MSTM_studio Documentation

Release 1.0.2

Leon Avakyan

Jun 20, 2023

Contents

1 Features **3**

2 Contents **5**

2.1 Installation 5

2.2 Manual: GUI 7

2.3 Manual: Scripting 17

3 Contacts **45**

Bibliography **47**

Python Module Index **49**

Index **51**

Python wrapper for multiple sphere T-matrix (MSTM) code and Mie theory to calculate surface plasmon resonance (SPR) spectrum and fit it to experiment.

CHAPTER 1

Features

- Materials defined from constant expression, from tabulated file or from analytical formula of Rioux et al for Au-Ag;
- Simple functional contributions (linear, lorentzian, gaussian)
- Mie theory contributions
- MSTM calculations: spectra and near-field intensity
- Fitting of experimental data by any of the mentioned contributions, including combinations
- Interactive graphical user interface
- Flexible Python scripting

CHAPTER 2

Contents

2.1 Installation

2.1.1 Source code

Source code of Python wrapper is available on GitHub <https://github.com/lavakyan/mstm-spectrum>. Stable version published on PyPi <https://pypi.org/project/mstm-studio/>.

The source code of MSTM is not included and should be obtained from <https://scattport.org/index.php/light-scattering-software/multiple-particle-scattering/468-mstm>. MSTM studio can be run without MSTM binary, but with restricted functionality.

For non-spherical particles (currently available only spheroids) the ScatterPy library is used (See *Binding with ScatterPy*).

2.1.2 Linux installation

Install from PyPi:

```
pip install mstm_studio
```

On systems without root access:

```
pip install mstm_studio --user
```

Running GUI with

```
python -m mstm_studio
```

May be required to explicitely specify python version, i.e. use `pip3` and `python3` in above commands.

Binding with MSTM

MSTM-studio will search for `mstm.x` binary in `~/bin` directory.

This can be altered by setting of *MSTM_BIN* environment variable, i.e. in bash:

```
export $MSTM_BIN=~/my_compiled_mstm/mstm_v3.bin
```

Note: MSTM can be compiled with gfortran as:

```
gfortran  mpidefs-serial.f90 mstm-intrinsics-v3.0.f90 mstm-modules-v3.0.f90 mstm-main-
↪v3.0.f90 -O2  -o mstm.x
```

This is serial compilation, for parallel the file `mpidefs-serial.f90` should be replaced. Consult the MSTM website for details.

2.1.3 Windows installation

The tested way is using Anaconda Python distribution <https://www.anaconda.com/>.

1. Open "Anaconda Prompt". The new terminal window should pop up.
2. Type in `pip install mstm_studio`. This may take a while since the dependent code will be downloaded and installed.
3. Check GUI by typing `python -m mstm_studio` in Anaconda Prompt or check python scripting by typing `import mstm_studio` in python console.

Binding with MSTM

4. Obtain the MSTM binary. Put it to some folder.
5. Setup environmental variable `MSTM_BIN` to point on the binary. The shell comannd `SETX MSTM_BIN="path_to_your_mstm_bin"` will do the temporary setup, which is useful for making `*.cmd` scripts. Permanent setup of environemnt variable should be done with graphical interface, see for example, <https://docs.oracle.com/en/database/oracle/r-enterprise/1.5.1/oread/creating-and-modifying-environment-variables-on-windows.html>.

Note: If you write *.cmd script to run gui, don't forget to update `PATH` variable to point on the Python distribution. The easist way is to type `echo %PATH%` in Anaconda Promt, and use the output in your script. Example of GUI running script is:

```
@ECHO OFF
PATH=C:\ProgramData\Anaconda3;C:\ProgramData\Anaconda3\Library\mingw-w64\bin;
↪C:\ProgramData\Anaconda3\Library\usr\bin;C:\ProgramData\Anaconda3\Library\bin;
↪C:\ProgramData\Anaconda3\Scripts;C:\ProgramData\Anaconda3\bin;
↪C:\ProgramData\Anaconda3\condabin;%PATH%
set MSTM_BIN="C:\Users\L\Desktop\mstm_studio old\mstm-spectrum\mstm.exe"
python.exe -m mstm_studio
PAUSE
```

The last command (`PAUSE`) is put to prevent console windows from closing after program is ended.

2.1.4 Binding with ScatterPy

For calculation of extinction spectra of isolated non-sphericla particle ScatterPy can be used. This library is available on github <https://github.com/TCvanLeth/ScatterPy> and PiPy repository.

Installation from PyPi: `pip install scatterpy` or `pip install scatterpy --user`

ScatterPy without Numba

ScatterPy requires Numba library for speeding up the calculation. However, it is possible to install without Numba:

1. Download scatterpy source code
2. Edit file `scatterpy/special.py`. Remove line:

   ```
   import numba as nb
   ```

   and add lines:

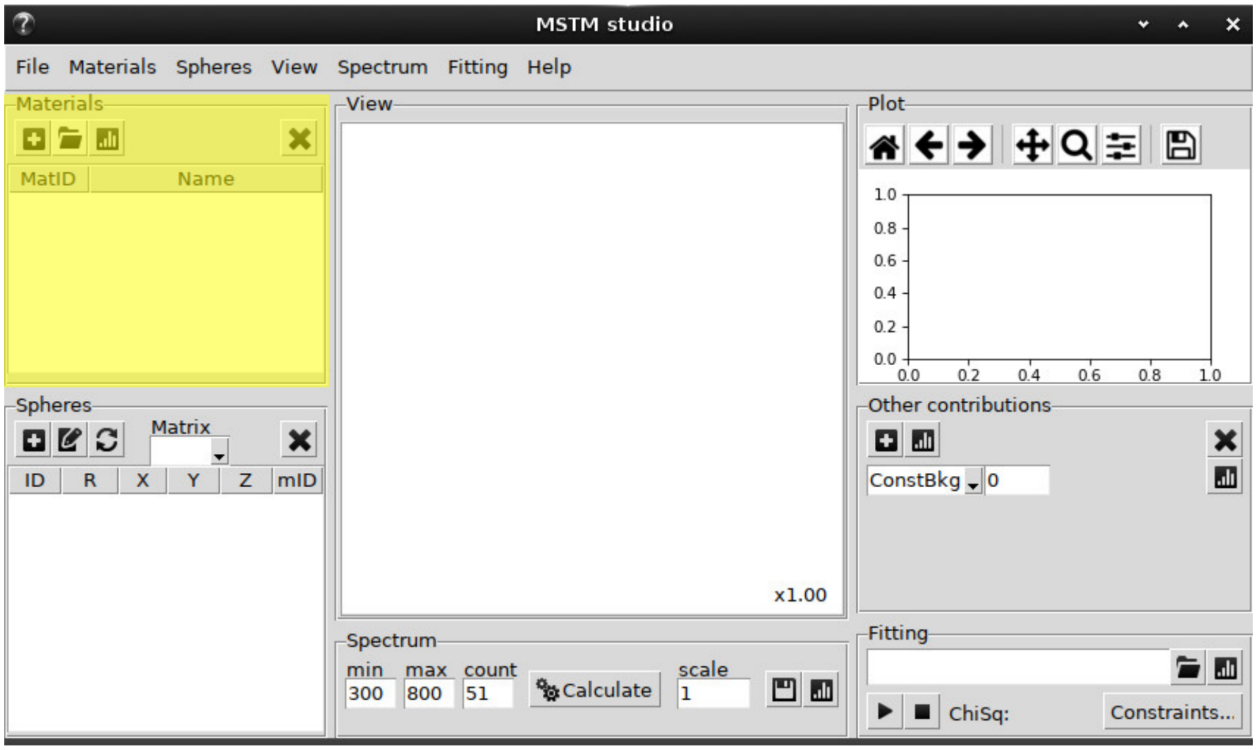
   ```
   try:
       import numba as nb
   except ImportError:
       print('WARNING: Numba support is disabled in ScatterPy')
   ```

3. Build and install: `python setup.py install` (Needed setuptools and may be other dev packages)

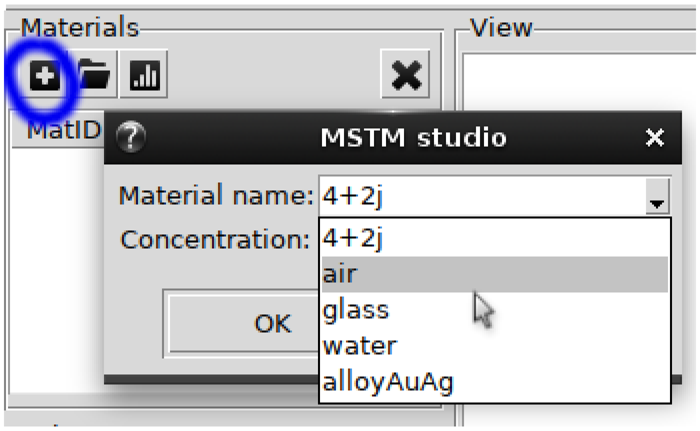
2.2 Manual: GUI

Interactive graphical interface for calculation and analysis of optical extinction spectra.

2.2.1 Materials



Material can be added from the predefined list:



Also the complex number can be typed in here.

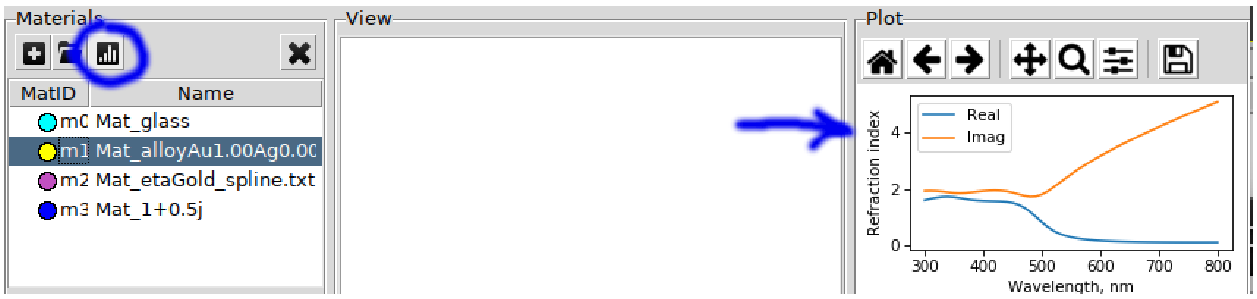
The option *AlloyAuAg* correspond to analytical parametrixation for silver-gold alloys [Rioux2014] and requires the specification of Au ratio ($0 \leq x \leq 1$).

Next button will add material from file with refractive index data stored in column format, i.e.:

```
lambda      n      k
 0.100      2.8883  1.3062
 0.101      2.8735  1.2439
 0.101      2.8564  1.1856
 ...
```

First column – wavelength in *nm* or *mum*, second and third – real and imaginary parts of refractive index.

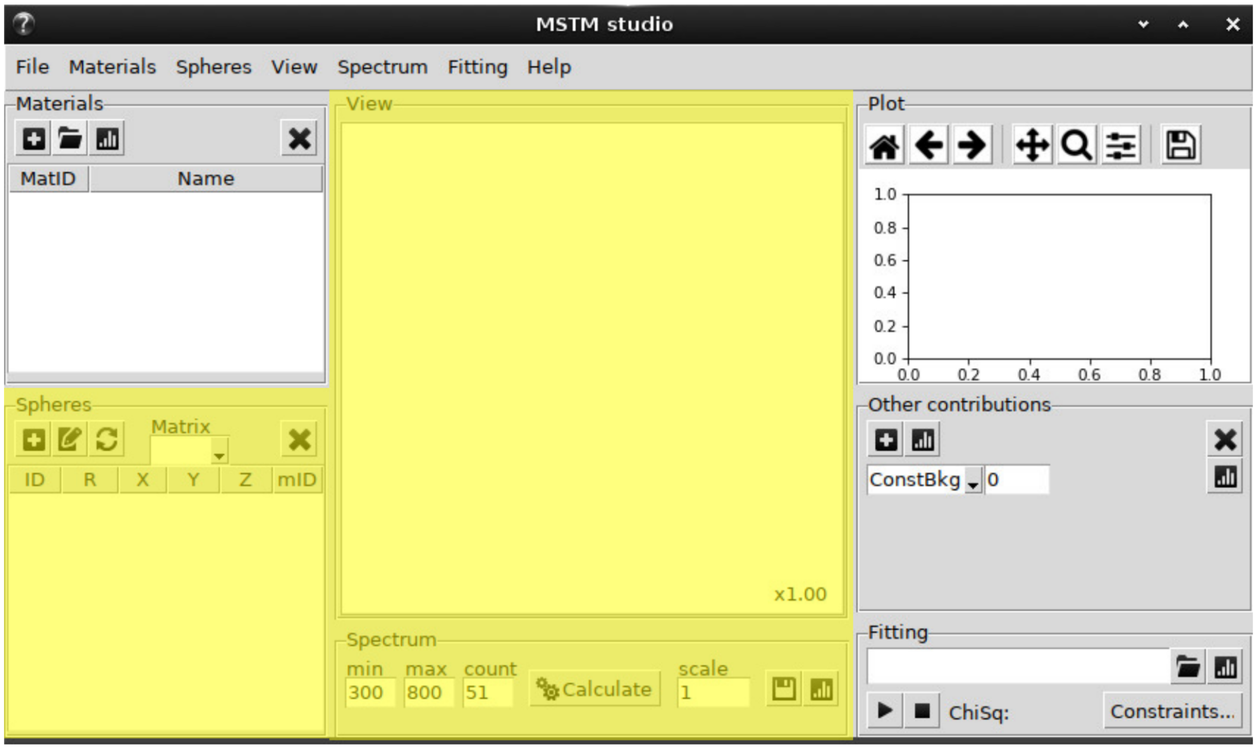
The refractive index data can be plotted to check sanity:



