MULASASS

Multi-**L**ayered **S**emi-**A**nalytic **S**cattering matrix **S**olver

Documentation, Release 0.6

Björn Sturmberg
Kokou Dossou

October 29, 2013

Brought to you with the support of,

The Australian Renewable Energy Agency,
CUDOS, an Australian Research Council Center of Excellence,
School of Physics, The University of Sydney,
School of Mathematical Sciences, University of Technology Sydney,
National Computational Infrastructure, Australia.

Contents

Chapter 1

Introduction

1.1 Description of MULASASS

MULASASS calculates the scattering matrices of a multi-layered structure, where each layer can be homogeneous or structured (down to sub-wavelength dimensions) and the materials may have complex, dispersive refractive indices. The scattering matrices are powerful tools from which many physical quantities, such as the total transmission, absorption in each layer, and the resonances of the structure, can be derived.

An advantage of MULASASS over other scattering matrix programs (for example CAMFR) is that the fields in each layer are considered in their natural basis with transmission scattering matrices converting fields between them. The fields in homogeneous layers are therefore expressed in terms of plane waves, while the natural basis in the periodically structured layers are Bloch modes. Expressing fields in their natural basis gives the terms of the scattering matrices intuitive meaning, providing access to greater physical insights. It is also advantages for the speed and accuracy of the numerical method.

Inherent to the scattering matrix approach is the requirement that the interfaces between layers be planar, *ie.* that each layer is uniform in one direction (here labelled z). In this nomenclature the incident field must have $k_z = k_\perp \neq 0$, but is unconstrained in $k_\parallel = k_{x,y}$. In our implementation, the only constraint placed on each layer in the x-y plane, is that it must be periodic, at least at the supercell level. This is because the modes of structured media are calculated using a vectorial Finite Element Method (FEM) routine with periodic boundary conditions. The scattering matrices of homogeneous media are calculated analytically resulting in excellent accuracy and speed.

MULASASS has been designed to handle lossy media with dispersive refractive indices, with the complex refractive index at each frequency being taken directly from tabulated results of experimental measurements. This is an advantage of frequency domain methods over time domain methods such as the Finite Difference Time Domain (FDTD) where refractive indices are included by analytic approximations such as the Drude model (for example MEEP). It is also possible to include media with lossless and/or non-dispersive refractive indices in MULASASS .

Taking full advantage of the boundary-element nature of the scattering matrix method it is possible to vary the thickness of a layer by a single, numerically inexpensive, matrix multiplication. Furthermore, MULASASS recognises when interfaces are repeated so that their scattering matrices need not be recalculated but rather just retrieved from memory, which takes practically no computation time.

By integrating a 2D finite element method calculation into the scattering matrix method

MULASASS provides a powerful, versatile tool for nanophotonic simulations that are both computationally efficient and physically insightful.

MULASASS is a completely open source package, utilising free, open source compilers, meshing programs and libraries! The low-level numerical routines are written in Fortran for optimal performance, while higher-level processing is done in python.

In summary, the advantages of MULASASS are;

- Calculates the scattering matrices between layers in their natural bases, for maximum physical insights.
- Designed to include lossy, dispersive materials, with frequency resolved (experimentally measured) refractive indices.
- FEM allows for arbitrary in-plane geometries, down to the periodicity of the supercell.
- Homogeneous layers are calculated analytically for optimal accuracy and speed.
- The scattering matrix method efficiently combines arbitrary number of layers into a stack.
- Synthesis of efficient Fortran routine with dynamic, high-level programming in Python.
- Completely open source package! Including FEM meshing program, Fortran FEM routine, Python multi-layered scattering matrix implementation.
- Integrated with highly optimised libraries (but also functions without these at slower speeds), including; BLAS, LAPACK, ARPACK, UMFPACK
- Vectorial FEM advantages?
- Get band structure at same time as t,r,a.
- Both/all polarasations at once

Chapter 2

Installation

MULASASS has been developed for use on Linux and Unix-like operating systems. It may be easily ported to other operating systems, but there are no current plans for doing so. If you are willing and able to do so, please get in contact! For now we take you through the installation process on Linux and Unix-like operating systems

2.1 GitHub Repository

By reading this document you have almost certainly found the GitHub repository of MULASASS . If not, go there now!

