NumBAT Documentation

Release 0.1

Bjorn Sturmberg

Mar 17, 2017

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

Introduction

NumBAT, the Numerical Brillouin Analysis Tool, integrates electromagnetic and acoustic mode solvers to calculate the interactions of optical and acoustic waves in waveguides.

NumBAT was developed by Bjorn Sturmberg, Kokou Dossou, Christian Wolff, Chris Poulton and Michael Steel in a collaboration between Macquarie University and the University of Technology Sydney, as part of the Australian Research Council Discovery Project DP160101691.

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 run the setup script

```
$ sudo /setup.sh
```

Or, if you prefer to do things manually, this is equivalent to

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install -y <dependencies>
$ cd backend/fortran/
$ make
$ cd ../../tests/
$ nosetests
```

where the <dependencies> packages are listed dependencies.txt.

This is all there is, there isn't any more.

Well there's more if you want to change it up.

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, but this can be easily adjusted in NumBAT/backend/fortran/Makefile

On non-ubuntu OS you may also need to compile a local version of Suitesparse, which is described in the next section.

Manual installation of 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, however this was some years ago and may need to be modified.

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 = /$Path_to_EMustack/NumBAT/backend/fortran/SS_installed/lib
line 86 INSTALL_INCLUDE = /$Path_to_EMustack/NumBAT/backend/fortran/SS_installed/
↪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_installed/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_installed/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_installed/lib/*.a NumBAT/backend/fortran/Lib/
$ cp SS_installed/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.Struct)
```

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

In the remainder of this chapter we go through a number of example simo.py files. But before we do, another quick tip about running simulations within screen sessions, which allow you to disconnect from servers leaving them to continue your processes.

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/tutorials/
$ python simo-tut_01-first_calc.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
```

Tutorial

In this section we go through a number of simple simulations that demonstrate the basic use of NumBAT.

Basic SBS Gain Calculation

```
print("\n Simulation time (sec.)", (end - start))
""" Calculate the backward SBS gain for modes in a
    silicon waveguide surrounded in air.
"""

import time
import datetime
import numpy as np
import sys
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
# k_AC: acoustic wavenumber

start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550 # Wavelength of EM wave in vacuum.
# Unit cell must be large to ensure fields are zero at boundary.
unitcell_x = 2.5*wl_nm
unitcell_y = unitcell_x
# Waveguide widths.
inc_a_x = 314.7
inc_a_y = 0.9*inc_a_x
# Shape of the waveguide.
inc_shape = 'rectangular'

# 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'

# Use specified parameters to create a waveguide object.
# Note use of rough mesh for demonstration purposes.
```

```
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        bkg_material=materials.Air,
                        inc_a_material=materials.Si,
                        lc_bkg=2, lc2=200.0, lc3=5.0)

# Expected effective index of fundamental guided mode.
n_eff = wguide.inc_a_material.n-0.1

# Calculate Electromagnetic modes.
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff)
# Print the wavevectors of EM modes.
print('\n k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values),4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values[0]*((wl_nm*1e-9)/(2.*np.pi)))
print("\n n_eff = ", np.round(n_eff_sim, 4))

# Choose acoustic wavenumber to solve for.
# Backward SBS - AC mode couples EM modes on +ve to -ve lightline, hence factor 2.
k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])
print('\n AC wavenumber (1/m) = ', np.round(k_AC, 4))
# Forward (intramode) SBS - EM modes on same lightline.
# k_AC = 0.0

# Calculate Acoustic modes.
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes,
    k_AC=k_AC, EM_sim=sim_EM_wguide)
# Print the frequencies of AC modes.
print('\n Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,
↪ 4))

# Calculate interaction integrals and SBS gain for PE and MB effects combined,
# as well as just for PE, and just for MB. Also calculate acoustic loss alpha.
SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival)
# Print the Backward SBS gain of the AC modes.
print("\n SBS_gain PE contribution \n", SBS_gain_PE[EM_ival1,EM_ival2,:]/alpha)
print("SBS_gain MB contribution \n", SBS_gain_MB[EM_ival1,EM_ival2,:]/alpha)
print("SBS_gain total \n", SBS_gain[EM_ival1,EM_ival2,:]/alpha)
# Mask negligible gain values to improve clarity of print out.
threshold = 1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival1,EM_ival2,:]/alpha, 0, threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival1,EM_ival2,:]/alpha, 0, threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival1,EM_ival2,:]/alpha, 0, threshold)
print("\n SBS_gain PE contribution \n", masked_PE)
print("SBS_gain MB contribution \n", masked_MB)
print("SBS_gain total \n", masked)

end = time.time()
print("\n Simulation time (sec.)", (end - start))
```

SBS Gain Spectra

```
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_zoom
↪')
```

```
""" Calculate the backward SBS gain spectra of a
    silicon waveguide surrounded in air.

    Show how to save simulation objects (eg. EM mode calcs)
    to expedite the process of altering later parts of
    simulations.
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 2.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 314.7
inc_a_y = 0.9*inc_a_x
inc_shape = 'rectangular'

num_EM_modes = 20
num_AC_modes = 20
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

# Use of a more refined mesh to produce field plots.
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        bkg_material=materials.Air,
                        inc_a_material=materials.Si,
                        lc_bkg=2, lc2=1000.0, lc3=10.0)


# Expected effective index of fundamental guided mode.
n_eff = wguide.inc_a_material.n-0.1

# Calculate Electromagnetic modes.
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff)
# Save calculated :Simmo: object for EM calculation.
np.savez('wguide_data', sim_EM_wguide=sim_EM_wguide)

# Once npz files have been saved from one simulation run,
# the previous three lines can be commented and the following
# two line uncommented. This provides precisely the same objects
# for the remainder of the simulation.
```

```
# npzfile = np.load('wguide_data.npz')
# sim_EM_wguide = npzfile['sim_EM_wguide'].tolist()

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# 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_min=0.4, xlim_max=0.4,
                         ylim_min=0.4, ylim_max=0.4, EM_AC='EM')

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values[0]*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff", np.round(n_eff_sim, 4))

# Choose acoustic wavenumber to solve for backward SBS
k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

# Calculate Acoustic modes.
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC, EM_sim=sim_EM_wguide)
# Save calculated :Simmo: object for AC calculation.
np.savez('wguide_data_AC', sim_AC_wguide=sim_AC_wguide)

# npzfile = np.load('wguide_data_AC.npz')
# sim_AC_wguide = npzfile['sim_AC_wguide'].tolist()

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,␣
↪4))

# 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.
# We want to get pdf files so set pdf_png='pdf'
# (default is png as these are easier to flick through).
plotting.plt_mode_fields(sim_AC_wguide, EM_AC='AC', pdf_png='pdf')

# Do not calculate the acoustic loss from our fields, instead set a Q factor.
set_q_factor = 1000.

# Calculate interaction integrals and SBS gain for PE and MB effects combined,
# as well as just for PE, and just for MB.
SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival, fixed_Q=set_q_factor)
# Save the gain calculation results
np.savez('wguide_data_AC_gain', SBS_gain=SBS_gain, SBS_gain_PE=SBS_gain_PE,
            SBS_gain_MB=SBS_gain_MB, alpha=alpha)

# Once npz files have been saved from one simulation run,
# the previous six lines can be commented and the following
# five line uncommented. This provides precisely the same objects
# for the remainder of the simulation.
# npzfile = np.load('wguide_data_AC_gain.npz')
# SBS_gain = npzfile['SBS_gain']
# SBS_gain_PE = npzfile['SBS_gain_PE']
# SBS_gain_MB = npzfile['SBS_gain_MB']
# alpha = npzfile['alpha']
```

```
# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 10  # GHz
freq_max = 25  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)
# Zoomed in version
freq_min = 11  # GHz
freq_max = 15  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_zoom
↪')
```

Investigating Dispersion

```
plt.close()
""" Calculate a dispersion diagram of the acoustic modes
    from k_AC = 0 (forward SBS) to k_AC = 2*k_EM (backward SBS).
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 2.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 314.7
inc_a_y = 0.9*inc_a_x
inc_shape = 'rectangular'
# Choose modes to include.
num_EM_modes = 20
num_AC_modes = 20
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        bkg_material=materials.Air,
                        inc_a_material=materials.Si,
                        lc_bkg=2, lc2=500.0, lc3=10.0)
```

