NumBAT Documentation

Release 1.9.0

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

Oct 31, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION TO NUMBAT

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 arbitrary 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 the article B.C.P Sturmberg at al., "Finite element analysis of stimulated Brillouin scattering in integrated photonic waveguides", *J. Lightwave Technol.* **37**, 3791-3804 (2019), available at https://dx.doi.org/10.1109/JLT.2019.2920844.

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 Citing NumBAT

If you use NumBAT in published work, we would appreciate a citation to B.C.P Sturmberg at al., "Finite element analysis of stimulated Brillouin scattering in integrated photonic waveguides", *J. Lightwave Technol.* **37**, 3791-3804 (2019), available at https://dx.doi.org/10.1109/JLT.2019.2920844 and https://arxiv.org/abs/1811.10219, and a link to the github page at https://github.com/michaeljsteel/NumBAT.

1.4 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.

We thank Christian Wolff, Mike Smith and Mikolaj Schmidt for contributions.

1.5 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`.

1.6 Support

Development of NumBAT has been supported in part by the Australian Research Council under Discovery Projects DP130100832, DP160101691, DP200101893 and DP220100488.

1.7 Release notes

1.7.1 Version 2.0

A number of API changes have been made in NumBAT 2.0 to tidy up the interface and make plotting and analysis simpler and more powerful. You will need to make some changes to existing files to run in NumBAT 2.0. Your best guide to new capabilities and API changes is to look through the code in the tutorial examples.

Some key changes you will need to make are as follows:

- The waveguide class `Struct` has been renamed to `Structure`
- The interface for creating materials has changed. You now call the `materials.make_material(` *name* `)` function. For example `material_a = materials.make_material('Vacuum')`
- To access an existing material in an existing `Struture` object (usually in a variable called `wguide` use `wguide.get_material(` *label* `)` For example, `mat_a = wguide.get_material('b')` where the allowed labels are `bkg` and the letters `a` to `r`.
- The member name for refractive index in a `Material` object has changed from `n` to `refindex_n`.
- The member name for density in a `Material` object has changed from `n` to `rho`.
- **Due to a change in parameters, the function `plotting.gain_spectra` is deprecated and replaced by `plotting.plot_gain_spectra` with the following changes:**
  - The frequency arguments `freq_min` and `freq_max` should now be passed in units of Hz, not GHz.
  - The argument `k_AC` has been removed.

- In all functions the parameter `prefix_str` has been renamed to `prefix` for brevity.

1.8 What does NumBAT actually calculate?

NumBAT performs three main types of calculations given a particular waveguide design:

- solve the electromagnetic modal problem using the finite element method (FEM).
- solve the elastic modal problem using FEM.
- calculate Brillouin gain coefficients and linewidths for a given triplet of two optical and one elastic mode, and use this to generate gain spectra.

Here we specify the precise mathematical problems been solved. For further details, see the NumBAT paper in the Journal of Lightwave Technolgoy at at https://dx.doi.org/10.1109/JLT.2019.2920844.

1.8.1 1. Electromagnetic modal problem

WRITE ME

1.8.2 2. Elastic modal problem

WRITE ME

1.8.3 3. SBS gain calculation modal problem

WRITE ME

CHAPTER

TWO

INSTALLING NUMBAT

2.1 Installing on Linux

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

NumBAT has been developed and tested on Ubuntu 23.04 with the following package versions: Python 3.11.4, Numpy 1.24.2, Arpack-NG, Suitesparse 7.1.0, and Gmsh 4.8.4. NumBAT also depends on the BLAS and LAPACK libraries. We strongly recommend linking NumBAT against optimised versions, such as the MKL library provided in the free Intel OneAPI library.

NumBAT has also been successfully installed by users on Debian, RedHat and on Windows 10 (installing Ubuntu after enabling the Windows Subystem for Linux - see 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. Please get in touch with Michael Steel <michael.steel@mq.edu.au> if you run into troubles.

Before installing, you may wish to update your system

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

There is no need to install NumBAT in a central location such as */usr/local* though you may choose to do so.

To download the current version from the git repository and install any missing library dependencies, use

```
$ git clone https://github.com/michaeljsteel/NumBAT.git
$ cd NumBAT/
$ sudo make installdeps
```

In this documentation, we indicate the root NumBAT directory (e.g. `/usr/local/NumBAT`, `/home/mike/NumBAT`) with <NumBAT>.

Before attempting to build the NumBAT module, open the file <NumBAT>`/backend/fortran/Makefile` in a text editor and check the settings associated with the variables `PLAT` that control the preferred math library. The starting makefile is setup to look for the Intel OneAPI library which is the recommended configuration.

You may now build the NumBAT module by running the following in the <NumBAT> directory.

```
$ make build
```

If this completes without error, you may run a short test suite with

```
$ make tests
```

To build the pdf documentation you are currently reading, use

```
$ make docs
```

Note however most of the figures will only be available after you have run all the example problems in the `tutorial lit_ex` and `JOSAB_tutorial` directories. This can be done by running `make` in each of those directories. Be aware that some of these problems are quite large and may require some time to complete depending on your computer's performance.

2.1.1 Other build configurations

The Fortran components (NumBAT source code and libraries) have been successfully compiled with Intel's `ifortran` as well as GCC's open-source `gfortran`. In this documentation we use `gfortran`, but this can be easily adjusted in `NumBAT/backend/fortran/Makefile`

2.2 Installing on MacOS

NumBAT can also be installed on MacOS, though this is currently somewhat experimental and has only been performed on certain versions of MacOS. Any comments on difficulties and solutions will be appreciated.

The following steps have worked for us:

1. Open a terminal window on your desktop.
2. Make a folder for NumBAT studies and clone the github repository:

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

3. If it is not already on your system, install the MacPorts package manager at this page.
4. Install the Gmsh mesh generation tool at this page. Just the main Gmsh installer is fine. The SDK and other features are not required.

   Install the Gmsh application into your Applications folder by dragging the Gmsh icon into Applications.
5. Install a current gcc (we used gcc):

```
$ sudo port install gcc13
```

6. Install the Lapack and Blas linear algebra libraries:

```
$ sudo port install lapack
```

   $ sudo port install blas
7. Install the Arpack eigensolver:

```
$ sudo port install arpack
```

8. Install the SuiteSparse matrix algebra suite:

```
$ sudo port install suitesparse
```

9. Install a current python (we used python 3.12):

   Use the standard installer at https://www.python.org/downloads/macos/.

   (Note that this will install everything in */Library/Frameworks* and **not** override the System python in */System/Library/Frameworks*.)
10. Install some python packages not included by default:

```
$ pip3 install numpy scipy matplotlib psutil
```

11. Check that the python installs work and create a matplotlib .config directory:

```
$ python3.12
$ import matplotlib
$ import numpy
$ import scipy
$ import psutil
```

12. Install the NumBAT matplotlib style file:

```
$ mkdir -pR $HOME/.matplotlib/stylelib/
$ cp <NumBAT>/backend/NumBATstyle.mplstyle $HOME/.matplotlib/stylelib
```

13. Move to the NumBAT fortran directory:

```
$ cd backend/fortran
```

14. Open the Makefile in a text editor and edit the lines at the top of the file so that the line *PLAT=MacOS* is active and the others are commented out.

15. Now we can build NumBAT

```
$ make - C Makefile
```

2.2.1 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 A First Calculation

We're now ready to start using NumBAT.

Let's jump straight in and run a simple calculation. Later in the chapter, we go deeper into some of the details that we will encounter in this first example.

3.1.1 Tutorial 1 – Basic SBS Gain Calculation

Simulations with NumBAT are generally carried out using a python script file.

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

Move into the tutorials directory and then run the script by entering:

```
$ python3 simo-tut_01-first_calc.py
```

After a short while, you should see some values for the SBS gain printed to the screen. In many more tutorials in the subsequent chapters, we will meet much more convenient forms of output, but for now let's focus on the steps involved in this basic calculation.

The sequence of operations (annotated in the source code below as Step 1, Step 2, etc) is:

1. **Add the NumBAT install directory to Python's module search path and then import** the NumBAT python modules.
2. Set parameters to define the structure shape and dimensions.
3. Set parameters determining the range of electromagnetic and elastic modes to be solved.
4. Construct the waveguide with `objects.Structure` out of a number of `materials.Material` objects. The generated mesh is shown in the figure below.
5. Solve the electromagnetic problem at a given *free space* wavelength λ. The function `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 elastic wavenumber from the difference of the pump and Stokes propagation constants and solve the elastic problem. `mode_calcs.calc_AC_modes()` returns an object containing the elastic mode profiles, frequencies and potentially other data at the specified propagation constant `k_AC`.
9. Display the elastic frequencies in Hz using `mode_calcs.nu_AC_all()`.
10. Calculate the total SBS gain, contributions from photoelasticity and moving boundary effects, and the elastic loss using `integration.gain_and_qs()`, and print them to the screen.

Note from this description that the eigenproblems for the electromagnetic and acoustic problems are framed in opposite senses. The electromagnetic problem finds the wavenumbers $k_{z,n}(\omega)$ (or equivalently the effective indices) of the modes at a given free space wavelength (ie. at a specified frequency $\omega = 2\pi c/\lambda$). The elastic solver, however, works in the opposite direction, finding the elastic modal frequencies $\nu_n(q_0)$ at a given elastic propagation constant q_0. While this might seem odd at first, it is actually the natural way to frame SBS calculations.

We emphasise again, that for convenience, the physical dimensions of waveguides are specified in nanometres. All other quantities in NumBAT are expressed in the standard SI base units.

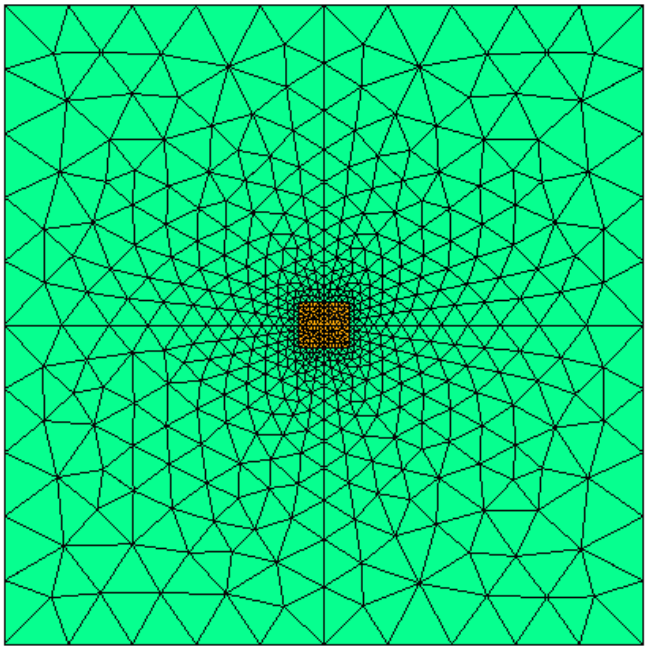


Fig. 1: Generated mesh for this example as displayed by `guide.check_mesh()`.

Here's the full source code for this tutorial:

```
""" 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
```

(continues on next page)

(continued from previous page)

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

# Naming conventions
# AC: acoustic
# EM: electromagnetic
# q_AC: acoustic wavevector

start = time.time()
print('\n\nCommencing NumBAT tutorial 1')

# Step 2
# Geometric Parameters - all in nm.
lambda_nm = 1550 # Wavelength of EM wave in vacuum.
# Unit cell must be large to ensure fields are zero at boundary.
unitcell_x = 2.5*lambda_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.
# to save the geometry and mesh as png files in backend/fortran/msh/
wguide = objects.Structure(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
                           material_bkg=materials.make_material("Vacuum"),
                           material_a=materials.make_material("Si_2016_Smith"),
                           lc_bkg=.1, # in vacuum background
                           lc_refine_1=5.0, # on cylinder surfaces
                           lc_refine_2=5.0) # on cylinder center

# Note use of rough mesh for demonstration purposes by turning this line on.
#wguide.check_mesh()

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

# Step 5
```

(continues on next page)

(continued from previous page)

```
# Estimate expected effective index of fundamental guided mode.
n_eff = wguide.get_material('a').refindex_n-0.1

# Calculate the Electromagnetic modes of the pump field.
sim_EM_pump = wguide.calc_EM_modes(num_modes_EM_pump, lambda_nm, n_eff)

# Display the wavevectors of EM modes.
v_kz=sim_EM_pump.kz_EM_all()
print('\n k_z of electromagnetic 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(lambda_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
q_AC = np.real(sim_EM_pump.kz_EM(0) - sim_EM_Stokes.kz_EM(0))

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

# Step 7
# Calculate Acoustic modes, using the mesh from the EM calculation.
sim_AC = wguide.calc_AC_modes(num_modes_AC, q_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:.5f}'.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_tot, SBS_gain_PE, SBS_gain_MB, linewidth_Hz, Q_factors, alpha = integration.
↪gain_and_qs(
    sim_EM_pump, sim_EM_Stokes, sim_AC, q_AC, EM_ival_pump=EM_ival_pump,
    EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival, fixed_Q=set_q_factor)

# SBS_gain_tot, SBS_gain_PE, SBS_gain_MB are 3D arrays indexed by pump, Stokes and␣
↪acoustic mode
# Extract those of interest as a 1D array:
SBS_gain_PE_ij = SBS_gain_PE[EM_ival_pump,EM_ival_Stokes,:]
SBS_gain_MB_ij = SBS_gain_MB[EM_ival_pump,EM_ival_Stokes,:]
SBS_gain_tot_ij = SBS_gain_tot[EM_ival_pump,EM_ival_Stokes,:]
```

(continues on next page)

(continued from previous page)

```
# Print the Backward SBS gain of the AC modes.
print("\nContributions to SBS gain [1/(WM)]")
print("Acoustic Mode number | Photoelastic (PE) | Moving boundary(MB) | Total")

for (m, gpe, gmb, gt) in zip(range(num_modes_AC), SBS_gain_PE_ij, SBS_gain_MB_ij, SBS_
↪gain_tot_ij):
    print('{0:8d}  {1:18.6e}  {2:18.6e}  {3:18.6e}'.format(m, gpe, gmb, gt))