2.2 Compilers

MULASASS is written in Python and Fortran, both of which can be compiled with open-source compilers. The Fortran components (MULASASS source code and libraries) have been successfully compiled with intel's ifortran as well as open-source gfortran. The Python components requires NumPy and SciPy to be installed, preferably as part of Endthought, which is free for academic use.

2.3 Easy installation

On Ubuntu the following may be installed from the 'Ubunutu software centre'

```
libsuitesparse-metis-3.1.0
libsuitesparse-metis-dev
```

In which case you can skip down to installing ARPACK in Sect. 2.5.

2.4 SuiteSparse

The FEM routine used in MULASASS makes use of the highly optimised UMFPACK (Unsymmetric MultiFrontal Package) direct solver for sparse matrices developed by Prof. Timothy A. Davis. This is distributed as part of the SuiteSparse libraries under the GPL license.

1. Download SuiteSparse from https://www.cise.ufl.edu/research/sparse/SuiteSparse/

2. Unpack SuiteSparse into EMUstack/Fortran_EMUstack/, it should create a directory there, SuiteSparse/ mkdir where you want SuiteSparse installed, in my case EMUstack/Fortran_EMUstack/SS_installed

```
$ mkdir SS_installed/lib SS_installed/include
```

edit SuiteSparse/SuiteSparse_config/SuiteSparse_config.mk for consistency across the whole build, use intel fortran compiler

```
line 75 F77 = gfortran --> ifort
```

set path to install folder

```
line 85 INSTALL_LIB =
    /suphys/bjorn/Usyd_Running/f2pytest/EMUstack/Fortran_EMUstack/SS_install/lib
line 86 INSTALL_INCLUDE =
    /suphys/bjorn/Usyd_Running/f2pytest/EMUstack/Fortran_EMUstack/SS_install/include
```

line 290ish commenting out all other references to these

```
F77 = ifort
CC = icc
BLAS  = -L/apps/intel-ct/12.1.9.293/mkl/lib/intel64 -lmkl_rt
LAPACK = -L/apps/intel-ct/12.1.9.293/mkl/lib/intel64 -lmkl_rt
```

```
$ mkdir the INSTALL_LIB and INSTALL_INCLUDE dirs
```

3. Download metis-4.0 http://glaros.dtc.umn.edu/gkhome/fsroot/sw/metis/OLD Unpack metis into SuiteSparse/

```
$ cd SuiteSparse/metis-4.0
```

optionally edit metis-4.0/Makefile.in as per SuiteSparse/ README.txt plus with -fPIC

```
CC = gcc
or
CC = icc
OPTFLAGS = -O3 -fPIC
```

```
$ make
$ cp f2pytest/EMUstack/Fortran_EMUstack/SuiteSparse/metis-4.0/libmetis.a
    f2pytest/EMUstack/Fortran_EMUstack/SS_install/lib/
```

4. in SuiteSparse/

```
$ make library
$ make install
$ cd SuiteSparse/UMFPACK/Demo
$ make fortran64
$ cp SuiteSparse/UMFPACK/Demo/umf4_f77zwrapper64.o into SS_install/lib/
```

Copy the libraries into EMUstack/Fortran_EMUstack/Lib/ so that EMUstack/ is a complete package that can be moved across machine without alteration.

```
$ cp SS_install/lib/*.a EMUstack/Fortran_EMUstack/Lib/
$ cp SS_install/lib/umf4_f77zwrapper64.o EMUstack/Fortran_EMUstack/Lib/
```