```
# Expected effective index of fundamental guided mode.
n_eff = wguide.inc_a_material.n-0.1

# Calculate Electromagnetic modes.
# sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff)
# np.savez('wguide_data', sim_EM_wguide=sim_EM_wguide)

# Assuming this calculation is run directly after simo-tut_02
# we don't need to recalculate EM modes, but can load them in.
npzfile = np.load('wguide_data.npz')
sim_EM_wguide = npzfile['sim_EM_wguide'].tolist()

# Will scan from forward to backward SBS so need to know k_AC of backward SBS.
k_AC = 2*sim_EM_wguide.Eig_values[0]
# Number of wavevectors steps.
nu_ks = 20

plt.clf()
plt.figure(figsize=(10,6))
ax = plt.subplot(1,1,1)
for i_ac, q_ac in enumerate(np.linspace(0.0,k_AC,nu_ks)):
    sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, q_ac, EM_sim=sim_EM_
↪wguide)
    prop_AC_modes = np.array([np.real(x) for x in sim_AC_wguide.Eig_values if abs(np.
↪real(x)) > abs(np.imag(x))])
    sym_list = integration.symmetries(sim_AC_wguide)

    for i in range(len(prop_AC_modes)):
        Om = prop_AC_modes[i]*1e-9
        if sym_list[i][0] == 1 and sym_list[i][1] == 1 and sym_list[i][2] == 1:
            sym_A, = plt.plot(np.real(q_ac/k_AC), Om, 'or')
        if sym_list[i][0] == -1 and sym_list[i][1] == 1 and sym_list[i][2] == -1:
            sym_B, = plt.plot(np.real(q_ac/k_AC), Om, 'vc')
        if sym_list[i][0] == 1 and sym_list[i][1] == -1 and sym_list[i][2] == -1:
            sym_C, = plt.plot(np.real(q_ac/k_AC), Om, 'sb')
        if sym_list[i][0] == -1 and sym_list[i][1] == -1 and sym_list[i][2] == 1:
            sym_D, = plt.plot(np.real(q_ac/k_AC), Om, '^g')

    print("wavevector loop", i_ac+1, "/", nu_ks)
ax.set_ylim(0,20)
ax.set_xlim(0,1)
plt.legend([sym_A, sym_B, sym_C, sym_D],['E',r'C$_2$',r'$\sigma_y$',r'$\sigma_x$'],␣
↪loc='lower right')
plt.xlabel(r'Axial wavevector (normalised)')
plt.ylabel(r'Frequency (GHz)')
plt.savefig('symetrised_dispersion.pdf', bbox_inches='tight')
plt.close()
```

Parameter Scan of Widths

```
plt.close()
""" Calculate the backward SBS gain spectra as a function of
    waveguide width, for silicon waveguides surrounded in air.

    Also shows how to use python multiprocessing library.
```

```
"""

import time
import datetime
import numpy as np
import sys
from multiprocessing import Pool
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.collections import PolyCollection
from matplotlib.colors import colorConverter

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


# Select the number of CPUs to use in simulation.
num_cores = 6

# Geometric Parameters - all in nm.
wl_nm = 1550
inc_shape = 'rectangular'

num_EM_modes = 20
num_AC_modes = 20
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

# Width previous simo's done for, with known meshing params
known_geo = 315.

def modes_n_gain(wguide):
    # Expected effective index of fundamental guided mode.
    n_eff = (wguide.inc_a_material.n-0.1) * wguide.inc_a_x/known_geo
    # Calculate Electromagnetic modes.
    sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff)
    # Backward SBS
    k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])
    # Calculate Acoustic modes.
    sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC,
        EM_sim=sim_EM_wguide)
    # Calculate interaction integrals and SBS gain.
    SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
        sim_EM_wguide, sim_AC_wguide, k_AC,
        EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival)

    return [sim_EM_wguide, sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha,␣
↪k_AC]
```

```
nu_widths = 6
waveguide_widths = np.linspace(300,400,nu_widths)
geo_objects_list = []
# Scale meshing to new structures.
for width in waveguide_widths:
    msh_ratio = (width/known_geo)
    unitcell_x = 2.5*wl_nm*msh_ratio
    unitcell_y = unitcell_x
    inc_a_x = width
    inc_a_y = 0.9*inc_a_x

    wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,
                            inc_a_y,inc_shape,
                            bkg_material=materials.Air,
                            inc_a_material=materials.Si,
                            lc_bkg=2, lc2=1000.0, lc3=10.0)
    geo_objects_list.append(wguide)


# Run widths in parallel across num_cores CPUs using multiprocessing package.
pool = Pool(num_cores)
width_objs = pool.map(modes_n_gain, geo_objects_list)
# np.savez('Simo_results', width_objs=width_objs)
# npzfile = np.load('Simo_results.npz')
# width_objs = npzfile['width_objs'].tolist()


n_effs = []
freqs_gains = []
interp_grid_points = 10000
interp_grid = np.linspace(10, 25, interp_grid_points)
for i_w, width_obj in enumerate(width_objs):
    interp_values = np.zeros(interp_grid_points)
    sim_EM = width_obj[0]
    sim_AC = width_obj[1]
    SBS_gain = width_obj[2]
    SBS_gain_PE = width_obj[3]
    SBS_gain_MB = width_obj[4]
    alpha = width_obj[5]
    k_AC = width_obj[6]
    # Calculate the EM effective index of the waveguide (k_AC = 2*k_EM).
    n_eff_sim = round(np.real((k_AC/2.)*((wl_nm*1e-9)/(2.*np.pi))), 4)
    n_effs.append(n_eff_sim)

    # Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
    freq_min = 10  # GHz
    freq_max = 25  # GHz
    plotting.gain_specta(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
        EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_
↪scan%i' % i_w)

    # Repeat calc to collect data for waterfall plot.
    tune_steps = 5e4
    tune_range = 10 # GHz
    detuning_range = np.append(np.linspace(-1*tune_range, 0, tune_steps),
                       np.linspace(0, tune_range, tune_steps)[1:])*1e9 # GHz
    phase_v = sim_AC.Eig_values/k_AC
```

```
    line_width = phase_v*alpha
    for AC_i in range(len(alpha)):
        gain_list = np.real(SBS_gain[EM_ival1,EM_ival2,AC_i]/alpha[AC_i]
                    *line_width[AC_i]**2/(line_width[AC_i]**2 + detuning_range**2))
        freq_list_GHz = np.real(sim_AC.Eig_values[AC_i] + detuning_range)*1e-9
        interp_spectrum = np.interp(interp_grid, freq_list_GHz, gain_list)
        interp_values += interp_spectrum
    freqs_gains.append(list(zip(interp_grid, interp_values)))

print('Widths', waveguide_widths)
print('n_effs', n_effs)

# Plot a 'waterfall' plot.
fig = plt.figure()
ax = fig.gca(projection='3d')
poly = PolyCollection(freqs_gains)
poly.set_alpha(0.7)
ax.add_collection3d(poly, zs=waveguide_widths, zdir='y')
ax.set_xlabel('Frequency (GHz)', fontsize=14)
ax.set_xlim3d(10,25)
ax.set_ylabel('Width (nm)', fontsize=14)
ax.set_ylim3d(waveguide_widths[0], waveguide_widths[-1])
ax.set_zlabel('Gain (1/Wm)', fontsize=14)
ax.set_zlim3d(0,1500)
# We change the fontsize of minor ticks label
plt.tick_params(axis='both', which='major', labelsize=12, pad=-2)
plt.savefig('gain_spectra_waterfall.pdf')
plt.close()
```

Convergence Study

```
print("Calculation time", time_list)
""" Calculate the convergence as a function of FEM mesh for
    backward SBS gain spectra of a silicon waveguide surrounded in air.
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Select the number of CPUs to use in simulation.
num_cores = 6
```