# Mask negligible gain values to improve clarity of print out.
threshold = 1e-3
masked_PE = np.where(np.abs(SBS_gain_PE_ij)>threshold, SBS_gain_PE_ij, 0)
masked_MB = np.where(np.abs(SBS_gain_MB_ij)>threshold, SBS_gain_MB_ij, 0)
masked_tot = np.where(np.abs(SBS_gain_tot_ij)>threshold, SBS_gain_tot_ij, 0)

print("\n Displaying gain results with negligible components masked out:")

print("AC Mode number | Photoelastic (PE) | Moving boundary(MB) | Total")
for (m, gpe, gmb, gt) in zip( range(num_modes_AC), masked_PE, masked_MB, masked_tot):
    print('{0:12d}  {1:19.6e}  {2:19.6e}  {3:16.6e}'.format(m, gpe, gmb, gt))


end = time.time()
print("\nSimulation time: {0:10.3f} secs.\n\n".format(end - start))
```

In the next three chapters, we meet many more examples that show the different capabilities of NumBAT and provided comparisons against analytic and experimental results from the literature.

For the remainder of this chapter, we will explore some of the details involved in specifying a wide range of waveguide structures.

3.2 General Simulation Procedures

Simulations with NumBAT are generally carried out using a python script file. This file is kept in its own directory which may or may not be within your NumBAT tree. 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 directory for future inspection, manipulation, plotting, etc.

These files can be edited using your choice of text editor (for instance `nano` or `vim`) or an IDE (for instance MS Visual Code or `pycharm`) which allow you to run and debug code within the IDE.

To save the results from a simulation that are displayed upon execution (the print statements in your script) use:

```
$ python3 ./simo-tut_01-first_calc.py | tee log-simo.log
```

To have direct access to the simulation objects upon the completion of a 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`.

3.3 Script Structure

As with our first example above, most NumBAT scripts proceed with a standard structure:

- importing NumBAT modules
- 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*.

3.4 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 human-readable `.json` files, which are stored in the directory `<NumBAT>/backend/material_data`.

These files not only provide the numerical values for optical and acoustic variables, but provide links to the origin of the data. Often they are taken from the literature and the naming convention allows users to select from different parameter values chosen by different authors for the same nominal material.

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 library contains the following materials:

- **Vacuum (or air)**
  - `Vacuum`
- **The chalcogenide glass Arsenic tri-sulfide**
  - `As2S3_2016_Smith`
  - `As2S3_2017_Morrison`
  - `As2S3_2021_Poulton`
- **Fused silica**
  - `SiO2_2013_Laude`
  - `SiO2_2015_Van_Laer`
  - `SiO2_2016_Smith`
  - `SiO2_2021_Smith`
  - `SiO2_smf28.json`
  - `SiO2GeO2_smf28.json`

- **Silicon**
  - `Si_2012_Rakich`
  - `Si_2013_Laude`
  - `Si_2015_Van_Laer`
  - `Si_2016_Smith`
  - `Si_2021_Poulton`
  - `Si_test_anisotropic`
- **Silicon nitride**
  - `Si3N4_2014_Wolff`
  - `Si3N4_2021_Steel`
- **Gallium arsenide**
  - `GaAs_2016_Smith`
- **Germanium**
  - `Ge_cubic_2014_Wolff`
- **Lithium niobate**
  - `LiNbO3_2021_Steel`
  - `LiNbO3aniso_2021_Steel`

Materials can easily be added to this library 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.

3.5 Waveguide Geometries

NumBAT encodes different waveguide structures through finite element meshes constructed using the `.geo` language used by the open source tool `Gmsh`. Most users will find they can construct all waveguides of interest using the existing templates. However, new templates can be added by adding a new `.geo` file to the `<NumBAT>/backend/fortran/msh` directory and adding a small quantity of python code to `<NumBAT>/backend/objects.py`. Interested users can get in touch with <michael.steel@mq.edu.au>.

The following figures give some examples of how material types and physical dimensions are represented in the mesh geometries. In particular, for each structure template, they identify the interpretation of the dimensional parameters (`inc_a_x`, `slab_b_y`, etc), material labels (`material_a`, `material_b` etc), and the grid refinement parameters (`lc_bkg`, `lc_refine_1`, `lc_refine_2`, etc). The captions for each structure also identify the mesh geometry template files in the directory `<NumBAT>/backend/fortran/msh` with filenames of the form `<prefix>_msh_template.geo` which define the structures and can give ideas for developing new structure files.

The NumBAT code for creating all these structures can be found in `<NumBAT>/docs/source/images/make_meshfigs.py`.

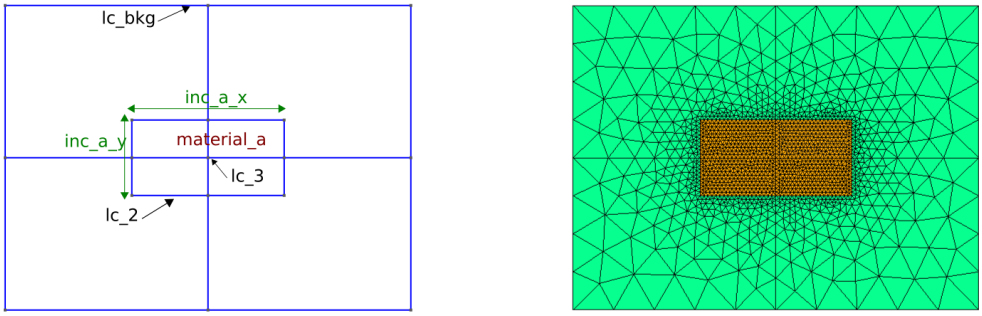


Fig. 2: Rectangular waveguide using shape `rectangular` (template `oneincl_msh`).

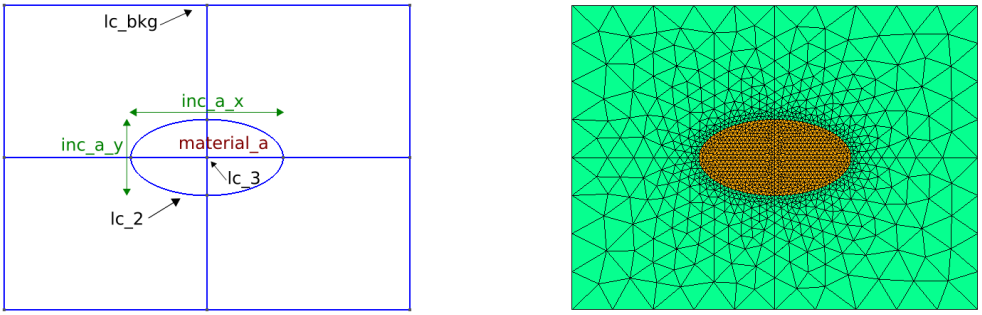


Fig. 3: Elliptical waveguide using shape `circular` (template `oneincl_msh`).

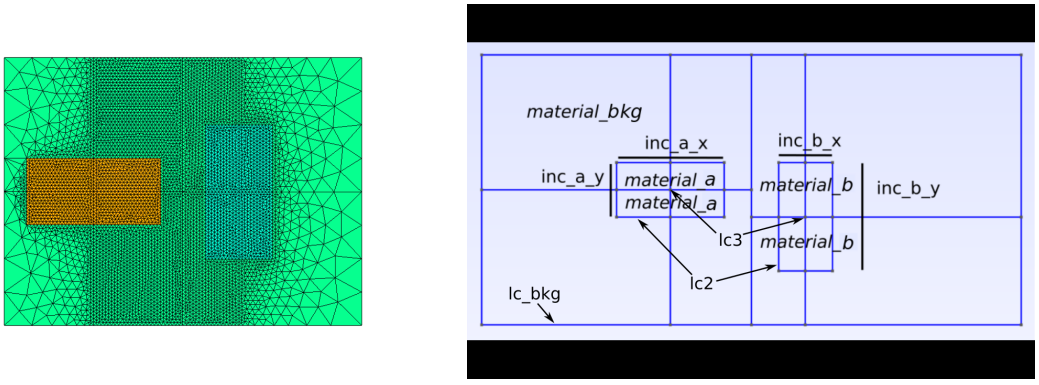


Fig. 4: Coupled rectangular waveguides using shape `rectangular` (template `twoincl_msh`).

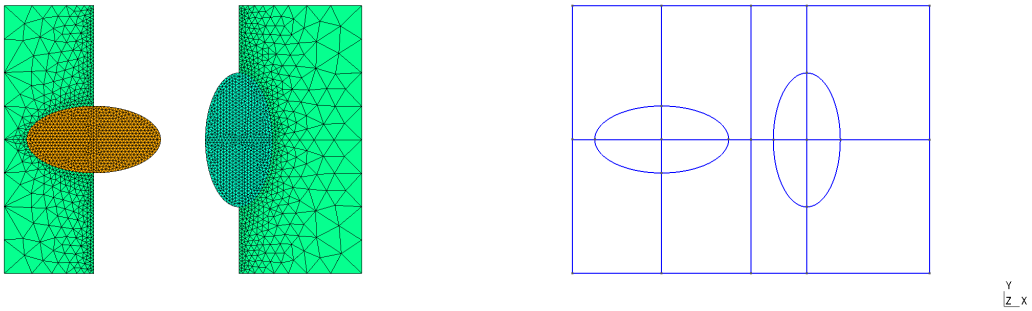


Fig. 5: Coupled circular waveguides using shape `circular` (template `twoincl_msh`). There appers to be a bug here!

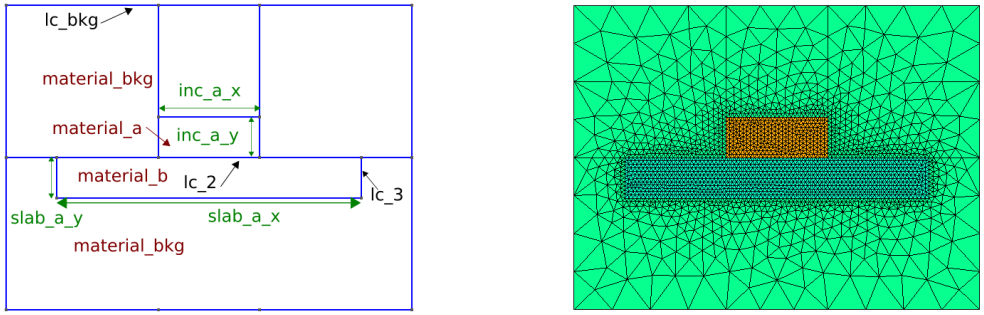


Fig. 6: A conventional rib waveguide using shape `rib` (template `rib`).

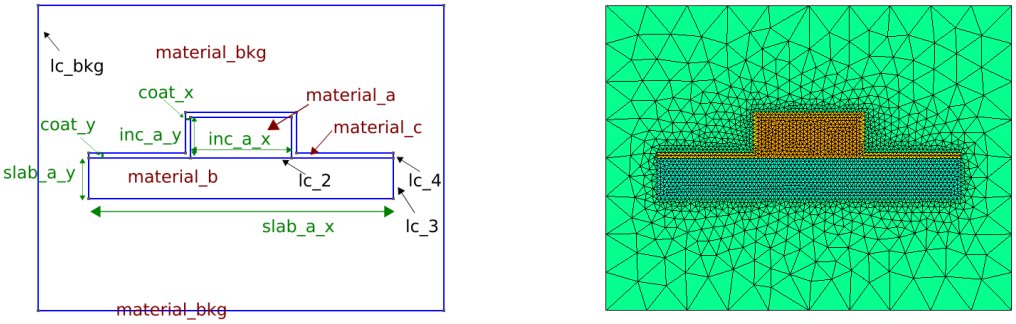


Fig. 7: A coated rib waveguide using shape `rib_coated` (template `rib_coated`).

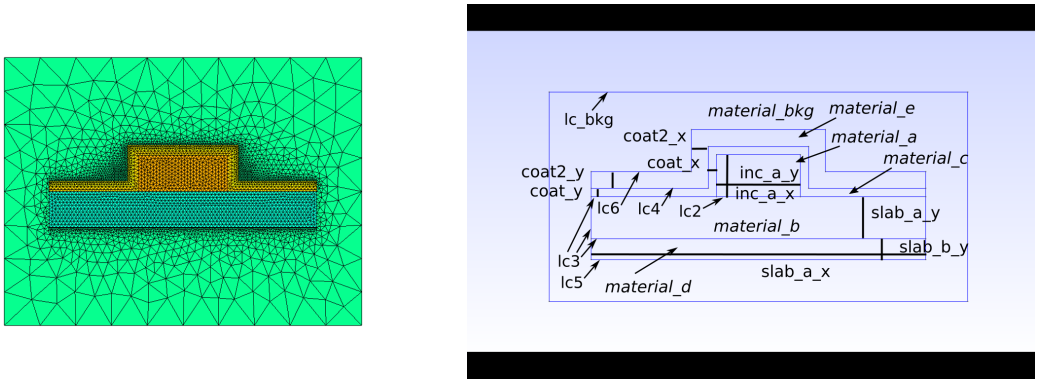


Fig. 8: A rib waveguide on two substrates using shape `rib_double_coated` (template `rib_double_coated`).

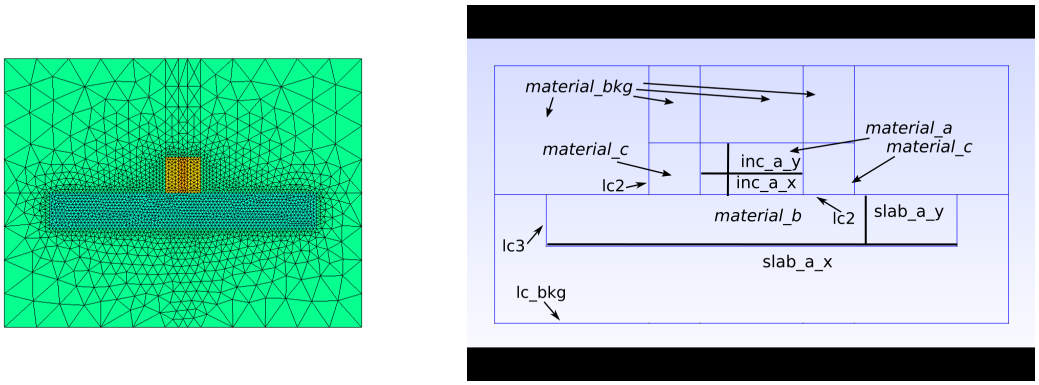


Fig. 9: A slot waveguide using shape `slot` (`material_a` is low index) (template `slot`).

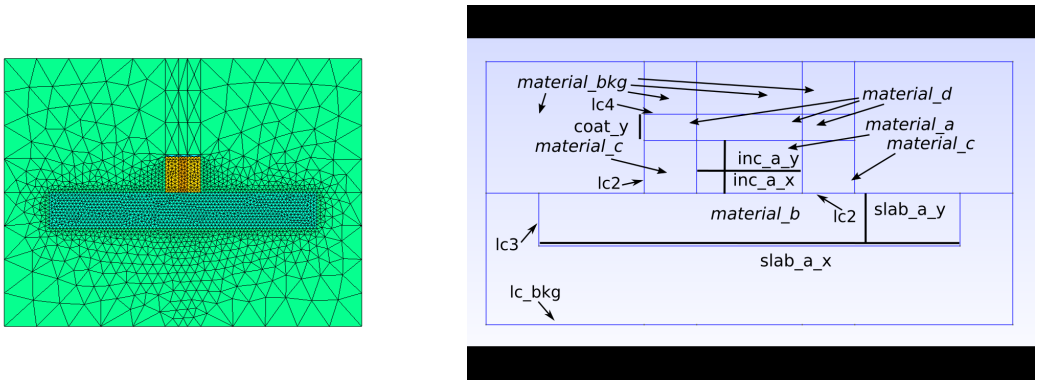


Fig. 10: A coated slot waveguide using shape `slot_coated` (`material_a` is low index) (template `slot_coated`).

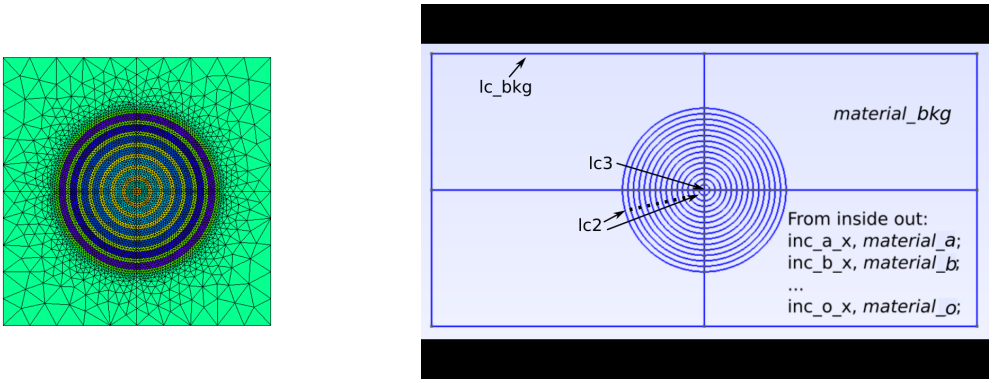


Fig. 11: A many-layered concentric structure using shape `onion` (template `onion`).

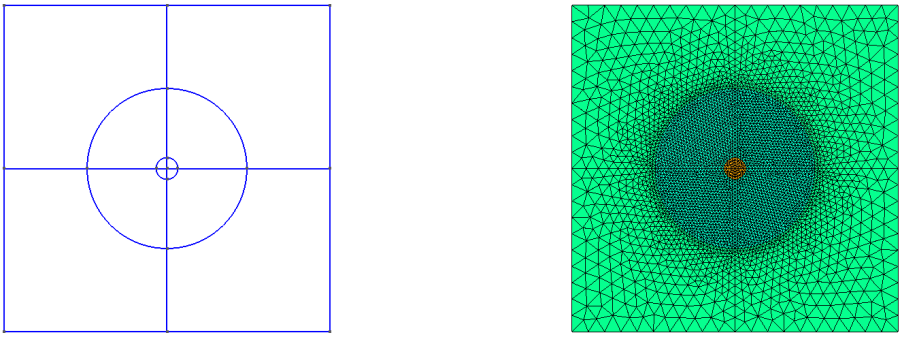


Fig. 12: A two-layered concentric structure with background using shape `onion2` (template `onion2`).

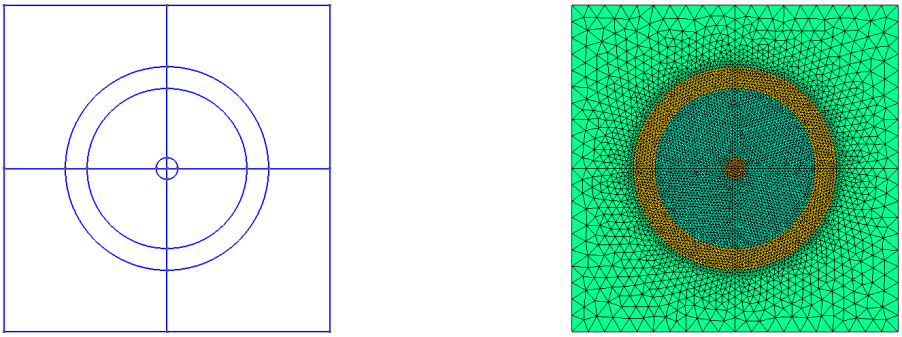


Fig. 13: A three-layered concentric structure with background using shape `onion3` (template `onion3`).

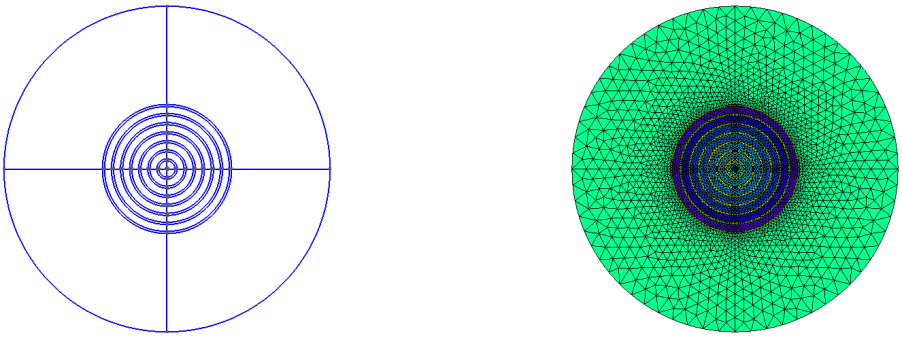


Fig. 14: A many-layered concentric structure with a circular outer boundary using shape `circ_onion` (template `circ_onion`).

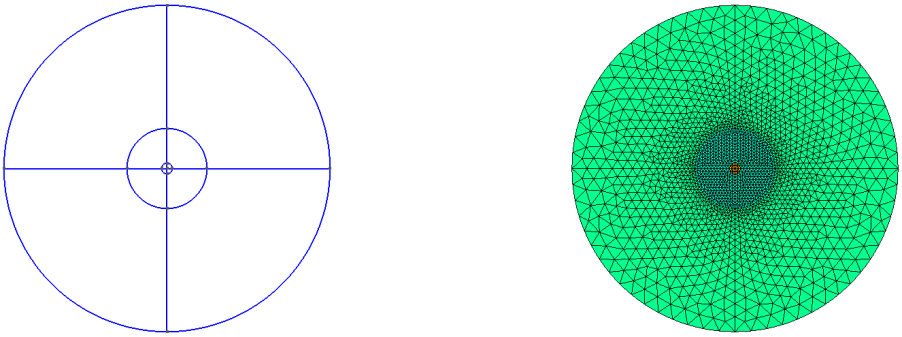


Fig. 15: A two-layered concentric structure with a circular outer boundary using shape `circ_onion2` (template `circ_onion2`).

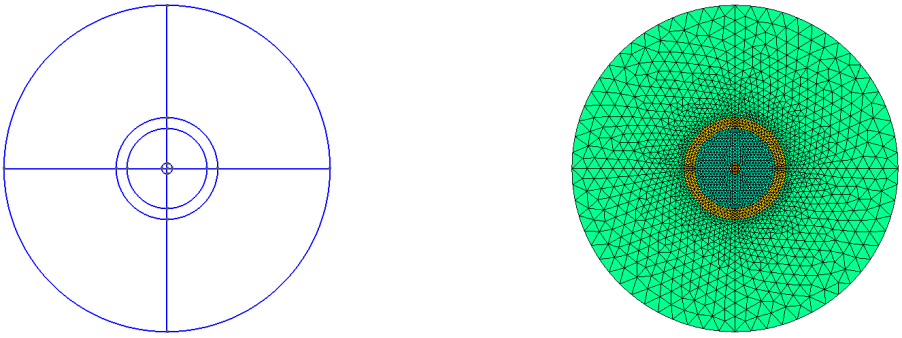


Fig. 16: A three-layered concentric structure with a circular outer boundary using shape `circ_onion3` (template `circ_onion3`).

Fig. 17: A trapezoidal rib structure using shape `trapezoidal_rib`.

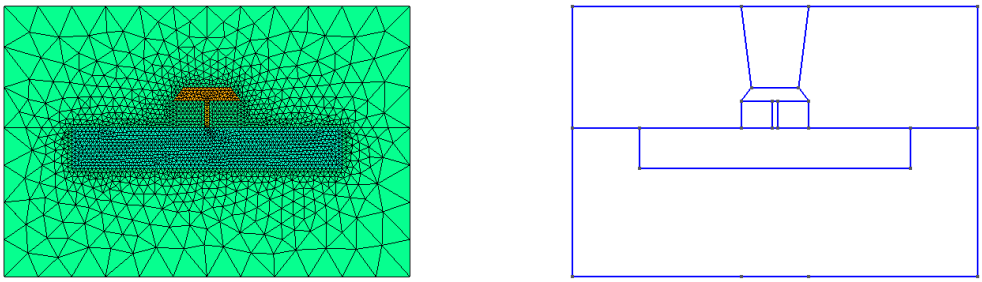


Fig. 18: A supported pedestal structure using shape `pedestal`.

3.6 Structure parameters

The parameters `lc_bkg`, `lc_refine_1`, `lc_refine_2` labelled in the above figures control the fineness of the FEM mesh and are set when constructing the waveguide, as discussed in the next chapter. The first parameter `lc_bkg` sets the reference background mesh size, typically as a fraction of the length of the outer boundary edge. A larger `lc_bkg` yields a coarser mesh. Reasonable starting values are `lc_bkg=0.1` (10 mesh points on the outer boundary) to `lc_bkg=0.05` (20 mesh points on the outer boundary).

As well as setting the overall mesh scale with `lc_bkg`, one can also refine the mesh near interfaces and near select points in the domain, as may be observed in the figures in the previous section. This helps to increase the mesh resolution in regions where there the electromagnetic and acoustic fields are likely to be strong and/or rapidly varying. This is achieved using the `lc_refine_n` parameters as follows. At the interface between materials, the mesh is refined to have characteristic length `lc_bkg/lc_refine_1`, therefore a *larger* `lc_refine_1` gives a *finer* mesh by a factor of `lc_refine_1` at these interfaces. The meshing program `Gmsh` 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 in the vicinity 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 more complicated structures, there are additional `lc_refine_<n>` parameters. To see their exact function, look for these expressions in 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 appropriate values depend strongly on the type of structure being studied, and so we strongly recommended carrying out a convergence test before delving into new structures (see Tutorial 5 for an example) starting from similar parameters as used in the tutorial simulations.

As will as giving low accuracy, a structure with too coarse a mesh is often the cause of the eigensolver failing to converge in which case NumBAT will terminate with an error. If you encounter such an error, try the calculation again with a slightly smaller value for `lc_bkg`, or slightly higher values for the `lc_refine_n` parameters.

On the other hand, it is wise to begin with relatively coarse meshes. It will be apparent that the number of elements scales roughly *quadratically* with the `lc_refine` parameters and so the run-time increases rapidly as the mesh becomes finer. For each problem, some initial experimentation to identify a mesh resolution that gives reasonable convergence in acceptable simulation is usually worthwhile.

3.7 Viewing the mesh

When NumBAT constructs a waveguide, the template `geo` file is converted to a concrete instatiation with the `lc_refine` and geometric parameters adjusted to the requested values. This file is then converted into a `gmsh .msh` file. When exploring new structures and their convergence behaviour, it is a very good idea to view the generated mesh frequently.

You can examine the resolution of your mesh by calling the `plot_mesh(<prefix>)` or `check_mesh()` methods on a waveguide `Structure` object. The first of these functions saves a pair of images of the mesh to a `<prefix>-mesh.png` file in the local directory which can be viewed with your preferred image viewer; the second opens the mesh in a `gmsh` window (see Tutorial 1 above).

In addition, the `.msh` file generated by NumBAT in any calculation is stored in `<NumBAT>/backend/fortran/msh/build` and can be viewed by running the command