2.5 ARPACK

1. IF you have made SuiteSparse yourself, then edit EMUstack/Fortran_EMUstack/zarpack_util.f specifying whether or not your fortran compiler has ETIME is a built in function of your compiler (INTRINSIC). On lines 768-771,

```
C     gfortran likes the following
C      INTRINSIC        ETIME
C     ifort likes the following
      EXTERNAL         ETIME
```

   ELSE IF, when running MULASASS there are errors involving zneupd or znaupd then

```
$ mv zarpack.f zarpack.f.bak
$ mv zarpack_util.f zarpack_util.f.bak
```

   and build ARPACK as described below.

2. ELSE IF you are using Ubuntu package versions of SuiteSparse, then there is some issue with zarpack_util.f You will need to build ARPACK yourself. Download it here http://www.caam.rice.edu/software/ARPACK/download.html (choosing the stable arpack96 version). Unpack into same directory as SuiteSparse/ in my case Fortran_EMUstack/ edit ARmake.inc line 28

```
home = $(HOME)/ARPACK -->
    /suphys/bjorn/Usyd_Running/f2pytest/EMUstack/Fortran_EMUstack/ARPACK
line 104 FC     = f77 --> ifort or gfort
line 105 FFLAGS = -O -cg89 --> -O -fPIC
line 115 MAKE   = /bin/make --> /usr/bin/make
```

```
$ make lib
```

2.6 Make MULASASS

Edit Fortran_EMUstack/Makefile to reflect what compiler you are using and how you installed the libraries. The Makefile has further details.

The installation of MULASASS is complete. However you almost certainly wish to have the freedom to create FEM mesh for your chosen parameters. To do this you will need to install the meshing program Gmsh and compile a simple Fortran routine.

2.7 Install Gmsh and gmsh_conversion.f

1. Install Gmsh,
   - IF using Ubuntu, then install Gmsh from the 'Ubunutu software centre'. Done.
   - ELSE,
     (a) Download cmake sourse folder, unzip/tar, install.
     (b) Download a stable version of Gmsh source code from http://geuz.org/gmsh/. To date, MULASASS has been successfully used with Gmsh versions 2.5.1, 2.6.1
     (c) Unzip/tar gmsh source code.
     (d) Follow gmsh_source/README.txt, with the following options to avoid non-standard libraries (only required for GUI anyway) and to install it into an appropriate directory (one you have write access to);

```
$ cmake -DENABLE\_FLTK=0 -DCMAKE\_INSTALL\_PREFIX=/opt ..
```

     (e) To test the installation, run

```
$ which gmsh
```

     which should give the path to the installation.
     (f) Gmsh can be launched by executing

```
$ gmsh
```

2. Compile the gmsh_conversion.f Fortran routine,

```
$ cd EMUstack/Data/gmsh_conversion/
```

   Edit Makefile, selecting your Fortran compiler on line 10.

```
# FC = gfortran
FC = ifort
```

   then,

```
$ make
```

3. It is also recommended to install Gmsh on your personal computer so that field plots may be opened locally. Versions of Gmsh can be downloaded for Windows, Mac or Linux.

2.7.1 Supercomputer example

As an example, the following process was used on the NCI supercomputer Raijin;

1. If there is a file 'cmake' in gmsh-source/build/ remove/rename it.
2. Load the cmake module,

```
$ module load cmake
```

3. Make sure the mkl module is loaded.

4. In the CMakeLists.txt file in the base directory, make the following changes in the section

```
if(ENABLE_BLAS_LAPACK)
...
    elseif(${CMAKE_SYSTEM_NAME} MATCHES "Linux")
       if(HAVE_64BIT_SIZE_T)
           set(MKL_PATH lib/em64t)
```

   change (the 2 occurrences of)

   set(MKL_PATH lib/em64t) → set(MKL_PATH mklroot/lib/intel64)

   where mklroot is the output of the command 'echo $MKLROOT' i.e. the absolute path the the MKL libraries (CMake can't/won't read environment variables).

