NumBAT Documentation

Release 1.10

Bjorn Sturmberg, Blair Morrison, Mike Smith, Christopher Poulton and Michael Steel

Jun 02, 2021

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 Introduction

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

1.2 Goals

NumBAT is designed primarily to calculate the optical gain response from stimulated Brillouin scattering (SBS) in integrated waveguides. It uses finite element algorithms to solve the electromagnetic and acoustic modes of a wide range of 2D waveguide structures. It can account for photoelastic/electrostriction and moving boundary/radiation pressure effects, as well as uniaxial optical anisotropy and general acoustic anisotropy.

NumBAT also supports user-defined material properties and we hope its creation will drive a community-driven set of standard properties and geometries which will allow all groups to test and validate each other's work.

A full description of the NumBAT physics and numerical algorithms is available in an arxiv paper submitted in September 2017.

NumBAT is open-source software and the authors welcome additions to the code. Details for how to contribute are available in *Contributing to NumBAT*.

1.3 Development team

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

CHAPTER

TWO

INSTALLATION

2.1 Installation

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

NumBAT has been developed on Ubuntu 18.04 with the following package versions: Python 3.6.9, Numpy 1.19.4, Suitesparse 4.4.6, and Gmsh 3.0.6. The integration with Gmsh has a history of version dependence - the first test script runs independently of Gmsh, so if this passes and the second script fails, look into Gmsh version issues. 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 download the git repository and run the setup script

```
$ git clone https://github.com/michaeljsteel/NumBAT.git

$ cd NumBAT/

$ ./setup.sh
```

or, depending on your system configuration as

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

Before doing so you may wish to update your system

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

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

```
$ 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.

For optimal results

```
$ cp NumBAT/backend/NumBATstyle.mplstyle ~/.config/matplotlib/stylelib/
```

or replace plt.style.use('NumBATstyle') in NumBAT/backend/plotting.py with your own prefered matplotlib style file.

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 OSes you may also need to compile a local version of Suitesparse, which is described in the next section.

2.1.1 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 we have used in the past, 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!

2.1.2 Contributing to NumBAT

NumBAT is open source software licensed under the GPL with all source and documentation available at github.com. We welcome additions to NumBAT code, documentation and the materials library. Interested users should fork the standard release from github and make a pull request when ready. For major changes, we strongly suggest contacting the NumBAT team before starting work at `michael.steel@mq.edu.au`.

CHAPTER

THREE

BASIC USAGE

3.1 Simulation Procedure

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 within this folder for future inspection, manipulation, plotting, etc.

Throughout the tutorial the script file will be called simo.py.

These files can be edited using your choice of text editor (for instance running the following in the terminal `$ nano simo.py`) or an IDE (for instance pycharm) which allow you to run and debug code within the IDE.

To start a simulation open a terminal and change into the directory containing the `simo.py` file.

To start we run an example simulation from the tutorials directory. To move to this directory in the terminal enter:

```
$ cd <path to installation>/NumBAT/tutorials
```

To run this script execute:

```
$ python3 simo.py
```

To save the results from the simulation that are displayed upon execution (the print statements in simo.py) use:

```
$ python3 ./simo.py | tee log-simo.log
```

This may require you to update the permissions for the simo.py file to make it executable. This is done in the terminal as:

```
$ chmod +x simo.py
```

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

```
$ python3 -i simo.py
```

This will execute the simo.py script and then return you into an interactive python session within the terminal. This terminal session provides the user experience of an ipython type shell where the python environment and all the simulation objects are as in the simo.py script. 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`.

3.2 Script Structure

As will be seen in the tutorials below, most NumBAT scripts proceed with a standard structure:

- defining materials
- defining waveguide geometries and associating them with material properties
- solving electromagnetic and acoustic modes
- calculating gain and other derived quantities

The following section provides some information about specifying material properties and waveguide structures, as well as the key parameters for controlling the finite-element meshing. Information on how to add new structures to NumBAT is provided in *Making New Mesh*.

3.3 Materials

In order to calculate the modes of a structure we must specify the acoustic and optical properties of all constituent materials.

In NumBAT, this data is read in from json files, which are stored in `<root>/NumBAT/backend/material_data`.

These files not only provide the numerical values for optical and acoustic variables, but record how these variables have been arrived at. Often they are taken from the literature.

The intention of this arrangement is to create a library of materials that can we hope can form a standard amongst the research community. They also allow users to check the sensitivity of their results on particular parameters for a given material.

At present, the material library contains:

- Vacuum
- As2S3_2016_Smith
- As2S3_2017_Morrison
- As2S3_2021_Poulton
- GaAs_2016_Smith
- Si_2013_Laude
- Si_2015_Van_Laer
- Si_2016_Smith
- Si_2021_Poulton
- SiO2_2013_Laude
- SiO2_2015_Van_Laer
- SiO2_2016_Smith
- SiO2_2021_Smith
- Si_test_anisotropic

All available materials are loaded into NumBAT into the materials.materials_dict dictionary, whose keys are the json file names. Materials can easily be added to this by copying any of these files as a template and modifying the properties to suit. The Si_test_anisotropic file contains all the variables that NumBAT is setup to read. We ask that stable parameters (particularly those used for published results) be added to the NumBAT git repository using the same naming convention.

3.4 Waveguide Geometries

The following figures give some examples of how material types and physical dimensions are represented in the mesh geometries. These can also be found in the directory:

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

as a series of `.png` file.

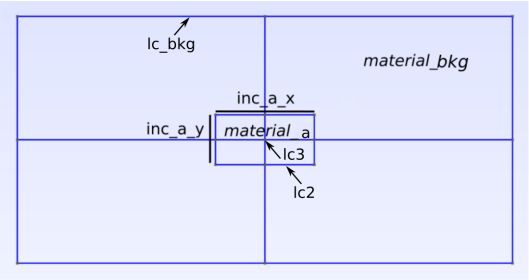


Fig. 3.1: Rectangular waveguide.

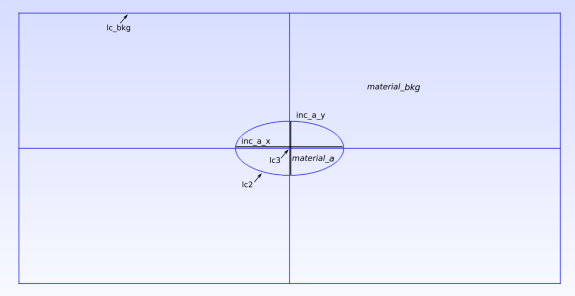


Fig. 3.2: Elliptical waveguide.

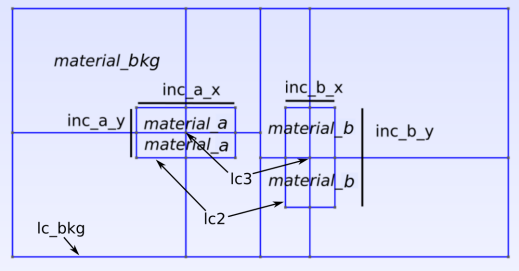


Fig. 3.3: Coupled rectangular waveguides.

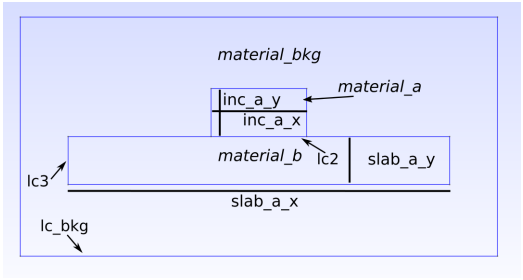


Fig. 3.4: A conventional rib waveguide.

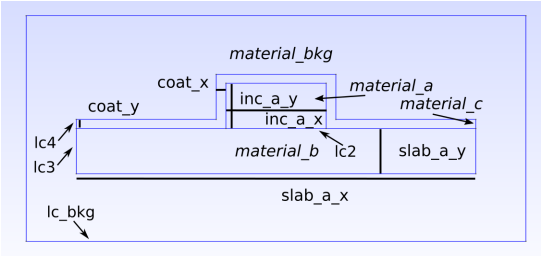


Fig. 3.5: A coated rib waveguide.

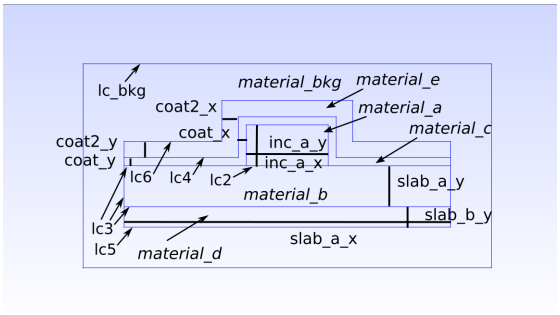


Fig. 3.6: A rib waveguide on two substrates.

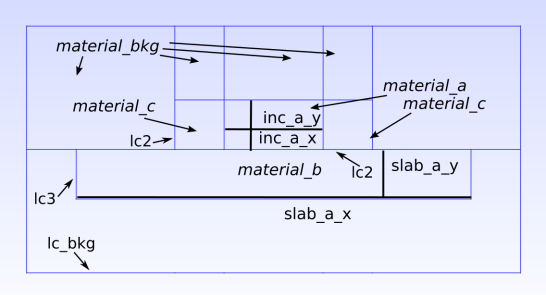


Fig. 3.7: A slot waveguide (`material_a` is low index).

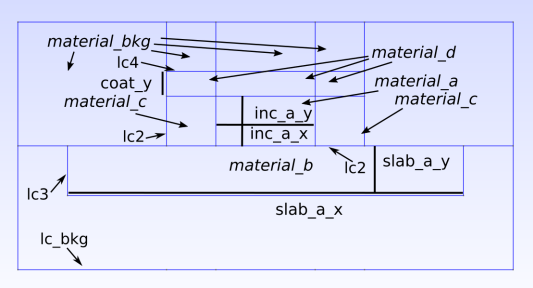


Fig. 3.8: A coated slot waveguide (`material_a` is low index).

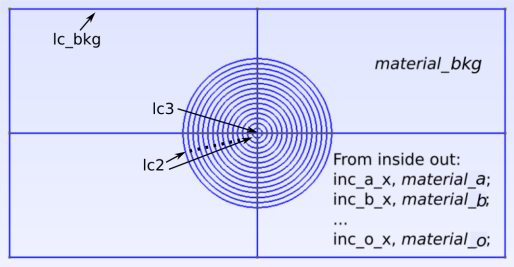


Fig. 3.9: A concentric layered structure.

The parameters `lc_bkg`, `lc_refine_1`, `lc_refine_2` to be encountered below set the fineness of the FEM mesh. `lc_bkg` sets the reference background mesh size, larger `lc_bkg` = larger (more coarse) mesh. In NumBAT it is also possible to refine the mesh near interfaces and near select points in the domain, as highlighted in the figures above. This is done using the `lc_refine_` commands, which we now discuss. At the interface between materials the mesh is refined to be `lc_bkg/lc_refine_1`, therefore larger `lc_refine_1` = 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 is typically also refined at the centers of important regions, such as in the center of a waveguide, which is done with `lc_refine_2`, which analogously to `lc_refine_1`, refines the mesh size at these points as `lc_bkg/lc_refine_2`. For definition of `lc_refine_3+` parameters see the particular .geo file.

Choosing appropriate values of `lc_bkg`, `lc_refine_1`, `lc_refine_2` is crucial for 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 tutorial simulations. 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.

You can also visually check the resolution of your mesh by setting `plt_mesh=True` or `check_mesh=True` when you define your `objects.Struct` - the first saves a png of the mesh (in NumBAT/backend/fortran/msh/) the second opens mesh in gmsh - (see Tutorial 1). The NumBAT generated .msh file is stored in NumBAT/backend/fortran/msh/ which can be viewed by running the following command

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

Users on WSL will need to first run an X listener (such as XMING) in Windows in order for the "plt_mesh=True" feature to work. Once the X listener is running, execute the following in the terminal:

```
$ sudo apt-get install x11-apps
$ export DISPLAY=:0
$ xclock
```

where the last command is simply to check the setup. Once this is confirmed to be operating smoothly, the "plt_mesh=True" command will then run as anticipated and generate two png files (one for the geometry and one for the mesh) in NumBAT/backend/fortran/msh/. Note the X windows that open must be manually closed for the calculation to continue, and after unexpected restarts the X window may no longer display output but the png files will contain the necessary features.

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.

3.5 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.

3.5.1 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
```

3.5.2 Terminating NumBAT simulations

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
```

CHAPTER

FOUR

EXAMPLES AND TUTORIALS

This chapter provides several resources for learning NumBAT, exploring its applications and validating it against literature results. You should begin by working through the sequence of tutorial exercises which are largely based on literature results. You may then select from examples drawn from a recent tutorial paper by Dr Mike Smith and colleagues, and a range of other literature studies.

4.1 Tutorial

In this section we walk through a number of simple simulations that demonstrate the basic use of NumBAT. *Literature Examples* looks at a number of literature examples taken from many of the well-known groups in this field. The full Python interface is documented in *Python Interface*.

4.1.1 Tutorial 1 – 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 (annotated in the source code below as Step 1, Step 2, etc) 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 electromagnetic mode profiles, propagation constants, and potentially other data which can be accessed through various methods.
6. Display the propagation constants in units of m^{-1} of the EM modes using `mode_calcs.kz_EM_all()`
7. Obtain the effective index of the fundamental mode using `mode_calcs.neff()`
8. Identify the desired acoustic wavenumber from the difference of the pump and Stokes propagation constants and solve the acoustic problem. `mode_calcs.calc_AC_modes()` returns an object containing the acoustic mode profiles, frequencies and potentially other data at the propagation constant `k_AC`.
9. Display the acoustic frequencies in Hz using `mode_calcs.nu_AC_all()`.
10. Calculate the total SBS gain, contributions from photoelasticity and moving boundary effects, and the acoustic loss using `integration.gain_and_qs()`.

```
""" Calculate the backward SBS gain for modes in a
    silicon waveguide surrounded in air.
"""

# Step 1

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()
print('\n\nCommencing NumBAT tutorial 1')

# Step 2
# 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 = 300
inc_a_y = 280
# Shape of the waveguide.
inc_shape = 'rectangular'

# Step 3
# 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'

# Step 4
# Use specified parameters to create a waveguide object.
# Note use of rough mesh for demonstration purposes, and use plt_mesh=True
# to save the geometry and mesh as png files in backend/fortran/msh/
```

```
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2016_Smith"),
                        lc_bkg=1, # in vacuum background
                        lc_refine_1=600.0, # on cylinder surfaces
                        lc_refine_2=300.0, # on cylinder center
                        plt_mesh=False)

# Explicitly remind ourselves what data we're using.
print('\nUsing material data: ', wguide.material_a)

# Step 5
# Estimate 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)

# Display the wavevectors of EM modes.
v_kz=sim_EM_pump.kz_EM_all()
print('\n k_z of EM modes [1/m]:')
for (i, kz) in enumerate(v_kz): print('{0:3d}  {1:.4e}'.format(i, np.real(kz)))

# 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)

# Step 6
# Find the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_pump.neff(0))
print("\n Fundamental optical mode ")
print(" n_eff = ", np.round(n_eff_sim, 4))

# Acoustic wavevector
k_AC = np.real(sim_EM_pump.kz_EM(0) - sim_EM_Stokes.kz_EM(0))

print('\n AC wavenumber (1/m) = ', np.round(k_AC, 4))

# Step 7
# 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.
v_nu=sim_AC.nu_AC_all()
print('\n Freq of AC modes (GHz):')
for (i, nu) in enumerate(v_nu): print('{0:3d}  {1:.4e}'.format(i, np.real(nu)*1e-9))

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

# Step 8
# 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 [1/(Wm)] PE contribution \n", SBS_gain_PE[EM_ival_pump,EM_ival_
↪Stokes,:])
print("SBS_gain [1/(Wm)] MB contribution \n", SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,
↪:])
print("SBS_gain [1/(Wm)] 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 Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
print("SBS_gain linewidth [Hz] \n", linewidth_Hz)

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

4.1.2 Tutorial 2 – SBS Gain Spectra

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

Here are some elements to note:

1. `np.savez()` and `np.load()` allow storage of arbitrary data in numpy `.npz` files between simulations to accelerate subsequent calculations. Use the flag `recalc_fields` to determine whether to recalculate the data from scratch or load data from an existing `.npz` file. The data is recovered as a :Simmo: object by calling the `tolist()` method. Note that numpy requires the `allow_pickle=True` flag for loading array data from file.
2. This tutorial and many subsequent ones can be made to run faster at the expense of accuracy by appending the argument `fast=1` to the command line. This has the effect of specifying a coarser FEM grid. In this case, the output data and fields directory begin with `ftut_02` rather than `tut_02`.
3. Both electric and magnetic fields can be selected using `EM_E` or `EM_H` as the value of `EM_AC` in `plotting.mode_fields`. These fields are stored in a folder `tut_02-fields/` within the tutorial folder.
4. By default, plots are exported as `png` format. Pass the option `pdf_png=pdf` to plot functions to generate a `pdf` output.
5. Plots of both spectra and modes are generated with a best attempt at font sizes, line widths etc, but the range of potential cases make it impossible to find a selection that works in all cases. Most plot functions therefore support the passing of a `plotting.Decorator` object that can vary the settings of these parameters and also pass additional commands to write on the plot axes. See the plotting API for details. This should be regarded as a relatively advanced NumBAT feature.
6. The `suppress_imimre` option suppresses plotting of the Im$[x]$, Im$[y]$ and Re$[z]$ components of the fields which in a lossless non-leaky problem should normally be zero at all points and therefore not useful to plot.
7. Vector field plots often require tweaking to get an attractive set of vector arrows. The `quiver_points` option controls the number of arrows drawn along each direction.
8. The plot functions and the `Decorator` class support many options. Consult the API chapter for details on how to fine tune your plots.

```
""" 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

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 = 300
inc_a_y = 280
inc_shape = 'rectangular'

num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 25
EM_ival_pump = 0
EM_ival_Stokes = 0
AC_ival = 'All'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  prefix_str = 'ftut_02-'
  refine_fac=1
else:
  prefix_str = 'tut_02-'
  refine_fac=5

print('\n\nCommencing NumBAT tutorial 2')

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2016_Smith"),
                        lc_bkg=1, lc_refine_1=120.0*refine_fac, lc_refine_2=60.
↪0*refine_fac)


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

recalc_fields=True     # run the calculation from scratch
recalc_fields=False   # reuse saved fields from previous calculation

if recalc_fields:
  # 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)
  sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
  np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
else:
  # Once npz files have been saved from one simulation run,
  # set recalc_fields=True to use the saved data
  #This provides precisely the same objects  for the remainder of the simulation.
  npzfile = np.load('wguide_data.npz', allow_pickle=True)
```

```
  sim_EM_pump = npzfile['sim_EM_pump'].tolist()
  npzfile = np.load('wguide_data2.npz', allow_pickle=True)
  sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

# Print the wavevectors of EM modes.
v_kz=sim_EM_pump.kz_EM_all()
print('\n k_z of EM modes [1/m]:')
for (i, kz) in enumerate(v_kz): print('{0:3d}  {1:.4e}'.format(i, np.real(kz)))

# 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,
# which specify the fraction of the axis to remove from the plot.
# For instance xlim_min=0.4 will remove 40% of the x axis from the left outer edge
# to the center. xlim_max=0.4 will remove 40% from the right outer edge towards the␣
↪center.
# This leaves just the inner 20% of the unit cell displayed in the plot.
# The ylim variables perform the equivalent actions on the y axis.

# Let's plot fields for only the fundamental (ival = 0) mode.
#decorator=plotting.Decorator()
#decorator.set_multiplot_axes_property('subplots_wspace',.4)

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ylim_min=0.4,
                          ylim_max=0.4, ivals=[EM_ival_pump], contours=True, EM_AC='EM_
↪E',
                          prefix_str=prefix_str, ticks=True, suppress_imimre=True)

#Repeat this plot in pdf output format
plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ylim_min=0.4,
                          ylim_max=0.4, ivals=[EM_ival_pump], contours=True, EM_AC='EM_
↪E',
                          pdf_png='pdf', prefix_str=prefix_str, ticks=True, suppress_
↪imimre=True)

# Plot the H fields of the EM modes - specified with EM_AC='EM_H'.
plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ylim_min=0.4,
                          ylim_max=0.4, ivals=[EM_ival_pump], EM_AC='EM_H',
                          prefix_str=prefix_str, ticks=True, suppress_imimre=True)

# Calculate the EM effective index of the waveguide.
n_eff_sim = np.real(sim_EM_pump.neff(0))
print("n_eff", np.round(n_eff_sim, 4))

# Acoustic wavevector
k_AC = np.real(sim_EM_pump.kz_EM(0) - sim_EM_Stokes.kz_EM(0))

if recalc_fields:
  # 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)
else:
  npzfile = np.load('wguide_data_AC.npz', allow_pickle=True)
  sim_AC = npzfile['sim_AC'].tolist()

# Print the frequencies of AC modes.
v_nu=sim_AC.nu_AC_all()
print('\n Freq of AC modes (GHz):')
```

```
for (i, nu) in enumerate(v_nu): print('{0:3d}  {1:.4e}'.format(i, np.real(nu)*1e-9))

# Plot the AC modes fields, important to specify this with EM_AC='AC'.
# The AC modes are calculated on a subset of the full unitcell,
# which excludes vacuum regions, so there is usually no need to restrict the area␣
↪plotted
# with xlim_min, xlim_max etc.
plotting.plot_mode_fields(sim_AC, EM_AC='AC', contours=False, prefix_str=prefix_str,
    ticks=True, suppress_imimre=True, quiver_points=20, ivals=[0])

if recalc_fields:
  # 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)
else:
  npzfile = np.load('wguide_data_AC_gain.npz', allow_pickle=True)
  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.
print("\n Displaying results with negligible components masked out")
# 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("SBS_gain [1/(Wm)] 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 [1/(Wm)] 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 [1/(Wm)] 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)
```

```
# Repeat this plot focusing on one frequency range
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))
```

The following figures show a selection of electromagnetic and acoustic mode profiles produced in this example.

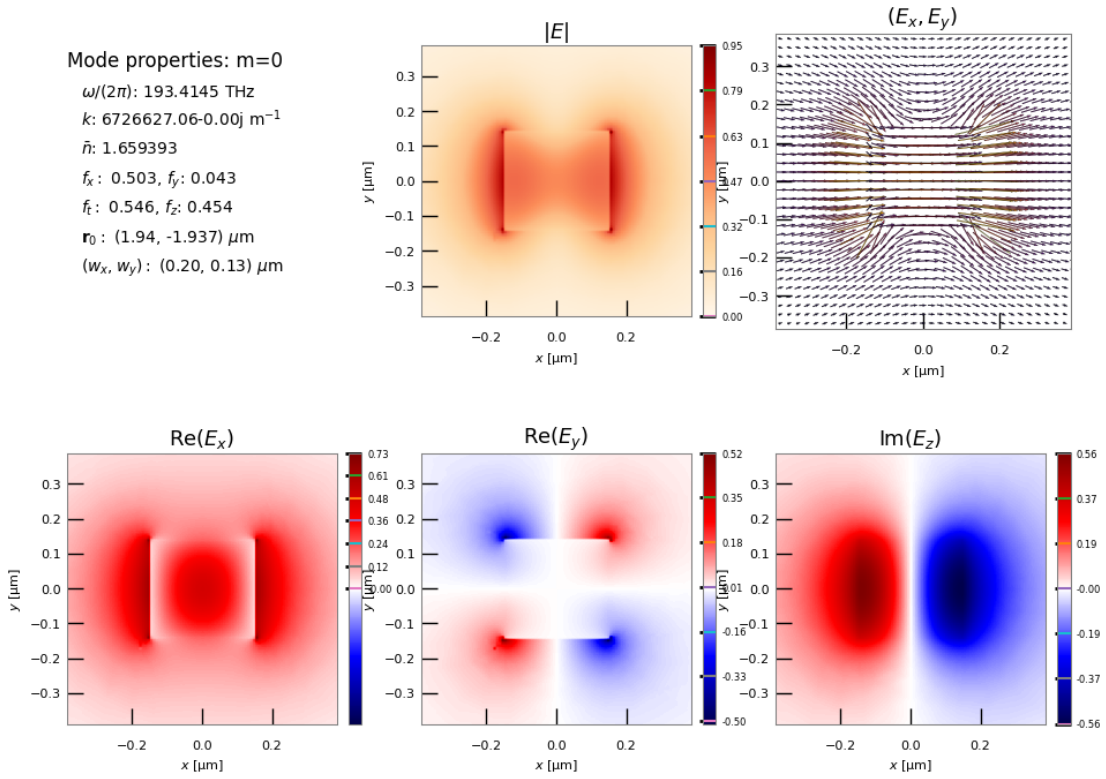


Fig. 4.1: Fundamental optical mode fields.

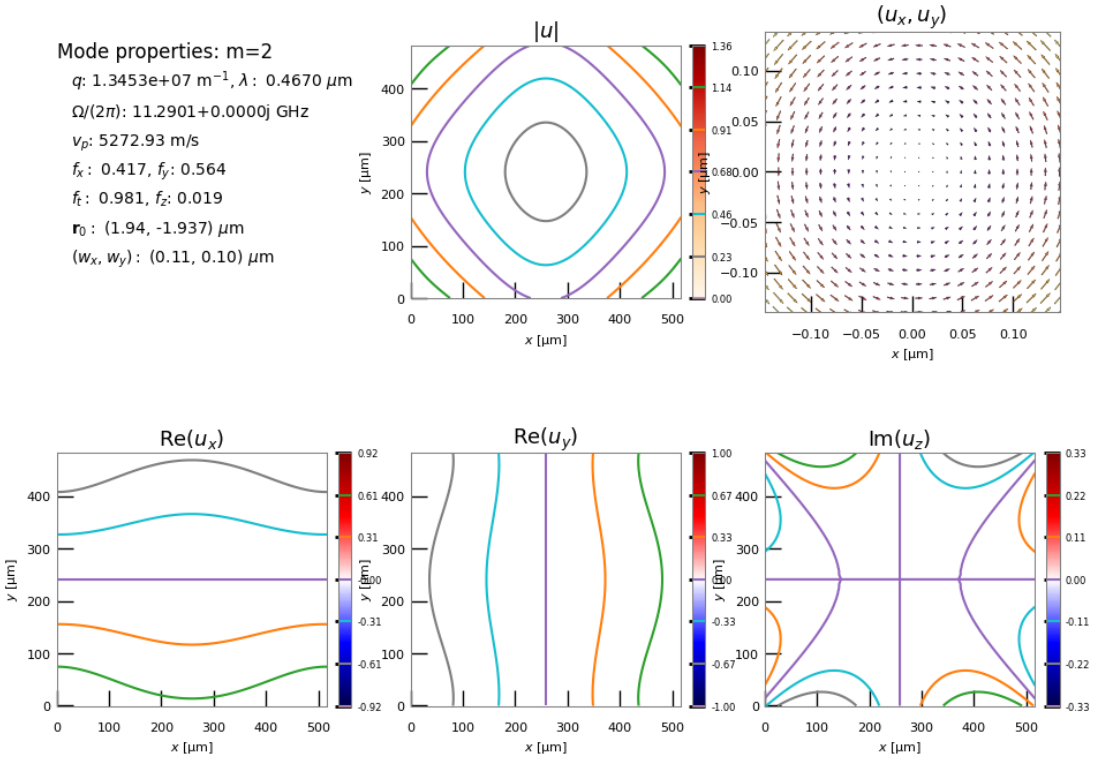


Fig. 4.2: Acoustic mode with high gain due to moving boundary effect.

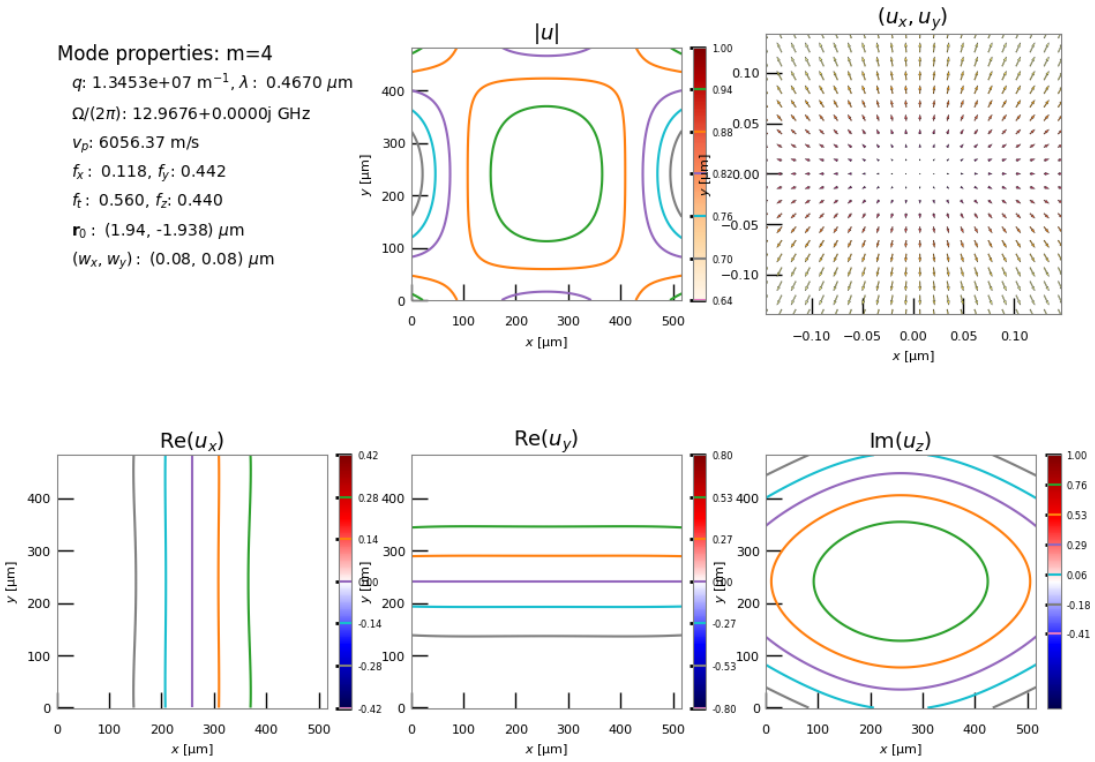


Fig. 4.3: Acoustic mode with high gain due to moving boundary effect.

This example also generates gain spectra.

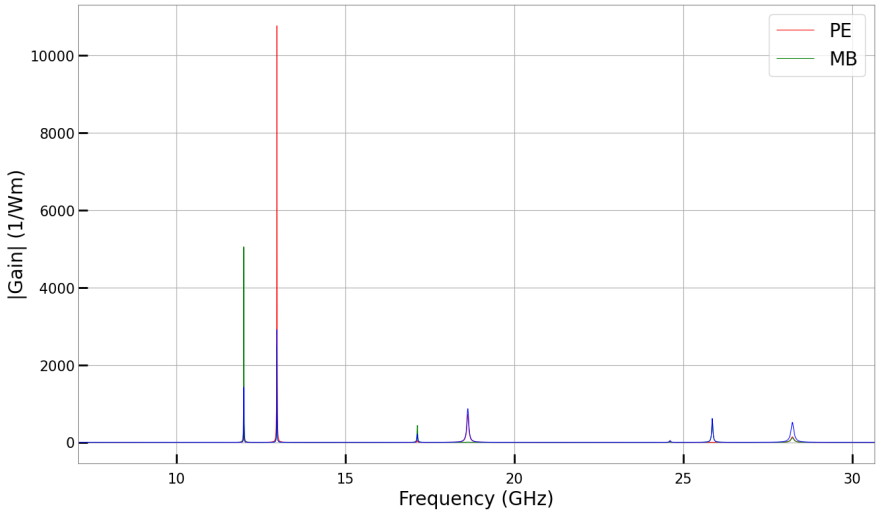


Fig. 4.4: Gain spectra showing gain due to the photoelastic effect, gain due to moving boundary effect, and the total gain.

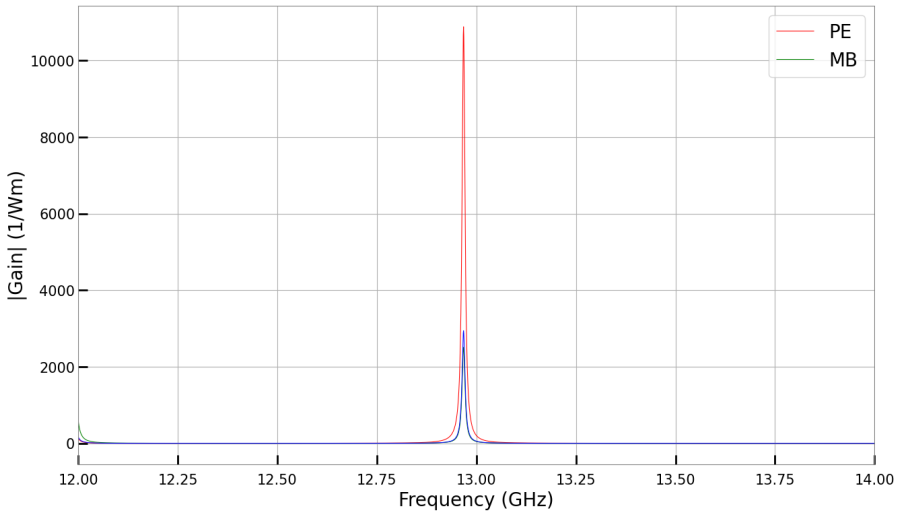


Fig. 4.5: Zoomed-in gain spectra from *Gain spectra showing gain due to the photoelastic effect, gain due to moving boundary effect, and the total gain.*.

4.1.3 Tutorial 3a – Investigating Dispersion and np.save/np.load

This example, contained in `tutorials/simo-tut_03_1-dispersion-npload.py` calculates the acoustic dispersion diagram for the problem in the previous tutorial and classifies the modes according to the point group symmetry class.

```
""" 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.
"""

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()
print('\n\nCommencing NumBAT tutorial 3a')

# Geometric Parameters - all in nm.
wl_nm = 1550
unitcell_x = 2.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 300
inc_a_y = 280
inc_shape = 'rectangular'
# Choose modes to include.
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 25
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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2016_Smith"),
                        lc_bkg=1, lc_refine_1=600.0, lc_refine_2=300.0)

# 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)
```

```
# sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
# np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)

# 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', allow_pickle=True)
sim_EM_pump = npzfile['sim_EM_pump'].tolist()
npzfile = np.load('wguide_data2.npz', allow_pickle=True)
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.kz_EM(0) - sim_EM_Stokes.kz_EM(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.nu_AC_all() 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_B1, = 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_B2, = 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_B3, = plt.plot(np.real(q_ac/k_AC), Om, '^g')

    print("Wavevector loop", i_ac+1, "/", nu_ks)
ax.set_ylim(0,25)
ax.set_xlim(0,1)
plt.legend([sym_A, sym_B1, sym_B2, sym_B3],['A',r'B$_1$',r'B$_2$',r'B$_3$'], 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))
```

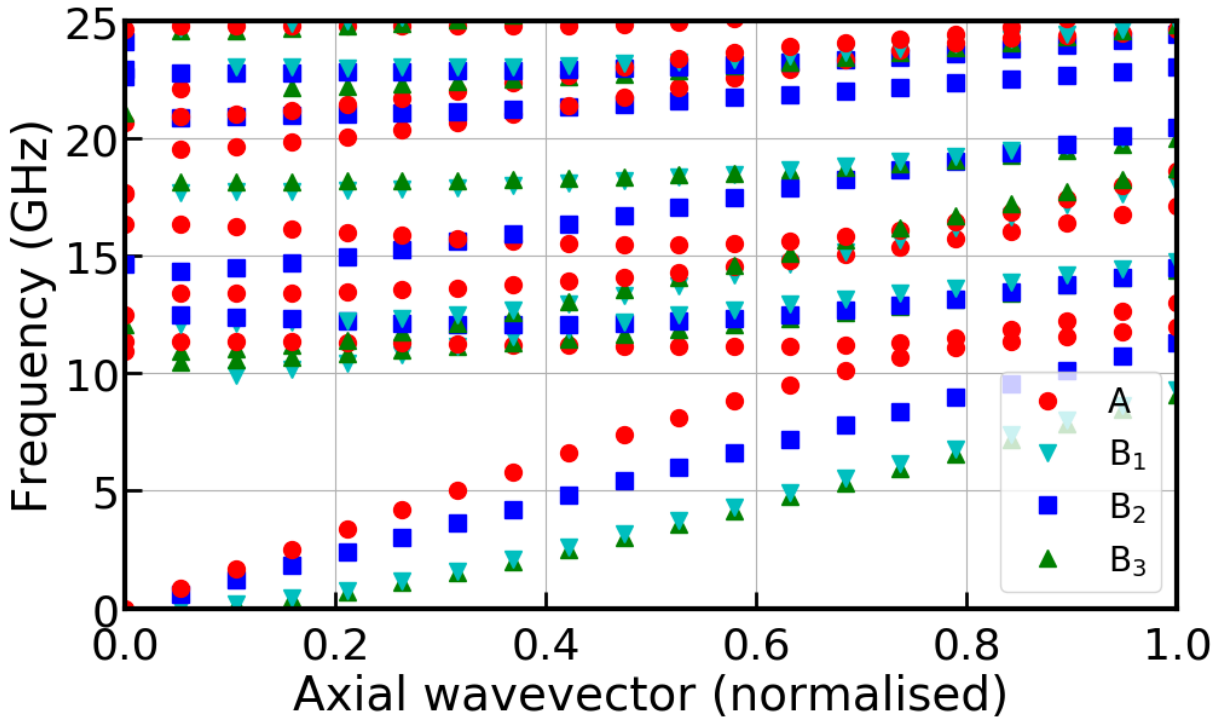


Fig. 4.6: Acoustic dispersion diagram with modes categorised by symmetry as in Table 1 of "Formal selection rules for Brillouin scattering in integrated waveguides and structured fibers" by C. Wolff, M. J. Steel, and C. G. Poulton https://doi.org/10.1364/OE.22.032489

4.1.4 Tutorial 3b – Investigating Dispersion and multiprocessing

This tutorial, contained in `tutorials/simo-tut_03_2-dispersion-multicore.py` continues the study of acoustic dispersion and demonstrates the use of Python multiprocessor calls to increase speed.

```
""" 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
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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  prefix_str = 'ftut_03-2-'
  refine_fac=1
  print('\n\nCommencing NumBAT tutorial 3b - fast mode')
else:
  prefix_str = 'tut_03-2-'
  refine_fac=5
  print('\n\nCommencing NumBAT tutorial 3b')

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2016_Smith"),
                        lc_bkg=1, lc_refine_1=120.0*refine_fac, lc_refine_2=60.
↪0*refine_fac)

# 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
    # Clear memory
    sim_AC = None

    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)
# Note pool.map() doesn't pass errors back from fortran routines very well.
# It's good practise to run the extrema of your simulation range through map()
# before launcing full multicore simulation.

# 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))
```

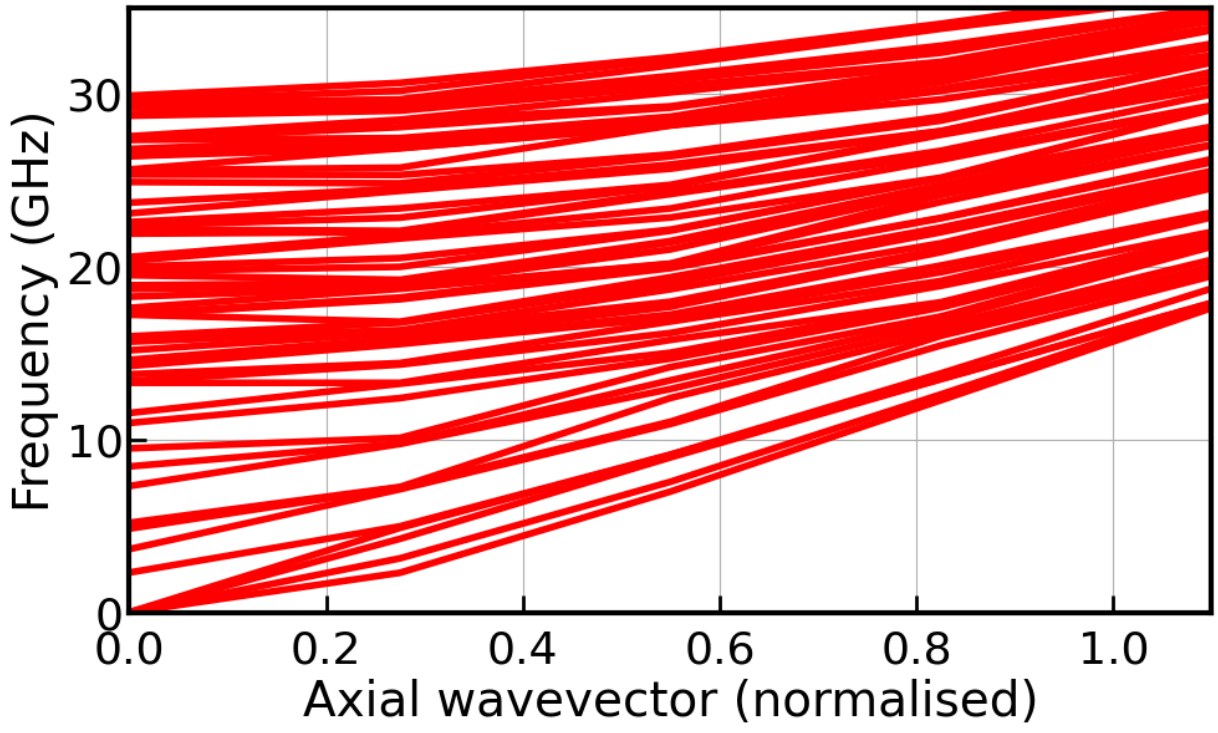


Fig. 4.7: Acoustic dispersion diagram ploted as lines.

4.1.5 Tutorial 4 – Parameter Scan of Widths

This tutorial, contained in `tutorials/simo-tut_04_scan_widths.py` demonstrates use of a parameter scan, in this case of the width of the silicon rectangular waveguide, to understand the behaviour of the Brillouin gain.

```
""" 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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  prefix_str = 'ftut_04-'
  refine_fac=1
  print('\n\nCommencing NumBAT tutorial 4 - fast mode')
else:
  prefix_str = 'tut_04-'
  refine_fac=5
```

```
  print('\n\nCommencing NumBAT tutorial 4')

# 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)
    ## Clear memory
    #sim_EM_pump = sim_EM_Stokes = sim_AC = None

    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.get_material("Vacuum"),
                            material_a=materials.get_material("Si_2016_Smith"),
                            lc_bkg=1, lc_refine_1=120.0*refine_fac, lc_refine_2=60.
↪0*refine_fac)
    geo_objects_list.append(wguide)


new_calcs=True # fixme
if new_calcs:
  # Run widths in parallel across num_cores CPUs using multiprocessing package.
  pool = Pool(num_cores)

  # Note pool.map() doesn't pass errors back from fortran routines very well.
  # It's good practise to run the extrema of your simulation range through map()
  # before launching full multicore simulation.
```

```
  width_objs = pool.map(modes_n_gain, geo_objects_list)
  np.savez('Simo_results', width_objs=width_objs)  # This generates a warning abut␣
↪ragged nested sequences. Is there an option to pool.map that would clean this up?

else:
  npzfile = np.load('Simo_results.npz', allow_pickle=True)
  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)

    print(sim_AC)
    # 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 = 50000
    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))
```

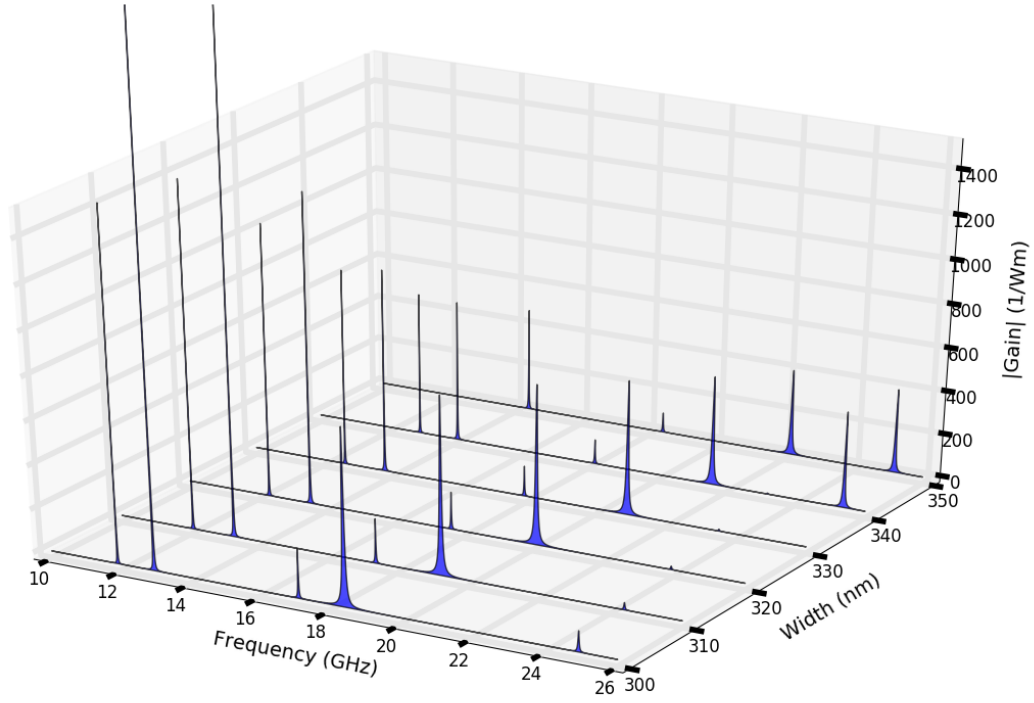


Fig. 4.8: Gain spectra as function of waveguide width.

4.1.6 Tutorial 5 – Convergence Study

This tutorial, contained in `tutorials/simo-tut_05_convergence_study.py` demonstrates a scan of numerical parameters for our standard silicon-in-air problem to test the convergence of the calculation results.