```
gmsh <msh_filename>.msh
```

In some error situations, NumBAT will explicitly suggest viewing the mesh and will print out the required command to do so.

CHAPTER

FOUR

TUTORIAL

4.1 Introduction

This chapter provides a sequence of graded tutorials for learning NumBAT, exploring its applications and validating it against literature results and analytic solutions where possible. Before attempting your own calculations with NumBAT, we strongly advise working through the sequence of tutorial exercises which are largely based on literature results.

You may then choose to explore relevant examples drawn from a recent tutorial paper by Dr Mike Smith and colleagues, and a range of other literature studies, which are provided in the following two chapters, *JOSA-B Tutorial Paper* and *Literature Examples*.

4.2 Some Key Symbols

As far as practical we use consisent notation and symbols in the tutorial files. The following list introduces a few commonly encountered ones. Note that with the exception of the free-space wavelength λ and the spatial dimensions of waveguide structures, which are both specified in nanometres (nm), all quantities in NumBAT should be expressed in the standard SI units. For example, elastic freqencies ν are expressed in Hz, not GHz.

`lambda_nm`
: This is the *free-space* optical wavelength λ satisfying $\lambda = 2\pi c/\omega$, where c is the speed of light and ω is the angular frequency. **For convenience, this parameter is specified in nm.**

  For most examples, we use the conventional value λ =1550 nm.

`omega, omega_EM, om_EM`
: This is the electromagnetic *angular* frequency $\omega = 2\pi c/\lambda$ specified in $\mathrm{rad.s}^{-1}$.

`k, beta, k_EM`
: This is the electromagnetic *wavenumber* or *propagation constant* k or β, specified in m^{-1}.

`neff, n_eff`
: This is the electromagnetic modal *effective index* $\bar{n} = ck/\omega$, which is dimensionless.

`nu, nu_AC`
: This is the acoustic frequency ν specified in Hz.

`Omega, Omega_AC, Om_AC`
: This is the acoustic *angular* frequency $\Omega = 2\pi\nu$ specified in $\mathrm{rad.s}^{-1}$.

`q, q_AC`
: This is the acoustic *wavenumber* or *propagation constant* $q = v_{ac}\Omega$, where v_{ac} is the phase speed of the wave. The acoustic wavenumber is specified in m^{-1}.

`m`
: This is an integer corresponding to the mode number m of an electromagnetic mode $\vec{E}_m(\vec{r})$ or an acoustic mode $\vec{u}_m(\vec{r})$.

  For both electromagnetic and acoustic modes, counting of modes begins with `m=0`.

For the electromagnetic problem in which frequency/free-space wavelength is the independent variable, the $m = 0$ mode has the *highest* effective index $\bar{n}$ and *highest* wavenumber k of any mode for a given angular frequency ω.

For the acoustic problem, the wavenumber q is the independent variable and we solve for frequency $\nu = \Omega/(2\pi)$. The $m = 0$ mode has the *lowest* frequency ν of any mode for a given wavenumber q.

`inc_a_x, inc_a_y, inc_b_x, inc_b_y, slab_a_x, slab_a_y, ... etc`
These are dimensional parameters specifying the lengths of different aspects of a given structure: rib height, fibre radius etc. **For convenience, these parameters are specified in nm.**

4.3 Tutorials

We now walk through a number of simple simulations that demonstrate the basic use of NumBAT located in the <NumBAT>/tutorials directory.

We will meet a significant number of NumBAT functions in these tutorials, though certainly not all. The full Python interface is documented in the section chap-pythonbackend-label.

4.3.1 Tutorial 2 – SBS Gain Spectra

The first example we met in the previous chapter only printed numerical data to the screen with no graphical output. This example, contained in <NUMBAT>/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.

As before, move to the <NUMBAT>/tutorials directory, and then run the calculation by entering:

```
$ python3 simo-tut_02-gain_spectra-npsave.py
```

Or you can take advantage of the Makefile provided in the directory and just type:

```
$ make tut02
```

Some of the tutorial problems can take a little while to run, especially if your computer is not especially fast. To save time, you can run most problems with a coarser mesh at the cost of somewhat reduced accuracy, by adding the flag fast=1 to the command line:

```
$ python3 simo-tut_02-gain_spectra-npsave.py fast=1
```

Or using the makefile technique, simply

```
$ make ftut02
```

The calculation should complete in a minute or so. You will find a number of new files in the tutorials directory beginning with the prefix tut_02 (or ftut_02 if you ran in fast mode).

Gain Spectra

The Brillouin gain spectra and plotted using the functions integration.gain_and_qs() and plotting.plot_gain_spectra(). The results are contained in the file tut_02-gain_spectra.png which can be viewed in any image viewer. On Linux, for instance you can use

```
$ eog tut_02_gain_spectra.png
```

to see this image:

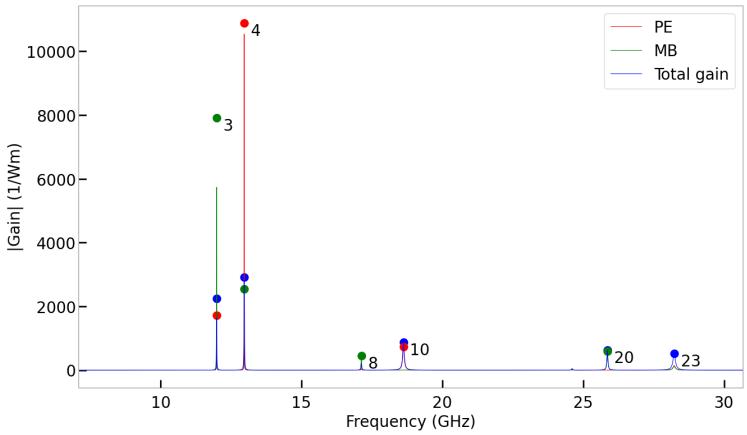


Fig. 1: Gain spectrum in `tut_02-gain_spectra.png` showing gain due to the photoelastic effect, gain due to moving boundary effect, and the total gain. The numbers near the main peaks identify the acoustic mode associated with the resonance.

Note how the different contributions from the photoelastic and moving-boundary effects are visible. In some cases, the total gain (blue) may be less than one or both of the separate effects if the two components act with opposite sign. *JOSA-B Tutorial Paper* and *Literature Examples*. (See Literature example 1 in the chapter *Literature Examples* for an interesting cexample of this phenomenon.)

Note also that prominent resonance peaks in the gain spectrum are labelled with the mode number m of the associated acoustic mode. This makes it easy to find the spatial profile of the most relevant modes (see below).

Mode Profiles

The choice of parameters for `plot_gain_spectra()` has caused several other files to be generated showing a zoomed-in version near the main peak, and the whole spectrum on log and dB scales:

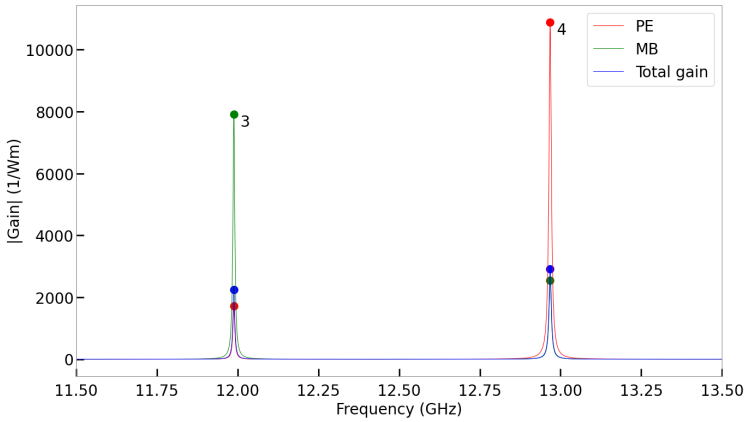


Fig. 2: Zoom-in of the gain spectrum in the previous figure in the file `tut_02-gain_spectra_zoom.png` .

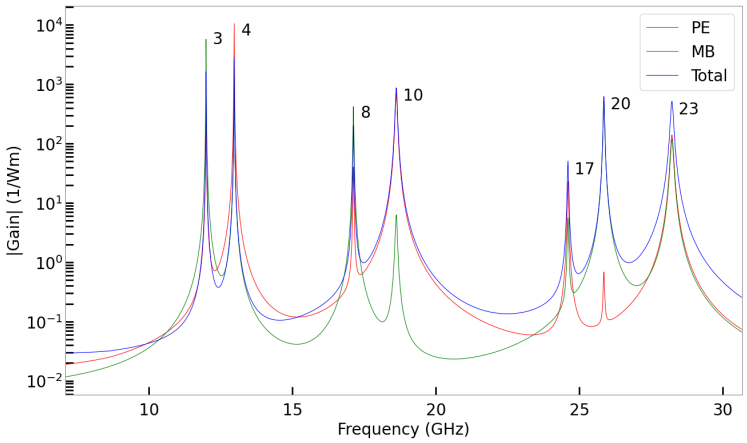


Fig. 3: Gain spectrum viewed on a log scale in the field `tut_02-gain_spectra-logy.png` .

This example has also generated plots of some of the electromagnetic and acoustic modes that were found in solving the eigenproblems. These are created using the calls to `plotting.plot_mode_fields()` and stored in the sub-directory `tut_02-fields`.

Note that a number of useful parameters are also displayed at the top-left of each mode profile. These parameters can also be extracted using a range of function calls on a `Mode` object (see the API docs).

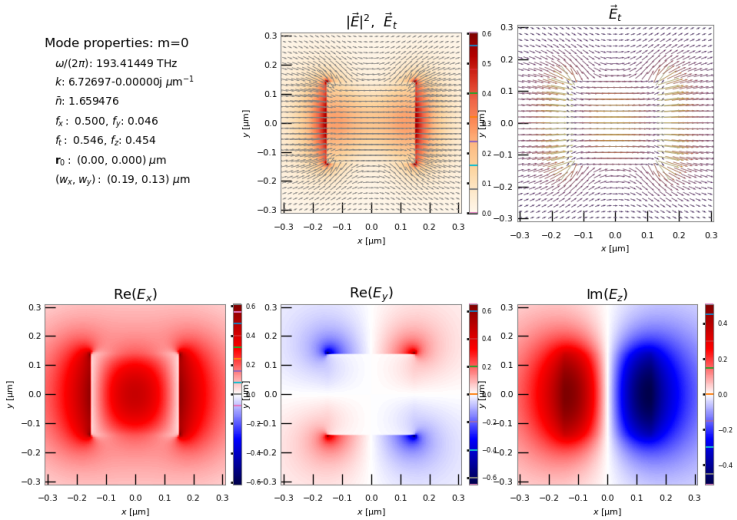


Fig. 4: Electric field profile of the fundamental ($m = 0$) optical mode profile stored in `tut_02-fields/EM_E_field_00.png`. The figures shows the modulus of the whole electric field $|\vec{E}|^2$, a vector plot of the transverse field $\vec{E}_t = (E_x, E_y)$, and the three components of the electric field. NumBAT chooses the phase of the mode profile such that the transverse components are real. Note that the E_z component is $\pi/2$ out of phase with the transverse components. (Since the structure is lossless, the imaginary parts of the transverse field, and the real part of E_z are zero).

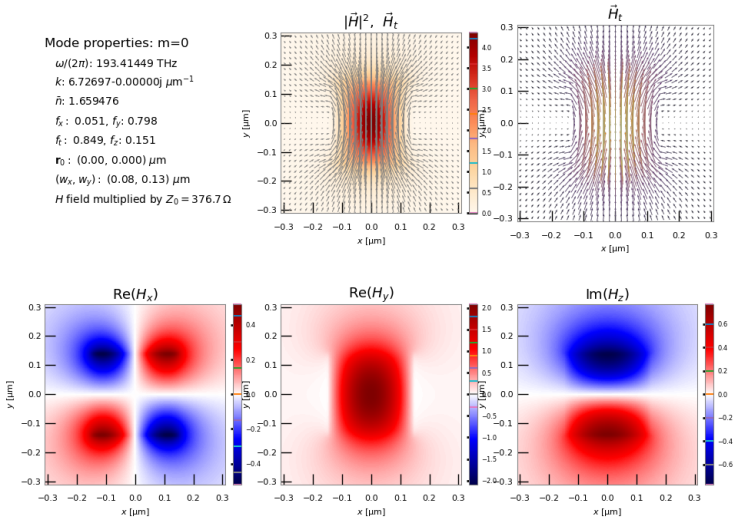


Fig. 5: Magnetic field profile of the fundamental ($m = 0$) optical mode profile showing modulus of the whole magnetic field $|\vec{H}|^2$, vector plot of the transverse field $\vec{H}_t = (H_x, H_y)$, and the three components of the magnetic field. Note that the H_z component is $\pi/2$ out of phase with the transverse components.

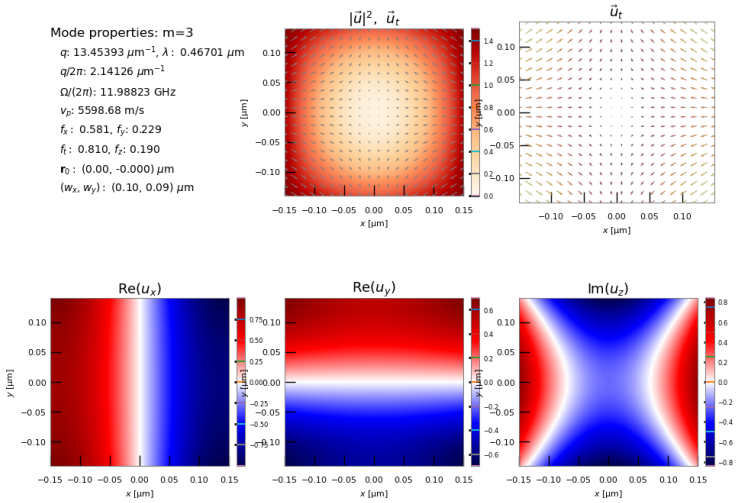


Fig. 6: Displacement field $\vec{u}(\vec{r})$ of the $m = 3$ acoustic mode with gain dominated by the moving boundary effect (green curve in gain spectra). Note that the frequency of $\Omega/(2\pi) = 11.99$ GHz (listed in the upper-left corner) corresponds to the first peak in the gain spectrum.

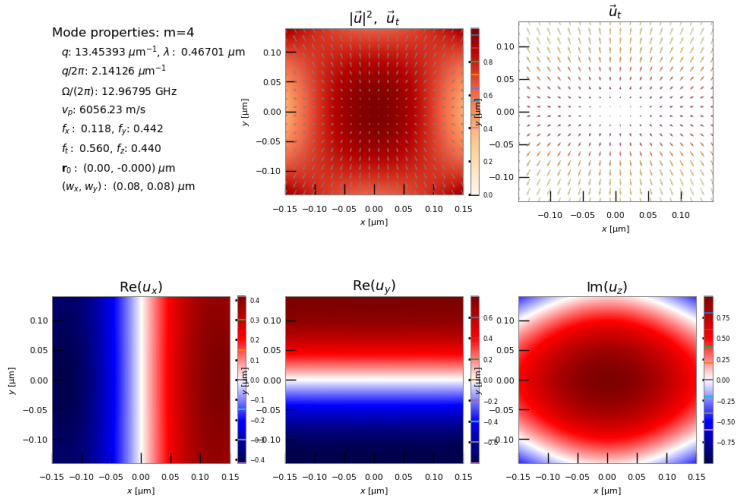


Fig. 7: Displacement field $\vec{u}(\vec{r})$ of the $m = 4$ acoustic mode with gain dominated by the photo-elastic effect (red curve in gain spectra). Note that the frequency of $\Omega/(2\pi) = 13.45$ GHz corresponds to the second peak in the gain spectrum.

Miscellaneous comments

Here are some further elements to note about this example:

1. When using the `fast=` mode, the output data and fields directory begin with `ftut_02` rather than `tut_02`.
2. It is frequently useful to be able to save and load the results of simulations to adjust plots without having to repeat the entire calculation. Here the flag `recalc_fields` determines whether the calculation should be done afresh and use previously saved data. This is performed using the `save_simulation()` and `load_simulation()` calls.
3. Plots of the modal field profiles are obtained using `plotting.plot_mode_fields`. Both electric and magnetic fields can be selected using `EM_E` or `EM_H` as the value of the `EM_AC` argument. The mode numbers to be plotted is specified by `ivals`. These fields are stored in a folder `tut_02-fields/` within the tutorial folder. Later we will see how an alternative approach in which we extract a `Mode` object from a `Simulation` which represent a single mode that is able to plot itself. This can be more convenient.
4. The overall amplitude of the modal fields is arbitrary. In NumBAT, the maximum value of the electric field is set to be 1.0, and this may be interpreted as a quantity in units of V/m, $\sqrt{\text{W}}$ or other units as desired. Importantly, the plotted *magnetic* field $\vec{H}(\vec{r})$ is multiplied by the impedance of free space $Z_0 = \sqrt{\mu_0/\epsilon_0}$ so that $Z_0\vec{H}(\vec{r})$ and $\vec{E}(\vec{r})$ *have the same units*, and the relative amplitudes between the electric and magnetic field plots are meaningful.
5. The `suppress_imimre` option suppresses plotting of the $\text{Im}[x]$, $\text{Im}[y]$ and $\text{Re}[z]$ components of the fields which in a lossless non-leaky problem should normally be zero at all points and therefore not useful to plot.
6. By default, plots are exported as `png` format. Pass the option `pdf_png=pdf` to plot functions to generate a `pdf` output.
7. 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.
8. 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.
9. The plot functions and the `Decorator` class support many options. Consult the API chapter for details on how to fine tune your plots.

The full code for this simulation is as follows:

```
print("\nSimulation time: {0:10.3f} secs.\n\n".format(end - start))
"""
    NumBAT Tutorial 2

    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.
lambda_nm = 1550
unitcell_x = 2*lambda_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 = 'ftut_02'
  refine_fac=1
else:
  prefix = 'tut_02'
  refine_fac=5

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

# Use of a more refined mesh to produce field plots.
wguide = objects.Structure(unitcell_x,inc_a_x,unitcell_y,inc_a_y,inc_shape,
```

(continues on next page)

(continued from previous page)

```
                        material_bkg=materials.make_material("Vacuum"),
                        material_a=materials.make_material("Si_2016_Smith"),
                        lc_bkg=.1, lc_refine_1=5.0*refine_fac, lc_refine_2=5.0*refine_
↪fac)


#wguide.check_mesh()

# Estimate expected effective index of fundamental guided mode.
n_eff = wguide.get_material('a').refindex_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, lambda_nm, n_eff)
  sim_EM_Stokes = mode_calcs.bkwd_Stokes_modes(sim_EM_pump)

  print('\nSaving EM fields')
  sim_EM_pump.save_simulation('tut02_wguide_data')
  sim_EM_Stokes.save_simulation('tut02_wguide_data2')
else:
  # Once npz files have been saved from one simulation run,
  # set recalc_fields=True to use the saved data
  sim_EM_pump = mode_calcs.load_simulation('tot02_wguide_data')
  sim_EM_Stokes = mode_calcs.load_simulation('tot02_wguide_data2')

# 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.

print('\nPlotting EM fields')
#Plot the E field of the pump mode
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=prefix, ticks=True)

#Plot the H field of the pump mode
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_H', prefix=prefix, ticks=True)

# Calculate the EM effective index of the waveguide.
```

(continues on next page)

(continued from previous page)

```
n_eff_sim = np.real(sim_EM_pump.neff(0))
print("n_eff", np.round(n_eff_sim, 4))

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

if recalc_fields:
  # Calculate and save acoustic modes.
  sim_AC = wguide.calc_AC_modes(num_modes_AC, q_AC, EM_sim=sim_EM_pump)

  print('Saving AC fields')
  sim_AC.save_simulation('tot02_wguide_data_AC')
else:
  sim_AC = mode_calcs.load_simulation('tot02_wguide_data_AC')

# 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:.5f}'.format(i, np.real(nu)*1e-9))

# 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.

print('\nPlotting acoustic modes')
plotting.plot_mode_fields(sim_AC, contours=True, prefix=prefix,
    ticks=True, quiver_points=20, ivals=range(10))

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, q_AC, EM_ival_pump=EM_ival_pump,
    EM_ival_Stokes=EM_ival_Stokes, AC_ival=AC_ival)
  # Save the gain calculation results
  np.savez('tut02_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, q_AC, 'comsol_ac_modes_1-5.dat',
    comsol_ivals=comsol_ivals)
```

(continues on next page)

(continued from previous page)

```
# Print the PE contribution to gain SBS gain of the AC modes.
print("\n Displaying results of first five modes 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.nu_AC_all()[0]) - 2e9  # Hz
freq_max = np.real(sim_AC.nu_AC_all()[-1]) + 2e9 # Hz
plotting.plot_gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix=prefix, dB=True, semilogy=True)

# Repeat this plot focusing on one frequency range
freq_min = 11.5e9  # Hz
freq_max = 13.5e9  # Hz
plotting.plot_gain_spectra(sim_AC, SBS_gain, SBS_gain_PE, SBS_gain_MB, linewidth_Hz,
    EM_ival_pump, EM_ival_Stokes, AC_ival, freq_min=freq_min, freq_max=freq_max,
    prefix=prefix, suffix='_zoom')

