NumBAT Documentation

Release 0.1

Bjorn Sturmberg

Aug 30, 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 16.04 with the following package versions: Python 3.5.3, Numpy 1.11.0, Suitesparse 4.4.6, and Gmsh 2.10.1. It has also been successfully installed by users on Debian, RedHat and on Windows 10 (installing Ubuntu after enabling the Windows Subystem for Linux - steps 3 here https://msdn.microsoft.com/en-au/commandline/wsl/install_guide) and with different versions of packages, but these installations have not been as thoroughly documented so may require user testing.

In general, you can simply run the setup script

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

from the NumBAT/ directory.

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/
$ nosetests3
```

where the <dependencies> packages are listed dependencies.txt. Note that it is safer to pip install matplotlib than apt-get'ing as will install matplotlib 2.0 without conflicting older versions.

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:

```
$ python3 simo.py
```

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

```
$ python3 -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

Geometries

To review how material types and physical dimensions are represented in the mesh geometries go to:

```
>>>  NumBAT/docs/msh_type_lib
```

and view the relevant .png file.

The parameters lc_bkg, lc2, lc3 set the fineness of the FEM mesh. lc_bkg sets the reference background mesh size, larger lc_bkg = larger (more coarse) mesh. In NumBAT the x-dimension of the unit cell is traditionally normalised to unity, in which case there will be lc_bkg mesh elements along the horizontal outside edge; in other words the outside edge is divided into lc_bkg elements. At the interface between materials the mesh is refined to be lc_bkg/lc2, therefore larger lc2 = finer mesh at these interfaces. The meshing program automatically adjusts the mesh size to smoothly transition from a point that has one mesh parameter to points that have other meshing parameters. The mesh

it typically also refined at the centers of important regions, eg in the center of a waveguide, which is done with lc3, which just like lc2 refines the mesh size at these points as lc_bkg/lc3.

Choosing appropriate values of lc_bkg, lc2, lc3 is crucial NumBAT to give accurate results. The values depend strongly on the type of structure being studied, and so it is recommended to carry out a convergence test before delving into new structures (see Tutorial 5) starting from similar parameters as used in the tutoarial simulations. You can also visually check the resolution of your mesh by setting check_msh=True when you define your objects.Struct (see Tutorial 1), or by running the following command

```
NumBAT/backend/fortran/msh$ gmsh <msh_name>.msh
```

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 python3
```

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

This example, contained in `tutorials/simo-tut_01-first_calc.py` calculates the backward SBS gain for a rectangular silicon waveguide surrounded by air.

The sequence of operations is

1. Import NumBAT modules
2. Define the structure shape and dimensions
3. Specify the electromagnetic and acoustic modes to be solved for
4. Construct the waveguide with `objects.Struct`
5. Solve the electromagnetic problem. `mode_calcs.calc_EM_modes` returns an object containing modes and their propagation constants as `Eig_values` in 1/m.
6. Convert the EM eigenvalue to an effective index
7. Identify the desired acoustic wavenumber and solve the acoustic problem. `mode_calcs.calc_AC_modes` returns an object containing the modes for propagation constant `k_AC` and acoustic frequencies as `Eig_values` in Hz.
8. Calculate the total SBS gain, contributions from photoelasticity and moving boundary effects, and the acoustic loss

```
""" 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 wavevector

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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
# Number of acoustic modes to solve for.
num_modes_AC = 20
# The EM pump mode(s) for which to calculate interaction with AC modes.
# Can specify a mode number (zero has lowest propagation constant) or 'All'.
EM_ival_pump = 0
# The EM Stokes mode(s) for which to calculate interaction with AC modes.
EM_ival_Stokes = 0
# 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,
                        material_bkg=materials.Vacuum,
                        material_a=materials.Si_2016_Smith,
                        lc_bkg=2, lc2=200.0, lc3=5.0, check_mesh=False)

# Explicitly remind ourselves what data we're using.
print('\nUsing %s material data from' % wguide.material_a.chemical)
print('Author:', wguide.material_a.author)
print('Year:', wguide.material_a.date)
print('Ref:', wguide.material_a.doi)

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

# Calculate the Electromagnetic modes of the pump field.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
# Print the wavevectors of EM modes.
print('\n k_z of EM modes \n', np.round(np.real(sim_EM_pump.Eig_values),4))
# Calculate the Electromagnetic modes of the Stokes field.
# For an idealised backward SBS simulation the Stokes modes are identical
# to the pump modes but travel in the opposite direction.
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
# # Alt
# sim_EM_Stokes = wguide.calc_EM_modes(wl_nm, num_modes_EM_Stokes, n_eff, Stokes=True)

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_pump.Eig_values[0]*((wl_nm*1e-9)/(2.*np.pi)))
print("\n Fundamental optical mode ")
print(" n_eff = ", np.round(n_eff_sim, 4))
# Acoustic wavevector
k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
print('\n AC wavenumber (1/m) = ', np.round(k_AC, 4))

# Calculate Acoustic modes, using the mesh from the EM calculation.
```

```
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
# Print the frequencies of AC modes.
print('\n Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.Eig_values)*1e-9, 4))

# 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. Also calculate acoustic loss alpha.
SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC, EM_ival_pump=EM_ival_pump,
    EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival, fixed_Q=set_q_factor)
# Print the Backward SBS gain of the AC modes.
print("\n SBS_gain PE contribution \n", SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:])
print("SBS_gain MB contribution \n", SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:])
print("SBS_gain total \n", SBS_gain[EM_ival_pump,EM_ival_Stokes,:])
# Mask negligible gain values to improve clarity of print out.
threshold = -1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival_pump,EM_ival_Stokes,:], 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)
print("SBS_gain linewidth [Hz] \n", linewidth_Hz)

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

SBS Gain Spectra

This example, contained in `tutorials/simo-tut_02-gain_spectra-npsave.py` considers the same structure but adds plotting of fields, gain spectra and techniques for saving and reusing data from earlier calculations.

Elements to note:

1. **Both electric and magnetic fields can be selected using `'EM_E'` or `'EM_H'` as the value of ‘‘EM_AC‘ in** `plotting.mode_fields`.
2. `np.savez` and `np.load` allow storage of arbitrary data between simulations.

```
""" 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.

    Show how to implement integrals in python
    and how to load data from Comsol.
"""
```

```
import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# 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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 20
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_02-'

# 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,
                        material_bkg=materials.Vacuum,
                        material_a=materials.Si_2016_Smith,
                        lc_bkg=2, lc2=200.0, lc3=5.0)


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

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

# 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_pump = npzfile['sim_EM_pump'].tolist()
```

```
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
# npzfile = np.load('wguide_data2.npz')
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

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

# Plot the E fields of the EM modes fields - specified with EM_AC='EM_E'.
# Zoom in on the central region (of big unitcell) with xlim_, ylim_ args.
# Only plot fields of fundamental (ival = 0) mode.
plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ylim_min=0.4,
                         ylim_max=0.4, ivals=[0], contours=True, EM_AC='EM_E',
                         prefix_str=prefix_str)
# Plot the H fields of the EM modes - specified with EM_AC='EM_H'.
plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ylim_min=0.4,
                         ylim_max=0.4, ivals=[0], EM_AC='EM_H', prefix_str=prefix_str)

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_pump.Eig_values[0]*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff", np.round(n_eff_sim, 4))
# Acoustic wavevector
k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

# Calculate Acoustic modes.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
# # Save calculated :Simmo: object for AC calculation.
# np.savez('wguide_data_AC', sim_AC=sim_AC)

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

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, EM_AC='AC', pdf_png='pdf', contours=True,
                         prefix_str=prefix_str)

# Calculate the acoustic loss from our fields.
# 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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC, EM_ival_pump=EM_ival_pump,
    EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)
# 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, linewidth_Hz=linewidth_Hz)

# # 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']
# linewidth_Hz = npzfile['linewidth_Hz']

# The following function shows how integrals can be implemented purely in python,
# which may be of interest to users wanting to calculate expressions not currently
# included in NumBAT. Note that the Fortran routines are much faster!
# Also shows how field data can be imported (in this case from Comsol) and used.
comsol_ivals = 5 # Number of modes contained in data file.
SBS_gain_PE_py, alpha_py, SBS_gain_PE_comsol, alpha_comsol = integration.gain_python(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC, 'Comsol_ac_modes_1-5.dat',
    comsol_ivals=comsol_ivals)

# Print the PE contribution to gain SBS gain of the AC modes.
# Mask negligible gain values to improve clarity of print out.
threshold = -1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:comsol_
↪ivals], 0, threshold)
print("\n\nSBS_gain PE NumBAT default (Fortran)\n", masked_PE)
masked = np.ma.masked_inside(SBS_gain_PE_py[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
print("SBS_gain python integration routines \n", masked)
masked = np.ma.masked_inside(SBS_gain_PE_comsol[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
print("SBS_gain from loaded Comsol data \n", masked)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = np.real(sim_AC.Eig_values[0])*1e-9 - 2  # GHz
freq_max = np.real(sim_AC.Eig_values[-1])*1e-9 + 2  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str)
# Zoomed in version
freq_min = 12  # GHz
freq_max = 14  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str, suffix_str='_zoom')

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

Investigating Dispersion and npsave npload

```
""" Calculate a dispersion diagram of the acoustic modes
    from k_AC ~ 0 (forward SBS) to k_AC = 2*k_EM (backward SBS).
    Load EM mode data from simo_tut_02.
"""
```

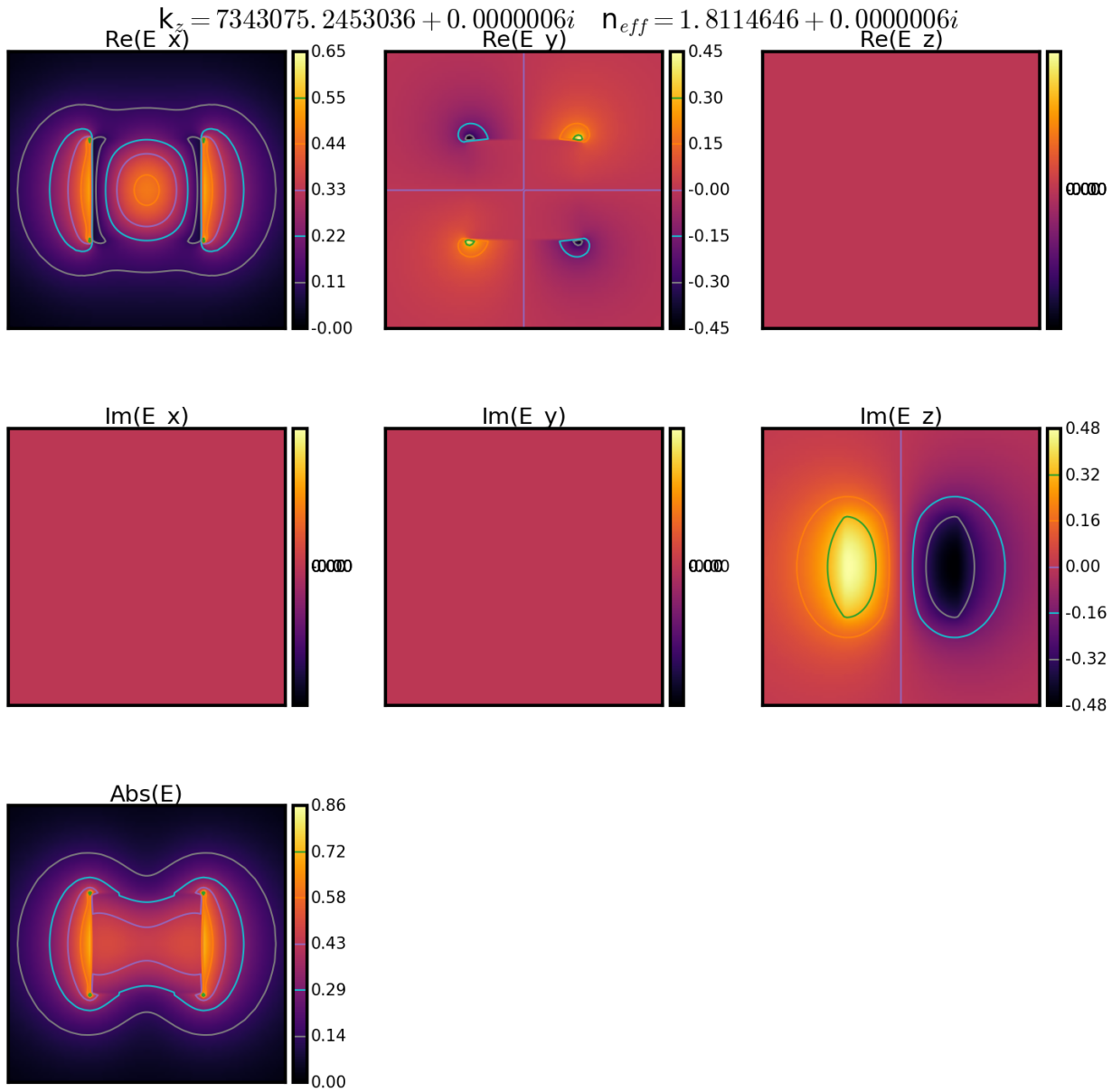

Fig. 3.1: Fundamental optical mode fields.

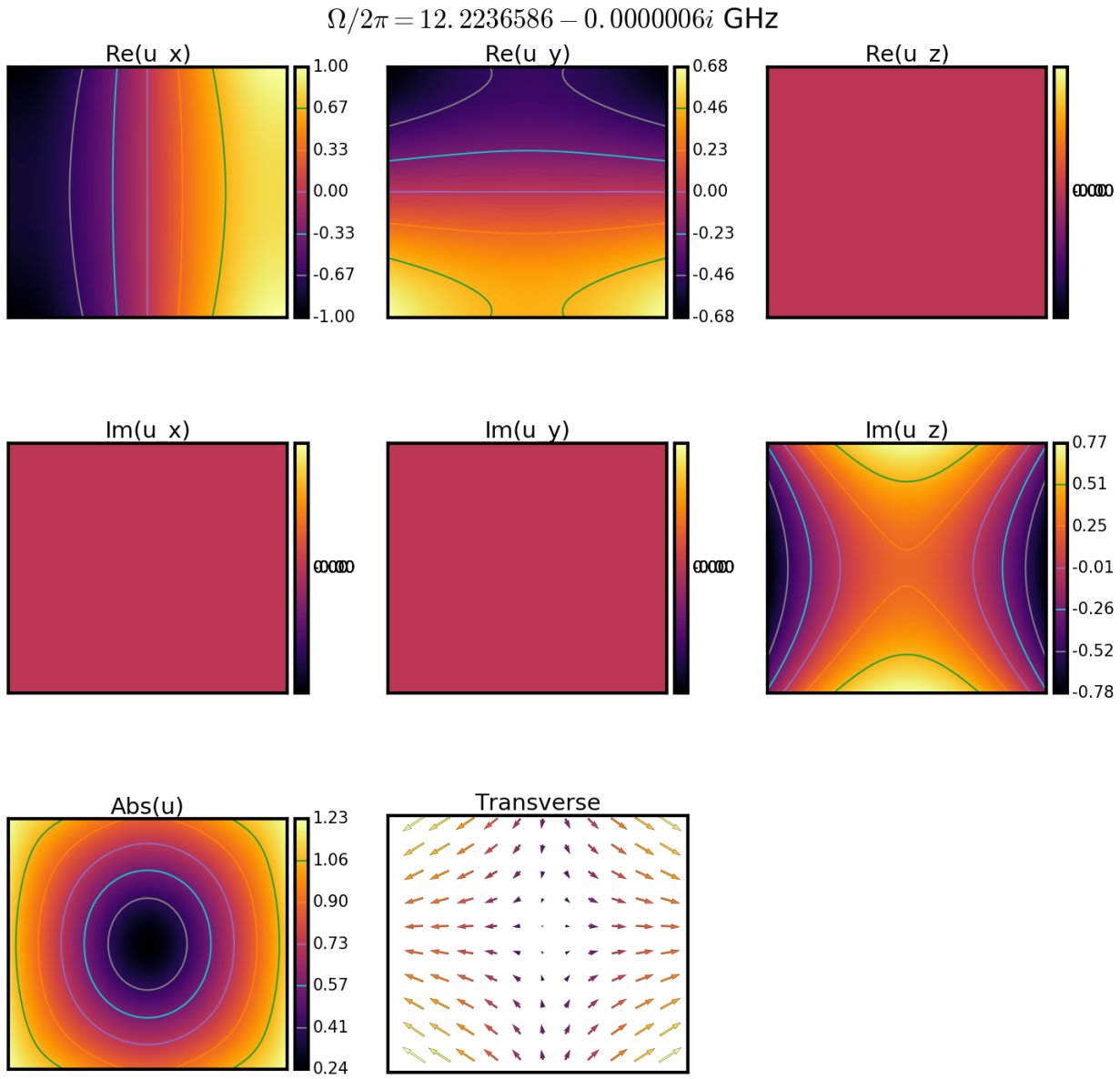

Fig. 3.2: Acoustic mode with high gain due to moving boundary effect.

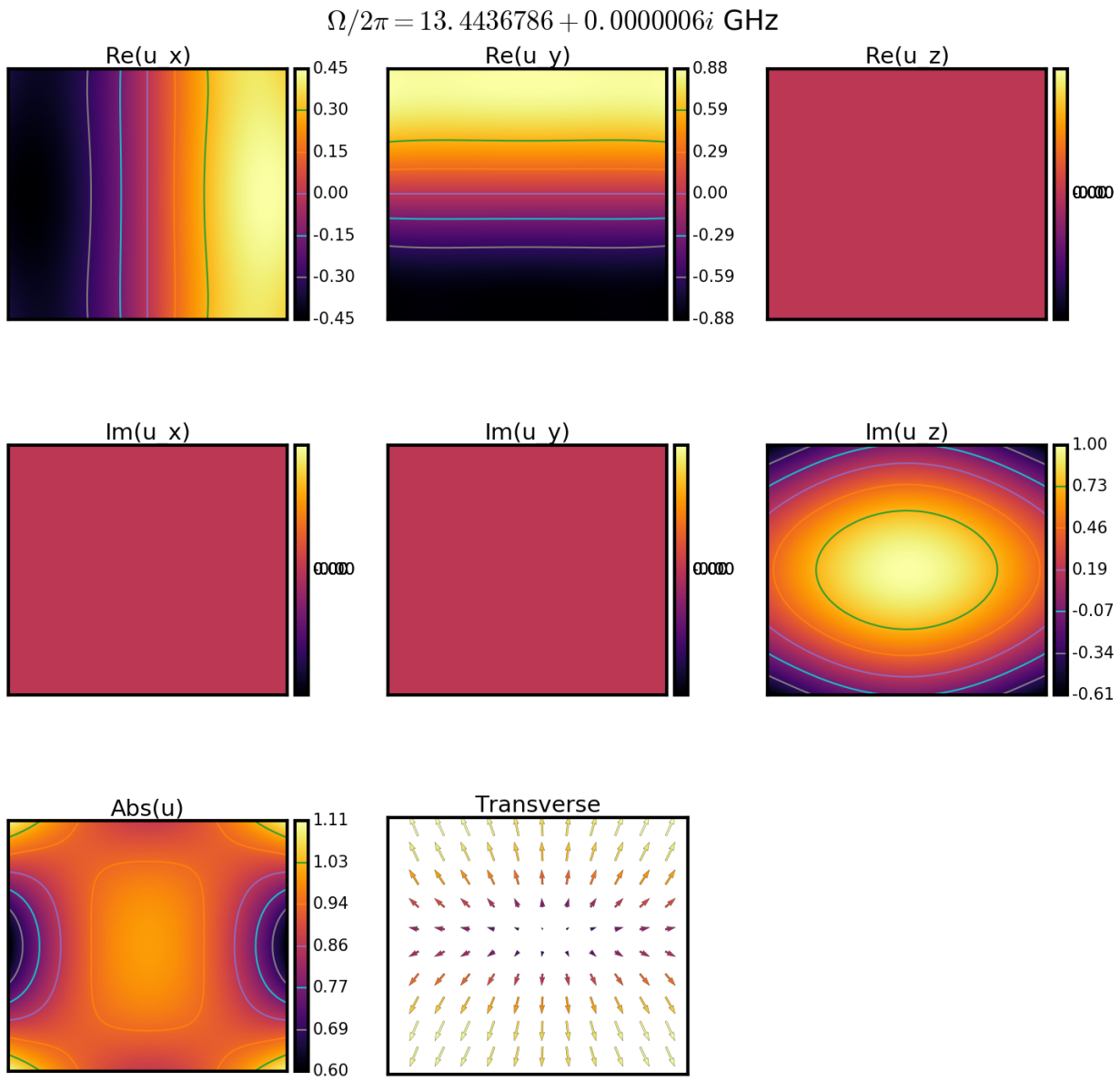

Fig. 3.3: Acoustic mode with high gain due to moving boundary effect.

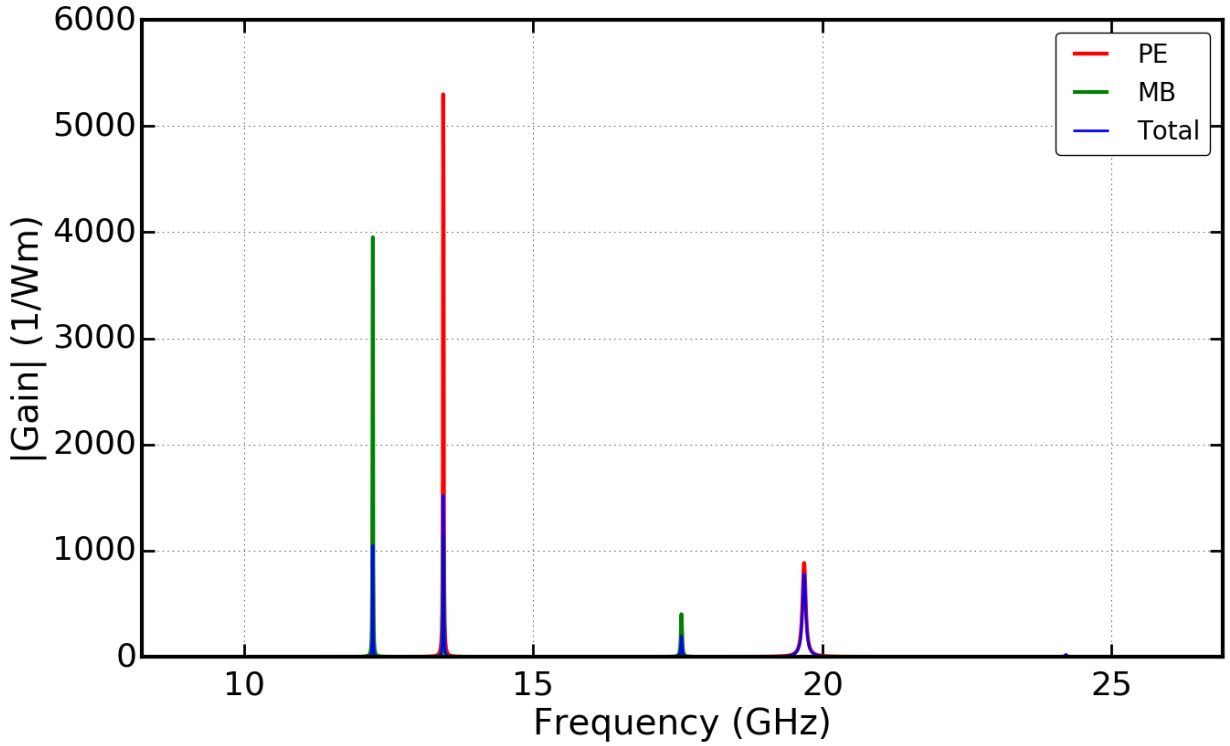

Fig. 3.4: Gain spectra showing gain due to photoelastic effec, gain due to moving boundary effect, and total gain.

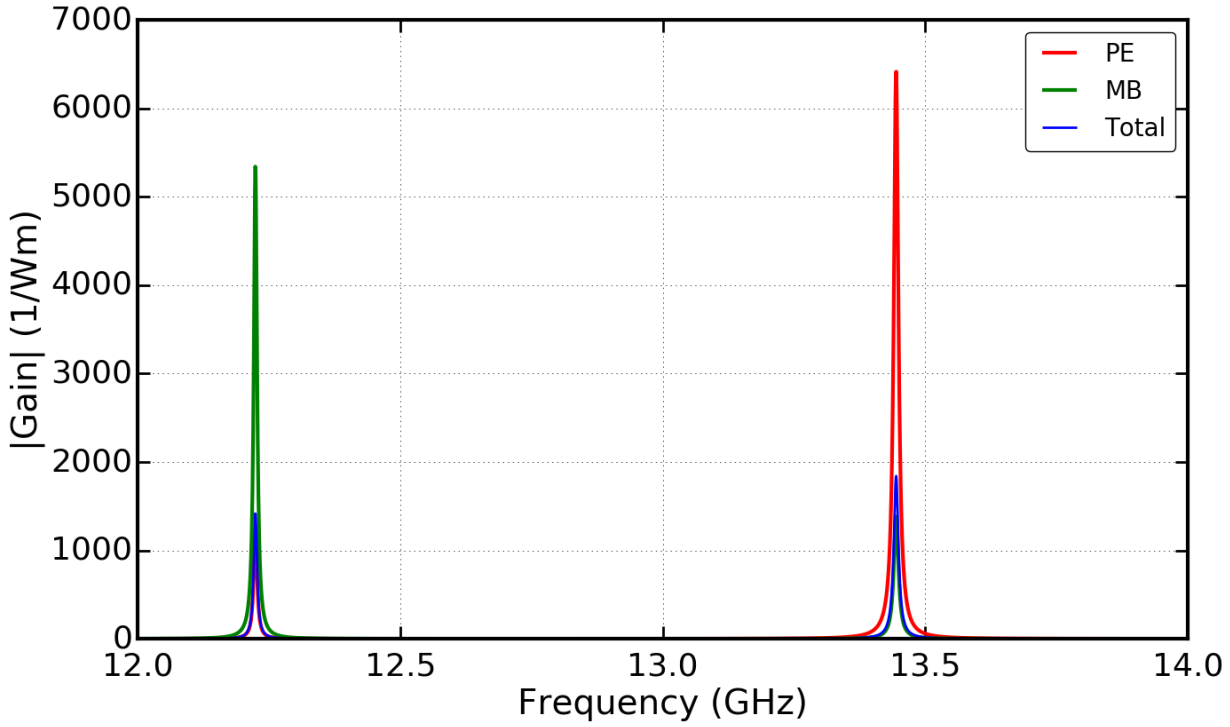

Fig. 3.5: Zoomed in gain spectra.