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

import time
import datetime
import numpy as np
import sys
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 = 300
inc_a_y = 280
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-'
print('\n\nCommencing NumBAT tutorial 5')

# Warning: The fine grids in this list will take considerable time to run!
lc_list = [20,100,500,1000,1500,2000,2500]
nu_lcs = len(lc_list)
lc_bkg_list = 1*np.ones(nu_lcs)
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)
    lc_refine_2 = lc_ref/2
```

```
    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.get_material("Vacuum"),
                            material_a=materials.get_material("Si_2016_Smith"),
                            lc_bkg=lc_bkg, lc_refine_1=lc_ref, lc_refine_2=lc_refine_
↪2, 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 = [3,4,8,10]
# 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)
```

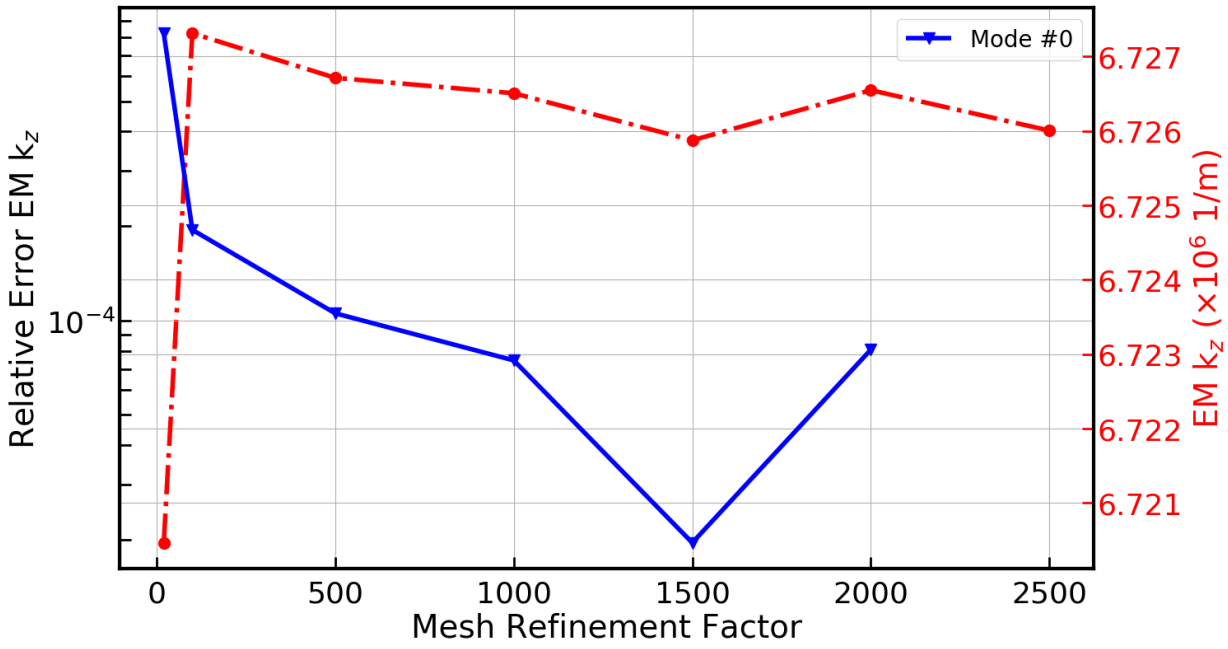


Fig. 4.9: Convergence of optical mode frequencies.

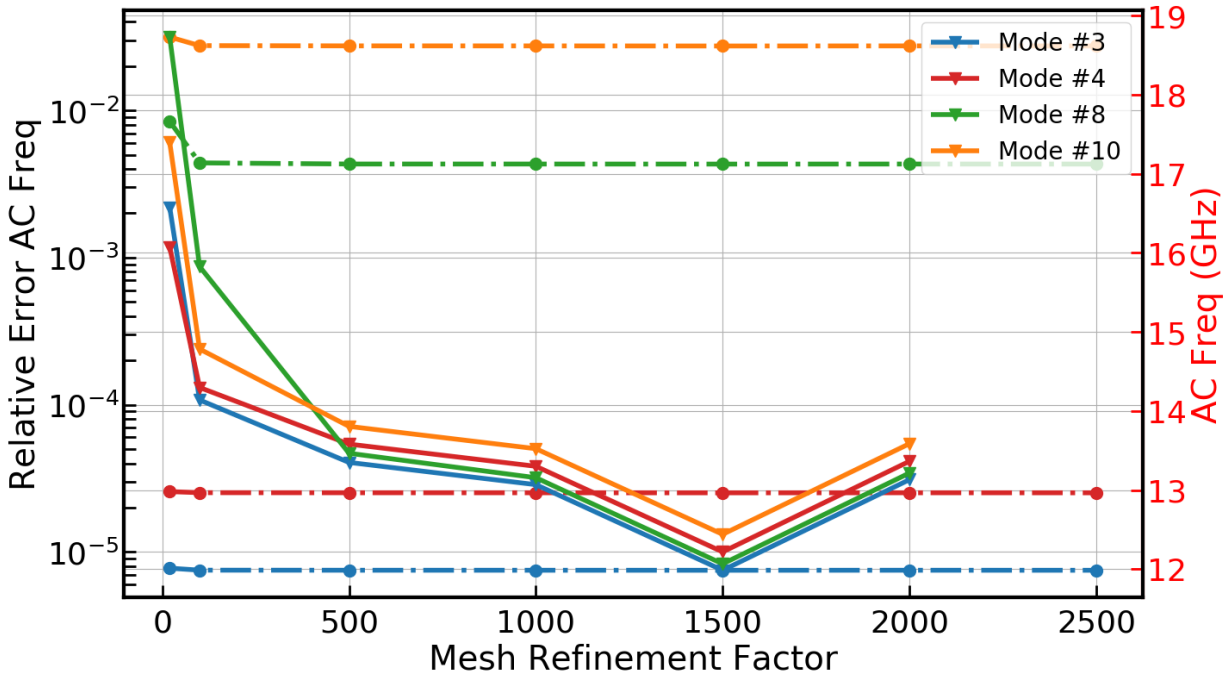


Fig. 4.10: Convergence of acoustic mode frequencies.

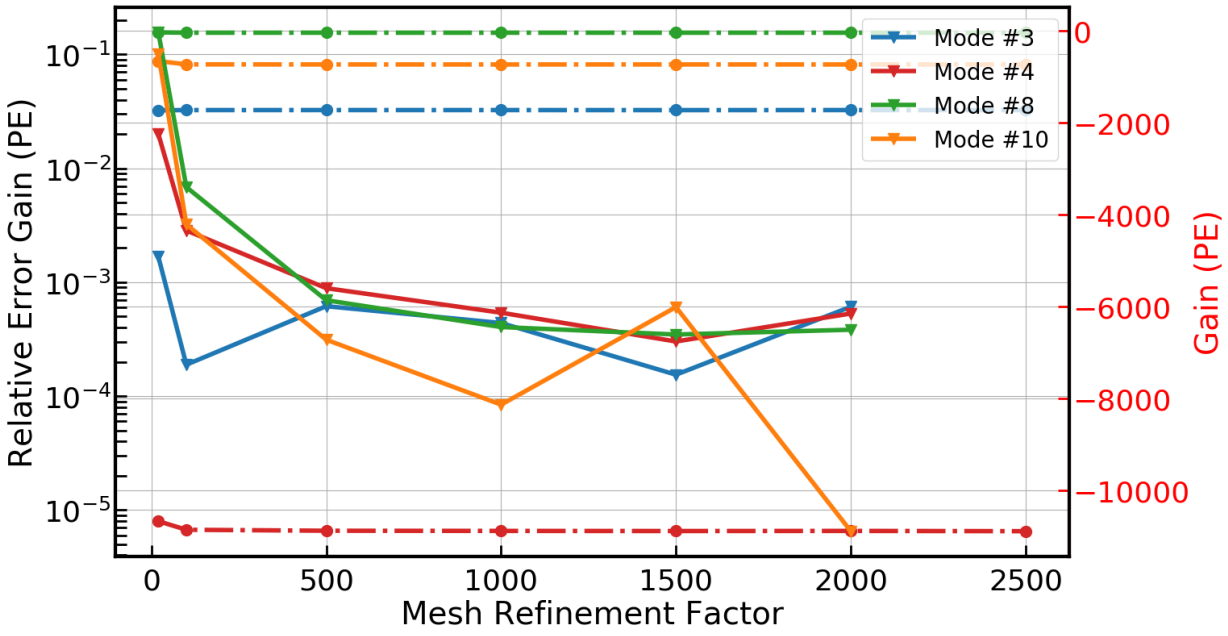


Fig. 4.11: Convergence of photoelastic gain.

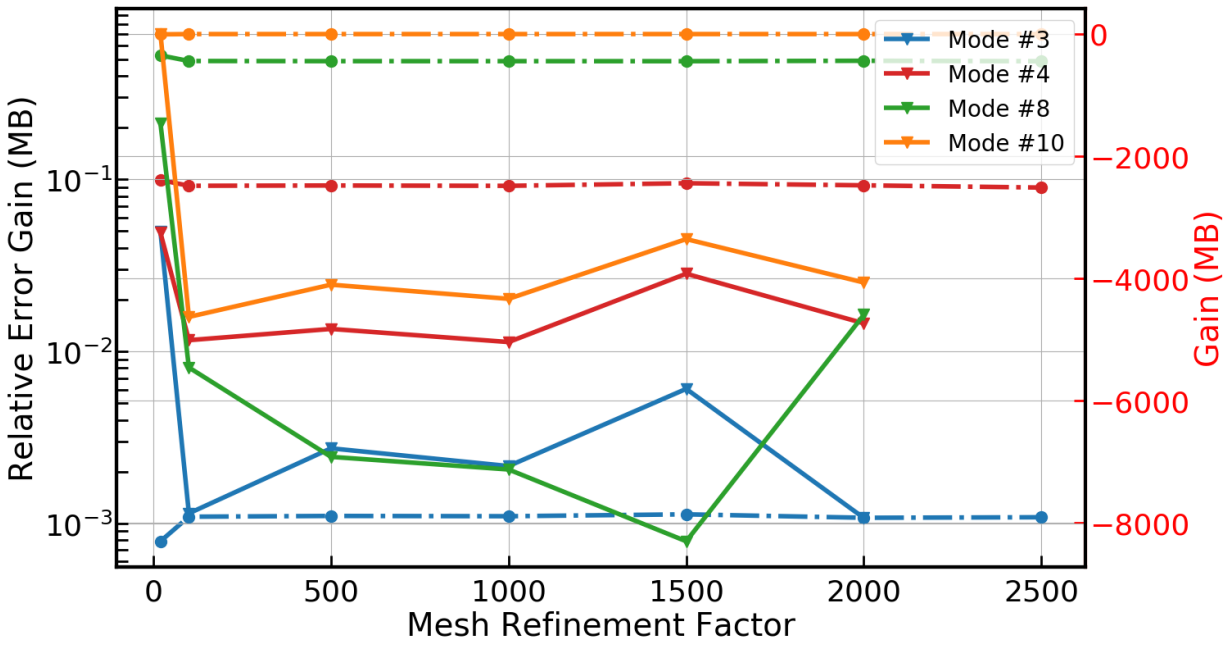


Fig. 4.12: Convergence of moving boundary gain.

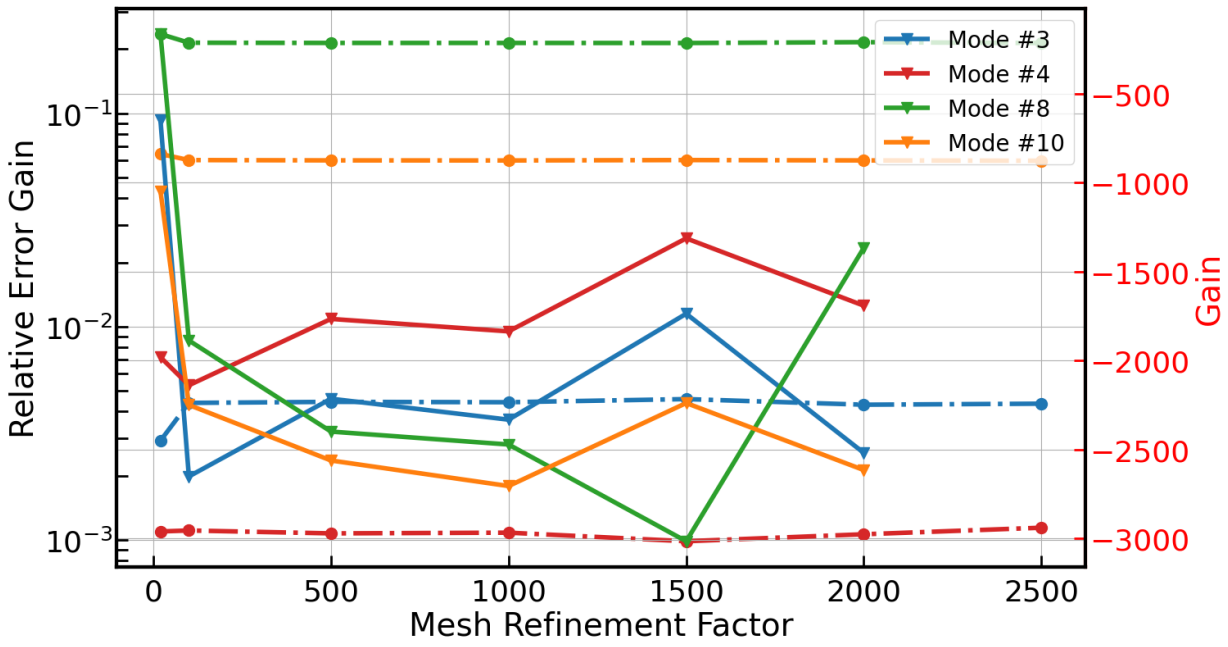


Fig. 4.13: Convergence of total gain.

4.1.7 Tutorial 6 – Silica Nanowire

In this tutorial, contained in `tutorials/simo-tut_06_silica_nanowire.py` we start to explore some different structures, in this case a silica nanowire surrounded by vacuum.

```
""" 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
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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  prefix_str = 'ftut_06-'
  refine_fac=1
  print('\n\nCommencing NumBAT tutorial 6 - fast mode')
else:
  prefix_str = 'tut_06-'
  refine_fac=5
  print('\n\nCommencing NumBAT tutorial 6')
```

```
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.get_material("Vacuum"),
                        material_a=materials.get_maerial("SiO2_2016_Smith"),
                        lc_bkg=1, lc_refine_1=120.0*refine_fac, lc_refine_2=40.
↪0*refine_fac)

# 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', allow_pickle=True)
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()
# plotting.plot_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.plot_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', allow_pickle=True)
# 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', allow_pickle=True)
# sim_AC = npzfile['sim_AC'].tolist()
# plotting.plot_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))
```

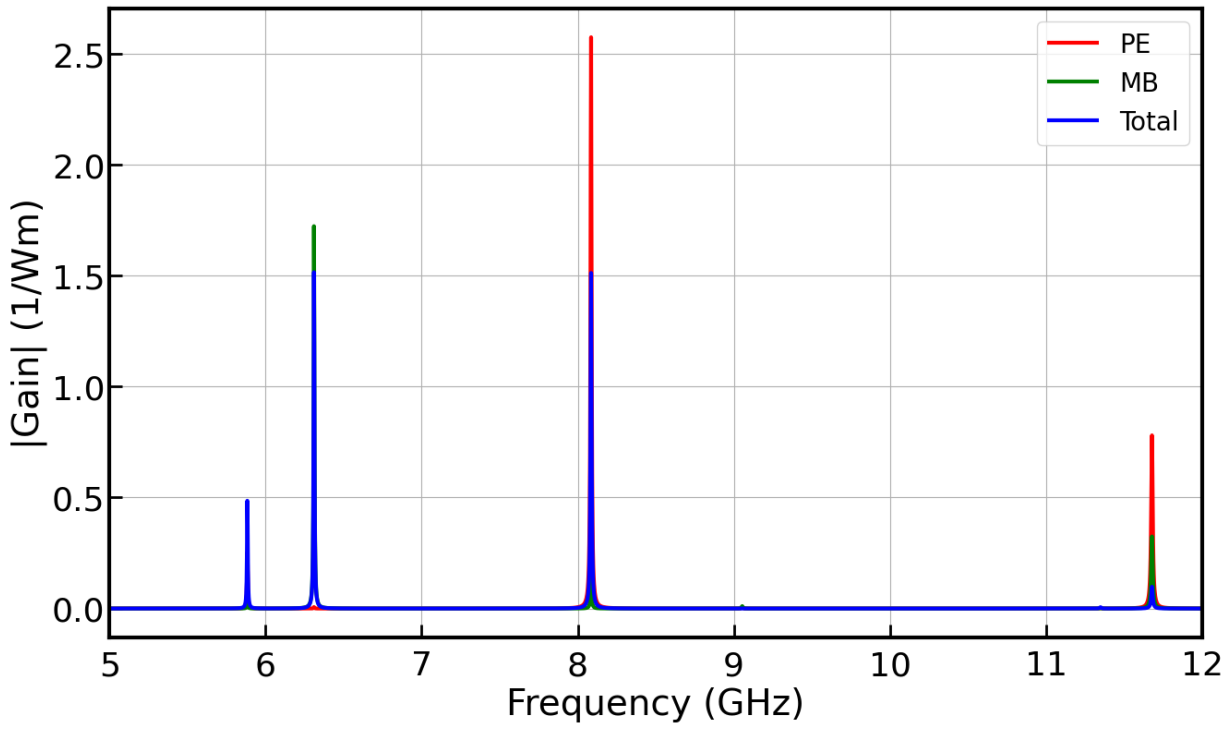


Fig. 4.14: Gain spectra showing gain due to photoelastic effect, gain due to moving boundary effect, and total gain.

4.1.8 Tutorial 7 – Slot Waveguide

This tutorial, contained in, `tutorials/simo-tut_07-slot.py` examines backward SBS in a more complex structure: chalcogenide soft glass (As_2S_3) embedded in a silicon slot waveguide on a silica slab.

```
""" 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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  prefix_str = 'ftut_07-'
  refine_fac=1
  print('\n\nCommencing NumBAT tutorial 7 - fast mode')
else:
  prefix_str = 'tut_07-'
  refine_fac=4
  print('\n\nCommencing NumBAT tutorial 7')
```

```
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.get_material("Vacuum"),             #␣
↪background
                        material_a=materials.get_material("As2S3_2017_Morrison"), #␣
↪slot
                        material_b=materials.get_material("SiO2_2013_Laude"),      #␣
↪slab
                        material_c=materials.get_material("Si_2016_Smith"),        #␣
↪walls of slot
                        lc_bkg=1, lc_refine_1=200.0*refine_fac, lc_refine_2=100.
↪0*refine_fac, plt_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=n_eff)
# np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz', allow_pickle=True)
# 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', allow_pickle=True)
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

# plotting.plot_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', allow_pickle=True)
# sim_AC = npzfile['sim_AC'].tolist()

# plotting.plot_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', allow_pickle=True)
# 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))
```

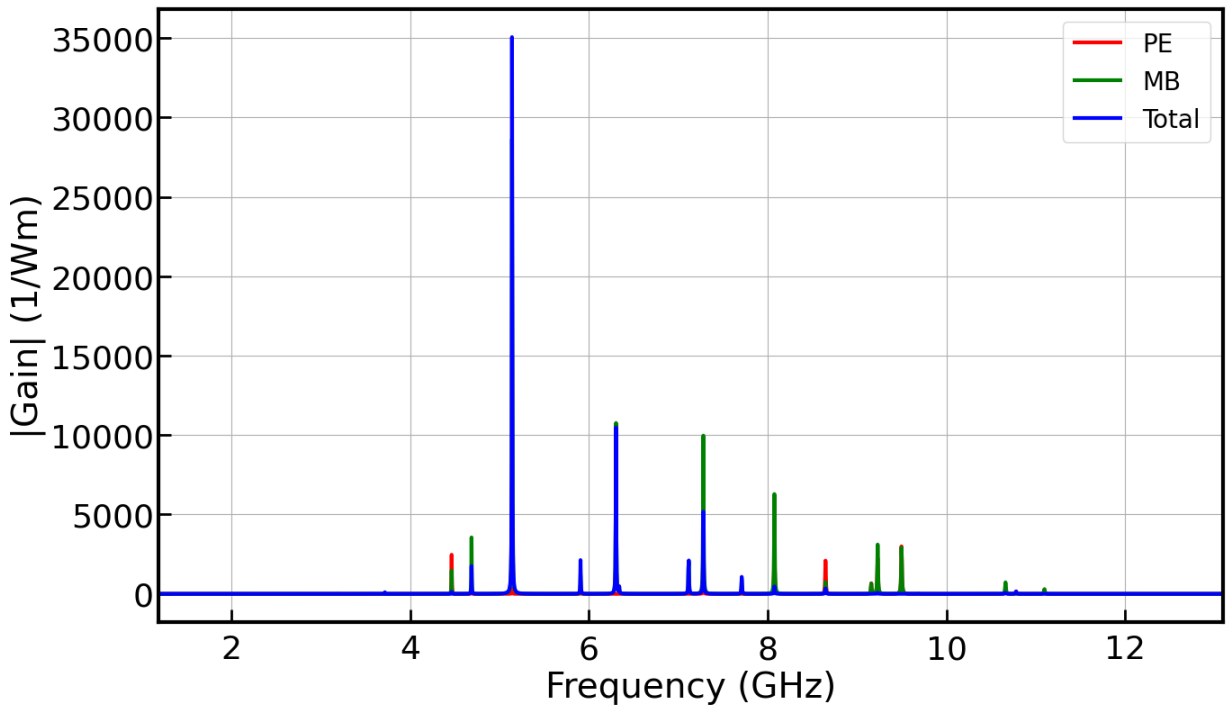


Fig. 4.15: Gain spectra showing gain due to photoelastic effect, gain due to moving boundary effect, and total gain.

4.1.9 Tutorial 8 – Slot Waveguide Scan Covering

This tutorial, contained in, `tutorials/simo-tut_08-slot_coated-scan.py` continues the study of the previous slot waveguide, by examining the thickness dependence of a silica capping layer.

```
""" 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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  prefix_str = 'ftut_08-'
  refine_fac=1
  print('\n\nCommencing NumBAT tutorial 8 - fast mode')
else:
  prefix_str = 'tut_08-'
```

```
  refine_fac=4
  print('\n\nCommencing NumBAT tutorial 8')

# 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.get_material("Vacuum"),
↪# background
                            material_a=materials.get_material("As2S3_2017_Morrison"),
↪# slot
                            material_b=materials.get_material("SiO2_2013_Laude"),
↪# slab
                            material_c=materials.get_material("Si_2016_Smith"),
↪# walls of slot
                            material_d=materials.get_material("SiO2_2013_Laude"),
↪# coating
                            lc_bkg=1, lc_refine_1=100.0*refine_fac, lc_refine_2=50.
↪0*refine_fac)

    # 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.plot_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
    # Clear memory
    wguide = sim_EM_pump = sim_EM_Stokes = sim_AC = None
    SBS_gain = SBS_gain_PE = SBS_gain_MB = linewidth_Hz = Q_factors = alpha = None

    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)
# Note pool.map() doesn't pass errors back from fortran routines very well.
# It's good practise to run the extrema of your simulation range through map()
# before launching full multicore simulation.

# 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] = np.real(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))
```

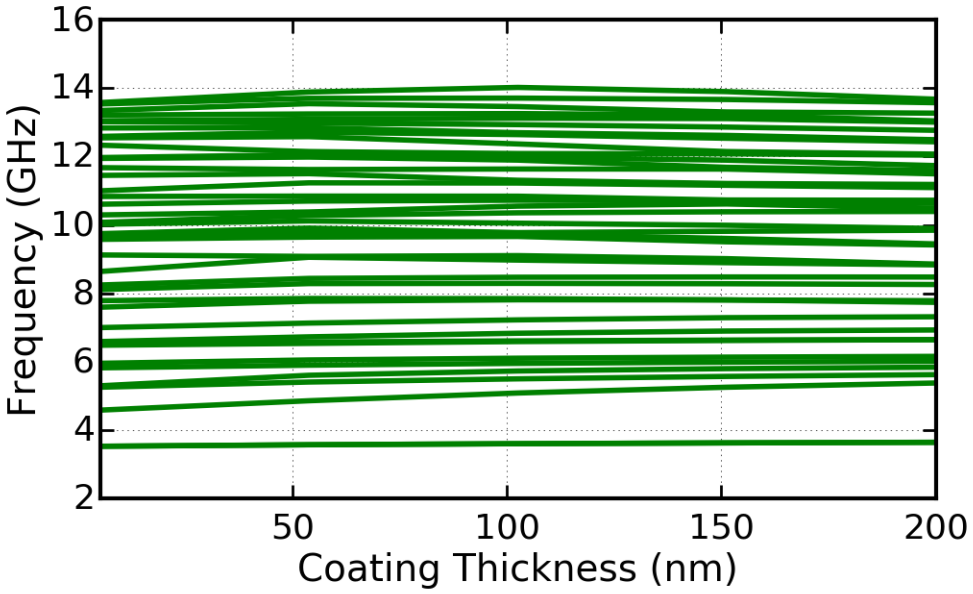


Fig. 4.16: Acoustic frequencies as function of covering layer thickness.

4.1.10 Tutorial 9 – Anisotropic Elastic Materials

This tutorial, contained in, `tutorials/simo-tut_09-anisotropy.py` improves the treatment of the silicon rectangular waveguide by accounting for the anisotropic elastic properties of silicon (simply by referencing a different material file for silicon).

```
""" 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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  print('\n\nCommencing NumBAT tutorial 9 - fast mode')
  prefix_str = 'ftut_09-'
  refine_fac=1
else:
  print('\n\nCommencing NumBAT tutorial 9')
  prefix_str = 'tut_09-'
  refine_fac=5

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_test_anisotropic"),
                        lc_bkg=1, lc_refine_1=200.0*refine_fac, lc_refine_2=1.
↪0*refine_fac)
```

```
# 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 [1/(Wm)] PE contribution \n", SBS_gain_PE[EM_ival_pump,EM_ival_
↪Stokes,:])
print("SBS_gain [1/(Wm)] MB contribution \n", SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,
↪:])
print("SBS_gain [1/(Wm)] 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 Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)

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

4.1.11 Tutorial 10 – Multilayered 'Onion'

This tutorial, contained in, `tutorials/simo-tut_10-onion.py` shows how to create multi-layered circular structures.

```
""" Example showing how the 'onion' geometry template can be used
    to simulate a circular Si waveguide clad in SiO2.
"""

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 = 3.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 800
inc_b_x = 500
inc_shape = 'onion'

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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  print('\n\nCommencing NumBAT tutorial 10 - fast mode')
  prefix_str = 'ftut_10-'
  refine_fac=1
else:
  print('\n\nCommencing NumBAT tutorial 10')
  prefix_str = 'tut_10-'
  refine_fac=5

# Use of a more refined mesh to produce field plots.
wguide = objects.Struct(unitcell_x,inc_a_x,inc_shape=inc_shape,
                        inc_b_x=inc_b_x,
                        unitcell_y=unitcell_y,
                        material_bkg=materials.get_material("Vacuum"),
```

```
                        material_a=materials.get_material("Si_2016_Smith"),
                        material_b=materials.get_material("SiO2_2016_Smith")
                        lc_bkg=1, lc_refine_1=20.0*refine_fac, lc_refine_2=1.0*refine_
↪fac, plt_mesh=True)


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

new_calcs=True

# Calculate Electromagnetic modes.
if new_calcs:
  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)

  sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
  np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
else:
  npzfile = np.load('wguide_data.npz', allow_pickle=True)
  sim_EM_pump = npzfile['sim_EM_pump'].tolist()
  npzfile = np.load('wguide_data2.npz', allow_pickle=True)
  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[0]*((wl_nm*1e-9)/(2.*np.pi)))
print("n_eff", np.round(n_eff_sim, 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.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ylim_min=0.3,
                        ylim_max=0.3, ivals=[EM_ival_pump], contours=True, EM_AC='EM_
↪E',
                        prefix_str=prefix_str, ticks=True, quiver_points=20, comps=[
↪'Et'])

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ylim_min=0.3,
                        ylim_max=0.3, ivals=[EM_ival_pump], contours=True, EM_AC='EM_
↪H',
                        prefix_str=prefix_str, ticks=True, quiver_points=20, comps=[
↪'Ht'])

# 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.
if new_calcs:
  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)
else:
  npzfile = np.load('wguide_data_AC.npz', allow_pickle=True)
  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.plot_mode_fields(sim_AC, EM_AC='AC', pdf_png='png', contours=False,
                          prefix_str=prefix_str, ticks=True, ivals=[0], quiver_
↪points=20)

# 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)

# 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))
```

4.1.12 Tutorial 11 – Two-layered 'Onion'

This tutorial, contained in, `tutorials/simo-tut_11a-onion2.py` demonstrates use of the two layered onion structure which generates a more efficient mesh than the full onion template.

```
""" Example showing how the 'onion' geometry template can be used
    to simulate a circular Si waveguide clad in SiO2.
"""

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 = 3.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 800
inc_b_x = 500
inc_shape = 'onion2'

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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  print('\n\nCommencing NumBAT tutorial 11a - fast mode')
  prefix_str = 'ftut_11a-'
  refine_fac=1
else:
  print('\n\nCommencing NumBAT tutorial 11a')
  prefix_str = 'tut_11a-'
  refine_fac=5

# Use of a more refined mesh to produce field plots.
wguide = objects.Struct(unitcell_x,inc_a_x,inc_shape=inc_shape,
                        inc_b_x=inc_b_x,
                        unitcell_y=unitcell_y,
                        material_bkg=materials.get_material("Vacuum"),
```

```
                        material_a=materials.get_material("Si_2016_Smith"),
                        material_b=materials.get_material("SiO2_2016_Smith"),
                        lc_bkg=.25, lc_refine_1=5.0*refine_fac, lc_refine_2=5*refine_
↪fac, plt_mesh=True)


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

new_calcs=True

# Calculate Electromagnetic modes.
if new_calcs:
  sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff, calc_EM_mode_
↪energy=True)
  np.savez('wguide_data', sim_EM_pump=sim_EM_pump)

  sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
  np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
else:
  npzfile = np.load('wguide_data.npz', allow_pickle=True)
  sim_EM_pump = npzfile['sim_EM_pump'].tolist()
  npzfile = np.load('wguide_data2.npz', allow_pickle=True)
  sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

print('EM modes:\n')
kz_EM_mu =np.real(sim_EM_pump.kz_EM_all())*1e-6
neff_EM =sim_EM_pump.neff_all()
ng_EM =sim_EM_pump.ngroup_all()
print('m    |   k_z [1/micron]   | neff   | ng')
for m in range(num_modes_EM_pump):
  print('{0:4d}  {1:12.6f}  {2:12.6f}  {3:12.6f}'.format(m, kz_EM_mu[m], neff_EM[m],␣
↪ng_EM[m]))

# 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))

# # 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.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ylim_min=0.3,
                        ylim_max=0.3, ivals=[0], contours=True, EM_AC='EM_E',
                        prefix_str=prefix_str, ticks=True, quiver_points=20, comps=[
↪'Et'])

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ylim_min=0.3,
                        ylim_max=0.3, ivals=[0], contours=True, EM_AC='EM_H',
                        prefix_str=prefix_str, ticks=True, quiver_points=20, comps=[
↪'Ht'])

# 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.
if new_calcs:
  sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, calc_AC_mode_
↪power=True)
```

```
  np.savez('wguide_data_AC', sim_AC=sim_AC)
else:
  npzfile = np.load('wguide_data_AC.npz', allow_pickle=True)
  sim_AC = npzfile['sim_AC'].tolist()

print('AC mode properties (GHz) \n')
nu_AC = np.real(sim_AC.nu_AC_all())*1e-9
vp_AC = np.real(sim_AC.vp_AC_all())
vg_AC = np.real(sim_AC.vg_AC_all())
print('m    |   nu [GHz]   | vp [m/s]   | vg [m/s]')
for m in range(num_modes_AC):
  print('{0:4d}  {1:12.6f}  {2:12.2f}  {3:12.2f}'.format(m, nu_AC[m], vp_AC[m], vg_
↪AC[m]))

sim_AC.calc_acoustic_losses()

plotting.plot_mode_fields(sim_AC, EM_AC='AC', pdf_png='png', contours=False,
                          prefix_str=prefix_str, ticks=True, ivals=[0], quiver_
↪points=20)

# 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)

# 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))
```

4.1.13 Tutorial 12 – SMF-28 fibre

This tutorial, contained in, `tutorials/simo-tut_12-smf28.py` models backward SBS in the standard SMF28 fibre using the onion2 template.

```
""" Example showing how the 'onion' geometry template can be used
    to simulate a circular Si waveguide clad in SiO2.
"""

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 = 3.5*wl_nm
unitcell_y = unitcell_x
inc_a_x = 800
inc_b_x = 500
inc_shape = 'onion2'

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'

if len(sys.argv)>1 and sys.argv[1]=='fast=1':  # choose between faster or more␣
↪accurate calculation
  print('\n\nCommencing NumBAT tutorial 11a - fast mode')
  prefix_str = 'ftut_11a-'
  refine_fac=1
else:
  print('\n\nCommencing NumBAT tutorial 11a')
  prefix_str = 'tut_11a-'
  refine_fac=5

# Use of a more refined mesh to produce field plots.
wguide = objects.Struct(unitcell_x,inc_a_x,inc_shape=inc_shape,
                        inc_b_x=inc_b_x,
                        unitcell_y=unitcell_y,
                        material_bkg=materials.get_material("Vacuum"),
```

```
                        material_a=materials.get_material("Si_2016_Smith"),
                        material_b=materials.get_material("SiO2_2016_Smith"),
                        lc_bkg=.25, lc_refine_1=5.0*refine_fac, lc_refine_2=5*refine_
↪fac, plt_mesh=True)


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

new_calcs=True

# Calculate Electromagnetic modes.
if new_calcs:
  sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff, calc_EM_mode_
↪energy=True)
  np.savez('wguide_data', sim_EM_pump=sim_EM_pump)

  sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)
  np.savez('wguide_data2', sim_EM_Stokes=sim_EM_Stokes)
else:
  npzfile = np.load('wguide_data.npz', allow_pickle=True)
  sim_EM_pump = npzfile['sim_EM_pump'].tolist()
  npzfile = np.load('wguide_data2.npz', allow_pickle=True)
  sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

print('EM modes:\n')
kz_EM_mu =np.real(sim_EM_pump.kz_EM_all())*1e-6
neff_EM =sim_EM_pump.neff_all()
ng_EM =sim_EM_pump.ngroup_all()
print('m    |   k_z [1/micron]   | neff   | ng')
for m in range(num_modes_EM_pump):
  print('{0:4d}  {1:12.6f}  {2:12.6f}  {3:12.6f}'.format(m, kz_EM_mu[m], neff_EM[m],␣
↪ng_EM[m]))

# 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))

# # 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.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ylim_min=0.3,
                        ylim_max=0.3, ivals=[0], contours=True, EM_AC='EM_E',
                        prefix_str=prefix_str, ticks=True, quiver_points=20, comps=[
↪'Et'])

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ylim_min=0.3,
                        ylim_max=0.3, ivals=[0], contours=True, EM_AC='EM_H',
                        prefix_str=prefix_str, ticks=True, quiver_points=20, comps=[
↪'Ht'])

# 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.
if new_calcs:
  sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, calc_AC_mode_
↪power=True)
```

```
  np.savez('wguide_data_AC', sim_AC=sim_AC)
else:
  npzfile = np.load('wguide_data_AC.npz', allow_pickle=True)
  sim_AC = npzfile['sim_AC'].tolist()

print('AC mode properties (GHz) \n')
nu_AC = np.real(sim_AC.nu_AC_all())*1e-9
vp_AC = np.real(sim_AC.vp_AC_all())
vg_AC = np.real(sim_AC.vg_AC_all())
print('m    |   nu [GHz]   | vp [m/s]   | vg [m/s]')
for m in range(num_modes_AC):
  print('{0:4d}  {1:12.6f}  {2:12.2f}  {3:12.2f}'.format(m, nu_AC[m], vp_AC[m], vg_
↪AC[m]))

sim_AC.calc_acoustic_losses()

plotting.plot_mode_fields(sim_AC, EM_AC='AC', pdf_png='png', contours=False,
                          prefix_str=prefix_str, ticks=True, ivals=[0], quiver_
↪points=20)

# 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)

# 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))
```

4.2 JOSA B Tutorial

Mike Smith et al. have used NumBAT throughout their 2021 SBS tutorial paper, published in J. Opt. Soc. Am. B. .. (see .. M. Smith et al, FIXME .. Generation of phonons from electrostriction in small-core optical waveguides .., *JOSA B* **3**, 042109 (2021). ..) This tutorial works through backward, forward, and intermodal forward SBS. The simulation scripts and resultant mode fields are shown below.

4.2.1 BSBS - Circular Waveguide - Silica

```
print("\n Simulation time (sec.)", (end - start))
"""
Script to evaluate backward Brillouin scattering in a cylindrical SiO2 waveguide
"""

# Import the necessary packages
import time
import datetime
import numpy as np
import sys
import math
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()

# Specify Geometric Parameters - all in [nm].
wl_nm = 1550 # Wavelength of EM wave in vacuum.
# Unit cell dimensions must be sufficiently large to ensure fields are zero at␣
↪outermost boundary.
unitcell_x = 4.01*wl_nm # be careful to ensure not whole integer multiples
unitcell_y = unitcell_x
inc_a_x = 1000 # Waveguide widths.
inc_a_y = inc_a_x
inc_shape = 'circular' # Shape of the waveguide.

# Specify number of electromagnetic modes and acoustic modes involved in the
# calculation for BSBS
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100
# The EM pump mode number for which to calculate interaction with AC modes. Typically␣
↪0 for BSBS.
EM_ival_pump = 1
# The EM Stokes mode number for which to calculate interaction with AC modes.␣
↪Typically 0 for BSBS.
EM_ival_Stokes = EM_ival_pump
```

```
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Output files are generated in a folder with the following prefix
prefix_str = 'bsbs-josab-1umSiO2'

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("SiO2_2021_Poulton"),
                        lc_bkg=0.05,  # mesh coarseness in background, larger lc_bkg␣
↪= coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger lc2 = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False)   # note requires x-windows configuration␣
↪to work

# Print information on material data in terminal
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)

# Initial guess for the EM effective index of the waveguide
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))

# A computation interruption if needed
# sys.exit("We interrupt your regularly scheduled computation to bring you ... for now
↪")

# Calculate the Electromagnetic modes of the Stokes field.
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

print("Plotting EM fields ")

plotting.plot_mode_fields(sim_EM_pump,
                          ivals=[EM_ival_pump],
                          EM_AC='EM_E', num_ticks=3,xlim_min=0.2, xlim_max=0.2, ylim_
↪min=0.2, ylim_max=0.2,
                          prefix_str=prefix_str, pdf_png='png', ticks=True, quiver_
↪points=10,
                          comps=['Et','Eabs'], n_points=1000, colorbar=True)

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

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

# Calculate total SBS gain, photoelastic and moving boundary contributions, as
# well as other important quantities
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)

# 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)

# Display these in terminal
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
#determining the location of the maximum gain
maxGainloc=np.argmax(abs(masked.data)) ;

print("Plotting acoustic mode corresponding to maximum")
plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,␣
↪ivals=[maxGainloc],
                          num_ticks=3, quiver_points=40, pdf_png='png',ticks=True,␣
↪comps=['ut','uabs'], colorbar=True)

# Displaying results for the maximum found in the selection
print("-----------------")
print("Displaying results for maximum gain value found:")
print("Greatest SBS_gain  [1/(Wm)] total \n", masked.data[maxGainloc])
print("displaying corresponding acoustic mode number (i.e., AC_field_#) for reference␣
↪\n",maxGainloc )
print("EM Pump Power [Watts] \n", sim_EM_pump.EM_mode_power[EM_ival_pump] )
print("EM Stokes Power [Watts] \n", sim_EM_Stokes.EM_mode_power[EM_ival_Stokes] )
print("EM angular frequency [THz] \n", sim_EM_pump.omega_EM/1e12 )
print("AC Energy Density [J*m^{-1}] \n", sim_AC.AC_mode_energy_elastic[maxGainloc] )
print("AC loss alpha [1/s] \n", alpha[maxGainloc] )
print("AC frequency [GHz] \n", sim_AC.Omega_AC[maxGainloc]/(1e9*2*math.pi) )
print("AC linewidth [MHz] \n", linewidth_Hz[maxGainloc]/1e6)

#since the overlap is not returned directly we'll have to deduce it
absQtot2 = (alpha[maxGainloc]*sim_EM_pump.EM_mode_power[EM_ival_pump]*sim_EM_Stokes.
↪EM_mode_power[EM_ival_Stokes]*sim_AC.AC_mode_energy_elastic[maxGainloc]*masked.
↪data[maxGainloc])/(2*sim_EM_pump.omega_EM*sim_AC.Omega_AC[maxGainloc]);
absQtot = pow(absQtot2,1/2)
```

```
print("Total coupling |Qtot| [W*m^{-1}*s] \n", absQtot )

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

Mode #1 $k_z = 4768812.257219 - 0.000000i$ $n_{eff} = 1.176419 - 0.000000i$

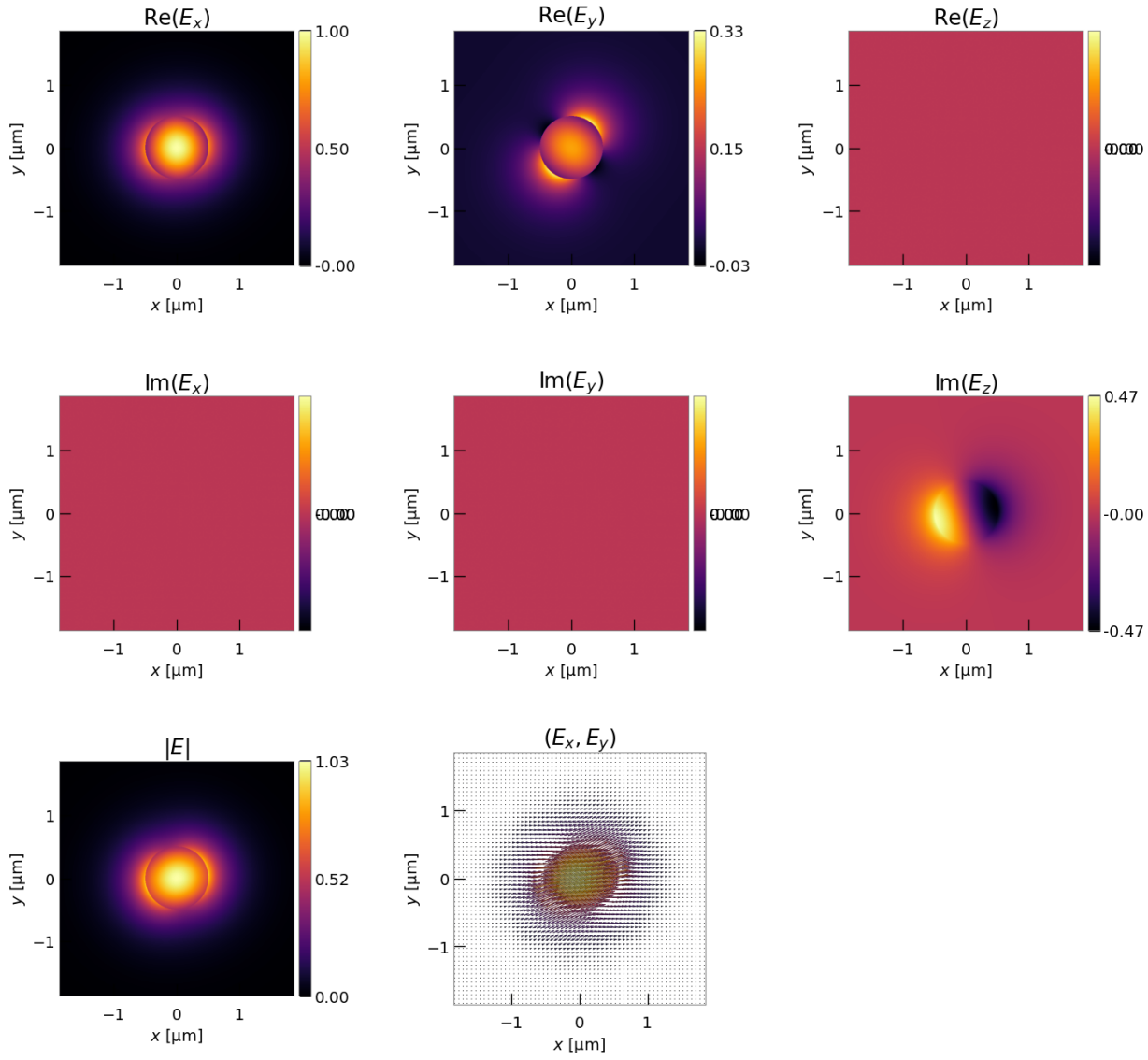


Fig. 4.17: Fundamental optical mode fields.

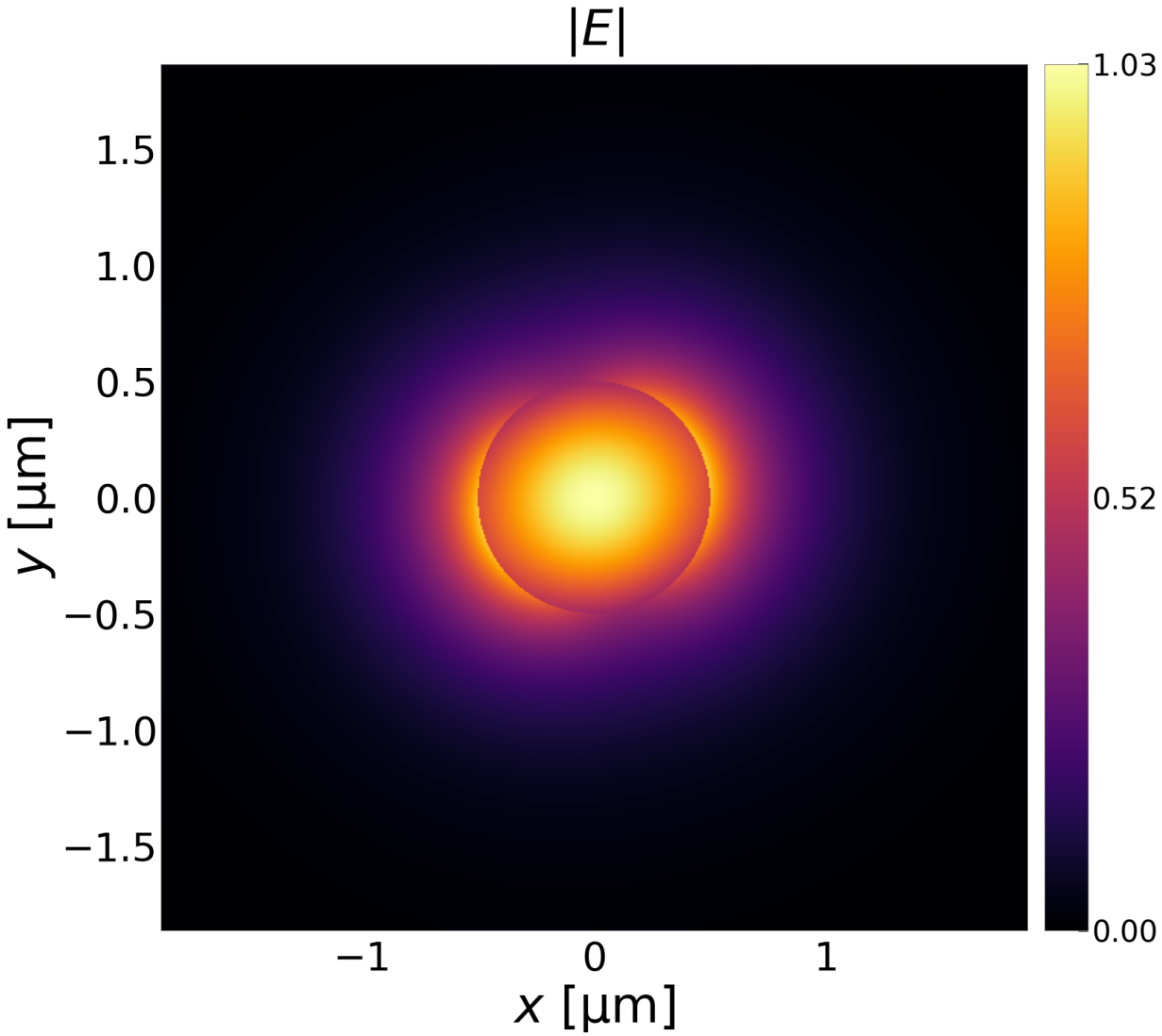


Fig. 4.18: Fundamental optical mode fields.

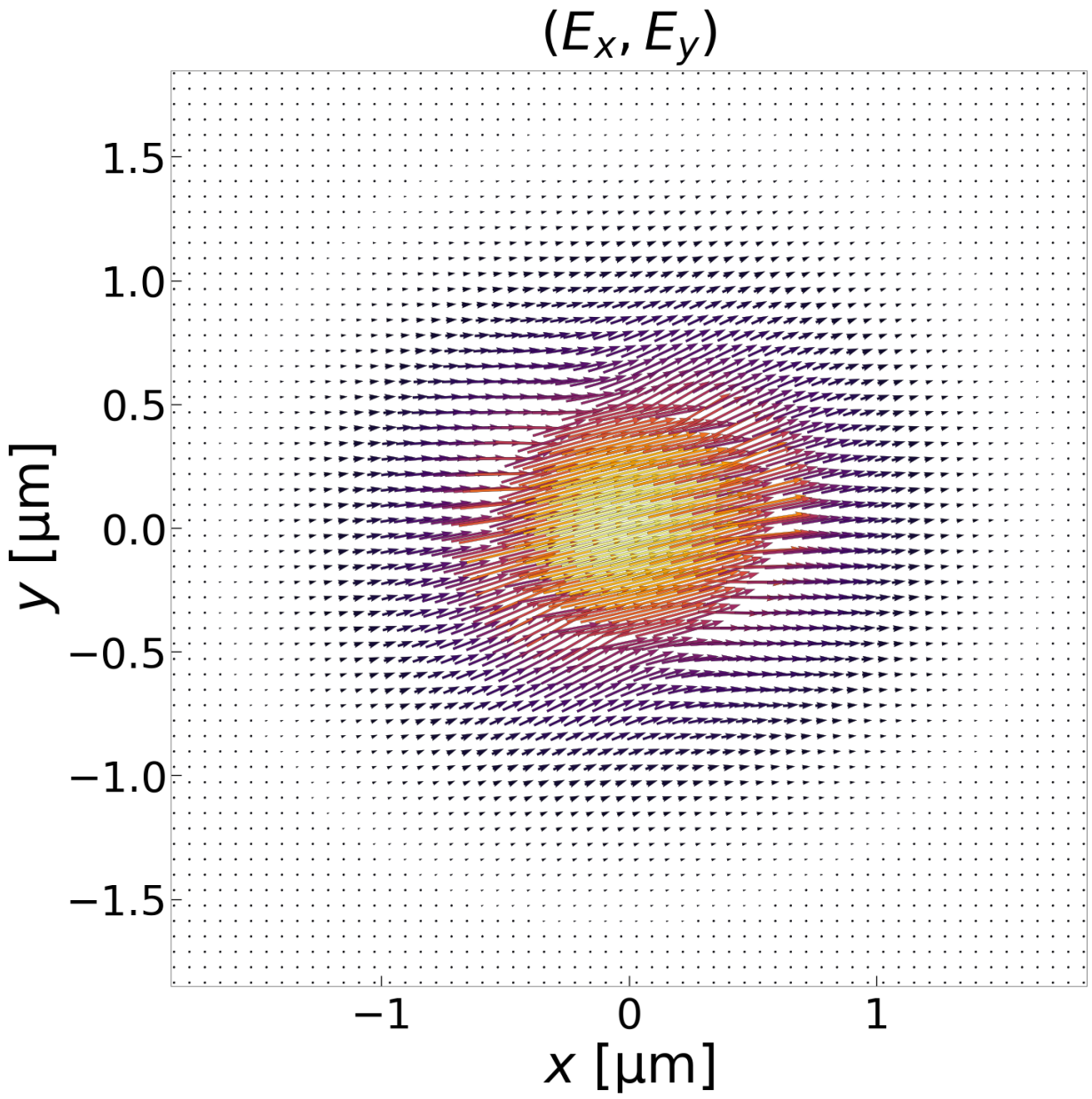


Fig. 4.19: Fundamental optical mode fields.

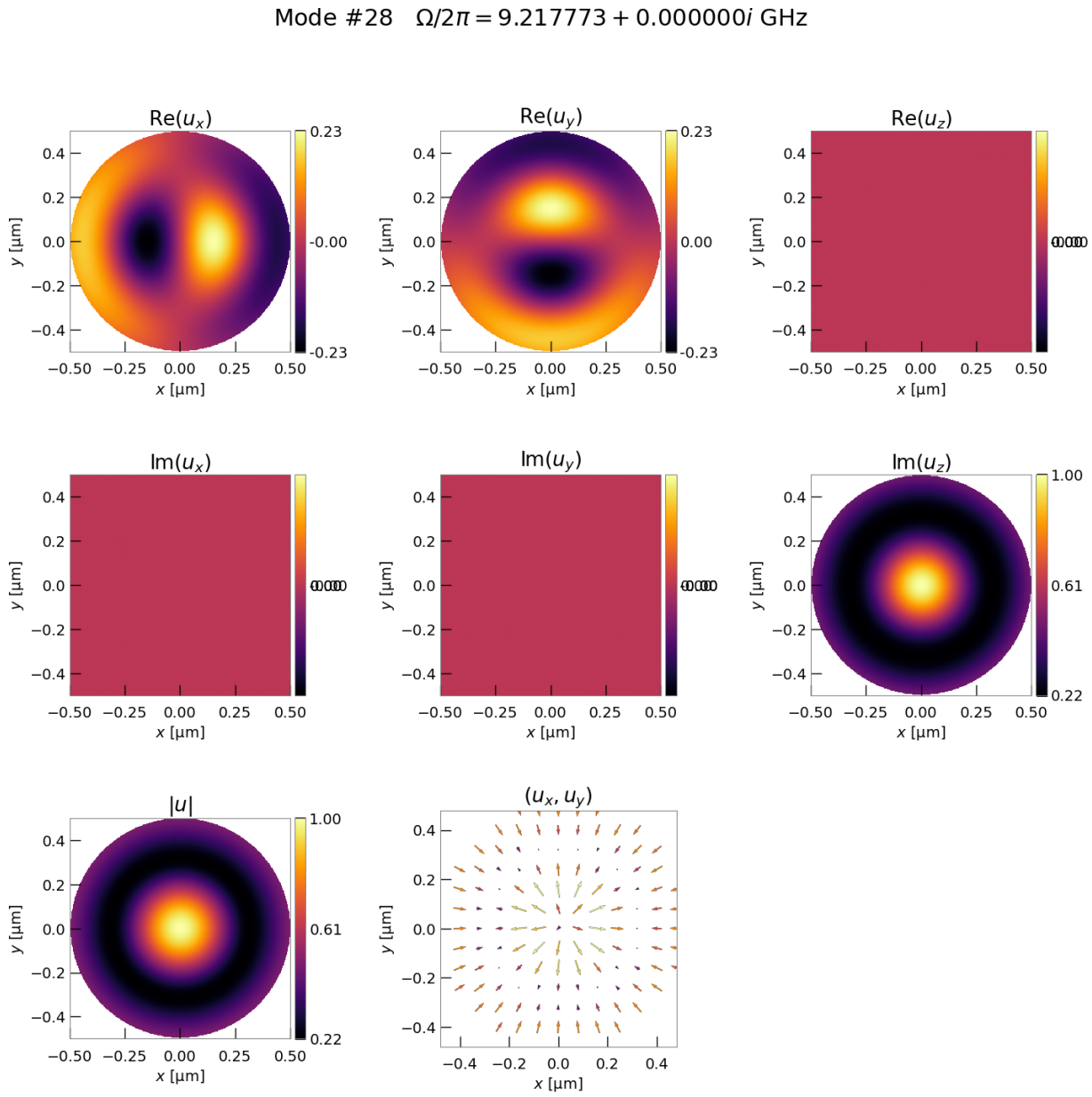


Fig. 4.20: Fundamental acoustic mode fields.

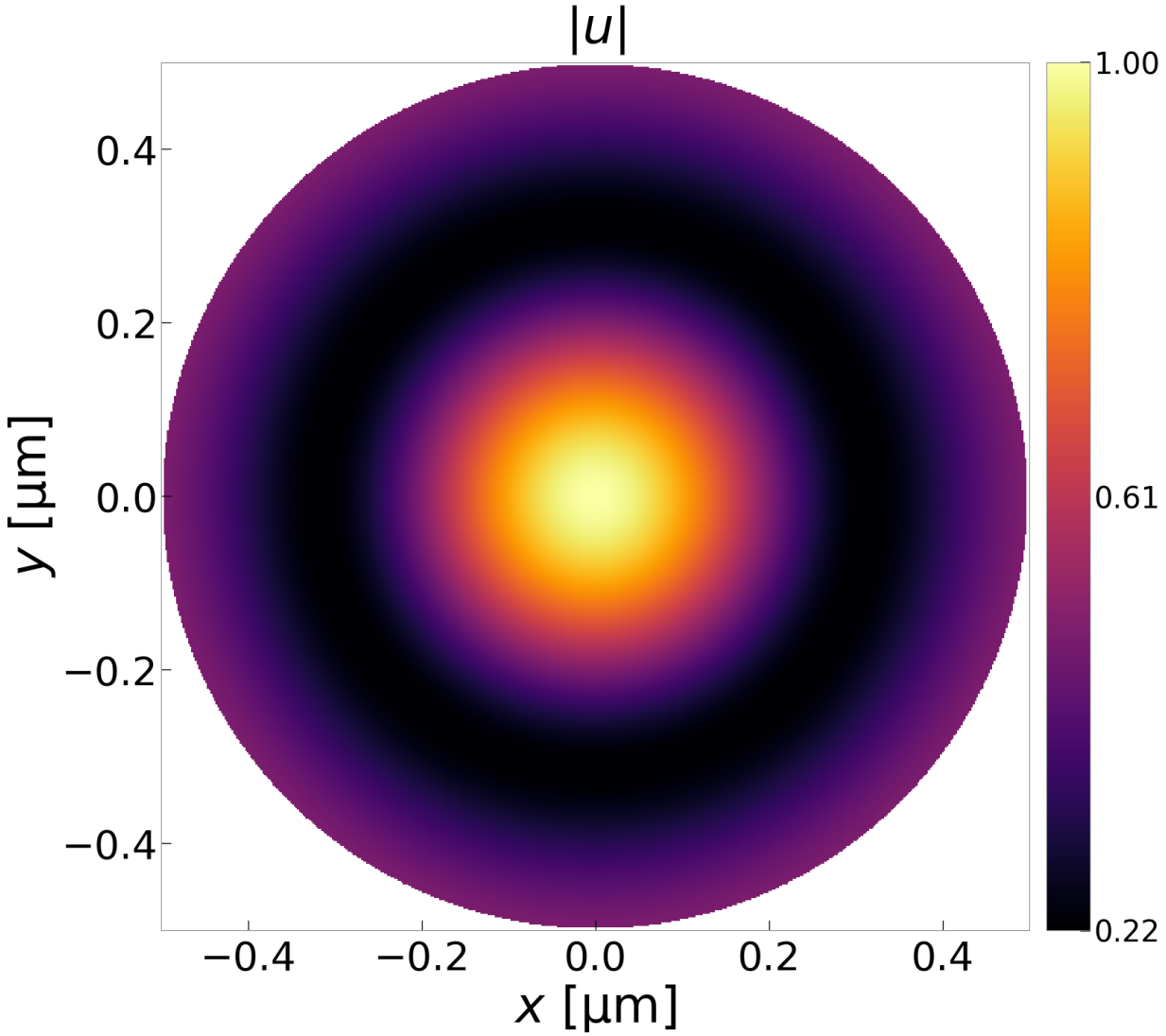


Fig. 4.21: Fundamental acoustic mode fields.

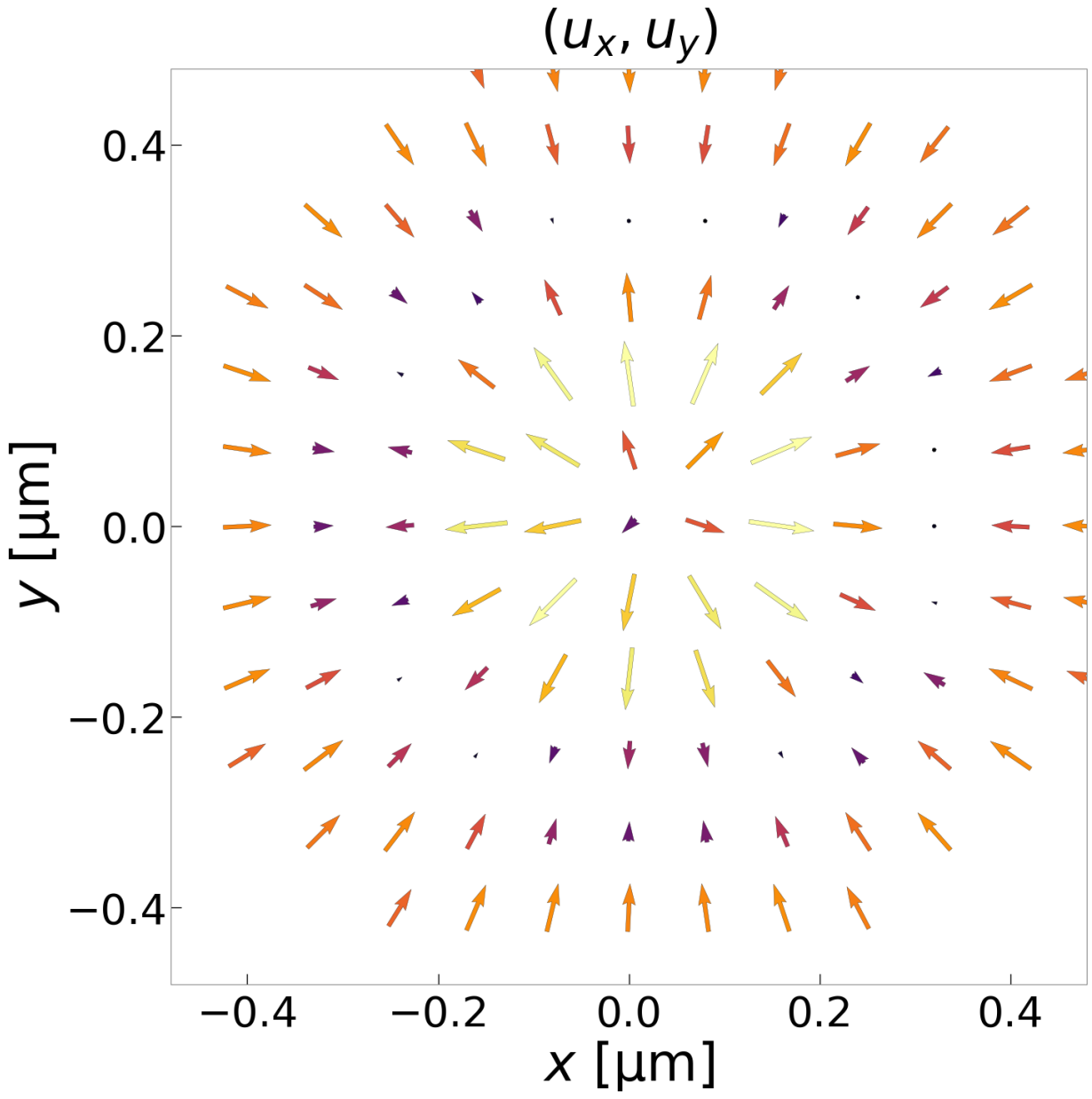


Fig. 4.22: Fundamental acoustic mode fields.

4.2.2 BSBS - Rectangular Waveguide - Silicon

