NumBAT Documentation

Release 0.1

Bjorn Sturmberg

Oct 20, 2016

Contents:

CHAPTER

ONE

INTRODUCTION

Introduction

EMUstack is an open-source simulation package for calculating light propagation through multi-layered stacks of dispersive, lossy, nanostructured, optical media. It implements a generalised scattering matrix method, which extends the physical intuition of thin film optics to complex structures.

At the heart of the scattering matrix approach is the requirement that each layer is uniform in one direction, here labelled z. In this nomenclature the incident field is unconstrained in $k_{\parallel} = k_{x,y}$ but must have $k_{\perp} = k_z \neq 0$.

In-plane each layer can be homogeneous, periodic in x or y, or double periodic (periodic in x and y). The modes of periodic (structured layers) are calculated using the Finite Element Method in respectively 1 or 2 dimensions, while the modes of homogeneous media are calculated analytically. This approach maximises the speed and accuracy of the calculations. These layers can be stacked in arbitrary order.

An advantage of EMUstack over other scattering matrix methods (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 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.

EMUstack 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. It is also possible to include media with lossless and/or non-dispersive refractive indices and EMUstack comes with a built in Drude model.

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, EMUstack 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.

EMUstack is a completely open source package, utilising free, open source compilers, meshing programs and libraries. All user interaction with EMUstack is done using the dynamic and easy to script language of python. The low-level numerical routines are written in Fortran for optimal performance making use of the LAPACK, ARPACK, and UMFPACK libraries. The Fortran routines are compiled as python subroutines using f2py. EMUstack currently comes with template FEM mesh for 1D and 2D gratings, Nanowire/Nanohole arrays, elliptical inclusions and split ring resonators. The mesh of other structures may be easily created using the open source program gmsh.

CHAPTER

TWO

INSTALLATION

Installation

The source code for NumBAT is hosted here on Github. Please download the latest release from here.

NumBAT has been developed on Ubuntu and is easiest to install on this platform. Simply 'sudo apt-get install' the packages listed in the dependencies.txt file and then run setup.sh.

```
$ sudo apt-get update
$ sudo apt-get -y install <dependencies>
$ /setup.sh
```

UPDATE: the current version of SuiteSparse is not fully compatible with 64 bit Linux... a solution to this is to backport SuiteSparse 3.4 from Ubuntu 12.04 using the method described here. Alternatively the pre-compiled libraries have been shown to work on Ubuntu 14.04

On other linux distributions either use the pre-compiled libraries of install them from the package manager or manually.

All that is required to use the pre-compiled libraries is to switch to a slightly modified Makefile and then run setup.sh.

```
$ cd backend/fortran/
$ mv Makefile Makefile_ubuntu
$ mv Makefile-pre_compiled_libs Makefile
$ cd ../../
$ /setup.sh
```

The Fortran components (NumBAT source code and libraries) have been successfully compiled with intel's ifortran as well as open-source gfortran. In this documentation we use gfortran.

NOTE: different versions of gmsh can give errors in the final test. This is okay, provided the test simulation ran, i.e. the test gives E rather than F.

SuiteSparse

The FEM routine used in NumBAT 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 a GPL license. It can be downloaded from https://www.cise.ufl.edu/research/sparse/SuiteSparse/

This is the process I followed in my installations. They are provided as little more than tips...

Unpack SuiteSparse into NumBAT/backend/fortran/, it should create a directory there; SuiteSparse/ Make a directory where you want SuiteSparse installed, in my case SS_installed

```
$ mkdir SS_installed/
```

edit SuiteSparse/SuiteSparse_config/SuiteSparse_config.mk for consistency across the whole build; i.e. if using intel fortran compiler

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

set path to install folder:

```
line 85 INSTALL_LIB = /$Path_to_EMustack/NumBAT/backend/fortran/SS_install/lib
line 86 INSTALL_INCLUDE = /$Path_to_EMustack/NumBAT/backend/fortran/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
```

Now make new directories for the paths you gave 2 steps back:

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

Download metis-4.0 and unpack metis into SuiteSparse/ Now move to the metis directory:

```
$ 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
```

Now make metis (still in SuiteSparse/metis-4.0/):

```
$ make
```

Now move back to NumBAT/backend/fortran/

```
$ cp SuiteSparse/metis-4.0/libmetis.a SS_install/lib/
```

and then move to SuiteSparse/ and execute the following:

```
$ 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 NumBAT/backend/fortran/Lib/ so that NumBAT/ is a complete package that can be moved across machine without alteration. This will override the pre-compiled libraries from the release (you may wish to save these somewhere).:

```
$ cp SS_install/lib/*.a NumBAT/backend/fortran/Lib/
$ cp SS_install/lib/umf4_f77zwrapper64.o NumBAT/backend/fortran/Lib/
```

NumBAT Makefile

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

Then finally run the setup.sh script!

CHAPTER

THREE

GUIDE

Simulation Structure

Simulations with NumBAT are generally carried out using a python script file. This file is kept in its own directory which is placed in the NumBAT directory. All results of the simulation are automatically created within this directory. This directory then serves as a complete record of the calculation. Often, we will also save the simulation objects (scattering matrices, propagation constants etc.) within this folder for future inspection, manipulation, plotting, etc.

Traditionally the name of the python script file begins with simo-. This is convenient for setting terminal alias' for running the script. Throughout the tutorial the script file will be called simo.py.

To start a simulation open a terminal and change into the directory containing the simo.py file. To run this script:

```
$ python simo.py
```

To have direct access to the simulation objects upon the completion of the script use,:

```
$ python -i simo.py
```

This will return you into an interactive python session in which all simulation objects are accessible. In this session you can access the docstrings of objects, classes and methods. For example:

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

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

In the remainder of this chapter we go through a number of example simo.py files.

Another tip to mention before diving into the examples is running simulations within screen_sesh. These allow you to disconnect from the terminal instance and are discusses in screen_sesh.

Screen Sessions

```
screen
```

is an extremely useful little linux command. In the context of long-ish calculations it has two important applications; ensuring your calculation is unaffected if your connection to a remote machine breaks, and terminating calculations that have hung without closing the terminal. For more information see the manual:

```
$ man screen
```

or see online discussions here, and here.

The screen session or also called screen instance looks just like your regular terminal/putty, but you can disconnect from it (close putty, turn off your computer etc.) and later reconnect to the screen session and everything inside of this will have kept running. You can also reconnect to the session from a different computer via ssh.

Basic Usage

To install screen:

```
$ sudo apt-get install screen
```

To open a new screen session:

```
$ screen
```

We can start a new calculation here:

```
$ cd NumBAT/examples/
$ python simo_040-2D_array.py
```

We can then detach from the session (leaving everything in the screen running) by typing:

```
Ctrl +a
Ctrl +d
```

We can now monitor the processes in that session:

```
$ top
```

Where we note the numerous running python processes that NumBAT has started. Watching the number of processes is useful for checking if a long simulation is near completion (which is indicated by the number of processes dropping to less than the specified num_cores).

We could now start another screen and run some more calculations in this terminal (or do anything else). If we want to access the first session we 'reattach' by typing:

```
Ctrl +a +r
```

Or entering the following into the terminal:

```
$ screen -r
```

If there are multiple sessions use:

```
$ screen -ls
```

to get a listing of the sessions and their ID numbers. To reattach to a particular screen, with ID 1221:

```
$ screen -r 1221
```

To terminate a screen from within type:

```
Ctrl+d
```

Or, taking the session ID from the previous example:

```
screen -X -S 1221 kill
```

Terminating NumBAT simos

If a simulation hangs, we can kill all python instances upon the machine:

```
$ pkill python
```

If a calculation hangs from within a screen session one must first detach from that session then kill python, or if it affects multiple instances, you can kill screen. A more targeted way to kill processes is using their PID:

```
$ kill PID
```

Or if this does not suffice be a little more forceful:

```
$ kill -9 PID
```

The PID is found from one of two ways:

```
$ top
$ ps -fe | grep username
```

Basic SBS Gain Calculation

```
print "SBS_gain \n", SBS_gain[EM_ival1,EM_ival2,:]/alpha
import time
import datetime
import numpy as np
import sys
from multiprocessing import Pool
sys.path.append("../backend/")

import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT

# Naming conventions
# AC: acoustic
# EM: electromagnetic
# q_acoustic: acoustic wavenumber

# Geometric Parameters - all in nm.
wl_nm = 1550
# Unit cell must be large to ensure fields are zero at boundary.
unitcell_x = 2.5*1550
unitcell_y = unitcell_x
# Waveguide width (x direction).
inc_a_x = 314.7
# Waveguide height (y direction).
inc_a_y = 0.9*inc_a_x
# Shape of the waveguide could also be 'circular'.
inc_shape = 'rectangular'