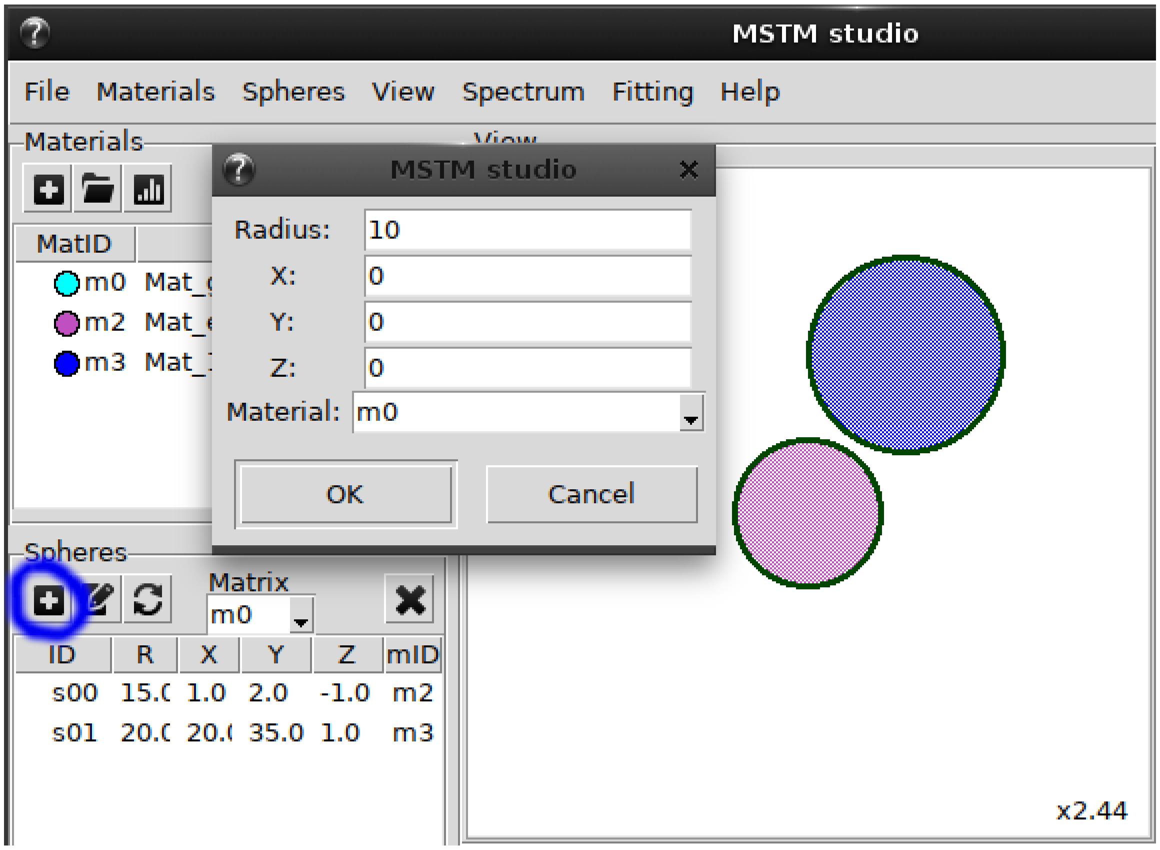
Cross button deletes the selected material.

2.2.2 Multi-Spheres T-Matrix

Mutli-spheres T-matrix calculations are done by calling the external binary MSTMcode written by Mischenko and Mackowski. Currently supported are the spectra calculations (extinction, scattering or absorbtion) and near field visualization. Both modes requires specification of the spheres geometry and thier material. GUI provides the following options:



Plus button – add new sphere.



Radius: 10
X: 0
Y: 0
Z: 0
Material: m0

ID	R	X	Y	Z	mID
s00	15.(	1.0	2.0	-1.0	m2
s01	20.(	20.(	35.0	1.0	m3

Important to specify material label for the sphere.

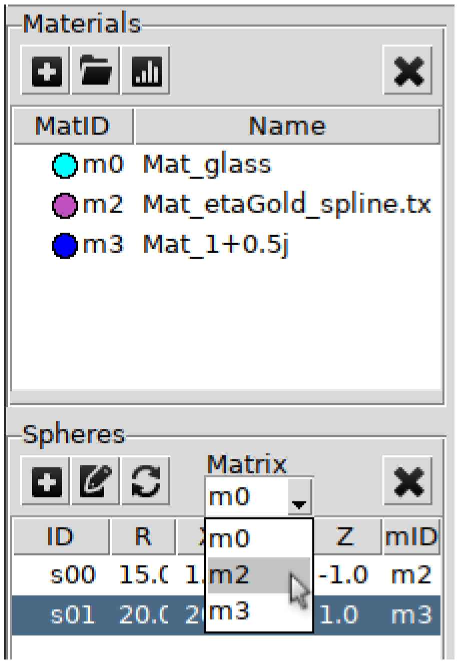
Button with pencil or doulbe-click on a table row – edit selected sphere.

Note: double click on a row in material table allow to change the viewed material color.

Circle-arrows – refresh 3D view.

3D view can be **rotated** with pressed left mouse button and **zoomed** in or out with mouse wheel.

Environment material by default is *m0*. This can be changed using menu:



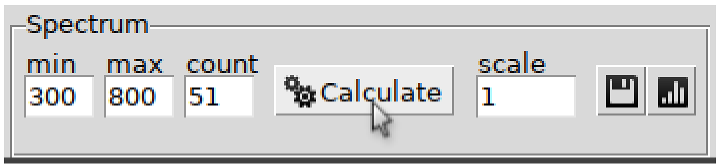
MatID	Name
m0	Mat_glass
m2	Mat_etaGold_spline.tx
m3	Mat_1+0.5j

ID	R	X	Z	mID
s00	15.(	1.	-1.0	m2
s01	20.(	2	1.0	m3

Cross button deletes the selected sphere.

Calculation modes:

MSTM spectrum



min	max	count	scale
300	800	51	1

The spectrum calculation may be configured by pressing the “Setup” button. The relevant options in “Setup MSTM” window are:

“Calculation” mode could be extinction (default mode), absorbtion or scattering spectrum;

“min” – minimal wavelength (in nm),

“max” – maximal wavelength (in nm),

“count” – number of wavelength points. By default the spacing is 10 nm;

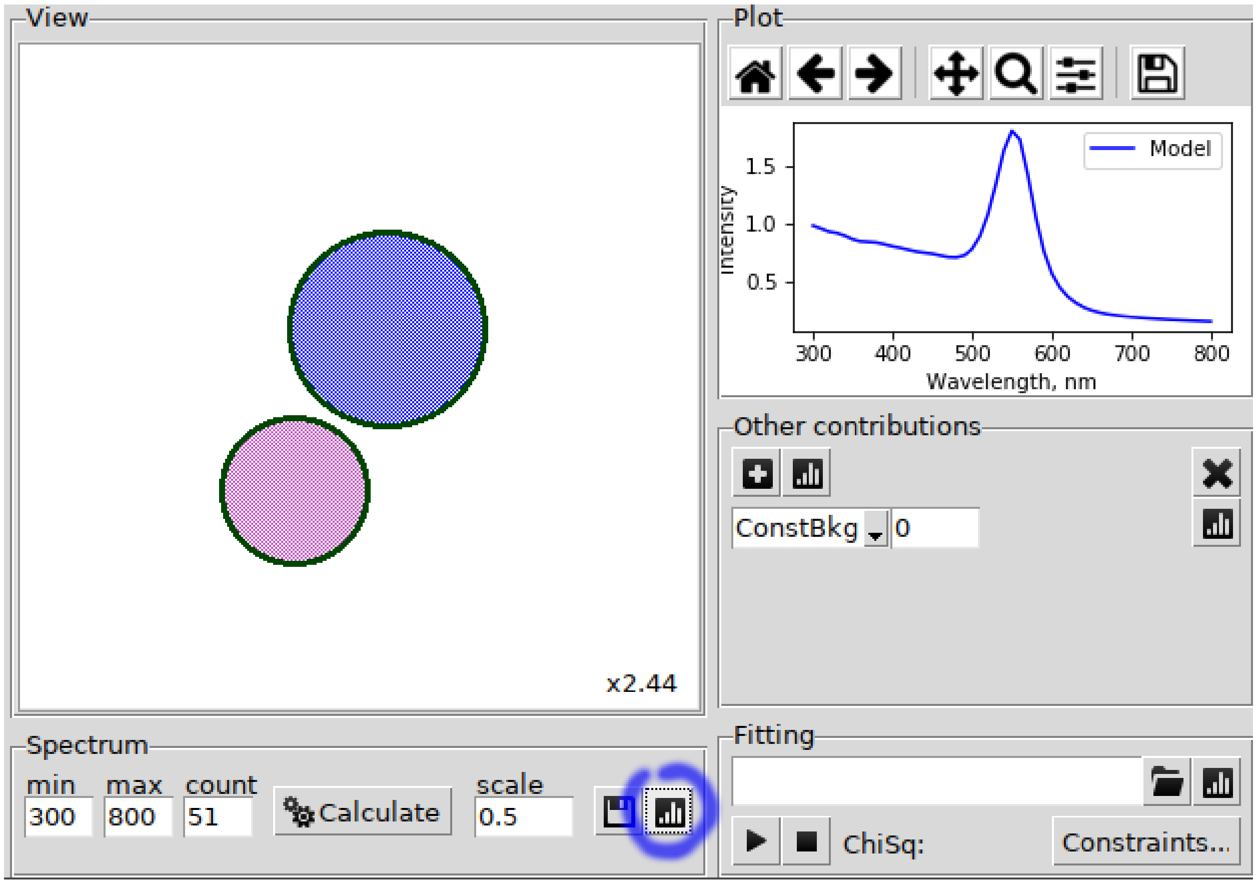
By default the averaging is performed. If “average over orientations” is unchecked the incidence beam angles must be specified, i.e. “Azimuth angle” and “Polar angle”. In this case the spectra for two polarization cases (parallel and orthogonal) will be obtained.

“Calculate” button of the main window runs MSTM binary in temporary directory (OS-dependent) and reads the results.

“scale” – total outer multiplier.

save button – save extinction to column file.

plot button – plot without re-calculation (i.e. with new *scale*).

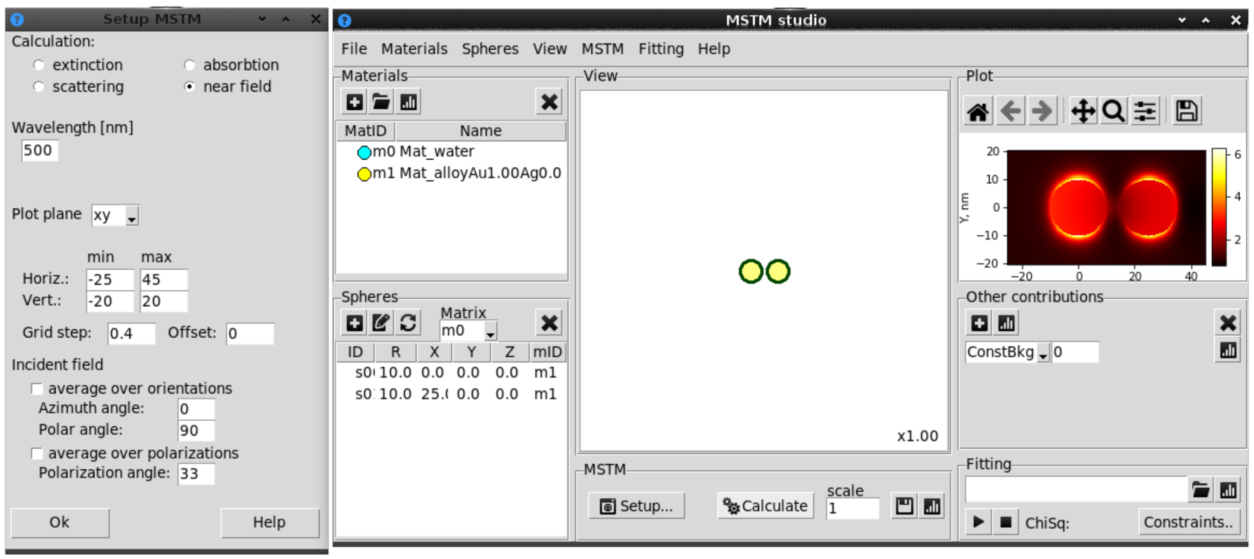


min: 300, max: 800, count: 51, scale: 0.5

ConstBkg 0

The plot controls are rendered by Matplotlib, and can depend on the library version. Generally, it is possible to zoom region of interest and save graphic as a raster or vector image.

MSTM near-field



Calculation:
extinction absorbtion
scattering [✓ near field]

Wavelength [nm]
500

Plot plane xy

	min	max
Horiz.:	-25	45
Vert.:	-20	20

Grid step: 0.4 Offset: 0

Incident field
average over orientations
Azimuth angle: 0
Polar angle: 90
average over polarizations
Polarization angle: 33

MatID	Name
m0	Mat_water
m1	Mat_alloyAu1.00Ag0.0

Matrix m0

ID	R	X	Y	Z	mID
s0	10.0	0.0	0.0	0.0	m1
s0	10.0	25.(	0.0	0.0	m1

ConstBkg 0

scale 1

Near field calculation should be enabled in “Setup MSTM” window by selection of “near field” mode. The adjustable options are:

“Wavelength [nm]” – the wave length of the incident beam;

“Plot plane” – the 2D plane orientation - XY, YZ or ZX;

“Horiz”, “Vert”, “min”, “max” – the minimal and maximal coordinate on the plane, in nm;

“Grid step” – grid grain size, in nm;

“Offset” – displace of the plane from the origin, can be positive or negative, in nm.

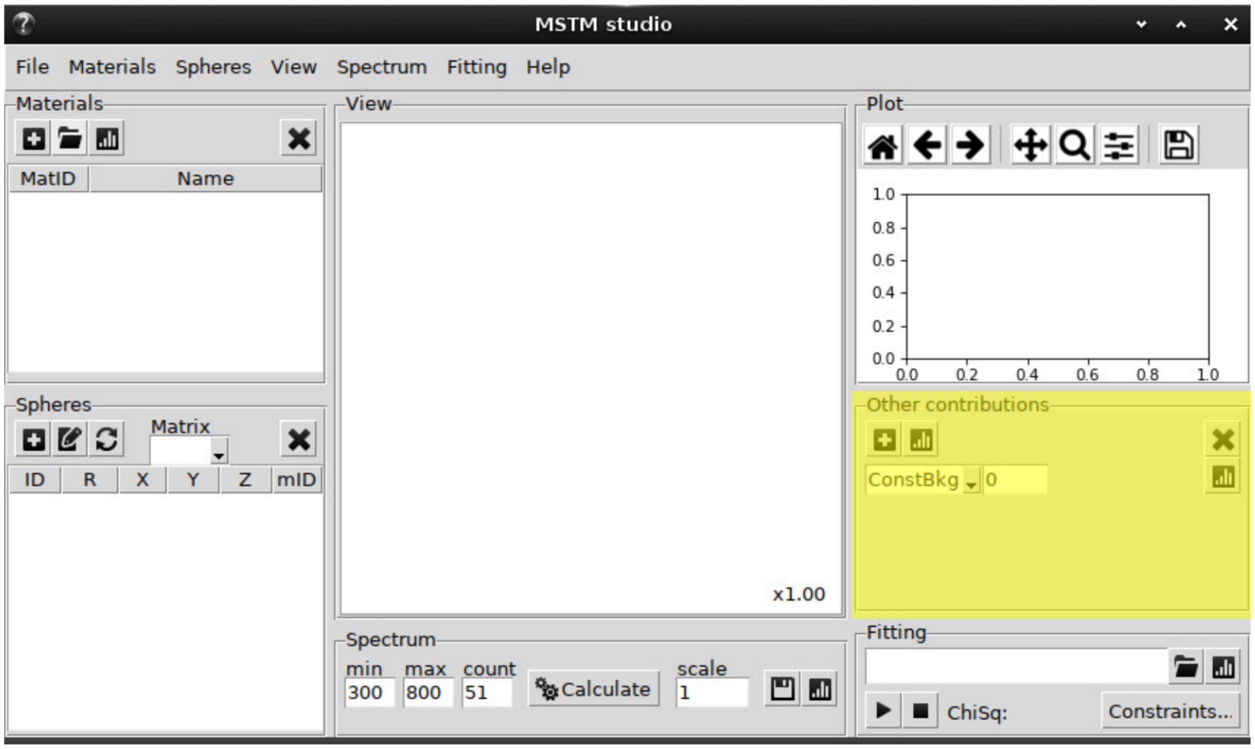
Averaging over the incidence beam orientation can not be performed, therefore “average over orientation” box is automatically unchecked. Orientation should be specified with “Azimuth angle” and “Polar angle” (in degrees).