```
print("\n Simulation time (sec.)", (end - start))
"""
Script to evaluate backward Brillouin scattering in a rectangular Si waveguide
"""

# Import the necessary packages
import time
import datetime
import numpy as np
import sys
import math
sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from fortran import NumBAT
from matplotlib import pyplot as plt
# Naming conventions
# AC: acoustic
# EM: electromagnetic
# k_AC: acoustic wavevector

start = time.time()

# Specify Geometric Parameters - all in [nm].
wl_nm = 1550 # Wavelength of EM wave in vacuum.
inc_a_x = 450 # Waveguide widths.
inc_a_y = 200
# Unit cell dimensions must be sufficiently large to ensure fields are zero at␣
↪outermost boundary.
unitcell_x = 3.01*wl_nm
unitcell_y = unitcell_x #be careful to ensure not whole integer multiples
inc_shape = 'rectangular' # Shape of the waveguide.

# Specify number of electromagnetic modes and acoustic modes involved in the
# calculation for BSBS
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100
# The mode number for the optical field. Typically 0 for BSBS.
EM_ival_pump = 0
# The EM Stokes mode number for which to calculate interaction with AC modes.␣
↪Typically 0 for BSBS.
EM_ival_Stokes = EM_ival_pump
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Output files are generated in a folder with the following prefix
prefix_str = 'bsbs-josab-450x200nmSi'

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2021_Poulton"),
```

```
                        lc_bkg=0.05, # mesh coarseness in background, larger lc_bkg =␣
↪coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False) # note requires x-windows configuration to␣
↪work

# Print information on material data in terminal
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)

# Initial guess for the EM effective index of the waveguide
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))

# A computation interruption if needed
#sys.exit("We interrupt your regularly scheduled computation to bring you ... for now
↪")

# Calculate the Electromagnetic modes of the Stokes field.
sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

# Generating images for the EM modes involved in the calculation
print("Plotting EM fields ")
plotting.plot_mode_fields(sim_EM_pump,
                          ivals=[EM_ival_pump],
                          EM_AC='EM_E', num_ticks=3,xlim_min=0.4, xlim_max=0.4, ylim_
↪min=0.4, ylim_max=0.4,
                          prefix_str=prefix_str, pdf_png='png', ticks=True, quiver_
↪points=10,
                          comps=['Et','Eabs'], n_points=1000, colorbar=True)

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

# Calculating the 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))

# Calculating 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.
AC_freqs_Hz = np.round(np.real(sim_AC.Eig_values)*1e-9, 4)
```

```
print('\n Freq of AC modes (GHz) \n', AC_freqs_Hz)

# Calculate total SBS gain, photoelastic and moving boundary contributions, as
# well as other important quantities
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)

# 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)

# Display these in terminal
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
#determining the location of the maximum gain
maxGainloc=np.argmax(abs(masked.data)) ;

print("Plotting acoustic mode corresponding to maximum")
plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,␣
↪ivals=[maxGainloc],
                          num_ticks=3, quiver_points=40, pdf_png='png',ticks=True,␣
↪comps=['ut','uabs'], colorbar=True)

# Displaying results for the maximum found in the selection
print("-----------------")
print("Displaying results for maximum gain value found:")
maxGainloc=np.argmax(abs(masked.data)) ;
print("Greatest SBS_gain  [1/(Wm)] total \n", masked.data[maxGainloc])
print("displaying corresponding acoustic mode number (i.e., AC_field_#) for reference␣
↪\n",maxGainloc )
print("EM Pump Power [Watts] \n", sim_EM_pump.EM_mode_power[EM_ival_pump] )
print("EM Stokes Power [Watts] \n", sim_EM_Stokes.EM_mode_power[EM_ival_Stokes] )
print("EM angular frequency [THz] \n", sim_EM_pump.omega_EM/1e12 )
print("AC Energy Density [J*m^{-1}] \n", sim_AC.AC_mode_energy_elastic[maxGainloc] )
print("AC loss alpha [1/s] \n", alpha[maxGainloc] )
print("AC frequency [GHz] \n", sim_AC.Omega_AC[maxGainloc]/(1e9*2*math.pi) )
print("AC linewidth [MHz] \n", linewidth_Hz[maxGainloc]/1e6)

#since the overlap is not returned directly we'll have to deduce it
absQtot2 = (alpha[maxGainloc]*sim_EM_pump.EM_mode_power[EM_ival_pump]*sim_EM_Stokes.
↪EM_mode_power[EM_ival_Stokes]*sim_AC.AC_mode_energy_elastic[maxGainloc]*masked.
↪data[maxGainloc])/(2*sim_EM_pump.omega_EM*sim_AC.Omega_AC[maxGainloc]);
absQtot = pow(absQtot2,1/2)
print("Total coupling |Qtot| [W*m^{-1}*s] \n", absQtot )

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

Let's also calculate the acoustic dispersion relation for this structure.

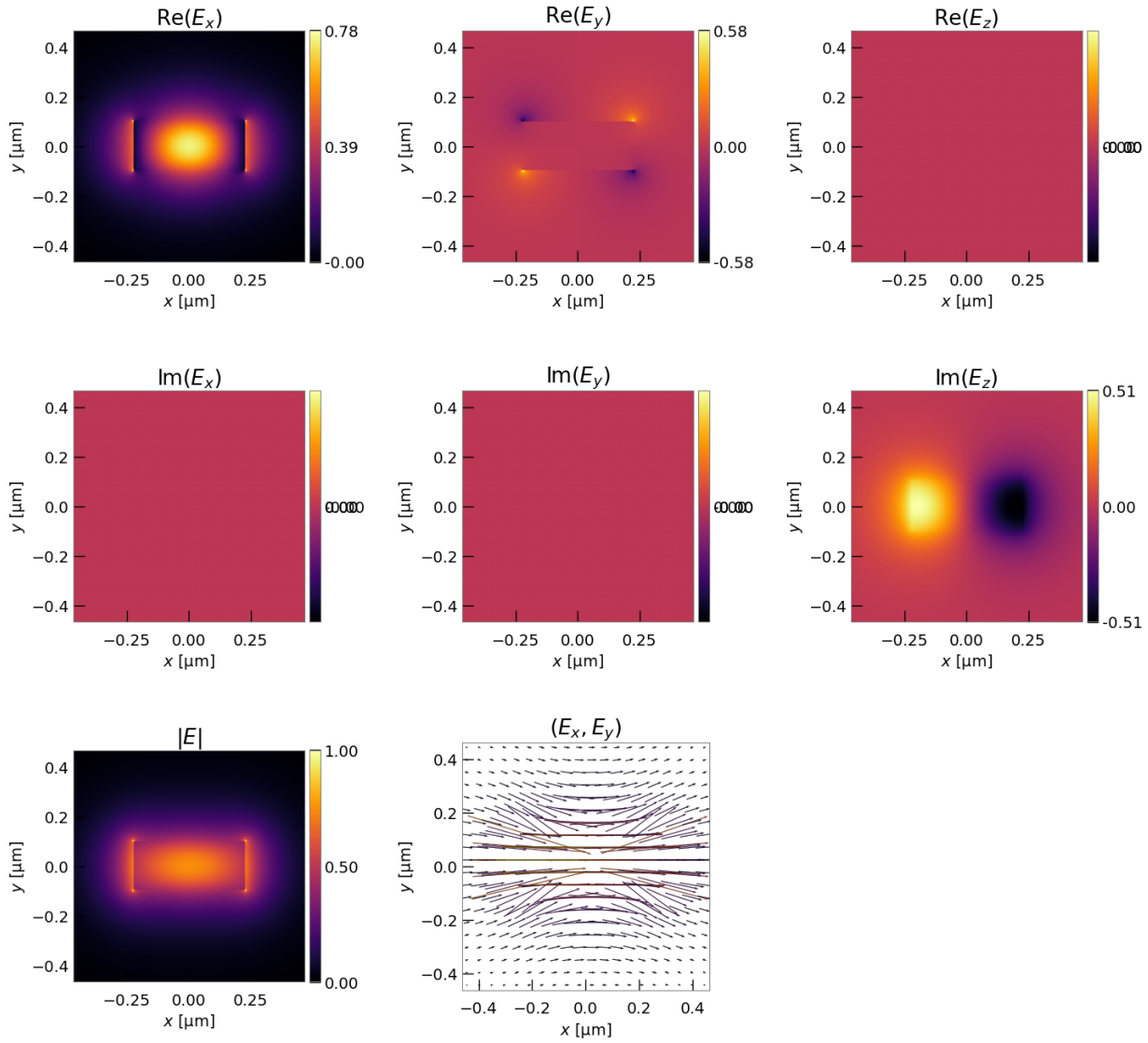


Fig. 4.23: Fundamental optical mode fields.

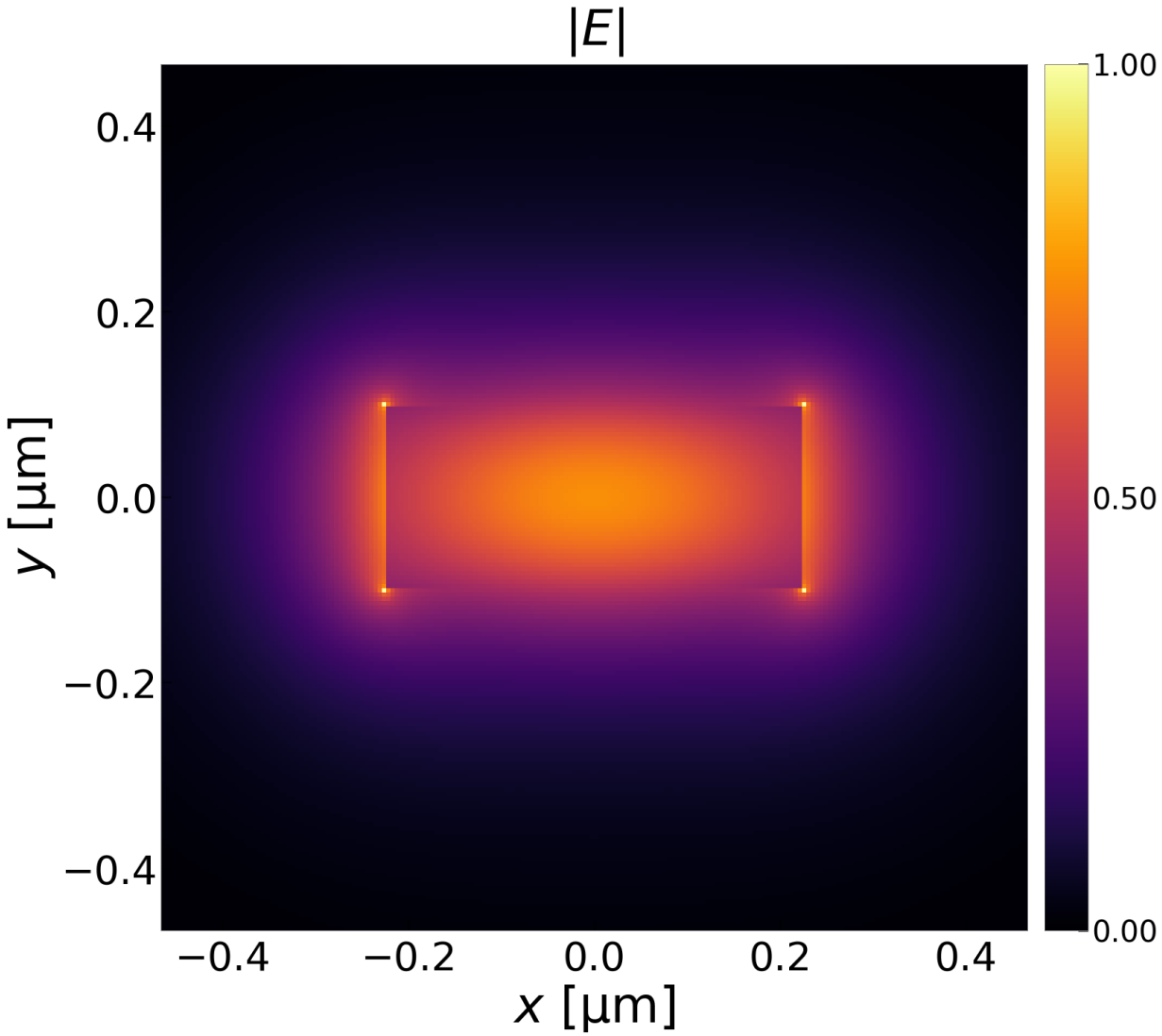


Fig. 4.24: Fundamental optical mode fields.

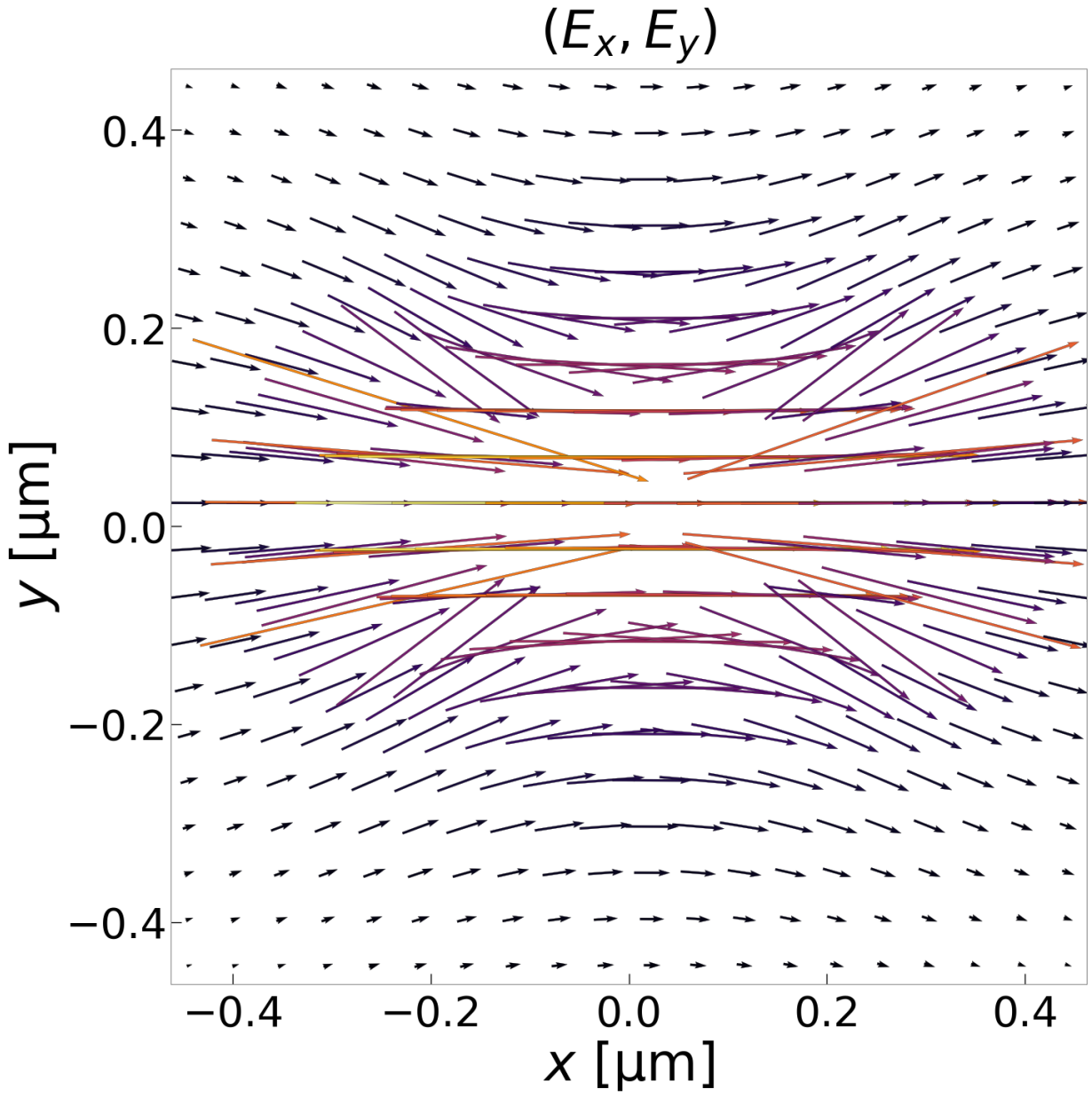


Fig. 4.25: Fundamental optical mode fields.

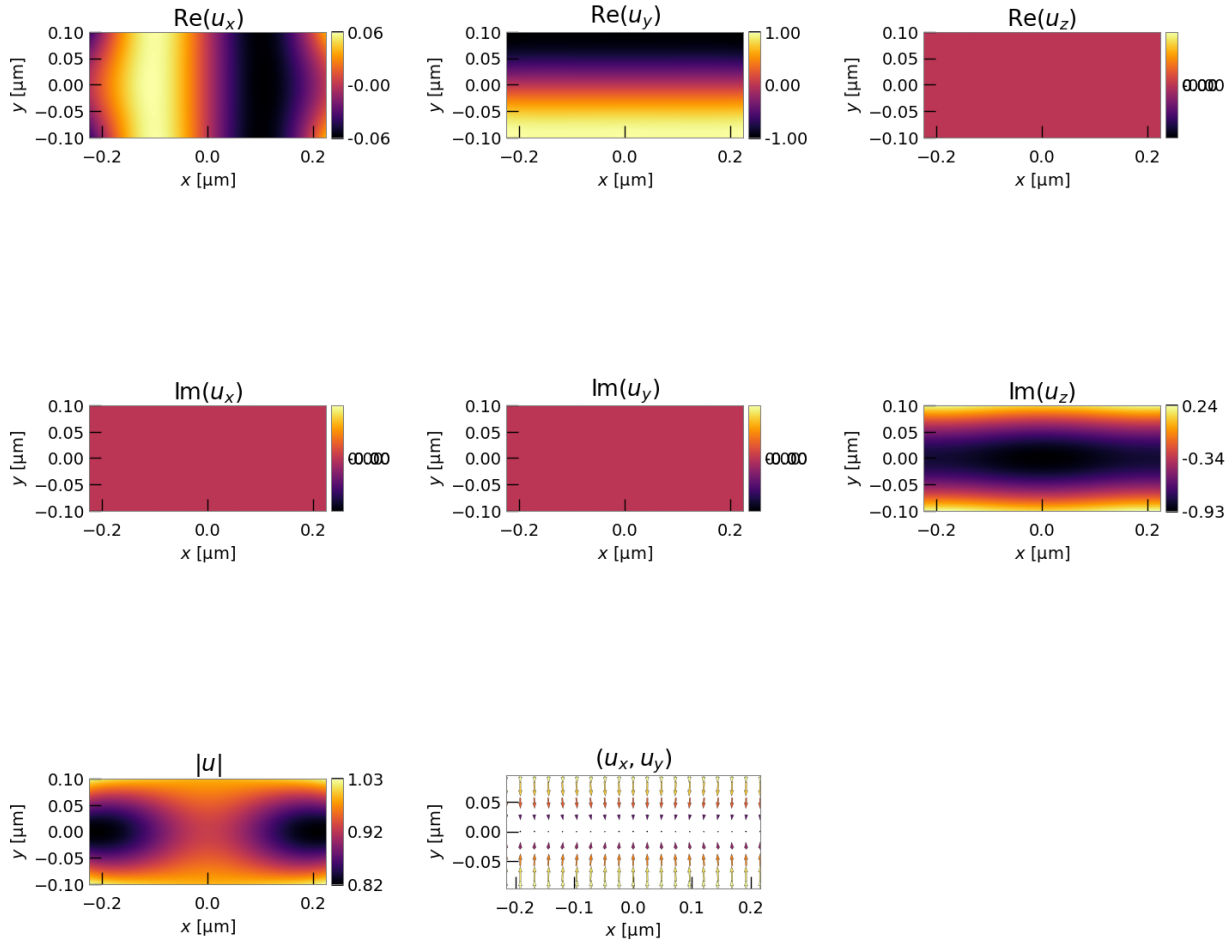


Fig. 4.26: Fundamental acoustic mode fields.

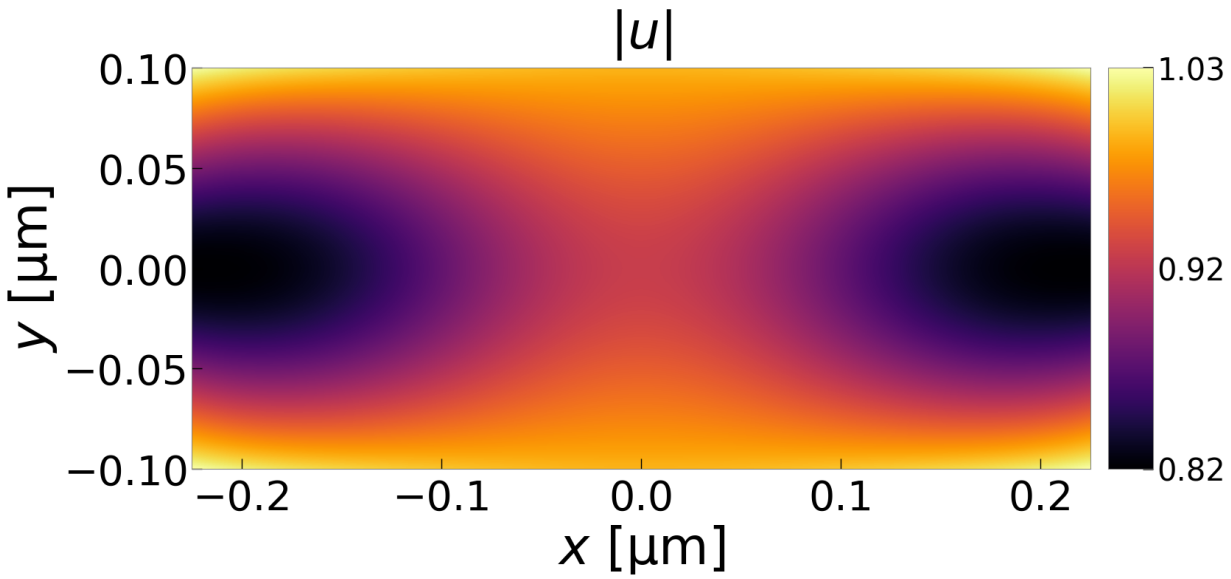


Fig. 4.27: Fundamental acoustic mode fields.

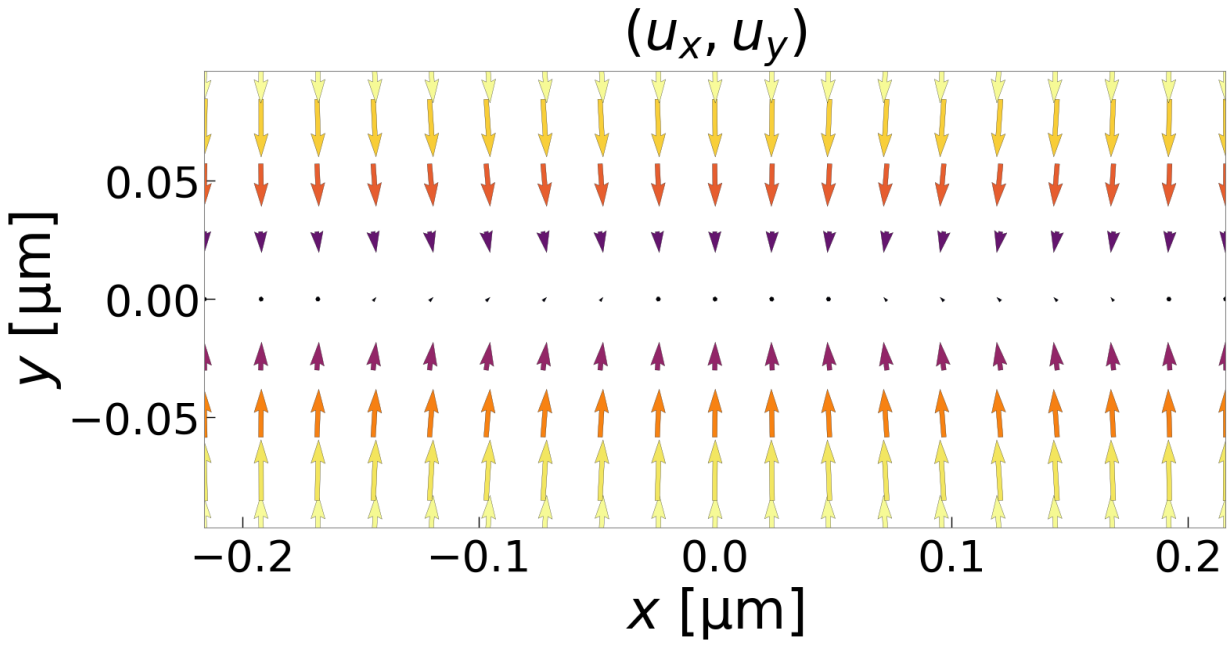


Fig. 4.28: Fundamental acoustic mode fields.

```
print("\n Simulation time (sec.)", (end - start))
"""
Calculate dispersion diagram of the acoustic modes in a rectangular Si waveguide
"""

# Import the necessary packages
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 = 3.01*wl_nm
unitcell_y = unitcell_x
inc_a_x = 450 # Waveguide widths.
inc_a_y = 200
inc_shape = 'rectangular'
# Choose modes to include.
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100
EM_ival_pump = 0
EM_ival_Stokes = EM_ival_pump
AC_ival = 'All'

# Use all specified parameters to create a waveguide object
wguide = objects.Struct(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                        material_bkg=materials.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2021_Poulton"),
                        lc_bkg=0.05, # mesh coarseness in background, larger lc_bkg =␣
↪coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False) # note requires x-windows configuration to␣
↪work

# 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)

# Print EM mode info
print('\n k_z of EM modes \n', np.round(np.real(sim_EM_pump.Eig_values),4))
n_eff_sim = np.real(sim_EM_pump.Eig_values[EM_ival_pump]*((wl_nm*1e-9)/(2.*np.pi)))
print("\n Fundamental optical mode ")
print(" n_eff = ", np.round(n_eff_sim, 4))

#   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])
# Number of wavevectors steps.
nu_ks = 50

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_B1, = 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_B2, = 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_B3, = plt.plot(np.real(q_ac/k_AC), Om, '^g')

    print("Wavevector loop", i_ac+1, "/", nu_ks)
ax.set_ylim(0,15)
ax.set_xlim(0,1)
plt.legend([sym_A, sym_B1, sym_B2, sym_B3],['A',r'B$_1$',r'B$_2$',r'B$_3$'], loc=
↪'lower right')

plt.xlabel(r'Axial wavevector (normalised)')
plt.ylabel(r'Frequency (GHz)')
plt.savefig('dispersioncurves_classified.png', bbox_inches='tight')
plt.close()

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

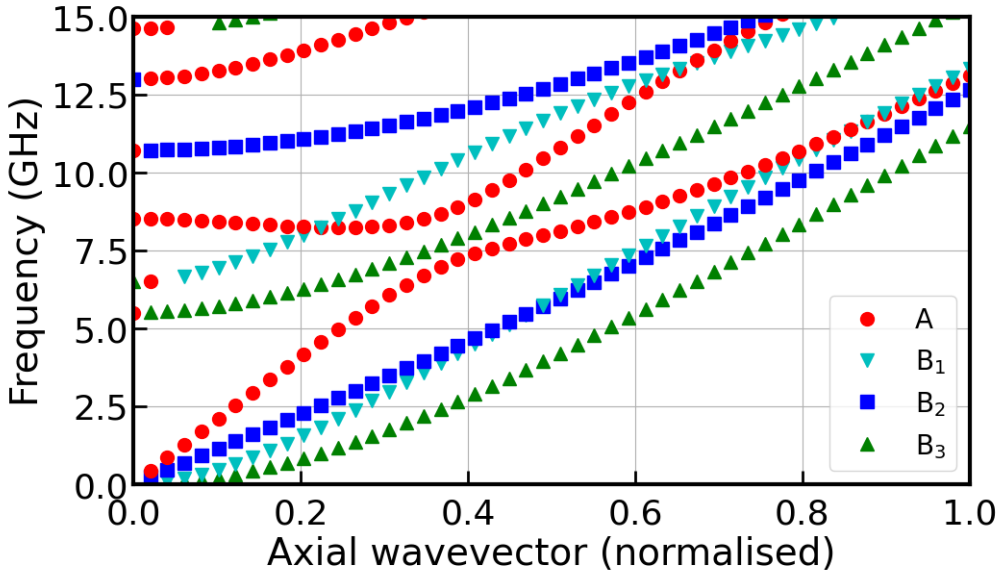


Fig. 4.29: Acoustic dispersion diagram with modes categorised by symmetry as in Table 1 of "Formal selection rules for Brillouin scattering in integrated waveguides and structured fibers" by C. Wolff, M. J. Steel, and C. G. Poulton https://doi.org/10.1364/OE.22.032489

4.2.3 FSBS - Circular Waveguide - Silica

```
print("\n Simulation time (sec.)", (end - start))
"""
Script to evaluate forward Brillouin scattering in a cylindrical SiO2 waveguide
"""

# Import the necessary packages
import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
import copy
import math
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()

# Specify Geometric Parameters - all in [nm].
wl_nm = 1550 # Wavelength of EM wave in vacuum.
# Unit cell dimensions must be sufficiently large to ensure fields are zero at␣
↪outermost boundary.
unitcell_x = 4.01*wl_nm #be careful to ensure not whole integer multiples
unitcell_y = unitcell_x
inc_a_x = 1000 # Waveguide widths.
inc_a_y = inc_a_x
inc_shape = 'circular' # Shape of the waveguide.

# Specify number of electromagnetic modes and acoustic modes involved in the
# calculation for FSBS
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100
# The EM pump mode(s) for which to calculate interaction with AC modes. Typically 0␣
↪for FSBS.
EM_ival_pump = 1
# The EM Stokes mode(s) for which to calculate interaction with AC modes. Typically 0␣
↪for FSBS.
EM_ival_Stokes = EM_ival_pump
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Output files are generated in a folder with the following prefix
prefix_str = 'fsbs-josab-1umSiO2'
```

```
# 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.get_material("Vacuum"),
                        material_a=materials.get_material("SiO2_2021_Poulton"),
                        lc_bkg=0.05, # mesh coarseness in background, larger lc_bkg =␣
↪coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger lc2 = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False) # note requires x-windows configuration to␣
↪work

# 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)

# Initial guess for the EM effective index of the waveguide
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)

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

# 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))

# A computation interruption if needed
# sys.exit("We interrupt your regularly scheduled computation to bring you something␣
↪completely different... for now")

#calculate the EM modes for the Stokes
sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)

# Generate images for the EM modes involved in the calculation
# note: use EM_AC='EM_H' for magnetic H field
print("Plotting EM fields ")

plotting.plot_mode_fields(sim_EM_pump,
                          ivals=[EM_ival_pump],
                          EM_AC='EM_E', num_ticks=3,xlim_min=0.2, xlim_max=0.2, ylim_
↪min=0.2, ylim_max=0.2,
                          prefix_str=prefix_str, pdf_png='png', ticks=True, quiver_
↪points=10,
                          comps=['Et','Eabs'], n_points=1000, colorbar=True)

# Specify an acoustic wavevector that is sufficiently close to zero and print
k_AC = 5
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)

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

# Calculate total SBS gain, photoelastic and moving boundary contributions, as
# well as other important quantities
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)

# 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)

# Display these in terminal
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
#determining the location of the maximum gain
maxGainloc=7 ;  #note sometimes its necessary to manually specify as certain values␣
↪are NOT possible by symmetry arguments

print("Plotting acoustic modes")

plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,␣
↪ivals=[maxGainloc],
                          num_ticks=3, quiver_points=40, pdf_png='png',ticks=True,␣
↪comps=['ut','uabs'], colorbar=True)

# Displaying results for the maximum found in the selection
print("-----------------")
print("Displaying results for maximum (physically realisable) \"gain\" value found:")
print("Greatest SBS_gain  [1/(Wm)] total \n", masked.data[maxGainloc])
print("displaying corresponding acoustic mode number (i.e., AC_field_#) for reference␣
↪\n",maxGainloc )
print("EM Pump Power [Watts] \n", sim_EM_pump.EM_mode_power[EM_ival_pump] )
print("EM Stokes Power [Watts] \n", sim_EM_Stokes.EM_mode_power[EM_ival_Stokes] )
print("EM angular frequency [THz] \n", sim_EM_pump.omega_EM/1e12 )
print("AC Energy Density [J*m^{-1}] \n", sim_AC.AC_mode_energy_elastic[maxGainloc] )
print("AC loss alpha [1/s] \n", alpha[maxGainloc] )
print("AC frequency [GHz] \n", sim_AC.Omega_AC[maxGainloc]/(1e9*2*math.pi) )
print("AC linewidth [MHz] \n", linewidth_Hz[maxGainloc]/1e6)

#since the overlap is not returned directly we'll have to deduce it
absQtot2 = (alpha[maxGainloc]*sim_EM_pump.EM_mode_power[EM_ival_pump]*sim_EM_Stokes.
↪EM_mode_power[EM_ival_Stokes]*sim_AC.AC_mode_energy_elastic[maxGainloc]*masked.
↪data[maxGainloc])/(2*sim_EM_pump.omega_EM*sim_AC.Omega_AC[maxGainloc]);
absQtot = pow(absQtot2,1/2)
```

```
print("Total coupling |Qtot| [W*m^{-1}*s] \n", absQtot )

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

Mode #1 $k_z = 4768812.257219 - 0.000000i$ $n_{eff} = 1.176419 - 0.000000i$

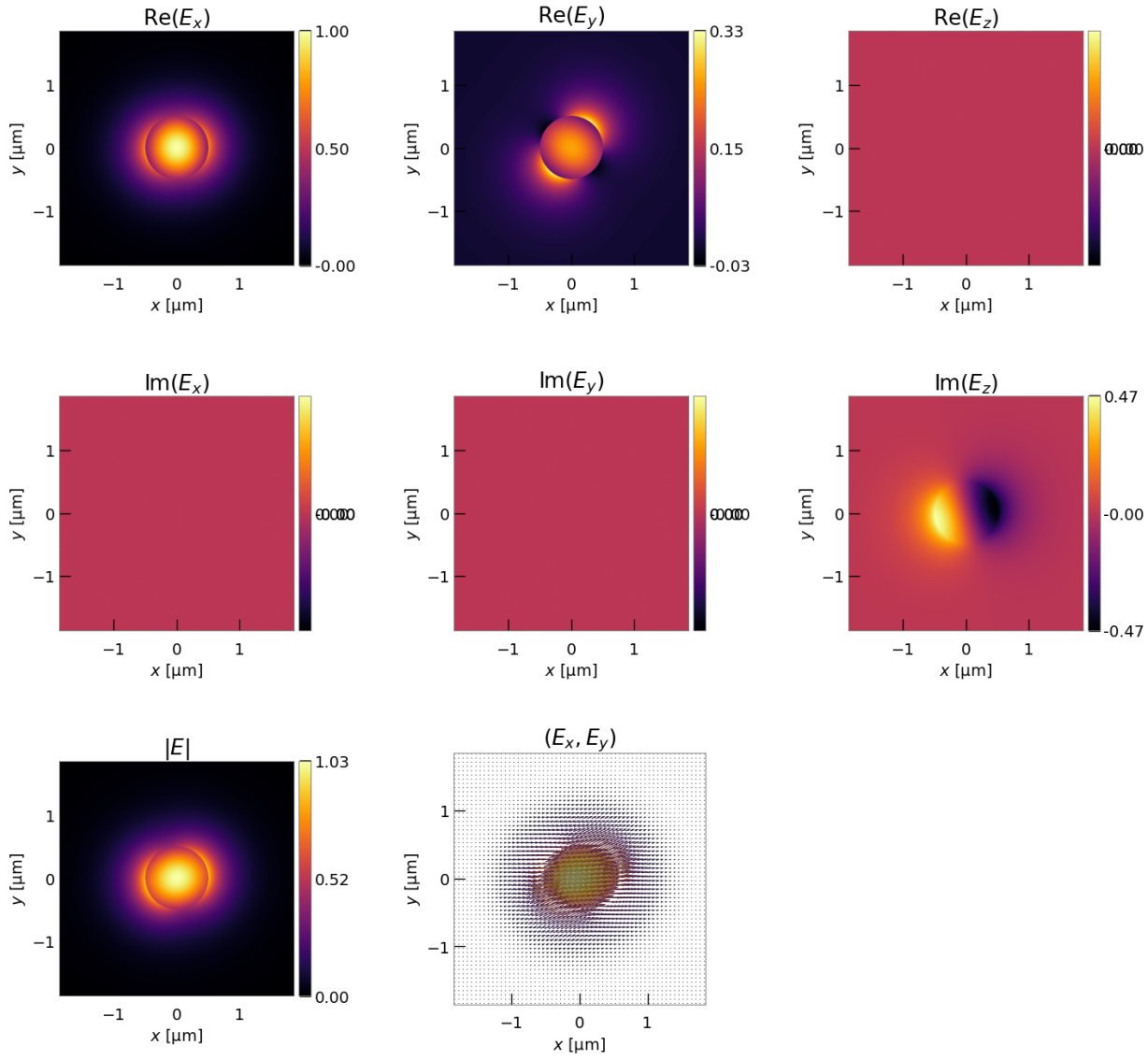


Fig. 4.30: Fundamental optical mode fields.

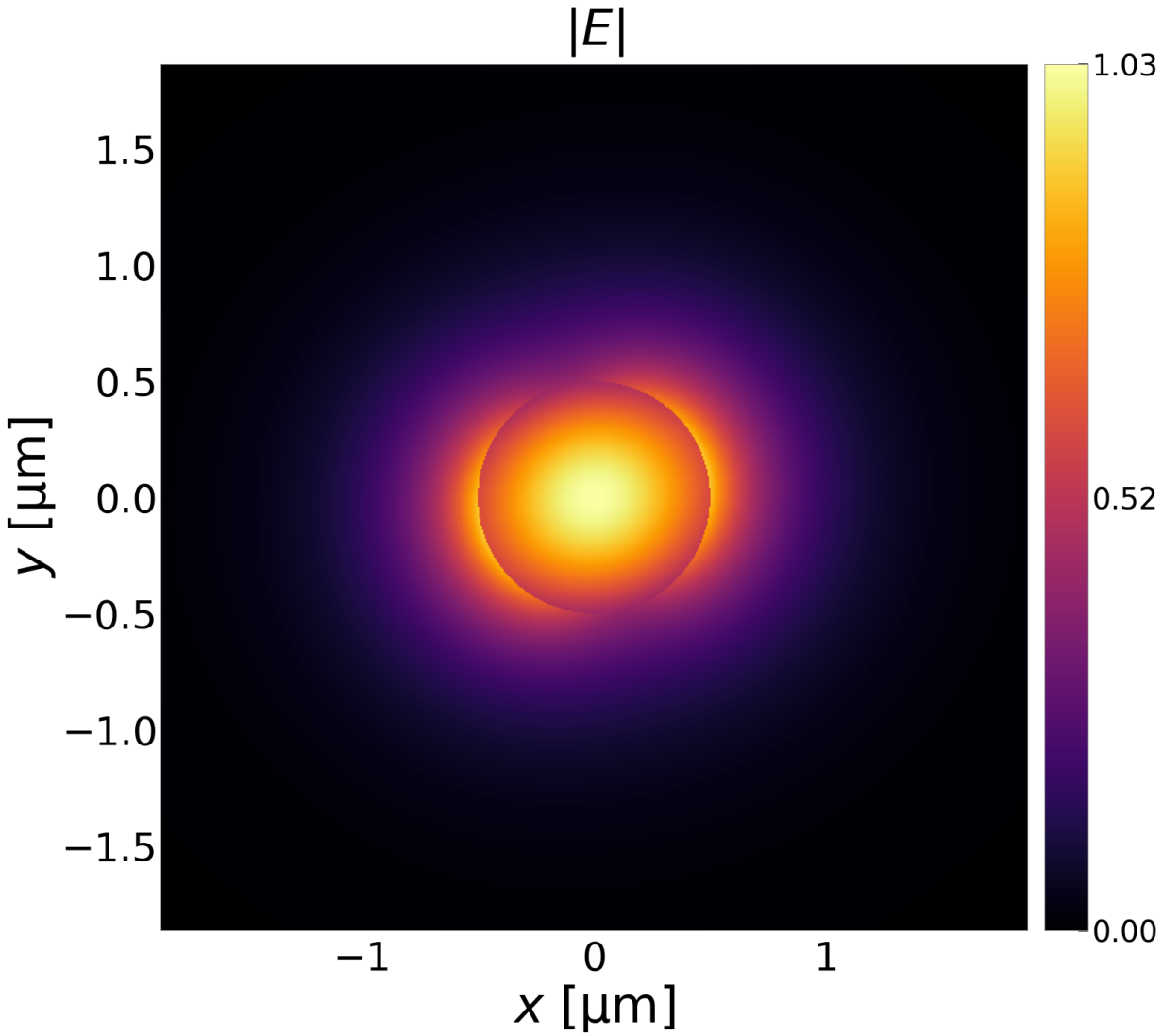


Fig. 4.31: Fundamental optical mode fields.