```
import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# 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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 20
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.Vacuum,
                        material_a=materials.Si_2016_Smith,
                        lc_bkg=3, lc2=2000.0, lc3=1000.0)

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

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

# 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_pump = npzfile['sim_EM_pump'].tolist()
npzfile = np.load('wguide_data2.npz')
sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

# Will scan from forward to backward SBS so need to know k_AC of backward SBS.
k_AC = np.real(sim_EM_pump.Eig_values[0] - sim_EM_Stokes.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.calc_AC_modes(num_modes_AC, q_ac, EM_sim=sim_EM_pump)
    prop_AC_modes = np.array([np.real(x) for x in sim_AC.Eig_values if abs(np.
↪real(x)) > abs(np.imag(x))])
    sym_list = integration.symmetries(sim_AC)

    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('tut_03_1-dispersion_npload_symmetrised.pdf', bbox_inches='tight')
plt.savefig('tut_03_1-dispersion_npload_symmetrised.png', bbox_inches='tight')
plt.close()

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

Investigating Dispersion and multiprocessing

```
""" Calculate a dispersion diagram of the acoustic modes
    from k_AC ~ 0 (forward SBS) to k_AC = 2*k_EM (backward SBS).
    Use python's (embarrassing parallel) multiprocessing package.
"""

import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
from multiprocessing import Pool

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
```

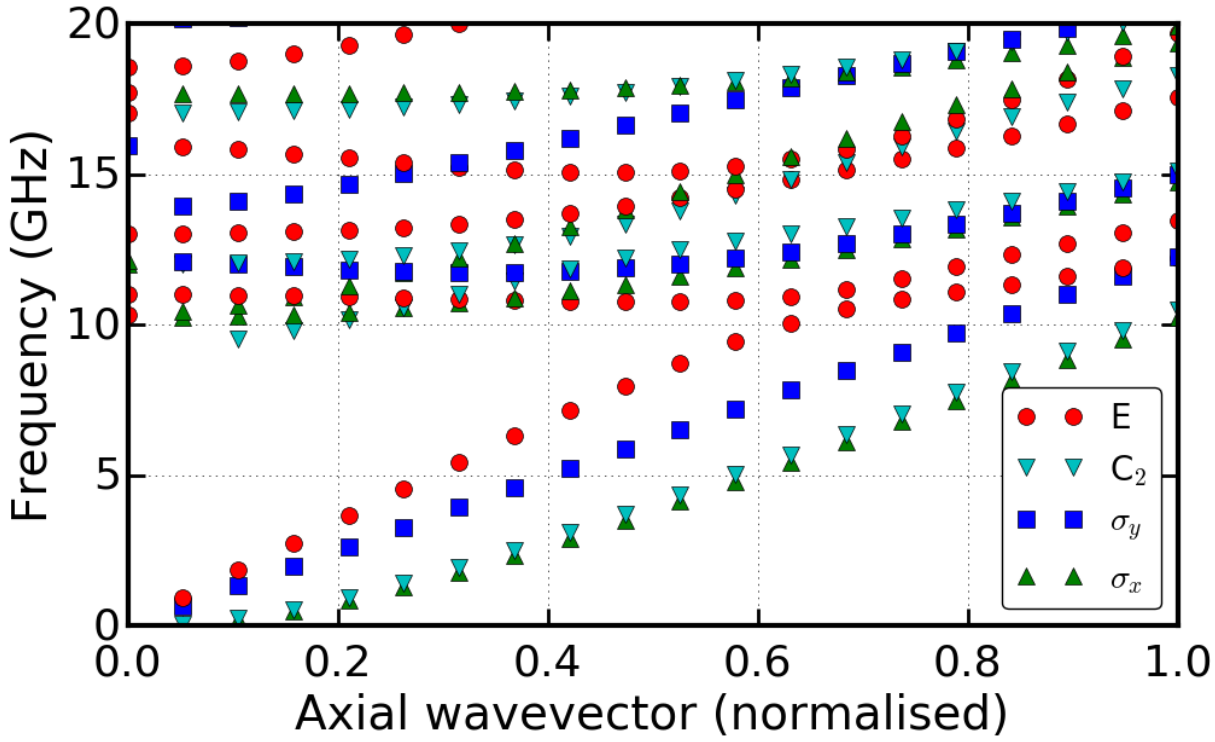

Fig. 3.6: Acoustic dispersion diagram with modes categorised by symmetry.

```
import integration
import plotting
from fortran import NumBAT


start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 3.0*wl_nm
unitcell_y = unitcell_x
inc_a_x = 800.
inc_a_y = 220.
inc_shape = 'rectangular'
# Choose modes to include.
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 60
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_03_2-'

# Note that this mesh is quite fine, may not be required if purely using dispersive␣
↪sims
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
```

```
                        material_bkg=materials.Vacuum,
                        material_a=materials.Si_2016_Smith,
                        lc_bkg=3, lc2=2000.0, lc3=1000.0)

# Estimated effective index of fundamental guided mode.
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic modes.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

# Will scan from forward to backward SBS so need to know k_AC of backward SBS.
k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

# Rather than calculating with a loop we can use pool to do a multi core sim
def ac_mode_freqs(k_ac):
    print('Commencing mode calculation for k_ac = %f'% k_ac)

    # Calculate the modes, grab the output frequencies only and convert to GHz
    sim_AC = wguide.calc_AC_modes(num_modes_AC, k_ac, EM_sim=sim_EM_pump)
    prop_AC_modes = np.array([np.real(x) for x in sim_AC.Eig_values if abs(np.
↪real(x)) > abs(np.imag(x))])
    mode_freqs = prop_AC_modes*1.e-9

    print('Completed mode calculation for width a_x = %f'% k_ac)

    # Return the frequencies and simulated k_ac value in a list
    return mode_freqs


# Now we utilise multi-core calculations to perform parallel simulations and speed up␣
↪the simulation
test_name = 'dispersion_multicore'
nu_ks = 5  # start with a low number of k_ac values to get an idea
acoustic_ks = np.linspace(5., k_AC*1.1, nu_ks)

num_cores = 5  # should be appropriate for individual machine/vm, and memory!
pool = Pool(num_cores)
pooled_mode_freqs = pool.map(ac_mode_freqs, acoustic_ks)

# We will pack the above values into a single array for plotting purposes, initialise␣
↪first
freq_arr = np.empty((nu_ks, num_modes_AC))
for i_w, sim_freqs in enumerate(pooled_mode_freqs):
    # Set the value to the values in the frequency array
    freq_arr[i_w] = sim_freqs

# Now that we have packed will save to a numpy file for better plotting and reference
file_name = 'freq_array_200'
np.save(file_name, freq_arr)
np.save(file_name+'_qs', acoustic_ks)  # and the q values

# Also plot a figure for reference
plot_range = num_modes_AC
plt.clf()
plt.figure(figsize=(10,6))
ax = plt.subplot(1,1,1)
```

```
for idx in range(plot_range):
    # slicing in the row direction for plotting purposes
    freq_slice = freq_arr[:, idx]
    plt.plot(acoustic_ks/k_AC, freq_slice, 'r')

# Set the limits and plot axis labels
ax.set_ylim(0,35)
ax.set_xlim(0,1.1)
plt.xlabel(r'Axial wavevector (normalised)')
plt.ylabel(r'Frequency (GHz)')
plt.savefig(prefix_str+test_name+'.pdf', bbox_inches='tight')
plt.savefig(prefix_str+test_name+'.png', bbox_inches='tight')
plt.close()

# Output the normalisation k value for reference
print("The 2kp is: %f" % k_AC)

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

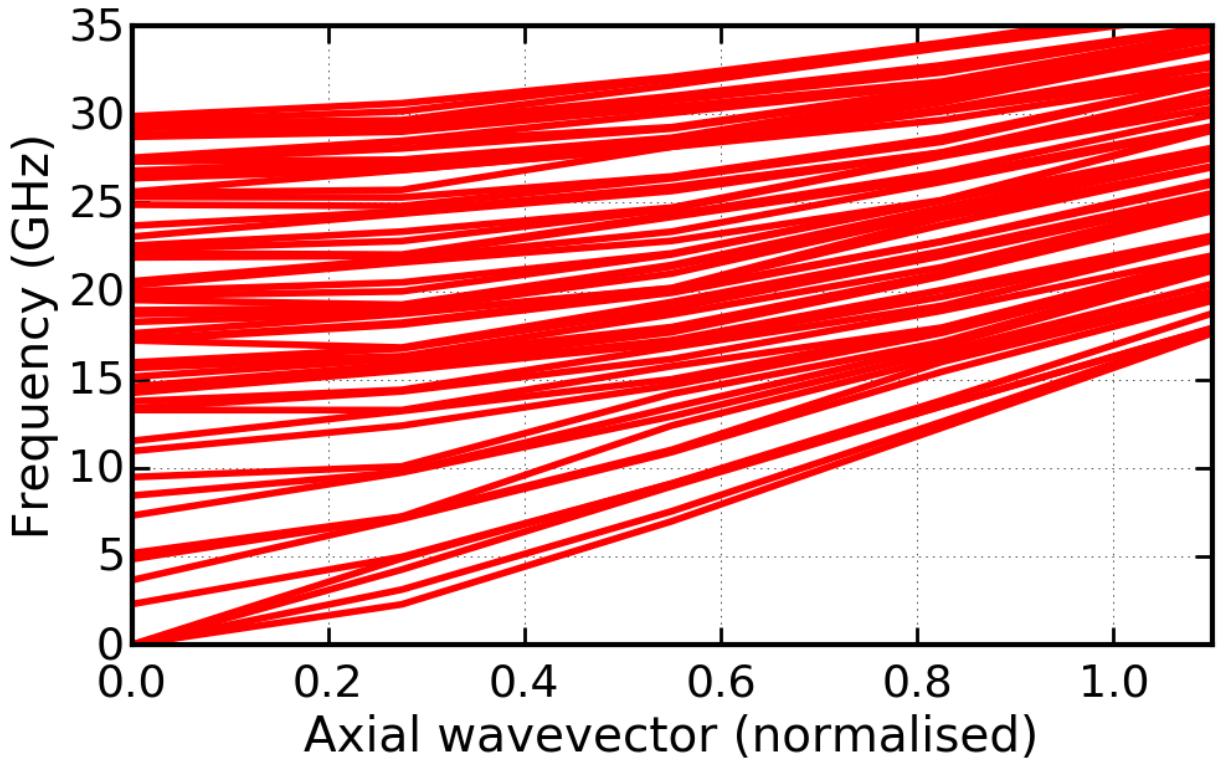

Fig. 3.7: Acoustic dispersion diagram ploted as lines.

Parameter Scan of Widths

```
""" 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
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

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

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

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

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 20
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_04-'

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

def modes_n_gain(wguide):
    print ('Commencing mode calculation for width a_x = %f' % wguide.inc_a_x)
    # Expected effective index of fundamental guided mode.
    n_eff = (wguide.material_a.n-0.1) * wguide.inc_a_x/known_geo
    # Calculate Electromagnetic modes.
    sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
    sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
    k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
    # Calculate Acoustic modes.
    sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
```

```
    # Calculate interaction integrals and SBS gain.
    SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.
↪gain_and_qs(
        sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
        EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)

    print ('Completed mode calculation for width a_x = %f' % wguide.inc_a_x)
    return [sim_EM_pump, sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_
↪AC]


nu_widths = 6
waveguide_widths = np.linspace(300,350,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,
                            material_bkg=materials.Vacuum,
                            material_a=materials.Si_2016_Smith,
                            lc_bkg=3, lc2=2000.0, lc3=1000.0)
    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
int_min = 10
int_max = 26
interp_grid = np.linspace(int_min, int_max, 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]
    linewidth_Hz = width_obj[5]
    k_AC = width_obj[6]
    # Calculate the EM effective index of the waveguide (k_AC = 2*k_EM).
    n_eff_sim = np.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 = np.real(sim_AC.Eig_values[0])*1e-9 - 5  # GHz
    freq_max = np.real(sim_AC.Eig_values[-1])*1e-9 + 5  # GHz
    plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_
↪AC,
        EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
        prefix_str=prefix_str, suffix_str='_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
    # Linewidth of Lorentzian is half the FWHM style linewidth.
    linewidth = linewidth_Hz/2
    for AC_i in range(len(linewidth_Hz)):
        gain_list = np.real(SBS_gain[EM_ival_Stokes,EM_ival_pump,AC_i]
                     *linewidth[AC_i]**2/(linewidth[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, abs(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(int_min,int_max)
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(prefix_str+'gain_spectra-waterfall.pdf')
plt.savefig(prefix_str+'gain_spectra-waterfall.png')
plt.close()

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

Convergence Study

```
""" Calculate the convergence as a function of FEM mesh for
    backward SBS gain spectra of a silicon waveguide
    surrounded in air.
"""

import time
```

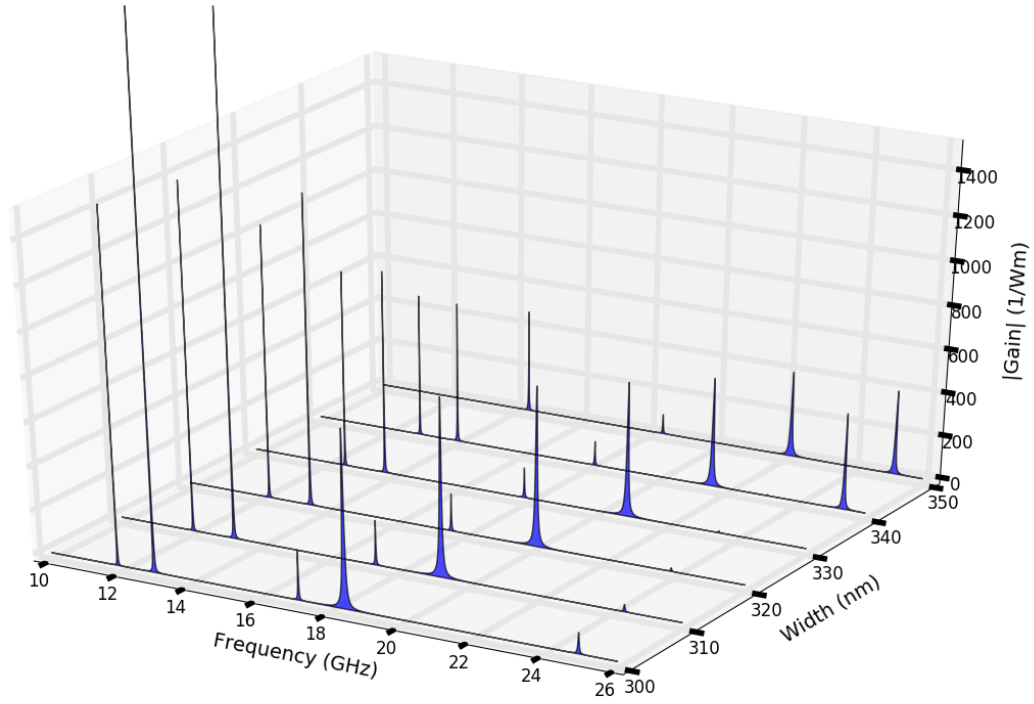

Fig. 3.8: Gain spectra as function of waveguide width.

```
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 20
EM_ival_pump = 0
```

```
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_05-'

nu_lcs = 4
lc_bkg_list = 4*np.ones(nu_lcs)
lc_list = np.linspace(5e2,5e3,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 = lc_ref
    lc_bkg = lc_bkg_list[i_lc]
    wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,
                            inc_a_y,inc_shape,
                            material_bkg=materials.Vacuum,
                            material_a=materials.Si_2016_Smith,
                            lc_bkg=lc_bkg, lc2=lc_ref, lc3=lc3, force_mesh=True)

    # Expected effective index of fundamental guided mode.
    n_eff = wguide.material_a.n-0.1
    # Calculate Electromagnetic modes.
    sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
    sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
    k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
    # Calculate Acoustic modes.
    sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
    # Calculate interaction integrals and SBS gain.
    SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.
↪gain_and_qs(
        sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
        EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)

    conv_list.append([sim_EM_pump, sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB])
    end = time.time()
    time_list.append(end - start)

# It is crucial that you preselect modes with significant gain!
# Otherwise you will observe large relative errors similar to dividing by zero.
rel_modes = [2,4,8]
# If you do not know the mode numbers of the significant AC modes you may wish to␣
↪simply plot them all
# by uncommenting the line below and check if the modes with large gain have low␣
↪relative errors.
# rel_modes = np.linspace(0,num_modes_AC-1,num_modes_AC)
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_ival_Stokes,EM_ival_pump,rel_mode]
        rel_mode_gain_PE[i_conv,i_m] = conv_obj[3][EM_ival_Stokes,EM_ival_pump,rel_
↪mode]
        rel_mode_gain_MB[i_conv,i_m] = conv_obj[4][EM_ival_Stokes,EM_ival_pump,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_ival_pump)
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(prefix_str+'convergence-freq_EM.pdf', bbox_inches='tight')
plt.savefig(prefix_str+'convergence-freq_EM.png', 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(prefix_str+'convergence-freq_AC.pdf', bbox_inches='tight')
plt.savefig(prefix_str+'convergence-freq_AC.png', 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(prefix_str+'convergence-Gain.pdf', bbox_inches='tight')
plt.savefig(prefix_str+'convergence-Gain.png', 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 (PE)")
ax2.set_ylabel(r"Relative Error Gain (PE)")
ax2.set_yscale('log', nonposx='clip')
```

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

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

Silica Nanowire

```
""" 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
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
```

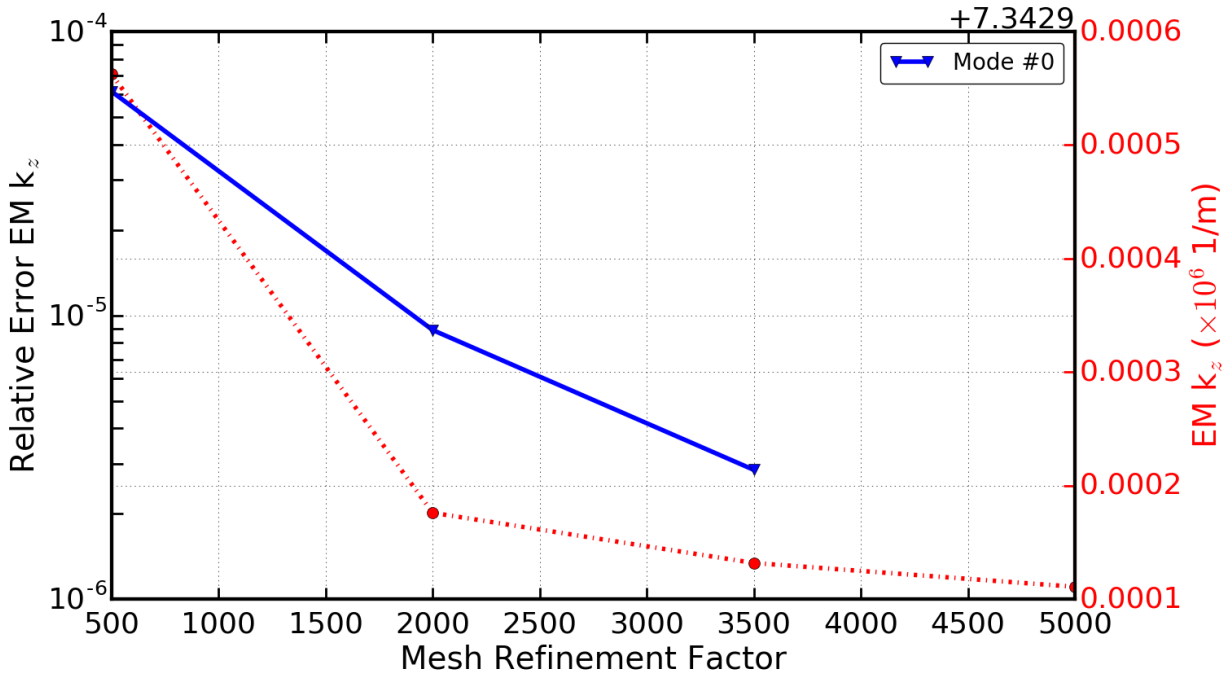

Fig. 3.9: Convergence of optical mode frequencies.