Also, polarization of the beam should be fixed and specified with “Polarization angle”.

Alternatively, the averaging over polarization can be done by **MSTM Studio** mimicing the natural polarization. In this case “average over polarization” box should be checked and number of polarization values (from 0 to 90 degrees) to be averaged should be specified (12 by default).

Obtained data can be saved as image or text file in the same way as for spectrum.

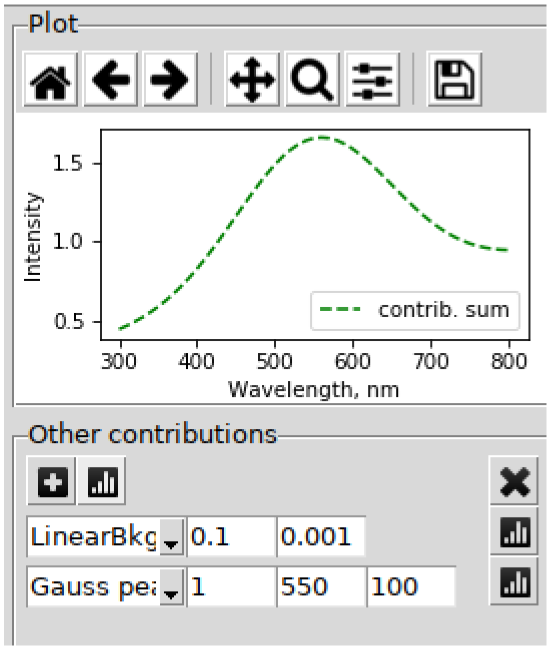
2.2.3 Mie and other contributions



Plus button – add new contribution,

plot button – plot the sum of all contributions,

cross button – delete the last contribution in the list.



Each contribution has the type, a number of parameters and individual plot button.

Possible contributions and thier parameters:

1. ConstBkg – constant background addition

   1.1. shift

2. LinearBkg – linear background addition (see `mstm_studio.contributions.LinearBackground`)

   2.1. shift

   2.2. slope

3. LorentzBackground – background contribution mimicing UV-absorbtion of silica glass (see `mstm_studio.contributions.LorentzBackground`) with center at ~ 250 nm.

   3.1. scale

   3.2. peak width Γ

4. Mie single – extinction of single sphere (see `mstm_studio.contributions.MieSingleSphere`)

   4.2. scale

   4.3. diameter (in nm)

   4.4. material label

   Note: the enviromnent material will be taken from *Matrix* field of *Spheres* panel.

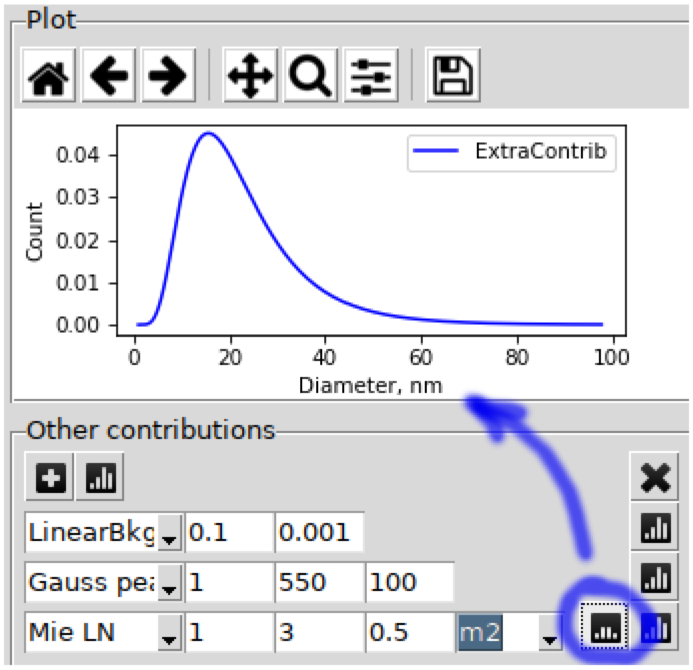
5. Mie LN – extinction of ensemble of non-interacting spheres with sizes disributed accordint to Log-Normal (LN) law (see `mstm_studio.contributions.MieLognormSpheres`). Actually, the cached version of the class optimized for the fitting is used.

   5.1. scale

   5.2. LN parameter μ

   5.3. LN parameter σ

   The new plot button will show the size distribution:



LinearBkg	0.1	0.001		
Gauss pe	1	550	100	
Mie LN	1	3	0.5	m2

   Note: the window may require widening to show all the parameters and buttons.

6. Lorentz peak – Lorentzian function (see `mstm_studio.contributions.LorentzPeak`)

6.1. scale

6.2. center

6.3. width

7. Gauss peak – Gaussian function (see `mstm_studio.contributions.GaussPeak`)

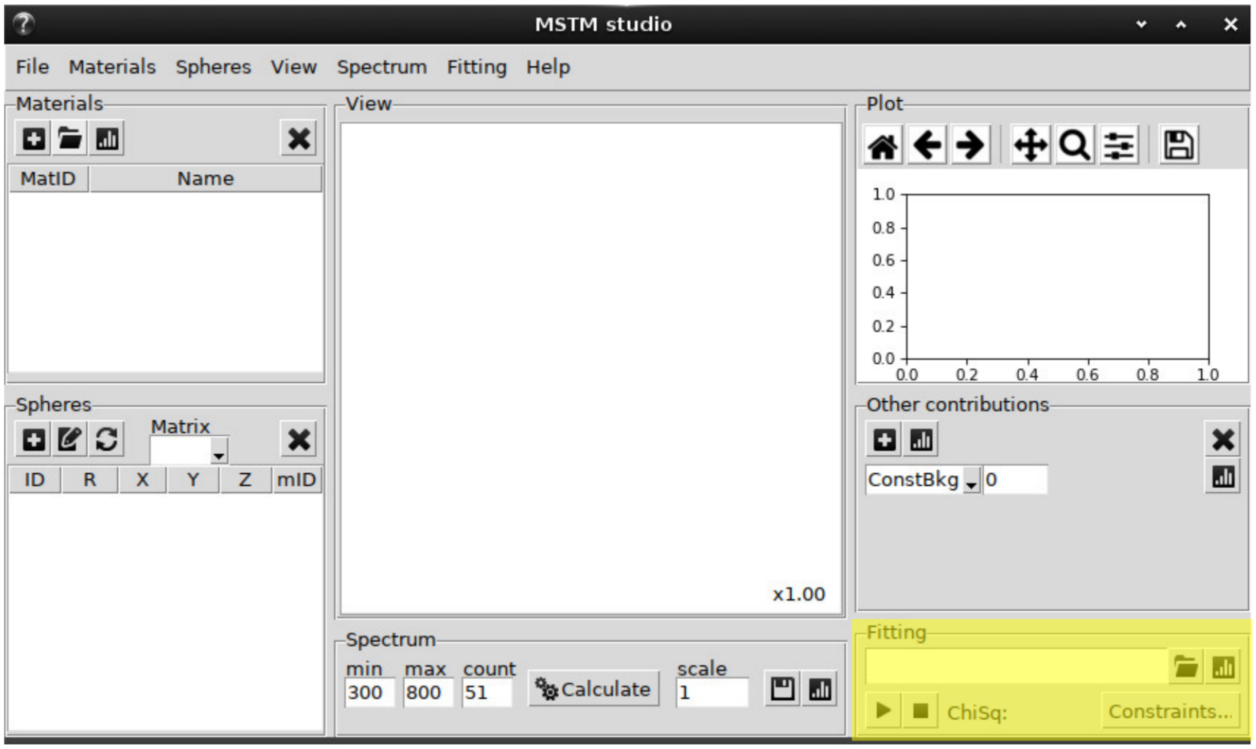
   7.1. scale

   7.2. center

   7.3. width

8. Au film – not implemented
9. bst-3Au/glass – not implemented

2.2.4 Fitting and constraints



Spectrum: min 300, max 800, count 51, scale 1

ConstBkg 0

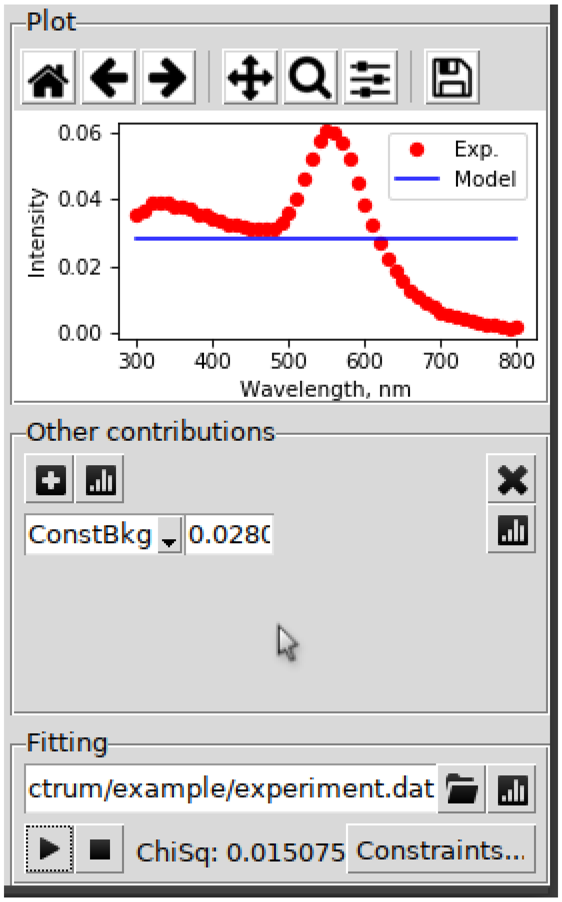
Open file – select file with column-formatted data to fit

Plot button – visualize data and compare it with theory

Play button – start fitting

Stop button – interrupt fitting

ChiSq – shows the obtained fitting quality parameter χ^2 (see :class:mstm_studio.mstm_spectrum.SPR).



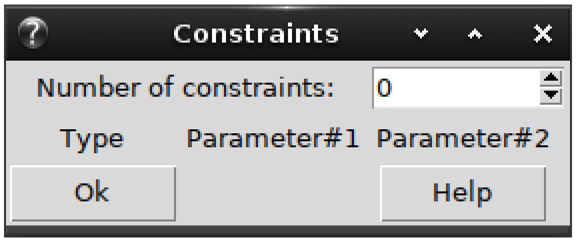
ConstBkg 0.028(

ctrum/example/experiment.dat

ChiSq: 0.015075 Constraints...

Constraints button – open new window with constraints options

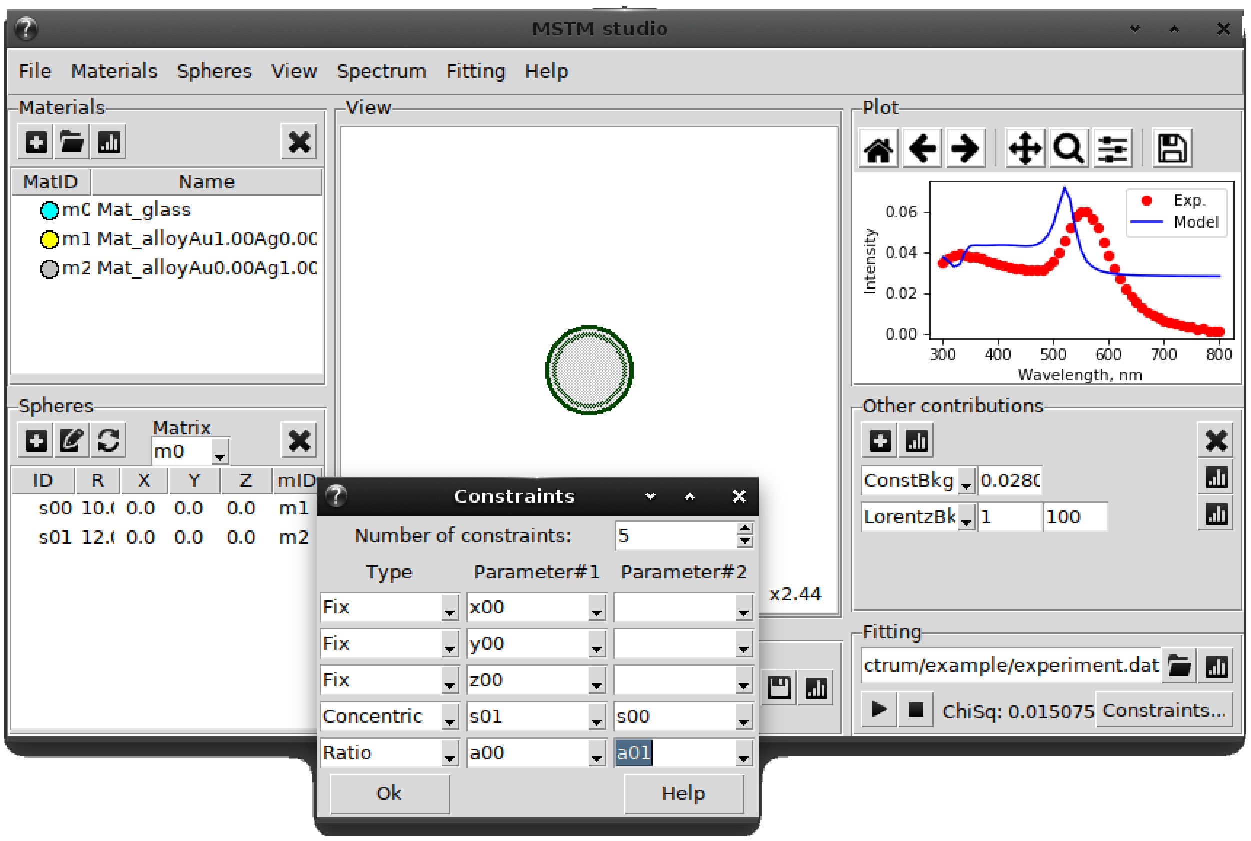
Empty constraints window:



Number of constraints: 0

Type Parameter#1 Parameter#2

Example of possible constraints for core-shell nanoparticle:



MatID	Name
m0	Mat_glass
m1	Mat_alloyAu1.00Ag0.0C
m2	Mat_alloyAu0.00Ag1.0C

Matrix: m0

ID	R	X	Y	Z	mID
s00	10.(	0.0	0.0	0.0	m1
s01	12.(	0.0	0.0	0.0	m2

Number of constraints: 5

Type	Parameter#1	Parameter#2
Fix	x00	
Fix	y00	
Fix	z00	
Concentric	s01	s00
Ratio	a00	a01

ConstBkg	0.028C	
LorentzBk	1	100

ctrum/example/experiment.dat

ChiSq: 0.015075

For details consult *Constraints*

2.3 Manual: Scripting

Use Python scripts for full control of calculations.