```
# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 2.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 2000
inc_a_y = 680
inc_shape = 'rectangular'

num_EM_modes = 20
num_AC_modes = 20
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'


nu_lcs = 4
lc_bkg_list = 2*np.ones(nu_lcs)
lc_list = np.linspace(2e2,3e3,nu_lcs)
x_axis = lc_bkg_list
x_axis = lc_list
conv_list = []
time_list = []
# Do not run in parallel, otherwise there are confusions reading the msh files!
for i_lc, lc_ref in enumerate(lc_list):
    start = time.time()
    print("\n Running simulation", i_lc+1, "/", nu_lcs)
    lc3 = 0.01*lc_ref
    lc_bkg = lc_bkg_list[i_lc]
    wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,
                            inc_a_y,inc_shape,
                            bkg_material=materials.Air,
                            inc_a_material=materials.Si,
                            lc_bkg=lc_bkg, lc2=lc_ref, lc3=lc3, force_mesh=True)

    # Expected effective index of fundamental guided mode.
    n_eff = wguide.inc_a_material.n-0.1
    # Calculate Electromagnetic modes.
    sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff)
    # Backward SBS
    k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])
    # Calculate Acoustic modes.
    sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC,
        EM_sim=sim_EM_wguide)
    # Calculate interaction integrals and SBS gain.
    SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
        sim_EM_wguide, sim_AC_wguide, k_AC,
        EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival)

    conv_list.append([sim_EM_wguide, sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_
↪MB, alpha])
    end = time.time()
    time_list.append(end - start)


rel_modes = [2,4,8]
rel_mode_freq_EM = np.zeros(nu_lcs,dtype=complex)
rel_mode_freq_AC = np.zeros((nu_lcs,len(rel_modes)),dtype=complex)
rel_mode_gain = np.zeros((nu_lcs,len(rel_modes)),dtype=complex)
```

```
rel_mode_gain_MB = np.zeros((nu_lcs,len(rel_modes)),dtype=complex)
rel_mode_gain_PE = np.zeros((nu_lcs,len(rel_modes)),dtype=complex)
for i_conv, conv_obj in enumerate(conv_list):
    rel_mode_freq_EM[i_conv] = conv_obj[0].Eig_values[0]
    for i_m, rel_mode in enumerate(rel_modes):
        rel_mode_freq_AC[i_conv,i_m] = conv_obj[1].Eig_values[rel_mode]
        rel_mode_gain[i_conv,i_m] = conv_obj[2][EM_ival1,EM_ival2,rel_mode]/conv_
↪obj[5][rel_mode]
        rel_mode_gain_PE[i_conv,i_m] = conv_obj[3][EM_ival1,EM_ival2,rel_mode]/conv_
↪obj[5][rel_mode]
        rel_mode_gain_MB[i_conv,i_m] = conv_obj[4][EM_ival1,EM_ival2,rel_mode]/conv_
↪obj[5][rel_mode]



xlabel = "Mesh Refinement Factor"
fig = plt.figure()
plt.clf()
ax1 = fig.add_subplot(1,1,1)
ax2 = ax1.twinx()
ax2.yaxis.tick_left()
ax2.yaxis.set_label_position("left")
EM_plot_Mk = rel_mode_freq_EM*1e-6
error0 = np.abs((np.array(EM_plot_Mk[0:-1])-EM_plot_Mk[-1])/EM_plot_Mk[-1])
ax2.plot(x_axis[0:-1], error0, 'b-v',label='Mode #%i'%EM_ival1)
ax1.plot(x_axis, np.real(EM_plot_Mk), 'r-.o',label=r'EM k$_z$')
ax1.yaxis.tick_right()
ax1.spines['right'].set_color('red')
ax1.yaxis.label.set_color('red')
ax1.yaxis.set_label_position("right")
ax1.tick_params(axis='y', colors='red')
handles, labels = ax2.get_legend_handles_labels()
ax2.legend(handles, labels)
ax1.set_xlabel(xlabel)
ax1.set_ylabel(r"EM k$_z$ ($\times 10^6$ 1/m)")
ax2.set_ylabel(r"Relative Error EM k$_z$")
ax2.set_yscale('log', nonposx='clip')
plt.savefig('convergence-freq_EM.pdf', bbox_inches='tight')
plt.close()

fig = plt.figure()
plt.clf()
ax1 = fig.add_subplot(1,1,1)
ax2 = ax1.twinx()
ax2.yaxis.tick_left()
ax2.yaxis.set_label_position("left")
for i_m, rel_mode in enumerate(rel_modes):
    rel_mode_freq_AC_plot_GHz = rel_mode_freq_AC[:,i_m]*1e-9
    error0 = np.abs((np.array(rel_mode_freq_AC_plot_GHz[0:-1])-rel_mode_freq_AC_plot_
↪GHz[-1])/rel_mode_freq_AC_plot_GHz[-1])
    ax2.plot(x_axis[0:-1], error0, '-v',label='Mode #%i'%rel_mode)
    ax1.plot(x_axis, np.real(rel_mode_freq_AC_plot_GHz), '-.o',label=r'AC Freq mode #
↪%i'%rel_mode)
ax1.yaxis.tick_right()
ax1.spines['right'].set_color('red')
ax1.yaxis.label.set_color('red')
ax1.yaxis.set_label_position("right")
ax1.tick_params(axis='y', colors='red')
```

```
handles, labels = ax2.get_legend_handles_labels()
ax2.legend(handles, labels)
ax1.set_xlabel(xlabel)
ax1.set_ylabel(r"AC Freq (GHz)")
ax2.set_ylabel(r"Relative Error AC Freq")
ax2.set_yscale('log', nonposx='clip')
plt.savefig('convergence-freq_AC.pdf', bbox_inches='tight')
plt.close()

fig = plt.figure()
plt.clf()
ax1 = fig.add_subplot(1,1,1)
ax2 = ax1.twinx()
ax2.yaxis.tick_left()
ax2.yaxis.set_label_position("left")
for i_m, rel_mode in enumerate(rel_modes):
    rel_mode_gain_plot = rel_mode_gain[:,i_m]
    error0 = np.abs((np.array(rel_mode_gain_plot[0:-1])-rel_mode_gain_plot[-1])/rel_
↪mode_gain_plot[-1])
    ax2.plot(x_axis[0:-1], error0, '-v',label=r'Mode #%i'%rel_mode)
    ax1.plot(x_axis, np.real(rel_mode_gain_plot), '-.o',label=r'Gain mode #%i'%rel_
↪mode)
ax1.yaxis.tick_right()
ax1.spines['right'].set_color('red')
ax1.yaxis.label.set_color('red')
ax1.yaxis.set_label_position("right")
ax1.tick_params(axis='y', colors='red')
handles, labels = ax2.get_legend_handles_labels()
ax2.legend(handles, labels)
ax1.set_xlabel(xlabel)
ax1.set_ylabel(r"Gain")
ax2.set_ylabel(r"Relative Error Gain")
ax2.set_yscale('log', nonposx='clip')
plt.savefig('convergence-Gain.pdf', bbox_inches='tight')
plt.close()

fig = plt.figure()
plt.clf()
ax1 = fig.add_subplot(1,1,1)
ax2 = ax1.twinx()
ax2.yaxis.tick_left()
ax2.yaxis.set_label_position("left")
for i_m, rel_mode in enumerate(rel_modes):
    rel_mode_gain_PE_plot = rel_mode_gain_PE[:,i_m]
    error0 = np.abs((np.array(rel_mode_gain_PE_plot[0:-1])-rel_mode_gain_PE_plot[-1])/
↪rel_mode_gain_PE_plot[-1])
    ax2.plot(x_axis[0:-1], error0, '-v',label=r'Mode #%i'%rel_mode)
    ax1.plot(x_axis, np.real(rel_mode_gain_PE_plot), '-.o',label=r'Gain mode #%i'%rel_
↪mode)
ax1.yaxis.tick_right()
ax1.spines['right'].set_color('red')
ax1.yaxis.label.set_color('red')
ax1.yaxis.set_label_position("right")
ax1.tick_params(axis='y', colors='red')
handles, labels = ax2.get_legend_handles_labels()
ax2.legend(handles, labels)
ax1.set_xlabel(xlabel)
ax1.set_ylabel(r"Gain")
```