end = time.time()
print("\nSimulation time: {0:10.3f} secs.\n\n".format(end - start))
```

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

This example, contained in `tutorials/simo-tut_03_1-dispersion-npload.py` calculates the elastic dispersion diagram – the relation between the acoustic wave number q and frequency Ω– for the problem in the previous tutorial. This is done by scanning over the elastic wavenumber `q_AC` and finding the eigenfrequencies for each value.

As discussed in *Formal selection rules for Brillouin scattering in integrated waveguides and structured fibers* by C. Wolff, M. J. Steel, and C. G. Poulton DOI:/10.1364/OE.22.032489, the elastic modes of any waveguide may be classified according to their representation of the point group symmetry class corresponding to the waveguide profile. For this problem, the waveguide is rectangular with symmetry group C_{2v} which has four symmetry classes, which are marked in the dispersion diagram.

This example also takes advantage of the ability to load and save simulation results to save repeated calculation. using the `save_simulation` and `load_simulation` methods defined in the `mode_calcs` module. The previous tutorial saved its electromagnetic results in the file `tut02_wguide_data.npz` using the `Simulation.save_simulation()` method, while these tutorial has recovered those results using `mode_calc.load_simulation()`. This can be a very useful technique when trying to adjust the appearance of plots without having to repeat the whole calculation effort.

Note: from now on, we do not include the code for each tutorial and refer the reader to the relevant files in the `<NumBAT>/tutorials` directory.

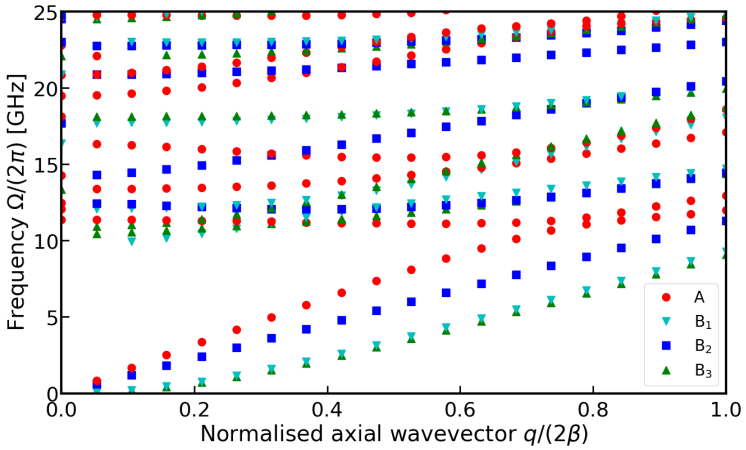


Fig. 8: Acoustic dispersion diagram with modes categorised by symmetry as in Table 1 of Wolff et al. *Opt. Express.* **22**, 32489 (2014).

4.3.3 Tutorial 3b – Investigating Dispersion and Multiprocessing

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

In this code as in the previous example, the acoustic modal problem is repeatedly solved at a range of different q values to build up a set of dispersion curves $\nu_m(q)$. Due to the large number of avoided and non-avoided crossings, it is usually best to plot dispersion curves like this with dots rather than joined lines. The plot generated below can be improved by increasing the number of q points sampled through the value of the variable `n_qs`, limited only by your patience.

The multiprocessing library runs each task as a completely separate process on the computer. Depending on the nature and number of your CPU, this may improve the performance considerably. This can also be easily extended to multiple node systems which will certainly improve performance. A very similar procedure using the `threading` library allows the different tasks to run as separate threads within the one process. However, due to the existence of the Python Global Interpreter Lock (GIL) which constrains what kinds of operations may run in parallel within Python , multiple threads will typically not improve the performance of NumBAT.

This tutorial also shows an example of saving data, in this case the array of acoustic wavenumbers and frequencies, to a text file using the `numpy` routine `np.savetxt` for later analysis.

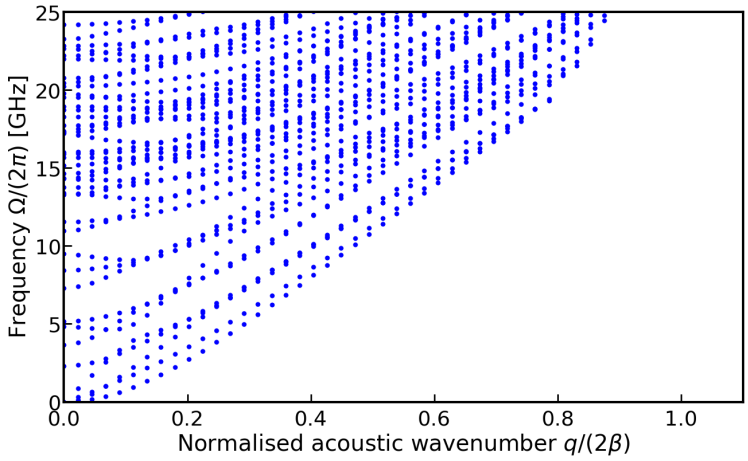


Fig. 9: Acoustic dispersion diagram. The elastic wave number q is scaled by the phase-matched SBS wavenumber 2β where β is the propagation constant of the optical pump mode.

4.3.4 Tutorial 4 – Parameter Scan of Widths

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

The results are displayed in a 3D plot. This may not be the most effective approach for this small data set but gives a sense of what is possible graphically. For a more effective plot, you might like to try the same calculation with around 30 values for the width rather than just 6.

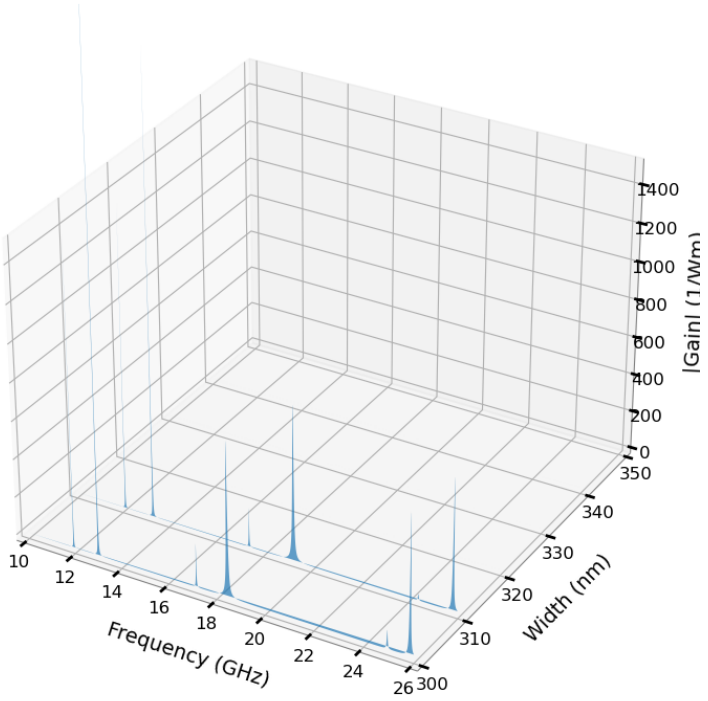


Fig. 10: Gain spectra as function of waveguide width.

4.3.5 Tutorial 5 – Convergence Study

This tutorial, contained in `simo-tut_05_convergence_study.py` demonstrates a scan of numerical parameters for our by now familiar silicon-in-air problem to test the convergence of the calculation results. This is done by scanning the value of the `lc_refine` parameters. The number of mesh elements (and simulation time) increases with roughly the square of the mesh refinement factor.

For the purpose of convergence estimates, the values calculated at the finest mesh (the rightmost values) are taken as the `exact` values, notated with the subscript 0, eg. β_0. The graphs below show both relative errors and absolute values for each quantity.

Once the convergence properties for a particular problem have been established, it can be useful to do exploratory work more quickly by adopting a somewhat coarser mesh, and then increase the resolution once again towards the end of the project to validate results before reporting them.