..note:: The easy way to point the MSTM binary in script is the usage of the *os* module:

```
import os
os.environ['MSTM_BIN'] = 'your path to mstm binary'
```

The default path is '~/bin/mstm.x'

2.3.1 Materials

The materials are characterized by refractive index, which is a square root (complex valued) of macroscopic dielectric function. Generally, this should be the spectral function, i. e.

$$n_c(\lambda) = n(\lambda) + i \cdot k(\lambda) = \sqrt{\epsilon_1(\omega) + i \cdot \epsilon_2(\omega)}$$

so that

$$n = \sqrt{(|\epsilon| + \epsilon_1)/2}$$
$$k = \sqrt{(|\epsilon| - \epsilon_1)/2}$$

with $\hbar\omega = 2\pi\hbar c/\lambda$.

Constant Material

The material with constant refreactive index can be specified as first constructor argument (*file_name*):

```
>>> from mstm_studio.mstm_spectrum import Material
>>> mat_glass = Material('1.5')
>>> mat_glass.get_n(500)
array(1.5)
```

Complex value can be supplied too:

```
>>> mat_lossy = Material('3+1j')
>>> mat_lossy.get_n(500)
array(3.)
>>> mat_lossy.get_k(500)
array(1.)
```

Also the predifned names can be used: *air*, *water*, *glass*.

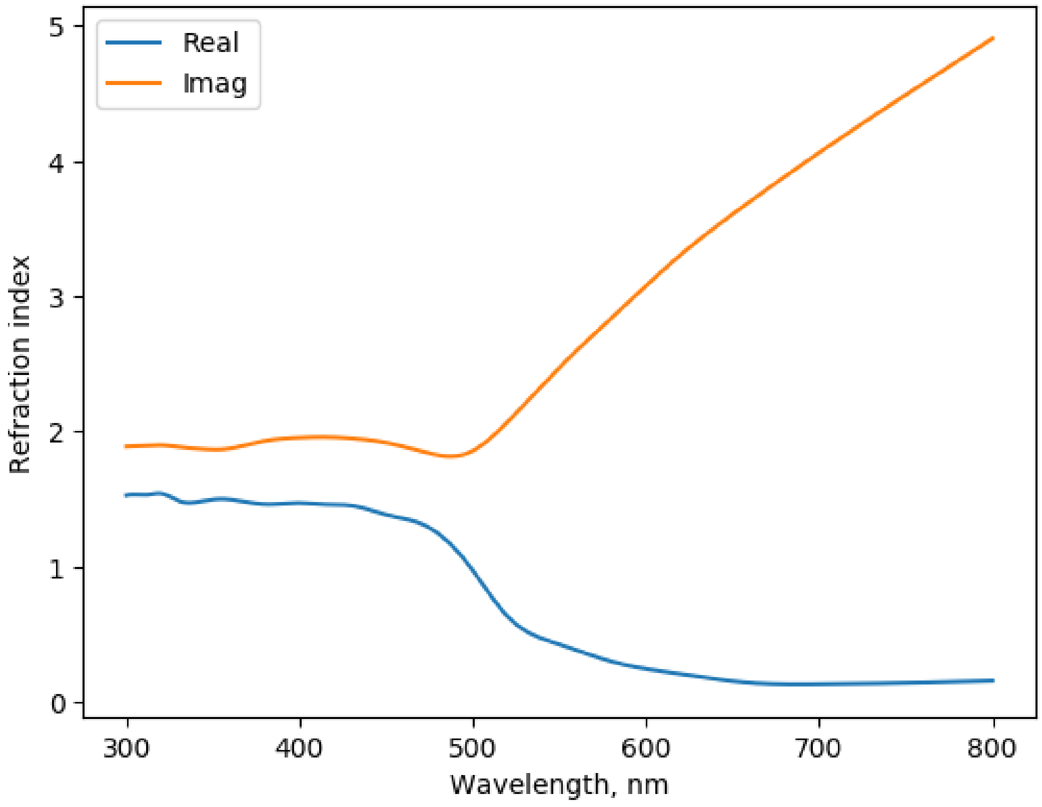
Loading from file

The tabular data on refractive index is convinient to store in file. The header of the file required to have special labels: `lambda n k`. The example file “etaGold.txt” can be found in directory “nk” of source distribution.

Assuming the file “etaGold.txt” is in the same directory where script is running, it can be loaded with

```
gold = Material('etaGold.txt')
fig, axs = gold.plot()
fig.savefig('loaded_gold.png', bbox_inches='tight')
```

Resulted plot



Note: The extending database of refractive indeces of materials <https://refractiveindex.info/>.

Material from numpy array

Material data can be specified directly by numpy (complex) array by passing *nk* or *eps*. Next examples shows loading of Drude-like dielectric function:

```
from mstm_studio.mstm_spectrum import Material
import numpy as np

wls = np.linspace(300, 800, 51)  # spectral region
omega = 1240. / wls              # freq. domian

omega_p = 9.        # plasma frequency, eV
gamma   = 0.03      # damping, eV
# Drude's dielectric function:
epsilon = 1 + omega_p**2 / (omega * (omega - 1j * gamma))

mat = Material('drude', eps=epsilon, wls=wls)
```

Material class members

class mstm_studio.mstm_spectrum.**Material**(*file_name*, *wls=None*, *nk=None*, *eps=None*)

Material class.

Use *get_n()* and *get_k()* methods to obtain values of refraction index at arbitraty wavelength (in nm).

Parameters:

file_name:

1. complex value, written in numpy format or as string;
2. one of the predefined strings (air, water, glass);
3. filename with optical constants.

File header should state *lambda*, *n* and *k* columns If either *nk= n + 1j*k* or *eps = re + 1j*im* arrays are specified, then the data from one of them will be used and filename content will be ignored.

wls: float array array of wavelengths (in nm) used for data interpolation. If None then `np.linspace(300, 800, 500)` will be used.

plot (*wls=None*, *fig=None*, *axs=None*)
plot `n` and `k` dependence from wavelength

Parameters:

wls: float array array of wavelengths (in nm). If None then `np.linspace(300, 800, 500)` will be used.

fig: matplotlib figure

axs: matplotlib axes

Return:

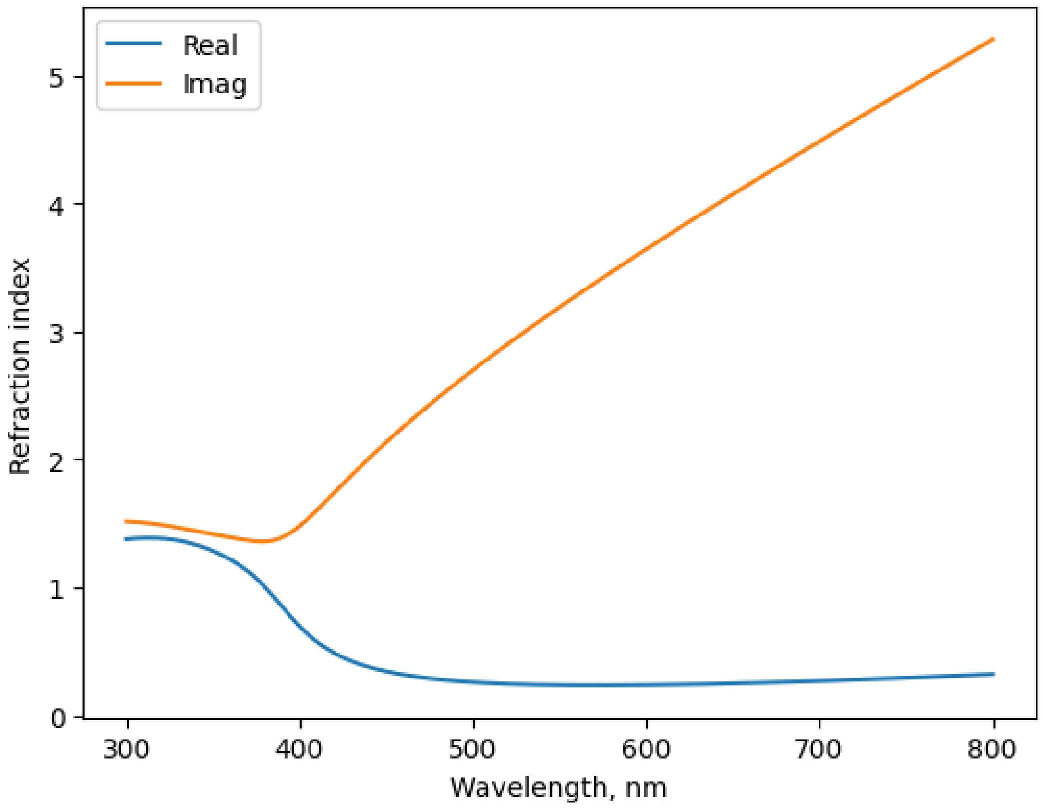
filled/created fig and axs objects

Analytical formula for AuAg

Silver, gold and thier alloy materials can be specified using analytical expression proposed in the study [Rioux2014]. Example for Au:Ag = 1:2 alloy:

```
from mstm_studio.alloy_AuAg import AlloyAuAg

au1ag2 = AlloyAuAg(x_Au=1./3)
fig, axs = au1ag2.plot()
fig.savefig('mat_au1ag2.png', bbox_inches='tight')
```

Resulted plot



class mstm_studio.alloy_AuAg.**AlloyAuAg**(*x_Au*)

Material class for AuAg alloys.

Use *get_n()* and *get_k()* to obtain values of refraction indexes (real and imaginary) at arbitraty wavelength (in nm) by model and code from Rioux et al doi:10.1002/adom.201300457

Parameters:

x_Au: float fraction of gold

Size correction for dielectric functions

Macroscopic dielectric function obtained for bulk samples can be applied to nanoparticles with caution. It is claimed that only particles of radius above 10 nm can be considered. However, the consideration can be extended to the sizes down to ~ 2 nm by inclusion of the most prominent effect – the decrease of the mean free path length of electrons due to finite size of the nanoparticles. The correction is applied to the γ parameter of the Drude function, so that we had to add the contribution

$$\Delta\epsilon(\omega, D) = \epsilon_{Drude,corr.}(\omega, D) - \epsilon_{Drude}(\omega, D = \infty)$$

to the experimental dielectric function given by the table.

Example for 3 nm gold nanoparticle:

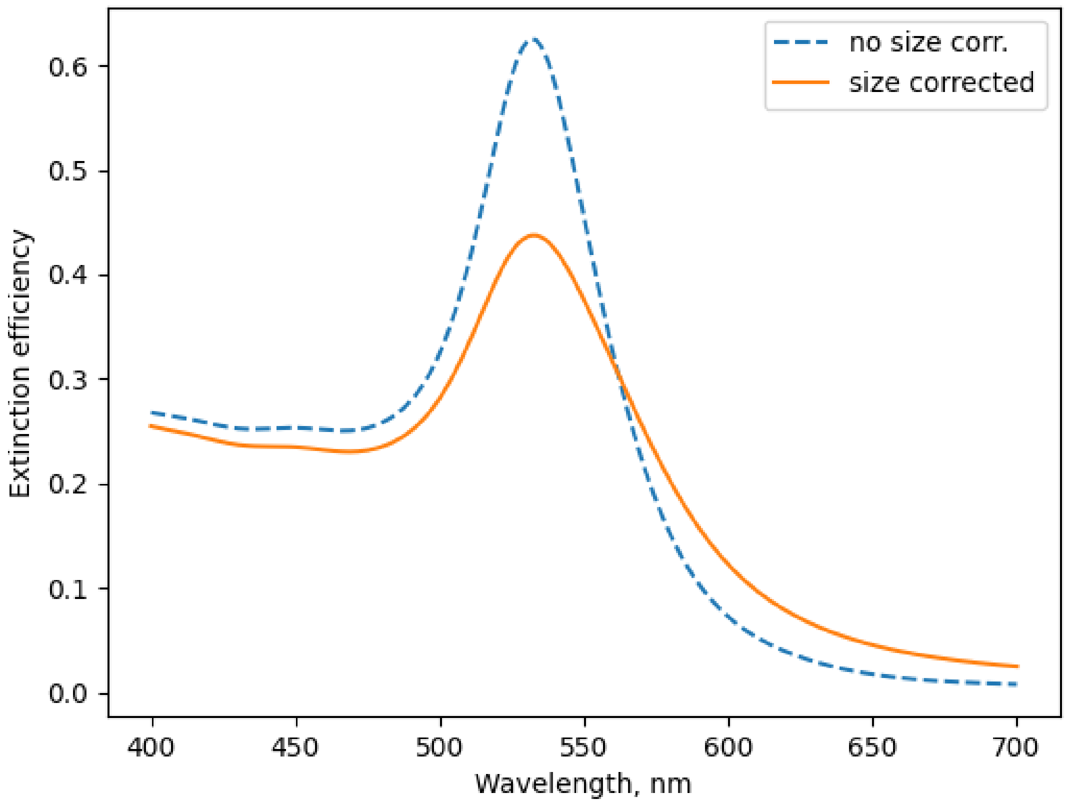
```
from mstm_studio.mstm_spectrum import Material
from mstm_studio.diel_size_correction import SizeCorrectedGold
from mstm_studio.contributions import MieSingleSphere
import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
D = 3  # particle size
wls = np.linspace(400, 700, 201)  # spectral region

gold = Material('etaGold.txt')  # bulk dielectric function
gold_corr = SizeCorrectedGold('etaGold.txt')  # corrected, D-dependent

mie = MieSingleSphere(wavelengths=wls, name='MieSphere')

mie.set_material(material=gold, matrix=1.5)
plt.plot(wls, mie.calculate(values=[1, D]), '--', label='no size corr.')

mie.set_material(material=gold_corr, matrix=1.5)
plt.plot(wls, mie.calculate(values=[1, D]), label='size corrected')
```

Resulted plot



Currently the corrections for gold and silver are implemented:

class mstm_studio.diel_size_correction.**SizeCorrectedGold**(*file_name*, *wls=None*, *nk=None*, *eps=None*)

Size correction for gold dielectric function (mean free path is limited by particle size) according to

A. Derkachova, K. Kolwas, I. Demchenko Plasmonics, 2016, 11, 941 doi: <10.1007/s11468-015-0128-7>

class mstm_studio.diel_size_correction.**SizeCorrectedSilver**(*file_name*, *wls=None*, *nk=None*, *eps=None*)

Size correction for gold dielectric function (mean free path is limited by particle size) according to

J.M.J. Santillán, F.A. Videla, M.B.F. van Raap, D. Muraca, L.B. Scaffardi, D.C. Schinca J. Phys. D: Appl. Phys., 2013, 46, 435301 doi: <10.1088/0022-3727/46/43/435301>

Also the general correction class is available:

class mstm_studio.diel_size_correction.**SizeCorrectedMaterial**(*file_name*, *wls=None*, *nk=None*, *eps=None*, *omega_p=10.0*, *gamma_b=0.1*, *v_Fermi=1.0*, *sc_C=0.0*)

Create material with correction to the finite crystal size. Only life-time limit (~1/gamma) is considered. This should be sufficient for the sizes above ~2 nm. The particles smaller than ~2 nm require more sofisticated modifications (band gap, etc.)