```
ax2.set_ylabel(r"Relative Error Gain")
ax2.set_yscale('log', nonposx='clip')
plt.savefig('convergence-Gain_PE.pdf', bbox_inches='tight')
plt.close()

fig = plt.figure()
plt.clf()
ax1 = fig.add_subplot(1,1,1)
ax2 = ax1.twinx()
ax2.yaxis.tick_left()
ax2.yaxis.set_label_position("left")
for i_m, rel_mode in enumerate(rel_modes):
    rel_mode_gain_MB_plot = rel_mode_gain_MB[:,i_m]
    error0 = np.abs((np.array(rel_mode_gain_MB_plot[0:-1])-rel_mode_gain_MB_plot[-1])/
↪rel_mode_gain_MB_plot[-1])
    ax2.plot(x_axis[0:-1], error0, '-v',label=r'Mode #%i'%rel_mode)
    ax1.plot(x_axis, np.real(rel_mode_gain_MB_plot), '-.o',label=r'Gain mode #%i'%rel_
↪mode)
ax1.yaxis.tick_right()
ax1.spines['right'].set_color('red')
ax1.yaxis.label.set_color('red')
ax1.yaxis.set_label_position("right")
ax1.tick_params(axis='y', colors='red')
handles, labels = ax2.get_legend_handles_labels()
ax2.legend(handles, labels)
ax1.set_xlabel(xlabel)
ax1.set_ylabel(r"Gain")
ax2.set_ylabel(r"Relative Error Gain")
ax2.set_yscale('log', nonposx='clip')
plt.savefig('convergence-Gain_MB.pdf', bbox_inches='tight')
plt.close()

print("Calculation time", time_list)
```

Silica Nanowire

```
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_
↪SiO2_NW')""" We've covered most of the features of NumBAT,
        in the following tutorials we'll show how to
        study differnt geometries and materials.

        Calculate the backward SBS gain spectra of a
    silicon waveguide surrounded in air.
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

import materials
import objects
```

```
import mode_calcs
import integration
import plotting
from fortran import NumBAT


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 550
inc_a_y = inc_a_x
inc_shape = 'circular'

num_EM_modes = 20
num_AC_modes = 40
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        bkg_material=materials.Air,
                        inc_a_material=materials.SiO2,
                        lc_bkg=3, lc2=2000.0, lc3=10.0)

# Expected effective index of fundamental guided mode.
n_eff = 1.4

# Calculate Electromagnetic Modes
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)
# np.savez('wguide_data', sim_EM_wguide=sim_EM_wguide)
# npzfile = np.load('wguide_data.npz')
# sim_EM_wguide = npzfile['sim_EM_wguide'].tolist()
# plotting.plt_mode_fields(sim_EM_wguide, xlim_min=0.4, xlim_max=0.4,
#                          ylim_min=0.4, ylim_max=0.4, EM_AC='EM', add_name='NW')
# plotting.plt_mode_fields(sim_EM_wguide, EM_AC='EM', add_name='NW')

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff = ", np.round(n_eff_sim, 4))

k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

shift_Hz = 4e9

# Calculate Acoustic modes.
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
    EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)
# np.savez('wguide_data_AC', sim_AC_wguide=sim_AC_wguide)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC_wguide = npzfile['sim_AC_wguide'].tolist()
# plotting.plt_mode_fields(sim_AC_wguide, EM_AC='AC', add_name='NW')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,␣
↪4))
```

```
set_q_factor = 1000.

# Calculate interaction integrals and SBS gain for PE and MB effects combined,
# as well as just for PE, and just for MB.
SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival, fixed_Q=set_q_factor)
# np.savez('wguide_data_AC_gain', SBS_gain=SBS_gain, SBS_gain_PE=SBS_gain_PE, SBS_
↪gain_MB=SBS_gain_MB, alpha=alpha)
# npzfile = np.load('wguide_data_AC_gain.npz')
# SBS_gain = npzfile['SBS_gain']
# SBS_gain_PE = npzfile['SBS_gain_PE']
# SBS_gain_MB = npzfile['SBS_gain_MB']
# alpha = npzfile['alpha']

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 0  # GHz
freq_max = 12  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_
↪SiO2_NW')
```

Slot Waveguide

```
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_slot
↪')""" Calculate the backward SBS gain spectra of a Si
    slot waveguide containing As2S3 surrounded in SiO2.
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = unitcell_x
inc_a_x = 150
inc_a_y = 190
inc_shape = 'slot'
inc_b_x = 250
```

```
# Current mesh template assume inc_b_y = inc_a_y
slab_a_y = wl_nm

num_EM_modes = 20
num_AC_modes = 40
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        inc_b_x =inc_b_x, slab_a_y=slab_a_y,
                        inc_a_material=materials.As2S3,
                        inc_b_material=materials.Si,
                        slab_a_material=materials.SiO2,
                        lc_bkg=3, lc2=2000.0, lc3=1000.0)
# In this case lc3 is meshing around ribs encasing the slot (the Si)

# Expected effective index of fundamental guided mode.
n_eff = 2.8

# Calculate Electromagnetic modes.
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)
# np.savez('wguide_data', sim_EM_wguide=sim_EM_wguide)
# npzfile = np.load('wguide_data.npz')
# sim_EM_wguide = npzfile['sim_EM_wguide'].tolist()

# plotting.plt_mode_fields(sim_EM_wguide, xlim_min=0.4, xlim_max=0.4,
#                          ylim_min=0.1, ylim_max=0.8, EM_AC='EM', add_name='slot')

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff = ", np.round(n_eff_sim, 4))

k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

# Specify the expected acoustic frequency (slightly low balled).
shift_Hz = 4e9

# Calculate Acoustic modes.
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
    EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)
# np.savez('wguide_data_AC', sim_AC_wguide=sim_AC_wguide)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC_wguide = npzfile['sim_AC_wguide'].tolist()

# plotting.plt_mode_fields(sim_AC_wguide, xlim_min=0.4, xlim_max=0.4,
#                          ylim_min=0.7, ylim_max=0.0, EM_AC='AC', add_name='slot')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,␣
↪4))

set_q_factor = 1000.

SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
```

```
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival, fixed_Q=set_q_factor)
# np.savez('wguide_data_AC_gain', SBS_gain=SBS_gain, SBS_gain_PE=SBS_gain_PE, SBS_
↪gain_MB=SBS_gain_MB, alpha=alpha)
# npzfile = np.load('wguide_data_AC_gain.npz')
# SBS_gain = npzfile['SBS_gain']
# SBS_gain_PE = npzfile['SBS_gain_PE']
# SBS_gain_MB = npzfile['SBS_gain_MB']
# alpha = npzfile['alpha']

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 5  # GHz
freq_max = 10  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_slot
↪')
```

Covered Slot Waveguide

```
        EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_
↪%i' %int(coat_y))""" Calculate the backward SBS gain spectra of a Si
    slot waveguide containing As2S3 surrounded in SiO2.

    This time include a capping layer of SiO2 and
    investigate the effect of this layer's thickness.
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = unitcell_x
inc_a_x = 150
inc_a_y = 190
inc_shape = 'slot'
inc_b_x = 250
# Current mesh template assume inc_b_y = inc_a_y
slab_a_y = wl_nm
```

```
num_EM_modes = 20
num_AC_modes = 40
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'


coat_y_list = np.linspace(50,200,4)
for coat_y in coat_y_list:
    wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                            inc_b_x =inc_b_x, slab_a_y=slab_a_y, coat_y=coat_y,
                            bkg_material=materials.Air,
                            inc_a_material=materials.As2S3,
                            inc_b_material=materials.Si,
                            slab_a_material=materials.SiO2,
                            coat_material=materials.SiO2,
                            lc_bkg=3, lc2=1500.0, lc3=700.0)

    # Expected effective index of fundamental guided mode.
    n_eff = 2.8

    # Calculate Electromagnetic modes.
    sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)

    k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

    shift_Hz = 4e9

    # Calculate Acoustic modes.
    sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
        EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)

    # plotting.plt_mode_fields(sim_AC_wguide, xlim_min=0.4, xlim_max=0.4,
    #                          ylim_min=0.7, ylim_max=0.0, EM_AC='AC', add_name='_%i
↪' %int(coat_y))

    set_q_factor = 1000.

    SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
        sim_EM_wguide, sim_AC_wguide, k_AC,
        EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival, fixed_Q=set_q_factor)

    # Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
    freq_min = 5  # GHz
    freq_max = 15  # GHz
    plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_
↪AC,
        EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max, add_name='_
↪%i' %int(coat_y))
```

Literature Examples

Having gotten familiar with NumBAT, we now set out to replicate a number of examples from the literature.

Rectangular Waveguide - Silica

```
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)
""" Replicating the results of
    Generation of phonons from electrostriction in small-core optical waveguides
    Laude et al.
    http://dx.doi.org/10.1063/1.4801936
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 7*wl_nm
unitcell_y = unitcell_x
inc_a_x = 1500
inc_a_y = 1000
inc_shape = 'rectangular'

# Optical Parameters
num_EM_modes = 20
num_AC_modes = 120
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

# Material parameters as in paper
# Silicon
n = 1.44
s = 2203  # kg/m3
c_11 = 78e9; c_12 = 16e9; c_44 = 31e9
p_11 = 0.12; p_12 = 0.270; p_44 = -0.073
eta_11 = 1.6e-3 ; eta_12 = 1.29e-3 ; eta_44 = 0.16e-3  # Pa s
SiO2_props = [n, 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.Air,
                        inc_a_material=materials.Material(SiO2_props),
                        lc_bkg=3, lc2=2000.0, lc3=20.0)

# Expected effective index of fundamental guided mode.
n_eff = 1.3

# Calculate Electromagnetic modes.
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff = ", np.round(n_eff_sim, 4))

k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

shift_Hz = 8e9

# Calculate Acoustic modes.
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
    EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,␣
↪4))

# Calculate interaction integrals and SBS gain for PE and MB effects combined,
# as well as just for PE, and just for MB.
SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 4  # GHz
freq_max = 13  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)
```

Rectangular Waveguide - Silicon

```
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)""" Replicating␣
↪the results of
    Generation of phonons from electrostriction in small-core optical waveguides
    Laude et al.
    http://dx.doi.org/10.1063/1.4801936
"""

import time
import datetime
import numpy as np
import sys
```

```
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 7*wl_nm
unitcell_y = unitcell_x
inc_a_x = 1500
inc_a_y = 1000
inc_shape = 'rectangular'

# Optical Parameters
num_EM_modes = 20
num_AC_modes = 800
EM_ival1=0
EM_ival2=EM_ival1
AC_ival='All'

# Material parameters as in paper
# Silicon
n = 3.47
s = 2331  # kg/m3
c_11 = 166e9; c_12 = 64e9; c_44 = 79e9
p_11 = -0.1; p_12 = -0.01; p_44 = -0.051
eta_11 = 1.6e-3 ; eta_12 = 1.29e-3 ; eta_44 = 0.16e-3  # Pa s
Si_props = [n, 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.Air,
                        inc_a_material=materials.Material(Si_props),
                        lc_bkg=3, lc2=2000.0, lc3=20.0)

# Expected effective index of fundamental guided mode.
n_eff = 3.4

# Calculate Electromagnetic modes.
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff = ", np.round(n_eff_sim, 4))

k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])
```

```
shift_Hz = 31e9

# Calculate Acoustic modes.
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
    EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,␣
↪4))

# Calculate interaction integrals and SBS gain for PE and MB effects combined,
# as well as just for PE, and just for MB.
SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 20  # GHz
freq_max = 45  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)
```

Tapered Fibre

```
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)""" Replicating␣
↪the results of
    Brillouin scattering self-cancellation
    Florez et al.
    http://dx.doi.org/10.1038/ncomms11759
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 550
inc_a_y = inc_a_x
```

```
inc_shape = 'circular'

num_EM_modes = 20
num_AC_modes = 40
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

# 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.Air,
                        inc_a_material=materials.SiO2,
                        lc_bkg=3, lc2=2000.0, lc3=20.0)

# Expected effective index of fundamental guided mode.
n_eff = 1.4

# Calculate Electromagnetic Modes
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)

plotting.plt_mode_fields(sim_EM_wguide, xlim_min=0.4, xlim_max=0.4,
                         ylim_min=0.4, ylim_max=0.4, EM_AC='EM', pdf_png='pdf')

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff = ", np.round(n_eff_sim, 4))

k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

shift_Hz = 4e9

# Calculate Acoustic Modes
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
    EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)

plotting.plt_mode_fields(sim_AC_wguide, EM_AC='AC', add_name='NW')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC_wguide.Eig_values)*1e-9,␣
↪4))

set_q_factor = 1000.

# Calculate interaction integrals and SBS gain for PE and MB effects combined,
# as well as just for PE, and just for MB.
SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
    sim_EM_wguide, sim_AC_wguide, k_AC,
    EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival, fixed_Q=set_q_factor)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 0  # GHz
freq_max = 12  # GHz
plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha, k_AC,
    EM_ival1, EM_ival2, AC_ival, freq_min=freq_min, freq_max=freq_max)
```

Tapered Fibre - Scanning Widths

```
plt.close()""" Replicating the results of
    Brillouin light scattering from surface acoustic waves in a subwavelength-
↪diameter optical fibre
    Beugnot et al.
    http://dx.doi.org/10.1038/ncomms6242
"""

import time
import datetime
import numpy as np
import sys
from multiprocessing import Pool
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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

# Select the number of CPUs to use in simulation.
num_cores = 5

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 5*wl_nm
unitcell_y = unitcell_x
inc_shape = 'circular'

num_EM_modes = 20
num_AC_modes = 70
EM_ival1 = 0
EM_ival2 = EM_ival1
AC_ival = 'All'

# Expected effective index of fundamental guided mode.
n_eff = 1.18

freq_min = 4
freq_max = 12

width_min = 600
width_max = 2000
num_widths = 300
inc_a_x_range = np.linspace(width_min, width_max, num_widths)
num_interp_pts = 2000


def modes_n_gain(inc_a_x):
    inc_a_y = inc_a_x
    # 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.Air,
                            inc_a_material=materials.Si,
                            lc_bkg=3, lc2=1000.0, lc3=5.0)

    sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff=n_eff)
    k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])
    shift_Hz = 4e9
    sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC=k_AC,
        EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)

    set_q_factor = 600.
    SBS_gain, SBS_gain_PE, SBS_gain_MB, alpha = integration.gain_and_qs(
        sim_EM_wguide, sim_AC_wguide, k_AC,
        EM_ival1=EM_ival1, EM_ival2=EM_ival2, AC_ival=AC_ival, fixed_Q=set_q_factor)

    interp_values = plotting.gain_specta(sim_AC_wguide, SBS_gain, SBS_gain_PE, SBS_
↪gain_MB, alpha, k_AC,
        EM_ival1, EM_ival2, AC_ival, freq_min, freq_max, num_interp_pts=num_interp_
↪pts)

    return interp_values

# Run widths in parallel across num_cores CPUs using multiprocessing package.
pool = Pool(num_cores)
width_objs = pool.map(modes_n_gain, inc_a_x_range)


gain_array = np.zeros((num_interp_pts, num_widths))
for w, width_interp in enumerate(width_objs):
    gain_array[:,w] = width_interp[::-1]

fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
blah = ax1.matshow(gain_array, aspect='auto', interpolation = 'none')

num_xticks = 5
num_yticks = 5
ax1.xaxis.set_ticks_position('bottom')
ax1.set_xticks(np.linspace(0,(num_widths-1),num_xticks))
ax1.set_yticks(np.linspace((num_interp_pts-1),0,num_yticks))
ax1.set_xticklabels(["%4.0f" % i for i in np.linspace(width_min,width_max,num_
↪xticks)])
ax1.set_yticklabels(["%4.0f" % i for i in np.linspace(freq_min,freq_max,num_yticks)])

plt.xlabel(r'Width ($\mu m$)')
plt.ylabel('Frequency (GHz)')
plt.savefig('gain-width_scan.pdf')
plt.close()
```

Waveguide on a Pedestal