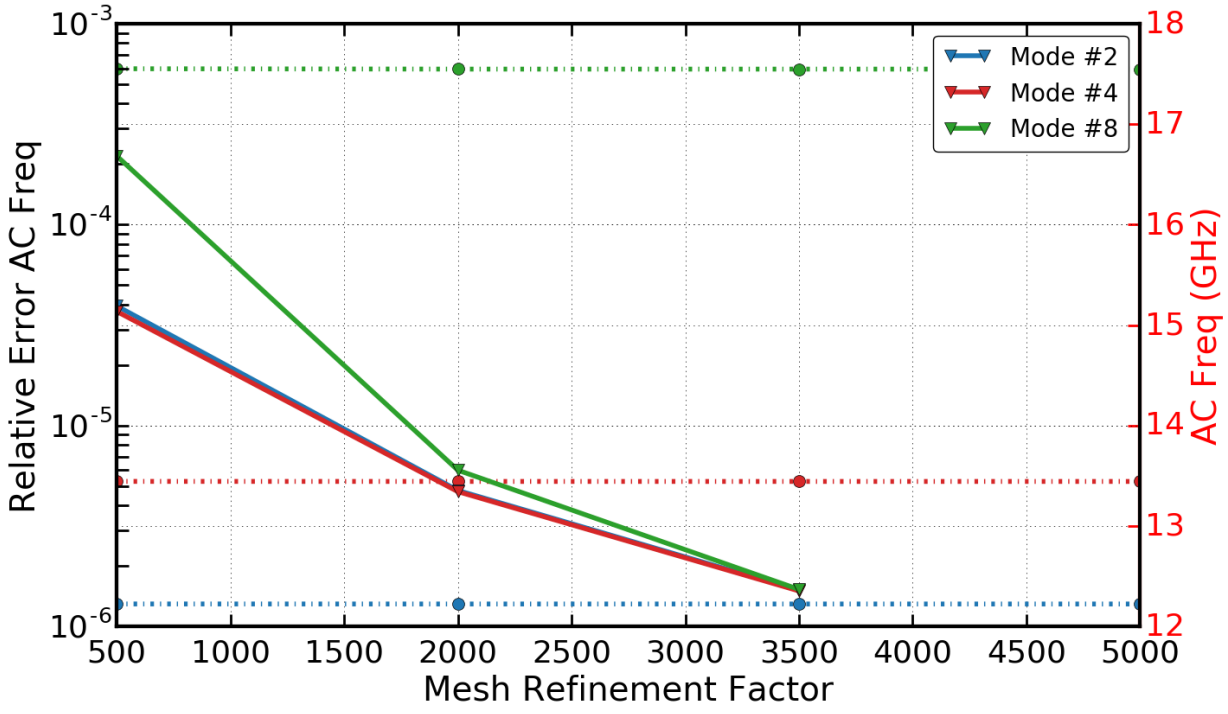

Fig. 3.10: Convergence of acoustic mode frequencies.

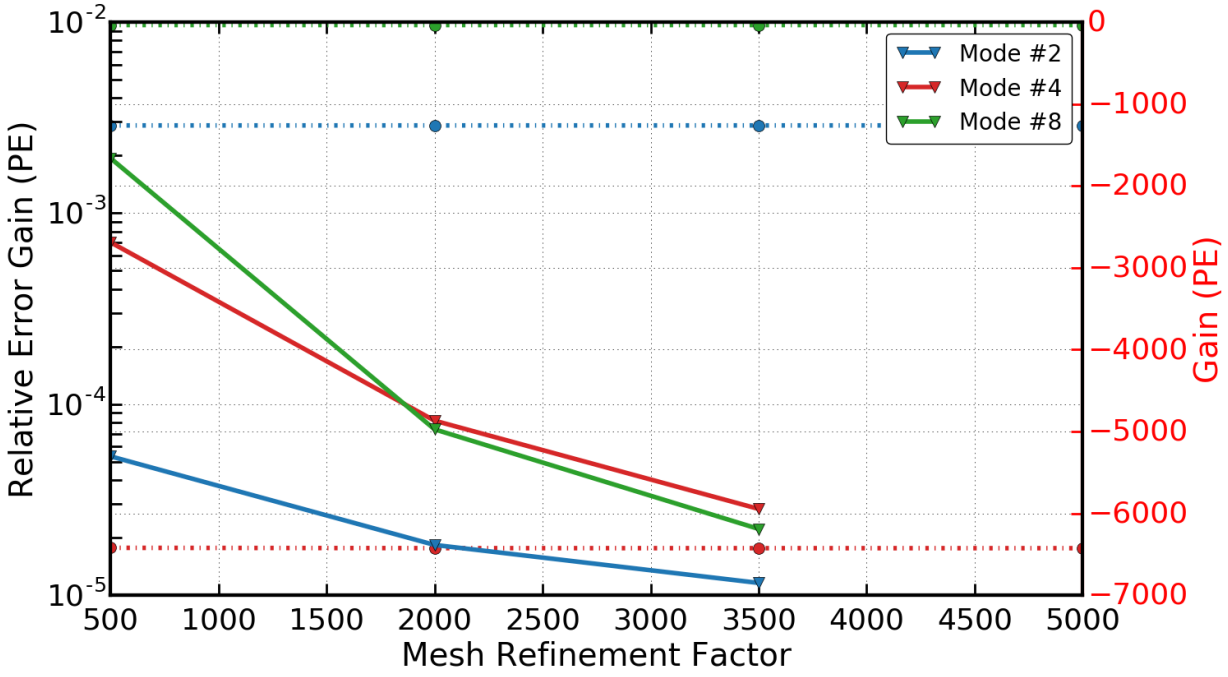

Fig. 3.11: Convergence of photoelastic gain.

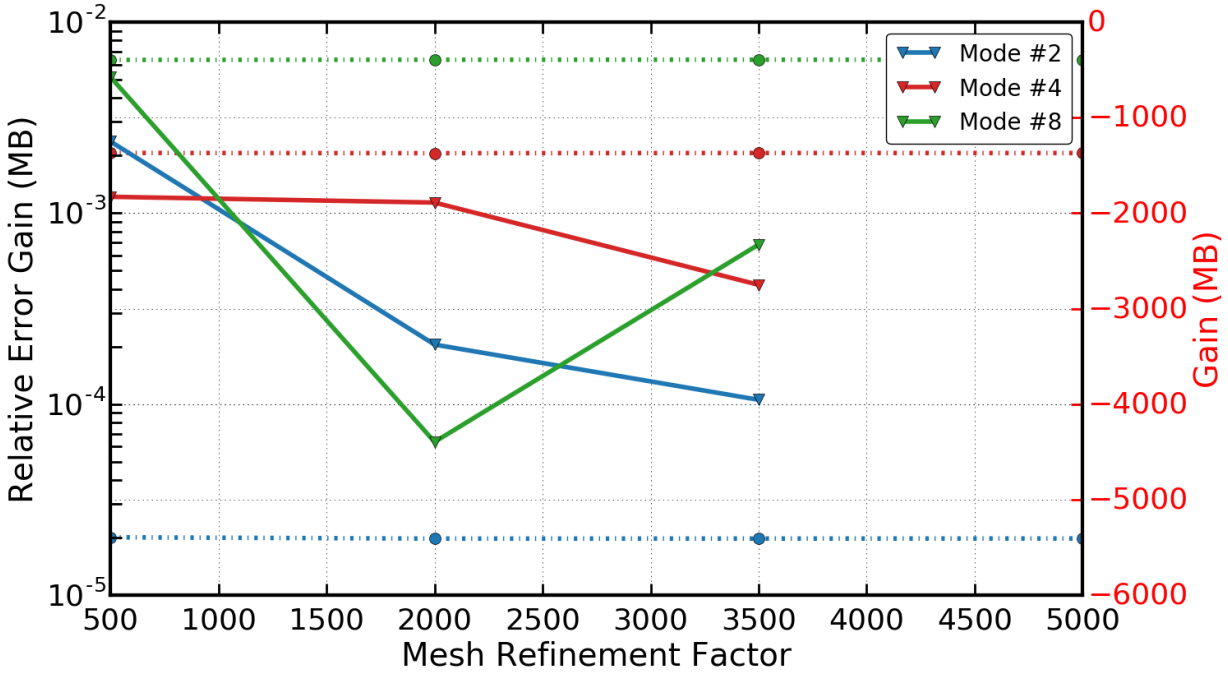

Fig. 3.12: Convergence of moving boundary gain.

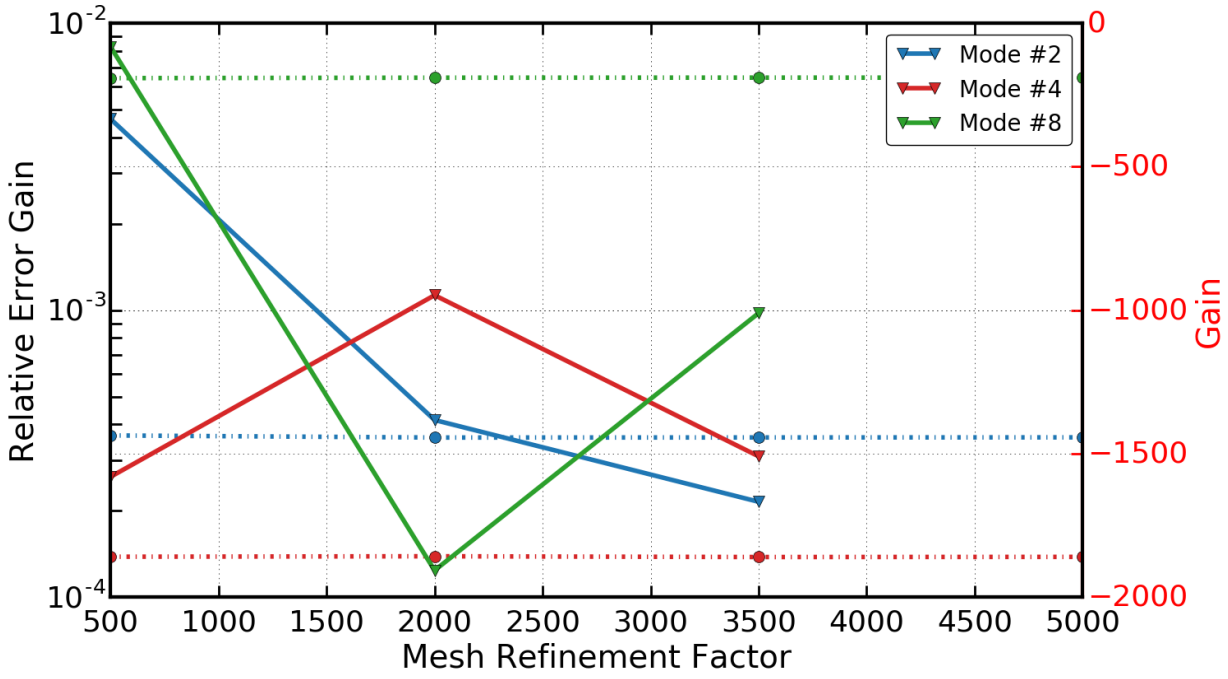

Fig. 3.13: Convergence of total gain.

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


start = time.time()

# 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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 40
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_06-'

wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.Vacuum,
                        material_a=materials.SiO2_2016_Smith,
                        lc_bkg=3, lc2=2000.0, lc3=1000.0)
```

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

# Calculate Electromagnetic Modes
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
# np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz')
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()
# plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4,
#                          ylim_min=0.4, ylim_max=0.4, EM_AC='EM_E',
#                          prefix_str=prefix_str, suffix_str='NW')
# plotting.plt_mode_fields(sim_EM_pump, EM_AC='EM_E', prefix_str=prefix_str, suffix_
↪str='NW')

sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
# np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
# npzfile = np.load('wguide_data2.npz')
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

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

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

shift_Hz = 4e9

# Calculate Acoustic modes.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)
# np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC = npzfile['sim_AC'].tolist()
# plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, suffix_str='NW')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, 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 = 12   # GHz

plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str, suffix_str='_SiO2_NW')

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

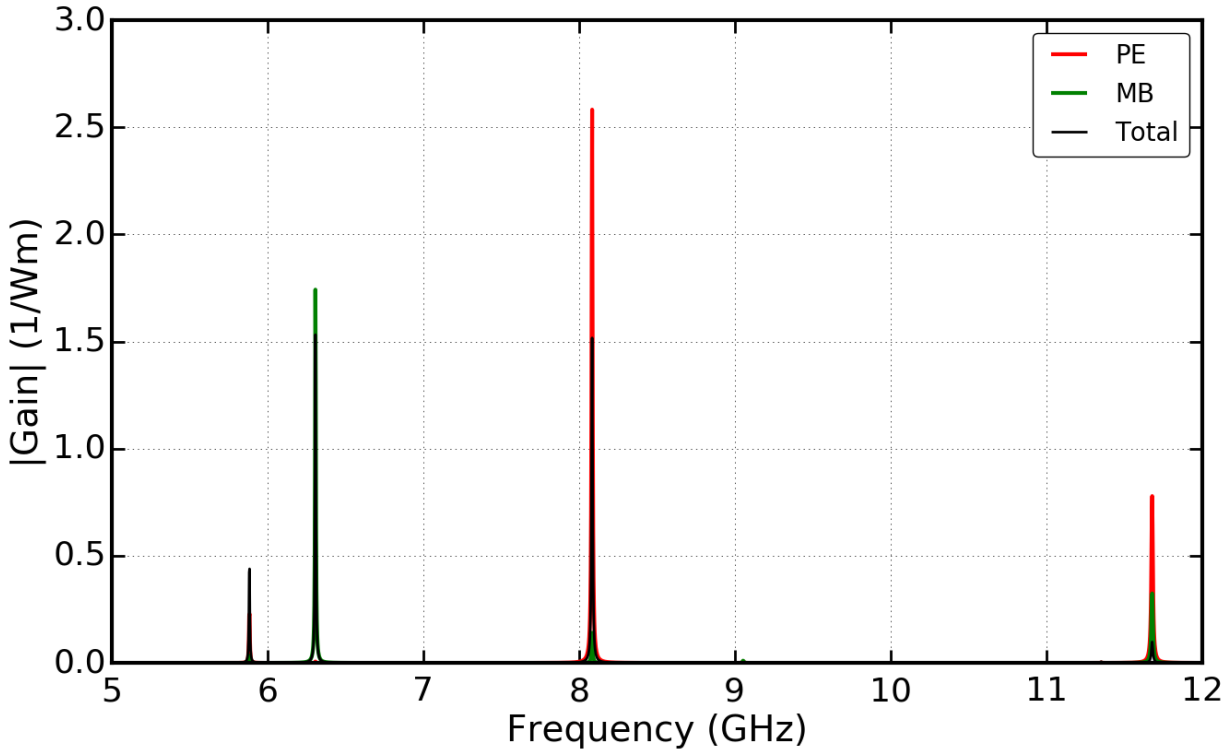

Fig. 3.14: Gain spectra showing gain due to photoelastic effec, gain due to moving boundary effect, and total gain.

Slot Waveguide

```
""" Calculate the backward SBS gain spectra of a Si
    slot waveguide containing As2S3 on a SiO2 slab.
"""

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

```
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = 0.3*unitcell_x
inc_shape = 'slot'
inc_a_x = 150
inc_a_y = 190
inc_b_x = 250
# Current mesh template assume inc_b_y = inc_a_y
slab_a_x = 2000
slab_a_y = 100

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 40
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_07-'

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, inc_b_x=inc_b_x,
                        material_bkg=materials.Vacuum,             # background
                        material_a=materials.As2S3_2017_Morrison, # slot
                        material_b=materials.SiO2_2013_Laude,     # slab
                        material_c=materials.Si_2016_Smith,       # walls of slot
                        lc_bkg=2, lc2=2000.0, lc3=1000.0)

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

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

sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
# np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
# npzfile = np.load('wguide_data2.npz')
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()
```

```
# plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4,
#                          ylim_min=0.1, ylim_max=0.8, EM_AC='EM_E',
#                          prefix_str=prefix_str, suffix_str='slot')

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

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

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

# Calculate Acoustic modes.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)
# np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC = npzfile['sim_AC'].tolist()

# plotting.plt_mode_fields(sim_AC, xlim_min=0.4, xlim_max=0.4,
#                          ylim_min=0.7, ylim_max=0.0, EM_AC='AC',
#                          prefix_str=prefix_str, suffix_str='slot')

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

set_q_factor = 1000.

SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, 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 = np.real(sim_AC.Eig_values[0])*1e-9 - 2  # GHz
freq_max = np.real(sim_AC.Eig_values[-1])*1e-9 + 2  # GHz

plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str, suffix_str='_slot')

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

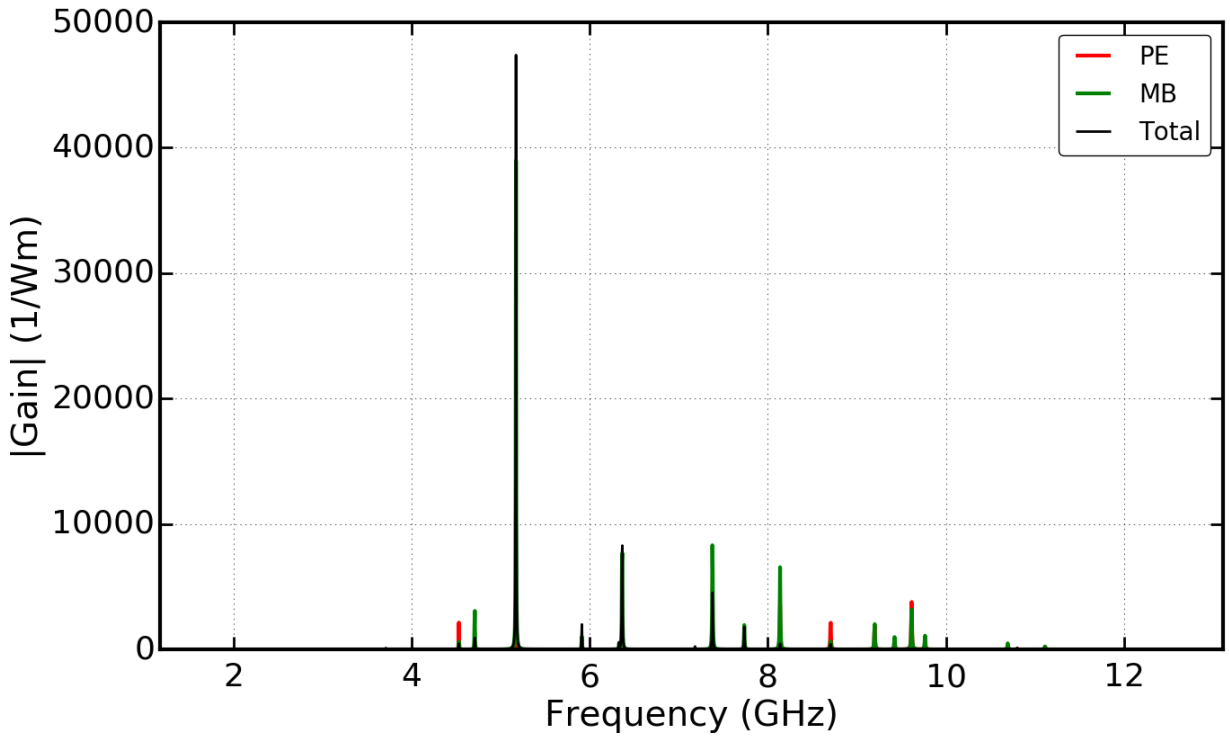

Fig. 3.15: Gain spectra showing gain due to photoelastic effec, gain due to moving boundary effect, and total gain.

Slot Waveguide Scan Covering

```
""" Calculate the backward SBS gain spectra of a Si
    slot waveguide containing As2S3 on a SiO2 slab.

    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
from multiprocessing import Pool
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT

start = time.time()
```

```
# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = 0.3*unitcell_x
inc_shape = 'slot_coated'
inc_a_x = 150
inc_a_y = 190
inc_b_x = 250
# Current mesh template assume inc_b_y = inc_a_y
slab_a_x = 1000
slab_a_y = 100

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 40
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'tut_08-'

# Function to return ac freqs for given coating thickness
def ac_mode_freqs(coat_y):
    print('Commencing mode calculation for coat_y = %f'% coat_y)

    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, inc_b_x=inc_b_x,
                            coat_y=coat_y,
                            material_bkg=materials.Vacuum,            # background
                            material_a=materials.As2S3_2017_Morrison, # slot
                            material_b=materials.SiO2_2013_Laude,     # slab
                            material_c=materials.Si_2016_Smith,       # walls of slot
                            material_d=materials.SiO2_2013_Laude,     # coating
                            lc_bkg=5, lc2=2000.0, lc3=1000.0)

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

    # Calculate Electromagnetic modes.
    sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
    sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

    k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

    shift_Hz = 4e9

    # Calculate Acoustic modes.
    sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_
↪Hz=shift_Hz)

    # plotting.plt_mode_fields(sim_AC, xlim_min=0.4, xlim_max=0.4,
    #                          ylim_min=0.7, ylim_max=0.0, EM_AC='AC',
    #                          prefix_str=prefix_str, suffix_str='_%i' %int(coat_y))

    set_q_factor = 1000.
```

```
    SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.
↪gain_and_qs(
        sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
        EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival,␣
↪fixed_Q=set_q_factor)

    # Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
    freq_min = 4 # np.real(sim_AC.Eig_values[0])*1e-9 - 2  # GHz
    freq_max = 14 # np.real(sim_AC.Eig_values[-1])*1e-9 + 2  # GHz

    plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_
↪AC,
        EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
        prefix_str=prefix_str, suffix_str='_%i' %int(coat_y))

    # Convert to GHz
    mode_freqs = sim_AC.Eig_values*1.e-9

    print('Completed mode calculation for coating coat_y = %f'% coat_y)

    # Return the frequencies and simulated k_ac value in a list
    return mode_freqs


nu_coats = 5
coat_min = 5
coat_max = 200
coat_y_list = np.linspace(coat_min,coat_max,nu_coats)

num_cores = 5  # should be appropriate for individual machine/vm, and memory!
pool = Pool(num_cores)
pooled_mode_freqs = pool.map(ac_mode_freqs, coat_y_list)

# We will pack the above values into a single array for plotting purposes, initialise␣
↪first
freq_arr = np.empty((nu_coats, num_modes_AC))
for i_w, sim_freqs in enumerate(pooled_mode_freqs):
    # Set the value to the values in the frequency array
    freq_arr[i_w] = sim_freqs

# Also plot a figure for reference
plot_range = num_modes_AC
plt.clf()
plt.figure(figsize=(10,6))
ax = plt.subplot(1,1,1)
for idx in range(plot_range):
    # slicing in the row direction for plotting purposes
    freq_slice = freq_arr[:, idx]
    plt.plot(coat_y_list, freq_slice, 'g')

# Set the limits and plot axis labels
ax.set_xlim(coat_min,coat_max)
plt.xlabel(r'Coating Thickness (nm)')
plt.ylabel(r'Frequency (GHz)')
plt.savefig(prefix_str+'freq_changes.pdf', bbox_inches='tight')
plt.savefig(prefix_str+'freq_changes.png', bbox_inches='tight')
plt.close()
```

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

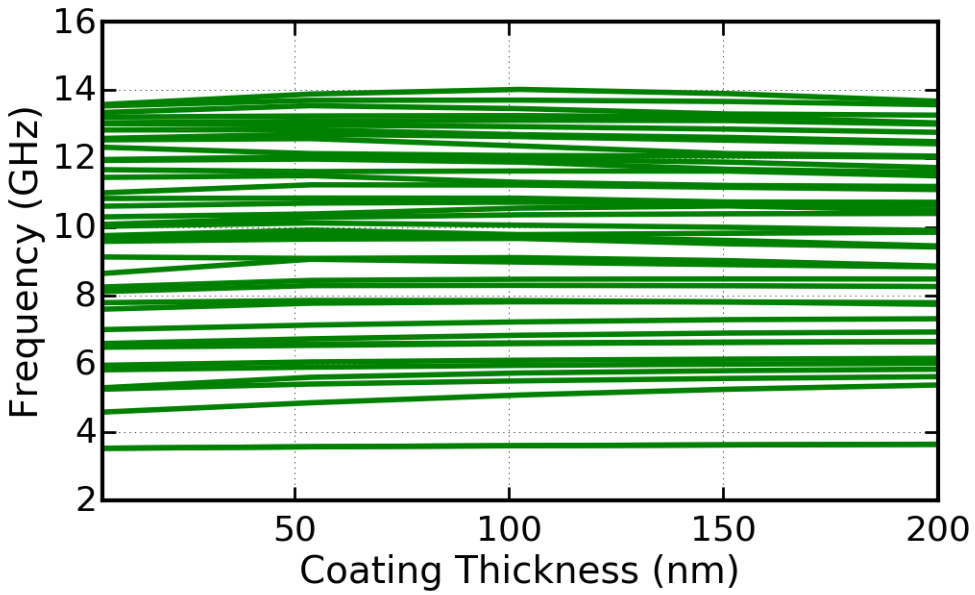

Fig. 3.16: Acoustic frequencies as function of covering layer thickness.

Anisotropic Elastic Materials