Parameters:

file_name, wls, nk, eps: same meaning as for Material

Parameters for size correction:

omega_p: plasma frequency (bulk) [eV]

gamma_b: life-time broadening (bulk) [eV]

v_Fermi: Fermi velocity (bulk) [nm/fs]

sc_C: size-corr. adj. parameter [unitless]

size of the particle is specified as *self.D*

get_gamma_corr()

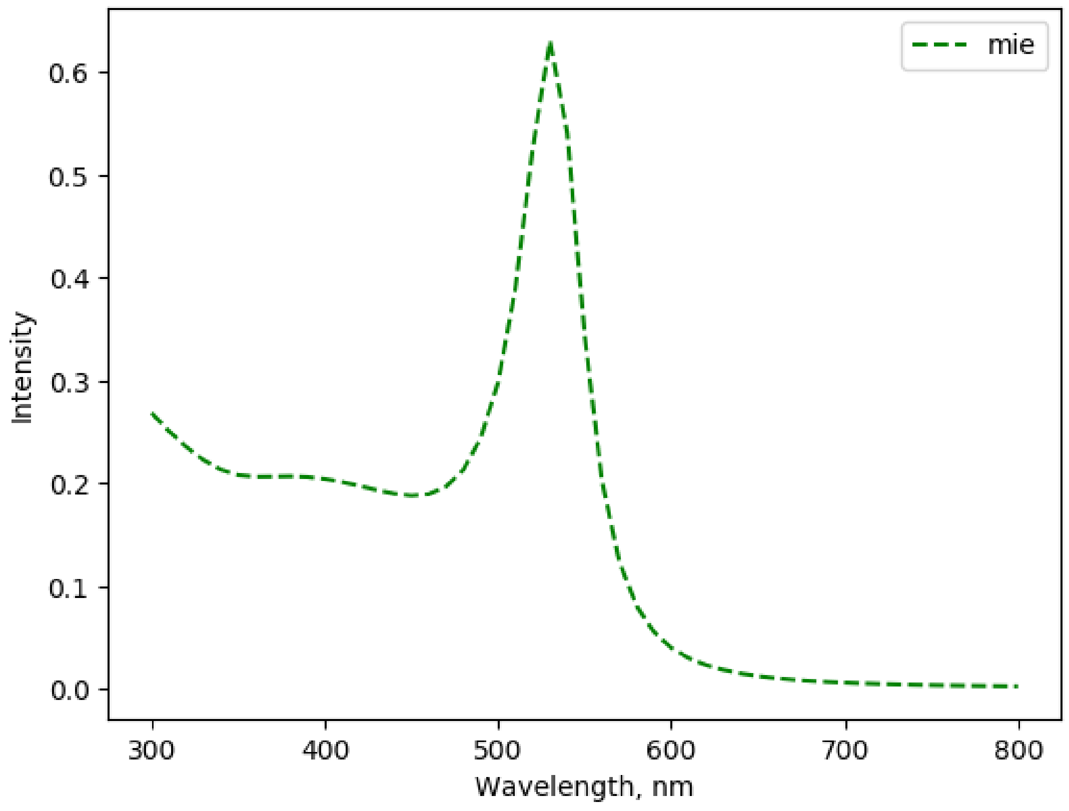
correction to the life time energy broadening (gamma) in the Drude low induced by the finite particle size

2.3.2 Simple functions and Mie theory

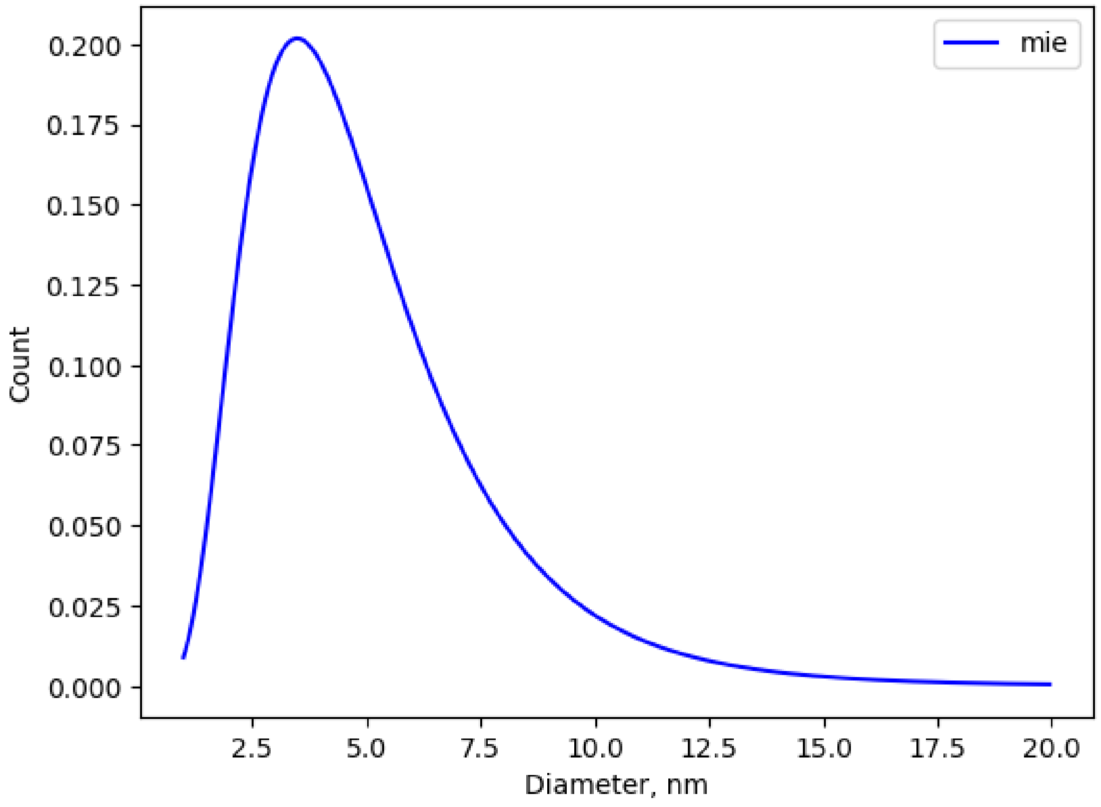
Example

The example how to obtain contribution to the extinction from Log-Normally distributed spheres. Other contributions are evaluated in similar way.

```
from mstm_studio.contributions import MieLognormSpheres
from mstm_studio.alloy_AuAg import AlloyAuAg
import numpy as np

mie = MieLognormSpheres(name='mie', wavelengths=np.linspace(300,800,51))
mie.set_material(AlloyAuAg(x_Au=1), 1.5)  # golden sphere in glass

values = [1, 1.5, 0.5]  # scale, mu, sigma
fig, _ = mie.plot(values)
fig.savefig('mie_contrib.png', bbox_inches='tight')
```



```
mie.MAX_DIAMETER_TO_PLOT = 20    # 100 nm is by default
fig, _ = mie.plot_distrib(values)
fig.savefig('mie_distrib.png', bbox_inches='tight')
```



Classes

Contributions to UV/vis extinction spectra other then obtained from MSTM.

class `mstm_studio.contributions.`**`Contribution`**(*wavelengths=[], name='ExtraContrib'*)

Abstract class to include contributions other then calculated by MSTM. All lightweight calculated contribtions (constant background, lorentz and guass peaks, Mie, etc.) should enhirit from it.

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

`calculate`(*values*)

This method should be overriden in child classes.

Parameters:

values: list of control parameters

Return:

numpy array of contribution values at specified wavelengths

`plot`(*values, fig=None, axs=None*)

plot contribution

Parameters:

values: list of parameters

fig: matplotlib figure

axs: matplotlib axes

Return:

filled/created fig and axs objects

set_wavelengths(*wavelengths*)
Modify wavelengths

class mstm_studio.contributions.**ConstantBackground**(*wavelengths=[]*, *name='ExtraContrib'*)
Constant background contribution $f(\lambda) = bkg$.

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

calculate(*values*)
Parameters:

values: [bkg]

Return:

numpy array

class mstm_studio.contributions.**LinearBackground**(*wavelengths=[]*, *name='ExtraContrib'*)
Two-parameter background $f(\lambda) = a \cdot \lambda + b$.

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

calculate(*values*)
Parameters:

values: list of control parameters *scale*, *mu* and *Gamma*

Return:

numpy array

class mstm_studio.contributions.**LorentzBackground**(*wavelengths=[]*, *name='ExtraContrib'*)
Lorentz peak in background. Peak center is fixed.

$$L(\lambda) = \frac{scale}{(\lambda - center)^2 + \Gamma^2}$$

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

calculate(*values*)
This method should be overriden in child classes.

Parameters:

values: list of control parameters

Return:

numpy array of contribution values at specified wavelengths

class `mstm_studio.contributions.`**`LorentzPeak`**(*wavelengths=[]*, *name='ExtraContrib'*)

Lorentz function

$$L(\lambda) = \frac{scale}{(\lambda - \mu)^2 + \Gamma^2}$$

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

`calculate`(*values*)

Parameters:

values: list of control parameters *scale*, *mu* and *Gamma*

Return:

numpy array

class `mstm_studio.contributions.`**`GaussPeak`**(*wavelengths=[]*, *name='ExtraContrib'*)

Gauss function

$$G(\lambda) = scale \cdot \exp\left(-\frac{(\lambda - \mu)^2}{2\sigma^2}\right)$$

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

`calculate`(*values*)

Parameters:

values: list of control parameters *scale*, *mu* and *sigma*

Return:

numpy array

class `mstm_studio.contributions.`**`MieSingleSphere`**(*wavelengths=[]*, *name='ExtraContrib'*)

Mie contribution from single sphere.

Details are widely discusses, see, for example [Kreibig_book1995]

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

`calculate`(*values*)

Parameters:

values: list of control parameters *scale*, *diameter*

Return:

extinction efficiency array of Mie sphere

set_material (*material*, *matrix=1.0*)

Define the material of sphere and environment

Parameters:

material: Material object material of the sphere

matrix: float, string or Material object material of the environment

Return:

True if properties were changed, False - otherwise.

class mstm_studio.contributions.**MieLognormSpheres** (*wavelengths=[]*, *name='ExtraContrib'*)

Mie contribution from an ensemble of spheres with sizes distributed by Log-Normal law

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

calculate (*values*)

Parameters:

values: list of control parameters *scale*, *mu* and *sigma*

Return:

Mie extinction efficiency of log-normally distributed spheres

lognorm (*x*, *mu*, *sigma*)

The shape of Log-Normal distribution:

$$LN(D) = \frac{1}{D\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log(D)-\mu)^2}{2\sigma^2}\right)$$

plot_distrib (*values*, *fig=None*, *axs=None*)

Plot size distribution

Parameters:

values: list of control parameters

fig: matplotlib figure

axs: matplotlib axes

Return:

filled/created fig and axs objects

set_material (*material*, *matrix=1.0*)

Define the material of sphere and environment

Parameters:

material: Material object material of the sphere

matrix: float, string or Material object material of the environment

Return:

True if properties were changed, False - otherwise.

class mstm_studio.contributions.**MieLognormSpheresCached**(*wavelengths=[], name='ExtraContrib'*)

Mie contribution from an ensemble of spheres with sizes distributed by Lognormal law.

Cached version - use it to speed-up fitting.

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

calculate(*values*)

Parameters:

values: list of control parameters *scale*, *mu* and *sigma*

Return:

Mie extinction efficiency of log-normally distributed spheres

2.3.3 Spheres setup

MSTM code requires explicit setup of the positons and sizes of spheres. There are several classes designed in MSTM-studio to help this setup.

Example: particles aggregate

Script to construct spheres with random sizes placed on a regular grid:

```
from mstm_studio.mstm_spectrum import LogNormalSpheres
from mstm_studio.alloy_AuAg import AlloyAuAg

spheres = LogNormalSpheres(9, 10.0, 0.2, 5., AlloyAuAg(1.))
while spheres.check_overlap():
    print('spheres are overlapping, regenerating...')
    spheres = LogNormalSpheres(9, 10.0, 0.2, 5., AlloyAuAg(1.))
print(spheres.a)
```

Sample output:

```
Box size estimated as: 77.0 nm
Desired number of particles: 9
Number of particles in a box: 8
Resulted number of particles: 8
spheres are overlapping, regenerating...
Box size estimated as: 77.0 nm
Desired number of particles: 9
Number of particles in a box: 8
Resulted number of particles: 8
[ 9.307773    8.61185299  9.92867988  8.84140858  9.87175352  8.71090184
  9.71505038 12.40459688]
```

Classes

class mstm_studio.mstm_spectrum.**Spheres**

Abstract collection of spheres

Object fields:

N: int number of spheres

x, y, z: numpy arrays coordinates of spheres centers

a: list or arrray spheres radii

materials: numpy array Material objects or strings

Creates empty collection of spheres. Use child classes for non-empty!