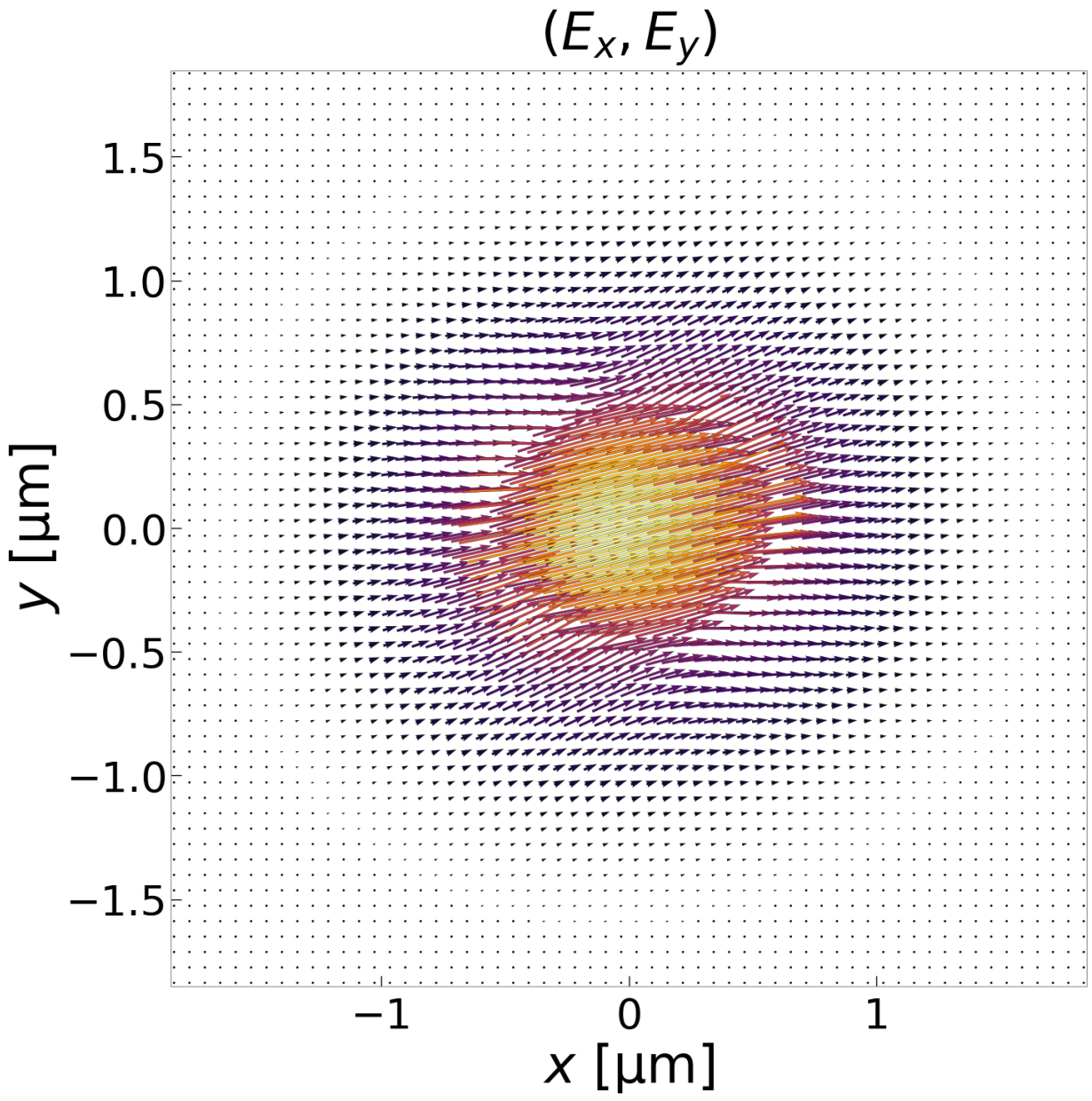


Fig. 4.32: Fundamental optical mode fields.

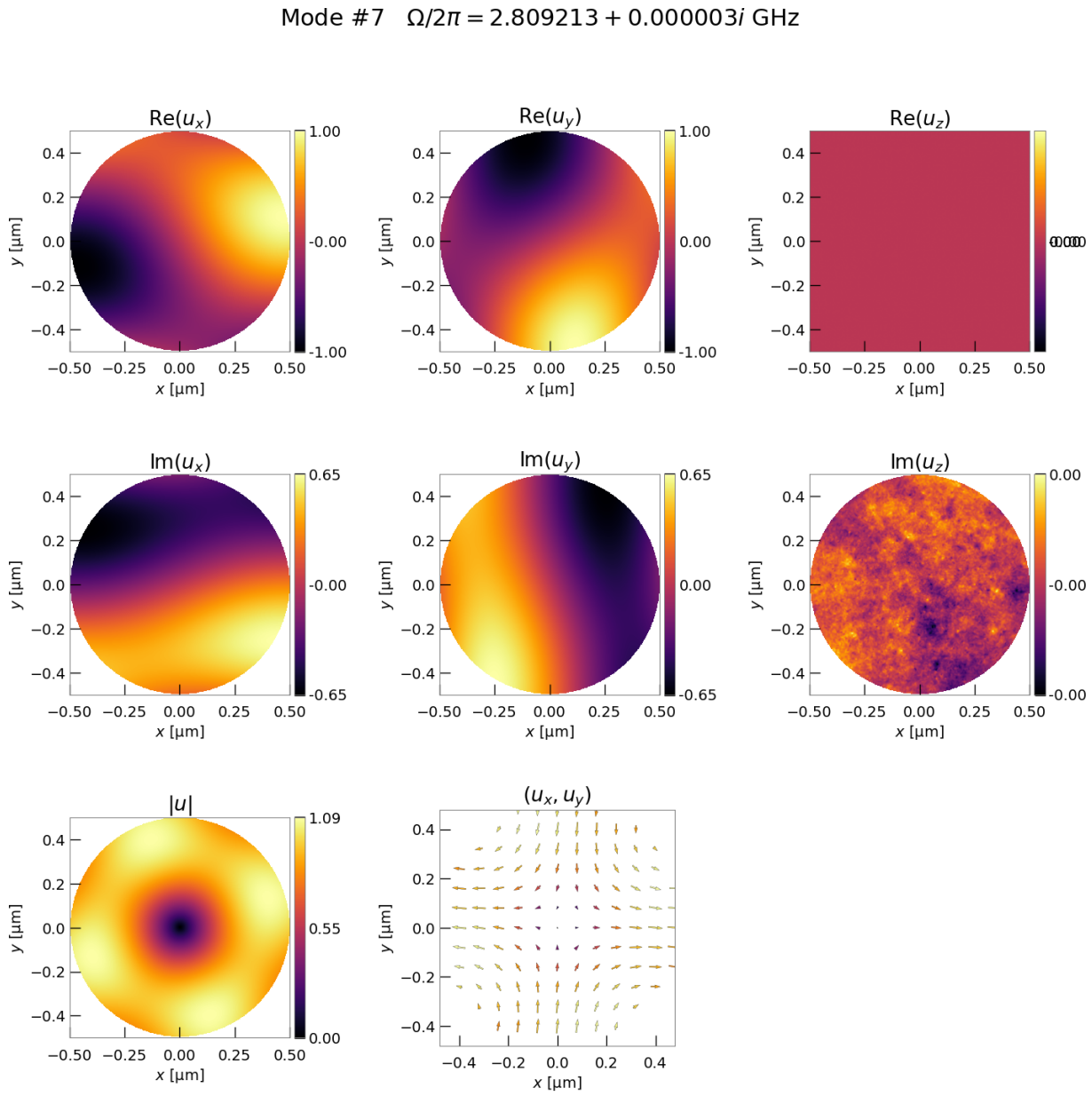


Fig. 4.33: Fundamental acoustic mode fields.

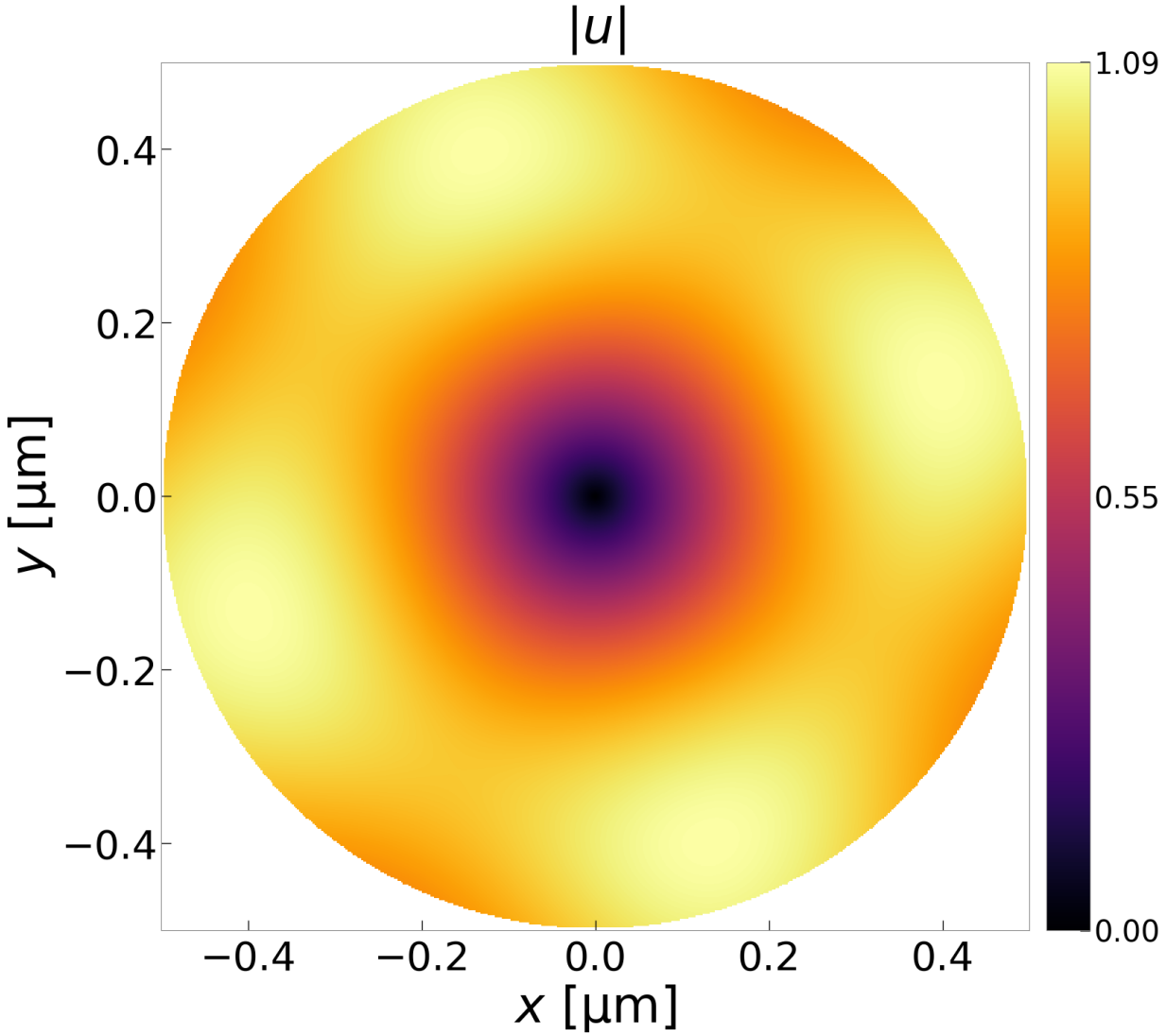


Fig. 4.34: Fundamental acoustic mode fields.

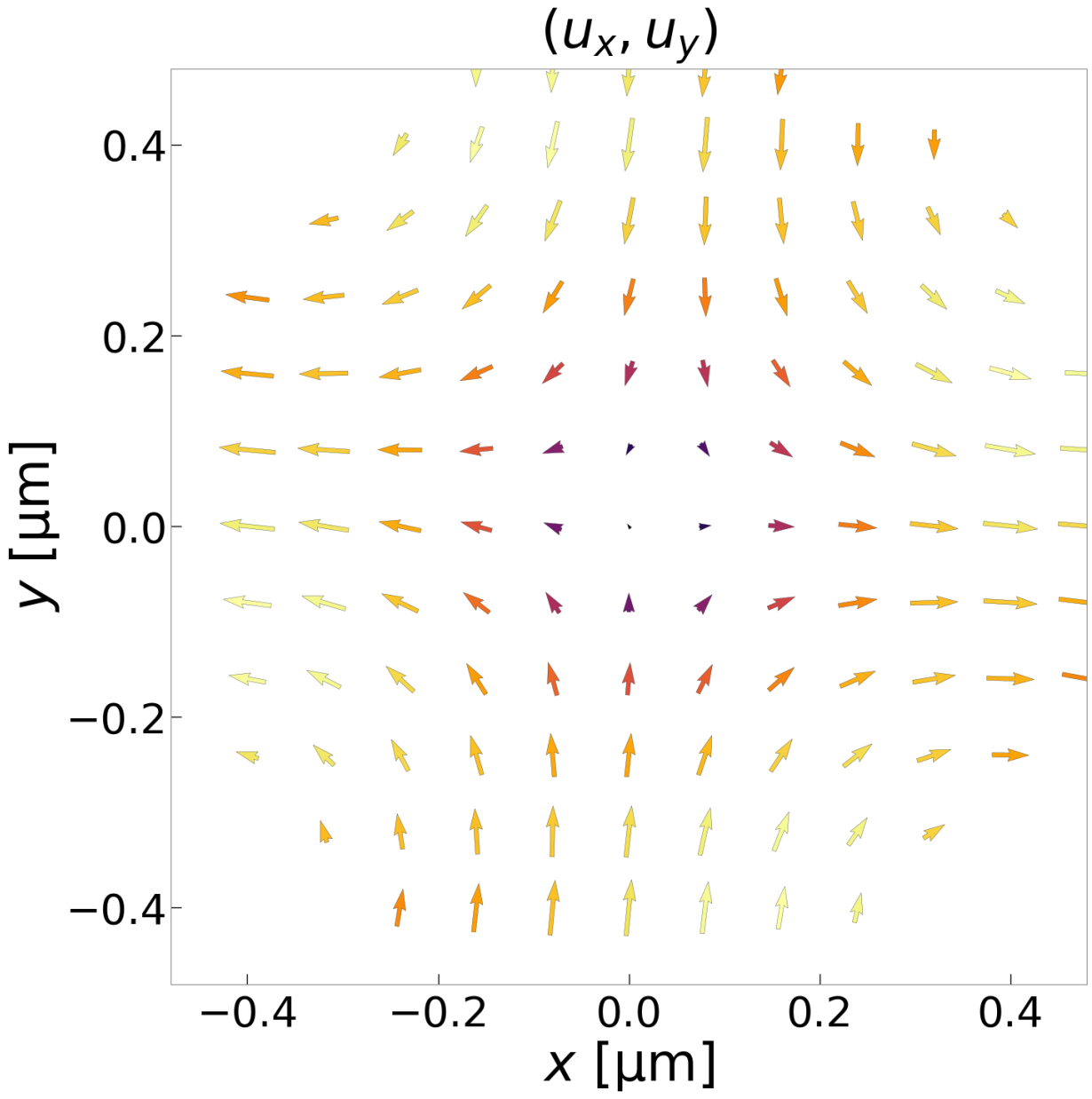


Fig. 4.35: Fundamental acoustic mode fields.

4.2.4 FSBS - Rectangular Waveguide - Silicon

```
print("\n Simulation time (sec.)", (end - start))
"""
Script to evaluate forward Brillouin scattering in a rectangular Si waveguide
"""

# Import the necessary packages
import time
import datetime
import numpy as np
import sys
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
import copy
import math
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()

# Specify Geometric Parameters - all in [nm].
wl_nm = 1550
unitcell_x = 3.01*wl_nm
unitcell_y = unitcell_x
inc_a_x = 450 # Waveguide widths.
inc_a_y = 200
inc_shape = 'rectangular'

# Specify number of electromagnetic modes and acoustic modes involved in the
# calculation for FSBS
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100
# The EM pump mode(s) for which to calculate interaction with AC modes. Typically 0␣
↪for FSBS.
EM_ival_pump = 0
# The EM Stokes mode(s) for which to calculate interaction with AC modes. Typically 0␣
↪for FSBS.
EM_ival_Stokes = EM_ival_pump
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Output files are generated in a folder with the following prefix
prefix_str = 'fsbs-josab-450x200nmSi'

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2021_Poulton"),
                        lc_bkg=0.05, # mesh coarseness in background, larger lc_bkg =␣
↪coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False) # note requires x-windows configuration to␣
↪work

# 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)

# Initial guess for the EM effective index of the waveguide
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)

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

# 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))

# A computation interruption if needed
# sys.exit("We interrupt your regularly scheduled computation to bring you something␣
↪completely different... for now")

#calculate the EM modes for the Stokes
sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)

# Generate images for the EM modes involved in the calculation
# note: use EM_AC='EM_H' for magnetic H field
print("Plotting EM fields ")
plotting.plot_mode_fields(sim_EM_pump,
                          ivals=[EM_ival_pump],
                          EM_AC='EM_E', num_ticks=3,xlim_min=0.4, xlim_max=0.4, ylim_
↪min=0.4, ylim_max=0.4,
                          prefix_str=prefix_str, pdf_png='png', ticks=True, quiver_
↪points=10,
                          comps=['Et','Eabs'], n_points=1000, colorbar=True)

# Specify an acoustic wavevector that is sufficiently close to zero and print
k_AC = 5
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)
```

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

# Calculate total SBS gain, photoelastic and moving boundary contributions etc
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)

# 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)

# Display these in terminal
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
#determining the location of the maximum gain
maxGainloc=6 ; #note sometimes its necessary to manually specify as certain values␣
↪are NOT possible by symmetry arguments

print("Plotting acoustic modes")

plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,␣
↪ivals=[maxGainloc],
                          num_ticks=3, quiver_points=40, pdf_png='png',ticks=True,␣
↪comps=['ut','uabs'], colorbar=True)

# Displaying results for the maximum found in the selection
print("-----------------")
print("Displaying results for maximum (physically realisable) \"gain\" value found:")
print("Greatest SBS_gain  [1/(Wm)] total \n", masked.data[maxGainloc])
print("displaying corresponding acoustic mode number (i.e., AC_field_#) for reference␣
↪\n",maxGainloc )
print("EM Pump Power [Watts] \n", sim_EM_pump.EM_mode_power[EM_ival_pump] )
print("EM Stokes Power [Watts] \n", sim_EM_Stokes.EM_mode_power[EM_ival_Stokes] )
print("EM angular frequency [THz] \n", sim_EM_pump.omega_EM/1e12 )
print("AC Energy Density [J*m^{-1}] \n", sim_AC.AC_mode_energy_elastic[maxGainloc] )
print("AC loss alpha [1/s] \n", alpha[maxGainloc] )
print("AC frequency [GHz] \n", sim_AC.Omega_AC[maxGainloc]/(1e9*2*math.pi) )
print("AC linewidth [MHz] \n", linewidth_Hz[maxGainloc]/1e6)

#since the overlap is not returned directly we'll have to deduce it
absQtot2 = (alpha[maxGainloc]*sim_EM_pump.EM_mode_power[EM_ival_pump]*sim_EM_Stokes.
↪EM_mode_power[EM_ival_Stokes]*sim_AC.AC_mode_energy_elastic[maxGainloc]*masked.
↪data[maxGainloc])/(2*sim_EM_pump.omega_EM*sim_AC.Omega_AC[maxGainloc]);
absQtot = pow(absQtot2,1/2)
print("Total coupling |Qtot| [W*m^{-1}*s] \n", absQtot )

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

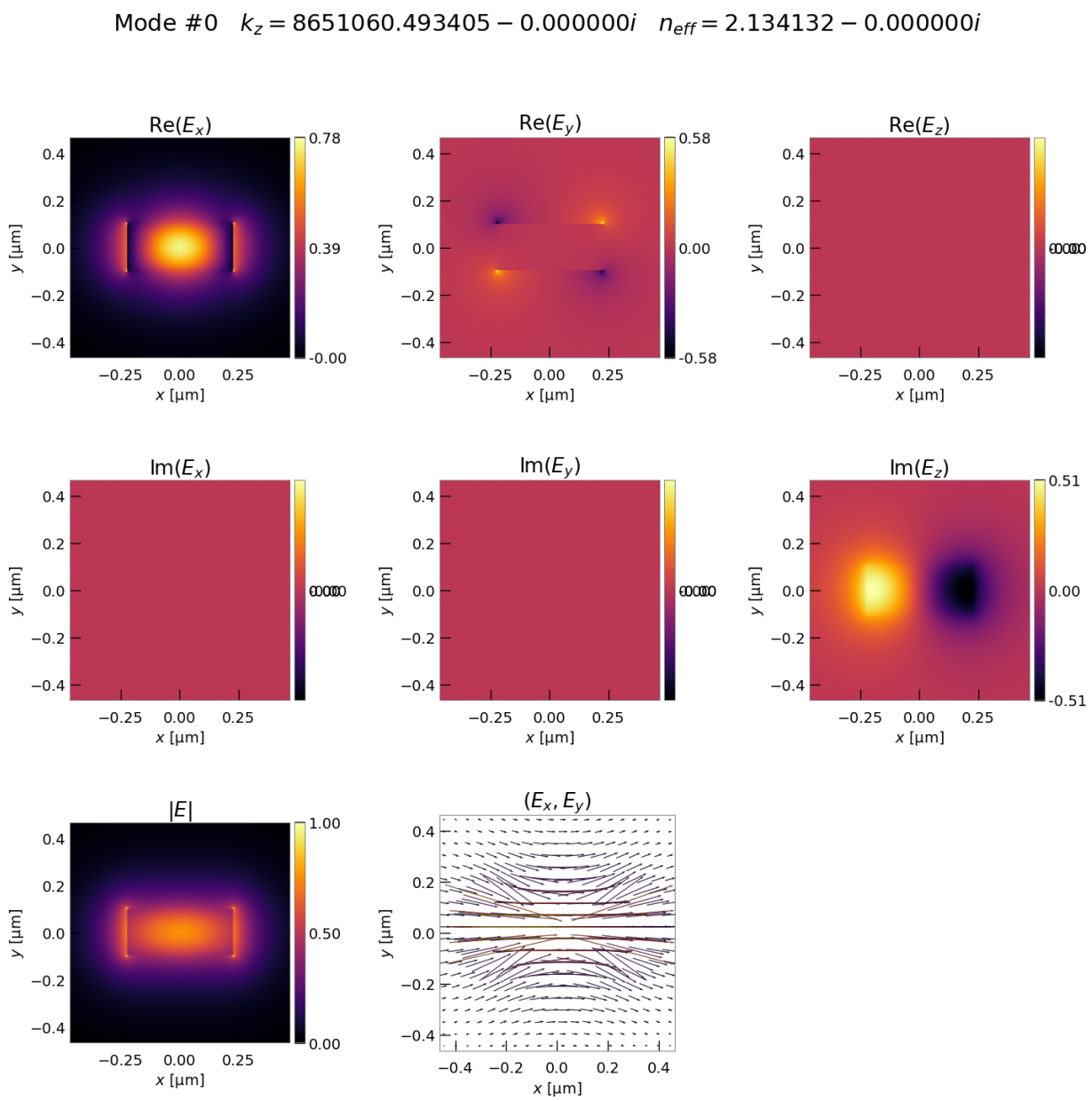


Fig. 4.36: Fundamental optical mode fields.

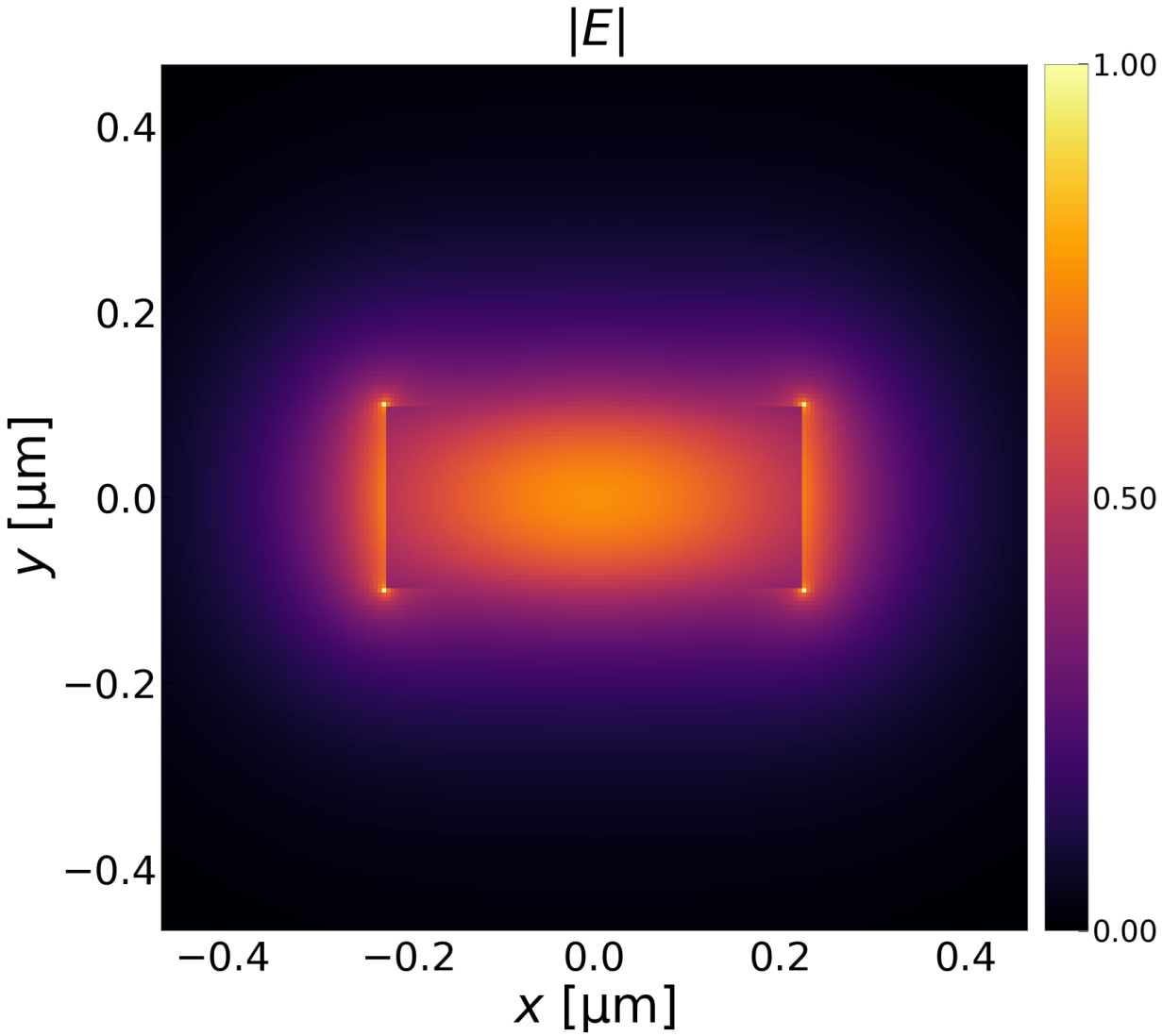


Fig. 4.37: Fundamental optical mode fields.

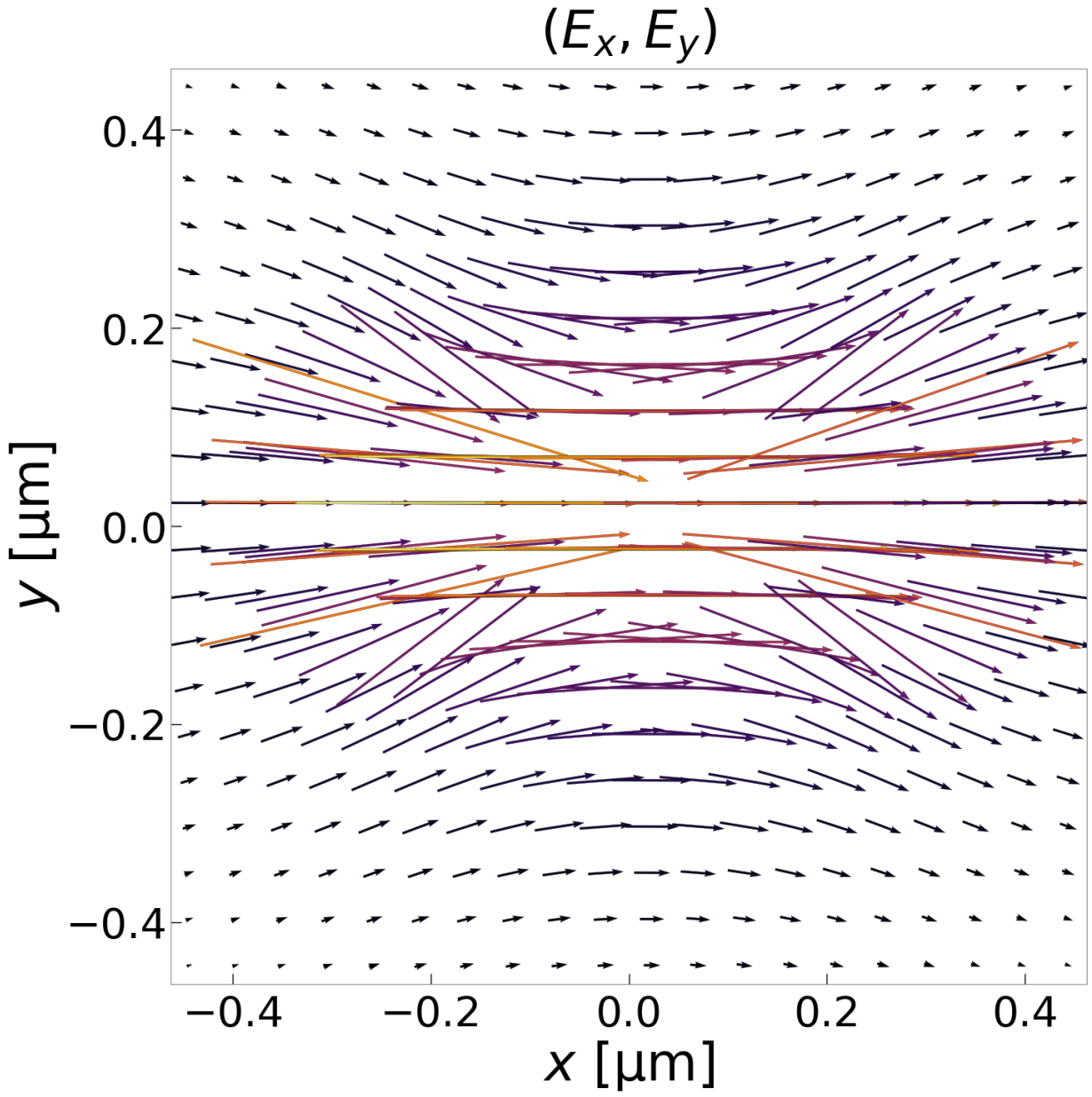


Fig. 4.38: Fundamental optical mode fields.

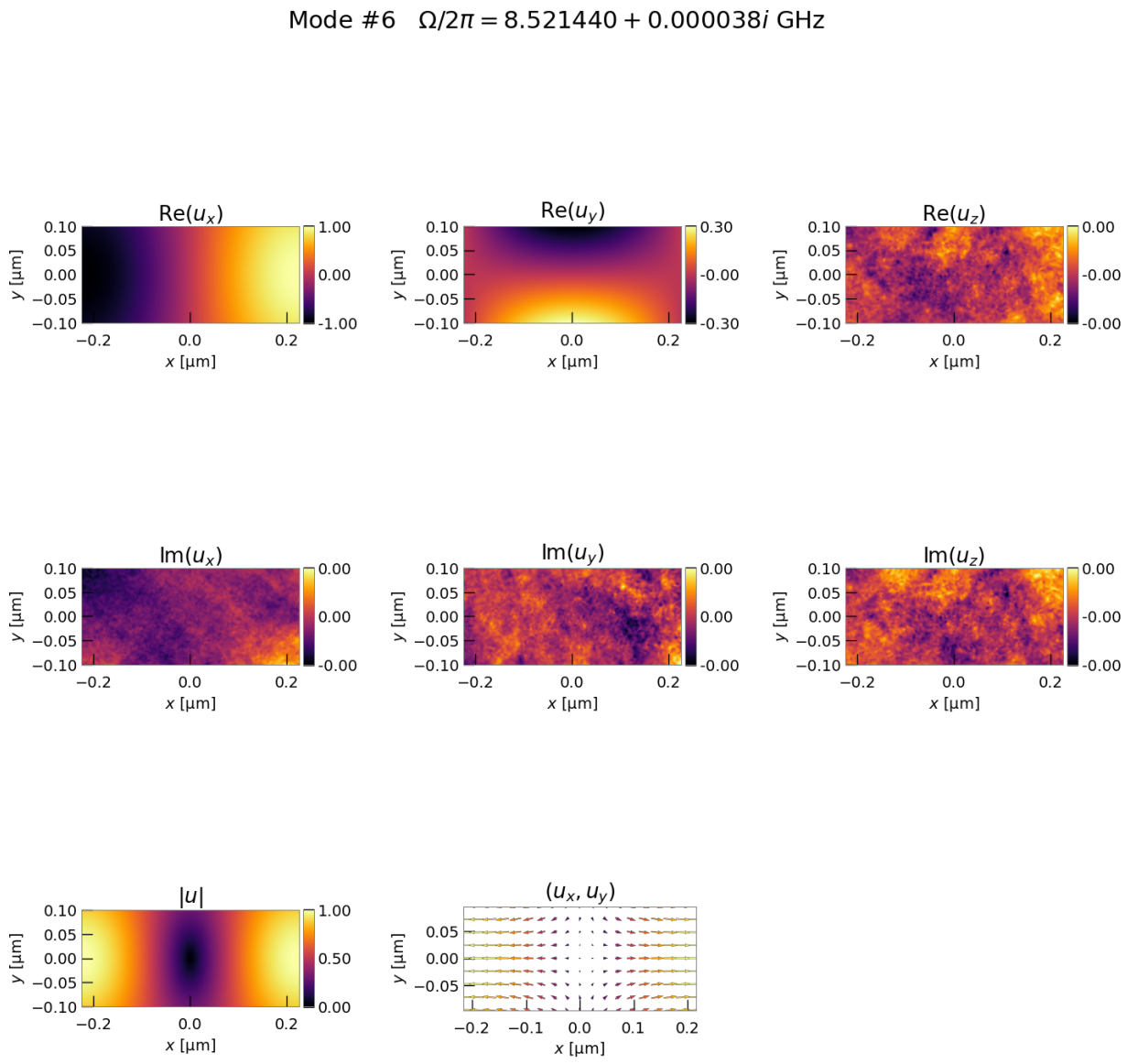


Fig. 4.39: Fundamental acoustic mode fields.

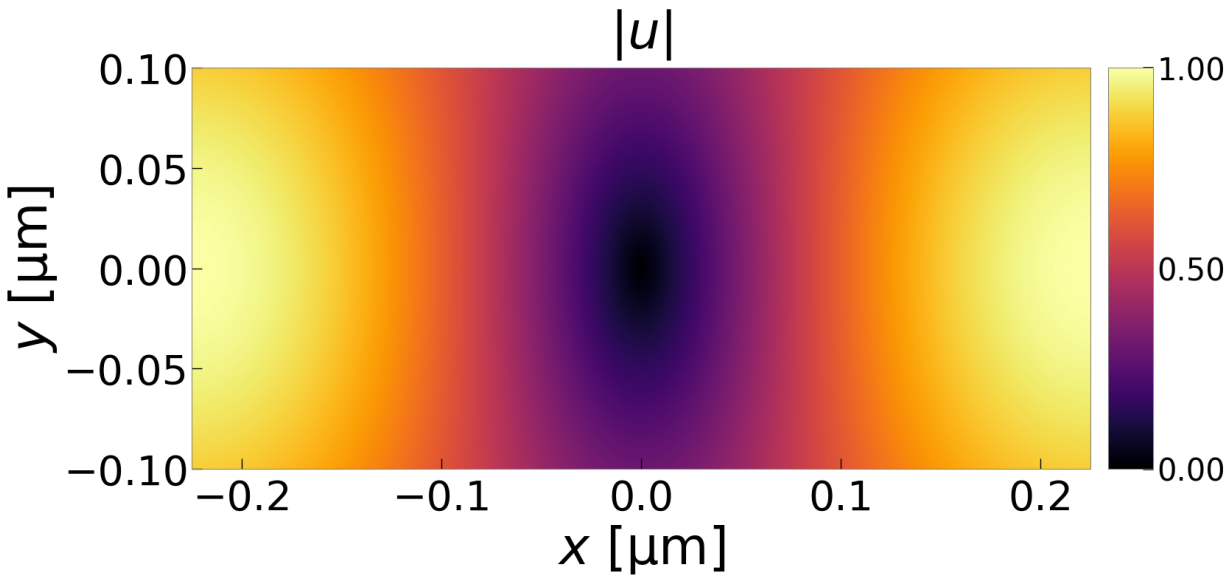


Fig. 4.40: Fundamental acoustic mode fields.

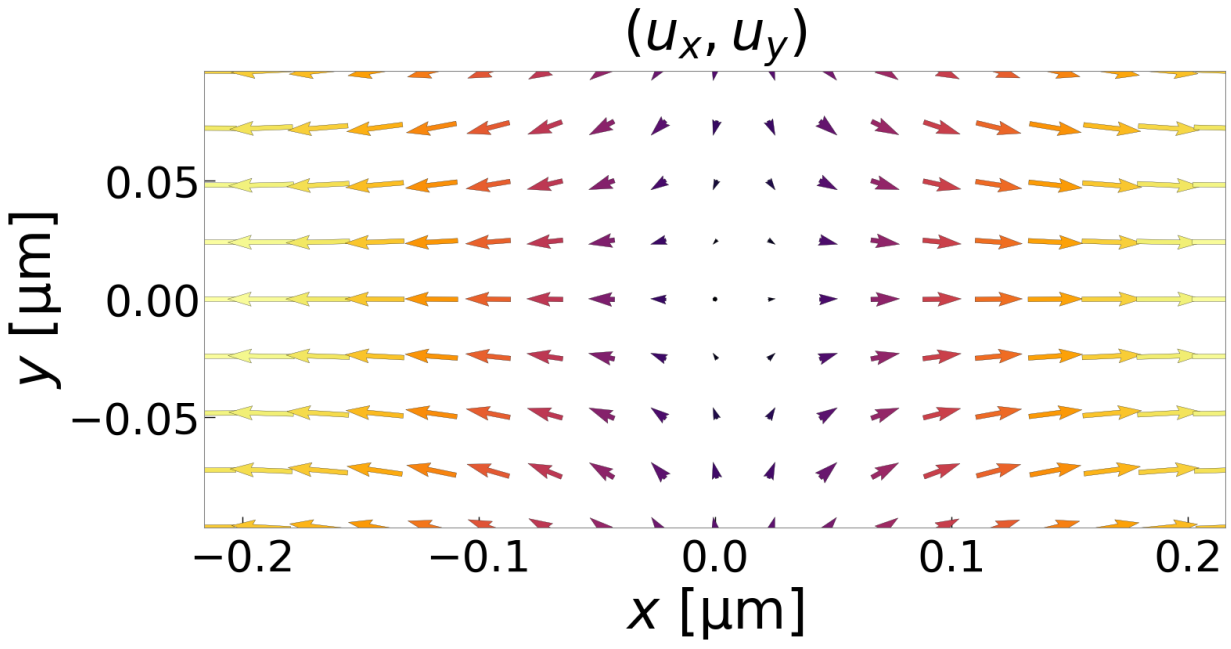


Fig. 4.41: Fundamental acoustic mode fields.

4.2.5 IFSBS - Circular Waveguide - Silica

```
print("\n Simulation time (sec.)", (end - start))
"""
Script to evaluate intermodal forward Brillouin scattering in a cylindrical SiO2␣
↪waveguide
"""

# Import the necessary packages
import time
import datetime
import numpy as np
import sys
import copy
from matplotlib.ticker import AutoMinorLocator
import math
sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from plotting import FieldDecorator
from fortran import NumBAT

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

start = time.time()

# Specify Geometric Parameters - all in [nm].
wl_nm = 1550 # Wavelength of EM wave in vacuum.
# Unit cell dimensions must be sufficiently large to ensure fields are zero at␣
↪outermost boundary.
unitcell_x = 4.01*wl_nm #be careful to ensure not whole integer multiples
unitcell_y = unitcell_x
inc_a_x = 1000 # Waveguide width.
inc_a_y = inc_a_x
inc_shape = 'circular' # Shape of the waveguide.

# Specify number of electromagnetic modes, acoustic modes, and which EM indices
# are involved in the calculation for intermodal FSBS
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100 # Number of acoustic modes to solve for.
# 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 = 1
# 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'

# Output files are generated in a folder with the following prefix
prefix_str = 'ifsbs-josab-1umSiO2'
```

```
# 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.get_material("Vacuum"),
                        material_a=materials.get_material("SiO2_2021_Poulton"),
                        lc_bkg=0.05, # mesh coarseness in background, larger lc_bkg =␣
↪coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger lc2 = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False) # note requires x-windows configuration to␣
↪work

# Initial guess for the EM effective index of the waveguide
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic Modes
print("Starting EM pump modes")
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff, debug=False)

print("Starting EM Stokes modes")
sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)

# Generate images for the EM modes involved in the calculation
print("Starting EM field plotting ")
plotting.plot_mode_fields(sim_EM_pump,
                          ivals=[EM_ival_pump,EM_ival_Stokes],
                          EM_AC='EM_E', num_ticks=3,xlim_min=0.2, xlim_max=0.2, ylim_
↪min=0.2, ylim_max=0.2,
                          prefix_str=prefix_str, pdf_png='png', ticks=True, quiver_
↪points=10,
                          comps=['Et','Eabs'], n_points=1000, colorbar=True)

# A computation interruption if needed
# sys.exit("We interrupt your regularly scheduled computation to bring you something␣
↪completely different... for now")

# 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))

# Calculate and print the acoustic wave vector
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)

# Calculate Acoustic Modes
print("Starting acoustic modes")
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, debug=False)

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

```
# Calculate total SBS gain, photoelastic and moving boundary contributions, as
# well as other important quantities
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)

# 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)

# Display these in terminal
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
# determining the location of the maximum gain
maxGainloc=6; #note sometimes its necessary to manually specify as certain values are␣
↪NOT possible by symmetry arguments

print("Plotting acoustic mode corresponding to maximum")
plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,␣
↪ivals=[maxGainloc],
                          num_ticks=3, quiver_points=40, pdf_png='png',ticks=True,␣
↪comps=['ut','uabs'], colorbar=True)

# Displaying results for the maximum found in the selection
print("-----------------")
print("Displaying results for maximum gain value found:")
print("Greatest SBS_gain  [1/(Wm)] total \n", masked.data[maxGainloc])
print("displaying corresponding acoustic mode number (i.e., AC_field_#) for reference␣
↪\n",maxGainloc )
print("EM Pump Power [Watts] \n", sim_EM_pump.EM_mode_power[EM_ival_pump] )
print("EM Stokes Power [Watts] \n", sim_EM_Stokes.EM_mode_power[EM_ival_Stokes] )
print("EM angular frequency [THz] \n", sim_EM_pump.omega_EM/1e12 )
print("AC Energy Density [J*m^{-1}] \n", sim_AC.AC_mode_energy_elastic[maxGainloc] )
print("AC loss alpha [1/s] \n", alpha[maxGainloc] )
print("AC frequency [GHz] \n", sim_AC.Omega_AC[maxGainloc]/(1e9*2*math.pi) )
print("AC linewidth [MHz] \n", linewidth_Hz[maxGainloc]/1e6)

#since the overlap is not returned directly we'll have to deduce it
absQtot2 = (alpha[maxGainloc]*sim_EM_pump.EM_mode_power[EM_ival_pump]*sim_EM_Stokes.
↪EM_mode_power[EM_ival_Stokes]*sim_AC.AC_mode_energy_elastic[maxGainloc]*masked.
↪data[maxGainloc])/(2*sim_EM_pump.omega_EM*sim_AC.Omega_AC[maxGainloc]);
absQtot = pow(absQtot2,1/2)
print("Total coupling |Qtot| [W*m^{-1}*s] \n", absQtot )

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

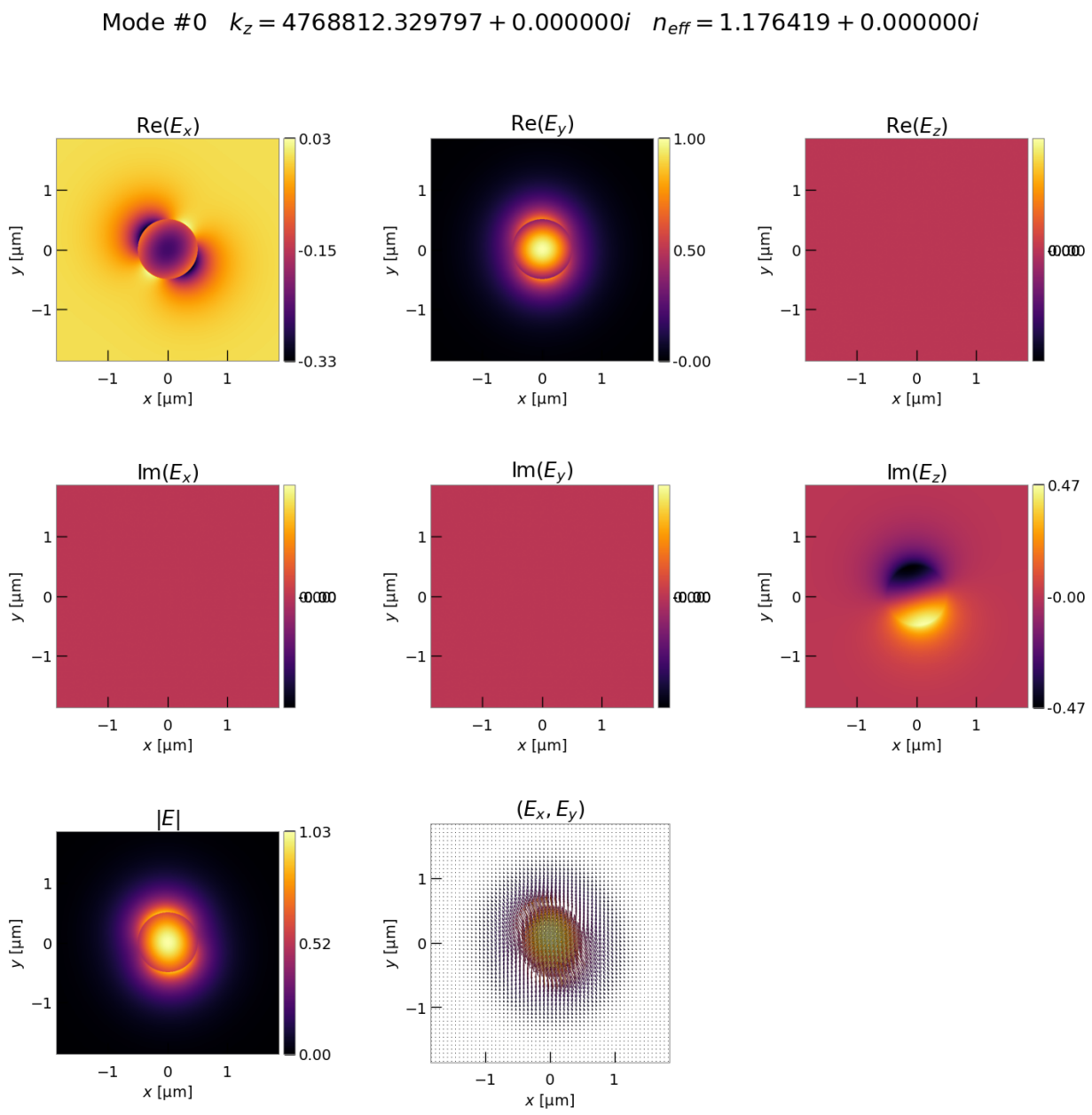


Fig. 4.42: Fundamental optical mode fields.

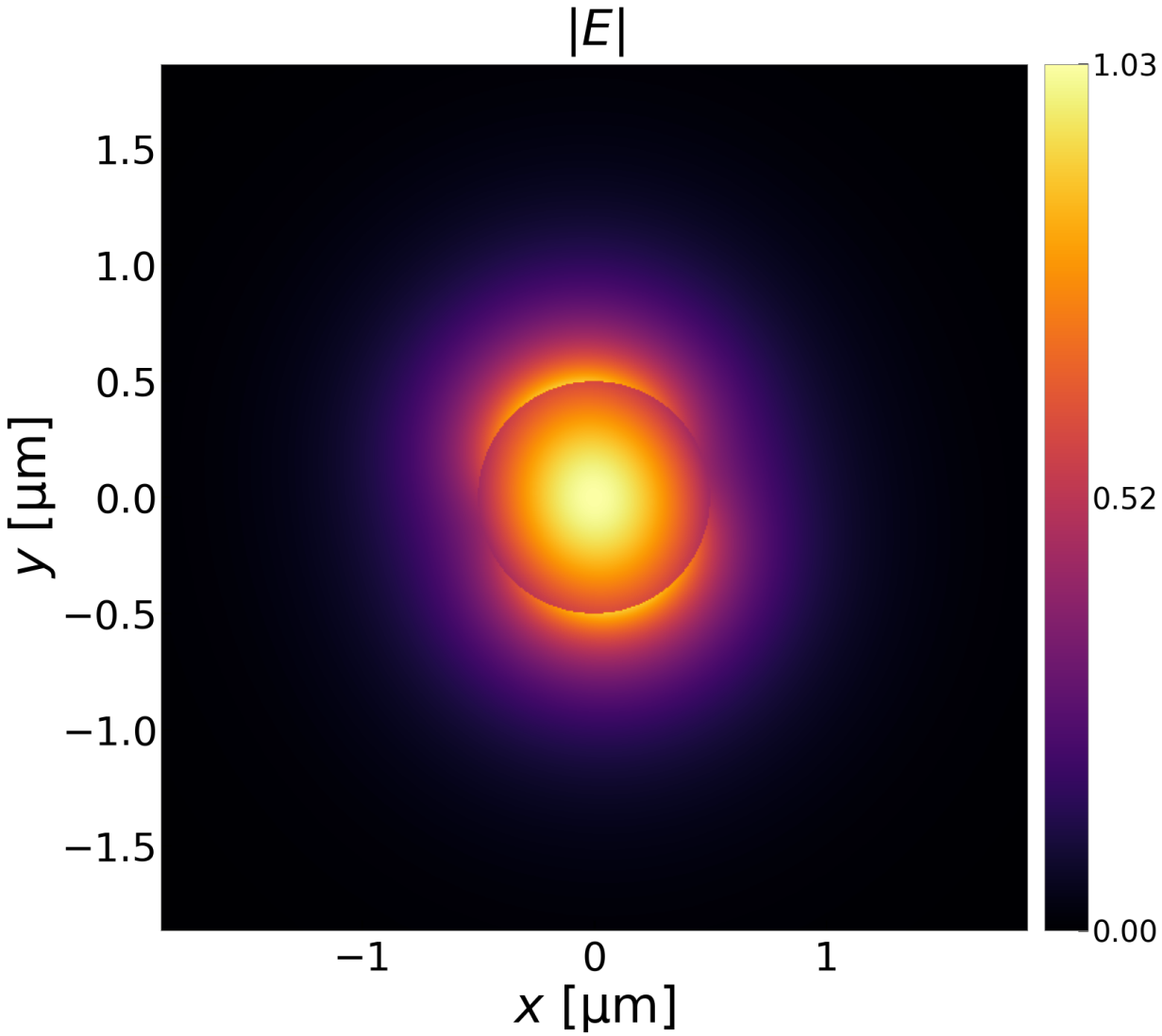


Fig. 4.43: Fundamental optical mode fields.

