding medium (inverse length unit)
- **k_particle** (*float or complex*) – Wavenumber inside sphere (inverse length unit)
- **radius** (*float*) – Radius of sphere (length unit)
- **l_max** (*int*) – Maximal multipole degree
- **m_max** (*int*) – Maximal multipole order
- **blocksize** (*int*) – Total number of index combinations
- **multi_to_single_index_map** (*function*) – A function that maps the SVWF indices (tau, l, m) to a single index

Returns T-matrix as ndarray

5.17 smuthi.vector_wave_functions module

smuthi.vector_wave_functions.**plane_vector_wave_function**(*x*, *y*, *z*, *kp*, *alpha*, *kz*, *pol*)

Electric field components of plane wave (PVWF).

$$\mathbf{\Phi}_j = \exp(\mathrm{i}\mathbf{k}\cdot\mathbf{r})\hat{\mathbf{e}}_j$$

with $\hat{\mathbf{e}}_0$ denoting the unit vector in azimuthal direction ('TE' or 's' polarization), and $\hat{\mathbf{e}}_1$ denoting the unit vector in polar direction ('TM' or 'p' polarization).

The input arrays should have one of the following dimensions:

- x,y,z: (N x 1) matrix
- kp,alpha,kz: (1 x M) matrix
- Ex, Ey, Ez: (M x N) matrix

or

- x,y,z: (M x N) matrix
- kp,alpha,kz: scalar
- Ex, Ey, Ez: (M x N) matrix

Parameters

- **x** (*numpy.ndarray*) – x-coordinate of position where to test the field (length unit)
- **y** (*numpy.ndarray*) – y-coordinate of position where to test the field
- **z** (*numpy.ndarray*) – z-coordinate of position where to test the field
- **kp** (*numpy.ndarray*) – parallel component of k-vector (inverse length unit)
- **alpha** (*numpy.ndarray*) – azimthal angle of k-vector (rad)
- **kz** (*numpy.ndarray*) – z-component of k-vector (inverse length unit)
- **pol** (*int*) – Polarization (0=TE, 1=TM)

Returns

- x-coordinate of PVWF electric field (numpy.ndarray)
- y-coordinate of PVWF electric field (numpy.ndarray)
- z-coordinate of PVWF electric field (numpy.ndarray)

smuthi.vector_wave_functions.**spherical_vector_wave_function**(*x*, *y*, *z*, *k*, *nu*, *tau*, *l*, *m*)

Electric field components of spherical vector wave function (SVWF). The conventions are chosen according to A. Doicu, T. Wriedt, and Y. A. Eremin: "Light Scattering by Systems of Particles", Springer-Verlag, 2006

Parameters

- **x** (*numpy.ndarray*) – x-coordinate of position where to test the field (length unit)
- **y** (*numpy.ndarray*) – y-coordinate of position where to test the field
- **z** (*numpy.ndarray*) – z-coordinate of position where to test the field
- **k** (*float or complex*) – wavenumber (inverse length unit)
- **nu** (*int*) – 1 for regular waves, 3 for outgoing waves
- **tau** (*int*) – spherical polarization, 0 for spherical TE and 1 for spherical TM
- **l** (*int*) – l=1,... multipole degree (polar quantum number)
- **m** (*int*) – m=-l,...,l multipole order (azimuthal quantum number)

Returns

- x-coordinate of SVWF electric field (numpy.ndarray)
- y-coordinate of SVWF electric field (numpy.ndarray)
- z-coordinate of SVWF electric field (numpy.ndarray)

smuthi.vector_wave_functions.**transformation_coefficients_vwf**(*tau*, *l*, *m*, *pol*, *kp=None*, *kz=None*, *pilm_list=None*, *taulm_list=None*, *dagger=False*)

Transformation coefficients B to expand SVWF in PVWF and vice versa:

$$B_{\tau lm,j}(x) = -\frac{1}{\mathrm{i}^{l+1}}\frac{1}{\sqrt{2l(l+1)}}(\mathrm{i}\delta_{j1}+\delta_{j2})(\delta_{\tau j}\tau_l^{|m|}(x)+(1-\delta_{\tau j}m\pi_l^{|m|}(x))$$

For the definition of the τ_l^m and π_l^m functions, see A. Doicu, T. Wriedt, and Y. A. Eremin: "Light Scattering by Systems of Particles", Springer-Verlag, 2006

Parameters

- **tau** (*int*) – SVWF polarization, 0 for spherical TE, 1 for spherical TM
- **l** (*int*) – l=1,... SVWF multipole degree
- **m** (*int*) – m=-l,...,l SVWF multipole order
- **pol** (*int*) – PVWF polarization, 0 for TE, 1 for TM
- **kp** (*numpy array*) – PVWF in-plane wavenumbers
- **kz** (*numpy array*) – complex numpy-array: PVWF out-of-plane wavenumbers

- **pilm_list** (*list*) – 2D list numpy-arrays: alternatively to kp and kz, pilm and taulm as generated with legendre_normalized can directly be handed
- **taulm_list** (*list*) – 2D list numpy-arrays: alternatively to kp and kz, pilm and taulm as generated with legendre_normalized can directly be handed
- **dagger** (*bool*) – switch on when expanding PVWF in SVWF and off when expanding SVWF in PVWF