   Still under 'elseif($CMAKE_SYSTEM_NAME MATCHES "Linux")', change

   set(MKL_LIBS_REQUIRED mkl_gf_lp64 iomp5 mkl_gnu_thread mkl_core guide pthread) → set(MKL_LIBS_REQUIRED mkl_core mkl_sequential mkl_intel_lp64)

   and

   find_all_libraries(LAPACK_LIBRARIES MKL_LIBS_REQUIRED "" $MKL_PATH) → find_all_libraries(LAPACK_LIBRARIES MKL_LIBS_REQUIRED $MKL_PATH "")

5. On using the cmake command you should see,

```
Found Blas(IntelMKL)
Found Lapack(IntelMKL)
```

6. Run cmake to create the appropriate makefile for the machine, and install gmsh.

```
$ cmake -DENABLE_FLTK=0 -DCMAKE_INSTALL_PREFIX=/home/562/bxs562/gmsh ..
$ make
$ make install
```

7. Lastly, add gmsh to your path, In .cshrc file (.bashrs will be similar but different) add the line,

```
set path=( /home/562/bxs562/gmsh/bin $path )
```

The installation of MULASASS is now fully complete!!!

2.8 Test the installation

While the installation is complete, you now probably want to check that everything has gone as planned... To do this (and to check sanity check any changes made in the future) MULASASS comes with multiple test calculations.

1. MULASASS ships with 2 sets of tests; one to test an installation on a new machine, and another to test against when making modifications on the same machine. These tests are located in test_installation_EMUstack/ and test_local_EMUstack/ respectively. The actual test calculations performed in these sets are identical, and on downloading the reference data tested against is also identical.

   This structure has been designed in response to installations on different machines giving slightly different results, particularly when different versions of gmsh have been used. To test a new installation please first run the tests in test_installation_EMUstack/. To do this

   ```
   $ cd test_installation_EMUstack/
   $ nosetests
   ```

   During testing, individual test results are displayed with . = pass F = FAIL.

2. Once you are satisfied with you installation and the test_installation_EMUstack/ tests have all passed, update the test reference data in test_local_EMUstack/. To do this uncomment the line beginning with testing.save_reference_data in test_local_EMUstack/test_case*.py and then

   ```
   $ cd test_local_EMUstack/
   $ nosetests
   ```

   and then re-comment those lines. Note that the test itself will fail giving the following message; "AssertionError: Reference results saved successfully, but tests will now pass trivially so let's not run them now."

3. When making updates and modifications on MULASASS you should check that your results are still consistent with the test cases to the accuracy as set in test_local_EMUstack/. To do this run

   ```
   $ cd test_local_EMUstack/
   $ nosetests
   ```

   with the testing.save_reference_data line commented out.

Enjoy MULASASS !!!

Chapter 3

Use

3.1 Python + Fortran

MULASASS is a synthesis of Python and Fortran. These are used for high-level, dynamic, object oriented programming, and lower-level, highly efficient, robust programming respectively. This coding philosophy was inspired by BlochCode and is shared by many other open-source projects such as CAMFR and MRCWA.

3.2 Templates

MULASASS comes with multiple template simulation scripts. These are located in 000-simmo_templates/ These are ready to be executed, which is done by running a simmo.py file in the command line, eg.

```
$ cd 000-simmo_templates/
$ python simo_template-grating.py
```

The template files are comprehensively commented, which we hope makes them self-explanatory. For now we just highlight the parameters most often changed.