```
plotting.plt_mode_fields(sim_AC_wguide, EM_AC='AC', add_name='slab', pdf_png='png')
""" Replicating the results of
    Interaction between light and highly confined hypersound in a silicon photonic␣
↪nanowire
    Van Laer et al.
```

```
    http://dx.doi.org/10.1038/nphoton.2015.11
"""

import time
import datetime
import numpy as np
import sys
sys.path.append("../backend/")
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

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


# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 2.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 450
inc_a_y = 230
inc_shape = 'rectangular'
slab_a_x = 15
slab_a_y = 300
slab_b_x = unitcell_x-400
slab_b_y = 800

num_EM_modes = 20
num_AC_modes = 80
EM_ival1=0
EM_ival2=EM_ival1
AC_ival='All'

# Material parameters as in paper
# Silicon
n = 3.5
s = 2330  # kg/m3
c_11 = 166e9; c_12 = 64e9; c_44 = 79e9  # Pa
p_11 = -0.09; p_12 = 0.017; p_44 = -0.051
eta_11 = 5.9e-3 ; eta_12 = 5.16e-3 ; eta_44 = 0.620e-3  # Pa
Si_props = [n, s, c_11, c_12, c_44, p_11, p_12, p_44,
                  eta_11, eta_12, eta_44]

# Silica
n = 1.44
s = 2203  # kg/m3
c_11 = 78e9; c_12 = 16e9; c_44 = 31e9
p_11 = 0.12; p_12 = 0.270; p_44 = -0.073
eta_11 = 1.6e-3 ; eta_12 = 1.29e-3 ; eta_44 = 0.16e-3  # Pa s
SiO2_props = [n, 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,
                        slab_a_x=slab_a_x, slab_a_y=slab_a_y,
                        slab_b_x=slab_b_x, slab_b_y=slab_b_y,
                        bkg_material=materials.Air,
                        inc_a_material=materials.Material(Si_props),
                        slab_a_material=materials.Material(SiO2_props),
                        slab_a_bkg_material=materials.Air,
                        slab_b_material=materials.Material(SiO2_props),
                        slab_b_bkg_material=materials.Air,
                        lc_bkg=2, lc2=4000.0, lc3=20.0)

# Expected effective index of fundamental guided mode.
n_eff = wguide.inc_a_material.n-0.1

# Calculate Electromagnetic Modes
sim_EM_wguide = wguide.calc_EM_modes(wl_nm, num_EM_modes, n_eff)

# plotting.plt_mode_fields(sim_EM_wguide,
#                          xlim_min=0.35, xlim_max=0.35, ylim_min=0.1, ylim_max=0.55,
#                          EM_AC='EM', add_name='slab', pdf_png='pdf')

# Print the wavevectors of EM modes.
print('k_z of EM modes \n', np.round(np.real(sim_EM_wguide.Eig_values), 4))

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_wguide.Eig_values[0]*((wl_nm*1e-9)/(2.*np.pi)))

k_AC = 2*np.real(sim_EM_wguide.Eig_values[0])

shift_Hz = 10e9

# Calculate Acoustic Modes
sim_AC_wguide = wguide.calc_AC_modes(wl_nm, num_AC_modes, k_AC,
    EM_sim=sim_EM_wguide, shift_Hz=shift_Hz)