```
""" Sanity check implementation of fully anisotropic
    tensors by feeding in same parameters of simo_tut_01.
"""

import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT

start = time.time()

# 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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 20
EM_ival_pump = 0
EM_ival_Stokes = 0
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,
                        material_bkg=materials.Vacuum,
                        material_a=materials.Si_test_anisotropic,
                        lc_bkg=2, lc2=200.0, lc3=5.0)


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

# Calculate the Electromagnetic modes of the pump field.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
# Print the wavevectors of EM modes.
print('\n k_z of EM modes \n', np.round(np.real(sim_EM_pump.Eig_values),4))
# Calculate the Electromagnetic modes of the Stokes field.
# For an idealised backward SBS simulation the Stokes modes are identical
# to the pump modes but travel in the opposite direction.
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
# # Alt
# sim_EM_Stokes = wguide.calc_EM_modes(wl_nm, num_modes_EM_Stokes, n_eff, Stokes=True)

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_pump.Eig_values[0]*((wl_nm*1e-9)/(2.*np.pi)))
print("\n Fundamental optical mode ")
print(" n_eff = ", np.round(n_eff_sim, 4))
# Acoustic wavevector
k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
print('\n AC wavenumber (1/m) = ', np.round(k_AC, 4))

# Calculate Acoustic modes, using the mesh from the EM calculation.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
# Print the frequencies of AC modes.
print('\n Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)
# Print the Backward SBS gain of the AC modes.
print("\n SBS_gain PE contribution \n", SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:])
print("SBS_gain MB contribution \n", SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:])
```

```
print("SBS_gain total \n", SBS_gain[EM_ival_pump,EM_ival_Stokes,:])
# Mask negligible gain values to improve clarity of print out.
threshold = -1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival_pump,EM_ival_Stokes,:], 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))
```

Literature Examples

Having gotten familiar with NumBAT, we now set out to replicate a number of examples from the literature. The examples are presented in chronological order. We note the particular importance of examples 5-8 for they include experimental and numerical results that are in good agreement.

2013 - Laude - AIP Adv - BSBS - Rectangular Waveguide - Silica

```
""" 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

    Replicating silica example.
"""

import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# 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_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 120
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'lit_01-'

# Use all specified parameters to create a waveguide object.
```

```
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.Vacuum,
                        material_a=materials.SiO2_2013_Laude,
                        lc_bkg=4, lc2=1000.0, lc3=100.0)

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

# Calculate Electromagnetic modes.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

# plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ivals=[0],
#                          ylim_min=0.4, ylim_max=0.4, EM_AC='EM_E',
#                          prefix_str=prefix_str, pdf_png='png')

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

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

shift_Hz = 8e9

# Calculate Acoustic modes.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)

# plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, pdf_png='png')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, 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_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    semilogy=True, prefix_str=prefix_str)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 9.5  # GHz
freq_max = 10  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
```

```
    prefix_str=prefix_str, suffix_str='_zoom')

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

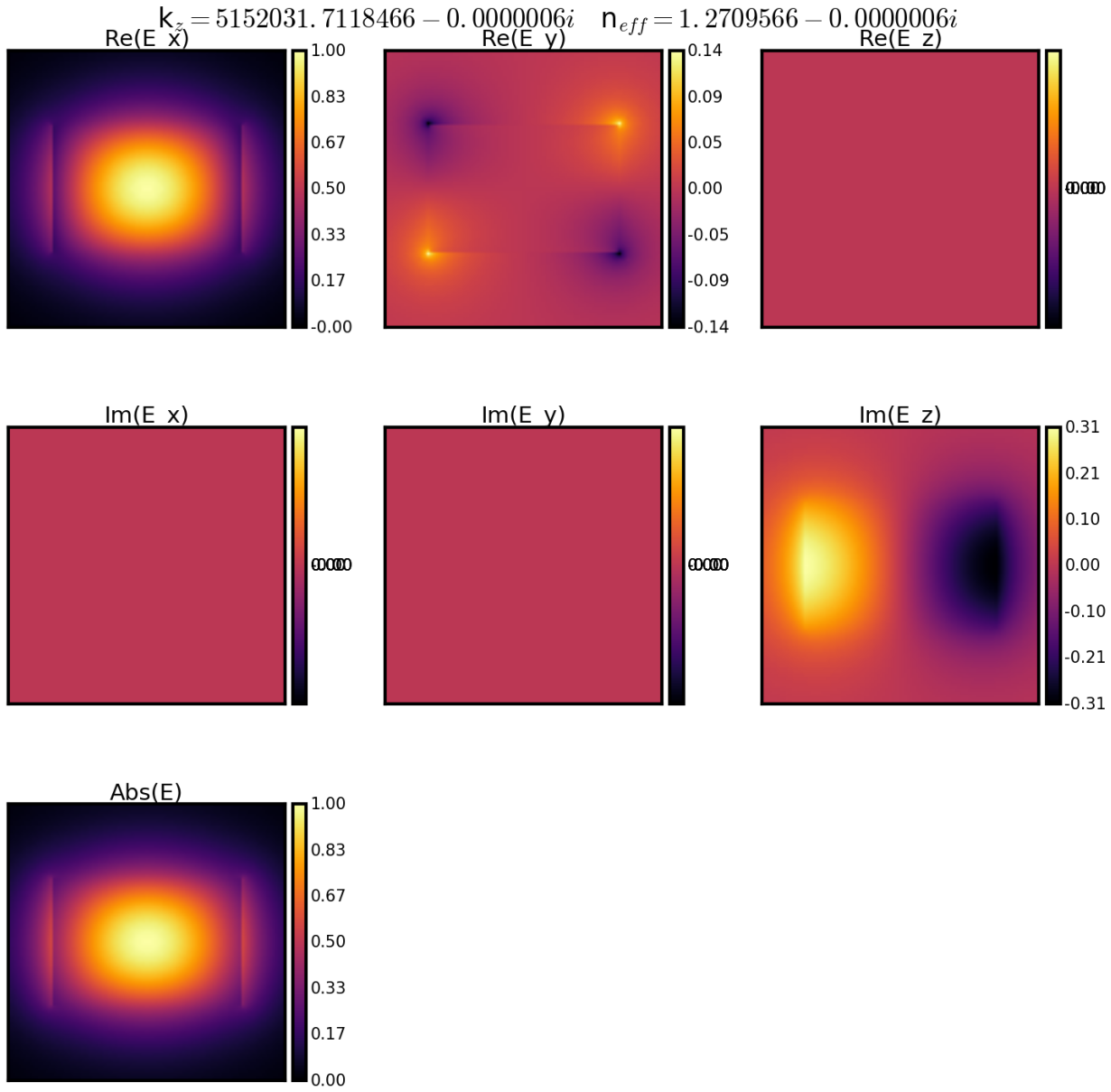

Fig. 3.17: Fundamental optical mode fields.

2013 - Laude - AIP Adv - BSBS - Rectangular Waveguide - Silicon

```
""" Replicating the results of
```

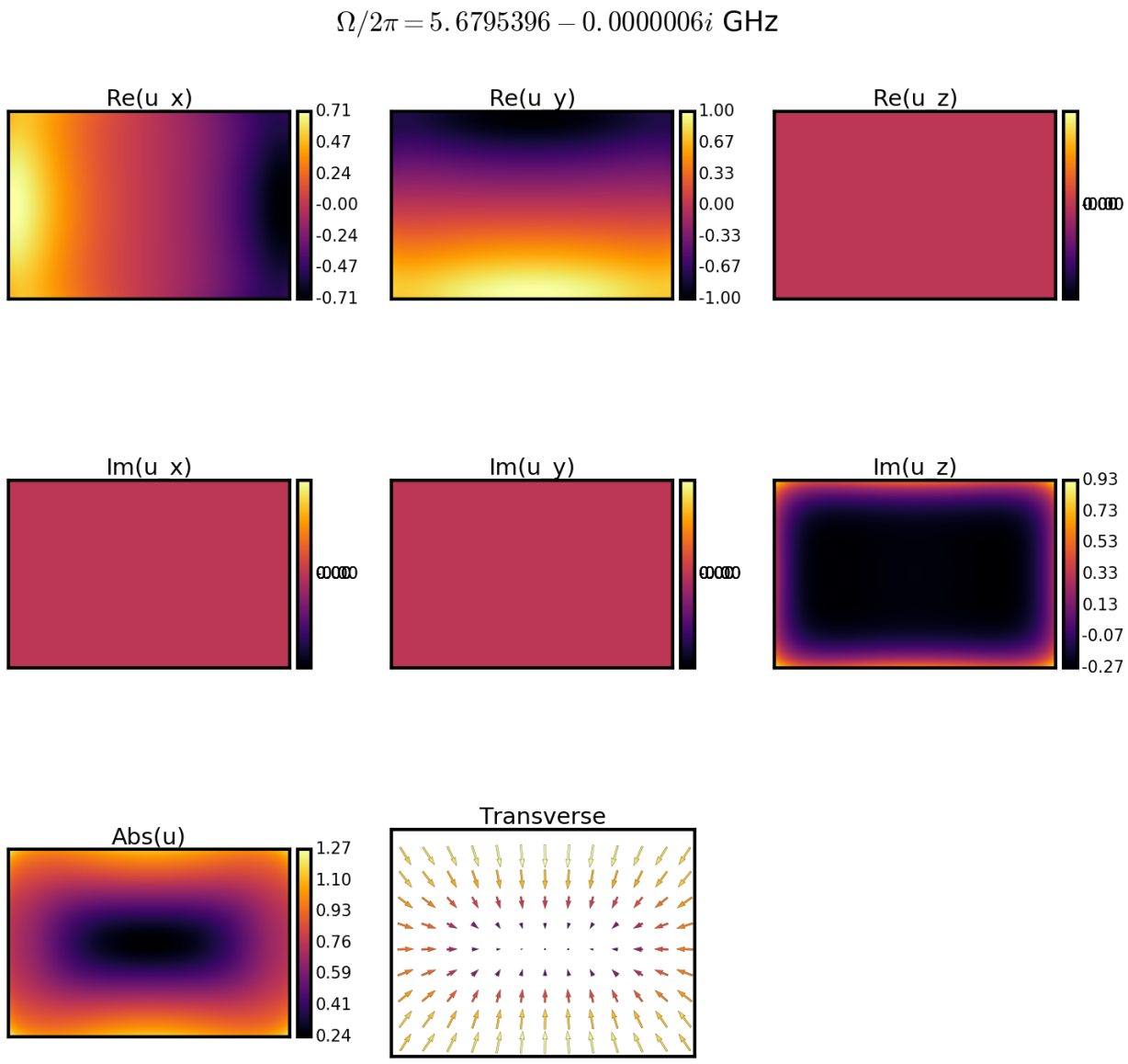

Fig. 3.18: High gain acoustic mode, marked as C in paper.

$$\Omega/2\pi = 9.7525866 - 0.0000006i \text{ GHz}$$

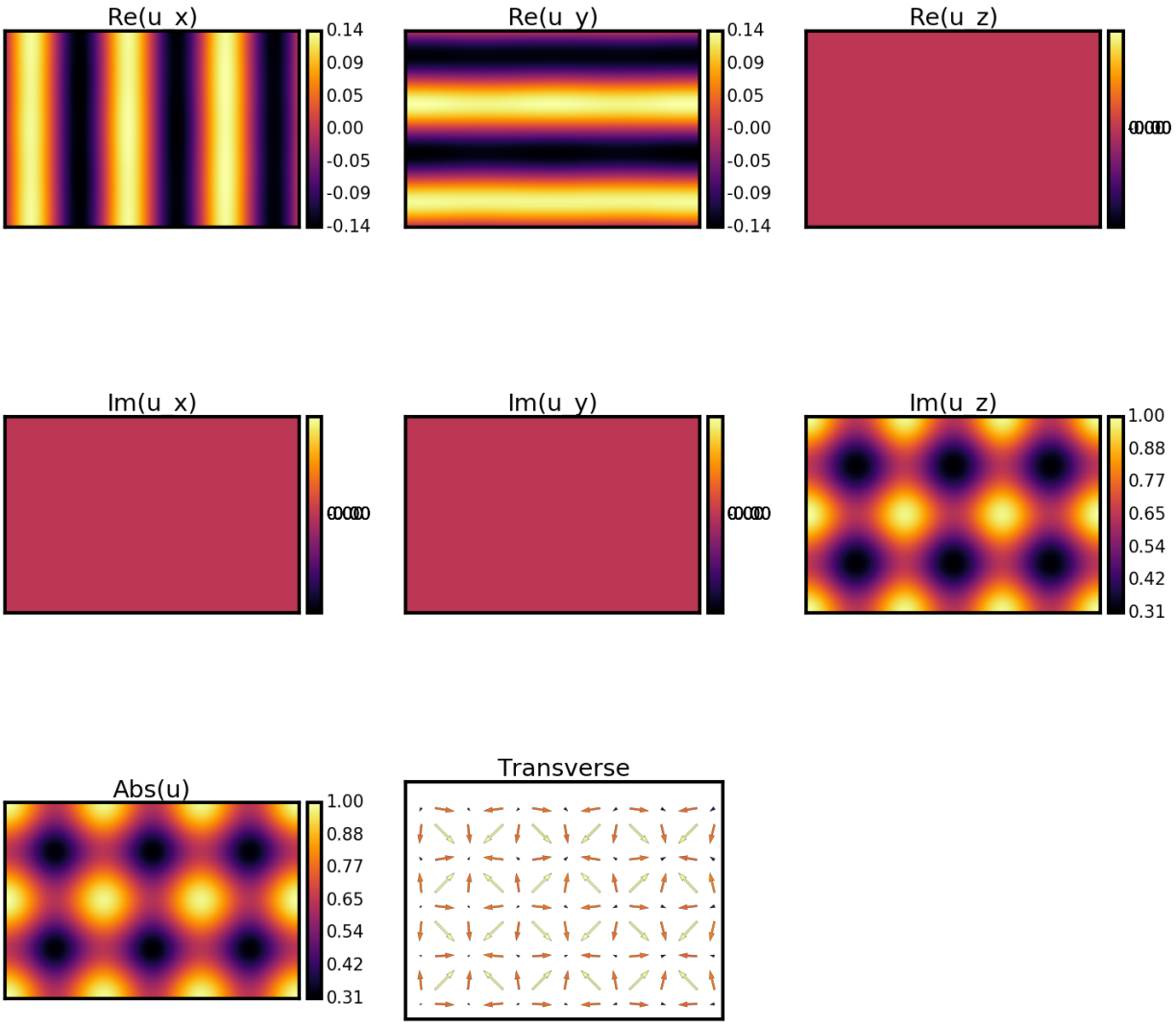

Fig. 3.19: High gain acoustic mode, marked as D in paper.

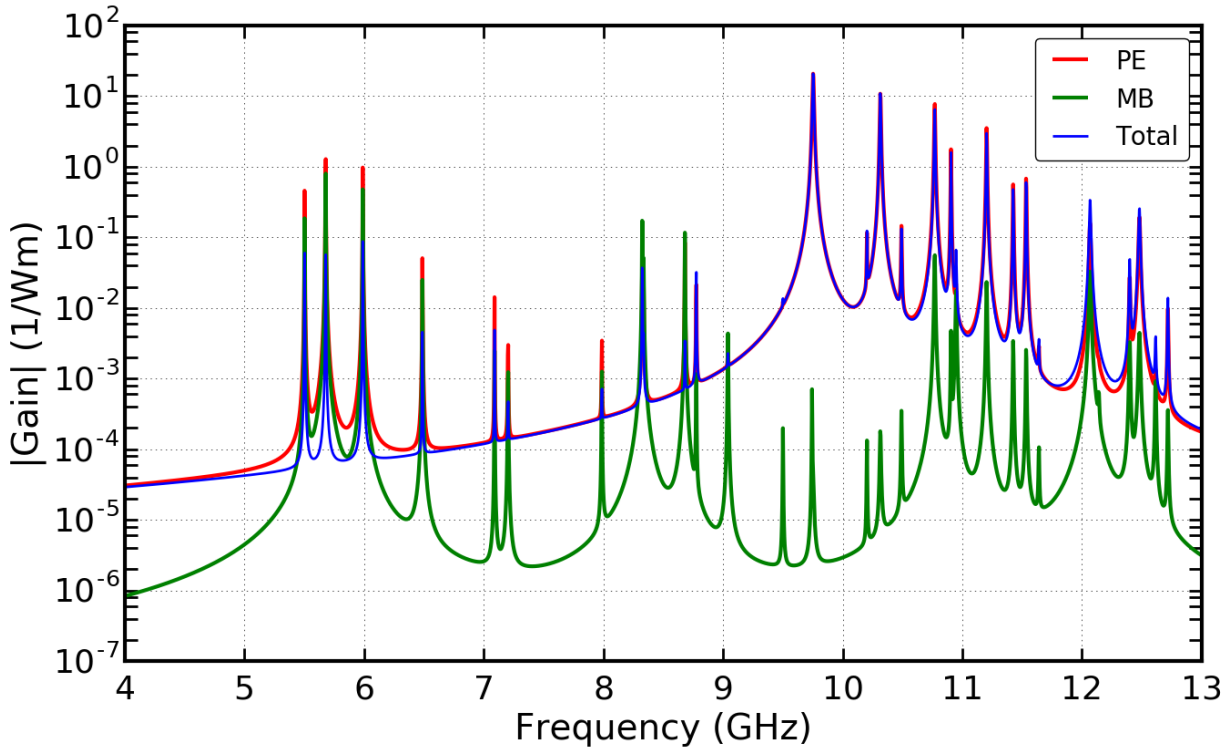

Fig. 3.20: Gain spectra on semilogy axis.

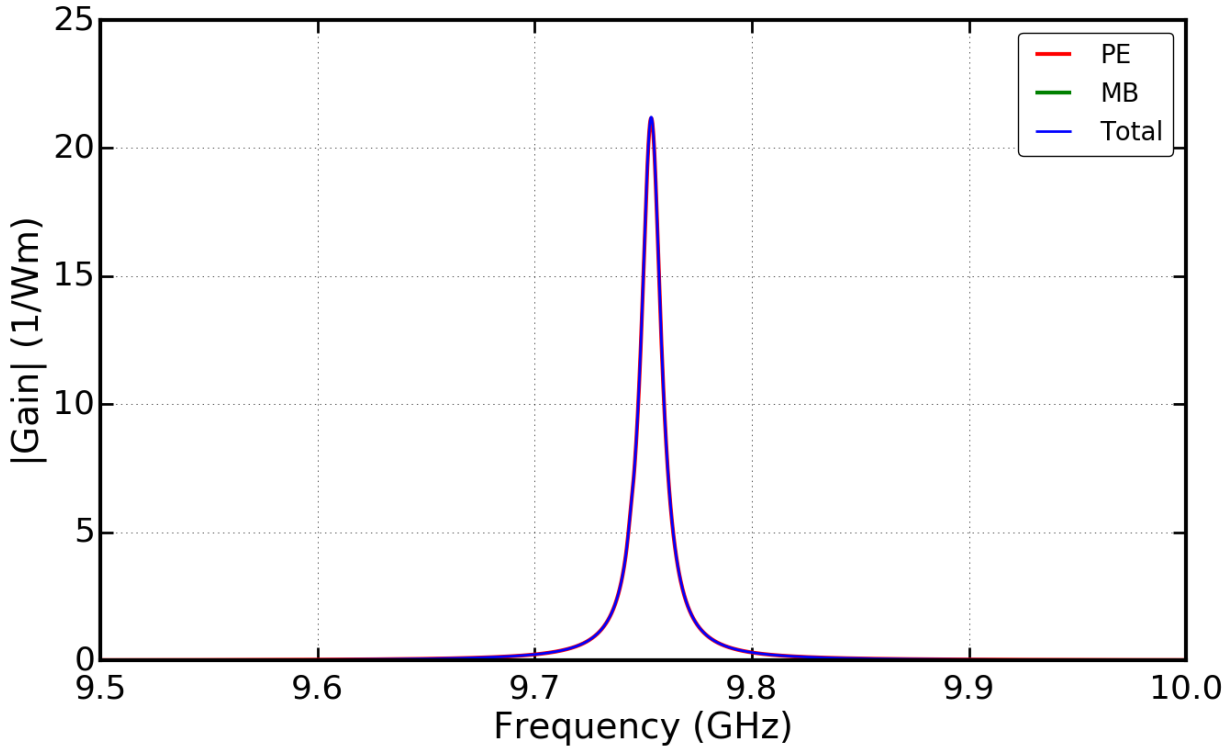

Fig. 3.21: Gain spectra zoomed in on mode D.

```
    Generation of phonons from electrostriction in
    small-core optical waveguides
    Laude et al.
    http://dx.doi.org/10.1063/1.4801936

    Replicating silicon example.
    Note requirement for lots of modes and therefore lots of memory.
"""

import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = unitcell_x*2/3
inc_a_x = 1500
inc_a_y = 1000
inc_shape = 'rectangular'

# Optical Parameters
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 800
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'lit_02-'

# Use all specified parameters to create a waveguide object.
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.Vacuum,
                        material_a=materials.Si_2013_Laude,
                        lc_bkg=5, lc2=1000.0, lc3=50.0)

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

# Calculate Electromagnetic modes.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
```

```
# plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.2, xlim_max=0.2, ivals=[0],
#                          ylim_min=0.2, ylim_max=0.2, EM_AC='EM_E',
#                          prefix_str=prefix_str, pdf_png='png')

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

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

shift_Hz = 31e9

# Calculate Acoustic modes.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)

# plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, pdf_png='png')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, 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_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    semilogy=True, prefix_str=prefix_str)

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

2014 - Beugnot - Nat Comm - BSBS - Tapered Fibre - Scanning Widths