Returns Transformation coefficient as array (size like kp).

smuthi.vector_wave_functions.**translation_coefficients_svwf**(*tau1*, *l1*, *m1*, *tau2*, *l2*, *m2*, *k*, *d*, *sph_hankel=None*, *legendre=None*, *exp_immphi=None*)

Coefficients of the translation operator for the expansion of an outgoing spherical wave in terms of regular spherical waves with respect to a different origin:

$$\mathbf{\Psi}^{(3)}_{\tau lm}(\mathbf{r}+\mathbf{d} = \sum_{\tau'}\sum_{l'}\sum_{m'} A_{\tau lm,\tau' l' m'}(\mathbf{d})\mathbf{\Psi}^{(1)}_{\tau' l' m'}(\mathbf{r})$$

for $|\mathbf{r}| < |\mathbf{d}|$.

Parameters

- **tau1** (*int*) – tau1=0,1: Original wave's spherical polarization
- **l1** (*int*) – l=1,...: Original wave's SVWF multipole degree
- **m1** (*int*) – m=-l,...,l: Original wave's SVWF multipole order
- **tau2** (*int*) – tau2=0,1: Partial wave's spherical polarization
- **l2** (*int*) – l=1,...: Partial wave's SVWF multipole degree
- **m2** (*int*) – m=-l,...,l: Partial wave's SVWF multipole order
- **k** (*float or complex*) – wavenumber (inverse length unit)
- **d** (*list*) – translation vectors in format [dx, dy, dz] (length unit) dx, dy, dz can be scalars or ndarrays
- **sph_hankel** (*list*) – Optional. sph_hankel[i] contains the spherical hankel funciton of degree i, evaluated at k*d where d is the norm of the distance vector(s)
- **legendre** (*list*) – Optional. legendre[l][m] contains the legendre function of order l and degree m, evaluated at cos(theta) where theta is the polar angle(s) of the distance vector(s)

Returns translation coefficient A (complex)

smuthi.vector_wave_functions.**translation_coefficients_svwf_out_to_out**(*tau1*, *l1*, *m1*, *tau2*, *l2*, *m2*, *k*, *d*, *sph_bessel=None*, *legendre=None*, *exp_immphi=None*)

Coefficients of the translation operator for the expansion of an outgoing spherical wave in terms of outgoing

spherical waves with respect to a different origin:

$$\mathbf{\Psi}^{(3)}_{\tau lm}(\mathbf{r}+\mathbf{d} = \sum_{\tau'}\sum_{l'}\sum_{m'} A_{\tau lm,\tau' l' m'}(\mathbf{d})\mathbf{\Psi}^{(3)}_{\tau' l' m'}(\mathbf{r})$$

for $|\mathbf{r}| > |\mathbf{d}|$.

Parameters

- **tau1** (*int*) – tau1=0,1: Original wave's spherical polarization
- **l1** (*int*) – l=1,...: Original wave's SVWF multipole degree
- **m1** (*int*) – m=-l,...,l: Original wave's SVWF multipole order
- **tau2** (*int*) – tau2=0,1: Partial wave's spherical polarization
- **l2** (*int*) – l=1,...: Partial wave's SVWF multipole degree
- **m2** (*int*) – m=-l,...,l: Partial wave's SVWF multipole order
- **k** (*float or complex*) – wavenumber (inverse length unit)
- **d** (*list*) – translation vectors in format [dx, dy, dz] (length unit) dx, dy, dz can be scalars or ndarrays
- **sph_bessel** (*list*) – Optional. sph_bessel[i] contains the spherical Bessel funciton of degree i, evaluated at k*d where d is the norm of the distance vector(s)
- **legendre** (*list*) – Optional. legendre[l][m] contains the legendre function of order l and degree m, evaluated at cos(theta) where theta is the polar angle(s) of the distance vector(s)

Returns translation coefficient A (complex)

5.18 smuthi.nfmds package

Check if NFM-DS is installed and otherwise do so.

smuthi.nfmds.**install_nfmds**()

5.19 smuthi.nfmds.t_matrix_axsym module

Functions to call NFM-DS (null field method with discrete sources) Fortran code by Doicu et al. for the generation of T-matrices for non-spherical particles. The Fortran code comes with the book A. Doicu, T. Wriedt, and Y. A. Eremin: Light Scattering by Systems of Particles, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2006. and can also be downloaded from https://scattport.org/index.php/programs-menu/t-matrix-codes-menu/239-nfm-ds

smuthi.nfmds.t_matrix_axsym.**taxsym_read_tmatrix**(*filename*, *l_max*, *m_max*)

Export TAXSYM.f90 output to SMUTHI T-matrix.