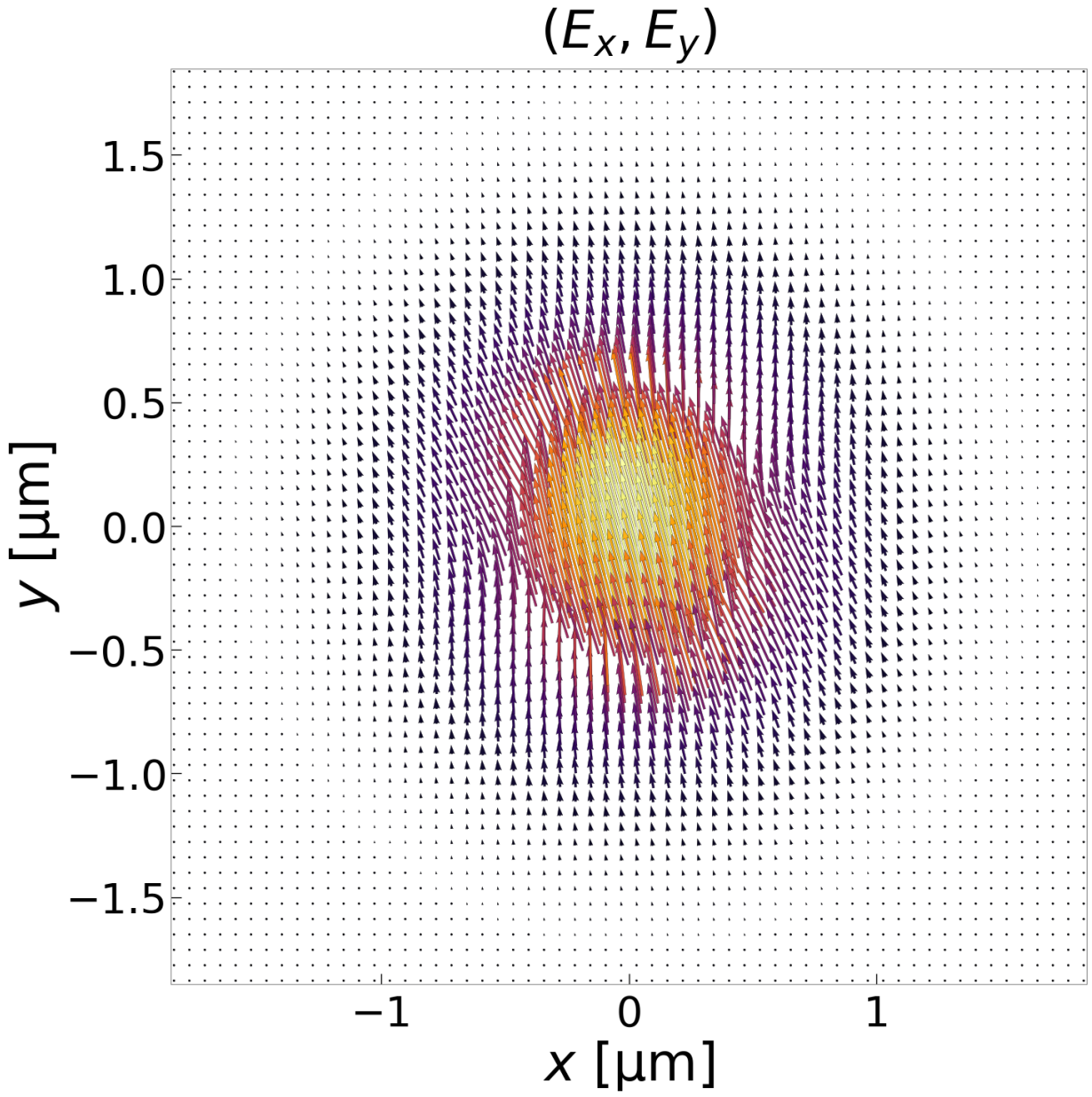


Fig. 4.44: Fundamental optical mode fields.

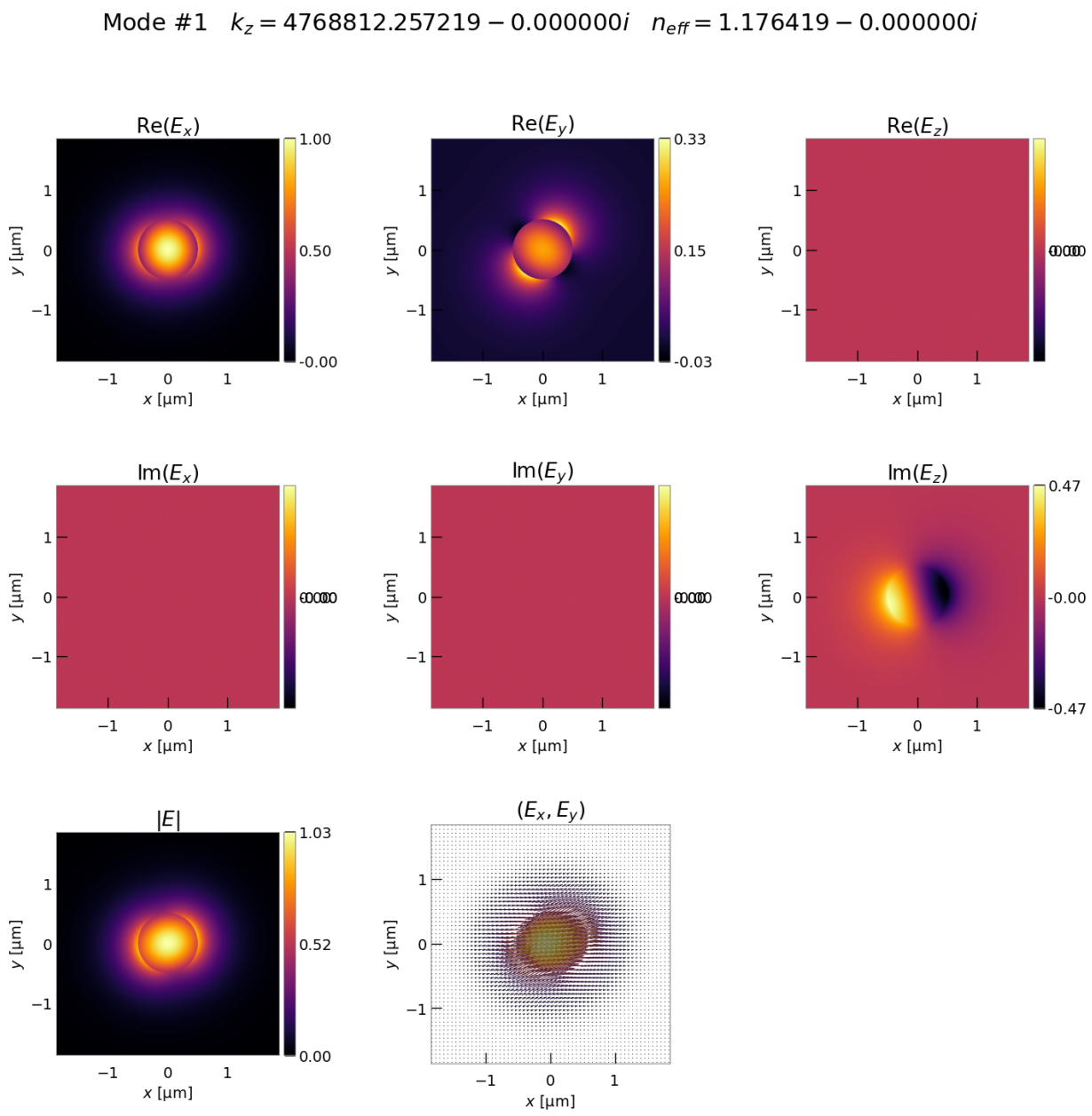


Fig. 4.45: Second order optical mode fields.

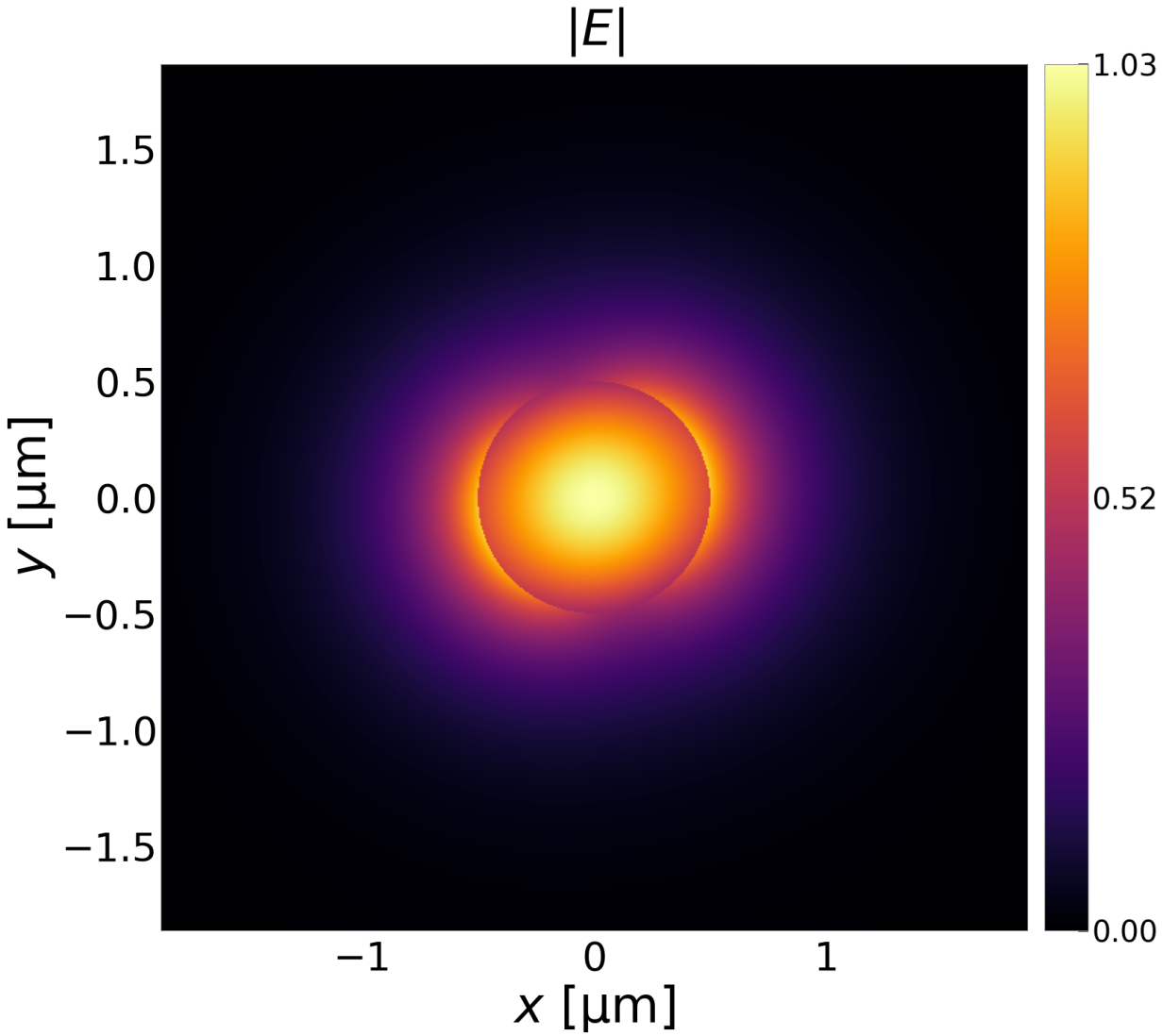


Fig. 4.46: Second order optical mode fields.

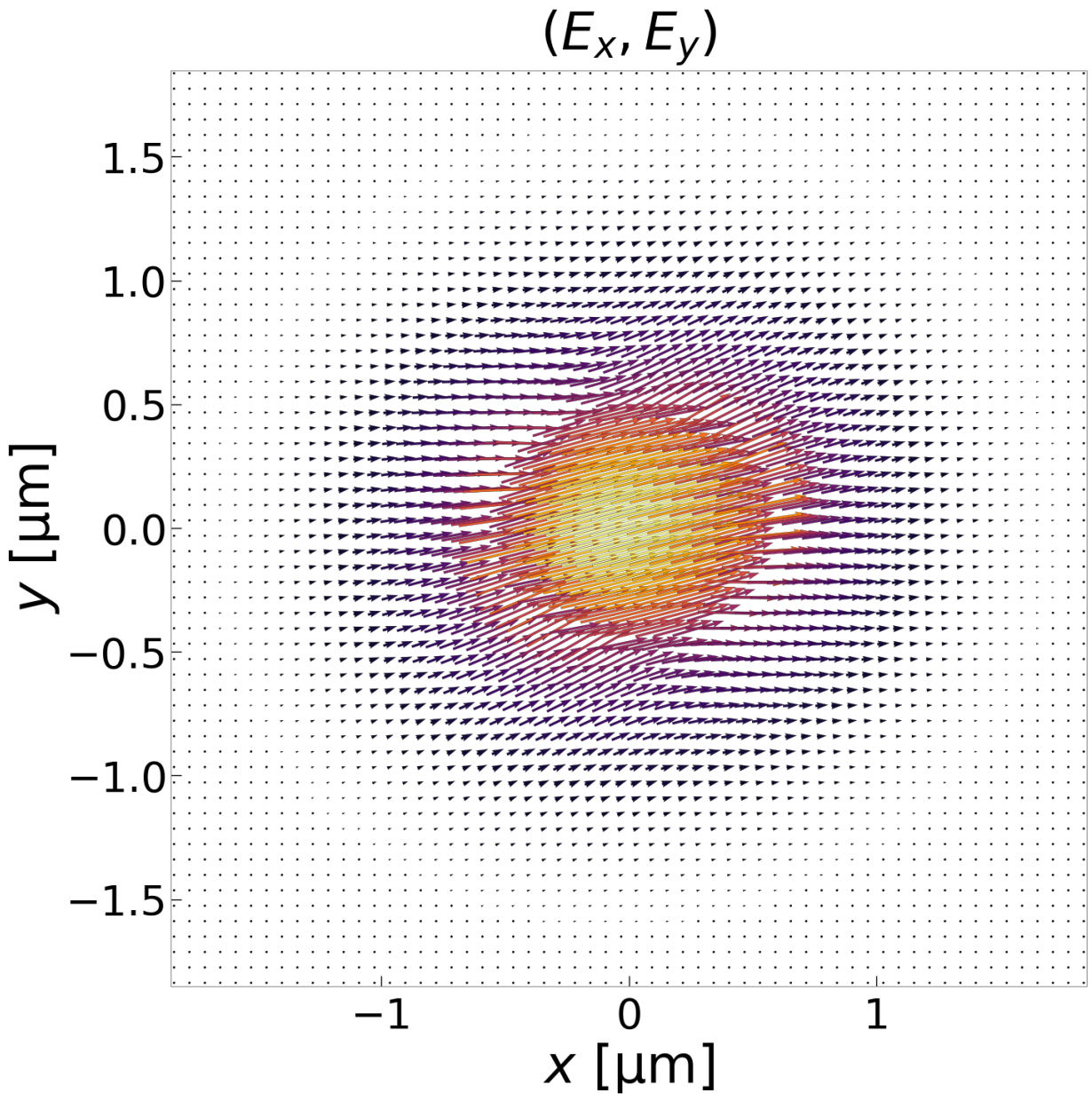


Fig. 4.47: Second order optical mode fields.

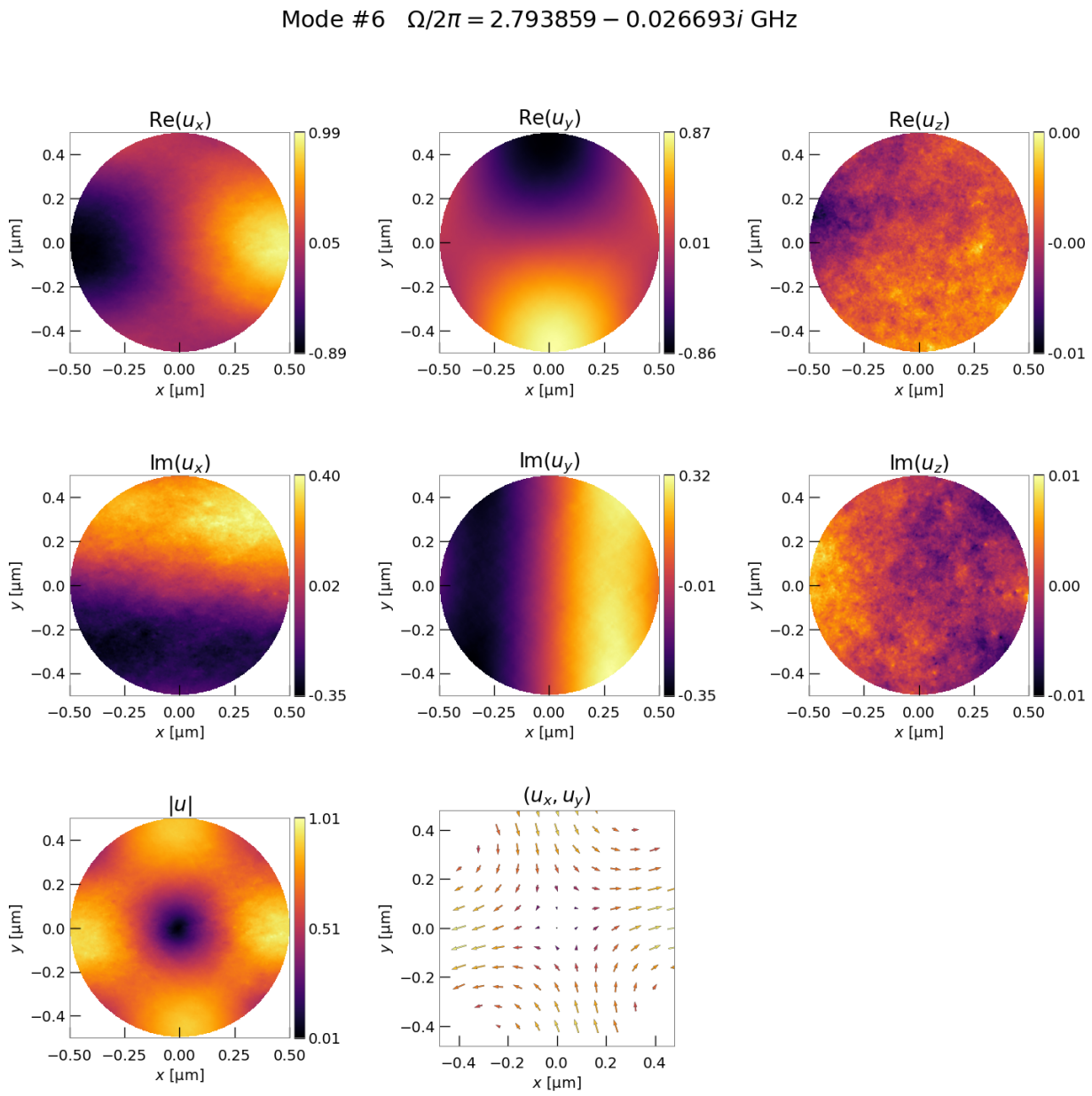


Fig. 4.48: Fundamental acoustic mode fields.

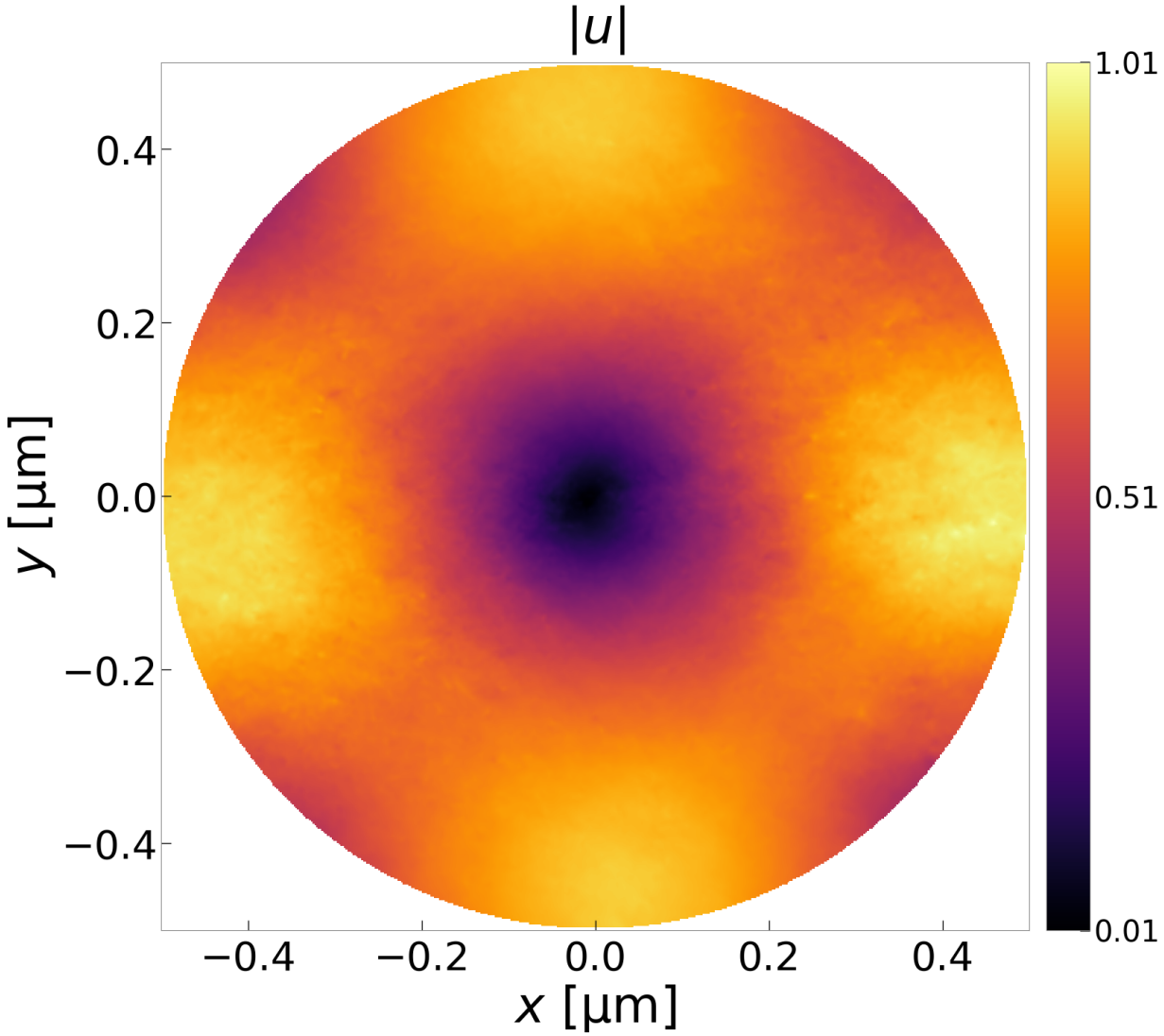


Fig. 4.49: Fundamental acoustic mode fields.

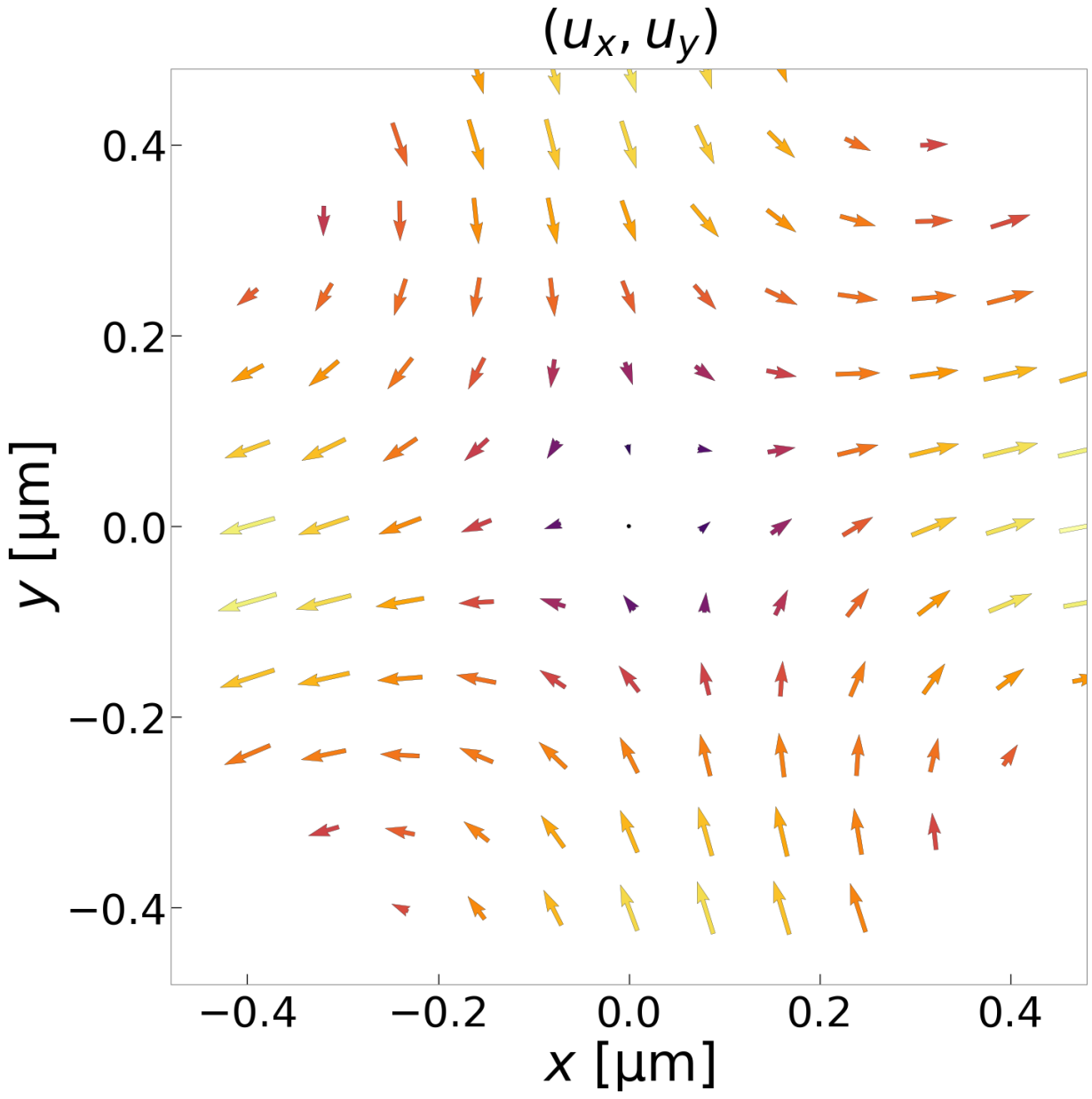


Fig. 4.50: Fundamental acoustic mode fields.

4.2.6 IFSBS - Rectangular Waveguide - Silicon

```
print("\n Simulation time (sec.)", (end - start))
"""
Script to evaluate intermodal forward Brillouin scattering in a rectangular Si␣
↪waveguide
"""

# Import the necessary packages
import time
import datetime
import numpy as np
import sys
import copy
from matplotlib.ticker import AutoMinorLocator
import math
sys.path.append("../backend/")
import materials
import objects
import mode_calcs
import integration
import plotting
from plotting import FieldDecorator
from fortran import NumBAT

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

start = time.time()

# Specify Geometric Parameters - all in [nm].
wl_nm = 1550
unitcell_x = 3.01*wl_nm
unitcell_y = unitcell_x
inc_a_x = 450 # Waveguide widths.
inc_a_y = 200
inc_shape = 'rectangular'

# Specify number of electromagnetic modes, acoustic modes, and which EM indices
# are involved in the calculation for intermodal FSBS
num_modes_EM_pump = 20
num_modes_EM_Stokes = num_modes_EM_pump
num_modes_AC = 100 # Number of acoustic modes to solve for.
# 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
# The AC mode(s) for which to calculate interaction with EM modes.
AC_ival = 'All'

# Output files are generated in a folder with the following prefix
prefix_str = 'ifsbs-josab-450x200nmSi'

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("Si_2021_Poulton"),
                        lc_bkg=0.05,  # mesh coarseness in background, larger lc_bkg␣
↪= coarser along horizontal outer edge
                        lc_refine_1=20.0, # mesh refinement factor near the interface␣
↪of waveguide, larger = finer along horizontal interface
                        lc_refine_2=30.0, # mesh refinement factor near the origin/
↪centre of waveguide
                        plt_mesh=False, # creates png file of geometry and mesh in␣
↪backend/fortran/msh/
                        check_mesh=False)   # note requires x-windows configuration␣
↪to work

# Initial guess for the EM effective index of the waveguide
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic Modes
print("Starting EM pump modes")
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff, debug=False)

print("Starting EM Stokes modes")
sim_EM_Stokes = mode_calcs.fwd_Stokes_modes(sim_EM_pump)

# Generate images for the EM modes involved in the calculation
print("Plotting EM fields ")
# print("no plotting")
plotting.plot_mode_fields(sim_EM_pump,
                         ivals=[EM_ival_pump,EM_ival_Stokes],
                         EM_AC='EM_E', num_ticks=3,xlim_min=0.4, xlim_max=0.4, ylim_
↪min=0.4, ylim_max=0.4,
                         prefix_str=prefix_str, pdf_png='png', ticks=True, quiver_
↪points=10,
                         comps=['Et','Eabs'], n_points=1000, colorbar=True)

# A computation interruption if needed
# sys.exit("We interrupt your regularly scheduled computation to bring you something␣
↪completely different... for now")

# 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))

# Calculate and print the acoustic wave vector
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)

# Calculate Acoustic Modes
print("starting acoustic modes")
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, debug=False)

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

```
# Calculate total SBS gain, photoelastic and moving boundary contributions, as
# well as other important quantities
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)

# 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)


# Display these in terminal
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)
#determining the location of the maximum gain
maxGainloc=np.argmax(abs(masked.data)) ;

print("Plotting acoustic mode corresponding to maximum")
plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,␣
↪ivals=[maxGainloc],
                          num_ticks=3, quiver_points=40, pdf_png='png',ticks=True,␣
↪comps=['ut','uabs'], colorbar=True)

# Displaying results for the maximum found in the selection
print("-----------------")
print("Displaying results for maximum gain value found:")
maxGainloc=np.argmax(abs(masked.data)) ;
print("Greatest SBS_gain  [1/(Wm)] total \n", masked.data[maxGainloc])
print("displaying corresponding acoustic mode number (i.e., AC_field_#) for reference␣
↪\n",maxGainloc )
print("EM Pump Power [Watts] \n", sim_EM_pump.EM_mode_power[EM_ival_pump] )
print("EM Stokes Power [Watts] \n", sim_EM_Stokes.EM_mode_power[EM_ival_Stokes] )
print("EM angular frequency [THz] \n", sim_EM_pump.omega_EM/1e12 )
print("AC Energy Density [J*m^{-1}] \n", sim_AC.AC_mode_energy_elastic[maxGainloc] )
print("AC loss alpha [1/s] \n", alpha[maxGainloc] )
print("AC frequency [GHz] \n", sim_AC.Omega_AC[maxGainloc]/(1e9*2*math.pi) )
print("AC linewidth [MHz] \n", linewidth_Hz[maxGainloc]/1e6)

#since the overlap is not returned directly we'll have to deduce it
absQtot2 = (alpha[maxGainloc]*sim_EM_pump.EM_mode_power[EM_ival_pump]*sim_EM_Stokes.
↪EM_mode_power[EM_ival_Stokes]*sim_AC.AC_mode_energy_elastic[maxGainloc]*masked.
↪data[maxGainloc])/(2*sim_EM_pump.omega_EM*sim_AC.Omega_AC[maxGainloc]);
absQtot = pow(absQtot2,1/2)
print("Total coupling |Qtot| [W*m^{-1}*s] \n", absQtot )

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

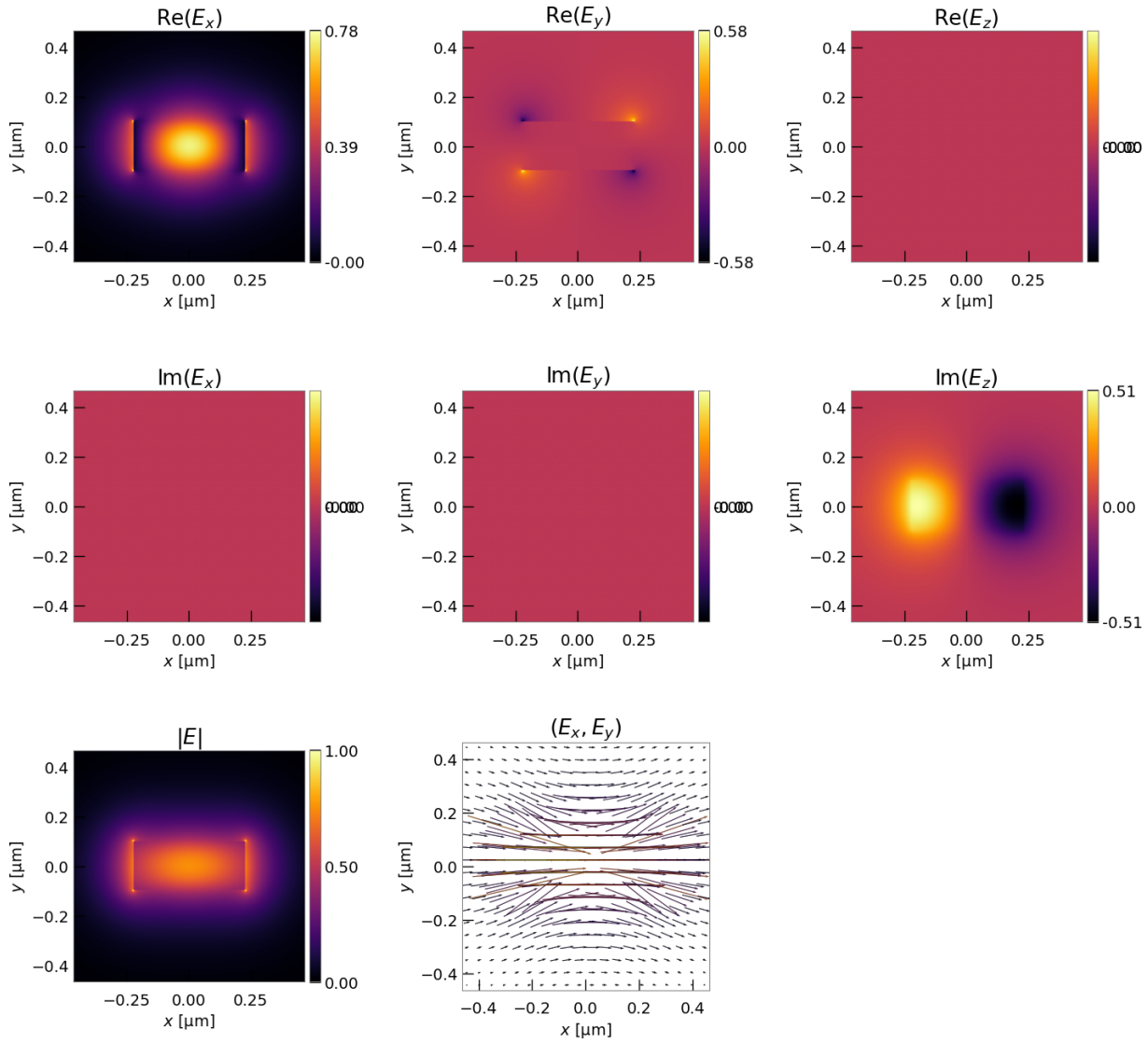


Fig. 4.51: Fundamental optical mode fields.

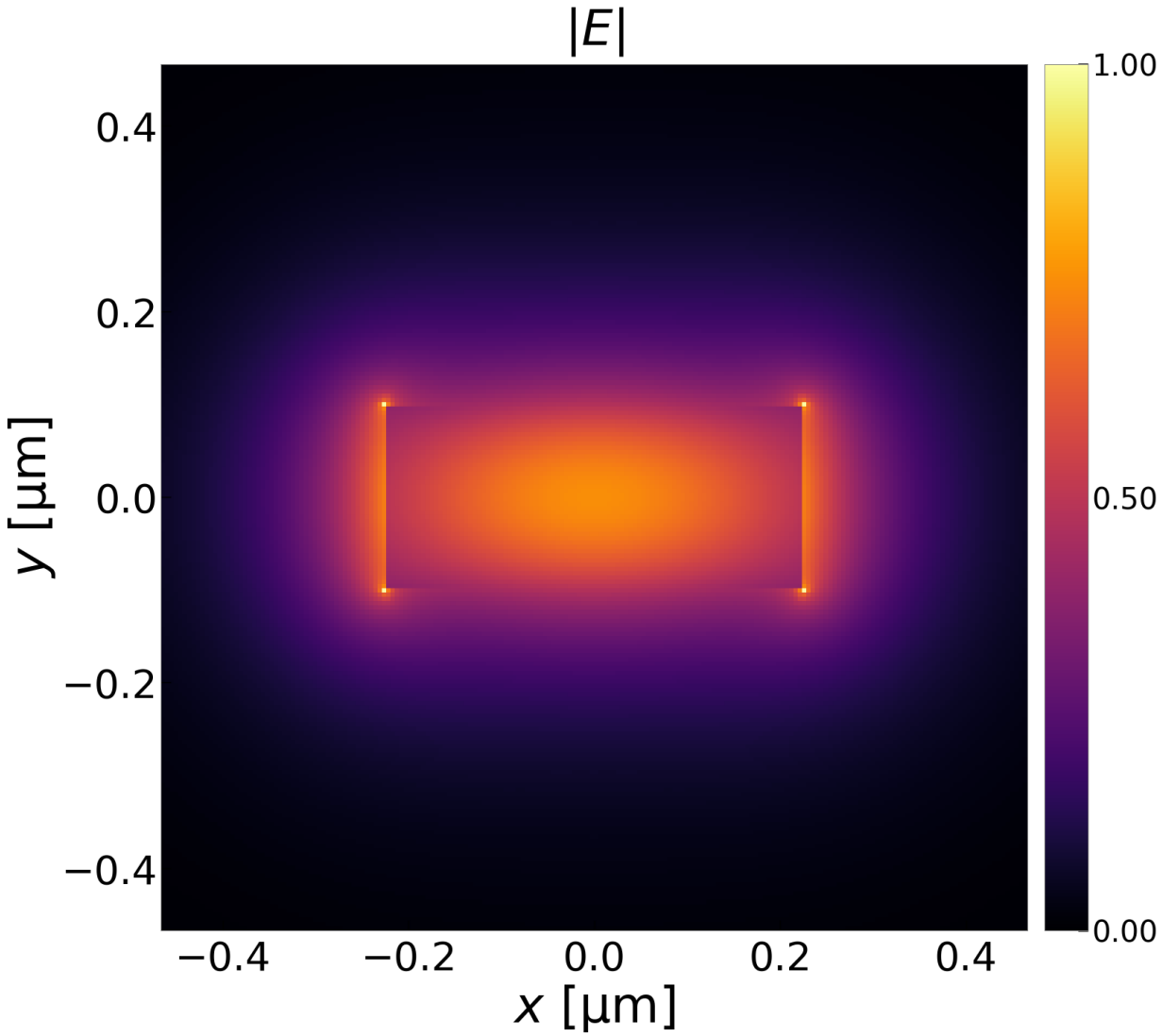


Fig. 4.52: Fundamental optical mode fields.

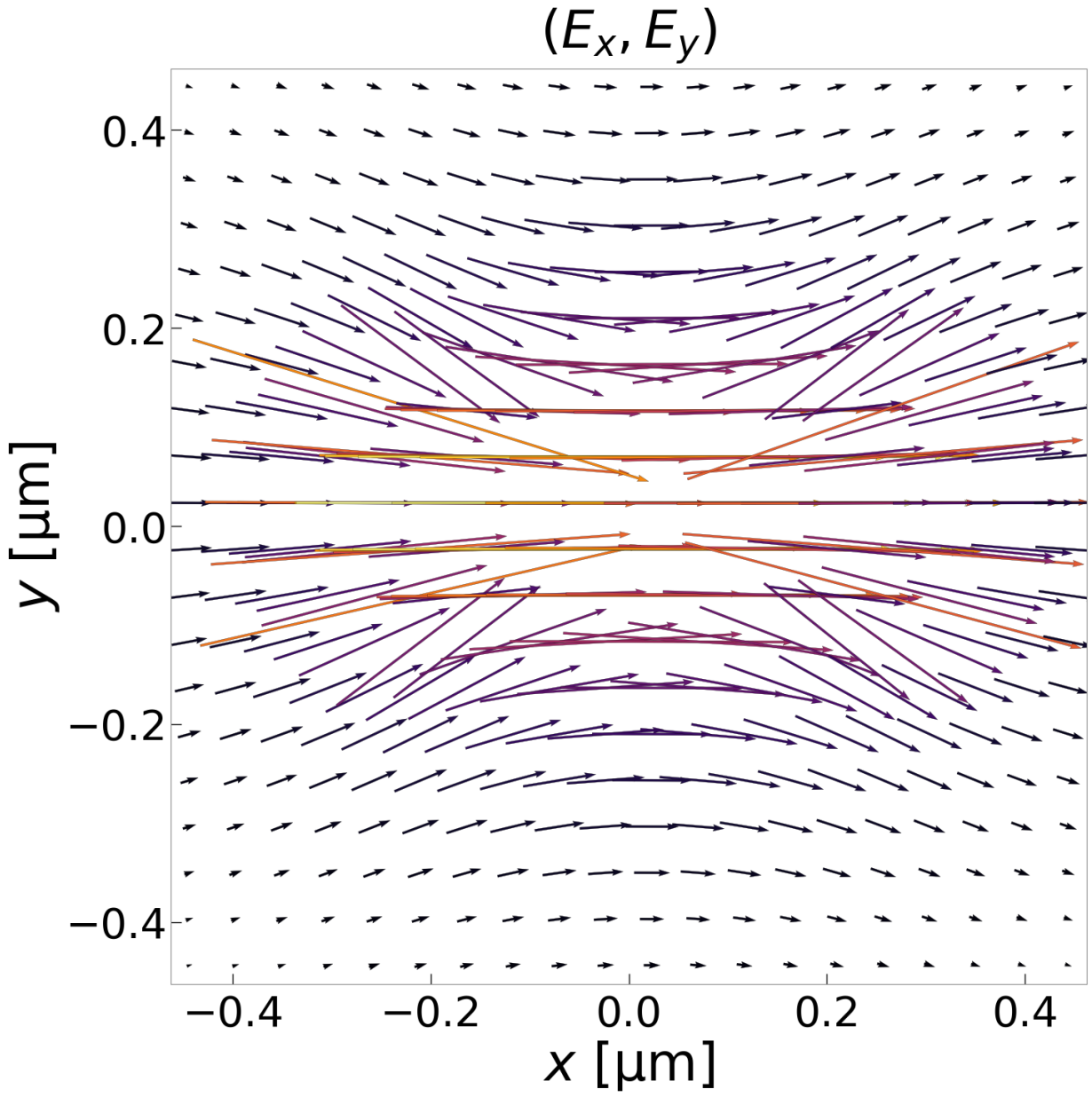


Fig. 4.53: Fundamental optical mode fields.

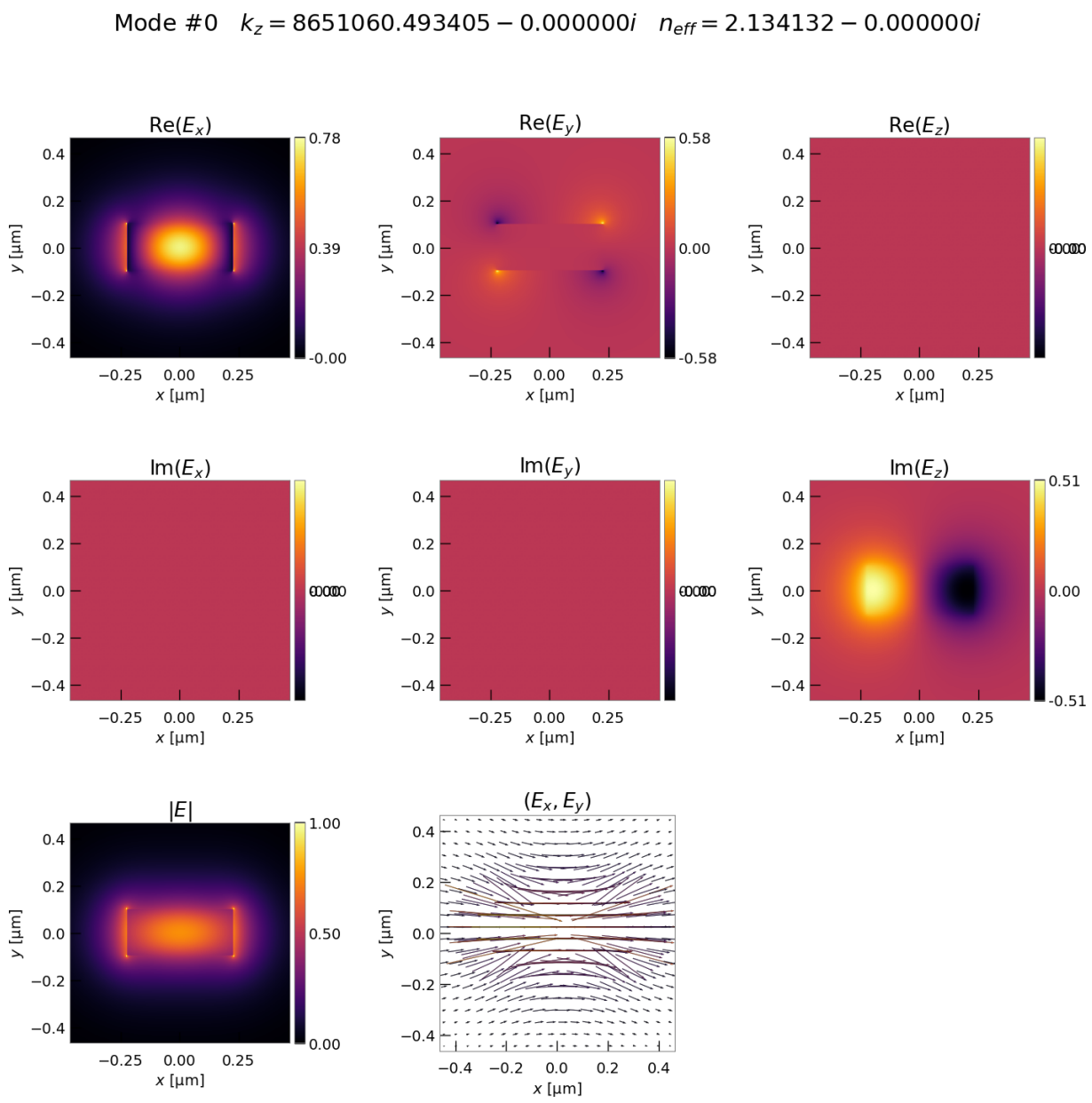


Fig. 4.54: Second order optical mode fields.

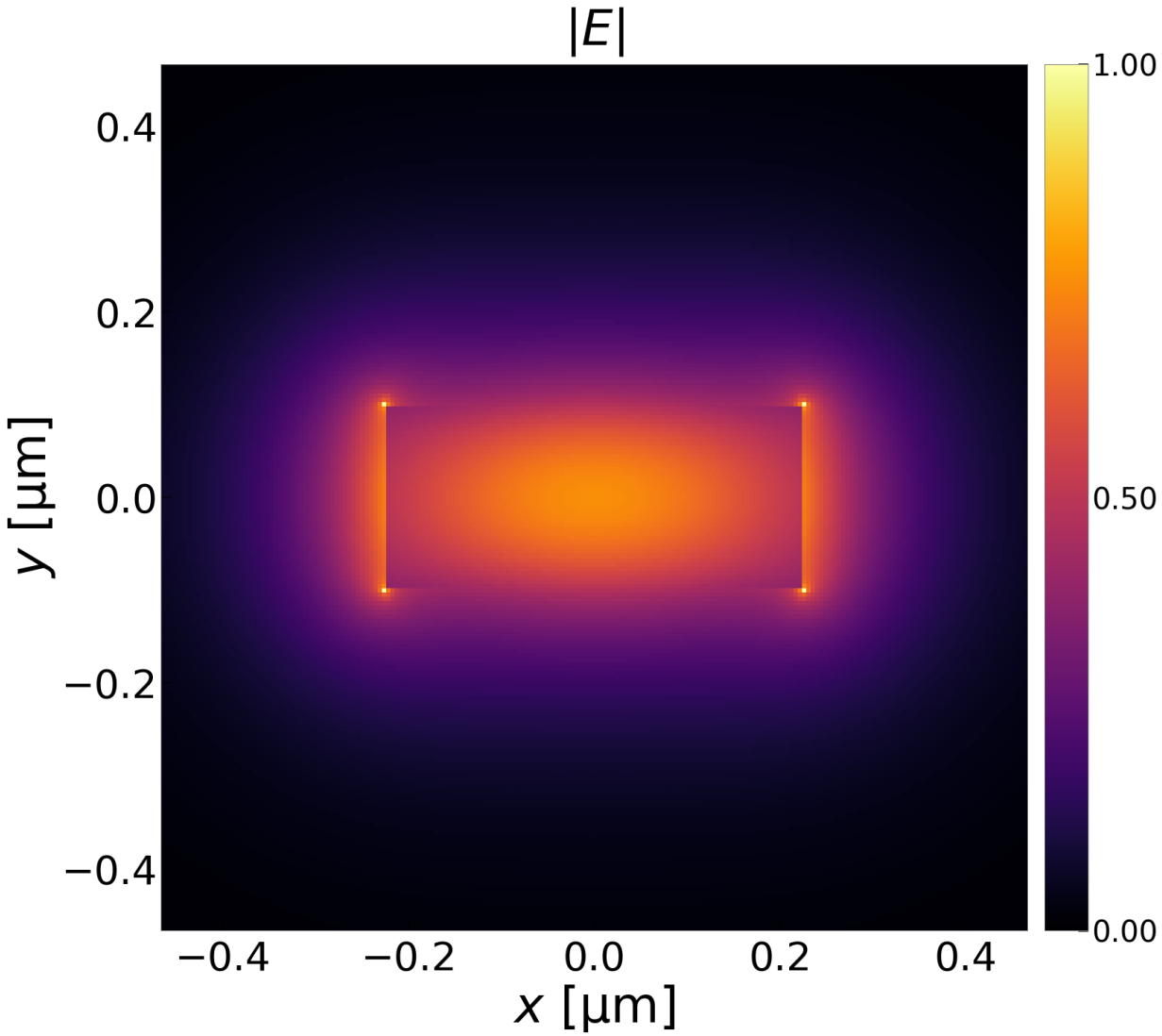


Fig. 4.55: Second order optical mode fields.