```
""" 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
```

$$\Omega/2\pi = 21.3627746 - 0.0000006i \text{ GHz}$$

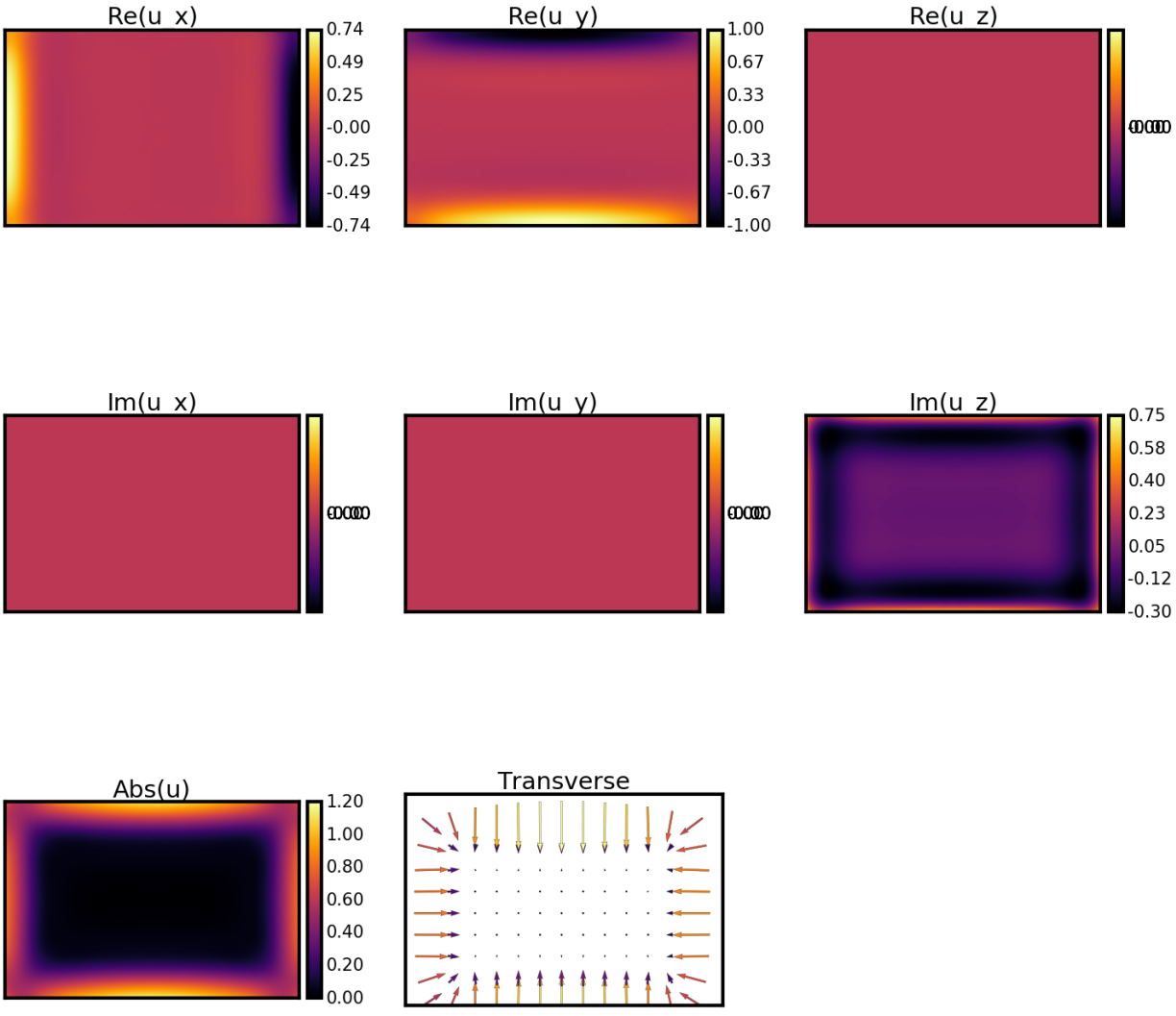

Fig. 3.22: High gain acoustic mode, marked as G in paper.

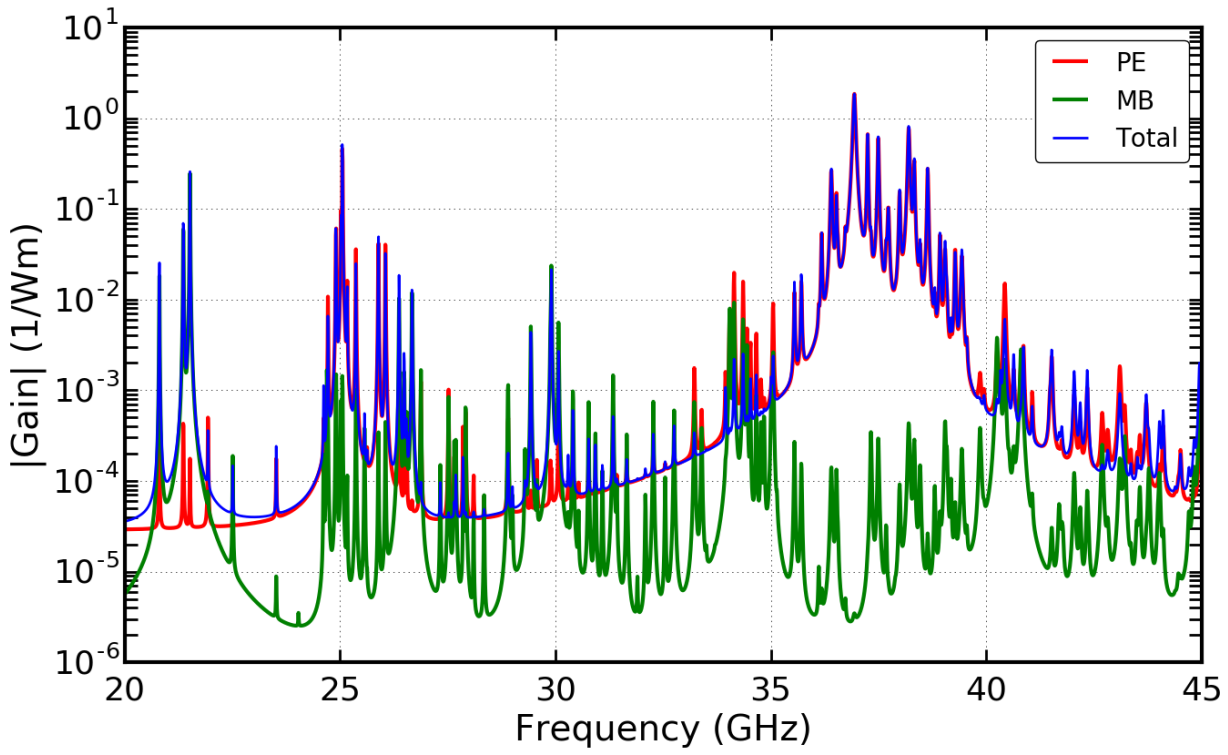

Fig. 3.23: Gain spectra on semilogy axis.

```
import datetime
import numpy as np
import sys
from multiprocessing import Pool
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

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

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = unitcell_x
inc_shape = 'circular'
```

```
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 80
EM_ival_pump = 0
EM_ival_Stokes = 0
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 = 1200
num_widths = 301
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,
                            material_bkg=materials.Vacuum,
                            material_a=materials.SiO2_2016_Smith,
                            lc_bkg=4, lc2=1000.0, lc3=100.0)

    sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
    sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
    k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
    shift_Hz = 4e9
    sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_
↪Hz=shift_Hz)

    set_q_factor = 600.
    SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.
↪gain_and_qs(
        sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
        EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)#,␣
↪fixed_Q=set_q_factor)

    interp_values = plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB,␣
↪linewidth_Hz, k_AC,
        EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min, freq_max, num_interp_pts=num_
↪interp_pts,
        save_fig=False, suffix_str='%i' %int(inc_a_x))

    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]

# np.savez('gain_array_data', gain_array=gain_array)

# npzfile = np.load('gain_array_data.npz')
# gain_array = npzfile['gain_array'].tolist()

fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
im = ax1.imshow(np.abs(gain_array), aspect='auto', interpolation='none',
                vmin=0, vmax=np.max(np.abs(gain_array)))#, cmap='jet')

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 (nm)')
plt.ylabel('Frequency (GHz)')
plt.savefig('lit_03-gain-width_scan.pdf')
plt.savefig('lit_03-gain-width_scan.png')
plt.close()

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

2015 - Van Laer - Nat Phot - FSBF - Waveguide on a Pedestal

Note the absence of an absorptive boundary causes issue of slab layer significantly distorting acoustic modes.

```
""" 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
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
```

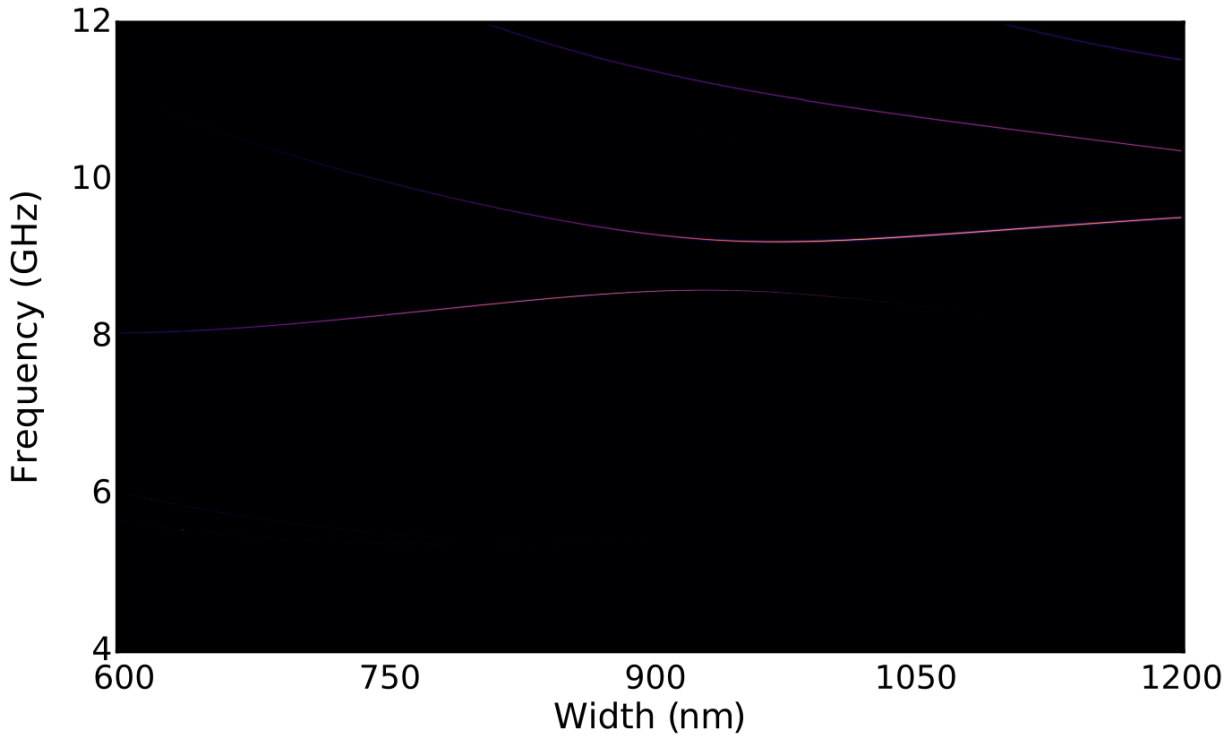

Fig. 3.24: Full acoustic wave spectrum for silica microwire, as per Fig. 4a in paper.

```
import integration
import plotting
from fortran import NumBAT


start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 4*wl_nm
unitcell_y = 0.5*unitcell_x
inc_a_x = 450
inc_a_y = 230
inc_shape = 'pedestal'
pillar_x = 15
pillar_y = 300
slab_a_x = 2000
slab_a_y = 800

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 60
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'lit_04-'
```

```
# 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,
                        pillar_x=pillar_x, pillar_y=pillar_y,
                        material_bkg=materials.Vacuum,            # background
                        material_a=materials.Si_2015_Van_Laer,    # rib
                        material_b=materials.SiO2_2015_Van_Laer,  # slab
                        material_c=materials.SiO2_2015_Van_Laer,  # pillar
                        lc_bkg=6, lc2=3000.0, lc3=100.0)

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

# Calculate Electromagnetic Modes
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

plotting.plt_mode_fields(sim_EM_pump, ivals=[0],
                         xlim_min=0.4, xlim_max=0.4, ylim_min=0.4, ylim_max=0.2,
                         EM_AC='EM_E', pdf_png='png', prefix_str=prefix_str)

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

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

k_AC = 5
shift_Hz = 8e9

# Calculate Acoustic Modes
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)

plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, pdf_png='png')

set_q_factor = 306

# 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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival, fixed_
↪Q=set_q_factor)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = np.real(sim_AC.Eig_values[0])*1e-9 - 2  # GHz
freq_max = np.real(sim_AC.Eig_values[-1])*1e-9 + 2  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str)

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

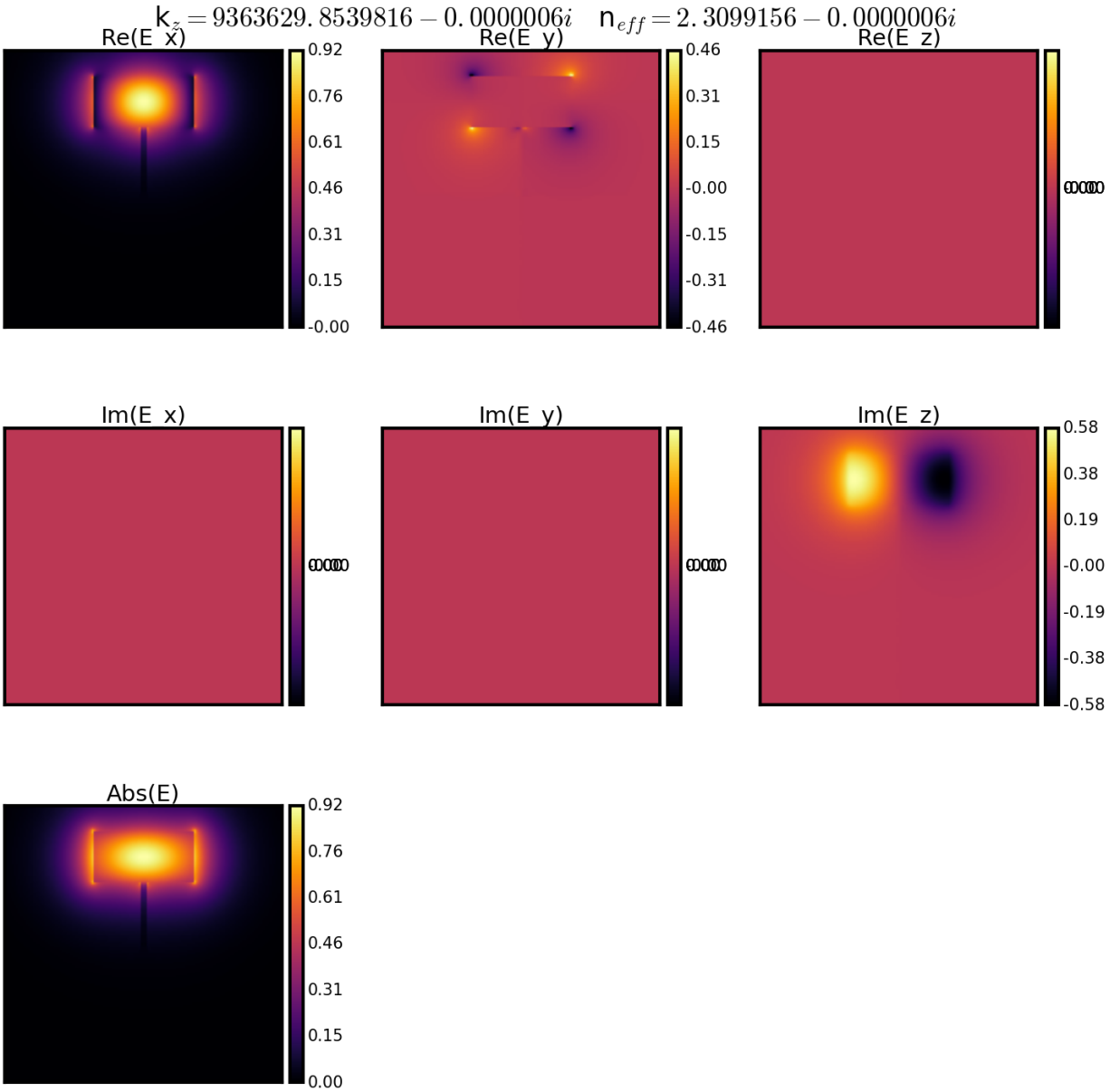

Fig. 3.25: Fundamental optical mode fields.

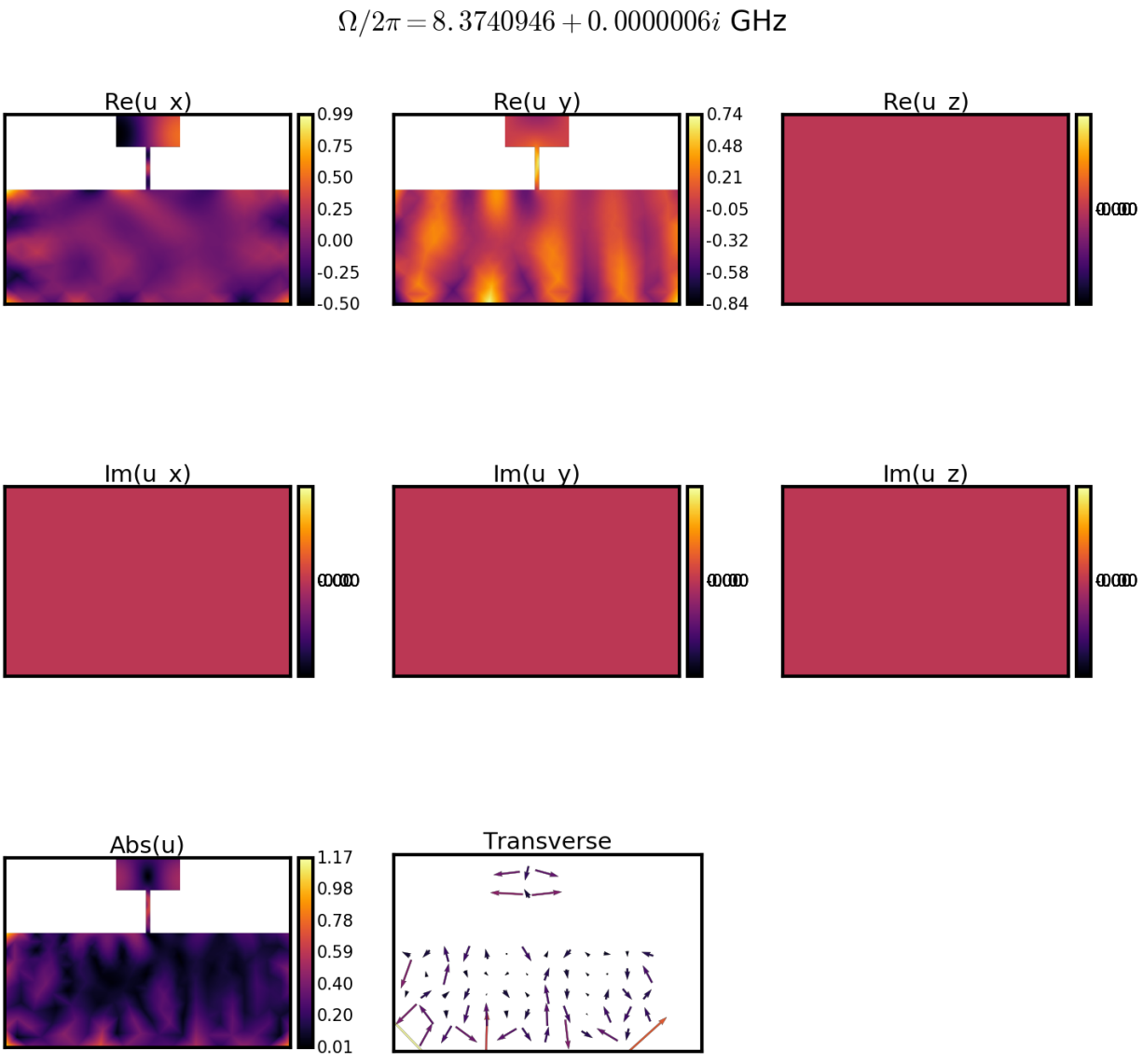

Fig. 3.26: Dominant high gain acoustic mode. Note how the absence of an absorptive boundary on the SiO2 slab causes this layer to significantly distorted the acoustic modes.

2015 - Van Laer - New J Phys - FSBF - Waveguide without Pedestal

```
""" Replicating the results of
    Net on-chip Brillouin gain based on suspended
    silicon nanowires
    Van Laer et al.
    http://dx.doi.org/10.1088/1367-2630/17/11/115005
"""

import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
import copy

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 5*wl_nm
unitcell_y = 0.5*unitcell_x
inc_a_x = 450
inc_a_y = 230
inc_shape = 'rectangular'

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 60
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

prefix_str = 'lit_05-'

# Rotate crystal axis of Si from <100> to <110>, starting with same Si_2016_Smith␣
↪data.
Si_110 = copy.deepcopy(materials.Si_2016_Smith)
Si_110.rotate_axis(np.pi/4,'y-axis', save_rotated_tensors=True)
# Use all specified parameters to create a waveguide object.
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.Vacuum,
                        material_a=Si_110, symmetry_flag=False,
                        lc_bkg=4, lc2=3000.0, lc3=2000.0)

# Expected effective index of fundamental guided mode.
```

```
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic Modes
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
# np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz')
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()

sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)
# np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
# npzfile = np.load('wguide_data2.npz')
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.45, xlim_max=0.45, ivals=[0],
                         ylim_min=0.45, ylim_max=0.45, EM_AC='EM_E',
                         prefix_str=prefix_str, pdf_png='png')

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

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

k_AC = 5 # close but not quite zero

# Calculate Acoustic Modes
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
# np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC = npzfile['sim_AC'].tolist()

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

plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str)

set_q_factor = 750

# 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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival, fixed_
↪Q=set_q_factor)

# Mask negligible gain values to improve clarity of print out.
threshold = 1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival_pump,EM_ival_Stokes,:], 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)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 9.2 # GHz
freq_max = 9.4 # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str, suffix_str='')

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

2016 - Florez - Nat Comm - BSBS - Tapered Fibre - Self Cancel - d = 550 nm

```
""" 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
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt

sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT


start = time.time()

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 2*wl_nm
unitcell_y = unitcell_x
inc_a_x = 550 # Diameter
inc_a_y = inc_a_x
inc_shape = 'circular'

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 40
EM_ival_pump = 0
EM_ival_Stokes = 0
```

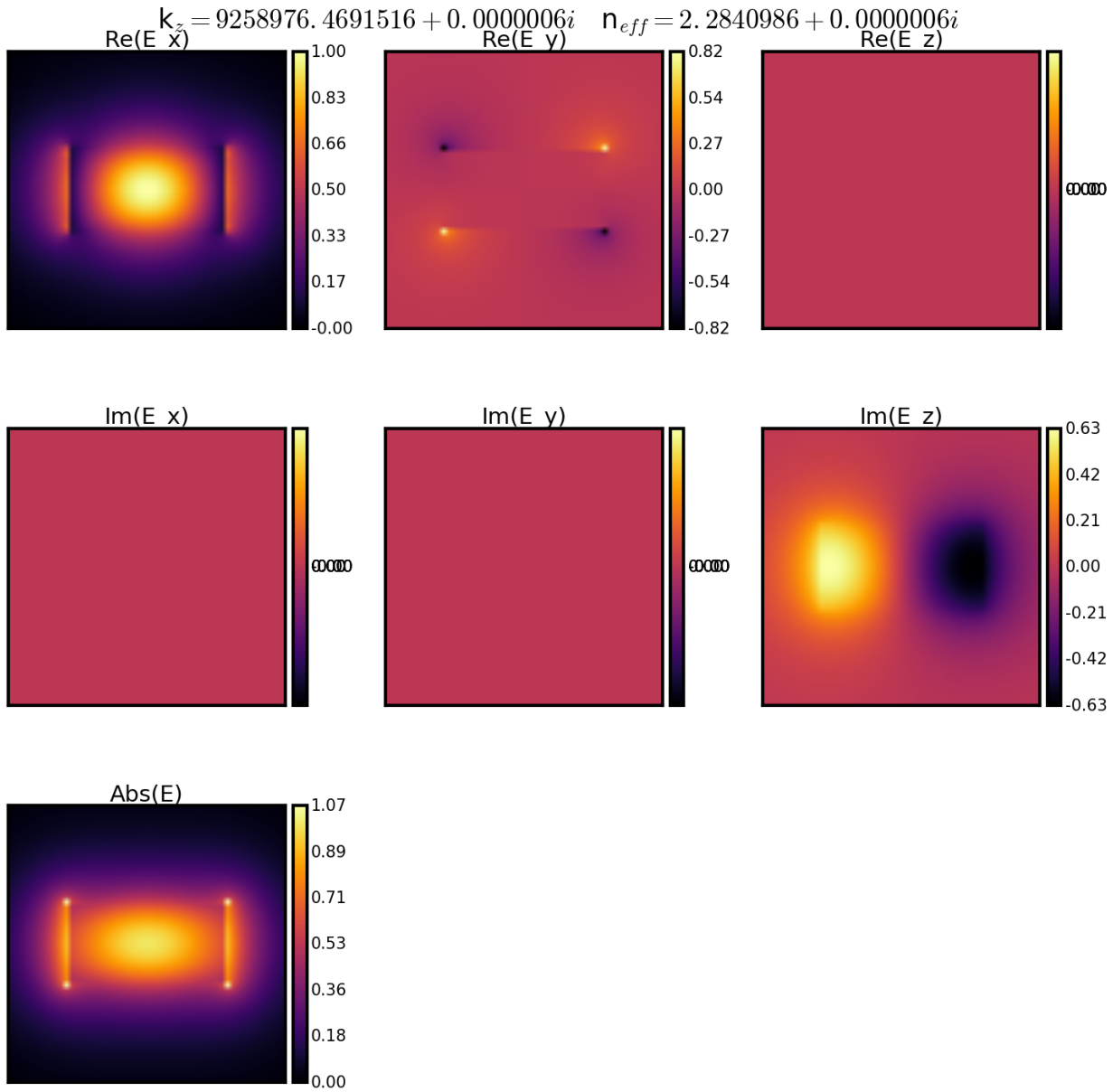

Fig. 3.27: Fundamental optical mode fields.