# Optical Parameters
# Permittivity
eps = 12.25
# Number of electromagnetic modes to solve for.
num_EM_modes = 20
# Number of acoustic modes to solve for.
num_AC_modes = 20
# The first EM mode(s) for which to calculate interaction with AC modes.
# Can specify a mode number (zero has lowest propagation constant) or 'All'
EM_ival1=0
# The second EM mode(s) for which to calculate interaction with AC modes.
EM_ival2=EM_ival1
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival='All'

# Acoustic Parameters
# Density
s = 2330  # kg/m3
# Stiffness tensor components
c_11 = 165.7e9; c_12 = 63.9e9; c_44 = 79.6e9  # Pa
# Photoelastic tensor components
p_11 = -0.094; p_12 = 0.017; p_44 = -0.051
# Acoustic loss tensor components
eta_11 = 5.9e-3 ; eta_12 = 5.16e-3 ; eta_44 = 0.620e-3  # Pa
```

```
# Put acoustic parameters together for convenience.
inc_a_AC_props = [s, c_11, c_12, c_44, p_11, p_12, p_44,
                  eta_11, eta_12, eta_44]

# Use all specified parameters to create a waveguide object.
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        bkg_material=materials.Material(1.0 + 0.0j),
                        inc_a_material=materials.Material(np.sqrt(eps)),
                        loss=False, inc_a_AC=inc_a_AC_props,
                        lc_bkg=0.1, lc2=40.0, lc3=20.0)


# Calculate Electromagnetic Modes
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes)
# Print the wavevectors of EM modes.
print 'k_z of EM wave \n', sim_EM_wguide.Eig_value
# Plot the EM modes fields, important to specify this with EM_AC='EM'.
# Zoom in on the central region (of big unitcell) with xlim, ylim args.
plotting.plt_mode_fields(sim_EM_wguide, xlim=0.4, ylim=0.4, EM_AC='EM')


# Choose acoustic wavenumber to solve for
# Backward SBS
# AC mode couples EM modes on +ve to -ve lightline, hence factor 2.
q_acoustic = 2*np.real(sim_EM_wguide.Eig_value[0])
print 'AC wavenumber (1/m) \n', q_acoustic
# Forward (intramode) SBS
# EM modes on same lightline.
# q_acoustic = 0.0


# Calculate Acoustic Modes
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, q_acoustic,
    num_AC_modes, EM_sim=sim_EM_wguide, shift_Hz=12e9)
# Print the frequencies of AC modes.
print 'Res freq of AC wave (GHz) \n', np.real(sim_AC_wguide.Eig_value)*1e-9
# Plot the AC modes fields, important to specify this with EM_AC='AC'.
# The AC modes are calculated on a subset of the full unitcell,
# which excludes vacuum regions, so no need to restrict area plotted.
plotting.plt_mode_fields(sim_AC_wguide, EM_AC='AC')