plotting.plt_mode_fields(sim_AC_wguide, EM_AC='AC', add_name='slab', pdf_png='png')
```

CHAPTER

FOUR

PYTHON BACKEND

objects module

objects.py is a subroutine of NumBAT. It contains the Struct objects that represent the structure being simulated.

class objects.**NumBAT**
Bases: object

class objects.**Struct**(*unitcell_x*, *inc_a_x*, *unitcell_y=None*, *inc_a_y=None*, *inc_shape='rectangular'*, *slab_a_x=None*, *slab_a_y=None*, *slab_b_x=None*, *slab_b_y=None*, *coat_y=None*, *inc_b_x=None*, *inc_b_y=None*, *two_inc_sep=None*, *incs_y_offset=None*, *inc_c_x=None*, *inc_d_x=None*, *inc_e_x=None*, *inc_f_x=None*, *inc_g_x=None*, *inc_h_x=None*, *inc_i_x=None*, *inc_j_x=None*, *inc_k_x=None*, *inc_l_x=None*, *inc_m_x=None*, *inc_n_x=None*, *inc_o_x=None*, *bkg_material=<materials.Material object>*, *inc_a_material=<materials.Material object>*, *slab_a_material=<materials.Material object>*, *slab_a_bkg_material=<materials.Material object>*, *slab_b_material=<materials.Material object>*, *slab_b_bkg_material=<materials.Material object>*, *coat_material=<materials.Material object>*, *inc_b_material=<materials.Material object>*, *inc_c_material=<materials.Material object>*, *inc_d_material=<materials.Material object>*, *inc_e_material=<materials.Material object>*, *inc_f_material=<materials.Material object>*, *inc_g_material=<materials.Material object>*, *inc_h_material=<materials.Material object>*, *inc_i_material=<materials.Material object>*, *inc_j_material=<materials.Material object>*, *inc_k_material=<materials.Material object>*, *inc_l_material=<materials.Material object>*, *inc_m_material=<materials.Material object>*, *inc_n_material=<materials.Material object>*, *inc_o_material=<materials.Material object>*, *loss=True*, *make_mesh_now=True*, *force_mesh=True*, *mesh_file='NEED_FILE.mail'*, *check_msh=False*, *lc_bkg=0.09*, *lc2=1.0*, *lc3=1.0*, *lc4=1.0*, *lc5=1.0*, *lc6=1.0*, *plotting_fields=False*, *plot_real=1*, *plot_imag=0*, *plot_abs=0*, *plot_field_conc=False*)
Bases: object

Represents a structured layer.

Parameters

- **unitcell_x** (*float*) – The horizontal period of the unit cell in nanometers.
- **inc_a_x** (*float*) – The horizontal diameter of the inclusion in nm.

Keyword Arguments

- **unitcell_y** (*float*) – The vertical period of the unit cell in nanometers. If None, unitcell_y = unitcell_x.
- **inc_a_y** (*float*) – The vertical diameter of the inclusion in nm.
- **inc_shape** (*str*) – Shape of inclusions that have template mesh, currently: 'circular', 'rectangular', 'slot'. Rectangular is default.
- **slab_a_x** (*float*) – The horizontal diameter in nm of the slab directly below the inclusion.
- **slab_a_y** (*float*) – The vertical diameter in nm of the slab directly below the inclusion.
- **slab_b_x** (*float*) – The horizontal diameter in nm of the slab separated from the inclusion by slab_a.
- **slab_b_y** (*float*) – The vertical diameter in nm of the slab separated from the inclusion by slab_a.
- **two_inc_sep** (*float*) – Separation between edges of inclusions in nm.
- **incs_y_offset** (*float*) – Vertical offset between centers of inclusions in nm.
- **coat_y** (*float*) – The thickness of any coat layer around the inclusion.
- **WHAT IS RADIUS WHAT IS DIAMETER** (*SPECIFY*) –
- **background** – A :Material: instance for the background medium.
- **inc_a_material** – A :Material: instance for the
- **inc_b_material** – A :Material: instance for the
- **slab_a_material** – A :Material: instance for the
- **slab_a_bkg_material** – A :Material: instance for the
- **slab_b_material** – A :Material: instance for the
- **slab_b_bkg_material** – A :Material: instance for the
- **coat_material** – A :Material: instance for the
- **bkg_AC** – A list of acoustic parameters
- **inc_a_AC** – A list of acoustic parameters
- **slab_a_AC** – A list of acoustic parameters
- **slab_a_bkg_AC** – A list of acoustic parameters
- **slab_b_AC** – A list of acoustic parameters
- **slab_b_bkg_AC** – A list of acoustic parameters
- **loss** (*bool*) – If False, Im(n) = 0, if True n as in :Material: instance.
- **make_mesh_now** (*bool*) – 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** (*bool*) – If True, a new mesh is created despite existence of mesh with same parameter. This is used to make mesh with equal period etc. but different lc refinement.

- **mesh_file** (*str*) – If using a set pre-made mesh give its name including .mail if 2D_array (eg. 600_60.mail), or .txt if 1D_array. It must be located in backend/fortran/msh/
- **lc_bkg** (*float*) – Length constant of meshing of background medium (smaller = finer mesh)
- **lc2** (*float*) – factor by which lc_bkg will be reduced on inclusion surfaces; lc_surface = cl_bkg / lc2.
- **lc3-6'** (*float*) – factor by which lc_bkg will be reduced at center of inclusions.
- **plotting_fields** (*bool*) – Unless set to true field data deleted. Also plots modes (ie. FEM solutions) in gmsh format. Plots epsilon*|E|^2 & choice of real/imag/abs of x,y,z components & field vectors. Fields are saved as gmsh files, but can be converted by running the .geo file found in Bloch_fields/PNG/
- **plot_real** (*bool*) – Choose to plot real part of modal fields.
- **plot_imag** (*bool*) – Choose to plot imaginary part of modal fields.
- **plot_abs** (*bool*) – Choose to plot absolute value of modal fields.

calc_AC_modes(*wl_nm*, *num_modes*, *k_AC*, *shift_Hz=None*, *EM_sim=None*, ***args*)

Run a simulation to find the Struct's acoustic modes.

Parameters

- **wl_nm** (*float*) – Wavelength of AC wave in vacuum.
- **num_modes** (*int*) – Number of AC modes to solve for.

Keyword Arguments

- **shift_Hz** (*float*) – Guesstimated frequency of modes, will be origin of FEM search. NumBAT will make an educated guess if shift_Hz=None. (Technically the shift and invert parameter).
- **(** (*EM_sim*) – Simmo: object): Typically an acoustic simulation follows on from an optical one. Supply the EM :Simmo: object so the AC FEM mesh can be constructed from this. This is done by removing vacuum regions.

Returns Simmo: object

calc_EM_modes(*wl_nm*, *num_modes*, *n_eff*, ***args*)

Run a simulation to find the Struct's EM modes.

Parameters

- **wl_nm** (*float*) – Wavelength of EM wave in vacuum.
- **num_modes** (*int*) – Number of EM modes to solve for.
- **n_eff** (*float*) – Guesstimated effective index of fundamental mode, will be origin of FEM search.

Returns Simmo: object

make_mesh()

Take the parameters specified in python and make a Gmsh FEM mesh. Creates a .geo and .msh file, then uses Fortran conv_gmsh routine to convert .msh into .mail, which is used in NumBAT FEM routine.

objects.**dec_float_str**(*dec_float*)

Convert float with decimal point into string with '_' in place of '.'

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**(*mat_props*)

Bases: object

Represents a material with a refractive index n.

TODO

Parameters **todo** –

Currently included materials are;

Semiconductors
Si
SiO2
As2S3
GaAs

materials.**isotropic_stiffness**(*E*, *v*)

Calculate the stiffness matrix components of isotropic materials, given the two free parameters: E: Youngs_modulus v: Poisson_ratio

Ref: http://www.efunda.com/formulae/solid_mechanics/mat_mechanics/hooke_isotropic.cfm

mode_calcs module

mode_calcs.py is a subroutine of NumBAT that contains methods to calculate the EM and Acoustic modes of a structure.

class mode_calcs.**NumBAT**

Bases: object

class mode_calcs.**Simmo**(*structure*, *wl_nm*, *num_modes=20*, *n_eff=None*, *shift_Hz=None*, *k_AC=None*, *EM_sim=None*)

Bases: object

Calculates the modes of a :Struc: object at a wavelength of wl_nm.

calc_AC_modes()

Run a Fortran FEM calculation to find the acoustic modes.

Returns a :Simmo: object that now has these key values:

Eig_values: a 1d array of Eigenvalues (frequencies) in [1/s]

sol1: the associated Eigenvectors, ie. the fields, stored as [field comp, node nu on element, Eig value, el nu]

calc_EM_modes()

Run a Fortran FEM calculation to find the optical modes.

Returns a :Simmo: object that now has these key values:

Eig_values: a 1d array of Eigenvalues (propagation constants) in [1/m]

sol1: the associated Eigenvectors, ie. the fields, stored as [field comp, node nu on element, Eig value, el nu]

integration module

mode_calcs.py is a subroutine of NumBAT that contains methods to calculate the EM and Acoustic modes of a structure.

class integration.**NumBAT**
Bases: object

integration.**gain_and_qs**(*sim_EM_wguide*, *sim_AC_wguide*, *k_AC*, *EM_ival1=0*, *EM_ival2=0*, *AC_ival=0*, *fixed_Q=None*, *typ_select_out=None*)
Calculate interaction integrals and SBS gain.

Implements Eqs. 33, 41, 45, 91 of Wolff et al. PRA 92, 013836 (2015) doi/10.1103/PhysRevA.92.013836 These are for Q_photoelastic, Q_moving_boundary, the Acoustic loss "alpha", and the SBS gain respectively.

Note there is a sign error in published Eq. 41. Also, in implementing Eq. 45 we use integration by parts, with a boundary integral term set to zero on physical grounds, and filled in some missing subscripts. We prefer to express Eq. 91 with the Lorentzian explicitly visible, which makes it clear how to transform to frequency space. The final integrals are

$$Q^{\mathrm{PE}} = \varepsilon_0 \int_A \mathrm{d}^2 r \sum_{ijkl} \varepsilon_r^2 e_i e_j p_{ijkl} \partial_k u_l^*,$$

$$Q^{\mathrm{MB}} = \int_C \mathrm{d}\mathbf{r} (\mathbf{u}^* \cdot \hat{\mathrm{n}}) \big[(\varepsilon_a - \varepsilon_b) \varepsilon_0 (\hat{\mathrm{n}} \times \mathbf{e}) \cdot (\hat{\mathrm{n}} \times \mathbf{e}) - (\varepsilon_a^{-1} - \varepsilon_b^{-1}) \varepsilon_0^{-1} (\hat{\mathrm{n}} \cdot \mathbf{d}) \cdot (\hat{\mathrm{n}} \cdot \mathbf{d}) \big],$$

$$\alpha = \frac{\Omega^2}{P_b} \int \mathrm{d}^2 r \sum_{ijkl} \partial_i u_j^* \eta_{ijkl} \partial_k u_l,$$

$$\Gamma = \frac{2\omega\Omega \mathrm{Re}(Q_1 Q_1^*)}{P_e P_e P_{ac}} \frac{1}{\alpha} \frac{\alpha^2}{\alpha^2 + \kappa^2}.$$

Parameters

- **(** (*sim_AC_wguide*) – Simmo: object): Contains all info on EM modes
- **(** – Simmo: object): Contains all info on AC modes
- **k_AC** (*float*) – Propagation constant of acoustic modes.

Keyword Arguments

- **EM_ival1** (*int/string*) – Specify mode number of EM mode 1 (pump mode) to calculate interactions for. Numbering is python index so runs from 0 to num_EM_modes-1, with 0 being fundamental mode (largest prop constant). Can also set to 'All' to include all modes.
- **EM_ival2** (*int/string*) – Specify mode number of EM mode 2 (stokes mode) to calculate interactions for. Numbering is python index so runs from 0 to num_EM_modes-1, with 0 being fundamental mode (largest prop constant). Can also set to 'All' to include all modes.

- **AC_ival** (*int/string*) – Specify mode number of AC mode to calculate interactions for. Numbering is python index so runs from 0 to num_AC_modes-1, with 0 being fundamental mode (largest prop constant). Can also set to 'All' to include all modes.
- **fixed_Q** (*int*) – Specify a fixed Q-factor for the AC modes, rather than calculating the acoustic loss (alpha).

Returns The SBS gain including both photoelastic and moving boundary contributions. SBS_gain_PE (num_modes_EM,num_modes_EM,num_modes_AC): The SBS gain for only the photoelastic effect. SBS_gain_MB (num_modes_EM,num_modes_EM,num_modes_AC): The SBS gain for only the moving boundary effect. alpha (num_modes_AC): The acoustic loss for each mode.

Return type SBS_gain (num_modes_EM,num_modes_EM,num_modes_AC)

integration.**symmetries**(*sim_wguide*, *n_points=10*)

Plot EM mode fields.

Parameters **sim_wguide** – A :Struct: instance that has had calc_modes calculated

Keyword Arguments **n_points** (*int*) – The number of points across unitcell to interpolate the field onto.

plotting module

plotting.py is a subroutine of NumBAT that contains numerous plotting routines.

Copyright (C) 2016 Bjorn Sturmberg

plotting.**gain_specta**(*sim_AC_wguide*, *SBS_gain*, *SBS_gain_PE*, *SBS_gain_MB*, *alpha*, *k_AC*, *EM_ival1*, *EM_ival2*, *AC_ival*, *freq_min*, *freq_max*, *num_interp_pts=3000*, *pdf_png='png'*, *add_name=''*)

Construct the SBS gain spectrum, built from Lorentzian peaks of the individual modes. Note the we use the spectral linewidth of the resonances

$$\gamma = v_g \alpha$$

where v_g the group velocity of the mode and $theta$ is the detuning frequency. We transform from k-space of Eq. 91 to frequency space

$$\Gamma = \frac{2\omega\Omega \mathrm{Re}(Q_1 Q_1^*)}{P_e P_e P_{ac}} \frac{1}{\alpha} \frac{\alpha^2}{\alpha^2 + \kappa^2},$$
$$\Gamma = \frac{2\omega\Omega \mathrm{Re}(Q_1 Q_1^*)}{P_e P_e P_{ac}} \frac{1}{\alpha} \frac{\gamma^2}{\gamma^2 + \theta^2},$$

Parameters

- **sim_AC_wguide** – An AC :Struct: instance that has had calc_modes calculated
- **SBS_gain** (*array*) – Totlat SBS gain of modes.
- **SBS_gain_PE** (*array*) – Moving Bountary gain of modes.
- **SBS_gain_MB** (*array*) – Photoelastic gain of modes.
- **alpha** (*array*) – Acoustic loss of each mode.
- **k_AC** (*float*) – Acoustic wavevector.
- **freq_min** (*float*) – Minimum of frequency range.

- **freq_max** (*float*) – Maximum of frequency range.

Keyword Args:

plotting.**plot_msh**(*x_arr*, *add_name=''*)
Plot EM mode fields.

Parameters **sim_wguide** – A :Struct: instance that has had calc_modes calculated

Keyword Arguments **n_points** (*int*) – The number of points across unitcell to interpolate the field onto.

plotting.**plt_mode_fields**(*sim_wguide*, *n_points=500*, *quiver_steps=50*, *xlim_min=None*, *xlim_max=None*, *ylim_min=None*, *ylim_max=None*, *EM_AC='EM'*, *stress_fields=False*, *pdf_png='png'*, *add_name=''*)
Plot EM mode fields. NOTE: z component of EM field needs comes scaled by 1/(i beta), which must be reintroduced!

Args: sim_wguide : A :Struct: instance that has had calc_modes calculated

Keyword Args: n_points (int): The number of points across unitcell to interpolate the field onto.

xlim_min (float): Limit plotted xrange to xlim_min:(1-xlim_max) of unitcell

xlim_max (float): Limit plotted xrange to xlim_min:(1-xlim_max) of unitcell

ylim_min (float): Limit plotted yrange to ylim_min:(1-ylim_max) of unitcell

ylim_max (float): Limit plotted yrange to ylim_min:(1-ylim_max) of unitcell

EM_AC (str): Either 'EM' or 'AC' modes

stress_fields (bool): Calculate acoustic stress fields

pdf_png (str): File type to save, either 'png' or 'pdf'

add_name (str): Add a string to the file name.

plotting.**zeros_int_str**(*zero_int*)
Convert integer into string with '0' in place of ' '.

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 modal fields. However, there are a few important things to know about the workings of these routines.

FEM Mode Solvers

Making New Mesh

At some point you may well wish to study a structure that is not described by an existing NumBAT mesh template. In this section we provide an example of how to create a new mesh. In this case we will create a rib waveguide that is has a coating surrounding the guiding region.

Creating a mesh is typically a two step process: first we define the points that define the outline of the structures, then we define the lines connecting the points and the surfaces formed out of the lines. The first step is best done in a text editor in direct code, while the second can be done using the open source program gmsh GUI.

To start we are going to make a copy of NumBAT/backend/fortran/msh/empty_msh_template.geo