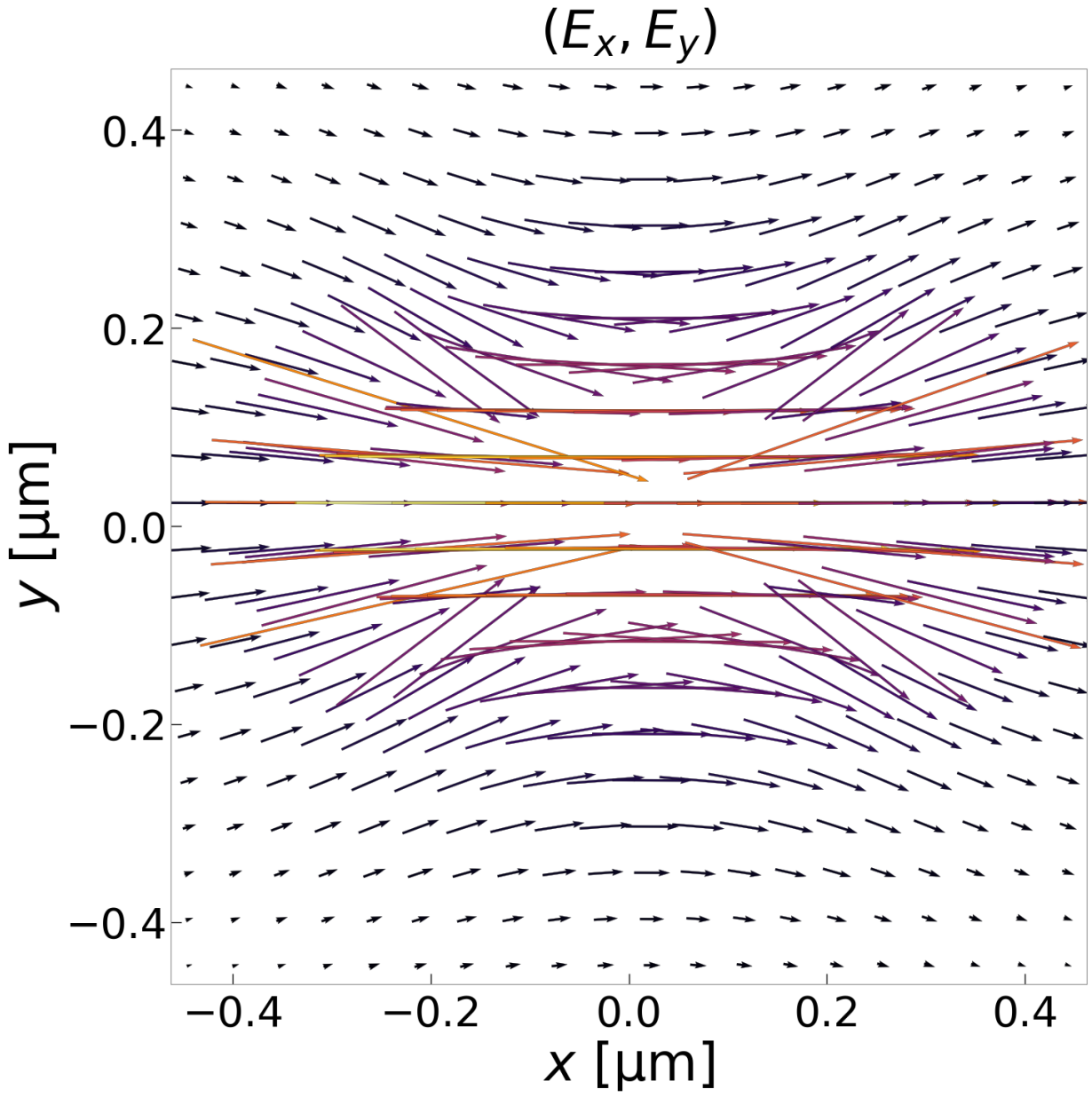


Fig. 4.56: Second order optical mode fields.

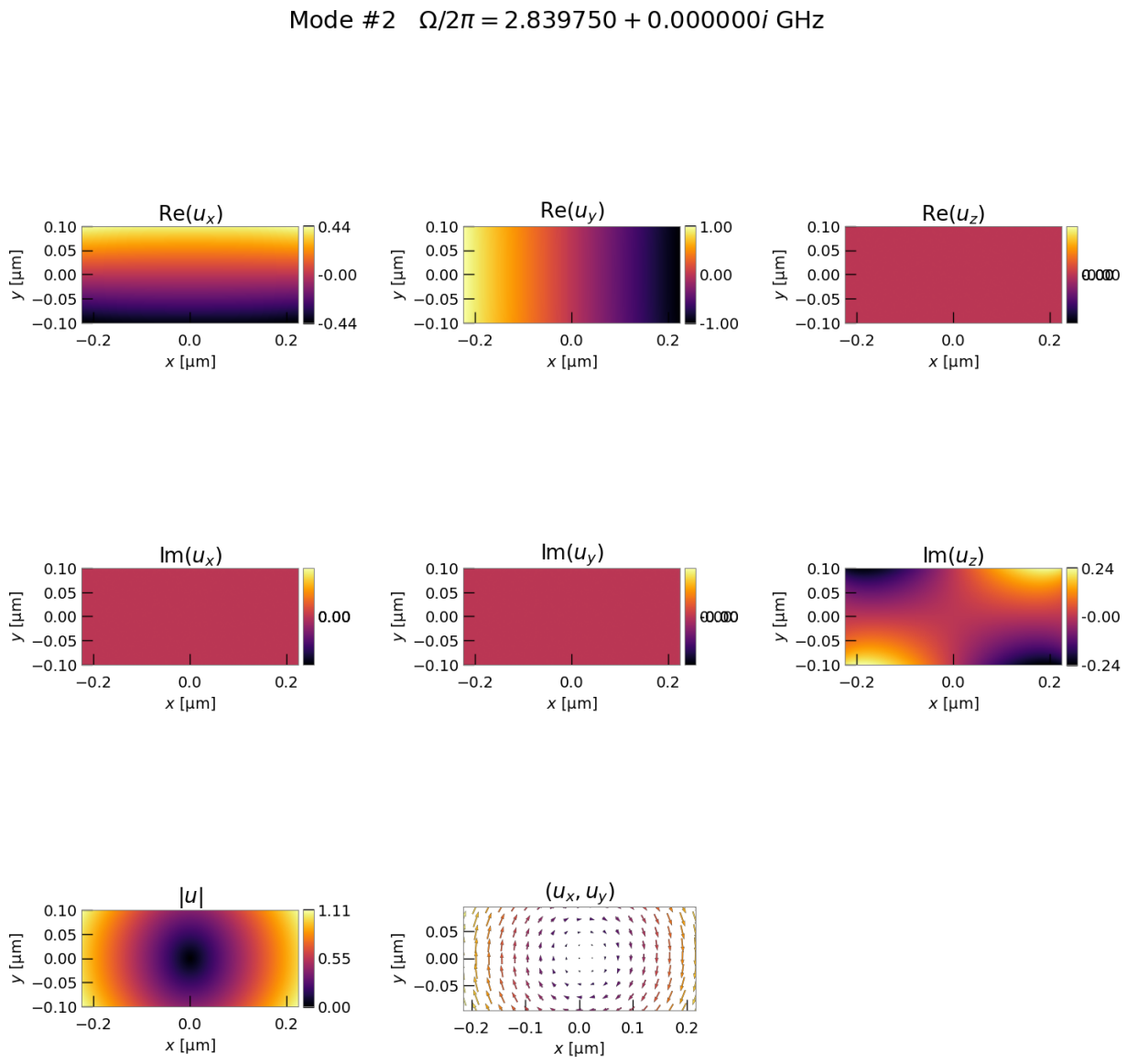


Fig. 4.57: Fundamental acoustic mode fields.

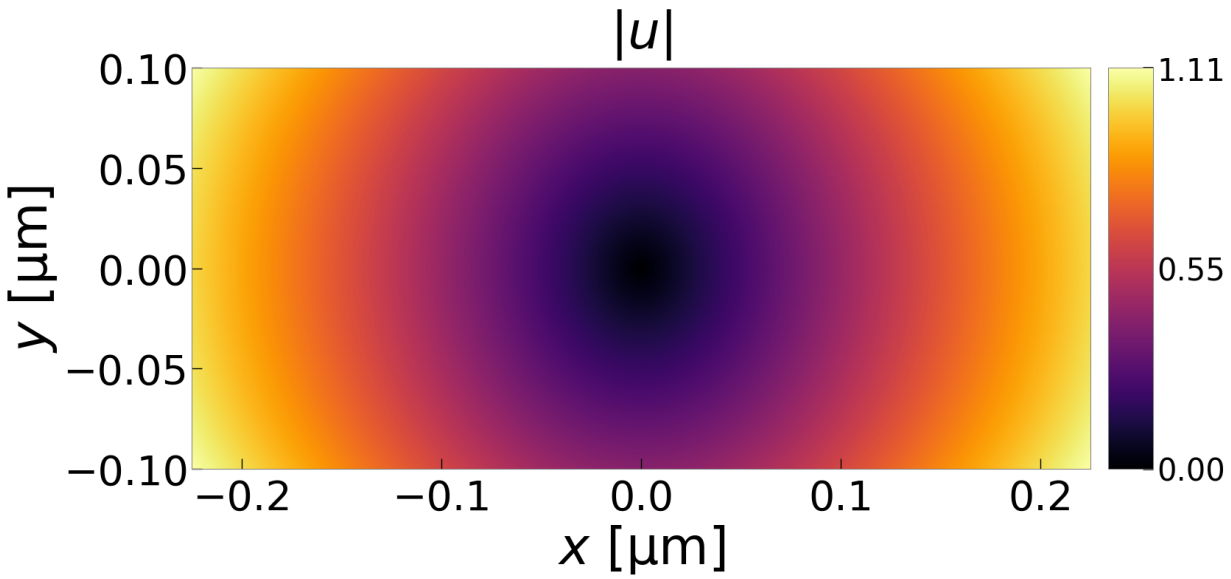


Fig. 4.58: Fundamental acoustic mode fields.

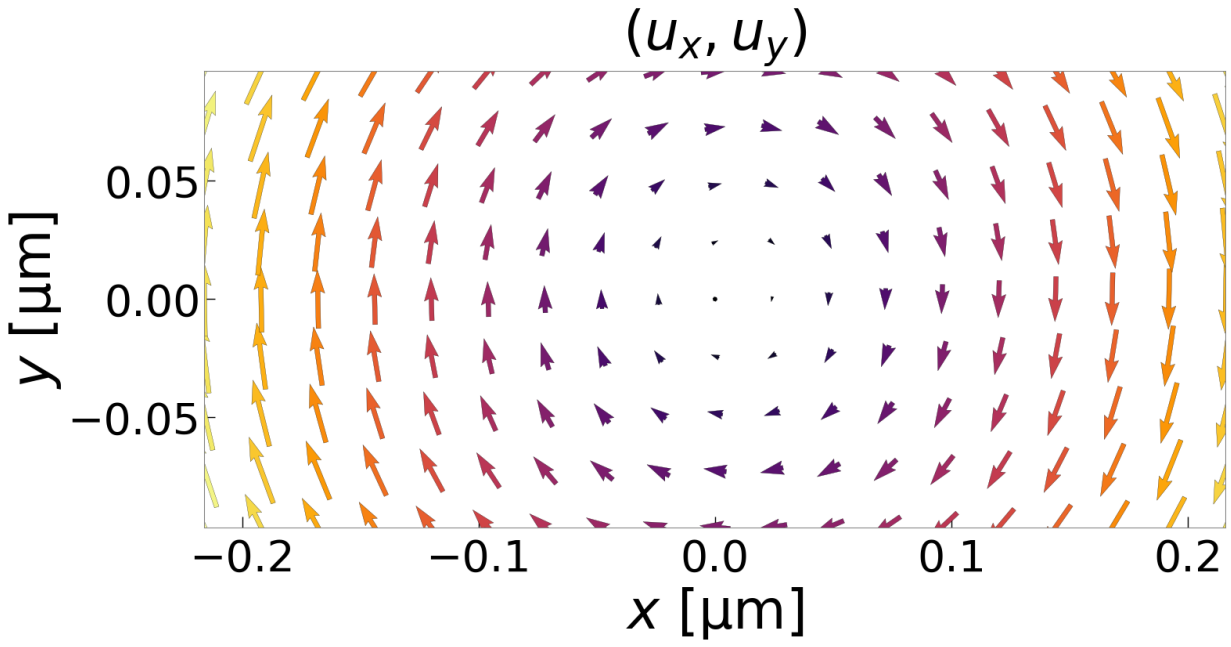


Fig. 4.59: Fundamental acoustic mode fields.

4.3 Literature Examples

Having become somewhat familiar with NumBAT, we now set out to replicate a number of examples from the recent literature located in the `lit_examples` directory. The examples are presented in chronological order. We note the particular importance of examples 5-8 which include experimental and numerical results that are in good agreement.

4.3.1 LitEx 1 – Laude and Beugnot, *AIP Advances* (2013): BSBS in a silica rectangular waveguide

This example `simo-lit_01-Laude-AIPAdv_2013-silica.py` is based on the calculation of backward SBS in a small rectangular silica waveguide described in V. Laude and J.-C. Beugnot, Generation of phonons from electrostriction in small-core optical waveguides, *AIP Advances* **3**, 042109 (2013).

Observe the use of a material named `materials.materials_dict["SiO2_2013_Laude"]` specifically modelled on the parameters in this paper. This technique allows users to easily compare exactly to other authors without changing their preferred material values for their own samples and experiments.

```
""" 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.get_material("Vacuum"),
                        material_a=materials.get_material("SiO2_2013_Laude"),
                        lc_bkg=1, lc_refine_1=400.0, lc_refine_2=50.0)

# 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.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ivals=[EM_ival_
↪pump],
                          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.plot_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, pdf_png='png')

# 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', pdf_png='png')

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

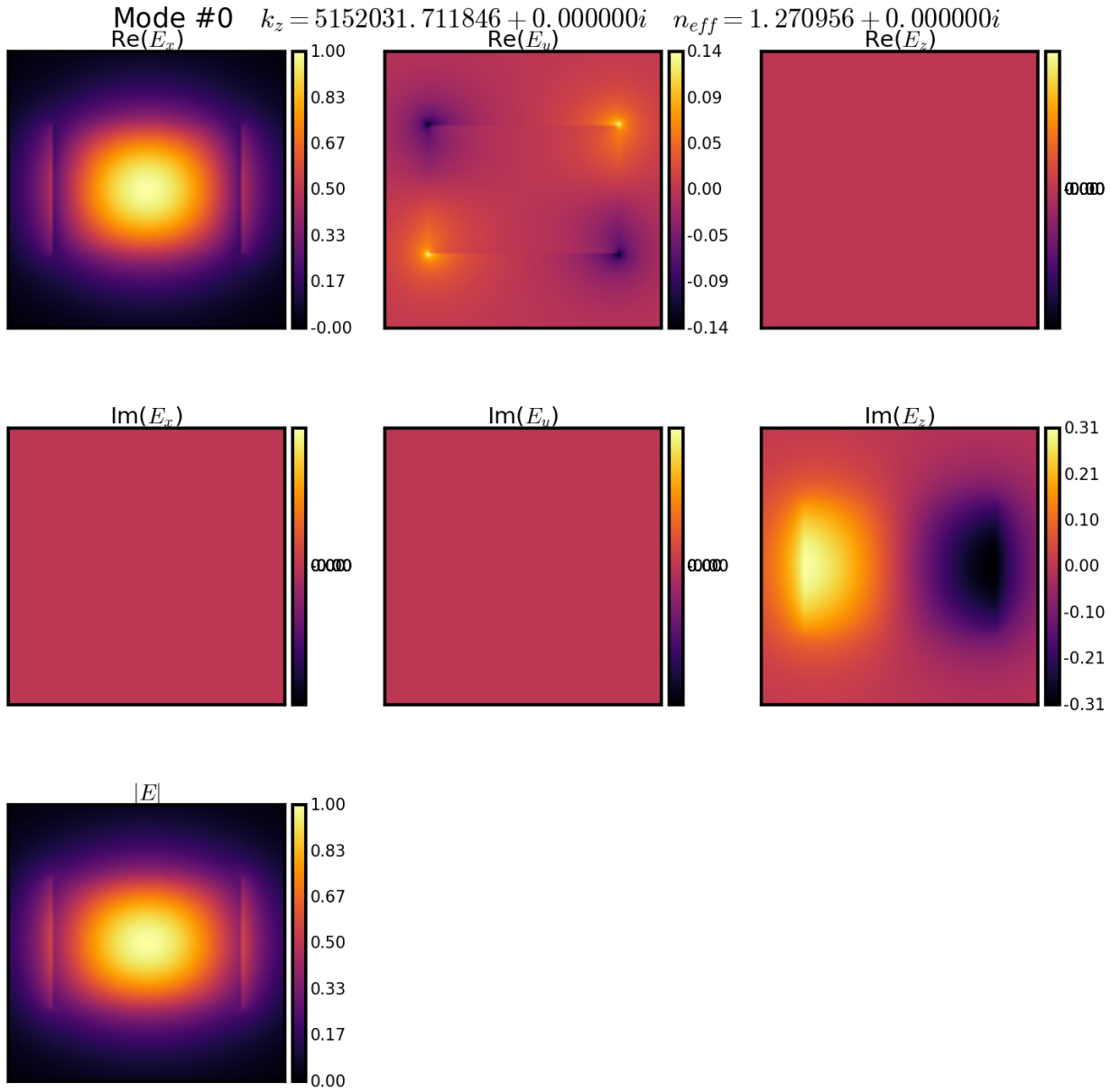


Fig. 4.60: Fundamental optical mode fields.

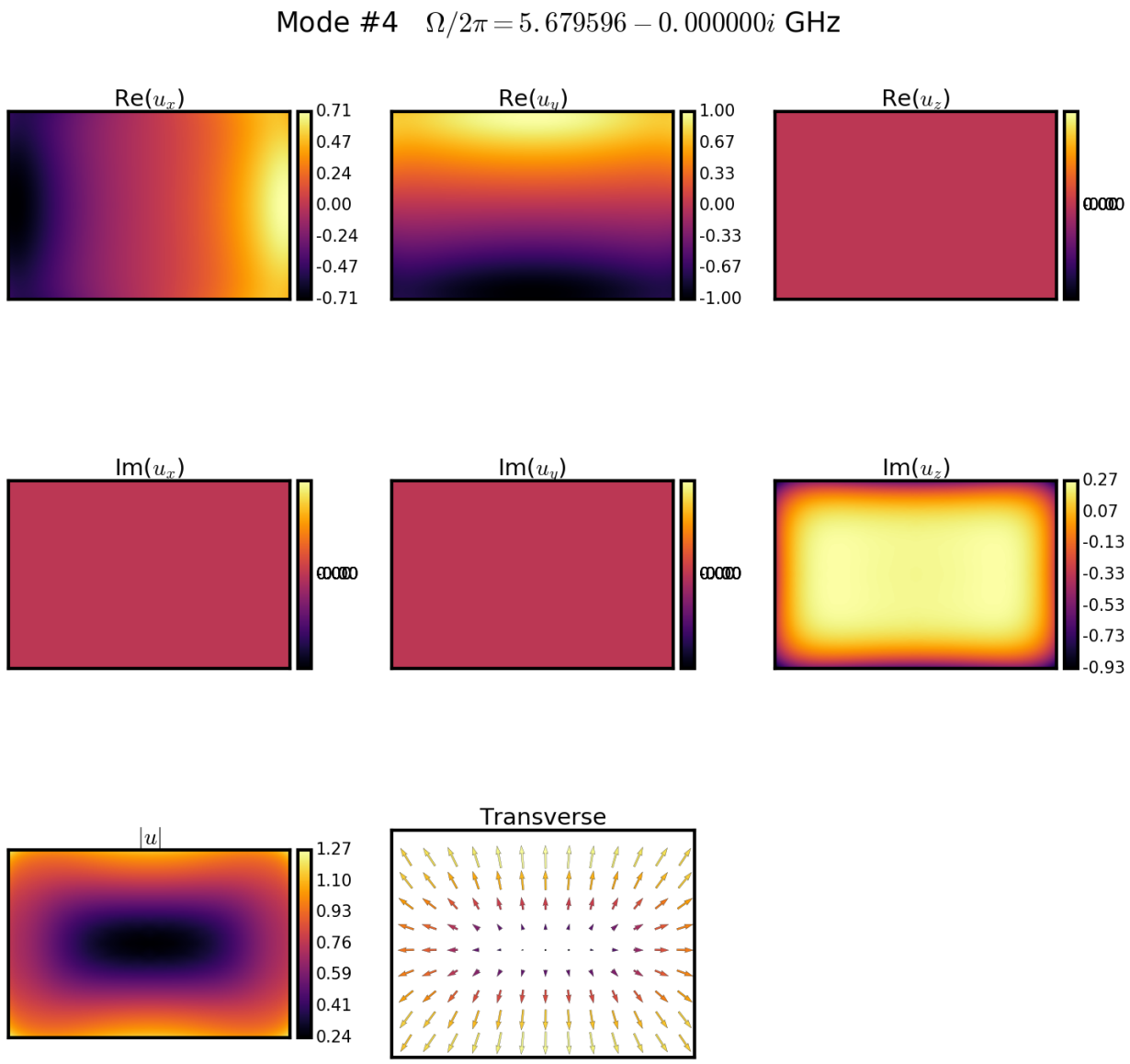


Fig. 4.61: High gain acoustic mode, marked as C in paper.

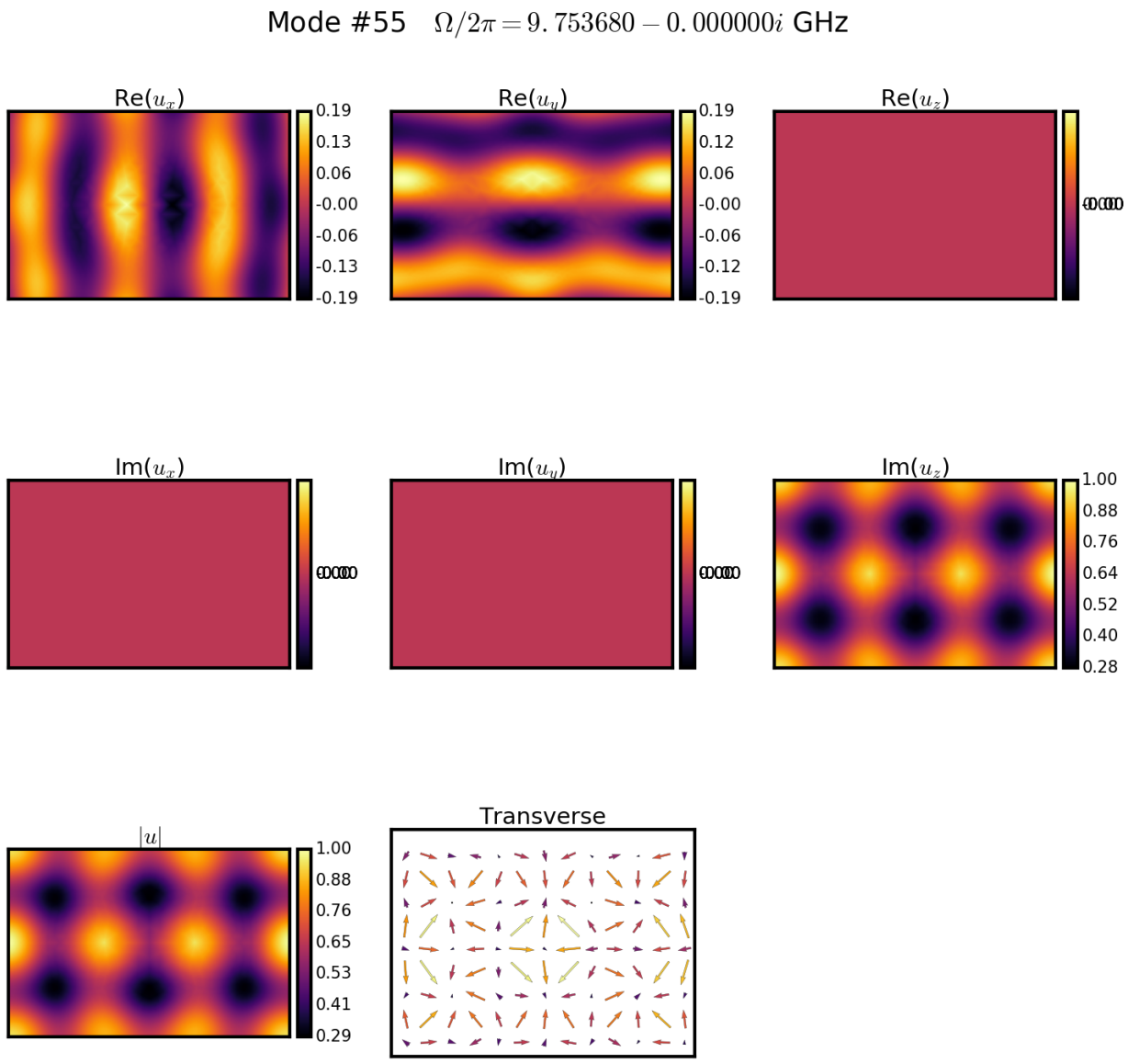


Fig. 4.62: High gain acoustic mode, marked as D in paper.

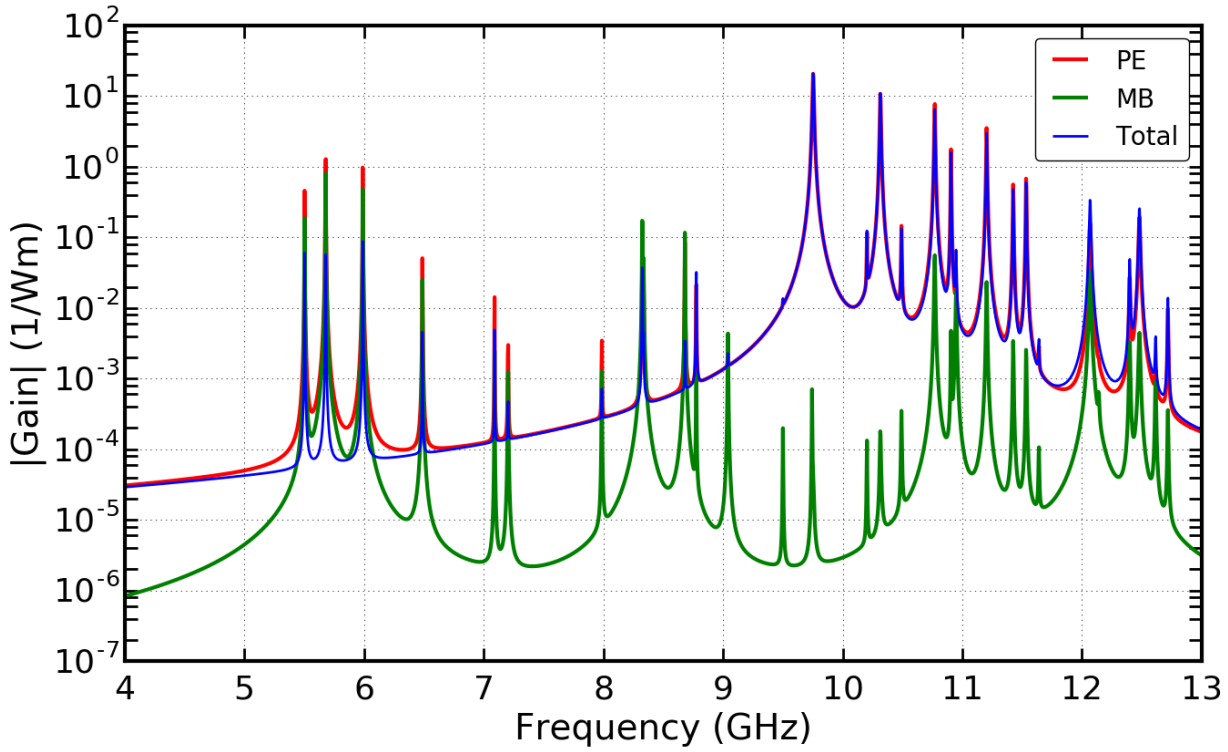


Fig. 4.63: Gain spectra on semilogy axis.

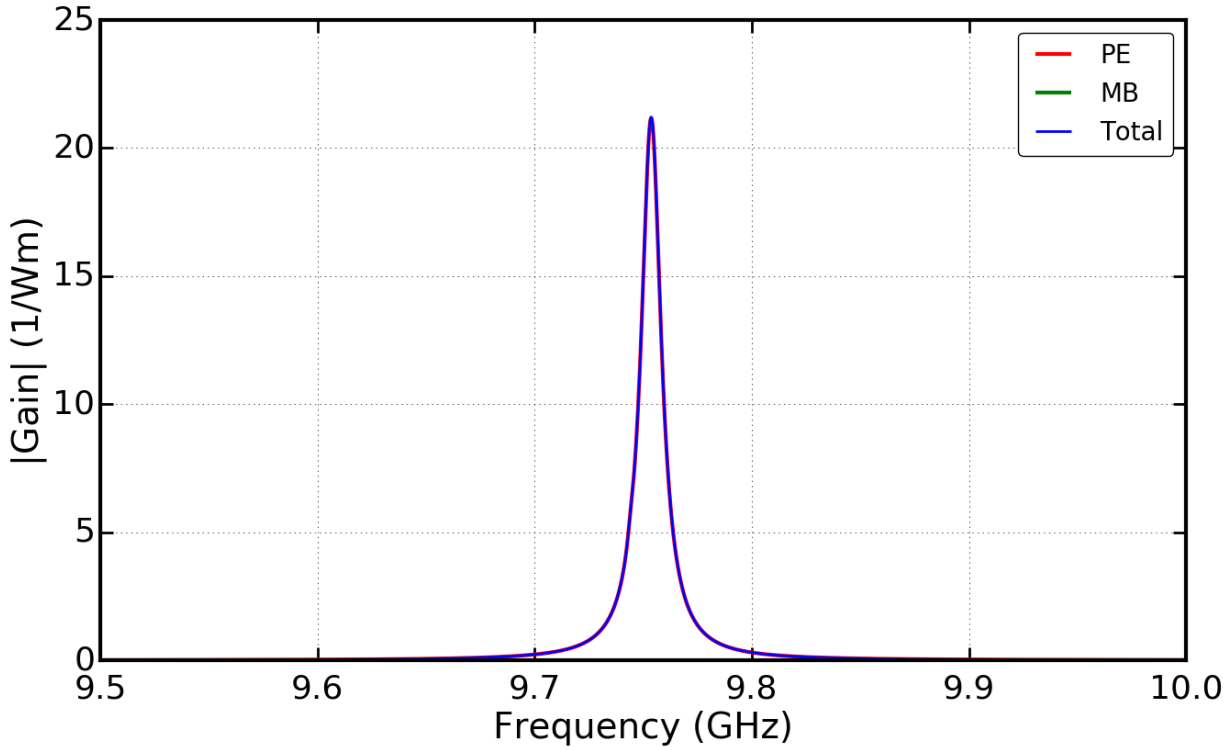


Fig. 4.64: Gain spectra zoomed in on mode D.

4.3.2 LitEx 2 – Laude and Beungot, *AIP Advances* (2013): BSBS in a rectangular silicon waveguide

This example in `simo-lit_02-Laude-AIPAdv_2013-silicon.py` again follows the paper of V. Laude and J.-C. Beugnot, Generation of phonons from electrostriction in small-core optical waveguides, *AIP Advances* **3**, 042109 (2013), but this time looks at the *silicon* waveguide case.

```
""" 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 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.get_material("Vacuum"),
```

```
                        material_a=materials.get_material("Si_2013_Laude"),
                        lc_bkg=1, lc_refine_1=400.0, lc_refine_2=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.plot_mode_fields(sim_EM_pump, xlim_min=0.2, xlim_max=0.2, ivals=[EM_ival_
↪pump],
                          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.plot_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, pdf_png='png')

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

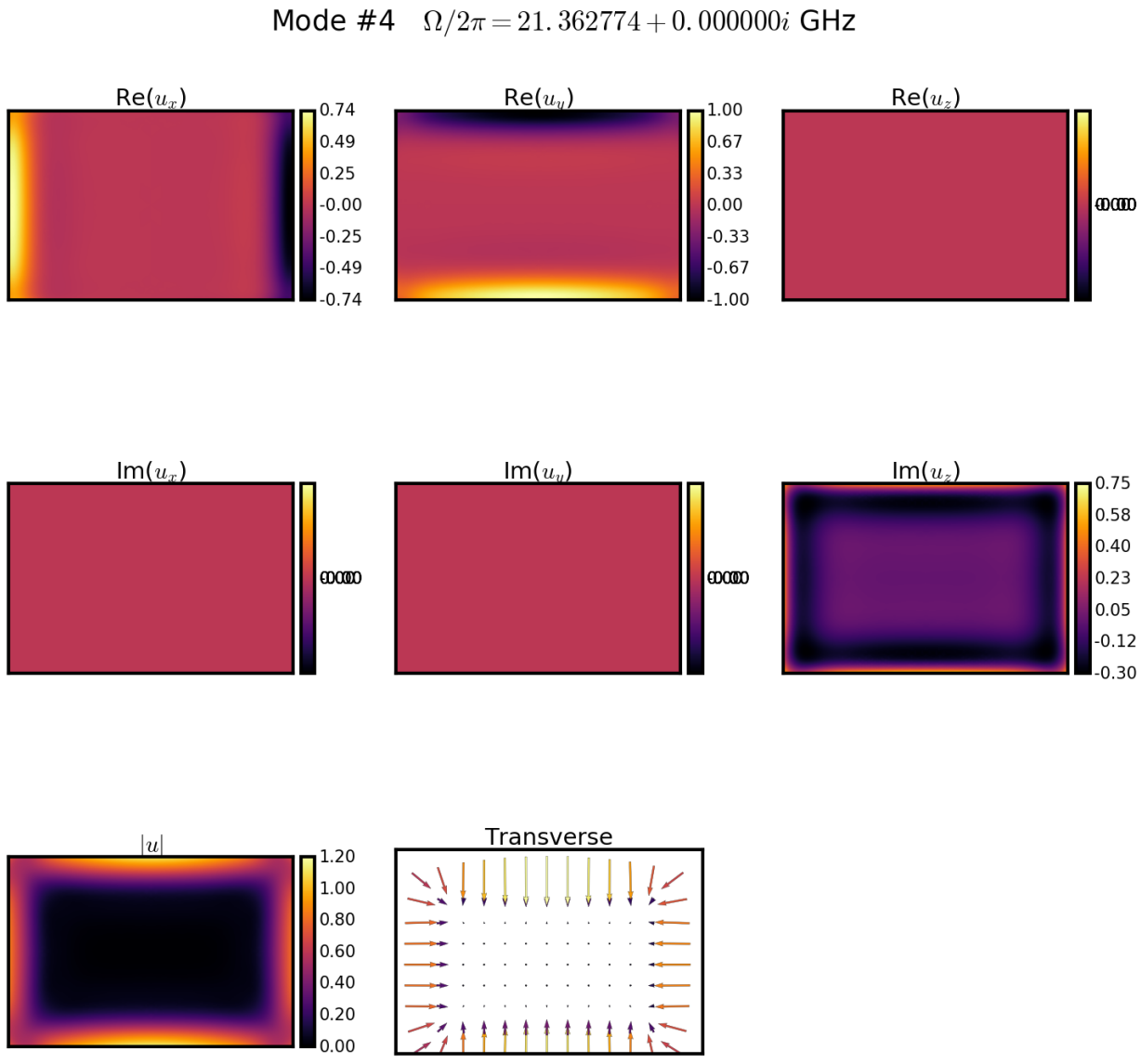


Fig. 4.65: High gain acoustic mode, marked as G in paper.

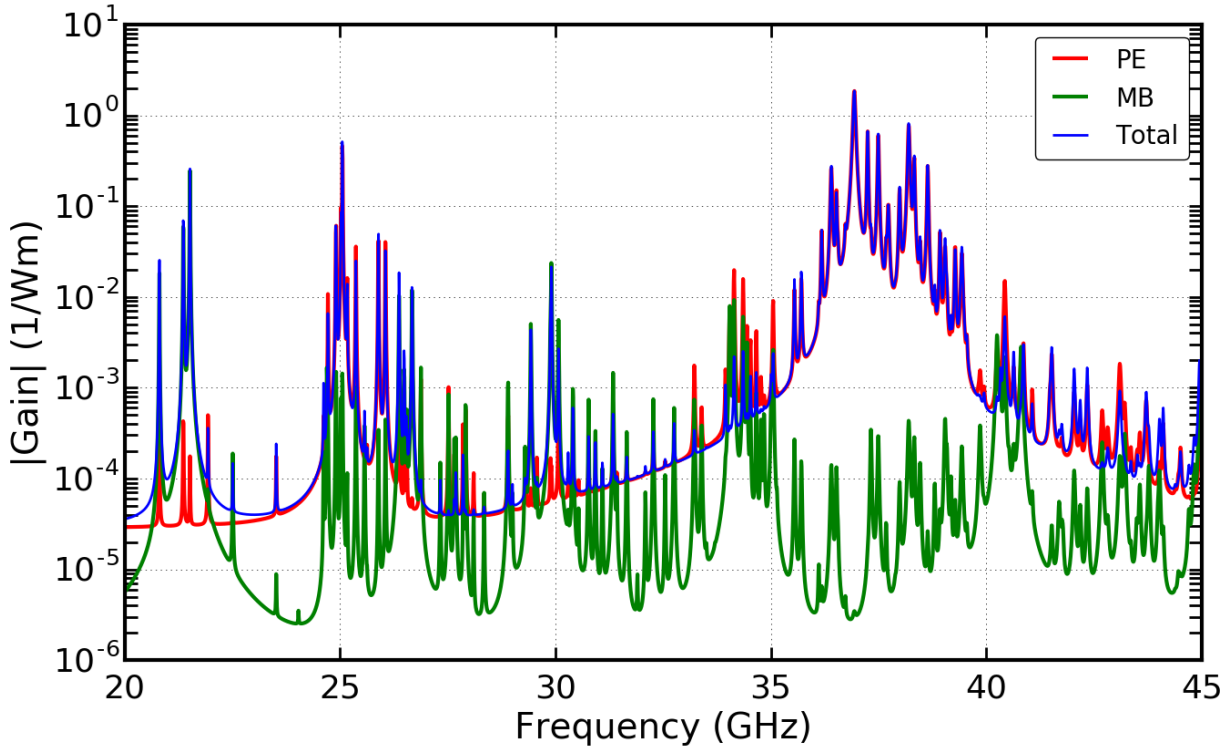


Fig. 4.66: Gain spectra on semilogy axis.

4.3.3 LitEx 3 – Beugnot *et al*, *Nature Communications* (2014): BSBS in a tapered fibre - scanning widths

This example, in `simo-lit_03-Beugnot-NatComm_2014.py`, is based on the calculation of backward SBS in a micron scale optical fibre described in J.-C. Beugnot *et al.*, Brillouin light scattering from surface acoustic waves in a subwavelength-diameter optical fibre, *Nature Communications* **5**, 5242 (2014).

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

import time
import datetime
import numpy as np
import sys
from multiprocessing import Pool
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.get_material("Vacuum"),
                            material_a=materials.get_material("SiO2_2016_Smith"),
                            lc_bkg=1, lc_refine_1=400.0, lc_refine_2=50.0)

    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))

    # Clear memory
    wguide = sim_EM_pump = sim_EM_Stokes = sim_AC = None
    SBS_gain = SBS_gain_PE = SBS_gain_MB = linewidth_Hz = Q_factors = alpha = None

    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)
# Note pool.map() doesn't pass errors back from fortran routines very well.
# It's good practise to run the extrema of your simulation range through map()
# before launcing full multicore simulation.


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))
```

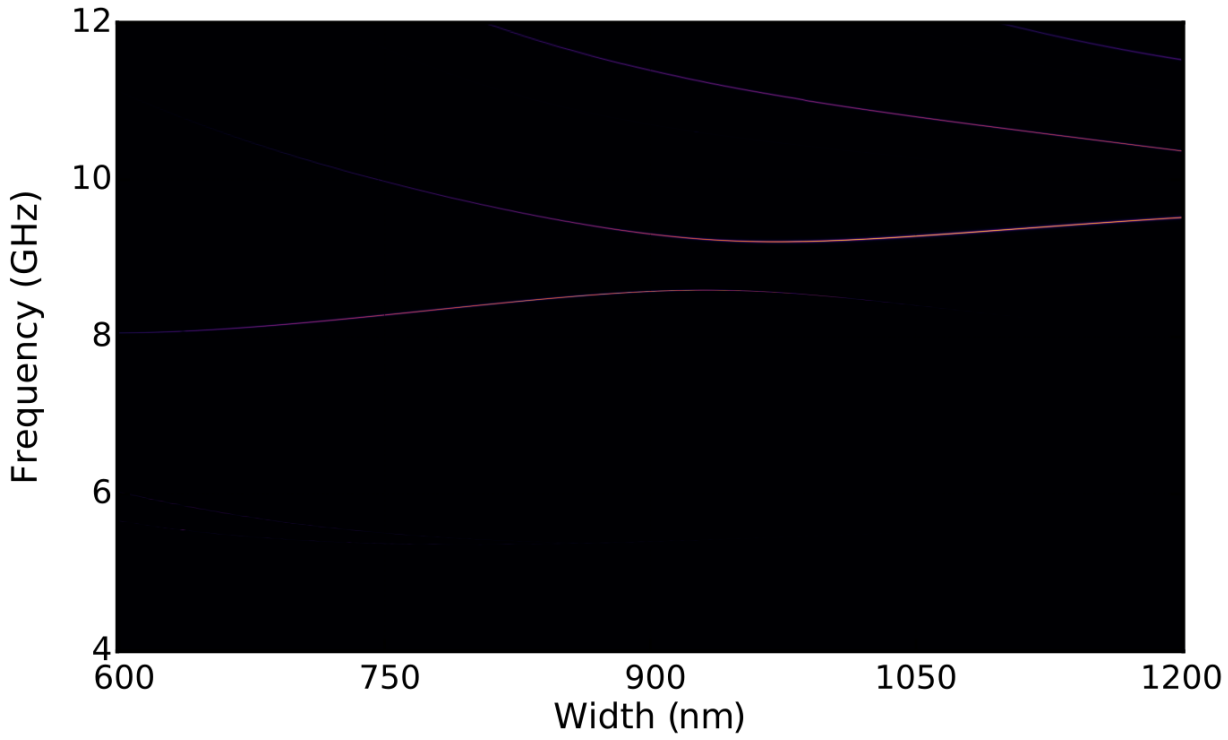


Fig. 4.67: Full acoustic wave spectrum for silica microwire, as per Fig. 4a in paper.

4.3.4 LitEx 4 – Van Laer *et al*, *Nature Photonics* (2015): FSBF in a waveguide on a pedestal

This example, in `simo-lit_04-pillar-Van_Laer-NatPhot_2015.py`, is based on the calculation of forward SBS in a pedestal silicon waveguide described in R. Van Laer *et al.*, Interaction between light and highly confined hypersound in a silicon photonic nanowire, *Nature Photonics* **9**, 199 (2015).

Note that the absence of an absorptive boundary in the acoustic model causes a problem where the slab layer significantly distorting acoustic modes. Adding this feature is a priority for a future release of NumBAT. The following example shows an approximate way to avoid the problem for now.

```
""" 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
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 = 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-pillar-'

# Rotate crystal axis of Si from <100> to <110>, starting with same Si_2016_Smith␣
↪data.
Si_110 = copy.deepcopy(materials.materials_dict["Si_2015_Van_Laer"])
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,
                        slab_a_x=slab_a_x, slab_a_y=slab_a_y,
                        pillar_x=pillar_x, pillar_y=pillar_y,
                        material_bkg=materials.materials_dict["Vacuum"],            #␣
↪background
                        material_a=Si_110,                          # rib
                        material_b=materials.materials_dict["SiO2_2015_Van_Laer"],  #␣
↪slab
                        material_c=materials.materials_dict["SiO2_2015_Van_Laer"],  #␣
↪pillar
                        lc_bkg=1, lc_refine_1=800.0, lc_refine_2=500.0)

# 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.fwd_Stokes_modes(sim_EM_pump)

plotting.plot_mode_fields(sim_EM_pump, ivals=[EM_ival_pump],
                          xlim_min=0.4, xlim_max=0.4, ylim_min=0.4, 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[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.plot_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, pdf_png='png')

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

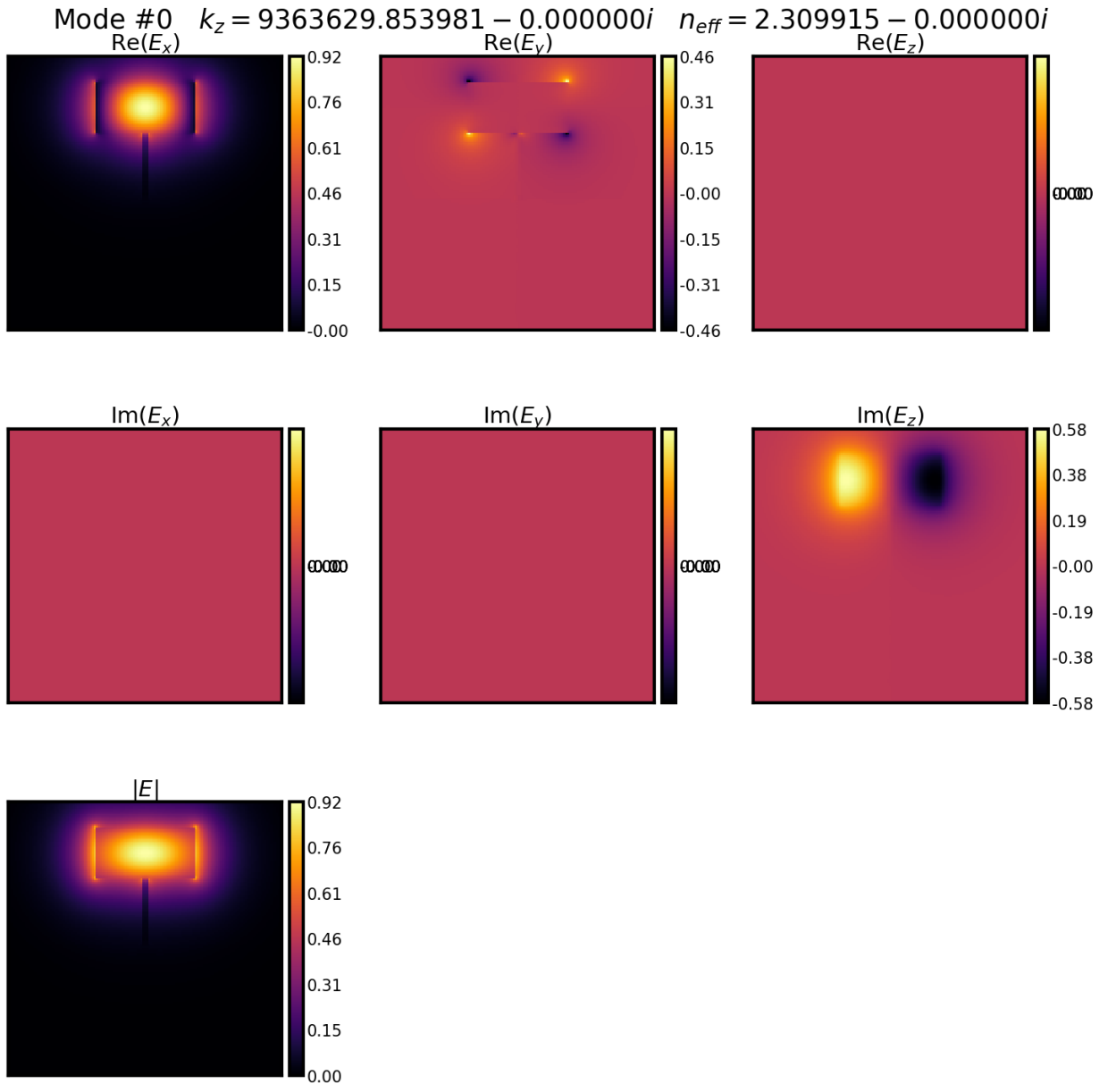


Fig. 4.68: Fundamental optical mode fields.

We may also choose to study the simplified situation where the pedestal is removed.

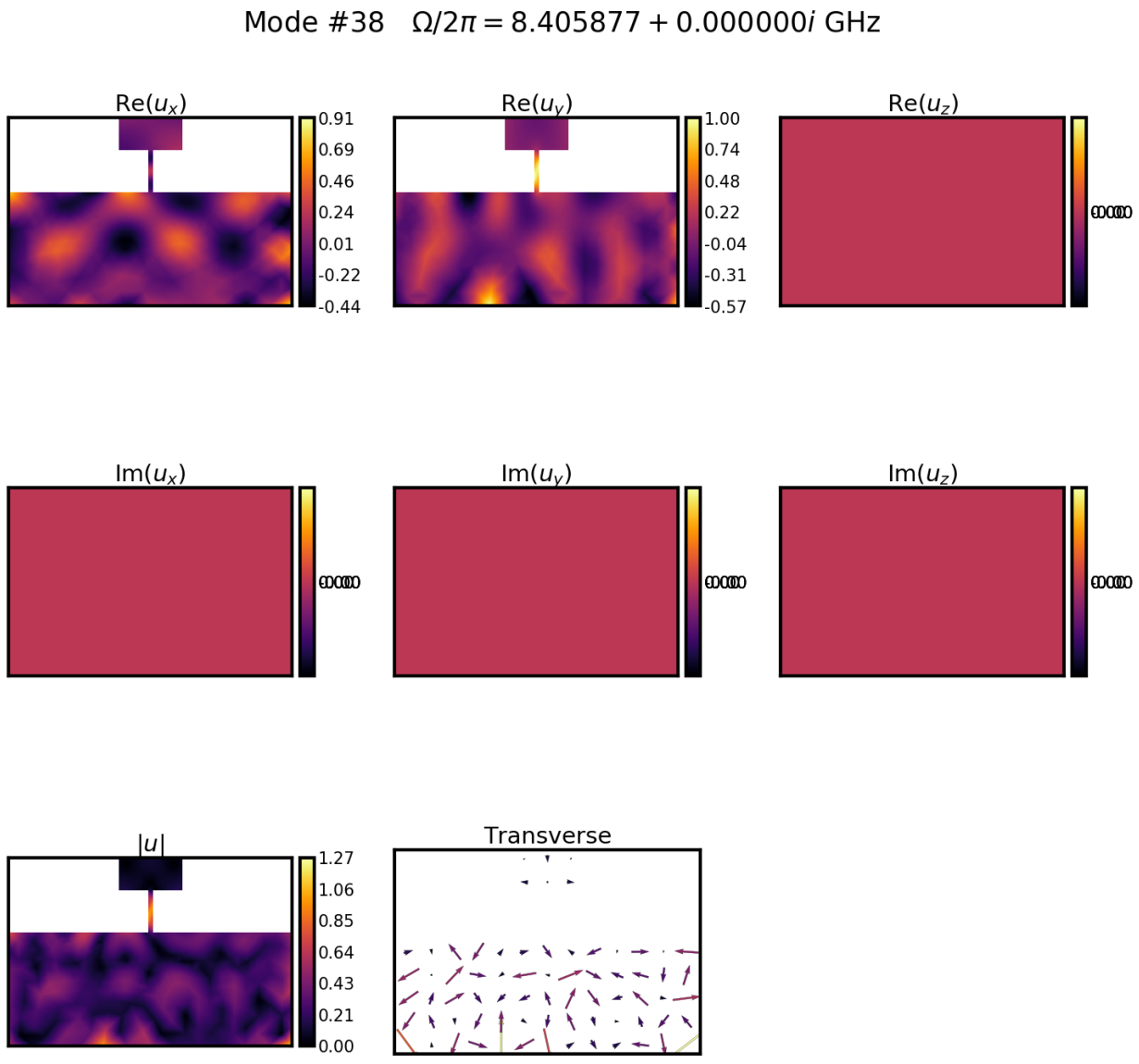


Fig. 4.69: Dominant high gain acoustic mode. Note how the absence of an absorptive boundary on the SiO2 slab causes this layer to significantly distorted the acoustic modes.