$$\Omega/2\pi = 9.2755896 - 0.0000126i \text{ GHz}$$

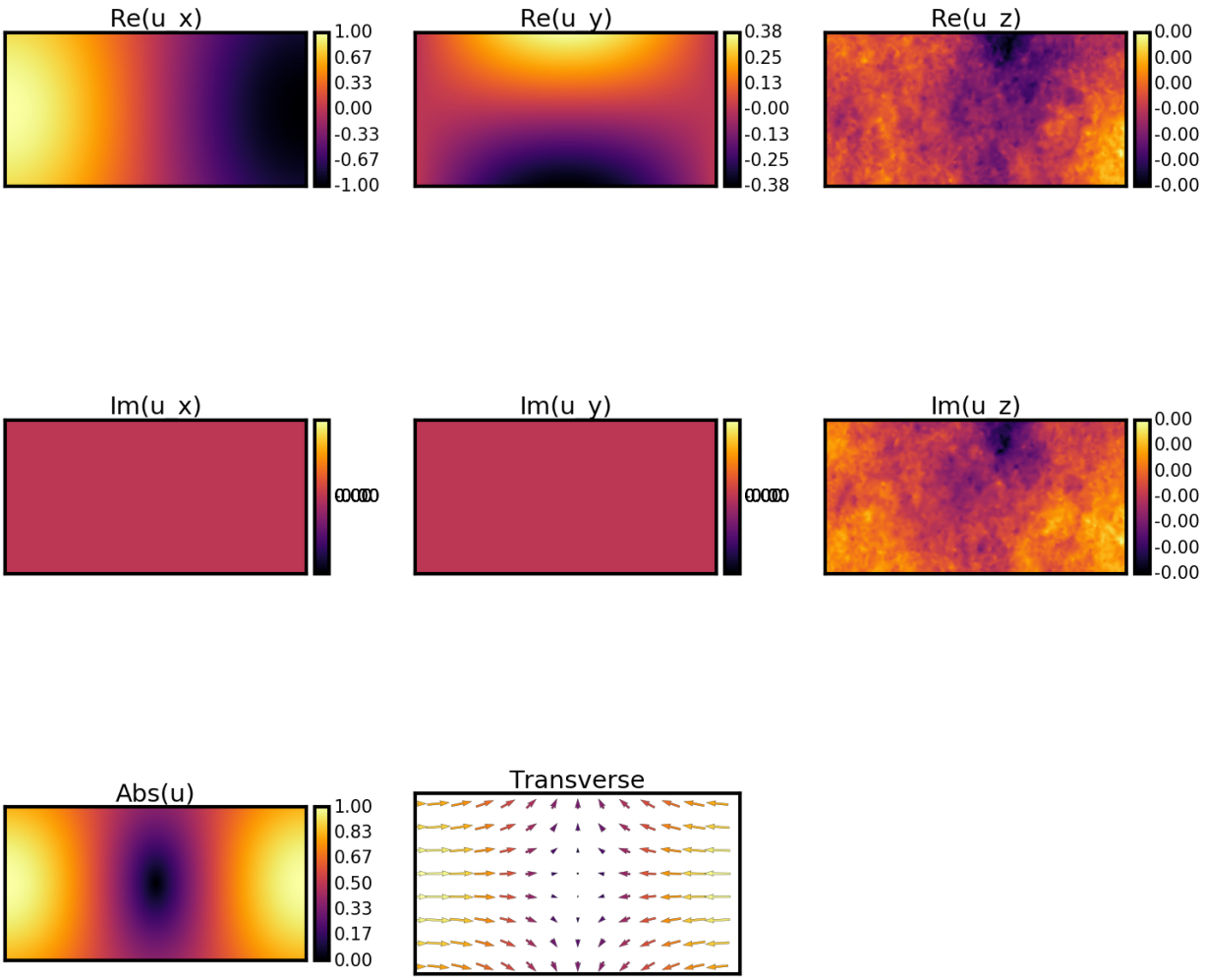

Fig. 3.28: Dominant high gain acoustic mode.

```
AC_ival = 'All'

prefix_str = 'lit_06_1-'

# Use all specified parameters to create a waveguide object.
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.Vacuum,
                        material_a=materials.SiO2_2013_Laude,
                        lc_bkg=3, lc2=2000.0, lc3=1000.0)

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

# Calculate Electromagnetic Modes
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ivals=[0],
                         ylim_min=0.3, ylim_max=0.3, EM_AC='EM_E',
                         prefix_str=prefix_str, pdf_png='png')

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

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])

shift_Hz = 4e9

# Calculate Acoustic Modes
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)

plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, suffix_str='')

# Print the frequencies of AC modes.
print('Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, 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 = 12  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
```

```
    prefix_str=prefix_str)

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

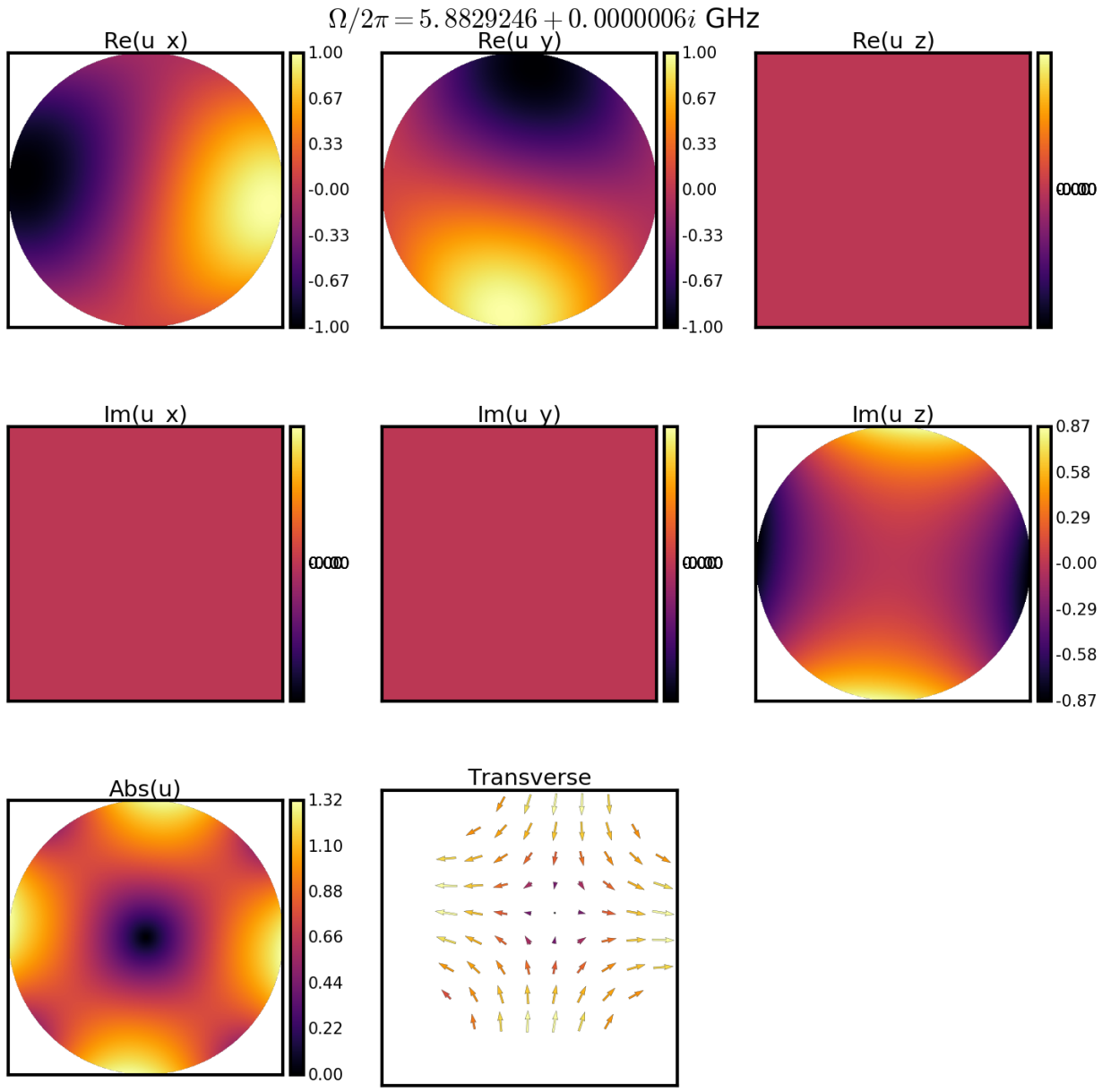

Fig. 3.29: TR21 acoustic mode fields of NW diameter 550 nm.

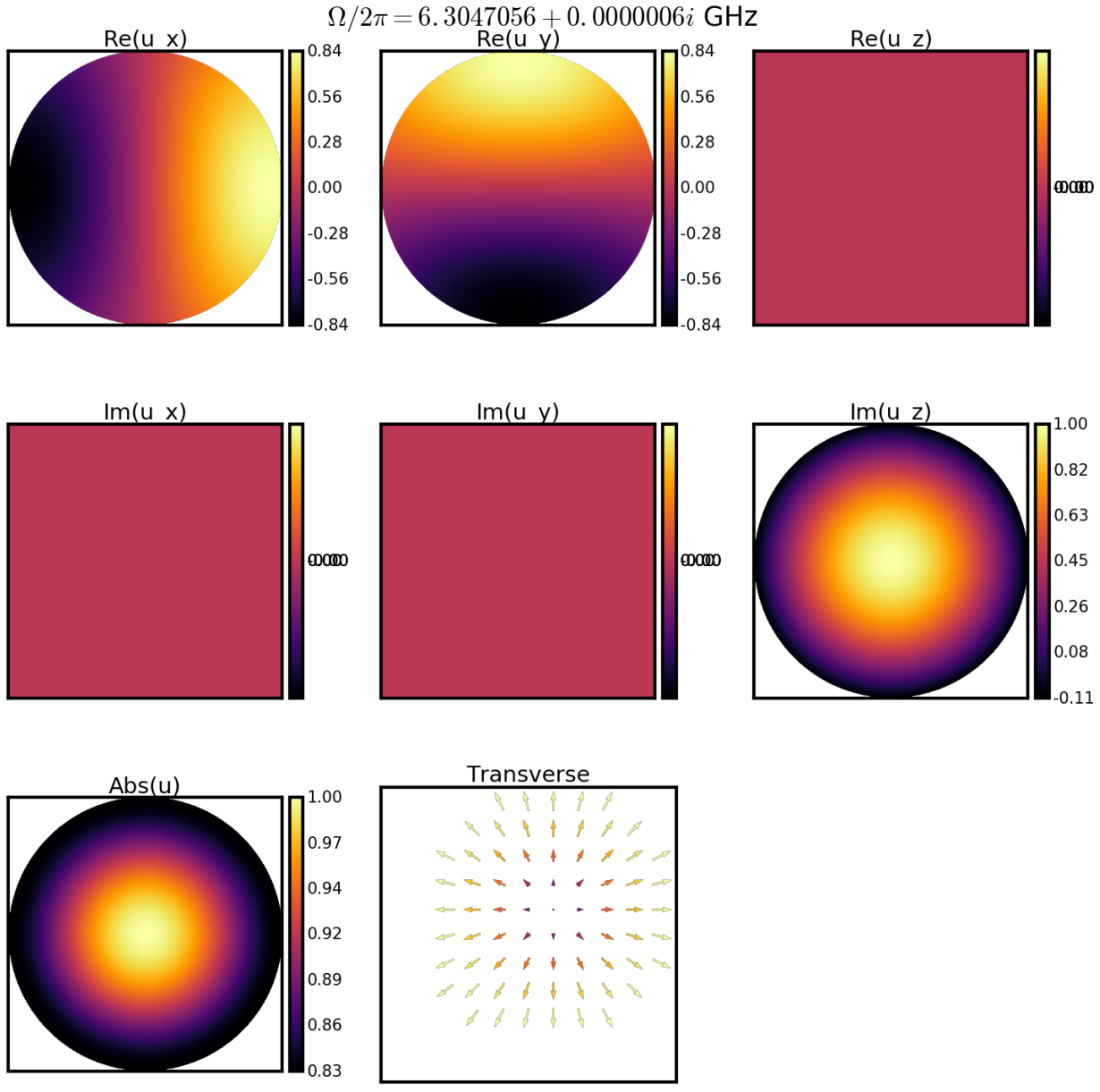

Fig. 3.30: R01 acoustic mode fields of NW diameter 550 nm.

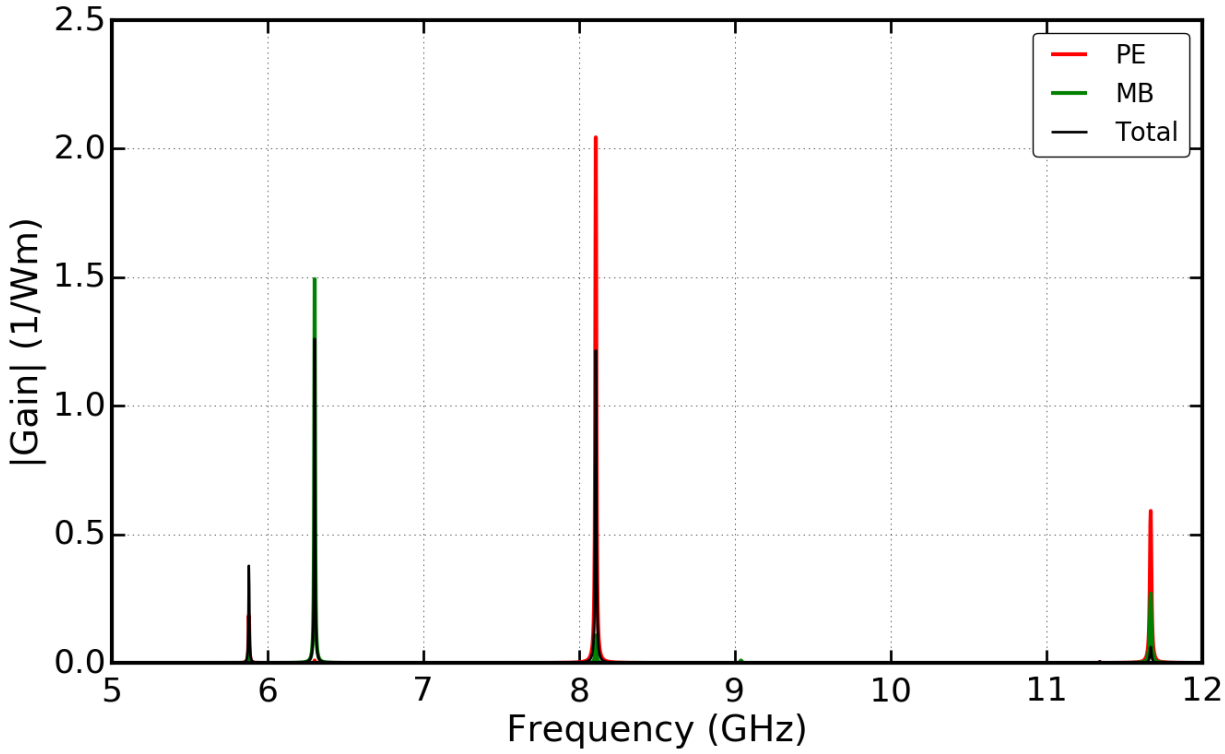

Fig. 3.31: Gain spectra of NW diameter 550 nm, matching blue curve of Fig. 3b in paper.

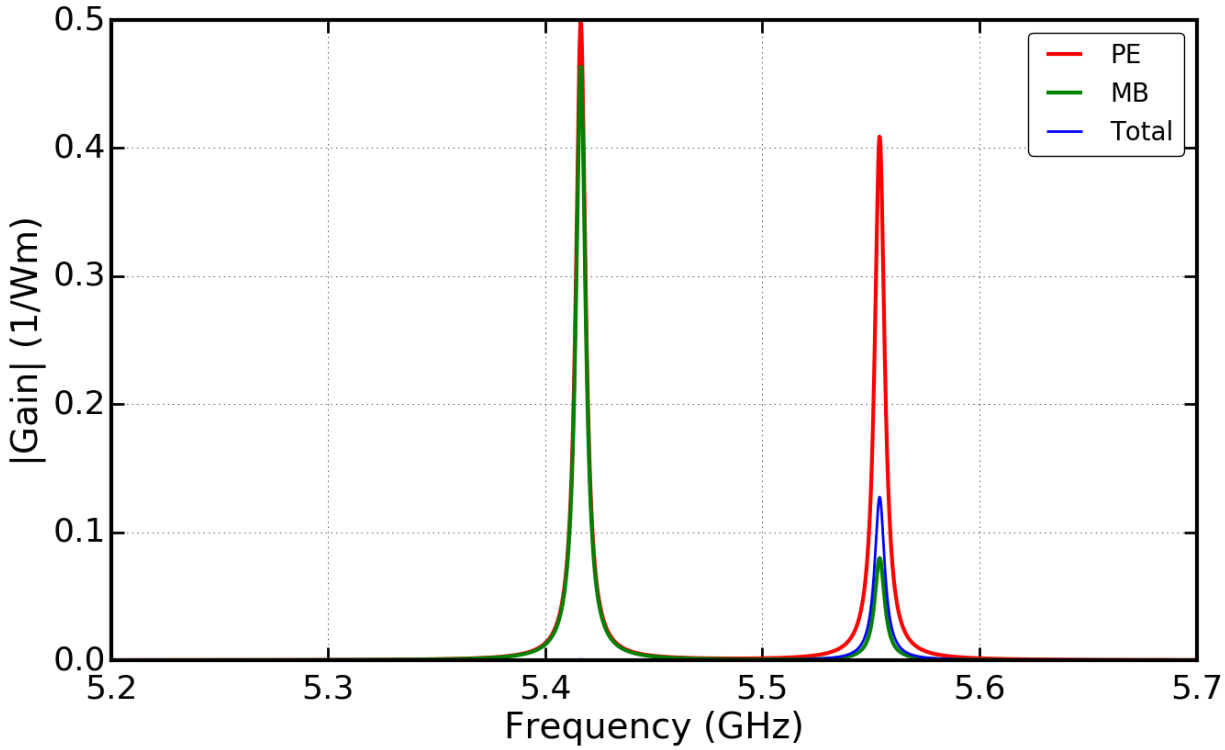

Fig. 3.32: Gain spectra of NW diameter 1160 nm, as in Fig. 4 of paper, showing near perfect cancellation at 5.4 GHz.

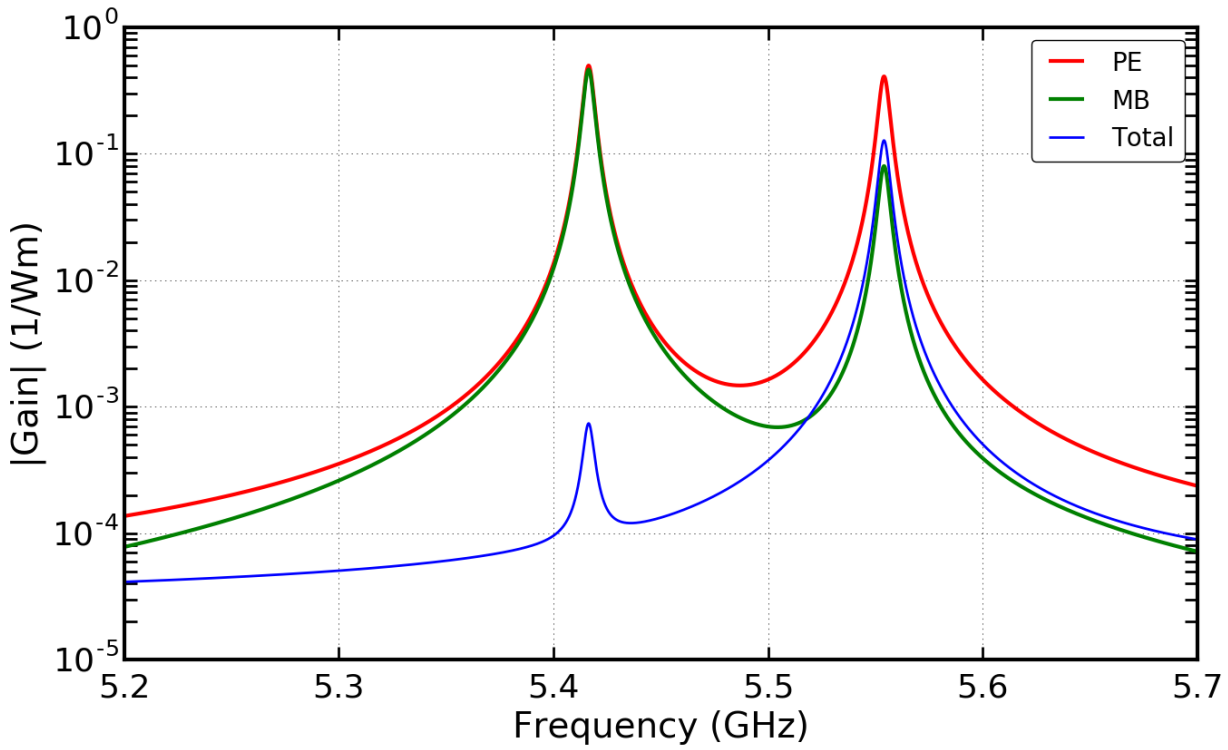

Fig. 3.33: Gain spectra of NW diameter 1160 nm, as in Fig. 4 of paper, showing near perfect cancellation at 5.4 GHz.

2016 - Florez - Nat Comm - BSBS - Tapered Fibre - Self Cancel - d = 1160 nm

2016 - Kittlaus - Nat Phot - FSBF - Rib Waveguide

```
""" Replicating the results of
    Large Brillouin amplification in silicon
    Kittlaus et al.
    http://dx.doi.org/10.1038/nphoton.2016.112
"""

import time
import datetime
import numpy as np
import sys
import copy

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 = 5*wl_nm
unitcell_y = 0.2*unitcell_x
# Waveguide widths.
inc_a_x = 1000
inc_a_y = 80
# Shape of the waveguide.
inc_shape = 'rib'

slab_a_x = 3000
slab_a_y = 130

# Number of electromagnetic modes to solve for.
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
# Number of acoustic modes to solve for.
num_modes_AC = 40
# The EM pump mode(s) for which to calculate interaction with AC modes.
# Can specify a mode number (zero has lowest propagation constant) or 'All'.
EM_ival_pump = 0
# The EM Stokes mode(s) for which to calculate interaction with AC modes.
EM_ival_Stokes = 0
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Si_110 = copy.deepcopy(materials.Si_2015_Van_Laer)
Si_110 = copy.deepcopy(materials.Si_2016_Smith)
Si_110.rotate_axis(np.pi/4,'y-axis', save_rotated_tensors=True)

prefix_str = 'lit_07-'

# 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,
                        slab_a_x=slab_a_x, slab_a_y=slab_a_y,
                        material_bkg=materials.Vacuum,
                        material_a=Si_110,
                        material_b=Si_110, symmetry_flag=False,
                        lc_bkg=5, lc2=4000.0, lc3=2000.0)
# Expected effective index of fundamental guided mode.
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic Modes
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
# np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz')
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()

sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)

plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ivals=[0],
```

```
                        ylim_min=0.3, ylim_max=0.3, EM_AC='EM_E', num_ticks=3,
                        prefix_str=prefix_str, pdf_png='png')

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

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

k_AC = 5

shift_Hz = 2e9

# Calculate Acoustic Modes
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)
# np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC = npzfile['sim_AC'].tolist()

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

plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, #ivals=[0,1,2,3,4,
↪5,6,7,8,9],
     num_ticks=3, xlim_min=0.1, xlim_max=0.1)

set_q_factor = 680.

