SMUTHI Documentation

Release 0.4

Amos Egel

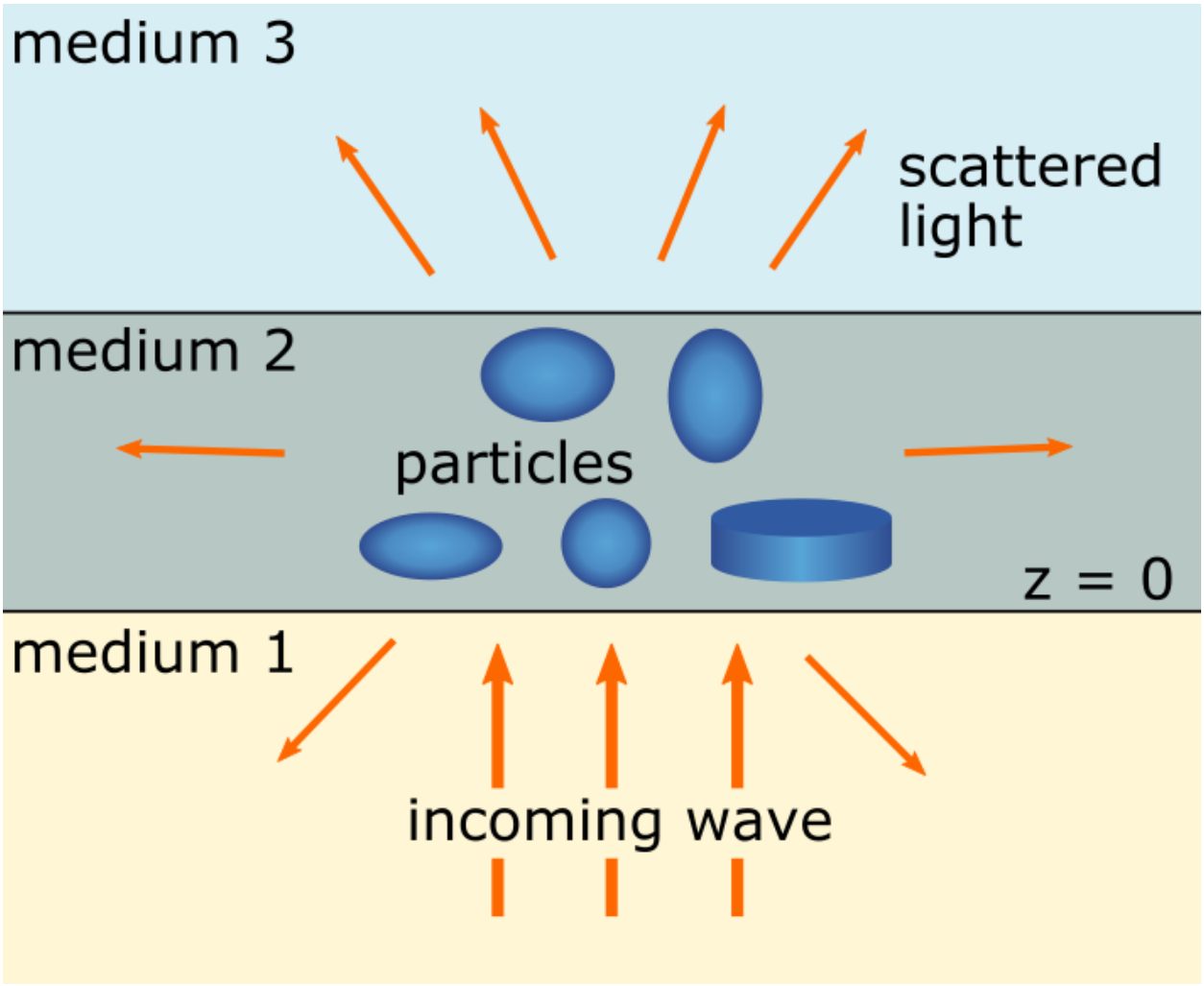
Aug 08, 2017

Contents

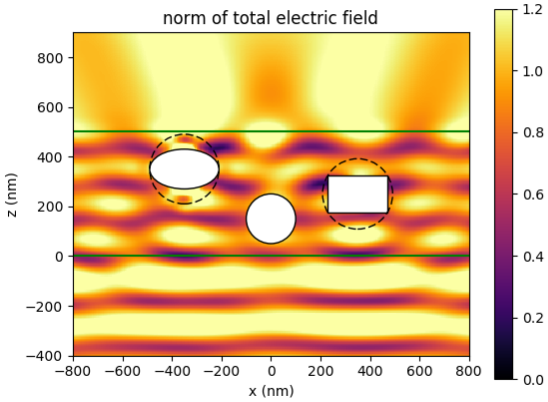
CHAPTER 1

What is it?

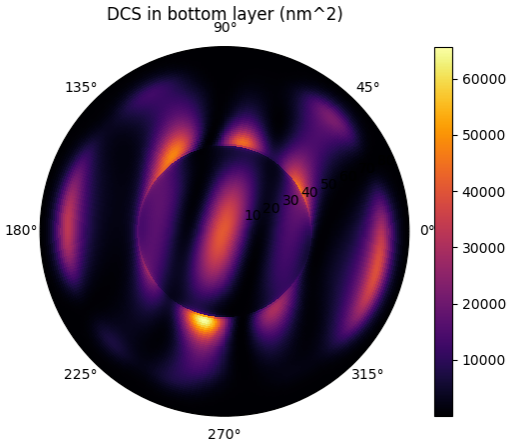
SMUTHI stands for ‘scattering by multiple particles in thin-film systems’. The software allows you to solve light scattering problems involving one ore multiple particles near or inside a system of planar layer interfaces. It is based on the T-matrix method for the single particle scattering, and on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

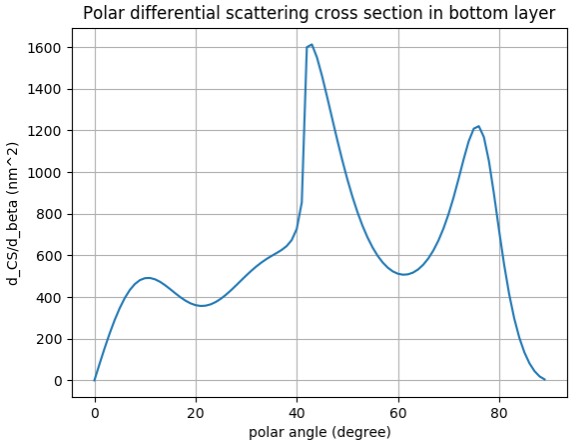


The software solves Maxwell's equations (3D wave optics) in frequency domain (one wavelength per simulation). An arbitrary number of spheres, spheroids and finite cylinders inside an arbitrary system of plane parallel layers can be modelled. For spheres, the T-matrix is given by the Mie-coefficients. 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".



With Smuthi, 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 addition, the far field power flux can be evaluated. For plane wave incidence, it 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 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 shown 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

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.

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
```

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
```

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.

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.
```

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*.

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

Parameters specified in the input file

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

Length unit

Declare here the unit in which you want to specify all lengths. It 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:

```
length unit: nm
```

Vacuum wavelength

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

```
vacuum wavelength: 550
```

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 $n_2 = 1$).

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 `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
  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
  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*.

Initial field

Currently, only plane waves are implemented as the initial excitation.

Specify the initial field in the following format:

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

Angle units can be 'degree' (otherwise, radians are used). 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.

Numerical parameters

Specify the contour of the sommerfeld integral in the complex `neff` plane where `neff = k_parallel / omega` refers to the effective refractive index of the partial wave. The contour is parameterized by its waypoints:

```
neff waypoints: [0, 0.5, 0.8-0.1j, 2-0.1j, 2.5, 4]
```

as well as its discretization scale:

```
neff discretization: 1e-3
```

The `neff waypoints` define a piecewise linear trajectory in the complex plane. This trajectory should start at `0` and end at a suitable real truncation parameter (somewhere above the highest layer refractive index). A simple contour would be for example `neff waypoints: [0, 4]`. However The trajectory can be deflected into the lower complex half plaen such that it does not come close to waveguide mode resonances of the layer system.

Post procesing

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

- evaluation of scattering and extinction cross sections
- evaluation of the electrical near field

Write for example:

```
post processing:
- task: evaluate cross sections
  show plots: false
  save plots: true
  save data: false
- task: evaluate near field
  show plots: false
  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 is 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.

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.

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.

smuthi.coordinates module

class smuthi.coordinates.**ComplexContour**(*neff_waypoints=[0, 1], neff_discretization=0.01*)

Trajectory of $n_{\mathrm{eff}} = \kappa/\omega$ in the complex plane for the evaluation of Sommerfeld integrals.

Parameters

- **neff_waypoints** – List of complex n_{eff} waypoints, that is, points through which the contour goes (linear between them).
- **neff_discretization** – Distance between adjacent n_{eff} values in the contour. Either as a list of floats (for different discretization in different linear segments) or as a float (uniform discretization for all segments)

neff()

Returns numpy-array of n_{eff} values that define the contour

smuthi.coordinates.**angular_frequency**(*vacuum_wavelength*)

Angular frequency $\omega = 2\pi c/\lambda$

Parameters **vacuum_wavelength** – 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.**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** – In-plane wavenumber κ (inverse length)

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

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

smuthi.far_field module

smuthi.field_expansion module

class smuthi.field_expansion.**PlaneWaveExpansion**(*k*, *k_parallel=None*, *azimuthal_angles=None*, *type=None*, *reference_point=None*, *valid_between=None*)

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 $\mathrm{d}^2\mathbf{k}_\parallel = \kappa \, \mathrm{d}\alpha \, \mathrm{d}\kappa$ and the double integral runs over $\alpha \in [0, 2\pi]$ and $\kappa \in [0, \kappa_{\mathrm{max}}]$. 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 list of 4-dimensional arrays. If the attributes k_parallel and azimuthal_angles have only a single entry, a discrete distribution is assumed:

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

Parameters

- **n_effective** (*ndarray*) – $n_{\mathrm{eff}} = \kappa/\omega$, can be float or complex numpy.array
- **azimuthal_angles** (*ndarray*) – α, from 0 to 2π
- **layer_system** (*smuthi.layers.LayerSystem*) – Layer system in which the field is expanded

k_parallel
array – Array of in-plane wavenumbers

azimuthal_angles
array – Azimuthal propagation angles of partial plane waves

layer_system
smuthi.layers.LayerSystem – Layer system object to which the plane wave expansion refers.

coefficients
list of numpy arrays – coefficients[i][j, pm, k, l] contains $g_{ij}^{\pm}(\kappa_k,\alpha_l)$, where $\pm$ is + for pm = 0 and $\pm$ is - for pm = 1, and the coordinates κ_k and α_l correspond to n_effective[k] times the angular frequency and azimuthal_angles[l], respectively.

azimuthal_angle_grid()
Meshgrid of azimuthal_angles with respect to n_effective

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

k_parallel_grid()
Meshgrid of n_effective with respect to azimuthal_angles

k_z()

k_z_grid()

class smuthi.field_expansion.**SphericalWaveExpansion**(*k*, *l_max*, *m_max=None*, *type=None*, *reference_point=None*, *valid_between=None*)

coefficients_tlm(*tau*, *l*, *m*)

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

smuthi.field_expansion.**ab5_coefficients**(*l1*, *m1*, *l2*, *m2*, *p*, *symbolic=False*)
a5 and b5 are the coefficients used in the evaluation of the SVWF translation operator. Their computation is based on the sympy.physics.wigner package and is performed with symbolic numbers.