`append`(*sphere*)
Append by data from SingleSphere object

Parameter:

sphere: SingleSphere

`check_overlap`(*eps=0.001*)
Check if spheres are overlapping

`delete`(*i*)
Delete element with index *i*

`extend`(*spheres*)
Append by all items from object *spheres*

`get_center`(*method=''*)
calculate center of masses in assumption of uniform density

Parameter:

method: string {''|'mass'} If method == 'mass' then center of masses (strictly speaking, volumes) is calculated. Otherwise all spheres are averaged evenly.

`load`(*filename*, *mat_filename='etaGold.txt'*, *units='nm'*)
Reads spheres coordinates and radii from file.

Parameters:

filename: string file to be read from

mat_filename: string all spheres will have this material (sphere-material storaging is not yet implemented)

units: string {'mum'|'nm'} distance units. If 'mum' then coordinated will be scaled (x1000)

`save`(*filename*)
Saves spheres coordinates and radii to file.

Parameter:

filename: string

class `mstm_studio.mstm_spectrum.`**`SingleSphere`**(*x*, *y*, *z*, *a*, *mat_filename='etaGold.txt'*)
Collection of spheres with only one sphere

Parameters:

x, y, z: float coordinates of spheres centers

a: float spheres radii

mat_filename: string, float, complex value or Material object material specification

class mstm_studio.mstm_spectrum.**ExplicitSpheres**(*N=0*, *Xc=[]*, *Yc=[]*, *Zc=[]*, *a=[]*, *mat_filename='etaGold.txt'*)

Create explicitely defined spheres

Parameters:

N: int number of spheres

Xc, Yc, Zc: lists or numpy arrays coordinates of the spheres centers

a: list or numpy array radii of the spheres

mat_filename: string, list of strings, Material or list of Materials specification of spheres material

Note: If only first array Xc is supplied, than all data is assumed zipped in it, i.e.: *Xc = [X1, Y1, Z1, a1, . . . , XN, YN, ZN, aN]*

class mstm_studio.mstm_spectrum.**LogNormalSpheres**(*N*, *mu*, *sigma*, *d*, *mat_filename='etaGold.txt'*)

The set of spheres positioned on the regular mesh with random Log-Normal distributed sizes. In the case overlapping of the spheres the sizes should(?) be regenerated.

Parameters:

N: int number of spheres

mu, sigma: floats parameters of Log-Normal distribution

d: float average empty space between spheres centers

mat_filename: string or Material object specification of spheres material

2.3.4 MSTM run

T-matrix formalism prposed by Waterman [Khlebtsov2013] is one of the generalization of Mie theory towards the multiple spherical targets. The Multi Sphere T-matrix (MSTM) Fortran code is developed by Mischnko and Mackowsky [Mackowski2011]. The *SPR* class implements functionality required for extinction spectra calculation in visible range. Note, that Fortran code have wider functionality, including near field calculations, angle-dependent calculations, etc, which are not currently implemented. Consult the MSTM website <http://eng.auburn.edu/users/dmckwski/scatcodes/> for details.

Example: core-shell particle

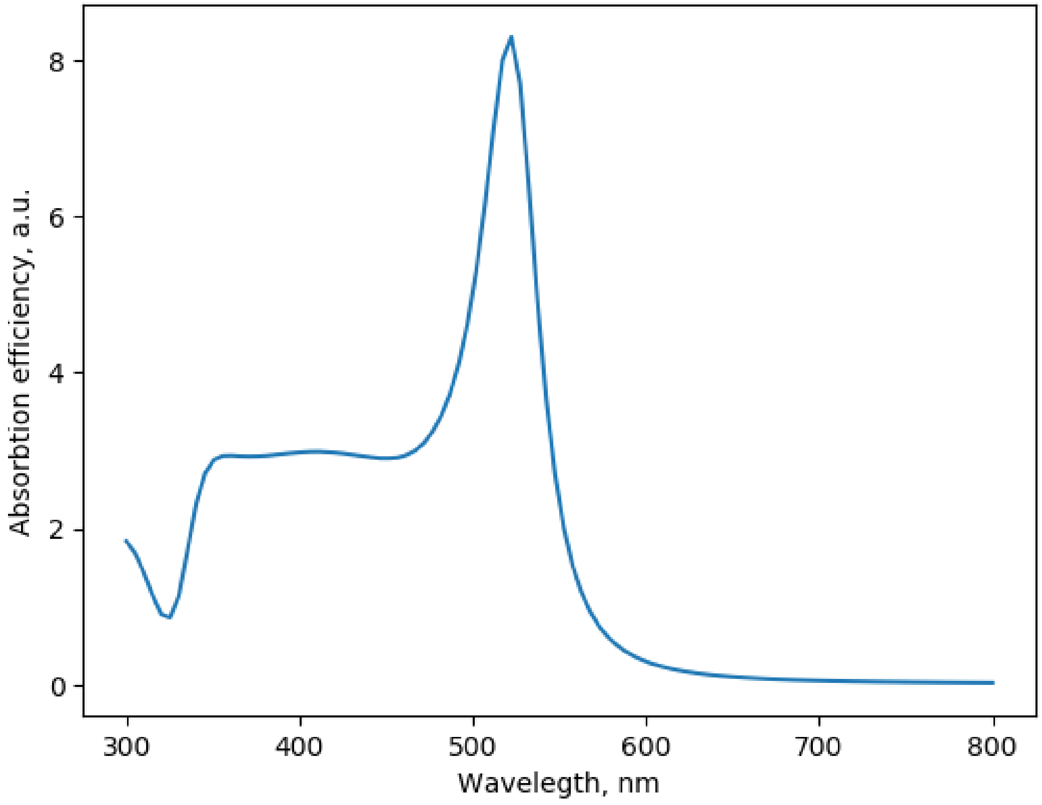
Absorbtion efficiency (normalized cross-section) of gold-silver core-shell particle.

```python
from mstm_studio.alloy_AuAg import AlloyAuAg
from mstm_studio.mstm_spectrum import SPR, ExplicitSpheres
import matplotlib.pyplot as plt
import numpy as np


wls = np.linspace(300, 800, 100)     # define wavelengths
spr = SPR(wls)                       # create SPR object
spr.environment_material = 'glass'   # set matrix

spheres = ExplicitSpheres(2, [0,0,0,10,0,0,0,12], mat_filename=[AlloyAuAg(1.),
↪AlloyAuAg(0.)])
spr.set_spheres(spheres)
spr.set_incident_field(fixed=False)
```

(continues on next page)

(continued from previous page)

```
spr.simulate()

plt.plot(spr.wavelengths, spr.absorbtion)
plt.xlabel('Wavelegth, nm')
plt.ylabel('Absorbtion efficiency, a.u.')
plt.savefig('core-shell_mstm.png', bbox_inches='tight')
```



Class

class mstm_studio.mstm_spectrum.**SPR**(*wavelengths*)

Class for calculation of surface plasmin resonance (SPR), running MSTM external code. The MSTM executable should be set in MSTM_BIN environment variable. Default is ~/bin/mstm.x

Parameter:

wavelengths: numpy array Wavelegths in nm

plot()

Plot results with matplotlib.pyplot

set_incident_field(*fixed=False*, *azimuth_angle=0.0*, *polar_angle=0.0*, *polarization_angle=0.0*)

Set incident wave orientation and polarization

Parameters:

fixed: bool True - fixed orientation and polarized light False - average over all orientations and polarizations

azimuth_angle, polar_angle: float (degrees)

polarization_angle: float (degrees) !sensible only for near field calculation! polarization angle relative to the *k*-*z* palne. 0 - X-polarized, 90 - Y-polarized (if *azimuth* and *polar* angles are zero).

simulate (*outfn=None*)
Start the simulation.

The inpuit parameters are read from object dictionary *paramDict*. Routine will prepare input file *scriptParams.inp* in the temporary folder, which will be deleted after calculation.

After calculation the result depends on the polarization setting. For polarized light the object fields will be filled:

extinction_par, extinction_ort, absorbtion_par, absorbtion_ort, scattering_par, scattering_ort.

While for orientation-averaged calculation just:

extinction, absorbtion and scattering.

write (*filename*)
Save results to file

2.3.5 Visualization of near field

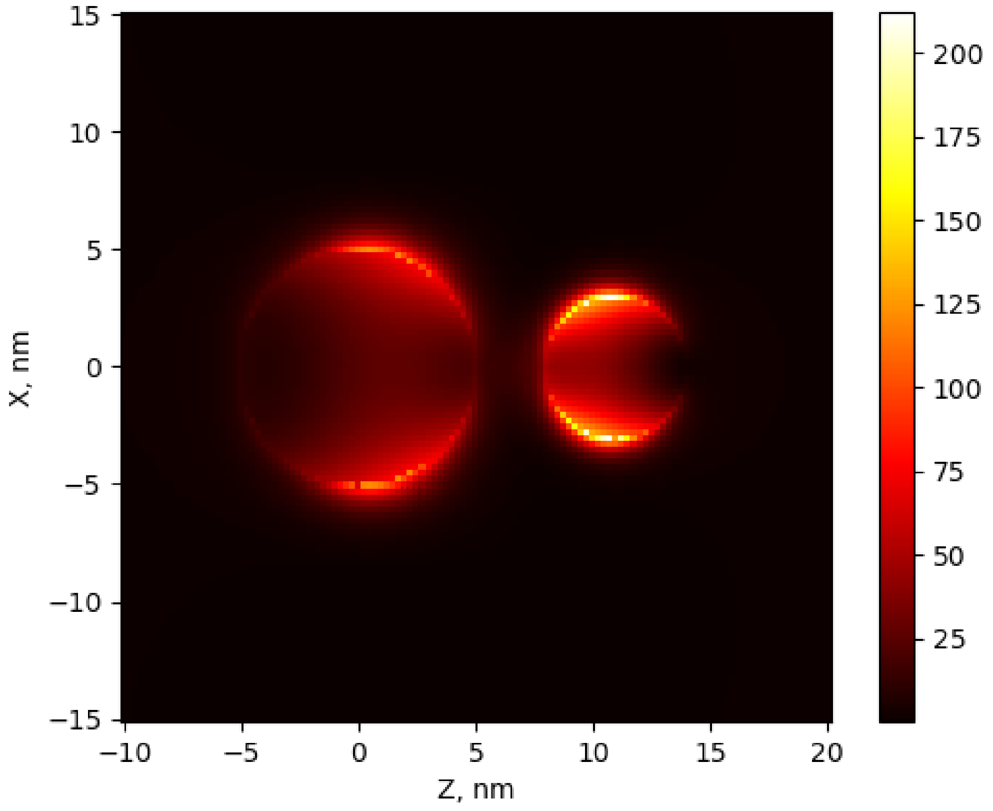
MSTM code can be used to calculate the distribution of the near (or local) field. The field is calculated on a rectangular region, specified by input (`nearfield.NearField.set_plane()`). Currently, only magnititude of electric field $|E|^2$ can be visualized.

Example: field distribution near two particles

Two silver spheres with radii 5 and 3 nm are placed at 0,0,0 and 0,0,11. Incident beam with wavelength 385 nm is directed by Z axis and have X polzarization.

```
from mstm_studio.nearfield import NearField
from mstm_studio.mstm_spectrum import ExplicitSpheres
from mstm_studio.alloy_AuAg import AlloyAuAg

mat = AlloyAuAg(x_Au=0.0)        # silver material
nf = NearField(wavelength=385)   # near the resonance
nf.environment_material = 1.5    # glass matrix
nf.set_plane(plane='xz', hmin=-10, hmax=20, vmin=-15, vmax=15, step=0.25)

spheres = ExplicitSpheres(2, [0, 0, 0, 5, 0, 0, 11, 3],
                          mat_filename=2*[mat])
nf.set_spheres(spheres)

nf.simulate()

nf.plot()
```

Resulting image



Class

class `mstm_studio.nearfield.`**`NearField`**(*wavelength*)

Calculate field distribution map at fixed wavelength

`plot`(*fig=None*, *axs=None*, *caxs=None*)

Show 2D field distribution

Parameters:

fig: matplotlib figure

axs: matplotlib axes

caxs: matplotlib axes for colorbar

Returns: filled/created fig and axs objects

`set_plane`(*plane='zx'*, *hmin=-10.0*, *hmax=10.0*, *vmin=-10.0*, *vmax=10.0*, *step=1.0*, *offset=0.0*)

Determine the plane and grid for near field computation.

plane: one of 'yz'|'zx'|'xy' hmin, hmax, vmin, vmax: horizontal and vertical sizes step: size of the grid grain offset: shift of the plane

`simulate`()

Run MSTM code to produce 2D map of field distribution.

write (*filename*)
save field data to text file

2.3.6 Non-spherical particles

T-matrix formulation allows to perform computational efficient calculations for single axially-symmtric objects. Currently supported only spheroid shape (rotational ellipsoid) using external library ScatterPy <https://github.com/TCvanLeth/ScatterPy>. The details can be found in [Mishchenko1998].

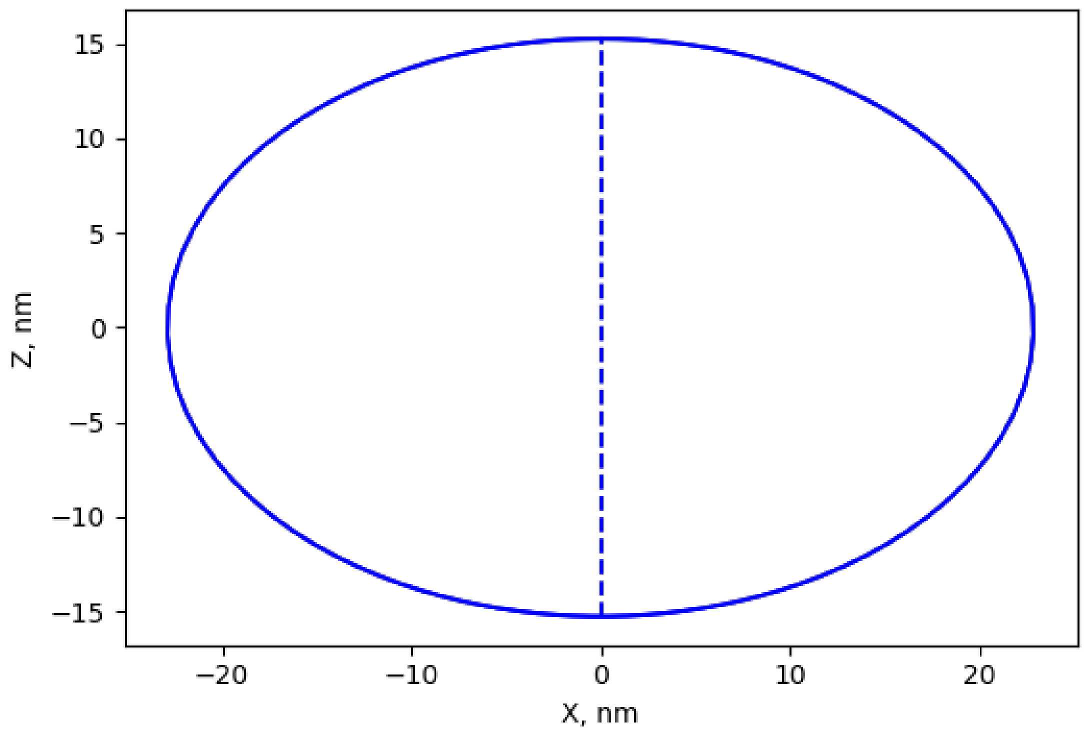
Example

Calculation of extinction for oblate spheroid with aspect ratio of $\alpha = a/c = 1.5$. The size is specified by diamter a_{eff} of equivalent-volume sphere.