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

    Making simplification of ignoring the pedestal.
"""

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 plotting import Decorator
from fortran import NumBAT


# use this class to add or alter features to the final plots
class EMDecorator(Decorator):
  def __init__(self):
    super().__init__()

  def extra_axes_commands(self, ax):
    ax.tick_params(axis='x',color='gray', which='both')
    ax.tick_params(axis='y',color='gray', which='both')
    if self.is_single_plot():
      ax.tick_params(axis='x',length=20)
      ax.tick_params(axis='y',length=20)
      ax.tick_params(axis='x',width=2)
      ax.tick_params(axis='y',width=2)

      ax.tick_params(axis='x',length=10,which='minor')
      ax.tick_params(axis='y',length=10,which='minor')
      ax.tick_params(axis='x',width=2, which='minor')
      ax.tick_params(axis='y',width=2, which='minor')


emdecorate=EMDecorator()
#acdecorate=ACDecorator()



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 = 485
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 = 'fig6-'

# Rotate crystal axis of Si from <100> to <110>, starting with same Si_2016_Smith␣
↪data.
Si_110 = copy.deepcopy(materials.get_material("Si_2016_Smith")
# Si_110 = copy.deepcopy(materials.materials_dict["Si_2015_Van_Laer"])
Si_110.rotate_axis(np.pi/4,'z-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.get_material("Vacuum"),
                        material_a=Si_110, symmetry_flag=False,
                        lc_bkg=.25, lc_refine_1=200.0, lc_refine_2=200.0)

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


doem=True
doac=True
new_calcs=False

if doem:
  # Calculate Electromagnetic Modes
  if new_calcs:
    sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
    np.savez(prefix_str+'wguide_data', sim_EM_pump=sim_EM_pump)
  else:
    npzfile = np.load(prefix_str+'wguide_data.npz', allow_pickle=True)
    sim_EM_pump = npzfile['sim_EM_pump'].tolist()

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

  plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.43, xlim_max=0.43, ivals=[EM_ival_
↪pump],
                            ylim_min=0.43, ylim_max=0.43, EM_AC='EM_E',
                            n_points=2000, quiver_points=10, prefix_str=prefix_str,␣
↪pdf_png='png',
                            ticks=True, comps=('Ex', 'Eabs', 'Et'),␣
↪decorator=emdecorate)

  # 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))


if doac:
  k_AC = 5 # close but not quite zero

  # Calculate Acoustic Modes
  if new_calcs:
    sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump)
    np.savez(prefix_str+'wguide_data_AC', sim_AC=sim_AC)
  else:
    npzfile = np.load(prefix_str+'wguide_data_AC.npz', allow_pickle=True)
    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.plot_mode_fields(sim_AC, ivals=(7,), EM_AC='AC', prefix_str=prefix_str,
                            pdf_png='png', comps=('ux','uy','ut','uabs'), ticks=True,
                            xlim_min=-0.05, ylim_min=-.05, xlim_max=-0.05, ylim_max=-.
↪05)

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)

# 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 Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 9.1 # GHz
freq_max = 9.3 # 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='', pdf_png='png')

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

Which gives good agreement for the gain spectrum.

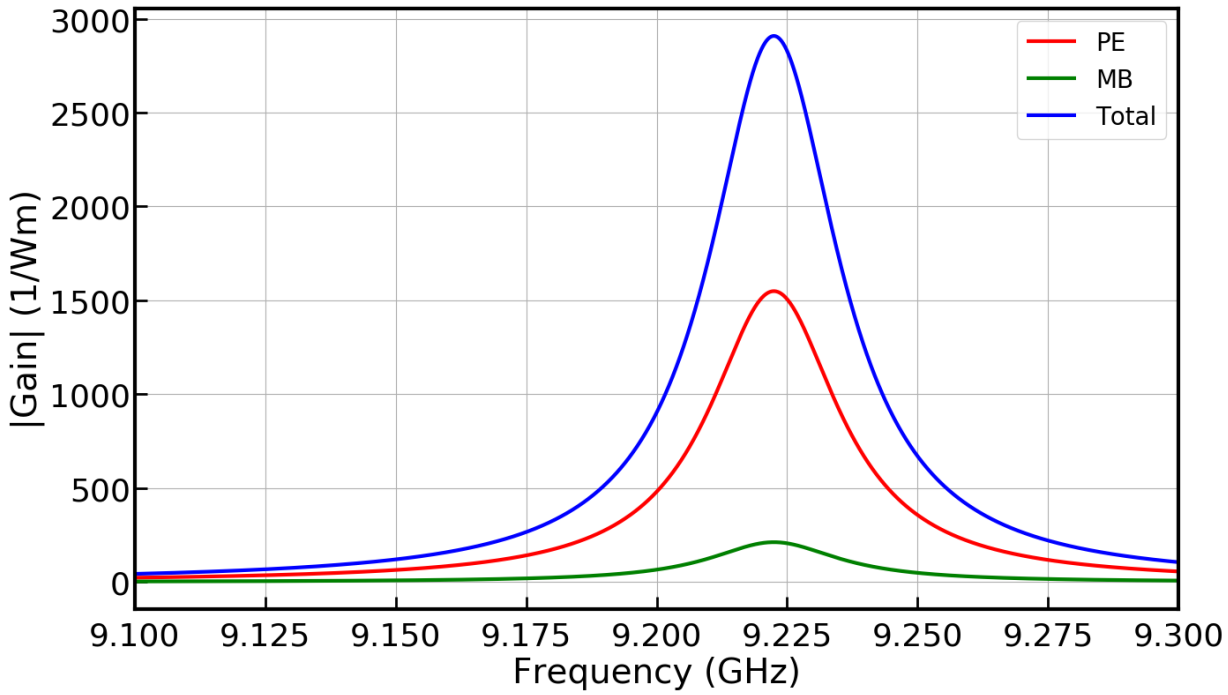


Fig. 4.70: Gain spectrum for the simplified case of a waveguide surrounded by vacuum.

4.3.5 LitEx 5 – 2015 - Van Laer *et al*, *New Journal of Physics* (2015): FSBF in a waveguide without a pedestal

This example, in `simo-lit_05-Van_Laer-NJP_2015.py`, continues the study of forward SBS in a pedestal silicon waveguide described in R. Van Laer *et al.*, Interaction between light and highly confined hypersound in a silicon photonic nanowire, *Nature Photonics* **9**, 199 (2015).

In this case, we simply remove the pedestal and model the main rectangular waveguide. This makes the acoustic loss calculation incorrect but avoids the problem of acoustic energy being excessively concentrated in the substrate.

```
print("\n Simulation time (sec.)", (end - start))
""" 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.get_material("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.get_material("Vacuum"),
                        material_a=Si_110, symmetry_flag=False,
                        lc_bkg=1, lc_refine_1=1200.0, lc_refine_2=800.0)

# 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.plot_mode_fields(sim_EM_pump, xlim_min=0.45, xlim_max=0.45,
                          ivals=[EM_ival_pump], ylim_min=0.45, ylim_max=0.45,
                          EM_AC='EM_E', n_points=1500,
                          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.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, pdf_png='png')

set_q_factor = 230 # NJP

# 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 Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 9.1 # 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='', pdf_png='png')

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

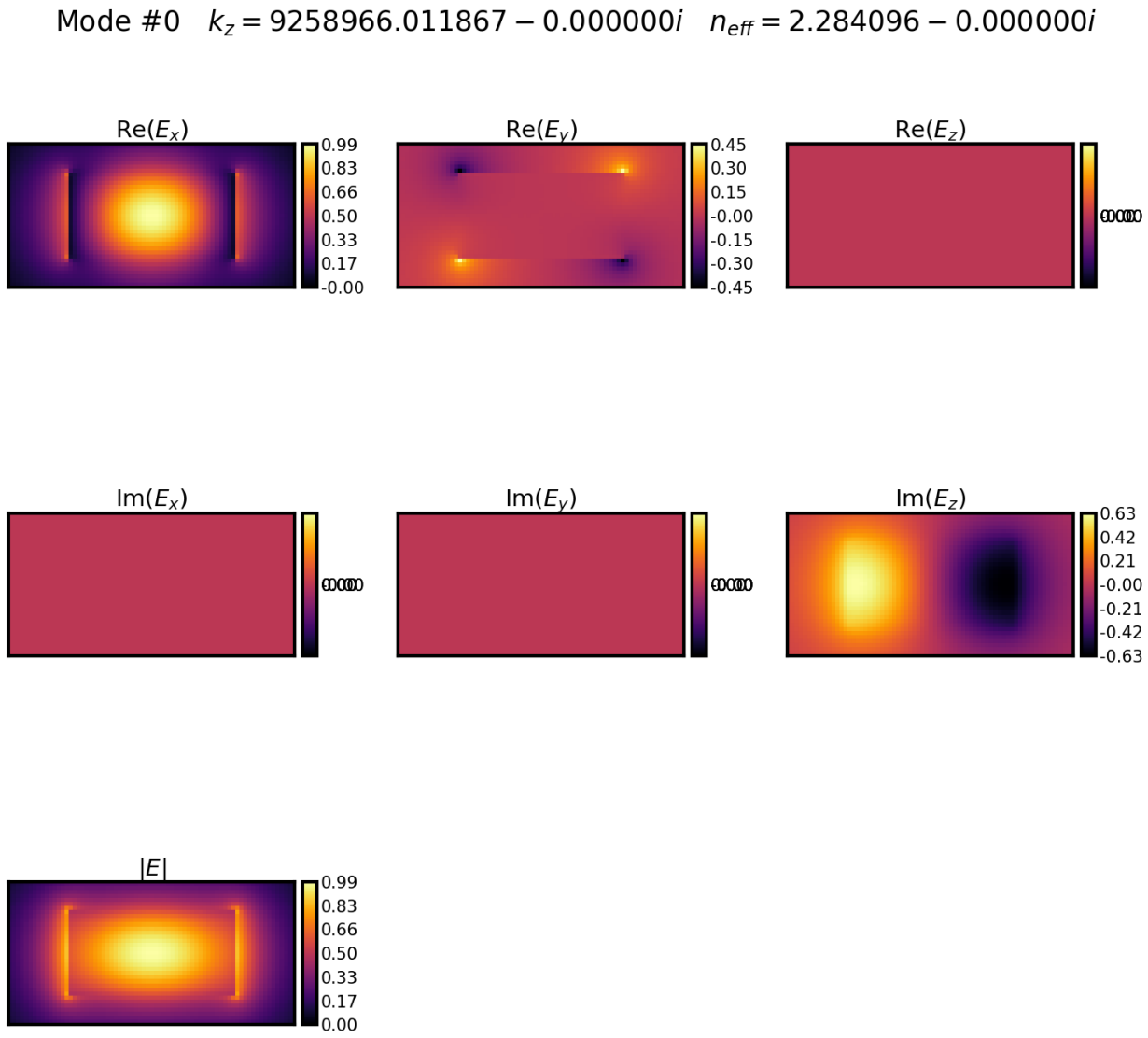


Fig. 4.71: Fundamental optical mode fields.

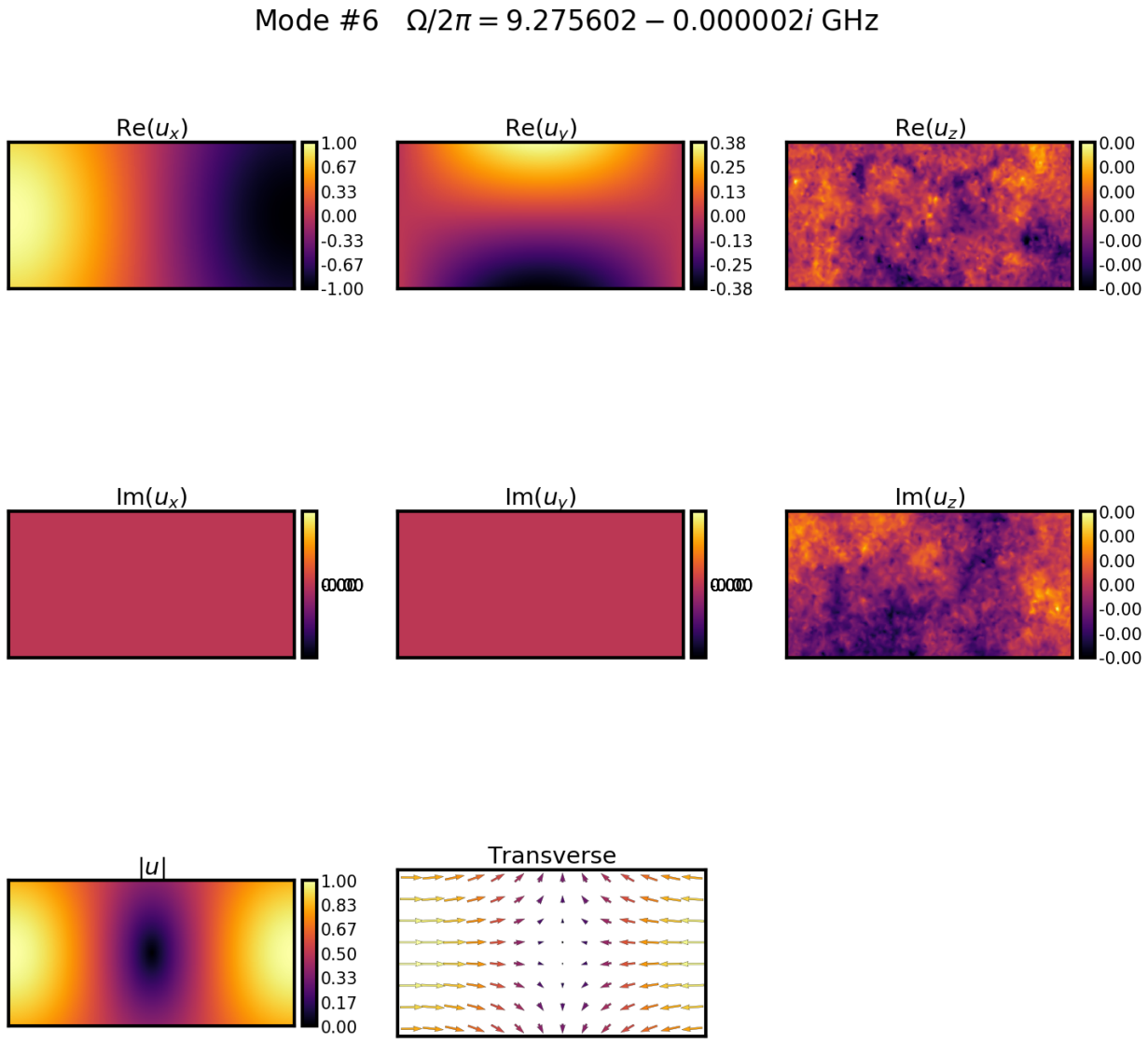


Fig. 4.72: Dominant high gain acoustic mode.

4.3.6 LitEx 6 – Florez *et al*, *Nature Communications* (2016): BSBS self-cancellation in a tapered fibre ($d = 550$ nm)

This example, in `simo-lit_06_1-Florez-NatComm_2016-d550nm.py`, looks at the phenomenon of Brillouin "self-cancellation" due to the electrostrictive and radiation pressure effects acting with opposite sign. This was described in O. Florez *et al.*, Brillouin self-cancellation, *Nature Communications* **7**, 11759 (2016).

```
print("\n Simulation time (sec.)", (end - start))
""" 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 plotting import Decorator
from fortran import NumBAT

# use this class to add or alter features to the final plots
class EMDecorator(Decorator):
  def __init__(self):
    super().__init__()

    #title_font=24
    #self._multi_sizes= {'title':title_font-2, 'subplot_title':title_font-5, 'cbar_
↪tick':title_font-10, 'ax_tick':title_font-10, 'ax_label':title_font-10 }
    #self._single_sizes= {'ax_label':80, 'subplot_title':80, 'cbar_tick':60, 'ax_tick
↪':70}
    ##self._single_sizes= {'ax_label':60, 'subplot_title':60, 'cbar_tick':40, 'ax_tick
↪':40}
    #self._is_single=True

  def extra_axes_commands(self, ax):
    circle1 = plt.Circle((0, 0), 0.275, color='black', fill=False)
    ax.add_artist(circle1)


class ACDecorator(Decorator):
  def __init__(self):
    super().__init__()
    #title_font=24
    #self._multi_sizes= {'title':title_font-2, 'subplot_title':title_font-5, 'cbar_
↪tick':title_font-10, 'ax_tick':title_font-10, 'ax_label':title_font-10 }
#
#    self._single_sizes= {'ax_label':30, 'subplot_title':40, 'cbar_tick':20, 'ax_tick
↪':30, 'title_pad':20}
```

```
#    self._single_sizes= {'ax_label':80, 'subplot_title':80, 'cbar_tick':60, 'ax_tick
↪':70, 'title_pad':25}
#    self._is_single=True

  def extra_axes_commands(self, ax):
    circle1 = plt.Circle((0, 0), 0.275, color='black', fill=False)
    ax.add_artist(circle1)

emdecorate=EMDecorator()
acdecorate=ACDecorator()


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
AC_ival = 'All'

prefix_str = 'fig10-'

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("SiO2_2013_Laude"),
                        lc_bkg=.25, lc_refine_1=170.0, lc_refine_2=85.0)

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

doem=True
doac=True
new_calcs=False

if doem:
  # Calculate Electromagnetic Modes
  if new_calcs:
    sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff)
    np.savez('wguide_data_florez', sim_EM_pump=sim_EM_pump)
  else:
    npzfile = np.load('wguide_data_florez.npz', allow_pickle=True)
    sim_EM_pump = npzfile['sim_EM_pump'].tolist()
  sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

  plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ivals=[EM_ival_
↪pump],
                            ylim_min=0.3, ylim_max=0.3, EM_AC='EM_E',
                            prefix_str=prefix_str, pdf_png='png', ticks=True,
```

```
                            decorator=emdecorate, quiver_points=20)

  # 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))

if not doac: sys.exit(0)

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
if new_calcs:
  sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_
↪Hz=shift_Hz)
  np.savez('wguide_data_florez_AC', sim_AC=sim_AC)
else:
  npzfile = np.load('wguide_data_florez_AC.npz', allow_pickle=True)
  sim_AC = npzfile['sim_AC'].tolist()

plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, suffix_str='',
ticks=True, ivals=[4,5], comps=('ut','uabs'),
xlim_min=-.1, ylim_min=-.1, xlim_max=-.1, ylim_max=-.1,
decorator=acdecorate, quiver_points=20, pdf_png='png',colorbar=True)

# 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, pdf_png='png')

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 5.86  # GHz
freq_max = 5.9  # 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='-5', pdf_png='png')
```

```
# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 6.28  # GHz
freq_max = 6.32  # 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='-6', pdf_png='png')

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 8.09  # GHz
freq_max = 8.13  # 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,
    prefix_str=prefix_str, suffix_str='-8', pdf_png='png')

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 11.65  # GHz
freq_max = 11.69  # 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='-11', pdf_png='png')

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

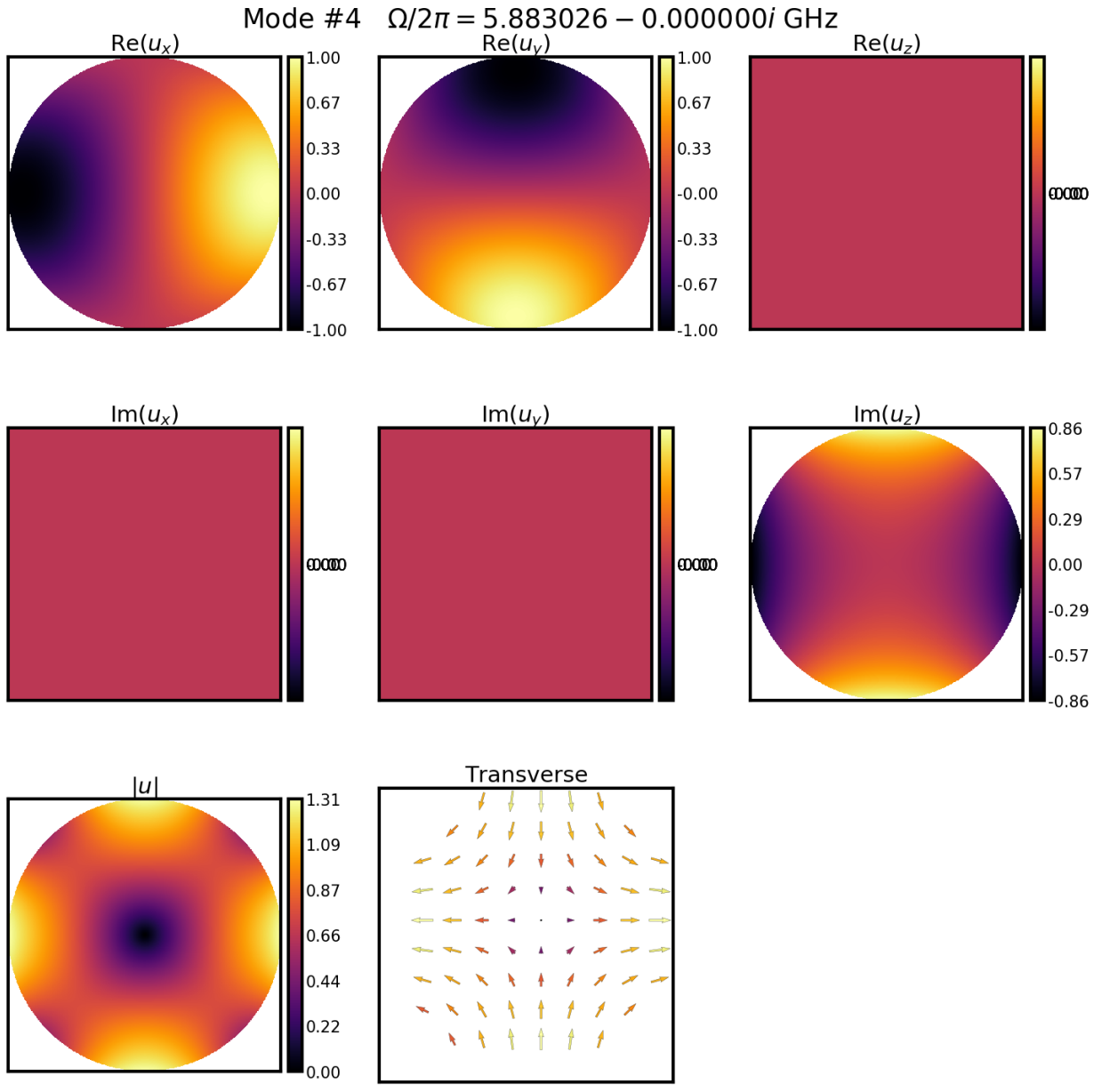


Fig. 4.73: TR_{21} acoustic mode fields of a nanowire with diameter 550 nm.

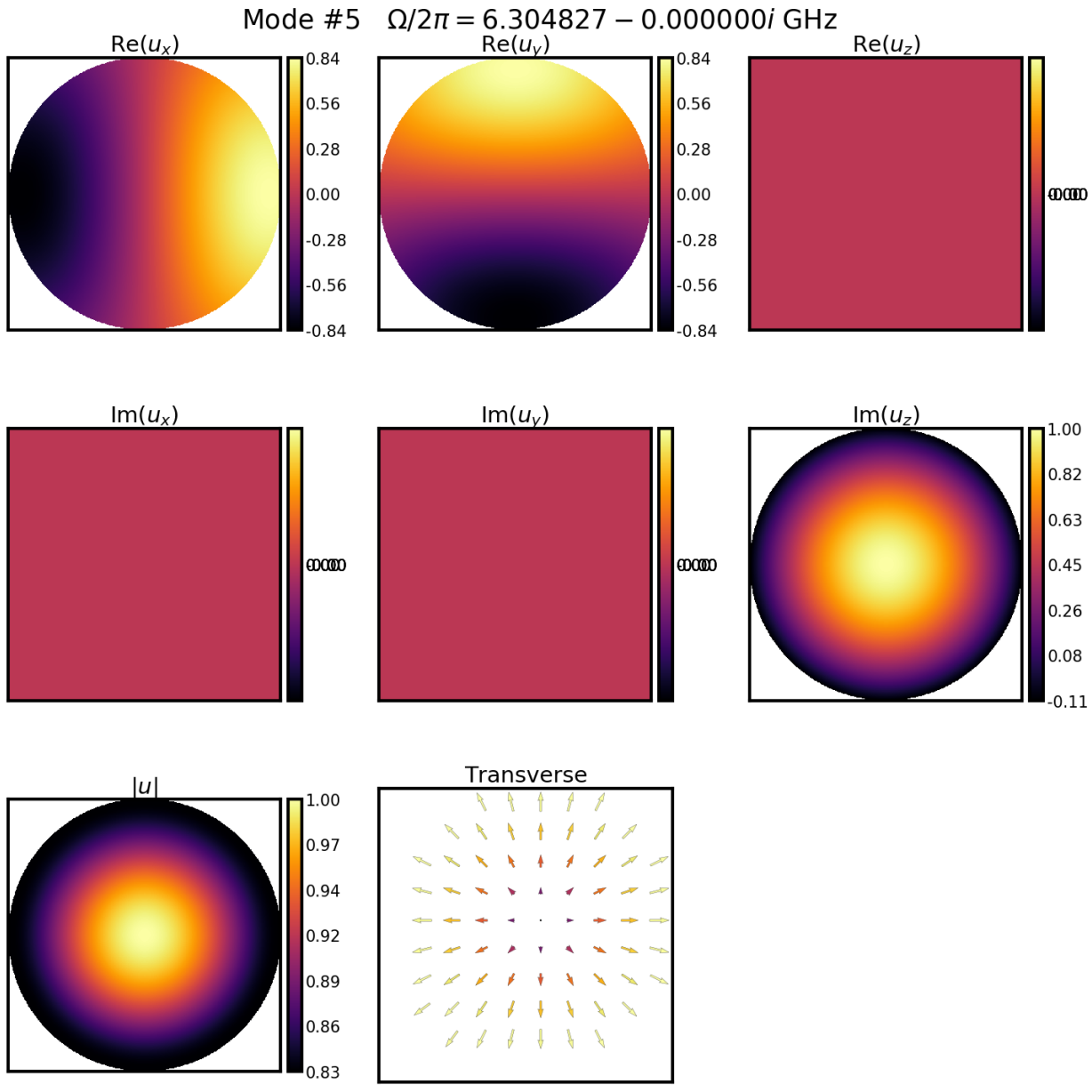


Fig. 4.74: R_{01} acoustic mode fields of a nanowire with diameter 550 nm.

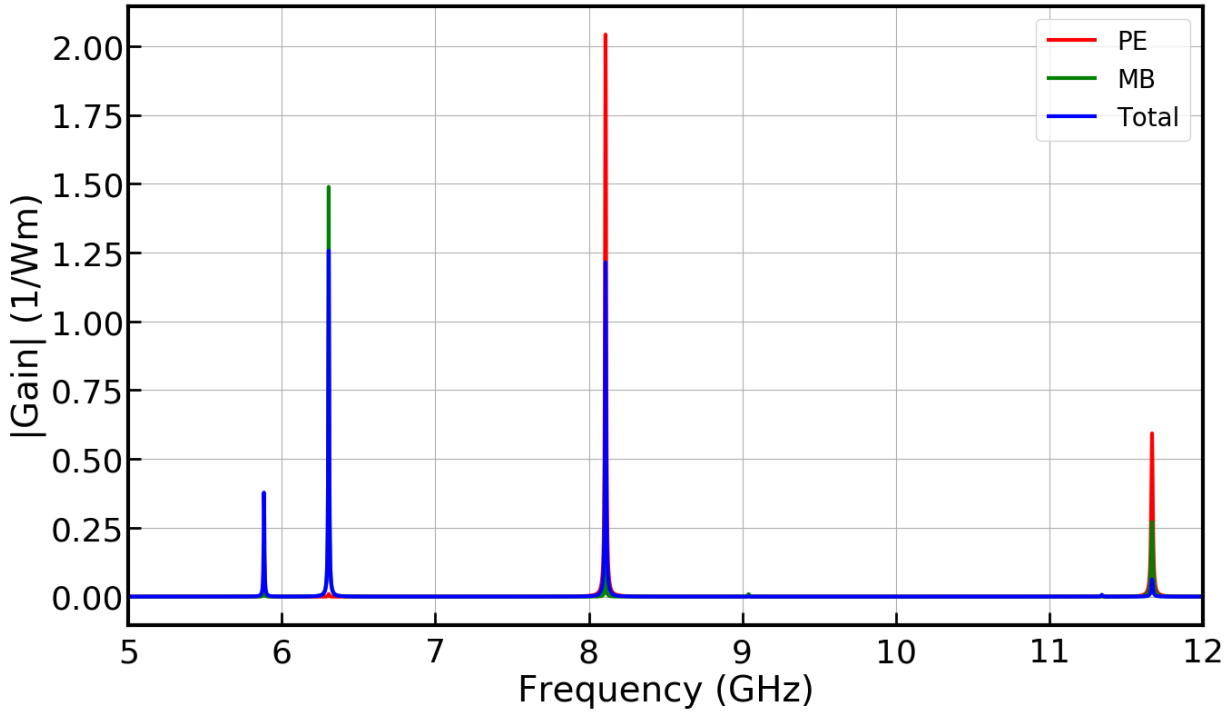
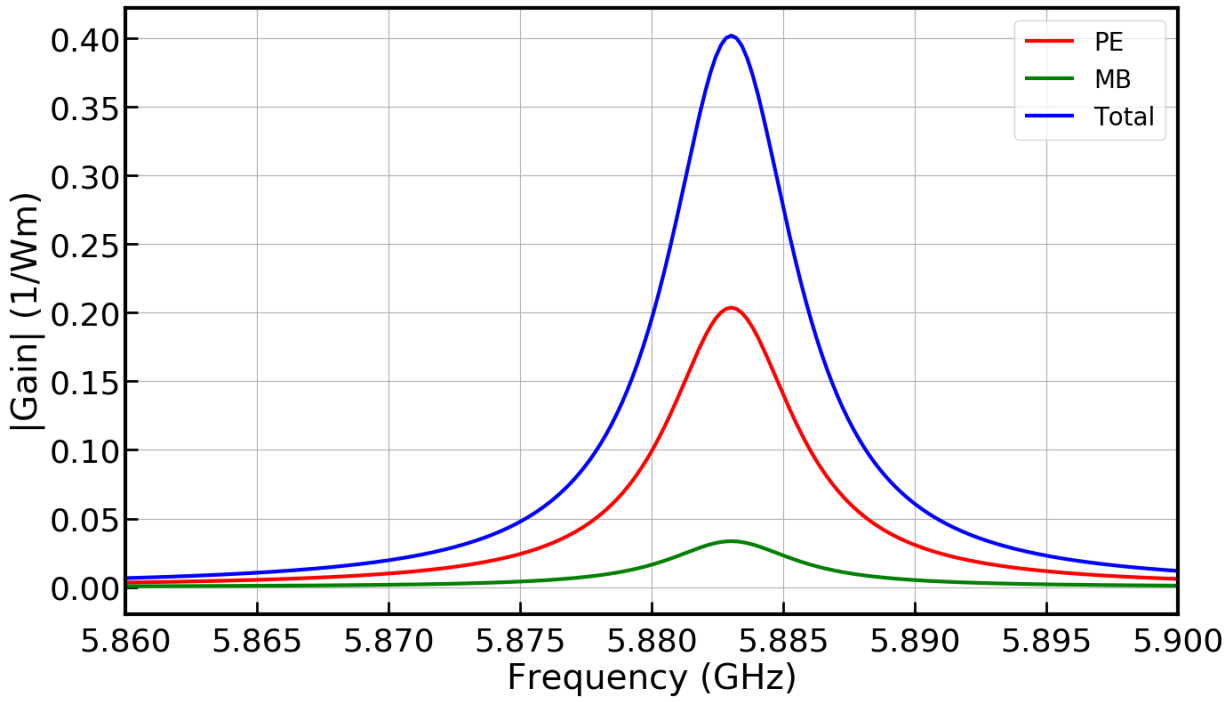
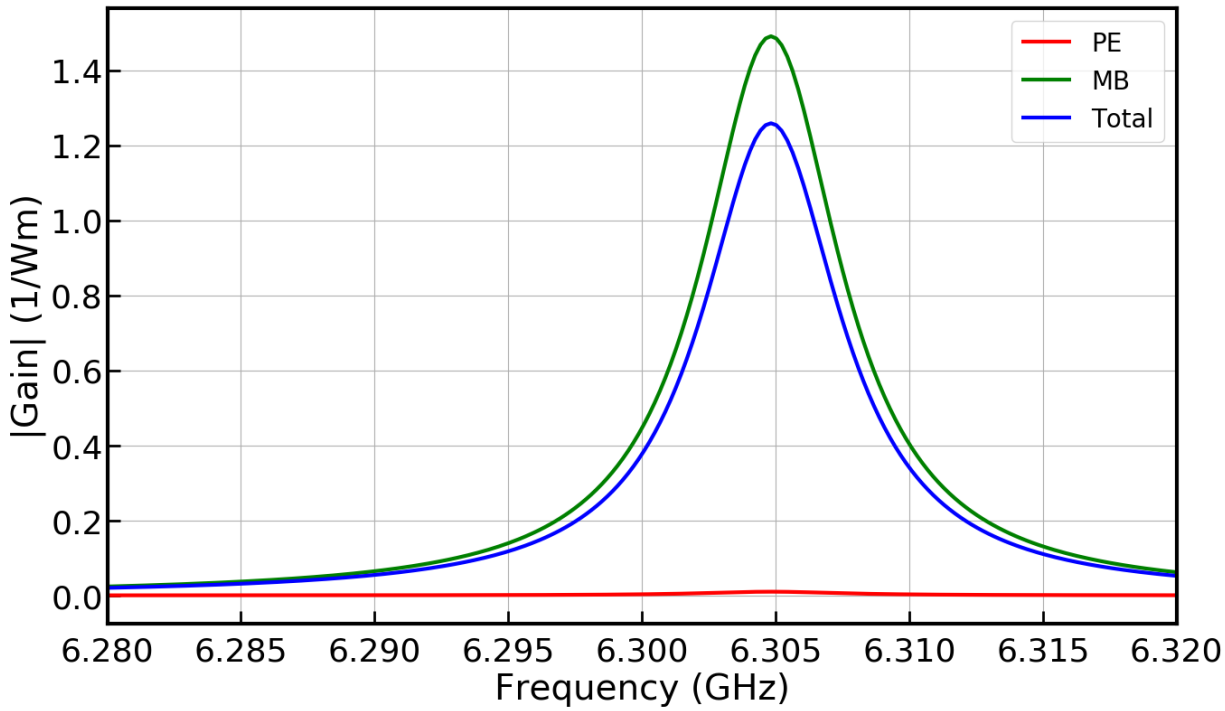
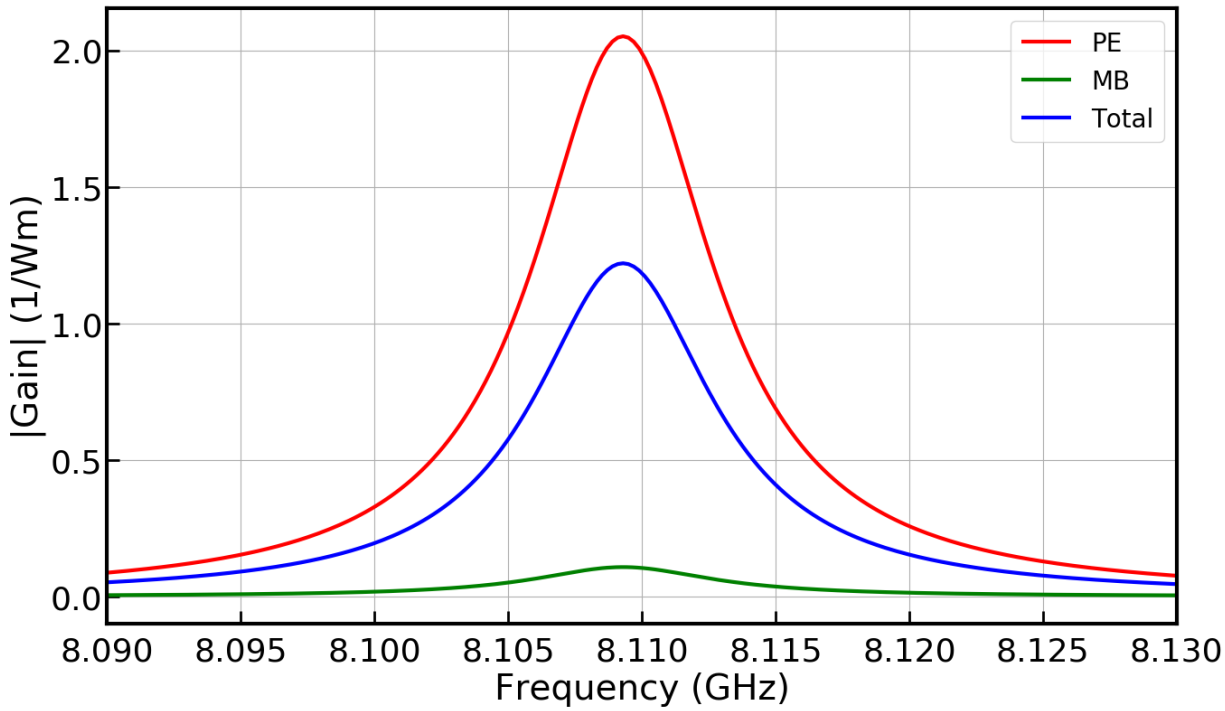


Fig. 4.75: Gain spectra of a nanowire with diameter 550 nm, matching blue curve of Fig. 3b in paper.




PE
MB
Total
|Gain| (1/Wm)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
6.280
6.285
6.290
6.295
6.300
6.305
6.310
6.315
6.320
Frequency (GHz)


PE
MB
Total
|Gain| (1/Wm)
0.0
0.5
1.0
1.5
2.0
8.090
8.095
8.100
8.105
8.110
8.115
8.120
8.125
8.130
Frequency (GHz)

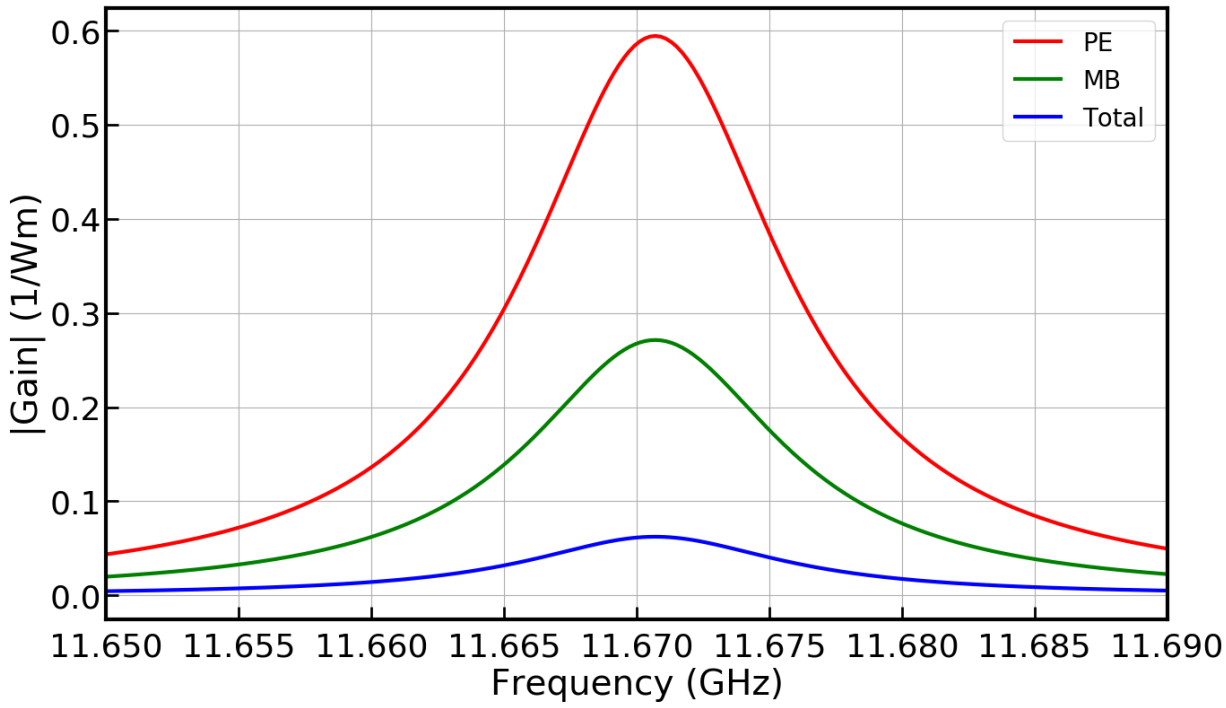


Fig. 4.76: Zoomed in gain spectra around gaint peaks of 550 nm diameter NW.

4.3.7 LitEx 6b – Florez *et al*, *Nature Communications* (2016): BSBS self-cancellation in a tapered fibre ($d = 1160$ nm)

This example, in `simo-lit_06_2-Florez-NatComm_2016-1160nm.py`, again looks at the paper O. Florez *et al.*, Brillouin self-cancellation, *Nature Communications* **7**, 11759 (2016), but now for a wider core.

```
print("\n Simulation time (sec.)", (end - start))
""" 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 = 3*wl_nm
unitcell_y = unitcell_x
inc_a_x = 1160 # 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
AC_ival = 'All'

prefix_str = 'lit_06_2-'

# 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.get_material("Vacuum"),
                        material_a=materials.get_material("SiO2_2013_Laude"),
                        lc_bkg=1, lc_refine_1=600.0, lc_refine_2=300.0)

# 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.plot_mode_fields(sim_EM_pump, xlim_min=0.3, xlim_max=0.3, ivals=[EM_ival_
↪pump],
                          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.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_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, pdf_png='pdf')

# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 5.2  # GHz
freq_max = 5.7  # 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, pdf_png='png')

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

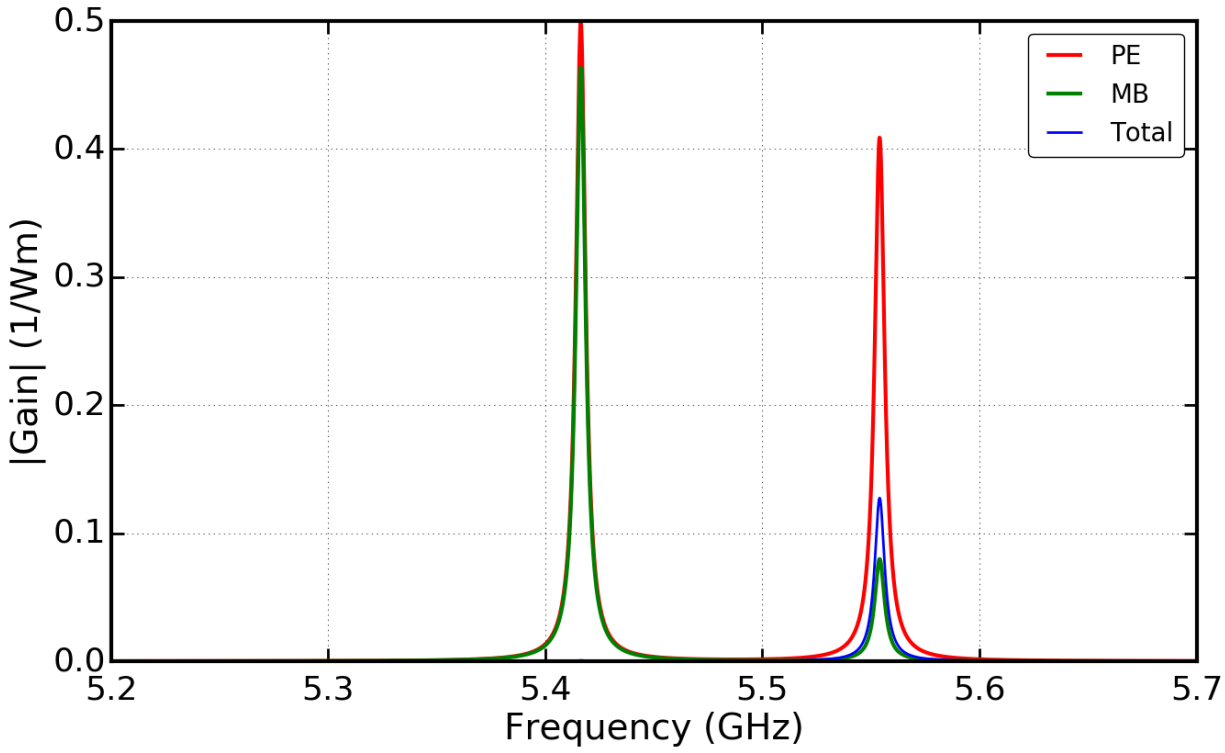


Fig. 4.77: Gain spectra of a nanowire with diameter 1160 nm, as in Fig. 4 of Florez, showing near perfect cancellation at 5.4 GHz.

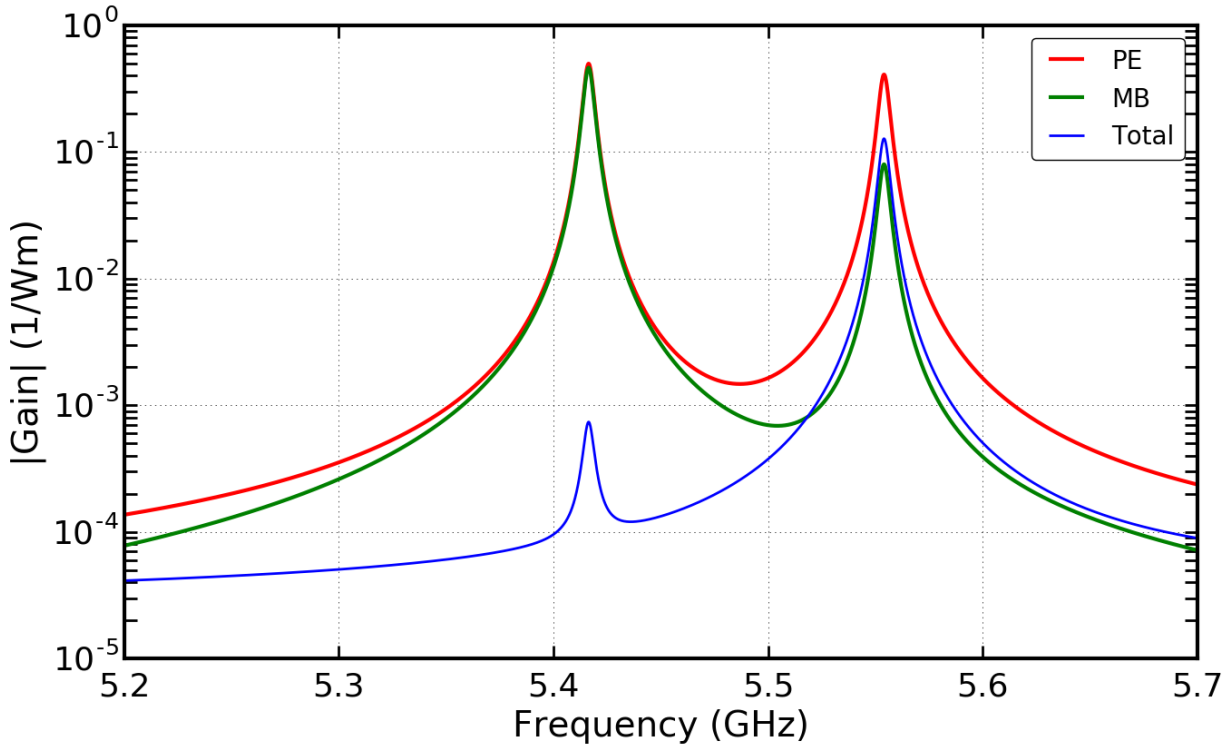


Fig. 4.78: Gain spectra of a nanowire with diameter 1160 nm, as in Fig. 4 of paper, showing near perfect cancellation at 5.4 GHz.

4.3.8 LitEx 7 – Kittlaus *et al*, *Nature Photonics* (2016), FSBF in a silicon rib waveguide

This example, in `../../lit_examples/simo-lit_07-Kittlaus-NatPhot_2016.py`, explores a first geometry showing large forward SBS in silicon as described in E. Kittlaus *et al.*, Large Brillouin amplification in silicon, *Nature Photonics* **10**, 463 (2016).