Parameters

- **l1** (*int*) – l=1,...: Original wave's SVWF multipole degree
- **m1** (*int*) – m=-l,...,l: Original wave's SVWF multipole order
- **l2** (*int*) – l=1,...: Partial wave's SVWF multipole degree
- **m2** (*int*) – m=-l,...,l: Partial wave's SVWF multipole order
- **p** (*int*) – p parameter
- **symbolic** (*bool*) – If True, symbolic numbers are returned. Otherwise, complex.

Returns A tuple (a5, b5) where a5 and b5 are symbolic or complex.

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

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

smuthi.field_expansion.**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.field_expansion.**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.field_expansion.**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)

smuthi.index_conversion module

smuthi.initial_field module

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

Base class for initial field classes

plane_wave_expansion(*layer_system*)

Virtual method to be overwritten.

spherical_wave_expansion(*particle*, *layer_system*)

Virtual method to be overwritten.

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

angular_frequency()

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

Evaluate the complex electric field corresponding to the plane 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

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.

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

smuthi.layers module

smuthi.linear_system module

smuthi.near_field module

smuthi.near_field.**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.near_field.**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.near_field.**scattered_electric_field**(*x*, *y*, *z*, *k_parallel*, *azimuthal_angles*, *vacuum_wavelength*, *particle_list*, *layer_system*)
Complex electric scttered near field. Return the x, y and z component of the scattered electric field.

Parameters

- **x** (*numpy array*) – x-coordinates of points in space where to evaluate field.
- **y** (*numpy array*) – y-coordinates of points in space where to evaluate field.
- **z** (*numpy array*) – z-coordinates of points in space where to evaluate field.
- **k_parallel** (*1D numpy array*) – In plane wavenumbers for the plane wave expansion
- **azimuthal_angles** (*1D numpy array*) – Azimuthal angles for the plane wave expansion
- **vacuum_wavelength** (*float*) – Vacuum wavelength
- **particle_list** (*list*) – List of smuthi.particle.Particle objects
- **layer_system** (*smuthi.layers.LayerSystem*) – Stratified medium

smuthi.near_field.**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=None*, *azimuthal_angles=None*, *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** (*array*) – 1-D Numpy array of in-plane wavenumbers for the plane wave expansion
- **azimuthal_angles** (*array*) – 1-D Numpy array of azimuthal angles for the plane wave expansion
- **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).

smuthi.particle_coupling module

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

smuthi.plane_wave_pattern module

smuthi.post_processing module

smuthi.read_input module

smuthi.simulation module

smuthi.spherical_functions module

smuthi.spherical_functions.**double_factorial**(*n*)
Return double factorial.

Parameters **n** (*int*) – Argument (non-negative)

Returns Double factorial of n

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.**factorial**(*n*)
Return factorial.

Parameters **n** (*int*) – Argument (non-negative)

Returns Factorial of n

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** (*array*) – cosine of theta (or kz/k)
- **st** (*array*) – 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_bessel**(*n*, *x*)
Spherical Bessel function. This is a wrapper for scipy.special.sph_jn to make it operate on numpy arrays.

As soon as some bug for complex arguments is resolved, this can be replaced by scipy.special.spherical_jn. https://github.com/ContinuumIO/anaconda-issues/issues/1415

Parameters

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

Returns Spherical Bessel function as array.

smuthi.spherical_functions.**spherical_hankel**(*n*, *x*)
Spherical Hankel function of first kind.

Parameters

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

Returns Spherical Hankel function as array.

smuthi.t_matrix module

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

Input: tau integer: spherical polarization, 0 for spherical TE and 1 for spherical TM l integer: l=1,... multipole degree (polar quantum number) 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)

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.

..todo:: testing

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

smuthi.vector_wave_functions module

smuthi.nfmds package

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

smuthi.nfmds.**install_nfmds**()

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

R

S

T