3.3 Setup

Right at the top, we can briefly describe the simulation that we are creating. We do this by (mis-)using the python docstring format, eg.

```
"""
Simulating the coupling of normally incident light into evanescent orders through a
metallic grating of period 120 nm. Included 3 PW orders.
"""
```

Next we need to specify the number of CPUs the simulation is to have access to. This is set by the num_cores parameter. Alternatively you can also specify the number of CPUs to leave free.

```
# Number of CPUs to use im simulation
num_cores = 5
# # Alternatively specify the number of CPUs to leave free on machine
# leave_cpus = 4
```

```
# num_cores = mp.cpu_count() - leave_cpus
```

3.4 Light

The properties of the incident light are specified in the 'Light' class within objects.py. If considering multiple wavelengths (or multiple k vectors) we create a list of Light object instances. The simulation will be carried out for each of these Light objects (in parallel if num_cores $>$ 1). Each Light instance has the following properties;

```
class Light(object):
    """ Represents the light incident on structure.

        Incident angles may either be specified by 'k_parallel' or by
        incident angles 'theta' and 'phi', together with the refractive
        index 'n_inc' of the incident medium.

        'wl_nm' and 'k_pll' are both in unnormalised units.

        INPUTS:

        - 'wl_nm'       : Wavelength, in nanometers.

        - 'max_order_PWs' : Maximum plane wave order to include.

        - 'k_parallel'  : The wave vector components (k_x, k_y)
            parallel to the interface planes. Units of nm^-1.

        - 'theta'       : Polar angle of incidence in degrees.

        - 'phi'         : Azimuthal angle of incidence in degrees.
    """
```

Note that this doesn't specify the polarisation. This is because every calculation contains both TE and TM polarisations, with the scattering matrices containing terms for scattering from TE $\rightarrow$ TE, TM $\rightarrow$ TM, TE $\rightarrow$ TM, and TM $\rightarrow$ TE. The matrices are structured as,

$$S_{12} = \begin{array}{|c|c|}\hline \text{TE} \rightarrow \text{TE} & \text{TM} \rightarrow \text{TE} \\ \hline \text{TE} \rightarrow \text{TM} & \text{TM} \rightarrow \text{TM} \\ \hline \end{array}$$

where S_{12} can be either a transmission or reflection scattering matrix. We can select the total transmittance and reflectance for a particular polarisation from these later on (see Sect. 3.6).

3.5 Layers

Having defined the properties of the incident light, we now need to define the structure that is being investigated.

MULASASS can calculate the scattering matrices (and derived properties) of an arbitrary number of layers, stacked in arbitrary combinations. For the scattering matrix approach to valid, each layer must be invariant along the axis normal to the plane of its interfaces (here

taken as the x-y plane). In the x-y plane each layer may be composed of an arbitrary collection of media, of arbitrary form. Or a layer may of course be homogeneous in the x-y plane.

MULASASS treats these 2 types of layers, homogeneous and structured, vastly differently. Fundamentally this is because homogeneous layers can be treated analytically, while structured layers must be simulated using numerical techniques (here FEM).

We define each individual, z-invariant, layer as an object. Note that all layers must have a the same periodicity. This is required so that the plane wave diffraction orders are defined consistently throughout a stack. All homogeneous layer must therefore have an artificial periodicity imposed on them.

3.5.1 Homogeneous layers

If a layer is homogeneous it is an instance of the objects.ThinFilm class with the following properties;

```
class ThinFilm(object):
    """ Represents an unstructured homogeneous film.

        INPUTS:

        - 'period'       : Artificial period imposed on homogeneous film
            to give consistently defined plane waves in terms of
            diffraction orders of structured layers.

        - 'height_nm'    : The thickness of the layer in nm or 'semi_inf'
            for a semi-infinte layer.

        - 'material'     : A :Material: instance specifying the n of
            the layer and related methods.

        - 'num_pw_per_pol' : Number of plane waves per polarisation.

        - 'loss'         : If False sets Im(n) = 0, if True leaves n as is.
    """
```

3.5.2 Structured layers

If a layer is structured it is an instance of the objects.NanoStruct class. If the layer is a 1D or 2D grating MULASASS comes with template FEM mesh and will automatically make a mesh to represent your specified geometry if 'make_mesh_now' is set to 'True'. Otherwise a new mesh will have to be made in gmsh and either used directly or set up as a new template file. The mesh files are created at the time a NanoStruc object is initialised.