```
print("\n Simulation time (sec.)", (end - start))
""" 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 = 6*wl_nm
unitcell_y = 0.4*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.materials_dict["Si_2015_Van_Laer"])
Si_110 = copy.deepcopy(materials.get_material("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.get_material("Vacuum"),
                        material_a=Si_110,
                        material_b=Si_110, symmetry_flag=False,
                        lc_bkg=1, lc_refine_1=2000.0, lc_refine_2=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.fwd_Stokes_modes(sim_EM_pump)

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ivals=[EM_ival_
↪pump],
                          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.plot_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, pdf_png='png')
```

```
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 Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] 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='', pdf_png='png')

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

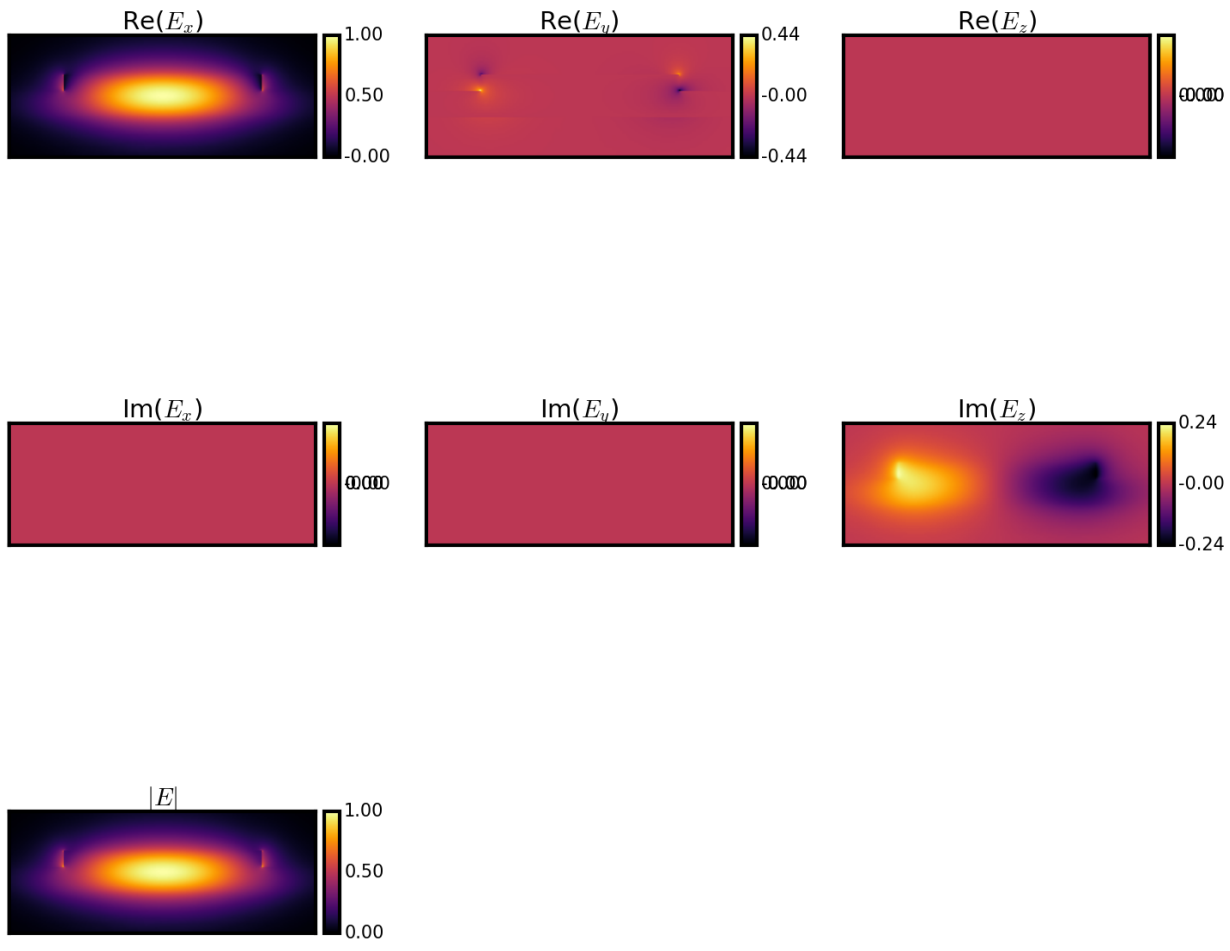


Fig. 4.79: Fundamental optical mode fields.

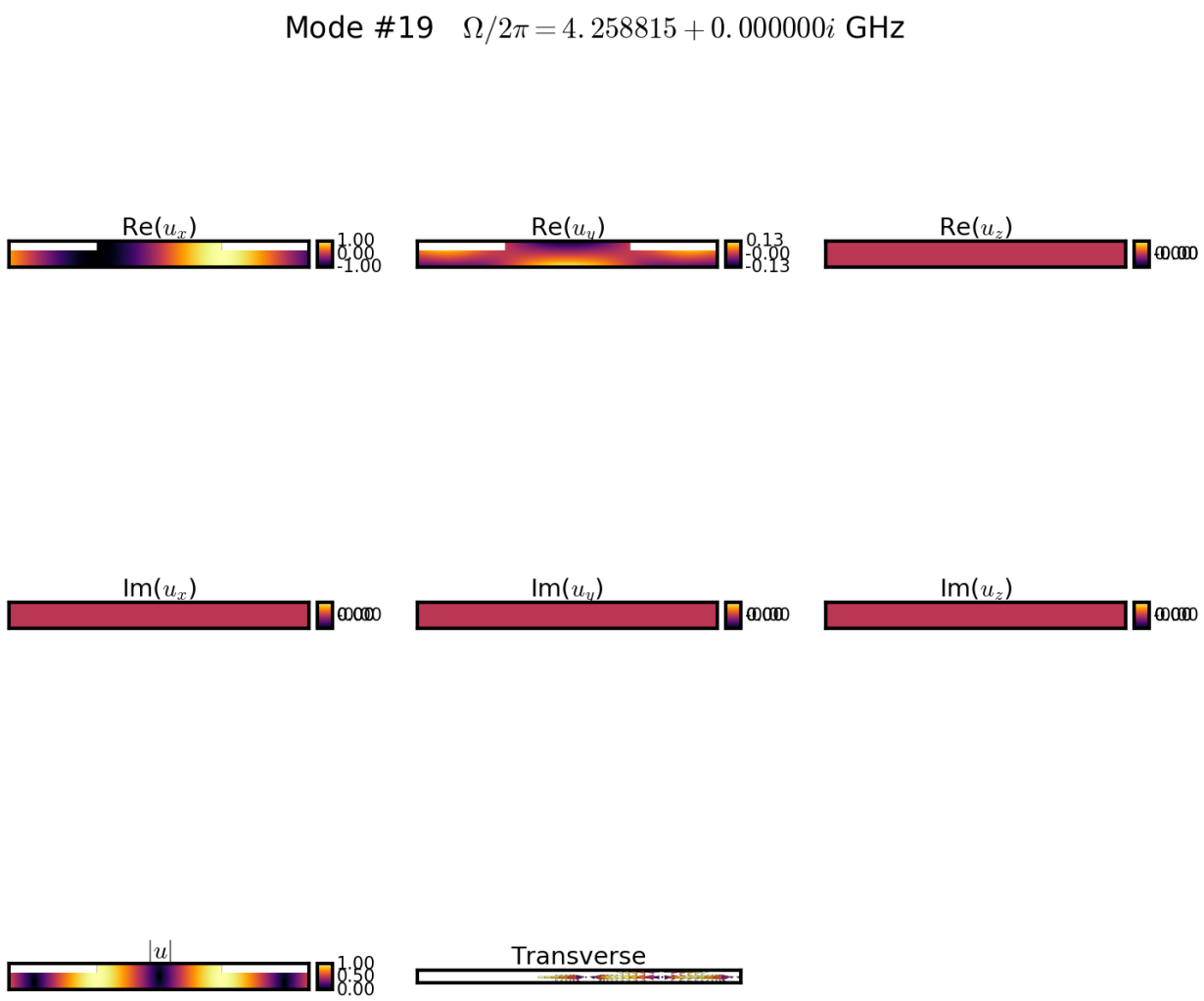


Fig. 4.80: Dominant high gain acoustic mode.

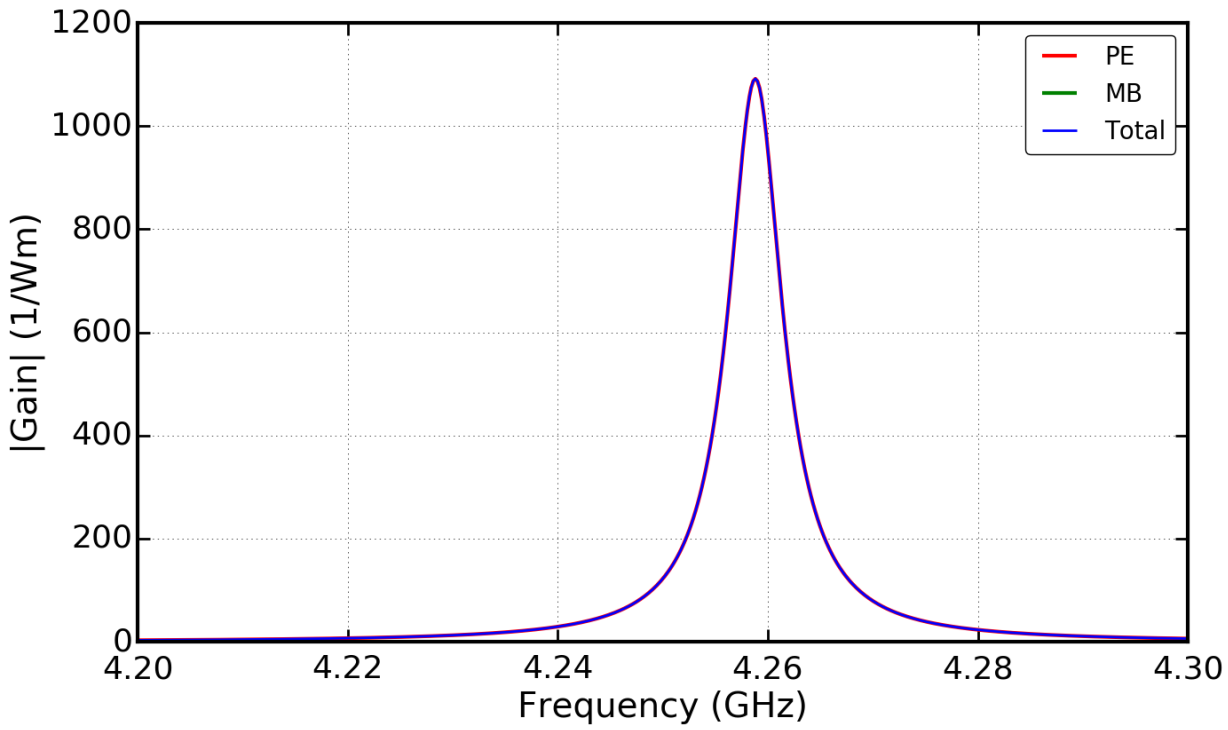


Fig. 4.81: Gain spectra showing gain due to photoelastic effect, gain due to moving boundary effect, and total gain.

4.3.9 LitEx 8 – Kittlaus *et al*, *Nature Communications* (2017): Intermodal FSBF in a silicon waveguide

This example (`simo-lit_08-Kittlaus-NatComm_2017.py`), also from the Yale group, examines intermode forward Brillouin scattering in silicon.

```
print("\n Simulation time (sec.)", (end - start))
""" Replicating the results of
    On-chip inter-modal Brillouin scattering
    Kittlaus et al.
    http://dx.doi.org/10.1038/ncomms15819
"""

import time
import datetime
import numpy as np
import sys
import copy
from matplotlib.ticker import AutoMinorLocator

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

class EMDecorator(Decorator):
  def __init__(self):
    super().__init__()

  def extra_axes_commands(self, ax):

    ax.tick_params(axis='x',color='gray', which='both')
    ax.tick_params(axis='y',color='gray', which='both')
    if self._is_single:
      ax.tick_params(axis='x',length=20)
      ax.tick_params(axis='y',length=20)
      ax.tick_params(axis='x',width=2)
      ax.tick_params(axis='y',width=2)

      ax.tick_params(axis='x',length=10,which='minor')
      ax.tick_params(axis='y',length=10,which='minor')
      ax.tick_params(axis='x',width=2, which='minor')
      ax.tick_params(axis='y',width=2, which='minor')

      ax.set_xticks(np.arange(-1.00,1.01,.1), minor=True)
      ax.set_yticks([-.3,0,.3])
      ax.set_yticks([-.5,-.4,-.2,-.1,.1,.2,.4,.5], minor=True)

    ax.set_aspect('equal')

emdecorate=EMDecorator()

class ACDecorator(plotting.Decorator):
  def __init__(self):
```

```
    super().__init__()

    self.set_singleplot_fontsize('ax_label',30)
    self.set_singleplot_fontsize('subplot_title',30)
    self.set_singleplot_fontsize('cbar_tick',20)
    self.set_singleplot_fontsize('ax_tick',30)

    self.set_singleplot_axes_property('cbar_pad','-30%')  # compensate for call to␣
↪set_aspect() below
    self.set_multiplot_axes_property('cbar_pad','-30%')   # compensate for call to␣
↪set_aspect() below
    self.set_singleplot_axes_property('cbar_size','2%')   # compensate for call to␣
↪set_aspect() below
    self.set_multiplot_axes_property('cbar_size','2%')    # compensate for call to␣
↪set_aspect() below

  def extra_axes_commands(self, ax):
    ax.set_aspect(3)
    pass

acdecorate=ACDecorator()
# 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 = 7*wl_nm
unitcell_y = 0.7*unitcell_x
# Waveguide widths.
inc_a_x = 1500
inc_a_y = 80
# Shape of the waveguide.
# Use double coated geometry to control meshing around rib waveguide.
inc_shape = 'rib_double_coated'

slab_a_x = 2850
slab_a_y = 135

# areas included purely
slab_b_y = 500
coat_x = 50
coat_y = 100
coat2_x = 100
coat2_y = 200

#original old gmsh set that works
lc_bkg = 4  # background
lc_refine_1 = 8000  # edge of rib
lc_refine_2 = 3000   # edge of slab_a
lc_refine_3 = 50    # edge of coat
lc_refine_4 = 20    # edge of slab_b
lc_refine_5 = 4     # edge of coat2
```

```
# ##working set
# lc_bkg = .5  # background
# lc_refine_1 = 1000  # edge of rib
# lc_refine_2 = 400   # edge of slab_a
# lc_refine_3 = 10    # edge of coat
# lc_refine_4 = 5   # edge of slab_b
# lc_refine_5 = 1   # edge of coat2

# #scaled working set: doesn't work
# lc_bkg = 1  # background
# lc_refine_1 = 2000  # edge of rib
# lc_refine_2 = 600   # edge of slab_a
# lc_refine_3 = 10    # edge of coat
# lc_refine_4 = 4   # edge of slab_b
# lc_refine_5 = 1  # edge of coat2

##scaled working set: doesn't work
#lc_bkg = .1  # background
#lc_refine_1 = 200  # edge of rib
#lc_refine_2 = 75   # edge of slab_a
#lc_refine_3 = 50    # edge of coat
#c5  = 50  # edge of slab_b
#lc_refine_5 = 50  # edge of coat2




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

prefix_str = 'fig16-'

# 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, slab_b_y=slab_b_y,
                        coat_x=coat_x, coat_y=coat_y, coat2_x=coat2_x, coat2_y=coat2_
↪y,
                        material_bkg=materials.get_material("Vacuum"),
                        material_a=Si_110, #plt_mesh=True,
                        material_b=Si_110, material_c=materials.get_material(["Vacuum
↪"),
                        material_d=materials.get_material(["Vacuum"), material_
↪e=materials.get_material("Vacuum"),
```

```
                        symmetry_flag=False,
                        lc_bkg=lc_bkg, lc_refine_1=lc_refine_1, lc_refine_2=lc_refine_
↪2,
                        lc_refine_3=lc_refine_3, lc_refine_4=lc_refine_4, lc_refine_
↪5=lc_refine_5)
# Expected effective index of fundamental guided mode.
n_eff = wguide.material_a.n-0.1

# Calculate Electromagnetic Modes
print("starting EM pump modes")
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, wl_nm, n_eff=n_eff, debug=True)
#np.savez('wguide_data', sim_EM_pump=sim_EM_pump)
# npzfile = np.load('wguide_data.npz', allow_pickle=True)
# sim_EM_pump = npzfile['sim_EM_pump'].tolist()

print("starting EM Stokes modes")
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', allow_pickle=True)
# sim_EM_Stokes = npzfile['sim_EM_Stokes'].tolist()

print("starting EM field plotting ")
plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4,
                        ivals=[EM_ival_pump,EM_ival_Stokes],
                        ylim_min=0.435, ylim_max=0.435, EM_AC='EM_E', num_ticks=3,
                        prefix_str=prefix_str, pdf_png='png', ticks=True,
                        decorator=emdecorate, quiver_points=20,
                        comps=('Ex','Ey', 'Ez','Eabs','Et'), n_points=2000,␣
↪colorbar=True)

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4,
                        ivals=[EM_ival_pump,EM_ival_Stokes],
                        ylim_min=0.435, ylim_max=0.435, EM_AC='EM_H', num_ticks=3,
                        prefix_str=prefix_str, pdf_png='png', ticks=True,
                        decorator=emdecorate, quiver_points=20,
                        comps=('Hx','Hy', 'Hz','Habs','Ht'), n_points=2000,␣
↪colorbar=True)

# 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
print("starting acoustic modes")
sim_AC = wguide.calc_AC_modes(num_modes_AC, k_AC, EM_sim=sim_EM_pump, shift_Hz=shift_
↪Hz, debug=True)
# np.savez('wguide_data_AC', sim_AC=sim_AC)
# npzfile = np.load('wguide_data_AC.npz', allow_pickle=True)
# 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))

selected_AC_modes = [7, 13, 23]
print("AC modes selected for field plotting", selected_AC_modes)
print("plotting acoustic modes")
plotting.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str, ivals=selected_
↪AC_modes,
                          num_ticks=3, xlim_min=-.05, xlim_max=-0.05, ylim_min=-.1,␣
↪ylim_max=-0.1,
                          quiver_points=20, pdf_png='png',ticks=True, comps=('ux','ut',
↪'uz','uabs'),
                          decorator=acdecorate, colorbar=True)


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 Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] 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='', pdf_png='png')

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

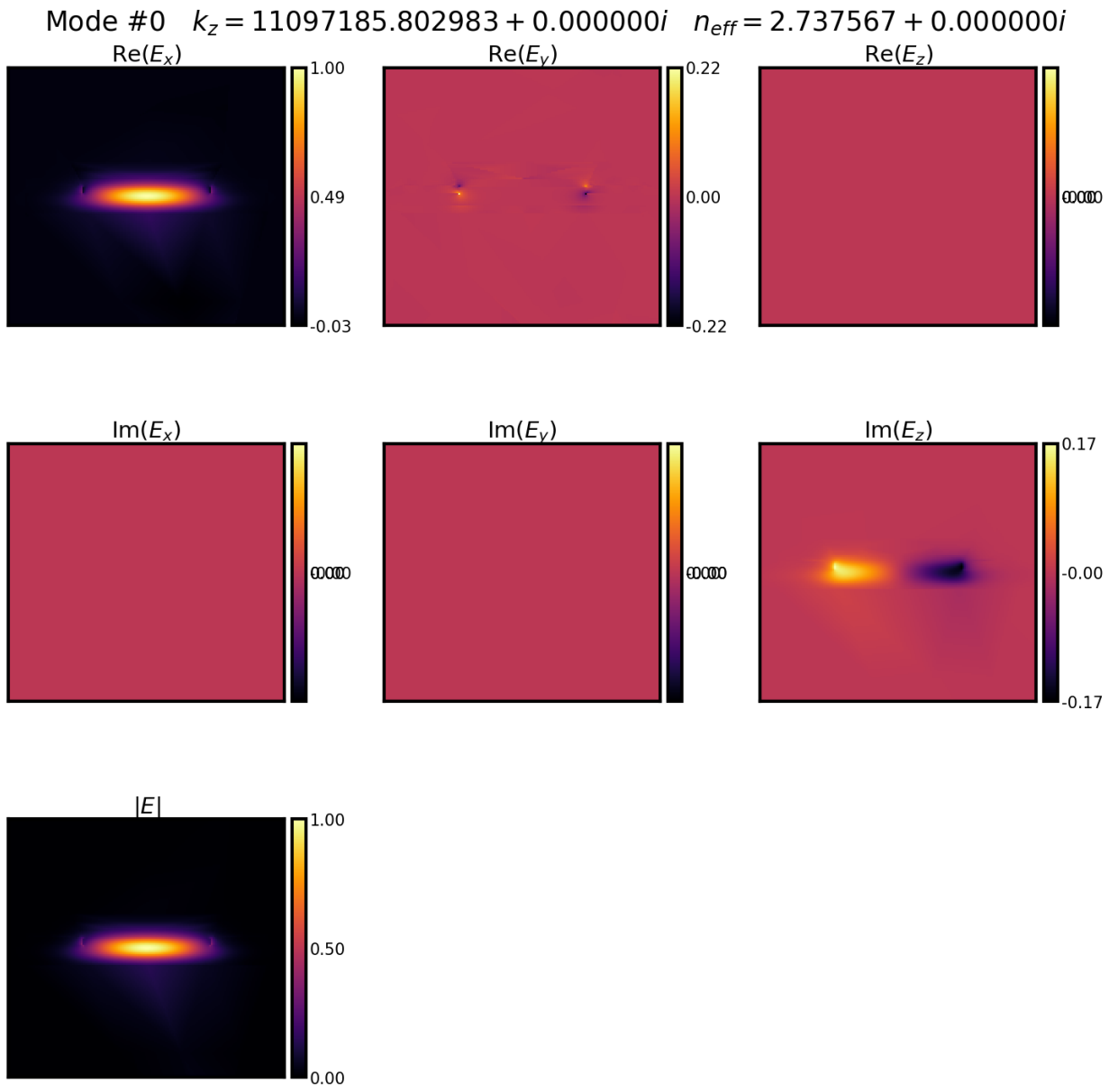


Fig. 4.82: Fundamental (symmetric TE-like) optical mode fields.

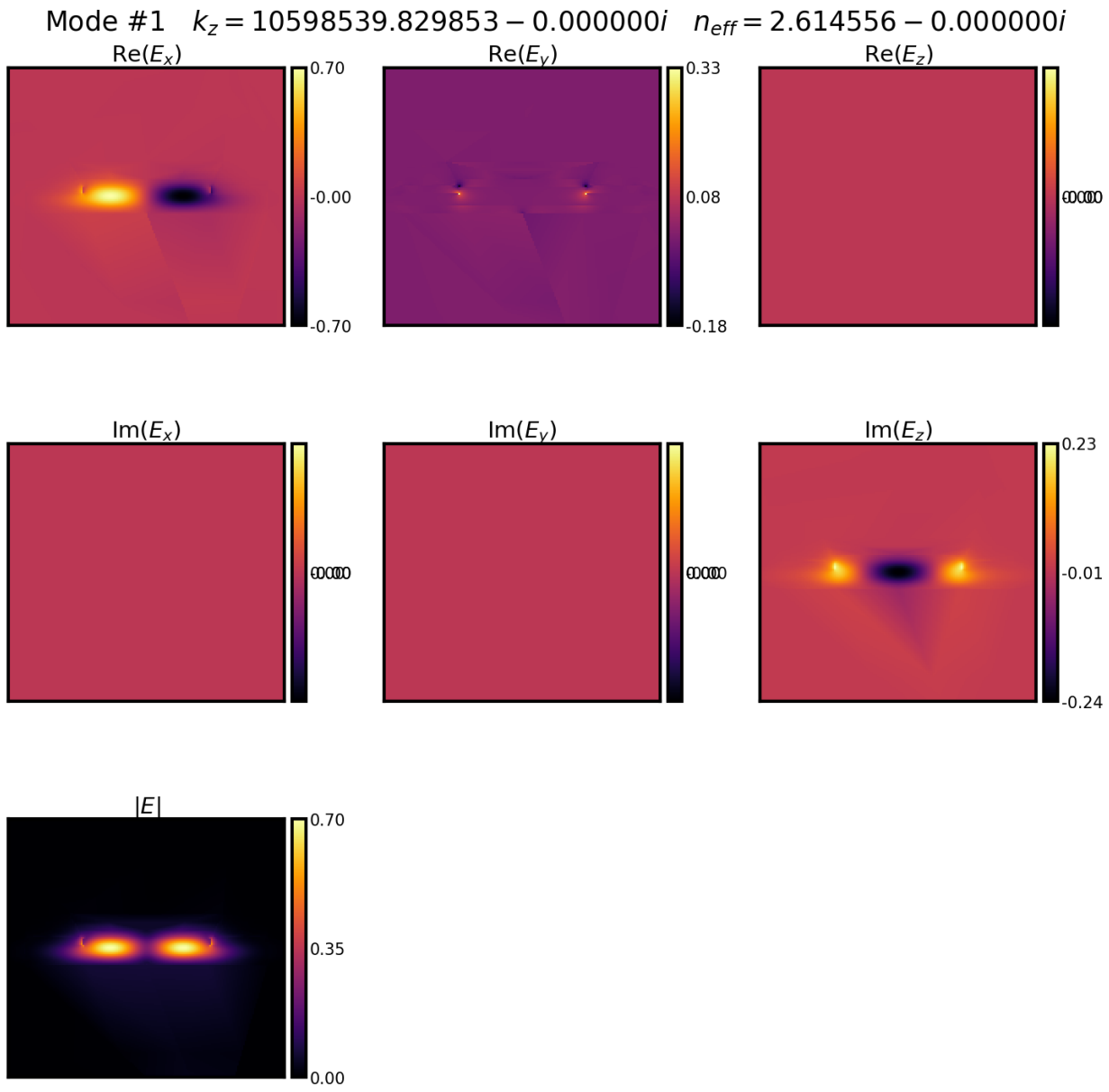


Fig. 4.83: 2nd lowest order (anti-symmetric TE-like) optical mode fields.

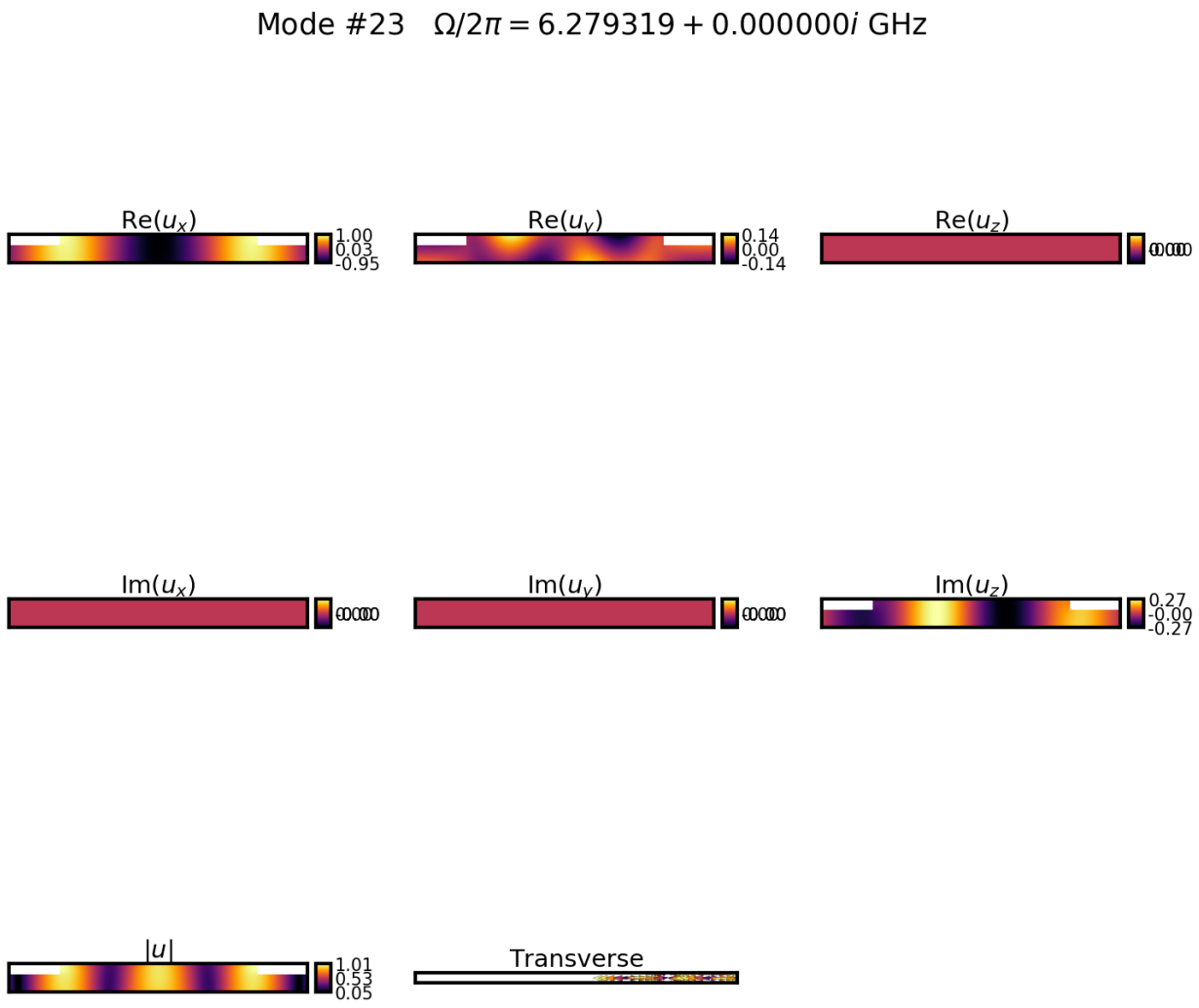


Fig. 4.84: Dominant high gain acoustic mode.

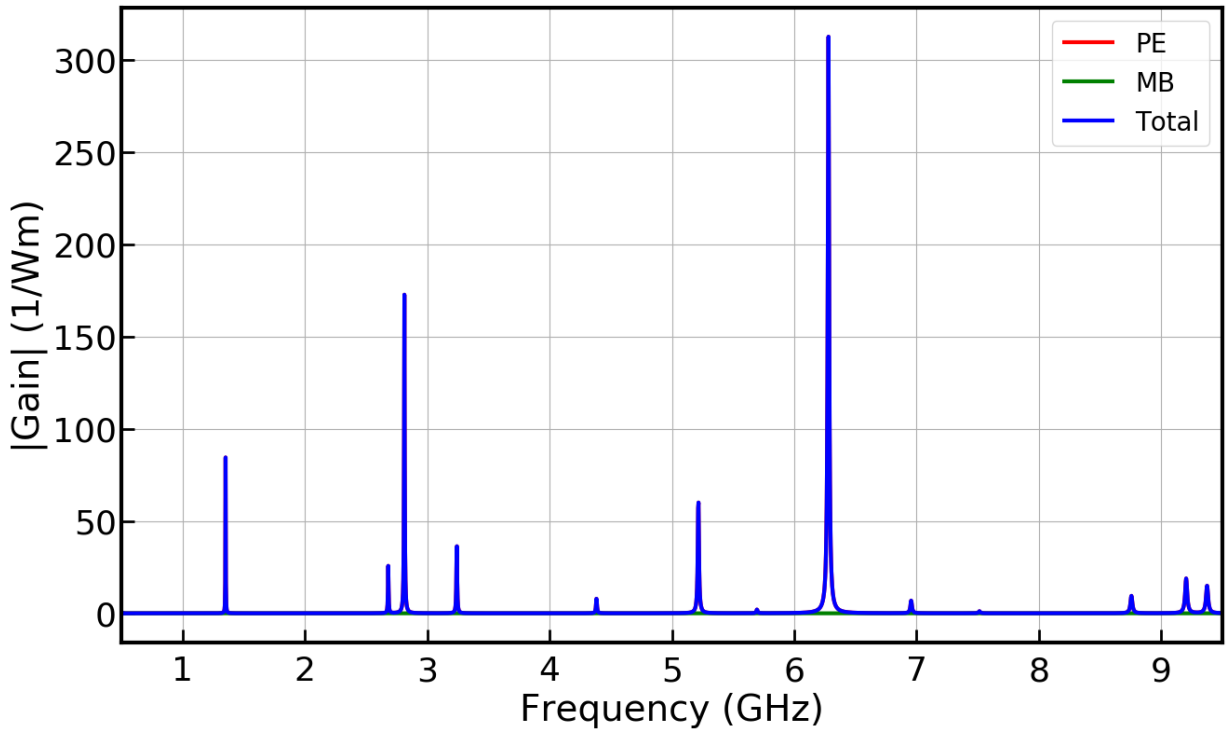


Fig. 4.85: Gain spectra showing gain due to photoelastic effect, gain due to moving boundary effect, and total gain.

4.3.10 LitEx 9 – Morrison *et al*, *Optica* (2017): BSBS in a chalcogenide rib waveguide

This example, in `simo-lit_09-Morrison-Optica_2017.py`, from the Sydney group examines backward SBS in a chalcogenide rib waveguide.

```
print("\n Simulation time (sec.)", (end - start))
""" Replicating the results of
    Compact Brillouin devices through hybrid
    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 = 6*wl_nm
unitcell_y = 0.75*unitcell_x
# Waveguide widths.
inc_a_x = 1900
inc_a_y = 680
# Shape of the waveguide.
inc_shape = 'rib_coated'

slab_a_x = 4000
slab_a_y = 1000

coat_x = 200
coat_y = 1000


# 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 = 100
# 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.get_material("Vacuum"),
                        material_a=materials.get_material("As2S3_2017_Morrison"), #␣
↪waveguide
                        material_b=materials.get_material("SiO2_2016_Smith"),     #␣
↪slab
                        material_c=materials.get_material("SiO2_2016_Smith"),     #␣
↪coating
                        lc_bkg=1, lc_refine_1=800.0, lc_refine_2=400.0)

# 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()

plotting.plot_mode_fields(sim_EM_pump, xlim_min=0.4, xlim_max=0.4, ivals=[EM_ival_
↪pump],
                          ylim_min=0.3, ylim_max=0.3, EM_AC='EM_E', num_ticks=3,
                          prefix_str=prefix_str, pdf_png='png')

# 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))


k_AC= 2.*9173922.1698

# Calculate Acoustic modes.
```

```
shift_Hz = 7.5*1e9 # select the lowest frequency to start FEM search from.
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.plot_mode_fields(sim_AC, EM_AC='AC', prefix_str=prefix_str,
     num_ticks=3, xlim_min=0.1, xlim_max=0.1, pdf_png='png')

# 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))

set_Q_factor = 190 # set the mechanic Q manually

# 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)
# 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 the Backward SBS gain of the AC modes.
print("\n Displaying results with negligible components masked out")
print("SBS_gain [1/(Wm)] PE contribution \n", masked_PE)
print("SBS_gain [1/(Wm)] MB contribution \n", masked_MB)
print("SBS_gain [1/(Wm)] total \n", masked)


# Construct the SBS gain spectrum, built from Lorentzian peaks of the individual␣
↪modes.
freq_min = 7.2  # GHz
freq_max = 8.1  # 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, pdf_png='png')

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

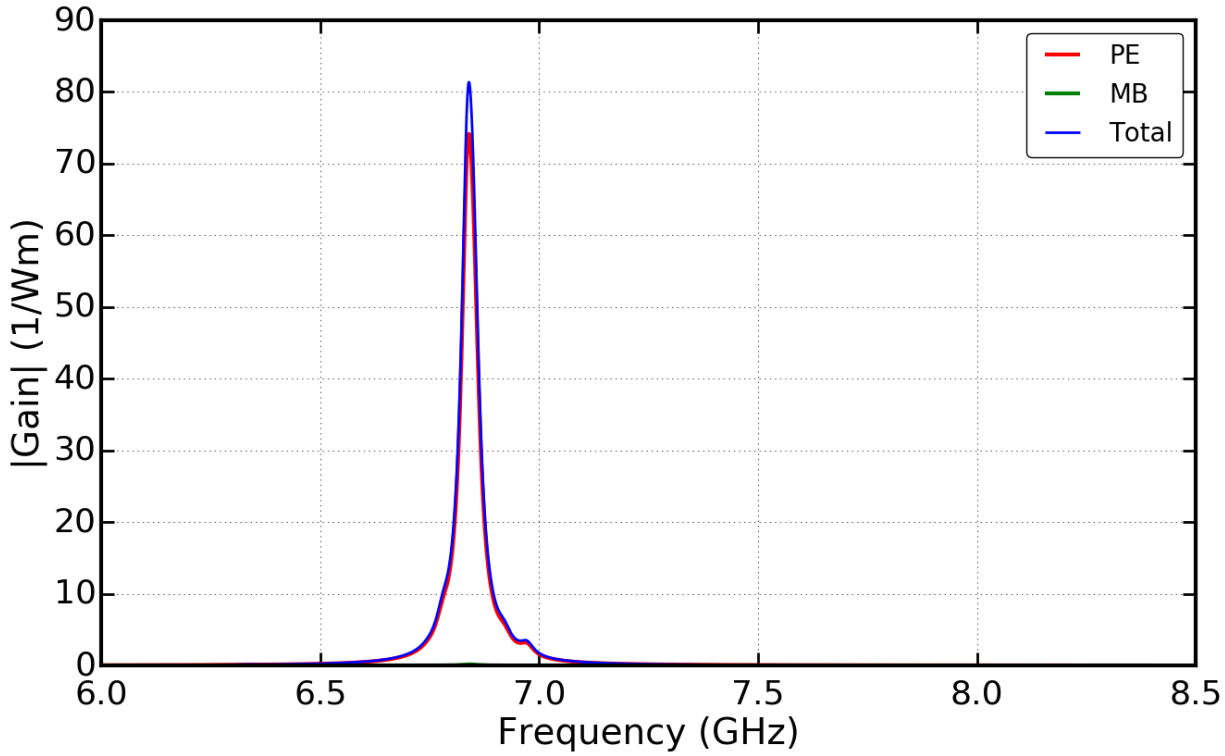


Fig. 4.86: Gain spectra showing gain due to photoelastic effect, gain due to moving boundary effect, and total gain.

CHAPTER

FIVE

PYTHON INTERFACE

This chapter provides a technical auto-generated summary of the NumBAT Python API.

The API consists of five core modules:

- `materials`, for defining waveguide materials and their properties;
- `objects`, for constructing waveguides from materials;
- `mode_calcs`, for the core calculation of electromagnetic and acoustic modes;
- `integration`, for performing calculations relating to SBS gain;
- `plotting`, for creating output plots of modes and gain functions.

5.1 materials module

The `materials` module provides functions for specifying all relevant optical and elastic properties of waveguide materials.

The primary class is *materials.Material* however users will rarely use this class directly. Instead, we generally specify material properties by writing new `.json` files stored in the folder `backend/materials_data`. Materials are then loaded to build waveguide structures using the function *materials.get_material()*.

class materials.**Material**(*data_file*)

Bases: object

Class representing a waveguide material.

Materials include the following properties and corresponding units:

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

load_cubic_crystal()

load_data_file(*dataloc*, *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

load_general_crystal()

load_tensors()

load_trigonal_crystal()

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.

class materials.**VoigtTensor4**(*sym*, *src_dict*, *src_file*)
Bases: object

dump()

load_isotropic()

read(*m*, *n*, *optional=False*)

rotate(*theta*, *rotation_axis*)

materials.**get_material**(*s*)

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.

5.2 objects module

The objects module provides functions for defining and constructing waveguides.

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*, *coat2_x=None*, *coat2_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=None*, *material_a=None*, *material_b=None*, *material_c=None*, *material_d=None*, *material_e=None*, *material_f=None*, *material_g=None*, *material_h=None*, *material_i=None*, *material_j=None*, *material_k=None*, *material_l=None*, *material_m=None*, *material_n=None*, *material_o=None*, *material_p=None*, *material_q=None*, *material_r=None*, *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*, *lc_refine_1=1.0*, *lc_refine_2=1.0*, *lc_refine_3=1.0*, *lc_refine_4=1.0*, *lc_refine_5=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', 'rib_double_coated', 'pedestal', 'onion', 'onion2', 'onion3'. 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_x** (*float*) – The width of the first coat layer around the inclusion.
- **coat_y** (*float*) – The thickness of the first coat layer around the inclusion.
- **coat2_x** (*float*) – The width of the second coat layer around the inclusion.
- **coat2_y** (*float*) – The thickness of the second 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 :Struct: parameters. If False, must provide mesh_file name of existing .mail that will be run despite :Struct: 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. It must be located in backend/fortran/msh/ Note: len(mesh_file) < 100.
- **plt_mesh** (*bool*) – Plot a png of the geometry and mesh files.
- **check_mesh** (*bool*) – Inspect the geometry and mesh files in gmsh.
- **lc_bkg** (*float*) – Length constant of meshing of background medium (smaller = finer mesh)
- **lc_refine_1** (*float*) – factor by which lc_bkg will be reduced on inclusion surfaces; lc_surface = lc_bkg / lc_refine_1. Larger lc_refine_1 = finer mesh.
- **lc_refine_2-6'** (*float*) – factor by which lc_bkg will be reduced on chosen surfaces; lc_surface = lc_bkg / lc_refine_2. see relevant .geo files.
- **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*, *debug=False*, ***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*, *debug=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 '.'

objects.**is_real_number**(*x*)

5.3 mode_calcs module

The mode_calcs module is responsible for the core engine to construct and solve the optical and elastic finite-element problems.

class mode_calcs.**Mode**(*simmo*, *m*)
Bases: object

This is a base class for both EM and AC modes.

add_mode_data(*d*)
Adds a dictionary of user-defined information about a mode.

Parameters **d** (*dict*) – Dict of (str, data) tuples of user-defined information about a mode.

analyse_mode(*v_x*, *v_y*, *m_Refx*, *m_Refy*, *m_Refz*, *m_Imfx*, *m_Imfy*, *m_Imfz*, *m_Absf*)
Perform a series of measurements on the mode f to determine polarisation fractions, second moment widths etc.

Parameters

- **v_x** (*array*) – Vector of x points.
- **v_y** (*array*) – Vector of y points.
- **m_Refx** (*array*) – Matrix of real part of fx.
- **m_Refy** (*array*) – Matrix of real part of fy.
- **m_Refz** (*array*) – Matrix of real part of fz.
- **m_Imfx** (*array*) – Matrix of imaginary part of fx.
- **m_Imfy** (*array*) – Matrix of imaginary part of fy.
- **m_Imfz** (*array*) – Matrix of imaginary part of fz.

center_of_mass()
Returns the centre of mass of the mode relative to the specified origin.

Return type float

center_of_mass_x()
Returns the x component moment of the centre of mass of the mode.

Return type float

center_of_mass_y()
Returns the y component moment of the centre of mass of the mode.

Return type float

field_fracs()
Returns tuple (*fx*, *fy*, *fz*, *ft*) of "fraction" of mode contained in *x*, *y*, *z* or *t* (sum of transverse *x*+*y*) components.

Note that *fraction* is defined through a simple overlap integral. It does not necessarily represent the fraction of energy density in the component.

Returns Tuple of mode fractions

Return type tuple(float, float, float, float)

get_mode_data()
Return dictionary of user-defined information about the mode.

Returns Dictionary of user-defined information about the mode.

Return type dict(str, obj)

is_AC()
Returns true if the mode is an acoustic mode.

Return type bool

is_EM()
Returns true if the mode is an electromagnetic mode.

Return type bool

is_poln_ex()
Returns true if mode is predominantly x-polarised (ie if *fx*>0.7).

Return type bool

is_poln_ey()
Returns true if mode is predominantly y-polarised (ie if *fy*>0.7).

Return type bool

is_poln_indeterminate()
Returns true if transverse polarisation is neither predominantly *x* or *y* oriented.

Return type bool

second_moment_widths()
Returns the second moment widths $(w_x, w_y, \sqrt{w_x^2 + w_y^2})$ of the mode relative to the specified origin.

Return type (float, float, float)

set_r0_offset(*x0*, *y0*)
Sets the transverse position in the grid that is to be regarded as the origin for calculations of center-of-mass.

This can be useful in aligning the FEM coordinate grid with a physically sensible place in the waveguide.

Parameters

- **x0** (*float*) – *x* position of nominal origin.
- **y0** (*float*) – *y* position of nominal origin.

w0()
Returns the combined second moment width $\sqrt{w_x^2 + w_y^2}$.

Return type float

wx()
Returns the x component moment of the second moment width.

Return type float

wy()
Returns the y component moment of the second moment width.

Return type float

class mode_calcs.**ModeAC**(*simmo*, *m*)
Bases: *mode_calcs.Mode*

Class representing a single acoustic (AC) mode.

analyse_mode(*v_x*, *v_y*, *m_Refx*, *m_Refy*, *m_Refz*, *m_Imfx*, *m_Imfy*, *m_Imfz*, *m_Absf*)

class mode_calcs.**ModeEM**(*simmo*, *m*)
Bases: *mode_calcs.Mode*

Class representing a single electromagnetic (EM) mode.

analyse_mode(*v_x*, *v_y*, *m_Refx*, *m_Refy*, *m_Refz*, *m_Imfx*, *m_Imfy*, *m_Imfz*, *m_Absf*)

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*, *calc_EM_mode_energy=False*, *calc_AC_mode_power=False*, *debug=False*)
Bases: object

Class for calculating the electromagnetic and/or acoustic modes of a Struct object.

Omega_AC(*m*)
Returns the frequency in 1/s of acoustic mode *m*.

Parameters **m** (*int*) – Index of the mode of interest.

Returns Angular requency Ω in Hz

Return type float

Omega_AC_all(*m*)
Return an array of the angular frequency in 1/s of all acoustic modes.

Returns numpy array of angular frequencies in 1/s

Return type array(float)

Qmech_AC(*m*)

Qmech_AC_all()

alpha_s_AC(*m*)

alpha_s_AC_all()

alpha_t_AC(*m*)

alpha_t_AC_all()

analyse_symmetries(*ptgrp*)

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: 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.

calc_acoustic_losses(*fixed_Q=None*)

get_modes()
Returns an array of class *Mode* containing the solved electromagnetic or acoustic modes.

Return type numarray(*Mode*)

is_AC()
Returns true if the solver is setup for an acoustic problem.

is_EM()
Returns true if the solver is setup for an electromagnetic problem.

kz_EM(*m*)
Returns the wavevector in 1/m of electromagnetic mode *m*.

Parameters **m** (*int*) – Index of the mode of interest.

Returns Wavevector k in 1/m.

Return type float

kz_EM_all()
Return an array of the wavevector in 1/m of all electromagnetic modes.

Returns numpy array of wavevectors in 1/m

Return type array(float)

linewidth_AC(*m*)

linewidth_AC_all()

neff(*m*)
Returns the effective index of EM mode *m*.

Parameters **m** (*int*) – Index of the mode of interest.

Return type float

neff_all()
Return an array of the effective index of all electromagnetic modes.

Returns numpy array of effective indices

Return type array(float)

ngroup_EM(*m*)
Returns the group index of electromagnetic mode *m*, if available, otherwise returns zero with a warning message.

Parameters **m** (*int*) – Index of the mode of interest.

Returns Group index of the mode.

Return type float

ngroup_EM_all()
Returns a numpy array of the group index of all electromagnetic modes, if available, otherwise returns a zero numarray with a warning message.

Returns numpy array of index of the mode.

Return type array(float)

ngroup_EM_available()
Returns true if a measure of the electromagnetic group index is available.

nu_AC(*m*)
Returns the frequency in Hz of acoustic mode *m*.

Parameters **m** (*int*) – Index of the mode of interest.

Returns Frequency ν in Hz

Return type float

nu_AC_all()
Return an array of the frequency in Hz of all acoustic modes.

Returns numpy array of frequencies in Hz

Return type array(float)

symmetry_classification(*m*)
If the point group of the structure has been specified, returns the symmetry class of the given mode.

Parameters **m** (*int*) – Index of the mode of interest.

Return type PointGroup

vg_AC(*m*)
Return group velocity of AC mode m in m/s

vg_AC_all()

vgroup_AC_available()
Returns true if a measure of the acoustic group velocity is available.

vp_AC(*m*)
Return phase velocity of all AC modes in m/s

vp_AC_all()
Return an array of the phase velocity in m/s of all acoustic modes.

Returns numpy array of elastic phase velocities in m/s

Return type array(float)

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:

5.4 integration module

The `integration` module is responsible for calculating gain and loss information from existing mode data.

`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

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

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

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

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

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 power loss for each mode in [1/s]. 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**(*m_n*, *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.

5.5 plotting module

The `plotting` module is responsible for generating all standard graphs.

class plotting.**Decorator**
Bases: `object`

extra_axes_commands(*ax*)
Function to make additions to a standard plot.

This function is called after all other plotting operations are performed. The default implementation does nothing. By subclassing :Decorator: and implementing this function, users may add extra features to a plot.

get_axes_property(*lab*)

get_cmap_limits(*comp*)

get_font_size(*lab*)

is_single_plot()
Returns True if this Decorator is for a single axes plot such as a spectrum or spatial map of a single field component.

set_cmap_limits(*d*)
Specify the lower and upper contour plot limits for a field component plot.

Parameters **d** (*dict*) – Dictionary mapping component ('x','y','z') to (zlo, zhi) tuple for contour plots.

set_multiplot_axes_property(*label*, *prop*)
Add or override an axes property for a multiple axes plot corresponding to the given label.

set_multiplot_fontsize(*label*, *sz*)
Override a font size for a mutiple axes plot corresponding to the given label.

set_singleplot_axes_property(*label*, *prop*)
Add or override an axes property for a single plot corresponding to the given label.

set_singleplot_fontsize(*label*, *sz*)
Override a font size for a single plot corresponding to the given label.

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=''*, *decorator=None*, *show_gains='All'*)
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_all_components**(*v_x*, *v_y*, *v_x_q*, *v_y_q*, *v_XX*, *v_YY*, *v_plots*, *vq_plots*, *v_labels*, *plps*, *sim_wguide*, *ival*, *suppress_imimre*, *vanilla_v_x*, *vanilla_v_y*)

plotting.**plot_component**(*v_x*, *v_y*, *v_XX*, *v_YY*, *plot*, *label*, *plps*, *sim_wguide*, *ival*, *comp*)

plotting.**plot_component_axes**(*ax*, *v_x*, *v_y*, *v_XX*, *v_YY*, *plot*, *v_label*, *plps*)

plotting.**plot_component_quiver**(*ax*, *v_x_q*, *v_y_q*, *vq_plots*, *plps*)

plotting.**plot_filename**(*plps*, *ival*, *label=None*)

plotting.**plot_mode_data**(*ax*, *v_x*, *v_y*, *v_plots*, *plps*, *sim_wguide*, *ival*, *vanilla_v_x*, *vanilla_v_y*)

plotting.**plot_mode_fields**(*sim_wguide*, *ivals=None*, *n_points=501*, *quiver_points=30*, *xlim_min=None*, *xlim_max=None*, *ylim_min=None*, *ylim_max=None*, *EM_AC='EM_E'*, *num_ticks=None*, *colorbar=True*, *contours=False*, *contour_lst=None*, *stress_fields=False*, *pdf_png='png'*, *prefix_str=''*, *suffix_str=''*, *ticks=False*, *comps=None*, *decorator=None*, *suppress_imimre=False*, *modal_gains_PE=None*, *modal_gains_MB=None*, *modal_gains=None*)

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.
- **modal_gains** (*float array*) – Pre-calculated gain for each acoustic mode given chosen optical fields.

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.**plot_supertitle**(*plps*, *sim_wguide*, *ival*)

plotting.**plt_mode_fields**(*sim_wguide*, *ivals=None*, *n_points=501*, *quiver_points=50*, *xlim_min=None*, *xlim_max=None*, *ylim_min=None*, *ylim_max=None*, *EM_AC='EM_E'*, *num_ticks=None*, *colorbar=True*, *contours=False*, *contour_lst=None*, *stress_fields=False*, *pdf_png='png'*, *prefix_str=''*, *suffix_str=''*, *ticks=False*, *comps=None*, *decorator=None*, *suppress_imimre=False*, *modal_gains_PE=None*, *modal_gains_MB=None*, *modal_gains=None*)

plotting.**save_figure**(*plt*, *figfile*)

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

CHAPTER

SIX

FORTRAN BACKEND

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.

6.1 FEM Mode Solvers

6.1.1 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, lc_refine_1};
Point(10) = {d/2+rx/2, -h+s1+s2, 0, lc_refine_1};
Point(11) = {d/2-rx/2, -h+s1+s2+ry, 0, lc_refine_1};
Point(12) = {d/2+rx/2, -h+s1+s2+ry, 0, lc_refine_1};
```

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. Remember to separate out your inclusion from the background by highlighting it when asked for “hole boundaries”. If the inclusion is complicated it is best to carve up the background area into smaller simpler areas that don’t have any inclusions (“holes”), for example see slot coated.

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.

6.1.2 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 'lc_refine_1' 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 'lc_refine_1' etc.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

N

O

P

Q

R

S

V

W

Z