# 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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival, fixed_
↪Q=set_q_factor)

# Mask negligible gain values to improve clarity of print out.
threshold = 1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival_pump,EM_ival_Stokes,:], 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)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 4.2  # GHz
freq_max = 4.3  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str, suffix_str='')
```

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

$$k_z = 10951864.5922676 - 0.0000006i \quad n_{eff} = 2.7017176 - 0.0000006i$$

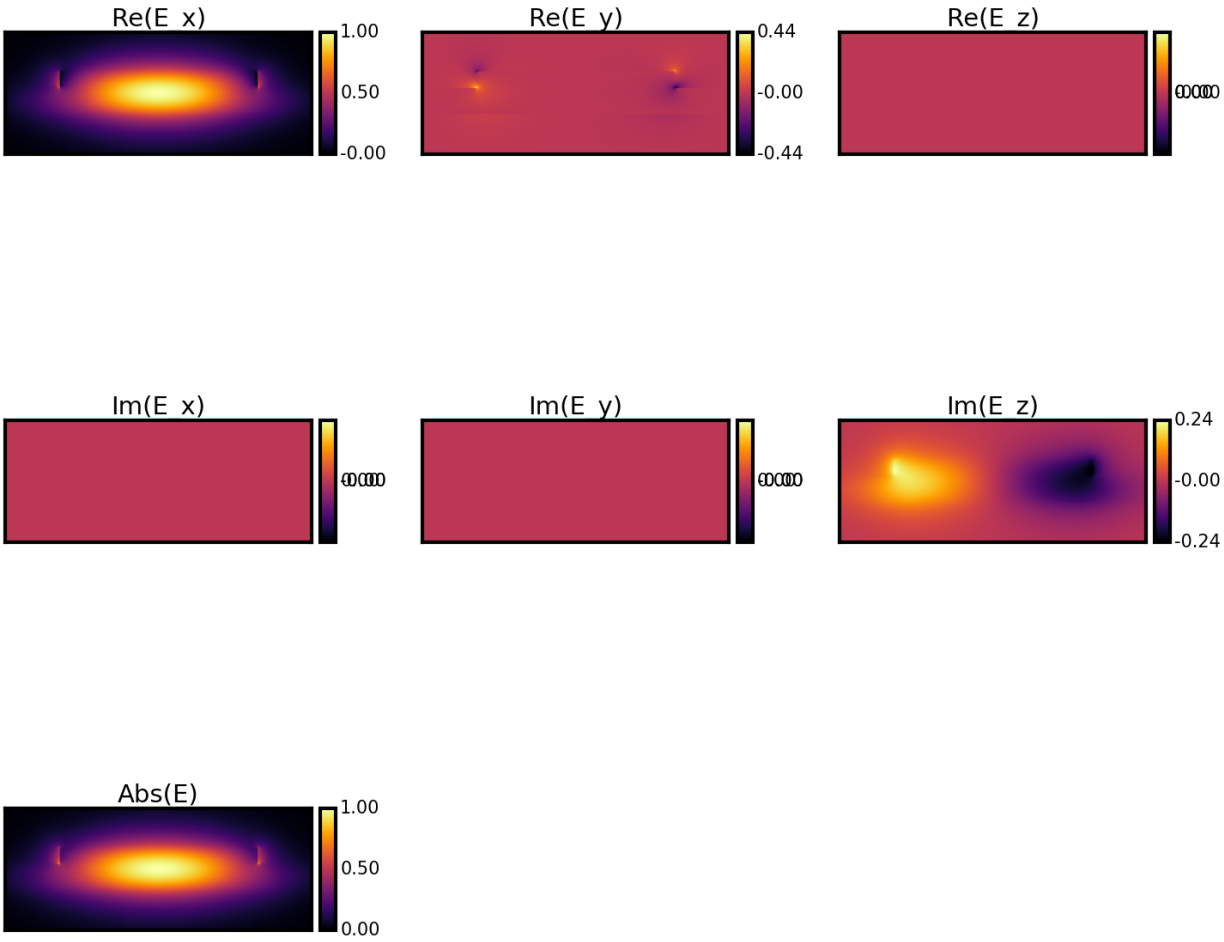

Fig. 3.34: Fundamental optical mode fields.

2017 - Kittlaus - Nat Comm - FSBF - Intermode

```
""" Replicating the results of
    On-chip inter-modal Brillouin scattering
    Kittlaus et al.
    http://dx.doi.org/10.1038/ncomms15819
"""

import time
```

$$\Omega/2\pi = 4.2588156 - 0.0000006i \text{ GHz}$$

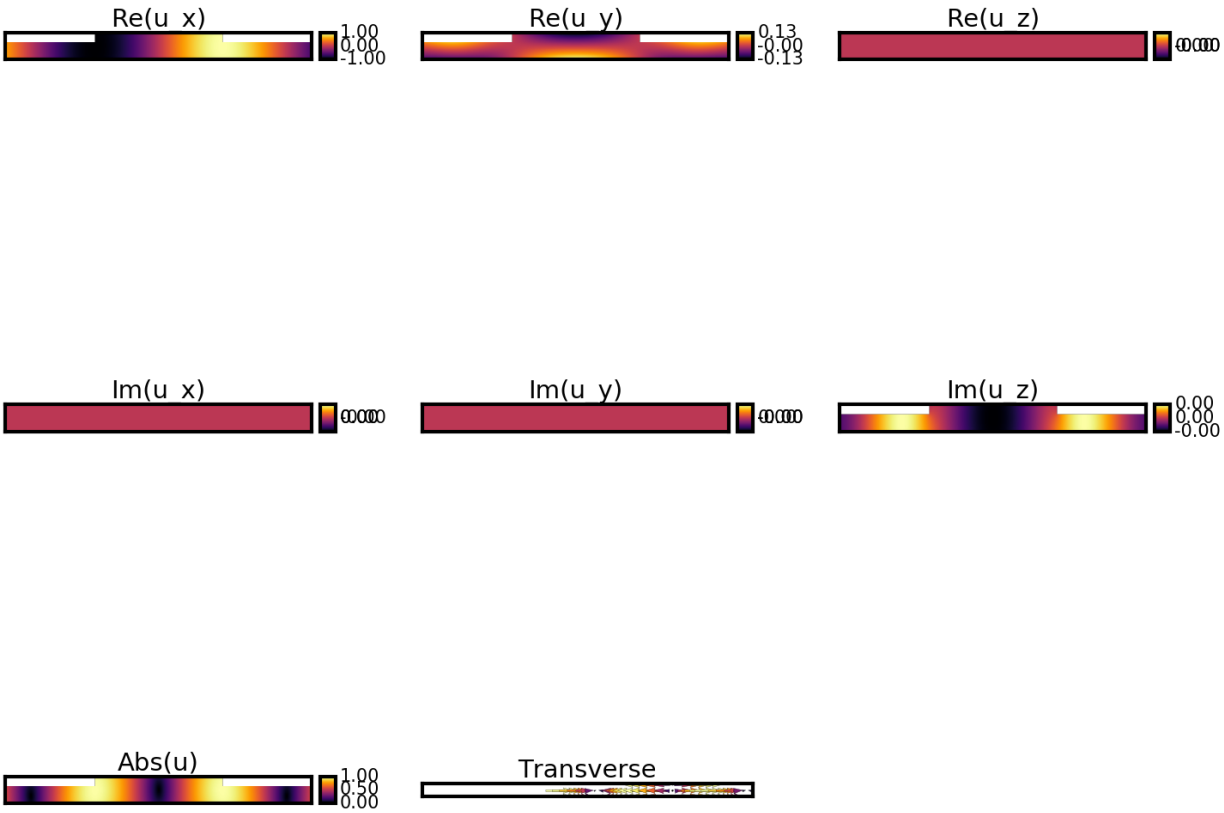

Fig. 3.35: Dominant high gain acoustic mode.

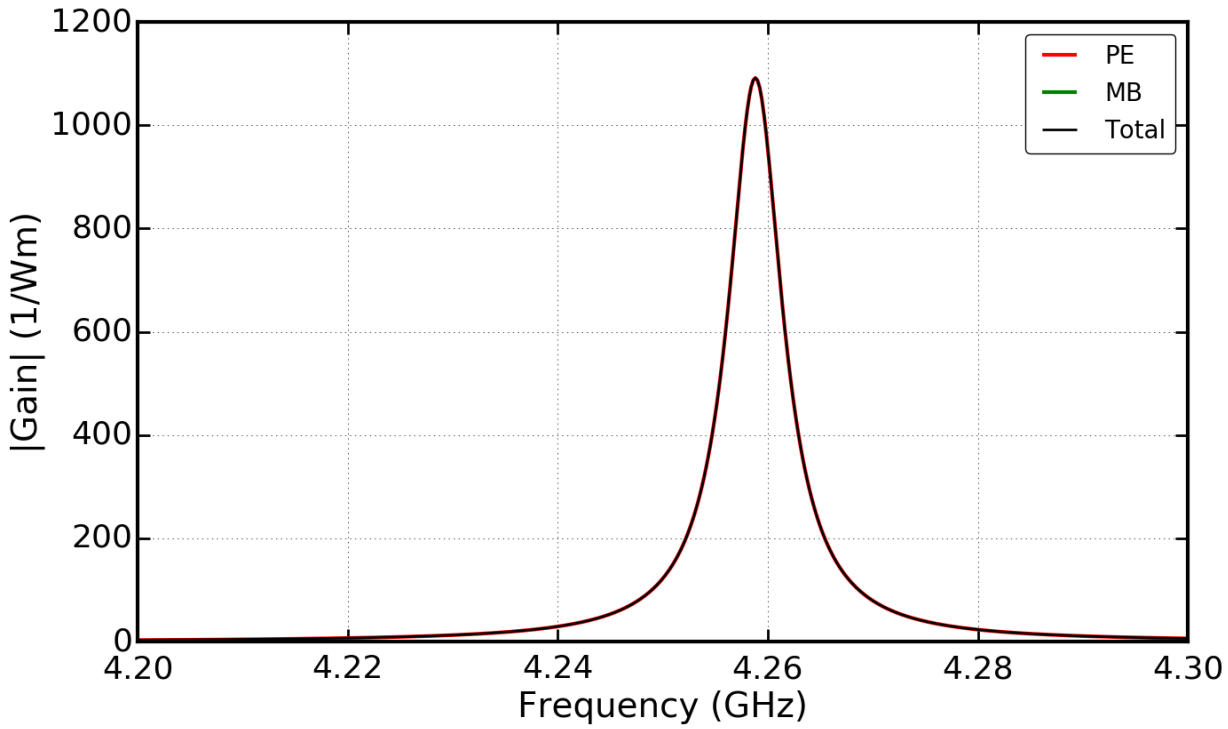

Fig. 3.36: Gain spectra showing gain due to photoelastic effec, gain due to moving boundary effect, and total gain.

```
import datetime
import numpy as np
import sys
import copy

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 = 5*wl_nm
unitcell_y = 0.2*unitcell_x
# Waveguide widths.
inc_a_x = 1500
inc_a_y = 80
```

```
# Shape of the waveguide.
inc_shape = 'rib'

slab_a_x = 2850
slab_a_y = 135

# Number of electromagnetic modes to solve for.
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
# Number of acoustic modes to solve for.
num_modes_AC = 35
# The EM pump mode(s) for which to calculate interaction with AC modes.
# Can specify a mode number (zero has lowest propagation constant) or 'All'.
EM_ival_pump = 0
# The EM Stokes mode(s) for which to calculate interaction with AC modes.
EM_ival_Stokes = 1 # INTERMODE SBS TE0 to TE1
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Si_110 = copy.deepcopy(materials.Si_2015_Van_Laer)
Si_110 = copy.deepcopy(materials.Si_2016_Smith)
Si_110.rotate_axis(np.pi/4,'y-axis', save_rotated_tensors=True)

prefix_str = 'lit_08-'

# Use 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,
                        material_bkg=materials.Vacuum,
                        material_a=Si_110,
                        material_b=Si_110, symmetry_flag=False,
                        lc_bkg=5, lc2=4000.0, lc3=2000.0)
# Expected effective index of fundamental guided mode.
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic Modes
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
# np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz')
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()

sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)
# np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
# npzfile = np.load('wguide_data2.npz')
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

plotting.plt_mode_fields(sim_EM_pump, xlim_min=0.35, xlim_max=0.35, ivals=[0,1],
                         ylim_min=0.3, ylim_max=0.3, EM_AC='EM_E', num_ticks=3,
                         prefix_str=prefix_str, pdf_png='png')

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

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
```

```
print('Intermode q_AC (Hz) \n', k_AC)

shift_Hz = 2e9

# Calculate Acoustic Modes
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz)
# np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC = npzfile['sim_AC'].tolist()

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

plotting.plt_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,
     num_ticks=3, xlim_min=0.1, xlim_max=0.1)

set_q_factor = 460.

# 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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival, fixed_
↪Q=set_q_factor)

# Mask negligible gain values to improve clarity of print out.
threshold = 1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival_pump,EM_ival_Stokes,:], 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)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 0.5  # GHz
freq_max = 9.5  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str, suffix_str='')

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

2017 - Morrison - Optica - BSBS - Chalcogenide Rib Waveguide

```
""" Replicating the results of
    Compact Brillouin devices through hybrid
```

$$k_z = 11190697.8320306 - 0.0000006i \quad n_{eff} = 2.7606356 - 0.0000006i$$

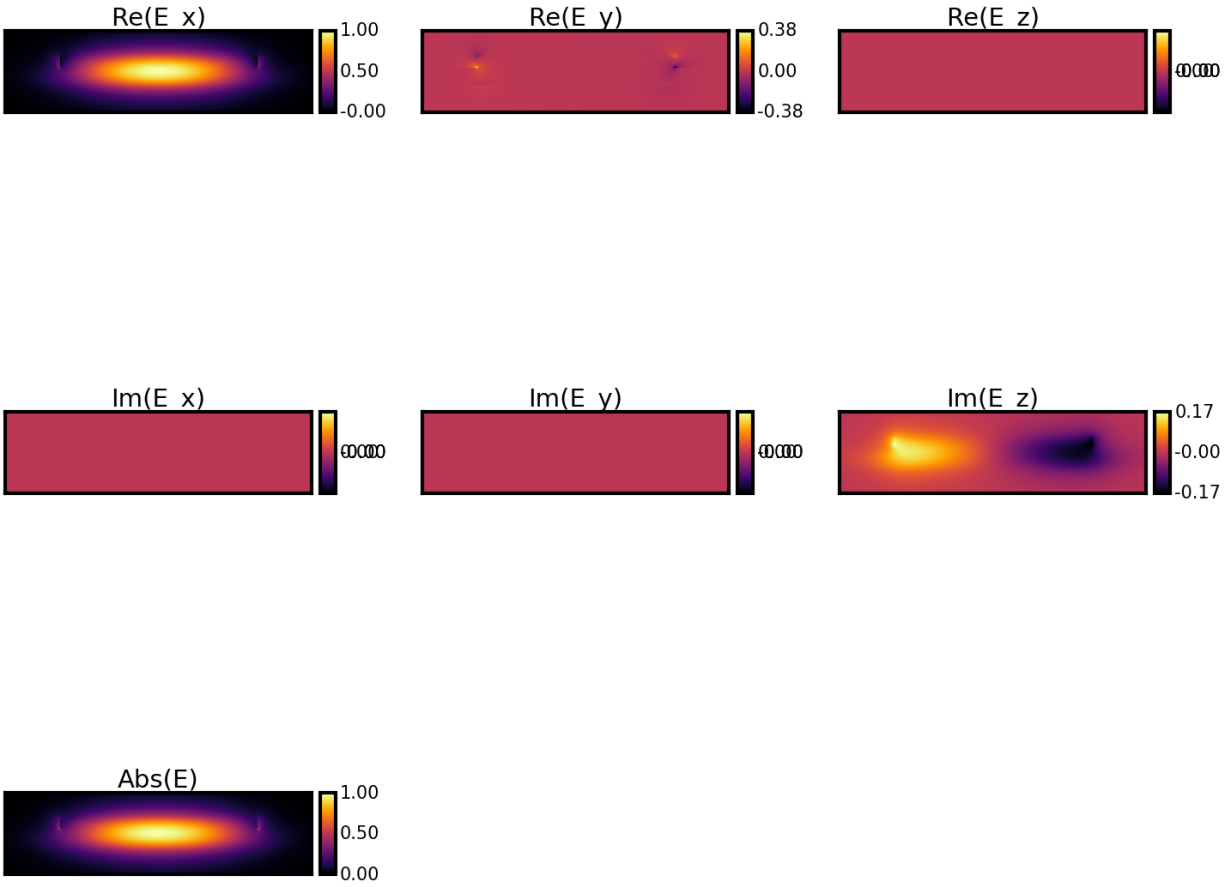

Fig. 3.37: Fundamental (symmetric TE-like) optical mode fields.

$$k_z = 10734656.3488916 - 0.0000006i \quad n_{eff} = 2.6481346 - 0.0000006i$$

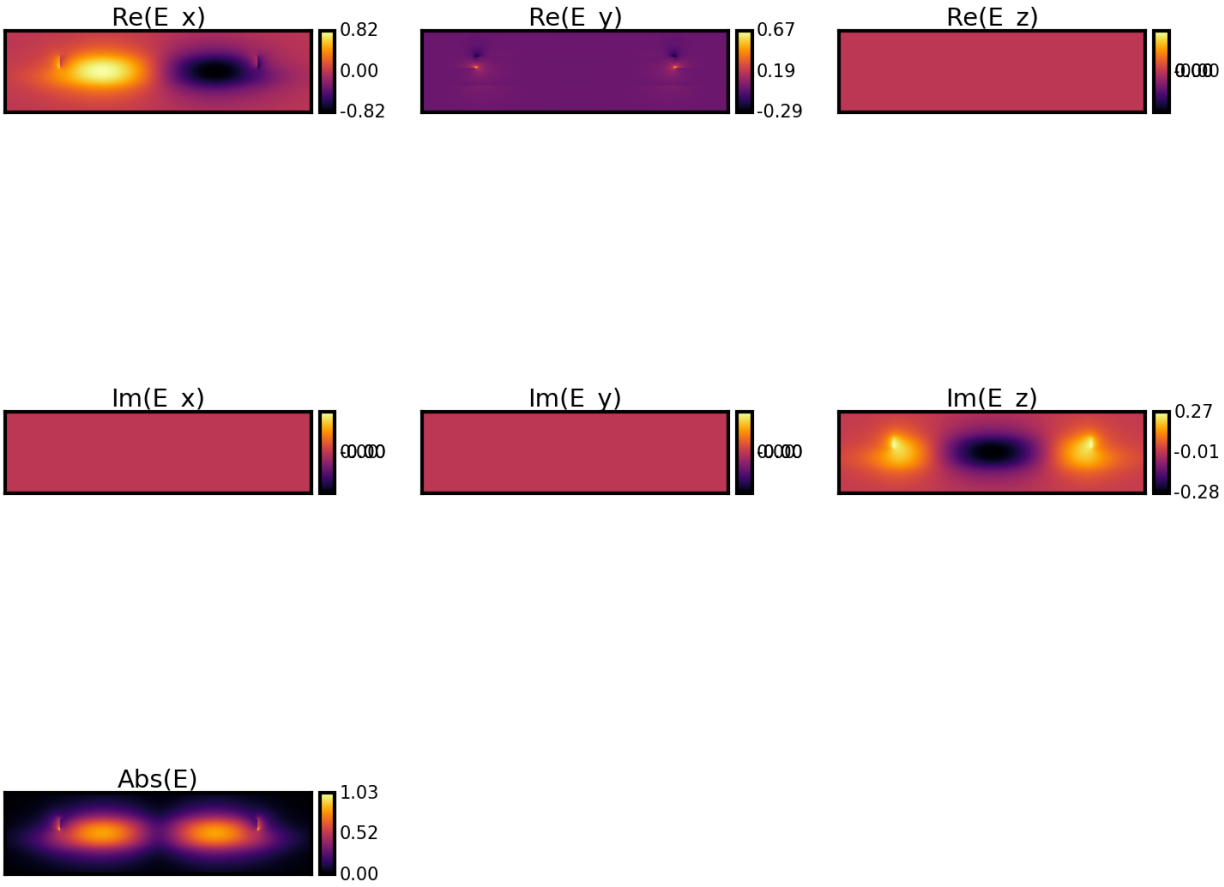

Fig. 3.38: 2nd lowest order (anti-symmetric TE-like) optical mode fields.

$$\Omega/2\pi = 5.9069856 + 0.0000006i \text{ GHz}$$

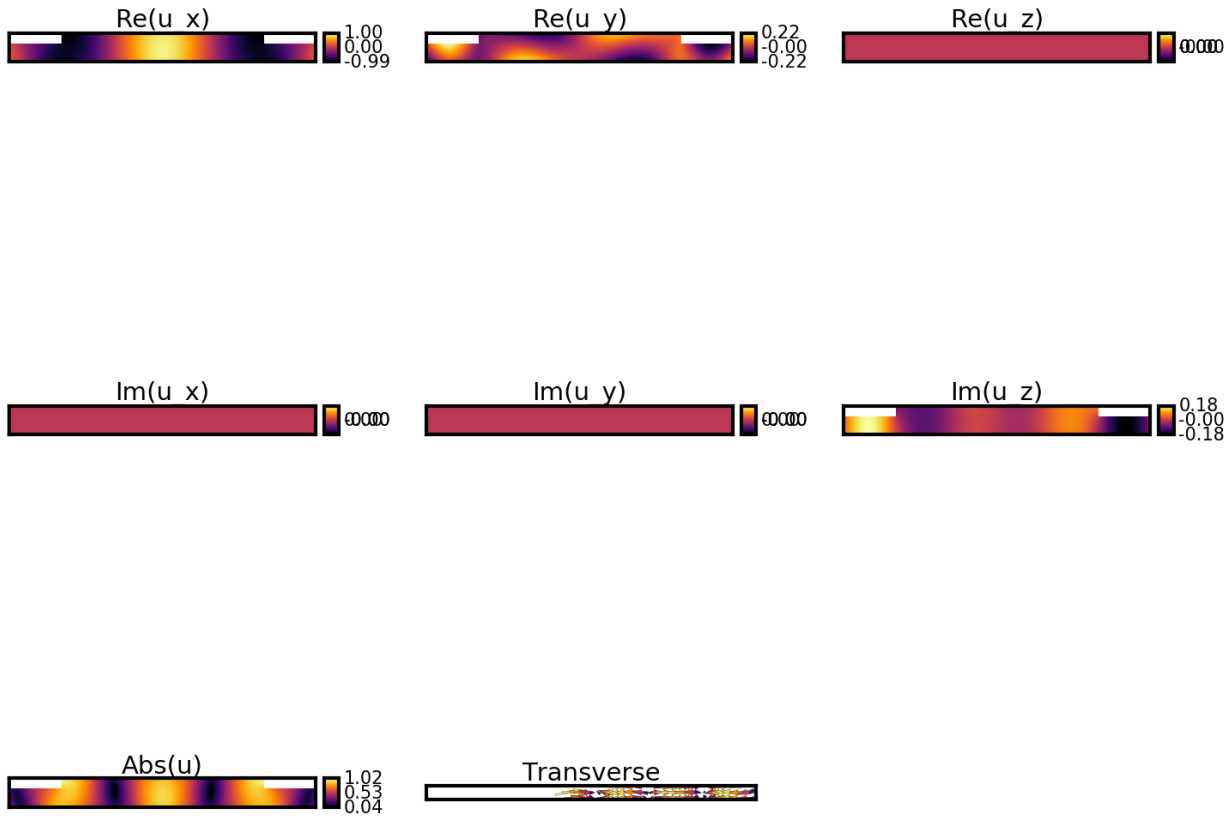

Fig. 3.39: Dominant high gain acoustic mode.