The other properties of structured layers are as follows;

```
class NanoStruct(object):
    """ Represents a structured layer.

        INPUTS:

        - 'geometry'    : Either 1D or 2D structure; '1D_grating', 'NW_array'.
```

```
- 'period'      : The diameter the unit cell in nanometers.

- 'diameter1'   : The diameter of the inclusion in nm.
- 'diameter2-16' : The diameter of further inclusions in nm.

- 'height_nm'   : The thickness of the layer in nm or 'semi_inf'
    for a semi-infinte layer.

- 'inclusion_a' : A :Material: instance for first inclusion,
    specified as dispersive refractive index (eg. materials.Si_c)
    or nondispersive complex number (eg. Material(1.0 + 0.0j)).

- 'inclusion_b' : " " for the second inclusion medium.
- 'background'  : " " for the background medium.

- 'loss'        : If False, Im(n) = 0, if True n as in :Material: instance.

- 'ellipticity' : If != 0, inclusion has given ellipticity, with b=diameter,
   a=diameter-ellipticity*diameter. NOTE: only implemented for 1 inclusion.

- 'square'      : If True, 'NW_array' has square NWs (ie. 2D grating).

- 'ff'          : The fill fraction of the inclusions. If non-zero,
    the specified diameters are overritten s.t. given ff is achieved,
    otherwise ff is calculated from parameters and stored in self.ff.
- 'ff_rand'     : If True, diameters overritten with random diameters,
    s.t. the ff is as assigned. Must provide non-zero dummy diameters.

- 'posx'        : Shift NWs laterally towards center (each other),
    posx is a fraction of the distance possible before NWs touching.
- 'posy'        : Shift NWs vertically " ".

- 'small_d'     : Distance between 2 inclusions of interleaved 1D grating.

- 'make_mesh_now' : If True, program creates a FEM mesh with provided
    NanoStruct parameters. If False, must provide mesh_file name of
    existing .mail that will be run despite NanoStruct parameters.

- 'force_mesh'  : If True, a new mesh is created despite existance of
    mesh with same parameter. This is used to make mesh with equal
    period etc. but different lc refinement.

- 'mesh_file'   : If using a set premade mesh give its name including
    .mail (eg. 600_60.mail), it must be located in EMUstack/Data/

- 'lc_bkg'      : Length constant of meshing of background medium.
- 'lc2'         : " " on inclusion surfaces. (smaller = finer mesh)
- 'lc3-6'       : " " from center of inclusions.

- 'plot_modes'  : Plot modes (ie. FEM solutions) in gmsh format,
    you get epsilon*|E|^2 & either real/imag/abs of
    x,y,z components, field vectors.
    NOTE: these plots are created in Output/Fields,
    and Output/FieldsPNG, which you must create.
```

```
        - 'plot_real'   : Plot the real part of modal fields.
        - 'plot_imag'   : Plot the imaginary part of modal fields.
        - 'plot_abs'    : Plot the absolute value of modal fields.
    """
```

3.6 Simulate_stack

We have now fully specified the properties of each layer in the problem and the light to be considered. The last thing we must specify is how to stack the layers together. Before doing this however, we first calculate the scattering matrices of each individual layer (between 2 semi-infinite air layers). This will allow us to stack the individual layers arbitrarily, changing their thicknesses with ease and recognise any repeated interfaces which need not be recalculated.

All of the real calculation occurs in the function simulate_stack defined in the simulation script itself. This function handles a single light instance and therefore must be called for each instance of light you are interested in. It firstly calculates the scattering matrices of each individual layer, storing this information in an instance of the layer object (which has the relevant scattering matrix calculation method defined on it).