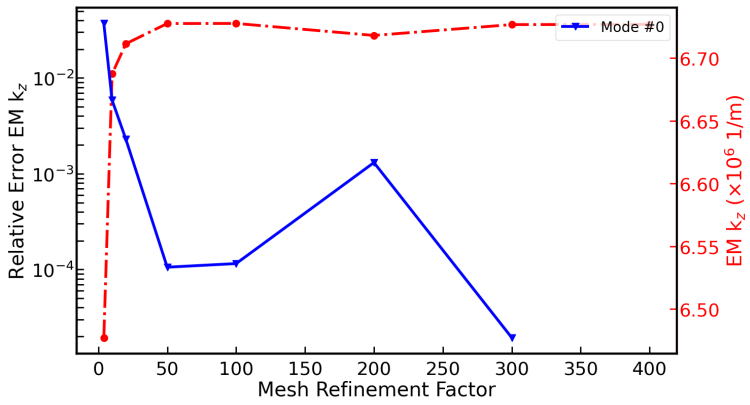


Fig. 11: Convergence of relative (blue) and absolute (red) optical wavenumbers $k_{z,i}$. The left axis displays the relative error $(k_{z,i} - k_{z,0})/k_{z,0}$. The right axis shows the absolute values of $k_{z,i}$.

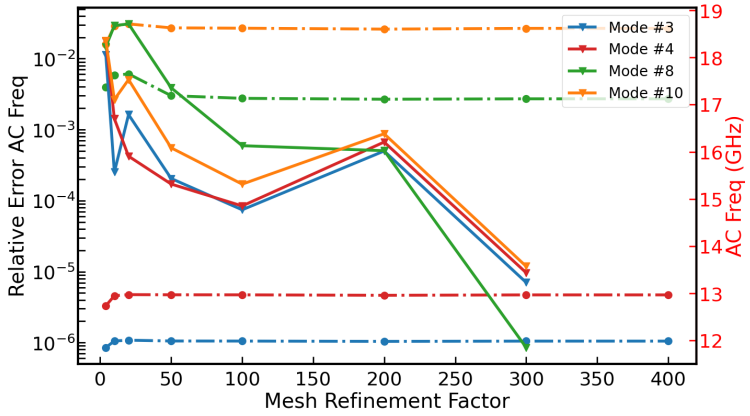


Fig. 12: Convergence of relative (solid, left) and absolute (chain, right) elastic mode frequencies ν_i.

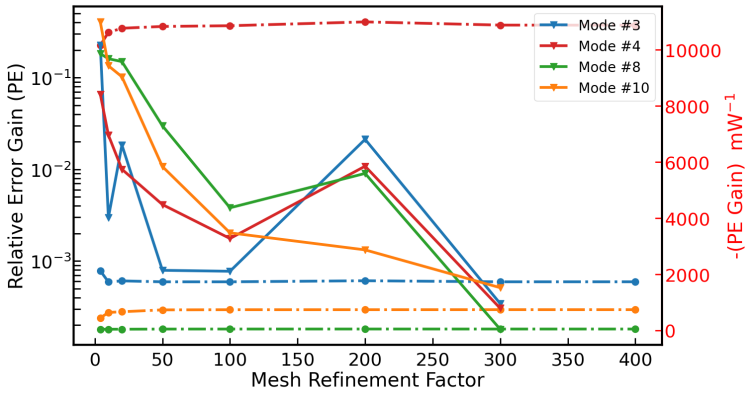


Fig. 13: Convergence of photoelastic gain G^{PE}. The absolute gain on the right hand side increases down the page because of the convention that NumBAT associates backward SBS with negative gain.

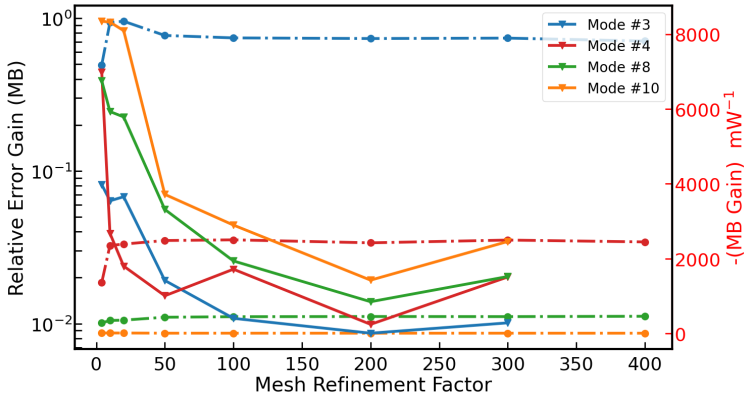


Fig. 14: Absolute and relative convergence of moving boundary gain G^{MB}.

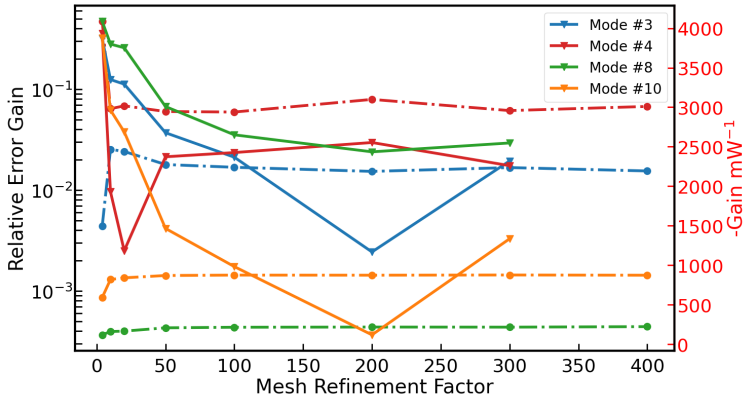


Fig. 15: Absolute and relative convergence of total gain G.

4.3.6 Tutorial 6 – Silica Nanowire

In this tutorial, contained in `simo-tut_06_silica_nanowire.py` we start to explore the Brillouin gain properties in a range of different structures, in this case a silica nanowire surrounded by vacuum.

The `gain-spectra` plot below shows the Brillouin gain as a function of Stokes shift. Each resonance peak is marked with the number of the acoustic mode associated with the resonance. This is very helpful in identifying which acoustic mode profiles to examine more closely. In this case, modes 5, 8 and 23 give the most signif-cant Brillouin gain. The number of modes labelled in the gain spectrum can be controlled using the parameter `mark_mode_thresh` in the function `plotting.plot_gain_spectra` to avoid many labels from modes giving negligible gain. Other parameters allow selecting only one type of gain (PE or MB), changing the frequency range, and plotting on log or dB scales.

It is important to remember that the total gain is not the simple sum of the photoelastic (PE) and moving boundary (MB) gains. Rather it is the coupling terms Q_{PE} and Q_{MB} which are added before squaring to give the total gain. Indeed the two effects may have opposite sign giving net gains smaller than either contribution.

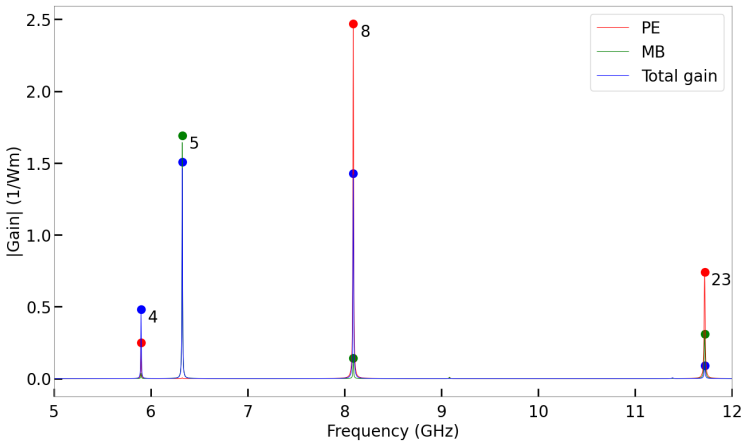


Fig. 16: Gain spectrum showing the gain due to the photoelastic effect (PE), the moving boundary effect (PB), and the net gain (Total).

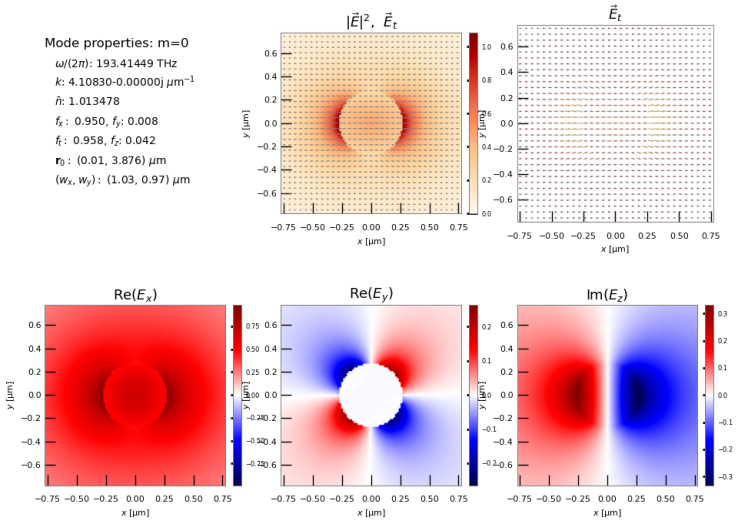


Fig. 17: Electromagnetic mode profile of the pump and Stokes field in the x-polarised fundamental mode of the waveguide.

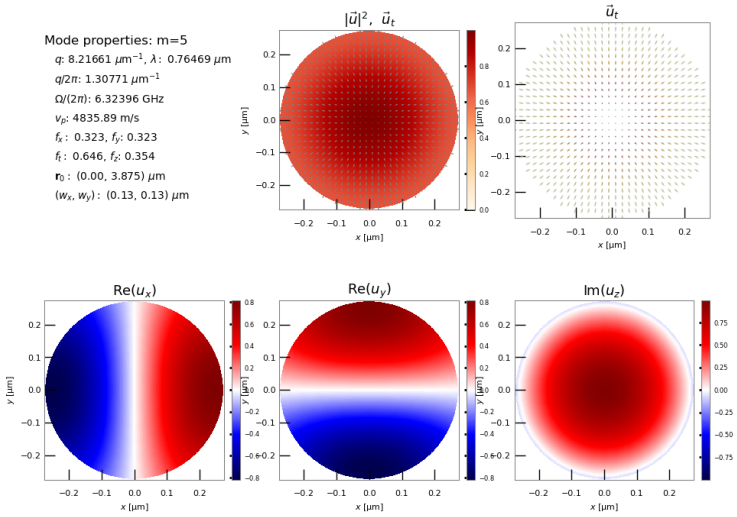


Fig. 18: Mode profiles for acoustic mode 5 which is visible as a MB-dominated peak in the gain spectrum.

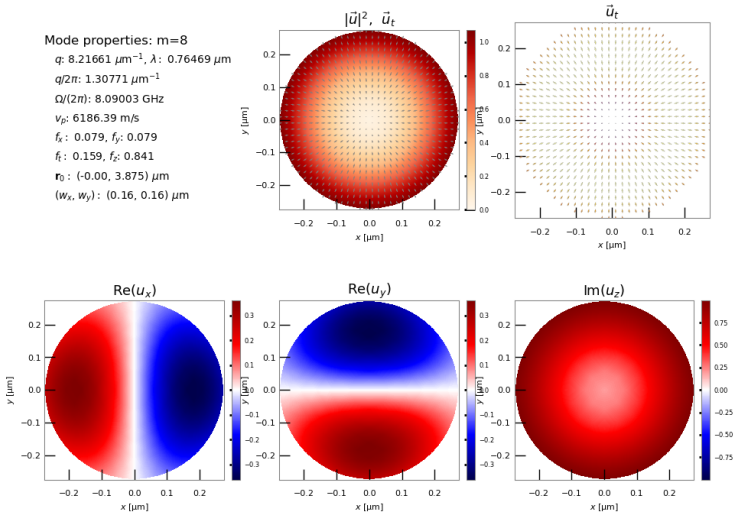


Fig. 19: Mode profiles for acoustic mode 8 which is visible as a PE-dominated peak in the gain spectrum.

4.3.7 Tutorial 7 – Slot Waveguide

This tutorial, contained in `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. This structure takes advantage of the slot effect which expels the optical field into the lower index medium, enhancing the fraction of the EM field inside the soft chalcogenide glass which guides the acoustic mode and increasing the gain.

Comparing the $m = 2$ and $m = 5$ acoustic mode profiles with the pump EM profile, it is apparent that the field overlap is favourable, where as the $m = 1$ mode gives zero gain due to its anti-symmetry relative to the pump field.

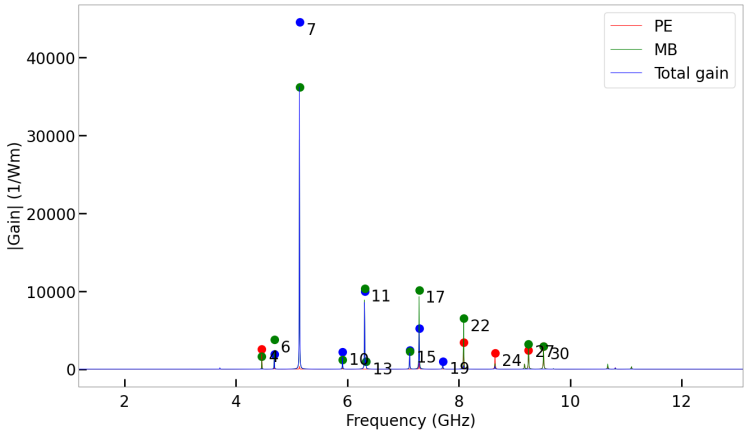


Fig. 20: Gain spectrum showing the gain due to the photoelastic effect (PE), the moving boundary effect (PB), and the net gain (Total).

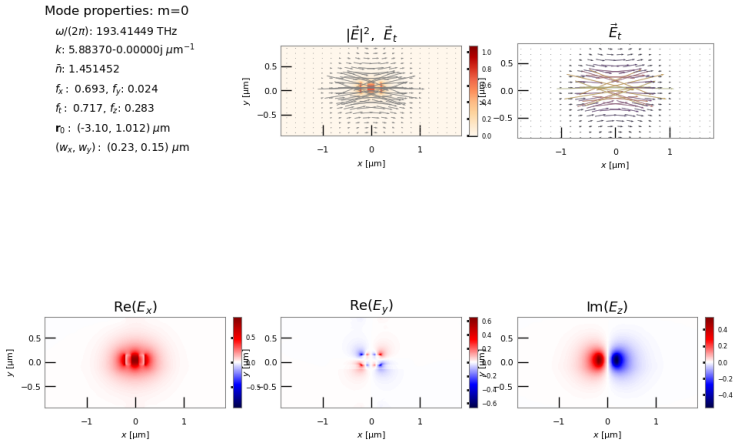


Fig. 21: Electromagnetic mode profile of the pump and Stokes field.

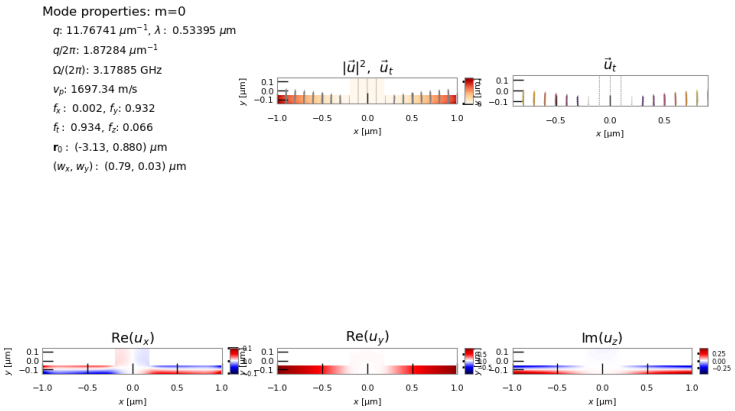


Fig. 22: Acoustic mode profiles for mode 0.

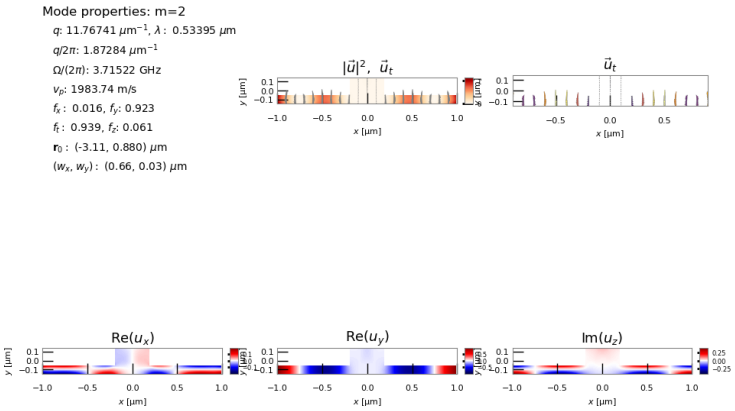


Fig. 23: Acoustic mode profiles for mode 2.

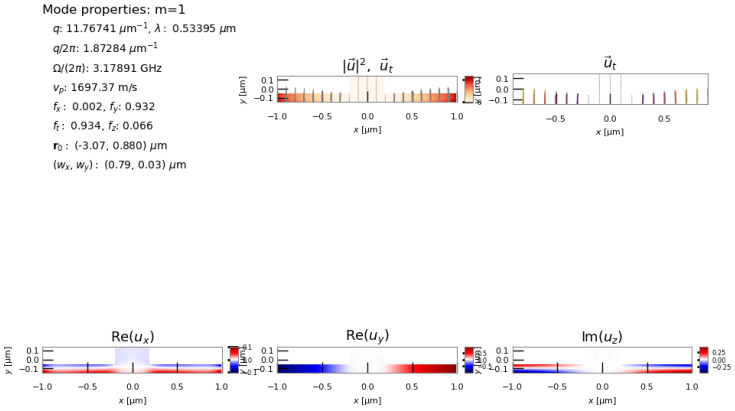


Fig. 24: Acoustic mode profiles for mode 1.

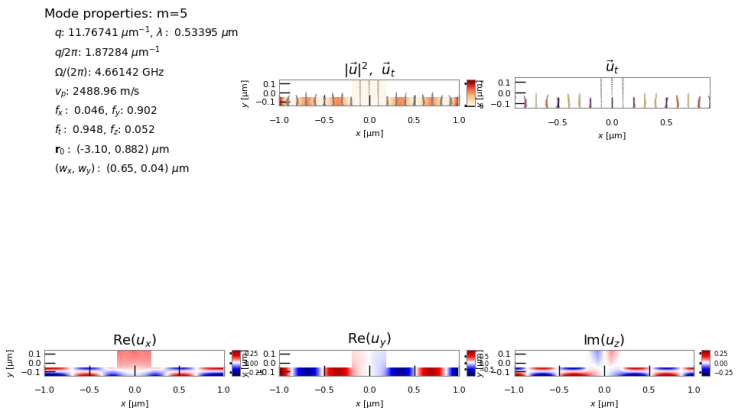


Fig. 25: Acoustic mode profiles for mode 5.

4.3.8 Tutorial 8 – Slot Waveguide Cover Width Scan

This tutorial, contained in `simo-tut_08-slot_coated-scan.py` continues the study of the previous slot waveguide, by examining the dependence of the acoustic spectrum on the thickness of a silica capping layer. As before, this parameter scan is accelerated by the use of multi-processing.

It is interesting to look at different mode profiles and try to understand why the eigenfrequency of some modes are more affected by the capping layer. The lowest mode, for instance, is noticeably unaffected.

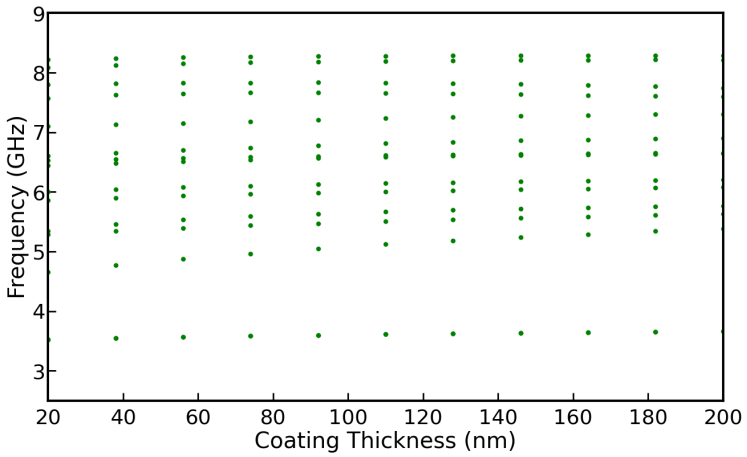


Fig. 26: Acoustic frequencies as function of covering layer thickness.

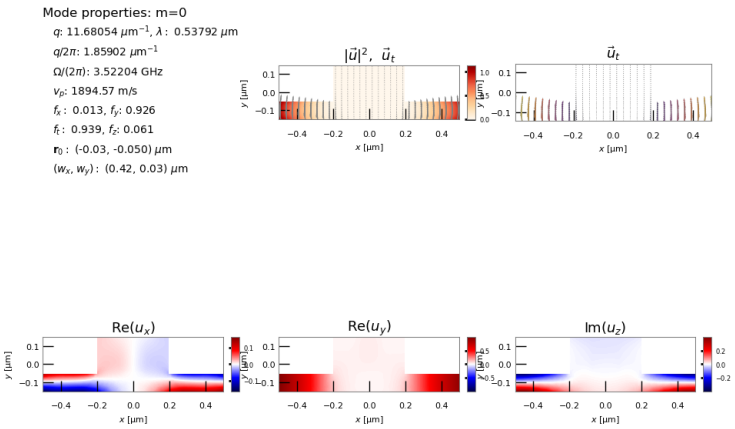


Fig. 27: Modal profiles of lowest acoustic mode.