```
$ cd NumBAT/backend/fortran/msh/
$ cp empty_msh_template.geo rib_coated_msh_template.geo
```

Step 1

Opening the new file in a text editor you see it contains points defining the unit cell. The points are defined as

```
Point(1) = {x, y, z, meshing_value}
```

We start by adding the two points that define the top of the substrate (the bottom will be the bottom edge of the unit cell at {0, -h} and {d, -h}). We use a placeholder slab thickness of 100 nm, which is normalised by the width of the unit cell.

```
slab1 = 100;
s1 = slab1/d_in_nm;
Point(5) = {0, -h+s1, 0, lc};
Point(6) = {d, -h+s1, 0, lc};
```

We then add a further layer on top of the bottom slab, this time using a placeholder thickness of 50 nm. Note that each point must be labeled by a unique number.:

```
slab2 = 50;
s2 = slab2/d_in_nm;
Point(7) = {0, -h+s1+s2, 0, lc};
Point(8) = {d, -h+s1+s2, 0, lc};
```

We next define the peak of the rib, which involves a width and a height,

```
ribx = 200;
riby = 30;
rx = ribx/d_in_nm;
ry = riby/d_in_nm;
Point(9) = {d/2-rx/2, -h+s1+s2, 0, lc2};
Point(10) = {d/2+rx/2, -h+s1+s2, 0, lc2};
Point(11) = {d/2-rx/2, -h+s1+s2+ry, 0, lc2};
Point(12) = {d/2+rx/2, -h+s1+s2+ry, 0, lc2};
```

Lastly we coat the whole structure with a conformal layer.

```
coatx = 20;
coaty = 20;
cx = coatx/d_in_nm;
cy = coaty/d_in_nm;
Point(13) = {0, -h+s1+s2+cy, 0, lc};
Point(14) = {d, -h+s1+s2+cy, 0, lc};
Point(15) = {d/2-rx/2-cx, -h+s1+s2+cy, 0, lc};
Point(16) = {d/2+rx/2+cx, -h+s1+s2+cy, 0, lc};
Point(17) = {d/2-rx/2-cx, -h+s1+s2+2*cy, 0, lc};
Point(18) = {d/2+rx/2+cx, -h+s1+s2+2*cy, 0, lc};
```

Step 2

To create the lines that connect the points, and the mesh surfaces it is easiest to use gmsh (although it can also be written directly in code). Open your geometry file in gmsh:

```
NumBAT/backend/fortran/msh$ gmsh rib_coated_msh_template.geo
```

Navigate through the side menu to Modules/Geometry/Elementary entities/Add and click “Straight line”. Now click consecutively on the point you wish to connect.

Navigate through the side menu to Modules/Geometry/Elementary entities/Add and click “Plane surface”. Now click on the boundary of each enclosed area.

Navigate through the side menu to Modules/Geometry/Physical groups/Add and click “Line”. Now click on the lines that make up each side of the unit cell boundary, pressing the “e” key to end your selection once the each side is fully highlighted.

Navigate through the side menu to Modules/Geometry/Physical groups/Add and click “Surface”. Now click on all the surfaces of a given material type (in this example there is only one surface per material). It is crucial to remember the order you defined the physical surfaces in. Now open the .geo file in a word editor, scroll to the bottom, and change the numbering of the physical surfaces so that the background material corresponds to 1, the waveguide is 2, the bottom substrate is 3, and the cladding is 4.

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).

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

PYTHON MODULE INDEX

INDEX