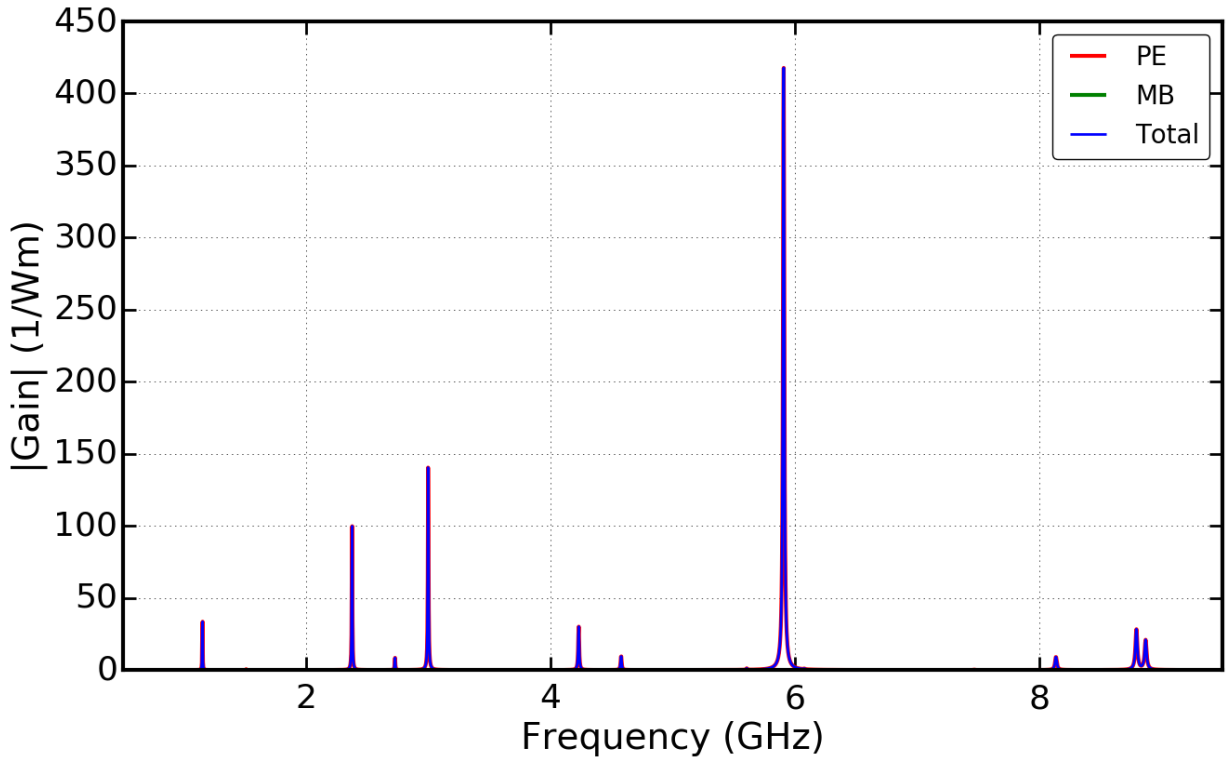

Fig. 3.40: Gain spectra showing gain due to photoelastic effec, gain due to moving boundary effect, and total gain.

```
    integration on Silicon
    Morrison et al.
    https://doi.org/10.1364/OPTICA.4.000847
"""

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 = 5*wl_nm
unitcell_y = 0.5*unitcell_x
# Waveguide widths.
inc_a_x = 1900
inc_a_y = 680
# Shape of the waveguide.
inc_shape = 'rib_coated'

slab_a_x = 3000
slab_a_y = 200

coat_x = 100
coat_y = 200


# Number of electromagnetic modes to solve for.
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
# Number of acoustic modes to solve for.
num_modes_AC = 30
# The EM pump mode(s) for which to calculate interaction with AC modes.
# Can specify a mode number (zero has lowest propagation constant) or 'All'.
EM_ival_pump = 0
# The EM Stokes mode(s) for which to calculate interaction with AC modes.
EM_ival_Stokes = 0
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

prefix_str = 'lit_09-'

# 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,
                        slab_a_x=slab_a_x, slab_a_y=slab_a_y, coat_x=coat_x, coat_
↪y=coat_y,
                        material_bkg=materials.Vacuum,
                        material_a=materials.As2S3_2016_Smith, # waveguide
                        material_b=materials.Si_2016_Smith,     # slab
                        material_c=materials.SiO2_2013_Laude,     # coating
                        lc_bkg=5, lc2=2000.0, lc3=1000.0)

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

# Calculate the Electromagnetic modes of the pump field.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff)
# # np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz')
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()

# Calculate the Electromagnetic modes of the Stokes field.
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
# np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
# npzfile = np.load('wguide_data2.npz')
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

# Print the wavevectors of EM modes.
```

```
print('\n k_z of EM modes \n', np.round(np.real(sim_EM_pump.Eig_values),4))

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

k_AC = np.real(sim_EM_pump.Eig_values[EM_ival_pump] - sim_EM_Stokes.Eig_values[EM_
↪ival_Stokes])
print('\n AC wavenumber (1/m) = ', np.round(k_AC, 4))

# Calculate Acoustic modes.
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
# # np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz')
# sim_AC = npzfile['sim_AC'].tolist()

# Print the frequencies of AC modes.
print('\n Freq of AC modes (GHz) \n', np.round(np.real(sim_AC.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, linewidth_Hz, Q_factors, alpha = integration.gain_
↪and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, k_AC,
    EM_ival_pump=EM_ival_pump, EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)
# Print the Backward SBS gain of the AC modes.
print("\n SBS_gain PE contribution \n", SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:])
print("SBS_gain MB contribution \n", SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:])
print("SBS_gain total \n", SBS_gain[EM_ival_pump,EM_ival_Stokes,:])
# Mask negligible gain values to improve clarity of print out.
threshold = -1e-3
masked_PE = np.ma.masked_inside(SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked_MB = np.ma.masked_inside(SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:], 0,␣
↪threshold)
masked = np.ma.masked_inside(SBS_gain[EM_ival_pump,EM_ival_Stokes,:], 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)


# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 7  # GHz
freq_max = 10  # GHz
plotting.gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, k_AC,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix_str=prefix_str)

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

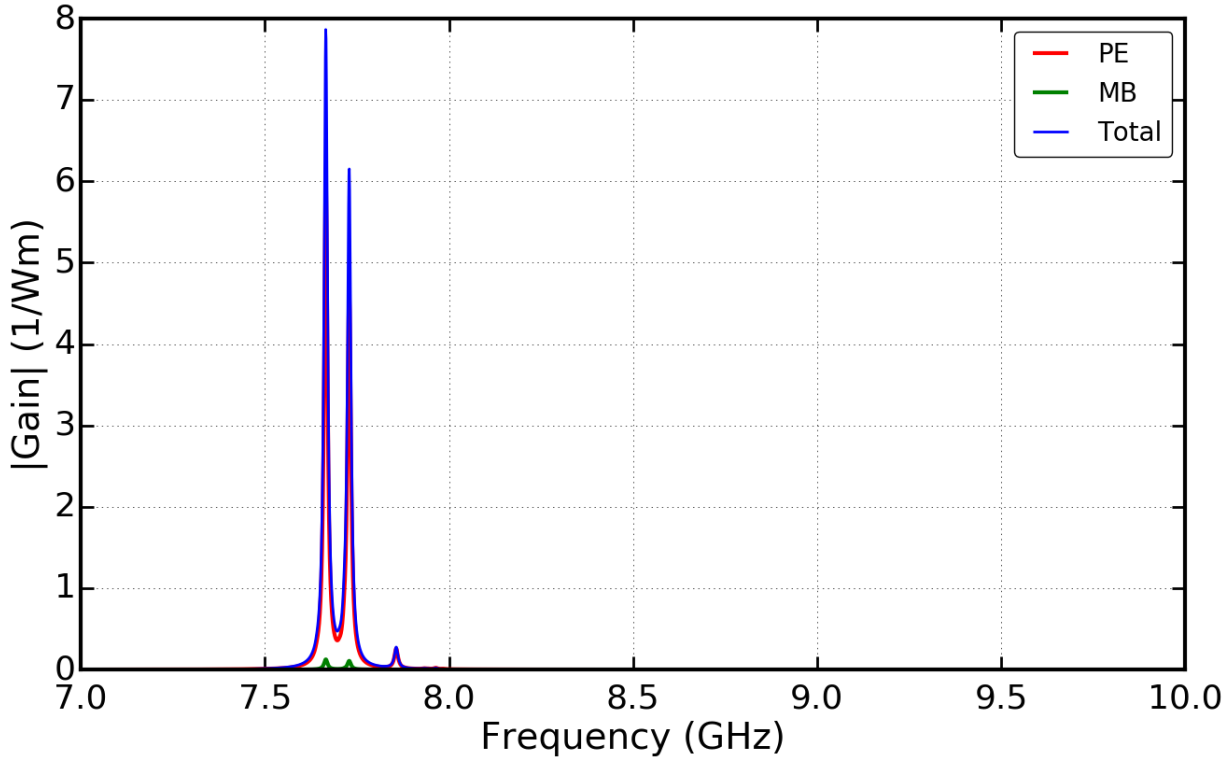

Fig. 3.41: Gain spectra showing gain due to photoelastic effec, gain due to moving boundary effect, and total gain.

CHAPTER

FOUR

PYTHON BACKEND

objects module

The `objects` module provides functions for defining and constructing waveguides. objects.py is a subroutine of NumBAT. It contains the Struct objects that represent the structure being simulated.

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_x=None*, *coat_y=None*, *inc_b_x=None*, *inc_b_y=None*, *two_inc_sep=None*, *incs_y_offset=None*, *pillar_x=None*, *pillar_y=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*, *material_bkg=<materials.Material object>*, *material_a=<materials.Material object>*, *material_b=<materials.Material object>*, *material_c=<materials.Material object>*, *material_d=<materials.Material object>*, *material_e=<materials.Material object>*, *material_f=<materials.Material object>*, *material_g=<materials.Material object>*, *material_h=<materials.Material object>*, *material_i=<materials.Material object>*, *material_j=<materials.Material object>*, *material_k=<materials.Material object>*, *material_l=<materials.Material object>*, *material_m=<materials.Material object>*, *material_n=<materials.Material object>*, *material_o=<materials.Material object>*, *material_p=<materials.Material object>*, *material_q=<materials.Material object>*, *material_r=<materials.Material object>*, *loss=True*, *symmetry_flag=True*, *make_mesh_now=True*, *force_mesh=True*, *mesh_file='NEED_FILE.mail'*, *check_mesh=False*, *plt_mesh=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', 'rib' 'slot_coated', 'rib_coated', 'pedestal', 'onion'. 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.
- **symmetry_flag** (*bool*) – True if materials all have sufficient symmetry that their tensors contain only 3 unique values. If False must specify full [3,3,3,3] tensors.
- **material_bkg** – A `Material` instance - check backend/msh_type_lib
- **material_a** – A `Material` instance - check backend/msh_type_lib
- **material_b** – A `Material` instance - check backend/msh_type_lib
- **material_c-r** – A `Material` instance - check backend/msh_type_lib
- **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/
- **plt_mesh** (*bool*) – Plot a png of the mesh.
- **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 = lc_bkg / lc2. Larger lc2 = finer mesh.
- **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(*num_modes*, *k_AC*, *shift_Hz=None*, *EM_sim=None*, ***args*)
Run a simulation to find the Struct's acoustic modes.

Parameters **num_modes** (*int*) – Number of AC modes to solve for.

Keyword Arguments

- **k_AC** (*float*) – Wavevector of AC modes.
- **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(*num_modes*, *wl_nm*, *n_eff*, *Stokes=False*, ***args*)
Run a simulation to find the Struct's EM modes.

Parameters

- **num_modes** (*int*) – Number of EM modes to solve for.
- **wl_nm** (*float*) – Wavelength of EM wave in vacuum.
- **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**(*data_file*)
Bases: object

Represents a material with:

Refractive index [] Density [kg/m3] Stiffness tensor component [Pa] Photoelastic tensor component [] Acoustic loss tensor component [Pa s]

load_data_file(*data_file*, *alt_path=''*)
Load data from json file.

Parameters

- **data_file** (*str*) – name of data file located in NumBAT/backend/material_data
- **alt_path** (*str*) – non standard path to data_file

rotate_axis(*theta*, *rotate_axis*, *save_rotated_tensors=False*)
Rotate crystal axis by theta radians.

Parameters

- **theta** (*float*) – Angle to rotate by in radians.
- **rotate_axis** (*str*) – Axis around which to rotate.

Keyword Arguments **save_rotated_tensors** (*bool*) – Save rotated tensors to csv.

Returns Material object with rotated tensor values.

materials.**isotropic_stiffness**(*E*, *v*)
Calculate the stiffness matrix components of isotropic materials, given the two free parameters.

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

Parameters

- **E** (*float*) – Youngs_modulus
- **v** (*float*) – Poisson_ratio

materials.**rotate_tensor**(*tensor_orig*, *theta*, *rotation_axis*)
Rotate all acoustic material tensor by theta radians around chosen rotation_axis.

Parameters

- **tensor_orig** (*array*) – Tensor to be rotated.
- **theta** (*float*) – Angle to rotate by in radians.
- **rotation_axis** (*str*) – Axis around which to rotate.

materials.**rotation_matrix_sum**(*i*, *j*, *k*, *l*, *tensor_orig*, *mat_R*)
Inner loop of rotation matrix summation.

mode_calcs module

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

Copyright (C) 2016 Bjorn Sturmberg, Kokou Dossou

class mode_calcs.**Simmo**(*structure*, *num_modes=20*, *wl_nm=1*, *n_eff=None*, *shift_Hz=None*, *k_AC=None*, *EM_sim=None*, *Stokes=False*)
Bases: object

Calculates the modes of a Struct 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 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]

AC_mode_energy_elastic: the elastic power in the acoutic modes.

calc_EM_modes()

Run a Fortran FEM calculation to find the optical modes.

Returns a Simmo object that 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]

EM_mode_power: the power in the optical modes. Note this power is negative for modes travelling in the negative z-direction, eg the Stokes wave in backward SBS.

mode_calcs.**bkwd_Stokes_modes**(*EM_sim*)

Defines the backward travelling Stokes waves as the conjugate of the forward travelling pump waves.

Returns a Simmo object that 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]

EM_mode_power: the power in the Stokes modes. Note this power is negative because the modes are travelling in the negative z-direction.

mode_calcs.**fwd_Stokes_modes**(*EM_sim*)

Defines the forward travelling Stokes waves as a copy of the forward travelling pump waves.

Returns a Simmo object that has these key values:

integration module

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

Copyright (C) 2016 Bjorn Sturmberg, Kokou Dossou

integration.**comsol_fields**(*data_file*, *n_points*, *ival=0*)

Load Comsol field data on (assumed) grid mesh.

integration.**gain_and_qs**(*sim_EM_pump*, *sim_EM_Stokes*, *sim_AC*, *k_AC*, *EM_ival_pump=0*, *EM_ival_Stokes=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

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

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

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

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

Parameters

- **sim_EM_pump** (Simmo object) – Contains all info on pump EM modes
- **sim_EM_Stokes** (Simmo object) – Contains all info on Stokes EM modes
- **sim_AC** (Simmo object) – Contains all info on AC modes
- **k_AC** (*float*) – Propagation constant of acoustic modes.

Keyword Arguments

- **EM_ival_pump** (*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_ival_Stokes** (*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. Note this will be negative for backwards SBS because gain is expressed as gain in power as move along z-axis in positive direction, but the Stokes waves experience gain as they propagate in the negative z-direction. Dimensions = [num_modes_EM_Stokes,num_modes_EM_pump,num_modes_AC].

SBS_gain_PE [The SBS gain for only the photoelastic effect.] The comment about negative gain (see SBS_gain above) holds here also. Dimensions = [num_modes_EM_Stokes,num_modes_EM_pump,num_modes_AC].

SBS_gain_MB [The SBS gain for only the moving boundary effect.] The comment about negative gain (see SBS_gain above) holds here also. Dimensions = [num_modes_EM_Stokes,num_modes_EM_pump,num_modes_AC].

alpha : The acoustic loss for each mode. Dimensions = [num_modes_AC].

Return type SBS_gain

integration.**gain_python**(*sim_EM_pump*, *sim_EM_Stokes*, *sim_AC*, *k_AC*, *comsol_data_file*, *comsol_ivals=1*)
Calculate interaction integrals and SBS gain in python. Load in acoustic mode displacement and calculate gain from this also.

integration.**grad_u**(*dx*, *dy*, *u_mat*, *k_AC*)
Take the gradient of field as well as of conjugate of field.

integration.**grid_integral**(*relevant_eps_effs*, *sim_AC_structure*, *sim_AC_Omega_AC*, *n_pts_x*, *n_pts_y*, *dx*, *dy*, *E_mat_p*, *E_mat_S*, *u_mat*, *del_u_mat*, *del_u_mat_star*, *AC_ival*)
Quadrature integration of AC energy density, AC loss (alpha), and PE gain.

integration.**interp_py_fields**(*sim_EM_pump*, *sim_EM_Stokes*, *sim_AC*, *k_AC*, *n_points*, *EM_ival_pump=0*, *EM_ival_Stokes=0*, *AC_ival=0*)
Interpolate fields from FEM mesh to square grid.

integration.**symmetries**(*sim_wguide*, *n_points=10*, *negligible_threshold=1e-05*)
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_spectra**(*sim_AC*, *SBS_gain*, *SBS_gain_PE*, *SBS_gain_MB*, *linewidth_Hz*, *k_AC*, *EM_ival_pump*, *EM_ival_Stokes*, *AC_ival*, *freq_min*, *freq_max*, *num_interp_pts=3000*, *save_fig=True*, *dB=False*, *dB_peak_amp=10*, *mode_comps=False*, *semilogy=False*, *pdf_png='png'*, *save_txt=False*, *prefix_str=''*, *suffix_str=''*)
Construct the SBS gain spectrum, built from Lorentzian peaks of the individual modes.

Parameters

- **sim_AC** – 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.
- **linewidth_Hz** (*array*) – Linewidth of each mode [Hz].
- **k_AC** (*float*) – Acoustic wavevector.
- **EM_ival_pump** (*int or 'All'*) – Which EM pump mode(s) to consider.
- **EM_ival_Stokes** (*int or 'All'*) – Which EM Stokes mode(s) to consider.
- **AC_ival** (*int or 'All'*) – Which AC mode(s) to consider.
- **freq_min** (*float*) – Minimum of frequency range.

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

Keyword Arguments

- **num_interp_pts** (*int*) – Number of frequency points to interpolate to.
- **dB** (*bool*) – Save a set of spectra in dB units.
- **dB_peak_amp** (*float*) – Set the peak amplitude of highest gain mode in dB.
- **mode_comps** (*bool*) – Plot decomposition of spectra into individual modes.
- **semilogy** (*bool*) – PLot y-axis on log scale.
- **save_fig** (*bool*) – Save figure at all.
- **pdf_png** (*str*) – Save figures as 'png' or 'pdf'.
- **save_txt** (*bool*) – Save spectra data to txt file.
- **prefix_str** (*str*) – String to be appended to start of file name.
- **suffix_str** (*str*) – String to be appended to end of file name.

plotting.**plot_msh**(*x_arr*, *prefix_str=''*, *suffix_str=''*)

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*, *ivals=None*, *n_points=500*, *quiver_steps=50*, *xlim_min=None*, *xlim_max=None*, *ylim_min=None*, *ylim_max=None*, *EM_AC='EM_E'*, *num_ticks=None*, *contours=False*, *contour_lst=None*, *stress_fields=False*, *pdf_png='png'*, *prefix_str=''*, *suffix_str=''*)

Plot E or H fields of EM mode, or the AC modes displacement fields.

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

Keyword Arguments

- **ivals** (*list*) – mode numbers of modes you wish to plot
- **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
- **num_ticks** (*int*) – Number of tick marks
- **contours** (*bool*) – Controls contours being overlaid on fields
- **contour_lst** (*list*) – Specify contour values
- **stress_fields** (*bool*) – Calculate acoustic stress fields
- **pdf_png** (*str*) – File type to save, either 'png' or 'pdf'
- **prefix_str** (*str*) – Add a string to start of file name
- **suffix_str** (*str*) – Add a string to end of 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 three 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. The third step involves adding some lines to the NumBAT backend.

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+ry, 0, lc};
Point(18) = {d/2+rx/2+cx, -h+s1+s2+2*cy+ry, 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 your favorite text editor, scroll to the bottom, and change the numbering of the physical surfaces to start at 1, and to increase by one per surface type. Eg. by tradition 1 is the background material, 2 is the waveguide, 3 is the bottom substrate, and 4 is the cladding.

```
Physical Surface(1) = {24};
Physical Surface(2) = {28};
Physical Surface(3) = {30};
Physical Surface(4) = {26};
```

The important thing is to make a note of the chosen labeling! This is best done by taking a screen-shot of the geometry in gmsh, labeling this with material types and physical dimensions, and then adding this file to the NumBAT/docs/msh_type_lib folder.

Step 3

The last step is to add your geometry to the make_mesh function in NumBAT/backend/objects.py.

This involves adding a new elif statement for the inc_shape, in this case 'rib_coated', and then adding lines that define how the final mesh will be created based on the template. This involves giving the mesh a name, specifying the number of element types, and modifying the template geometric parameters. See objects.py for details.

One last thing, if the geometry contains only rectangular shapes, and all elements are therefore linear (rather than curvi-linear), you should also add the inc_shape name to the self.linear_element_shapes list in objects.py. This will ensure that the most efficient semi-analytic integration routines are used. If NumBAT is not told that the mesh is linear it will default to using numerical quadrature.

FEM Errors

There are 2 main errors that can be easily triggered within the Fortran FEM routines. These 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

```
VALPR_64: info_32 != 0 :
VALPR_64: iparam_32(5) =
VALPR_64: number of converged values =
py_calc_modes.f: convergence problem with valpr_64
py_calc_modes.f: You should probably increase resolution of mesh!
py_calc_modes.f: n_conv != nval :
```

Long story short, this indicates that the FEM mesh is too coarse for solutions for higher order Bloch modes (Eigenvaules) to converge. 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