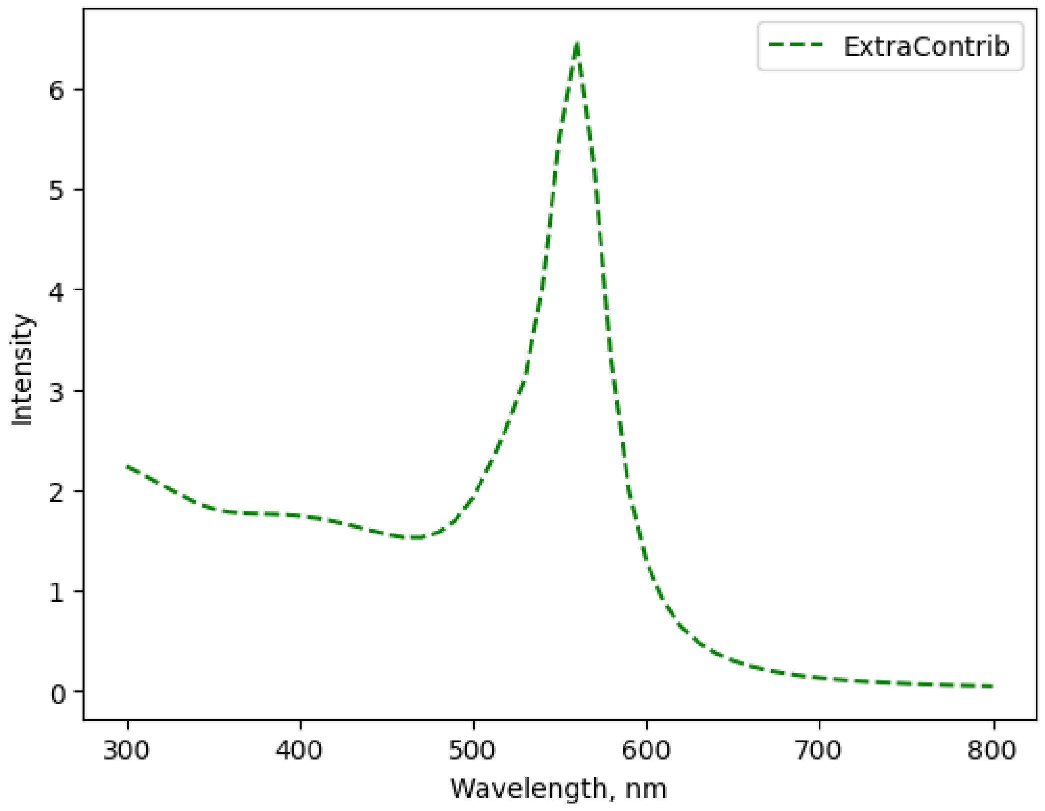
The spheroid size can be derived as

$$a = a_{eff}\alpha^{1/3}$$
$$c = a/\alpha$$

```
from mstm_studio.alloy_AuAg import AlloyAuAg
from mstm_studio.contrib_spheroid import SpheroidSP
import numpy as np


wls = np.linspace(300, 800, 51)  # range for calculation, in nm
SIZE = 20  # nm, particle diameter
ASPECT = 1.5  # a / c = horiz. axis / rot. axis

sph = SpheroidSP(wavelengths=wls)  # create object
sph.set_material(AlloyAuAg(x_Au=1), 1.5)  # particle and matrix refr. ind.
```



```
fig, axs = sph.plot_shape([1, SIZE, ASPECT])
fig.savefig('spheroid_shape.png', bbox_inches='tight')

ext_sph = sph.calculate([1, SIZE, ASPECT])
fig, axs = sph.plot([1, SIZE, ASPECT])  # scale, diameter, aspect
fig.savefig('spheroid_ext.png', bbox_inches='tight')
```



Classes

Contributions to optical extinction spectra from axial-symmetric particles. Currently, spheroids.

class `mstm_studio.contrib_spheroid.`**`SpheroidSP`**(*wavelengths=[]*, *name='ExtraContrib'*)

Extinction from spheroid calculated in T-matrix approach using external library *ScatterPy* <https://github.com/TCvanLeth/ScatterPy>

Parameters:

wavelengths: list or numpy array wavelengths in nm

name: string optional label

`calculate`(*values*)

Parameters:

values: list of parameters *scale*, *size* and *aspect* Scale is an arbitrary multiplier. Size parameter is the radius of equivelent-volume sphere. The aspect ratio is "the ratio of horizontal to rotational axes" according to scatterpy/shapes.py

Return:

extinction efficiency array for spheroid particle

`plot_shape`(*values*, *fig=None*, *axs=None*)

Plot shape profile. Spatial shape is achieved by rotation over vertical axis.

Parameters:

values: list of control parameters *scale*, *size* and *aspect* fig: matplotlib figure

axs: matplotlib axes

Return:

filled/created fig and axs objects

2.3.7 Fitting

Fitting of experimental spectra is a powerful tool for study of plasmonic nanoparticles. Fitting with Mie theory is routinely used to provide information about particle sizes, but fitting with MSTM can solve even agglomerates (packs) of nanoparticles, where Mie theory is not applicable, see [Avakyan2017] for example. Another application is the fitting with core-shell or multi-layered particles.

The MSTM-studio used hard-coded target (penalty) function which is minimized during fitting (ChiSq):

$$\chi^2 = \sum_i \left(y_i^{(\mathrm{fit})} - y_i^{(\mathrm{dat})} \right)^2,$$

where index *i* enumerates wavelengths.

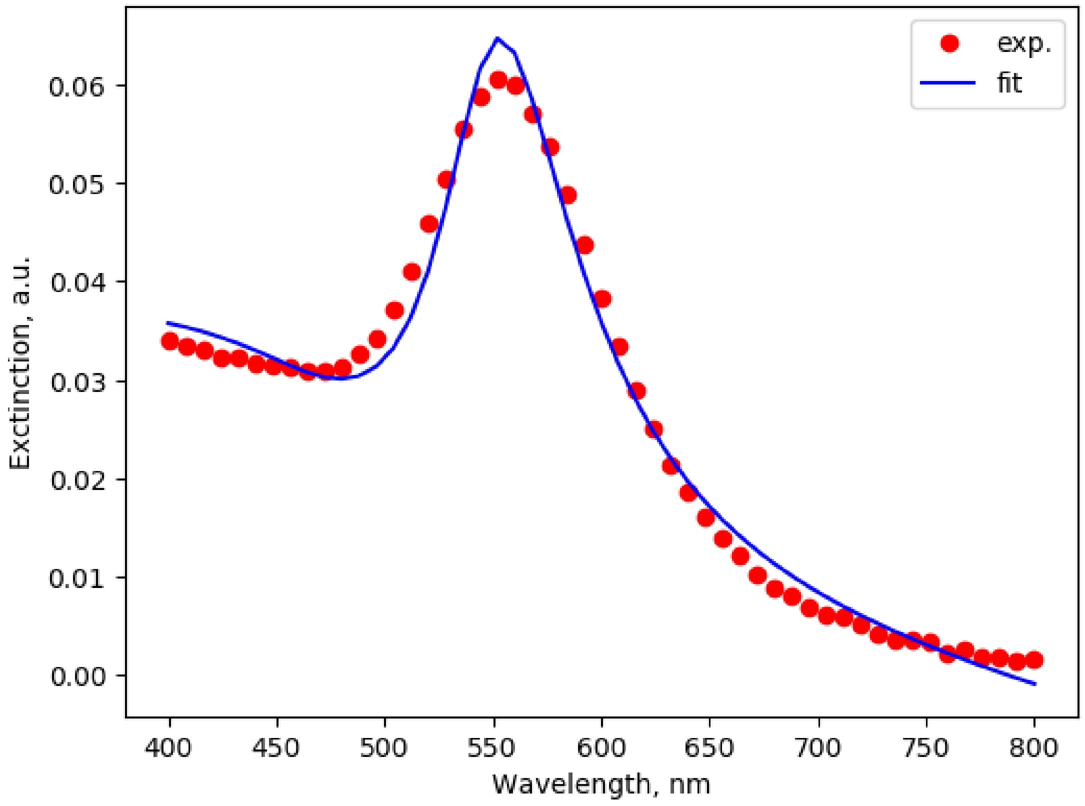
Example: fit with Mie theory

Exampels experimental file, included in distribution, is the extinction spectra of gold particles laser-impregnated in glass, synthesized and studied by Maximilian Heinz [Avakyan2017].

```
from mstm_studio.alloy_AuAg import AlloyAuAg
from mstm_studio.contributions import LinearBackground, MieLognormSpheresCached
from mstm_studio.fit_spheres_optic import Fitter

fitter = Fitter(exp_filename='experiment.dat')        # load experiment from tabbed␣
↪file
fitter.set_extra_contributions(
    [LinearBackground(fitter.wls),                    # wavelengths from experiment
     MieLognormSpheresCached(fitter.wls, 'LN Mie')],  # cached version for faster␣
↪fittting
    [0.02, 0.0001, 0.1, 2.0, 0.4])                    # initial values for a, b, C,␣
↪mu, sigma
fitter.extra_contributions[1].set_material(AlloyAuAg(1.), 1.5)  # gold particles in␣
↪glass
fitter.set_spheres(None)  # no spheres - no slow MSTM runs

# run fit (takes ~20 seconds on 2GHz CPU)
fitter.run()

fitter.report_result()

# plot results
import matplotlib.pyplot as plt
plt.plot(fitter.wls, fitter.exp,  'ro', label='exp.')
plt.plot(fitter.wls, fitter.calc, 'b-', label='fit')
plt.xlabel('Wavelength, nm')
plt.ylabel('Exctinction, a.u.')
plt.legend()
plt.savefig('fit_by_Mie.png', bbox_inches='tight')
```

Output (final part):

```
ChiSq:       0.000219
Optimal parameters
    ext00:  0.035177        (Varied:True)
    ext01:  -0.000049       (Varied:True)
    ext02:  0.007908        (Varied:True)
    ext03:  4.207724        (Varied:True)
    ext04:  0.284066        (Varied:True)
    scale:  7030.322097     (Varied:True)
```



The low value of *ChiSq* and inspecting of agreement between theoretical and experimental curves are indicate on *acceptable* fitting. The names of fitting parameters are explained in Constraints subsection (see `Parameter`). In this example the *ext00* and *ext01* are the parameters *a* and *b* of linear contribution, *ext02* is a scale multiplier for Mie contribution, *ext03* and *ext04* correspond to *mu* and *sigma* parameters of Log-Normal distribution (see `mstm_spectrum.MieLognormSpheres`). The last parameter, the common *scale* multiplier 100 % correlates with *ext02*, resulting in spurious absolute values. If needed, the particle concentration can be estimated from thier product $scale \times ext02$ or by constraining one of them during fitting.

Fitter class

class `mstm_studio.fit_spheres_optic.`**`Fitter`**(*exp_filename*, *wl_min=300*, *wl_max=800*, *wl_npoints=51*, *extra_contributions=None*, *plot_progress=False*)

Class to perform fit of experimental Exctinction spectrum

Field:

tolerance: float stopping criterion, default is 1e-4

Parameters:

exp_filename: str name of file with experimental data

wl_min, wl_max: float wavelength bounds for fitting (in nm).

wl_npoints: int number of wavelengths where spectra will be calcualted and compared.

extra_contributions: list of Contribution objects If *None*, then ConstantBackground will be used. Assuming that first element is a background. If you don't want any extra contribution, set to empty list *[]*.

plot_progress: bool Show fitting progress using matplotlib. Should be turned off when run on parallel cluster without gui.

`add_constraint`(*cs*)
Adds constraints on the parameters. Usefull for the case of core-shell and layered structures.

Parameter:

cs: Contraint object or list of Contraint objects

`get_extra_contributions`()
Return a list of current extra contributions to the spectrum

`report_freedom`()
Returns string with short summary before fitting

`report_result`(*msg=None*)
Returns string with short summary of fitting results

`run`(*maxsteps=400*)
Start fitting.

Parameters:

maxsteps: int limits number of steps performed

`set_callback`(*func*)
Set callback function which will be called on each step of outer optimization loop.

Parameter:

func: function(values) where values – list of values passed from optimization routine

`set_extra_contributions`(*contributions*, *initial_values=None*)
Add extra contributions and initialize corresponding params.

Parameters:

contributions: list of Contribution objests

initial_values: float array

`set_matrix`(*material='AIR'*)
set refraction index of matrix material

material [{'AIR'|'WATER'|'GLASS'} or float] the name of material or refraction index value.

`set_spheres`(*spheres*)
Specify the spheres to be fit.

Paramerer:

spheres: list of mstm_spectrum.Sphere objects If *None* then MSTM will not be run.

2.3.8 Constraints

The constraints allow to speed-up or direct the fitting. Thier setup requires specification of variable names, which are described in Parameter class documentation:

class mstm_studio.fit_spheres_optic.**Parameter**(*name*, *value=1*, *min=None*, *max=None*, *internal_loop=False*)

Class for parameter object used for storage of parameter's name, value and variation limits.

Parameter naming conventions:

scale - outer common multiplier

ext%i - extra parameter, like background, peaks or Mie contributions

a%i - sphere radius

x%i, *y%i*, *z%i* - coordinates of sphere center

where *%i* is a number (0, 1, 2, . . .)