Parameters

- **filename** (*str*) – Name of the file containing the T-matrix output of TAXSYM.f90
- **l_max** (*int*) – Maximal multipole degree
- **m_max** (*int*) – Maximal multipole order

Returns T-matrix as numpy.ndarray

smuthi.nfmds.t_matrix_axsym.**taxsym_run**()
Call TAXSYM.f90 routine.

smuthi.nfmds.t_matrix_axsym.**taxsym_write_input_cylinder**(*vacuum_wavelength=None*, *layer_refractive_index=None*, *particle_refractive_index=None*, *cylinder_height=None*, *cylinder_radius=None*, *use_ds=True*, *nint=None*, *nrank=None*, *filename='T_matrix_cylinder.dat'*)
Generate input file for the TAXSYM.f90 routine for the simulation of a finite cylinder.

Parameters

- **vacuum_wavelength** (*float*) –
- **layer_refractive_index** (*float*) – Real refractive index of layer (complex values are not allowed)
- **particle_refractive_index** (*float or complex*) – Complex refractive index of cylinder
- **cylinder_height** (*float*) – Height of cylinder (length unit)
- **cylinder_radius** (*float*) – Radius of cylinder (length unit)
- **use_ds** (*bool*) – Flag to switch the use of discrete sources on (True) and off (False)
- **nint** (*int*) – Nint parameter for internal use of NFM-DS (number of points along integral). Higher value is more accurate and takes longer
- **nrank** (*int*) – l_max used internally in NFM-DS
- **filename** (*str*) – Name of the file in which the T-matrix is stored

smuthi.nfmds.t_matrix_axsym.**taxsym_write_input_spheroid**(*vacuum_wavelength=None*, *layer_refractive_index=None*, *particle_refractive_index=None*, *semi_axis_c=None*, *semi_axis_a=None*, *use_ds=True*, *nint=None*, *nrank=None*, *filename='T_matrix_spheroid.dat'*)
Generate input file for the TAXSYM.f90 routine for the simulation of a spheroid.

Parameters

- **vacuum_wavelength** (*float*) –
- **layer_refractive_index** (*float*) – Real refractive index of layer (complex values are not allowed)
- **particle_refractive_index** (*float or complex*) – Complex refractive index of spheroid
- **semi_axis_c** (*float*) – Semi axis of spheroid along rotation axis
- **semi_axis_a** (*float*) – Semi axis of spheroid along lateral direction
- **use_ds** (*bool*) – Flag to switch the use of discrete sources on (True) and off (False)

- **nint** (*int*) – Nint parameter for internal use of NFM-DS (number of points along integral). Higher value is more accurate and takes longer
- **nrank** (*int*) – l_max used internally in NFM-DS
- **filename** (*str*) – Name of the file in which the T-matrix is stored

smuthi.nfmds.t_matrix_axsym.**tmatrix_cylinder**(*vacuum_wavelength=None*, *layer_refractive_index=None*, *particle_refractive_index=None*, *cylinder_height=None*, *cylinder_radius=None*, *l_max=None*, *m_max=None*, *use_ds=True*, *nint=None*, *nrank=None*)

Return T-matrix for finite cylinder, using the TAXSYM.f90 routine from the NFM-DS.

Parameters

- **vacuum_wavelength** (*float*) –
- **layer_refractive_index** (*float*) – Real refractive index of layer (complex values are not allowed).
- **particle_refractive_index** (*float or complex*) – Complex refractive index of spheroid
- **cylinder_height** (*float*) – Semi axis of spheroid along rotation axis
- **cylinder_radius** (*float*) – Semi axis of spheroid along lateral direction
- **l_max** (*int*) – Maximal multipole degree
- **m_max** (*int*) – Maximal multipole order
- **use_ds** (*bool*) – Flag to switch the use of discrete sources on (True) and off (False)
- **nint** (*int*) – Nint parameter for internal use of NFM-DS (number of points along integral). Higher value is more accurate and takes longer
- **nrank** (*int*) – l_max used internally in NFM-DS

Returns T-matrix as numpy.ndarray

smuthi.nfmds.t_matrix_axsym.**tmatrix_spheroid**(*vacuum_wavelength=None*, *layer_refractive_index=None*, *particle_refractive_index=None*, *semi_axis_c=None*, *semi_axis_a=None*, *l_max=None*, *m_max=None*, *use_ds=True*, *nint=None*, *nrank=None*)

T-matrix for spheroid, using the TAXSYM.f90 routine from the NFM-DS.

Parameters

- **vacuum_wavelength** (*float*) –
- **layer_refractive_index** (*float*) – Real refractive index of layer (complex values are not allowed).
- **particle_refractive_index** (*float or complex*) – Complex refractive index of spheroid
- **semi_axis_c** (*float*) – Semi axis of spheroid along rotation axis
- **semi_axis_a** (*float*) – Semi axis of spheroid along lateral direction
- **l_max** (*int*) – Maximal multipole degree

- **m_max** (*int*) – Maximal multipole order
- **use_ds** (*bool*) – Flag to switch the use of discrete sources on (True) and off (False)
- **nint** (*int*) – Nint parameter for internal use of NFM-DS (number of points along integral). Higher value is more accurate and takes longer
- **nrank** (*int*) – l_max used internally in NFM-DS

Returns T-matrix as numpy.ndarray

Python Module Index

S

Index

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W