A Stack object is defined within the function, where the simulation instances are listed from bottom to top. Multiple Stack objects may be defined within each simulation_stack call and the heights of layers may be changed (see simmo_template-NWs). Loops varying the Stack composition/ordering/heights are all able to be implemented within this function.

To finally execute the calculation we must call the simulation_stack with each light instance in light_list. This is simply done with the Python 'map' function. However we in general wish to consider many light instance in one simulation script. This poses an 'embarrassingly parallel' problem in the frequency domain, where each frequency/k-vector can be evaluated totally independently of all others.

To exploit this in MULASASS , we use the Python 'multiprocessing' package. This contains the function 'Pool', which is handed our mapped list of simulation_stack calls for each light instance in light_list. 'Pool' then feeds the light_list instances through simulation_stack managing the number of concurrent subprocesses (similar to threads in regular parallel computing) to be num_cores.

At the end of the simulation_stack calculations you are left with a list of Stack objects, one for each light instance in light_list. These have the properties of the Stack class as defined in stack.py.

```
class Stack(object):
    """ Represents a stack of layers evaluated at one frequency.

        This includes the semi-infinite input and output layers.

        INPUTS:

          - 'layers' : a tuple of :ThinFilm:s and :NanoStruct:s
            ordered from top to bottom layer.

          - 'heights_nm' : a tuple of the heights of the inside layers,
            i.e., all layers except for the top and bottom. This
            overrides any heights specified in the :ThinFilm: or
```

```
        :NanoStruct: objects.

    """
```

To plot quantities across the light_list instances (often wavelengths) results must be taken from each element of this list. This is implemented in many of the plotting functions.

3.7 Plotting

Various plotting functions are included with MULASASS . These are located and described in plotting.py. An exception to this is the plotting of modal field plots. This is done directly in the FEM routine and is therefore specified when creating the NanoStruct instance with the 'plot_modes' variable. IF these are to be saved the directories

- Output/Fields/
- Output/FieldsPNG/

must be created inside the simulation instance directory (eg. 000-simmo_templates).

Viewing of field plots, or FEM mesh is done using Gmsh. Geometry files have the extension .geo, FEM mesh files are .msh and .mail where .msh can be displayed with Gmsh and .mail are used by the Fortran FEM routine (the conversion FYI is done in gmsh_conversion.f). Field plots are stored as .pos files. The E-field components plotted include the

- x-component (x_re.pos),
- y-component (y_re.pos),
- z-component (z_re.pos),
- vectorial plot (v_re.pos),
- $|E|^2$ (_abs2.pos),
- energy, $\epsilon|E|^2$ (_abs2_eE.pos).

To open a file in Gmsh from the command line run,

```
gmsh simmo_location/Output/Fields/msh_name.ext
```

where the .ext can be any of the formats discussed above.
Otherwise you may also open files from your file manager if Gmsh has been installed.
To convert all energy distribution plots into .png files run

```
gmsh simmo_location/Output/FieldsPNG/All_plots_png_abs2_eE.geo
```

3.8 Help & Debugging

If you wish to have direct access to the simulation results (for further manipulation, debugging etc.) run the simulation script as

```
$ python -i simo_template-grating.py
```

which, after the calculations are complete, will return you into an interactive session of python, in which all simulation objects are accessible. In this session you can access the docstrings of objects/classes/methods by typing

```
>>> from pydoc import help
>>> help(objects.Light)
```

where we have accessed the docstring of the 'Light' class from objects.py.

3.9 Notes for running MULASASS on NCI supercomputers

Make sure

```
intel-fc
intel-cc
intel-mkl
python
python...-matplotlib
```

are all loaded. This is best done by adding

```
module load intel-fc/12.1.9.293
module load intel-cc/12.1.9.293
module load intel-mkl/12.1.9.293
module load python/2.7.3
module load python/2.7.3-matplotlib
```

to the bottom of your .login, .profile, and .cshrc/.bashrc files (or different versions numbers).

Chapter 4

Contributing