# Calculate interaction integrals
SBS_gain, Q_PE, Q_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, q_acoustic,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival)
# Print the Backward SBS gain of the AC modes.
print "SBS_gain \n", SBS_gain[EM_ival1,EM_ival2,:]/alpha
```

CHAPTER

FOUR

PYTHON BACKEND

objects module

materials module

materials.py is a subroutine of NumBAT that defines Material objects, these represent dispersive lossy refractive indices and possess methods to interpolate n from tabulated data.

class materials.**Material**(*n*)

Bases: object

Represents a material with a refractive index n.

If the material is dispersive, the refractive index at a given wavelength is calculated by linear interpolation from the initially given data *n*. Materials may also have *n* calculated from a Drude model with input parameters.

Parameters **n** – Either a scalar refractive index, an array of values *(wavelength, n)*, or *(wavelength, real(n), imag(n))*, or omega_p, omega_g, eps_inf for Drude model.

Currently included materials are;

Semiconductors	Metals	Transparent oxides
Si_c	Au	TiO2
Si_a	Au_Palik	TiO2_anatase
SiO2	Ag	ITO
CuO	Ag_Palik	ZnO
CdTe	Cu	SnO2
FeS2	Cu_Palik	FTO_Wenger
Zn3P2		FTO_Wengerk5
AlGaAs		
Al2O3		
Al2O3_PV		
GaAs		
InGaAs	**Drude**	**Other**
Si3N4	Au_drude	Air
MgF2		H2O
InP		Glass
InAs		Spiro
GaP		Spiro_nk

Continued on next page

Table 4.1 – continued from previous page

Ge		
AlN		
GaN		
MoO3		
ZnS		
AlN_PV		
		Experimental incl.
		CH3NH3PbI3
		Sb2S3
		Sb2S3_ANU2014
		Sb2S3_ANU2015
		GO_2014
		GO_2015
		rGO_2015
		SiON_Low
		SiON_High
		Low_Fe_Glass
		Perovskite_00
		Perovskite
		Perovskite_b2b
		Ge_Doped

n(*wl_nm*)
Return n for the specified wavelength.

materials.**plot_n_data**(*data_name*)

mode_calcs module

integration module

plotting module

CHAPTER

FIVE

FORTRAN BACKENDS

The intention of NumBAT is that the Fortran FEM routines are essentially black boxes. They are called from mode_calcs.py and return the modes (Eigenvalues) of a structured layer, as well as some matrices of overlap integrals that are then used to compute the scattering matrices.

There are however a few important things to know about the workings of these routines.

2D FEM Mode Solver

2D Mesh

2D FEM mesh are created using the open source program gmsh. In general they are created automatically by EMUstack using the templates files for each inclusion shape. These are stored in backend/fortran/msh. For an up to date list of templates see the 'inc_shape' entry in the NanoStruct docstring.

An advantage of using the FEM to calculate the modes of layers is that there is absolutely no constraints on the content of the unit cell. If you wish to create a different structure this can be done using gmsh, which is also used to view the mesh files (select files with the extension .msh).

Note that the area of the unit cell must always be unity! This has been assumed throughout the theoretical derivations.

FEM Errors

There are 2 errors that can be easily triggered within the Fortran FEM routines. These both cause them to simulation to abort and the terminal to be unresponsive (until you kill python or the screen session as described in screen_sesh).

The first of these is

```
Error with _naupd, info_32 =           -3
Check the documentation in _naupd.
Aborting...
```

Long story short, this indicates that the FEM mesh is too coarse for solutions for higher order Bloch modes (Eigenvaules) to converge. To see this run the simulation with FEM_debug = 1 (in mode_calcs.py) and it will print the number of converged Eigenvalues nconv != nval. This error is easily fixed by increasing the mesh resolution. Decrease 'lc_bkg' and/or increase 'lc2' etc.

The second error is

```
Error with _naupd, info_32 =           -8
Check the documentation in _naupd.
Aborting...
```

This is the opposite problem, when the mesh is so fine that the simulation is overloading the memory of the machine. More accurately the memory depends on the number of Eigenvalues being calculated as well as the number of FEM mesh points. The best solution to this is to increase 'lc_bkg' and/or decrease 'lc2' etc.

2D FEM Mode Solver

2D Mesh

2D FEM mesh are created using the open source program gmsh. In general they are created automatically by EMUstack using the templates files for each inclusion shape. These are stored in backend/fortran/msh. For an up to date list of templates see the 'inc_shape' entry in the NanoStruct docstring.

An advantage of using the FEM to calculate the modes of layers is that there is absolutely no constraints on the content of the unit cell. If you wish to create a different structure this can be done using gmsh, which is also used to view the mesh files (select files with the extension .msh).

Note that the area of the unit cell must always be unity! This has been assumed throughout the theoretical derivations.

FEM Errors

There are 2 errors that can be easily triggered within the Fortran FEM routines. These both cause them to simulation to abort and the terminal to be unresponsive (until you kill python or the screen session as described in screen_sesh).

The first of these is

```
Error with _naupd, info_32 =           -3
Check the documentation in _naupd.
Aborting...
```

Long story short, this indicates that the FEM mesh is too coarse for solutions for higher order Bloch modes (Eigenvaules) to converge. To see this run the simulation with FEM_debug = 1 (in mode_calcs.py) and it will print the number of converged Eigenvalues nconv != nval. This error is easily fixed by increasing the mesh resolution. Decrease 'lc_bkg' and/or increase 'lc2' etc.

The second error is

```
Error with _naupd, info_32 =           -8
Check the documentation in _naupd.
Aborting...
```

This is the opposite problem, when the mesh is so fine that the simulation is overloading the memory of the machine. More accurately the memory depends on the number of Eigenvalues being calculated as well as the number of FEM mesh points. The best solution to this is to increase 'lc_bkg' and/or decrease 'lc2' etc.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

m

M

N

P