Parameters:

name: string name of parameter used for constraints etc

value: float initial value of parameter

min, max: float bounds for parameter variation (optional)

internal_loop [bool] if *True* the parameter will be allowed to vary in internal (fast) loop, which does not require MSTM recalculation. Note: this flag will be removed in future.

varied: bool if *True* – will be changed during fit

Example: fit by core-shell

Fit the same experiment as above, but using model of core-shell particle, just to illustrate the technique.

```
from mstm_studio.alloy_AuAg import AlloyAuAg
from mstm_studio.contributions import LinearBackground, MieLognormSpheresCached
from mstm_studio.mstm_spectrum import ExplicitSpheres, Profiler
from mstm_studio.fit_spheres_optic import Fitter, FixConstraint, ConcentricConstraint

fitter = Fitter(exp_filename='experiment.dat')         # load experiment from tabbed␣
↪file
fitter.set_extra_contributions(
    [LinearBackground(fitter.wls)],       # wavelengths from experiment
    [0.02, 0.0001])                       # initial values for a, b

spheres = ExplicitSpheres(2, [0,0,0,10,0,0,0,12], mat_filename=[AlloyAuAg(1.),
↪AlloyAuAg(0.)])
fitter.set_spheres(spheres) # core-shell Au@Ag particle
fitter.set_matrix(1.5)      # in glass

fitter.add_constraint(ConcentricConstraint(0, 1))   # 0 -> 1
fitter.add_constraint(FixConstraint('x00'))
fitter.add_constraint(FixConstraint('y00'))
fitter.add_constraint(FixConstraint('z00'))

# run fit (takes ~200 seconds on 2GHz CPU)
with Profiler():
```

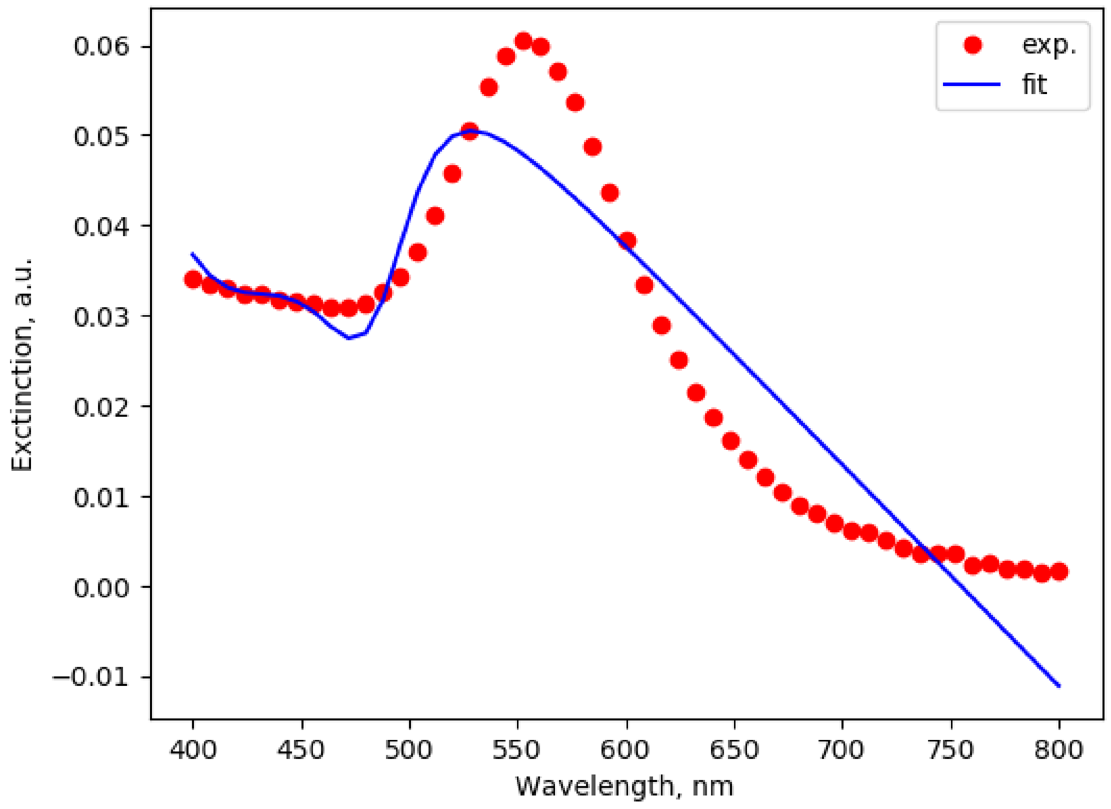
(continues on next page)

(continued from previous page)

```
    fitter.run()

fitter.report_result()

# plot results
import matplotlib.pyplot as plt
plt.plot(fitter.wls, fitter.exp,  'ro', label='exp.')
plt.plot(fitter.wls, fitter.calc, 'b-', label='fit')
plt.xlabel('Wavelength, nm')
plt.ylabel('Exctinction, a.u.')
plt.legend()
plt.savefig('fit_by_core-shell.png', bbox_inches='tight')
```

Output (final part):

```
ChiSq:      0.002354
Optimal parameters
        a00:        1.284882        (Varied:True)
        a01:        1.958142        (Varied:True)
        ext00:      0.186312        (Varied:True)
        ext01:      -0.000247       (Varied:True)
        scale:      -0.063814       (Varied:True)
        x00:        0.000000        (Varied:False)
        x01:        0.000000        (Varied:False)
        y00:        0.000000        (Varied:False)
        y01:        0.000000        (Varied:False)
        z00:        0.000000        (Varied:False)
        z01:        0.000000        (Varied:False)
```



The fiting quality demonstrated by parameter ChiSq is ~10 times worse comparing when used the ensemble of non-interacting gold particles. The figure shows unacceptable fitting quality too.

Constraints classes

class `mstm_studio.fit_spheres_optic.`**`Constraint`**

Abstract constraint class. All other should inherit from it.

`apply`(*params*)

Modify the params dict according to given constranint algorithm.

Note: Abstract method!

class `mstm_studio.fit_spheres_optic.`**`FixConstraint`**(*prm*, *value=None*)

Fix value of parameter with name *prm* to *value*.

Parameters:

prm: string parameter name

value: float if *None* than initial value will be used.

`apply`(*params*)

Apply fix constraint

class `mstm_studio.fit_spheres_optic.`**`EqualityConstraint`**(*prm1*, *prm2*)

Fix two parameters with names *prm1* and *prm2* being equal

`apply`(*params*)

Apply equality constraint

class `mstm_studio.fit_spheres_optic.`**`ConcentricConstraint`**(*i1*, *i2*)

Two spheres with common centers.

i1 and *i2* – indexes of spheres

`apply`(*params*)

Apply concentric constraint

class `mstm_studio.fit_spheres_optic.`**`RatioConstraint`**(*prm1*, *prm2*, *ratio=1*)

Maintain ratio of two variables, *prm1*/*prm2* = *ratio*

`apply`(*params*)

Apply Ratio constraint

`set_ratio`(*ratio*)

Set ratio of $prm1/prm2 = ratio$.

CHAPTER 3

Contacts

GitHub: https://github.com/lavakyan/mstm-spectrum

E-mail: laavakyan_at_sfedu.ru

Bibliography

[Rioux2014] D. Rioux, S. Vallières, S. Besner, P. Muñoz, E. Mazur, and M. Meunier, “An Analytic Model for the Dielectric Function of Au, Ag, and their Alloys” Adv. Opt. Mater. (2014) *2* 176-182 <http://dx.doi.org/10.1002/adom.201300457>

[Kreibig_book1995] U. Kreibig, M. Vollmer, “Optical Properties of Metal Clusters” (1995) 553

[Khlebtsov2013] N. Khlebtsov, “T-matrix method in plasmonics: An overview” J. Quant. Spectrosc. Radiat. Transfer (2013) *123*, 184-217, Peter C. Waterman and his scientific legacy

[Mackowski2011] D. Mackowski, M. Mishchenko,”A Multiple Sphere T-matrix Fortran Code for Use on Parallel Computer Clusters” J. Quant. Spectrosc. Radiat. Transfer (2011) *112*, 2182–2192

[Mishchenko1998] M. Mishchenko, L. Travis, “Capabilities and limitations of a current FORTRAN implementation of the T-matrix method for randomly oriented, rotationally symmetric scatterers ” (1998) 309

[Avakyan2017] L. Avakyan, M. Heinz, A. Skidanenko, K. Yablunovskiy, J. Ihlemann, J. Meinertz, C. Patzig, M. Dubiel, L. Bugaev “Insight on agglomerates of gold nanoparticles in glass based on surface plasmon resonance spectrum: Study by multi-spheres T-matrix method” J. Phys.: Condens. Matter (2018) *30*, 045901-045909 <https://doi.org/10.1088/1361-648X/aa9fcc>

Python Module Index

m

mstm_studio.contrib_spheroid, 37
mstm_studio.contributions, 25

Index

A

add_constraint() (*mstm_studio.fit_spheres_optic.Fitter method*), 40

AlloyAuAg (*class in mstm_studio.alloy_AuAg*), 21

append() (*mstm_studio.mstm_spectrum.Spheres method*), 30

apply() (*mstm_studio.fit_spheres_optic.ConcentricConstraint method*), 43

apply() (*mstm_studio.fit_spheres_optic.Constraint method*), 43

apply() (*mstm_studio.fit_spheres_optic.EqualityConstraint method*), 43

apply() (*mstm_studio.fit_spheres_optic.FixConstraint method*), 43

apply() (*mstm_studio.fit_spheres_optic.RatioConstraint method*), 43

C

calculate() (*mstm_studio.contrib_spheroid.SpheroidSP method*), 37

calculate() (*mstm_studio.contributions.ConstantBackground method*), 26

calculate() (*mstm_studio.contributions.Contribution method*), 25

calculate() (*mstm_studio.contributions.GaussPeak method*), 27

calculate() (*mstm_studio.contributions.LinearBackground method*), 26

calculate() (*mstm_studio.contributions.LorentzBackground method*), 26

calculate() (*mstm_studio.contributions.LorentzPeak method*), 27

calculate() (*mstm_studio.contributions.MieLognormSpheres method*), 28

calculate() (*mstm_studio.contributions.MieLognormSpheresCached method*), 29

calculate() (*mstm_studio.contributions.MieSingleSphere method*), 27

check_overlap() (*mstm_studio.mstm_spectrum.Spheres method*), 30

ConcentricConstraint (*class in mstm_studio.fit_spheres_optic*), 43

ConstantBackground (*class in mstm_studio.contributions*), 26

Constraint (*class in mstm_studio.fit_spheres_optic*), 43

Contribution (*class in mstm_studio.contributions*), 25

D

delete() (*mstm_studio.mstm_spectrum.Spheres method*), 30

E

EqualityConstraint (*class in mstm_studio.fit_spheres_optic*), 43

ExplicitSpheres (*class in mstm_studio.mstm_spectrum*), 30

extend() (*mstm_studio.mstm_spectrum.Spheres method*), 30

F

Fitter (*class in mstm_studio.fit_spheres_optic*), 39

FixConstraint (*class in mstm_studio.fit_spheres_optic*), 43

G

GaussPeak (*class in mstm_studio.contributions*), 27

get_center() (*mstm_studio.mstm_spectrum.Spheres method*), 30

get_extra_contributions() (*mstm_studio.fit_spheres_optic.Fitter method*), 40

get_gamma_corr() (*mstm_studio.diel_size_correction.SizeCorrectedM method*), 23

L

LinearBackground (*class in mstm_studio.contributions*), 26

load() (*mstm_studio.mstm_spectrum.Spheres method*), 30
lognorm() (*mstm_studio.contributions.MieLognormSpheres method*), 28
LogNormalSpheres (*class in mstm_studio.mstm_spectrum*), 31
LorentzBackground (*class in mstm_studio.contributions*), 26
LorentzPeak (*class in mstm_studio.contributions*), 27

M

Material (*class in mstm_studio.mstm_spectrum*), 19
MieLognormSpheres (*class in mstm_studio.contributions*), 28
MieLognormSpheresCached (*class in mstm_studio.contributions*), 28
MieSingleSphere (*class in mstm_studio.contributions*), 27
mstm_studio.contrib_spheroid (*module*), 37
mstm_studio.contributions (*module*), 25

N

NearField (*class in mstm_studio.nearfield*), 34

P

Parameter (*class in mstm_studio.fit_spheres_optic*), 41
plot() (*mstm_studio.contributions.Contribution method*), 25
plot() (*mstm_studio.mstm_spectrum.Material method*), 20
plot() (*mstm_studio.mstm_spectrum.SPR method*), 32
plot() (*mstm_studio.nearfield.NearField method*), 34
plot_distrib() (*mstm_studio.contributions.MieLognormSpheres method*), 28
plot_shape() (*mstm_studio.contrib_spheroid.SpheroidSP method*), 37

R

RatioConstraint (*class in mstm_studio.fit_spheres_optic*), 43
report_freedom() (*mstm_studio.fit_spheres_optic.Fitter method*), 40
report_result() (*mstm_studio.fit_spheres_optic.Fitter method*), 40
run() (*mstm_studio.fit_spheres_optic.Fitter method*), 40

S

save() (*mstm_studio.mstm_spectrum.Spheres method*), 30
set_callback() (*mstm_studio.fit_spheres_optic.Fitter method*), 40
set_extra_contributions() (*mstm_studio.fit_spheres_optic.Fitter method*), 40
set_incident_field() (*mstm_studio.mstm_spectrum.SPR method*), 32
set_material() (*mstm_studio.contributions.MieLognormSpheres method*), 28
set_material() (*mstm_studio.contributions.MieSingleSphere method*), 27
set_matrix() (*mstm_studio.fit_spheres_optic.Fitter method*), 40
set_plane() (*mstm_studio.nearfield.NearField method*), 34
set_ratio() (*mstm_studio.fit_spheres_optic.RatioConstraint method*), 43
set_spheres() (*mstm_studio.fit_spheres_optic.Fitter method*), 40
set_wavelengths() (*mstm_studio.contributions.Contribution method*), 26
simulate() (*mstm_studio.mstm_spectrum.SPR method*), 33
simulate() (*mstm_studio.nearfield.NearField method*), 34
SingleSphere (*class in mstm_studio.mstm_spectrum*), 30
SizeCorrectedGold (*class in mstm_studio.diel_size_correction*), 22
SizeCorrectedMaterial (*class in mstm_studio.diel_size_correction*), 23
SizeCorrectedSilver (*class in mstm_studio.diel_size_correction*), 22
Spheres (*class in mstm_studio.mstm_spectrum*), 29
SpheroidSP (*class in mstm_studio.contrib_spheroid*), 37
SPR (*class in mstm_studio.mstm_spectrum*), 32

W

write() (*mstm_studio.mstm_spectrum.SPR method*), 33
write() (*mstm_studio.nearfield.NearField method*), 34