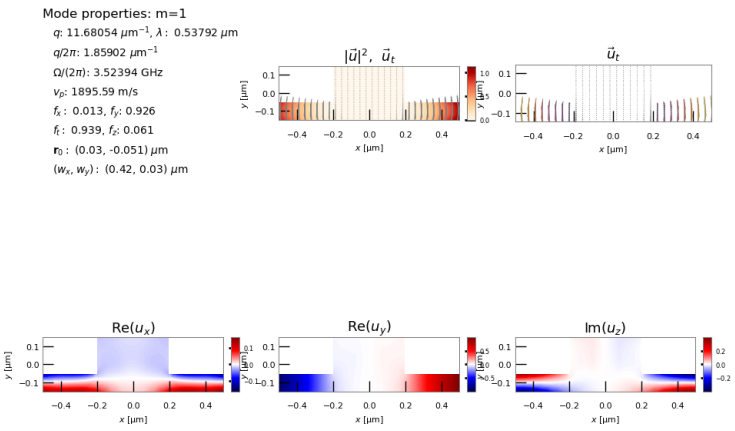


Fig. 28: Modal profiles of second acoustic mode.

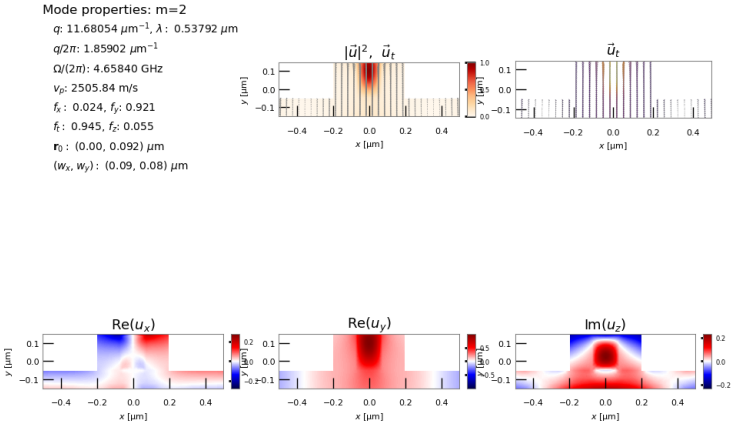


Fig. 29: Modal profiles of third acoustic mode.

4.3.9 Tutorial 9 – Anisotropic Elastic Materials

This tutorial, contained in `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).

The data below compares the gain spectrum compared to that found with the simpler isotropic stiffness model used in Tutorial 2. The results are very similar but the isotropic model shows two smaller peaks at high frequency.

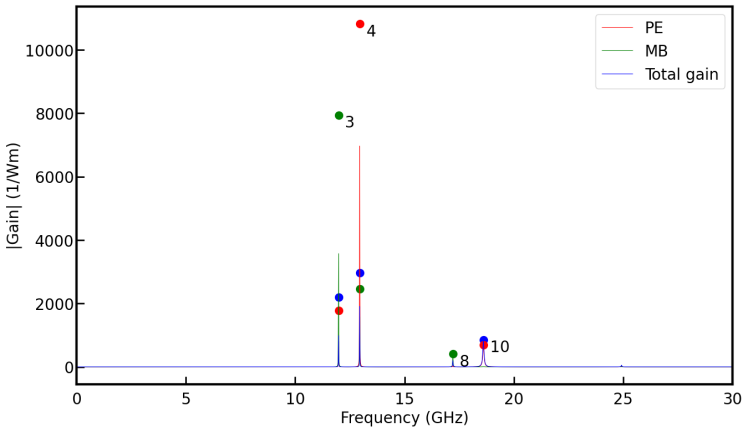


Fig. 30: Gain spectrum with anisotropic stiffness model of silicon.

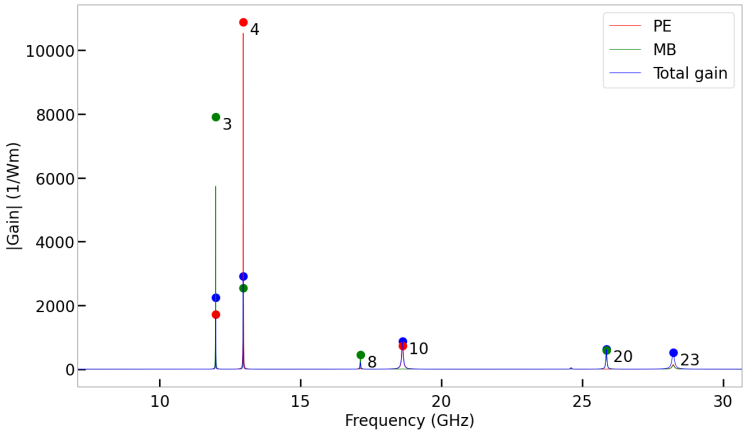


Fig. 31: Gain spectrum from Tutorial 2 with isotropic stiffness model of silicon.

4.3.10 Tutorial 11 – Two-layered 'Onion'

This tutorial, contained in `simo-tut_11a-onion2.py` demonstrates use of a two-layer `onion` structure for a backward intra-modal SBS configuration. Note that with the inclusion of the background layer, the two-layer onion effectively creates a three-layer geometry with core, cladding, and background surroundings. This is the ideal structure for investigating the cladding modes of an optical fibre. It can be seen by looking through the optical mode profiles in `tut_11a-fields/EM*.png` that this particular structure supports five cladding modes in addition to the three guided modes (the TM_0 mode is very close to cutoff).

Next, the gain spectrum and the mode profiles of the main peaks indicate as expected, that the gain is optimal for modes that are predominantly longitudinal in character.

The accompanying tutorial `simo-tut_11b-onion3.py` introduces one additional layer and would be suitable for studying the influence of the outer polymer coating of an optical fibre or depressed cladding W fibre.

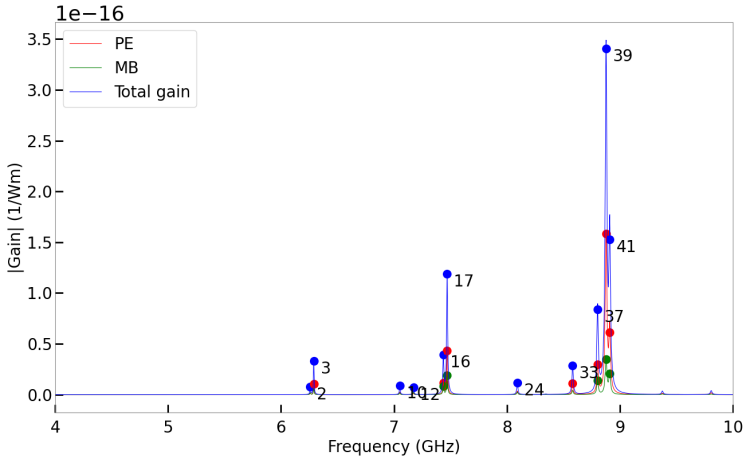


Fig. 32: Gain spectrum for the two-layer structure in `tut_11a`.

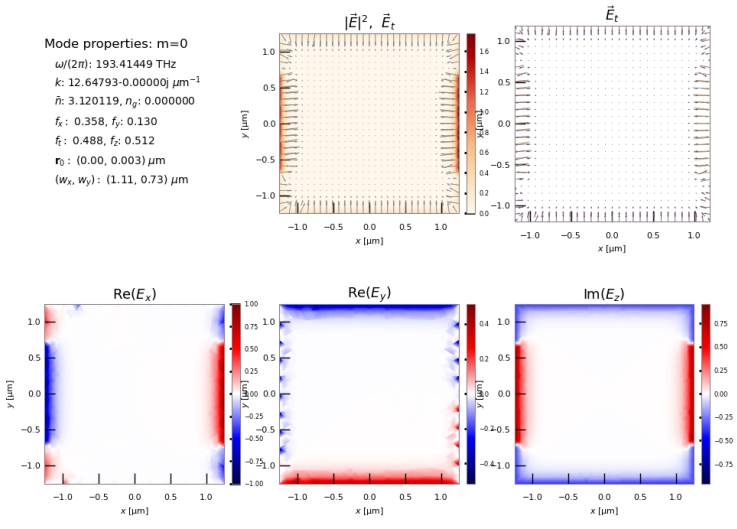


Fig. 33: Mode profile for fundamental optical mode.

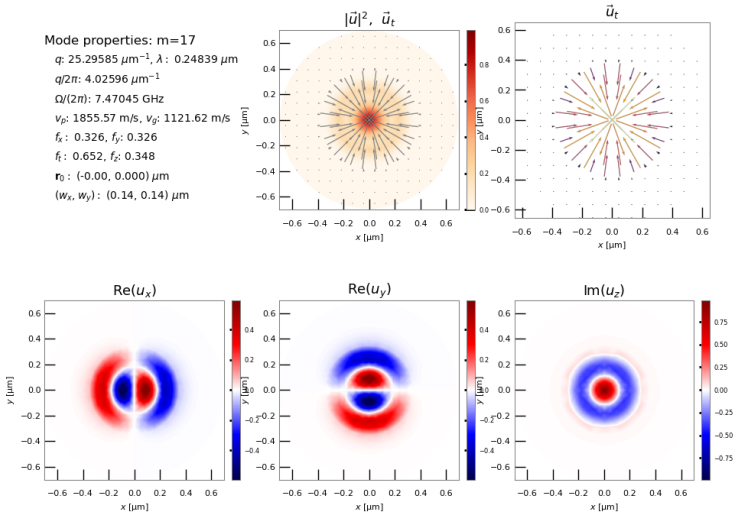


Fig. 34: Mode profile for acoustic mode 17.

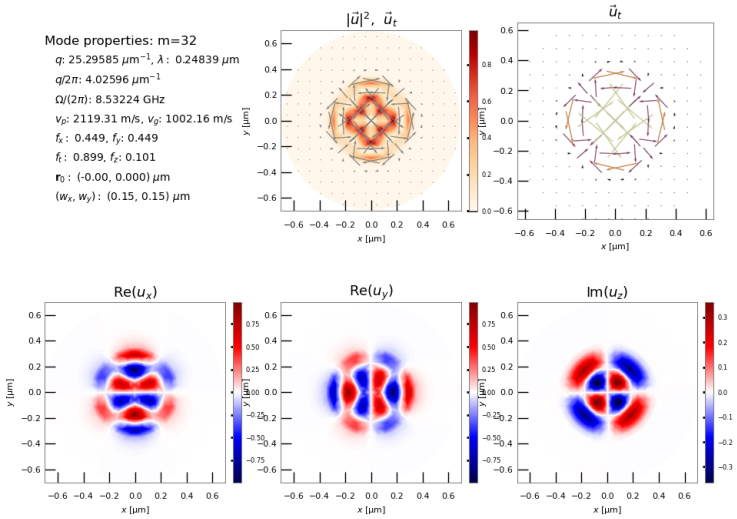


Fig. 35: Mode profile for acoustic mode 32.

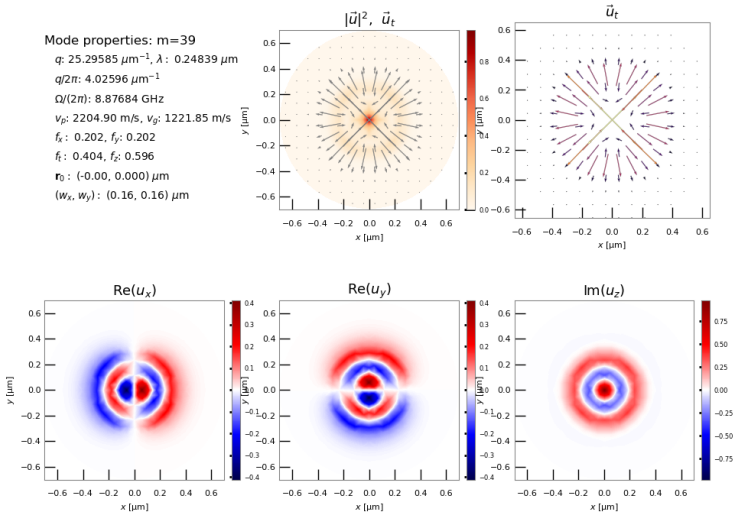


Fig. 36: Mode profile for acoustic mode 39.

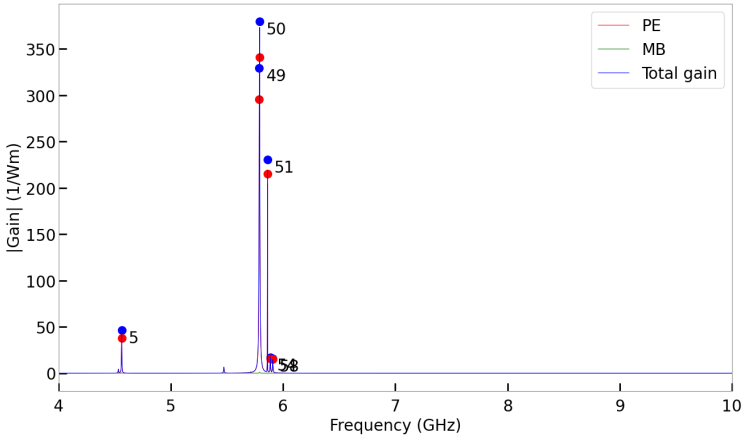


Fig. 37: Gain spectrum for the three-layer structure in `tut_11b`.

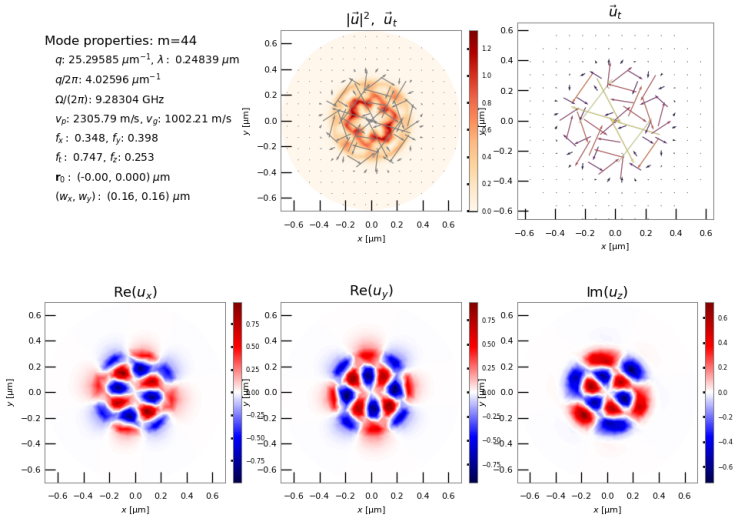


Fig. 38: Mode profile for acoustic mode 44.

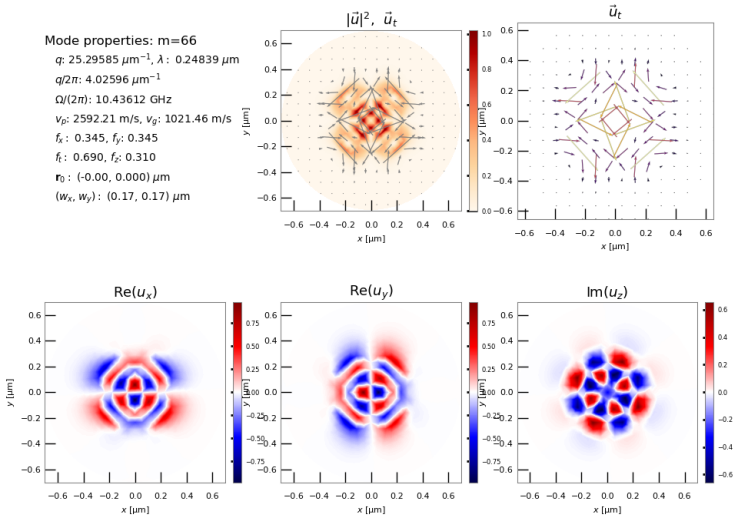


Fig. 39: Mode profile for acoustic mode 66.

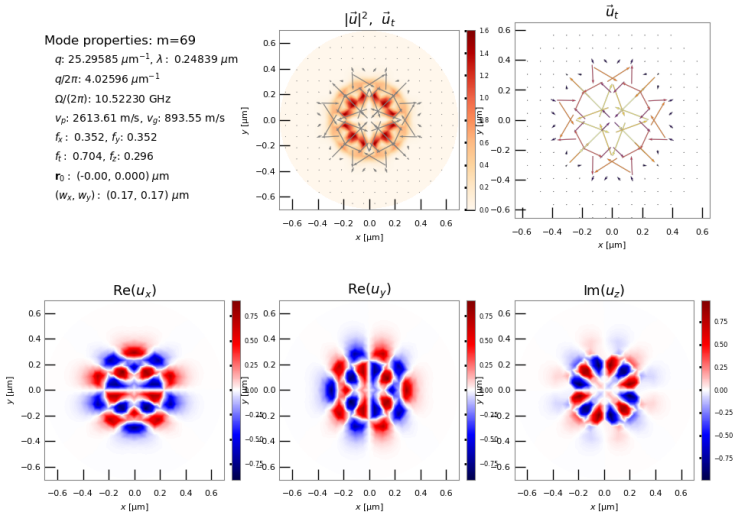


Fig. 40: Mode profile for acoustic mode 66.

4.3.11 Tutorial 12 – Valdiating the calculation of the EM dispersion of a two-layer fibre

How can we be confident that NumBAT's calculations are actually correct? This tutorial and the next one look at rigorously validating some of the modal calculations produced by NumBAT.

This tutorial, contained in `simo-tut_12.py`, compares analytic and numerical calculations for the dispersion relation of the electromagnetic modes of a cylindrical waveguide. This can be done in both a low-index contrast (SMF-28 fibre) and high-index contrast (silicon rod in silica) context. We calculate the effective index $\bar{n}$ and normalised waveguide parameter $b = (\bar{n}^2 - n_{\text{cl}}^2)/(n_{\text{co}}^2 - n_{\text{cl}}^2)$ as a function of the normalised freqency $V = ka\sqrt{n_{\text{co}}^2 - n_{\text{cl}}^2}$ for radius a and wavenumber $k = 2\pi/\lambda$. As in several previous examples, this is accomplished by a scan over the wavenumber k.

The numerical results (marked with crosses) are compared to the modes found from the roots of the rigorous analytic dispersion relation (solid lines). We also show the predictions for the group index $n_g = \bar{n} + V\frac{d\bar{n}}{dV}$. The only noticeable departures are right at the low V-number regime where the fields become very extended into the cladding and interact significantly with the boundary. The results could be improved in this regime by choosing a larger domain at the expense of a longer calculation.

As this example involves the same calculation at many values of the wavenumber k, we again use parallel processing techniques. However, in this case we demonstrate the use of *threads* (multiple simultaneous strands of execution within the same process) rather than a pool of separate processes. Threads are light-weight and can be started more efficiently than separate processes. However, as all threads share the same memory space, some care is needed to prevent two threads reading or writing to the same data structure simultaneously. This is dealt with using the helper functions and class in the *numbattools.py* module.

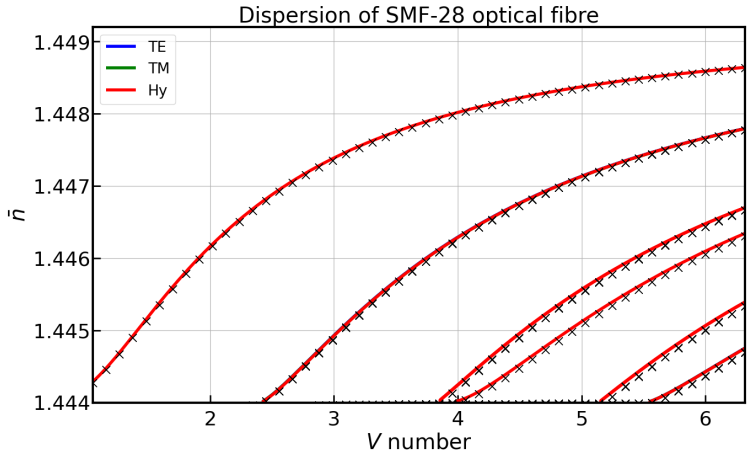


Fig. 41: Optical effective index as a function of normalised frequency for SMF-28 fibre.

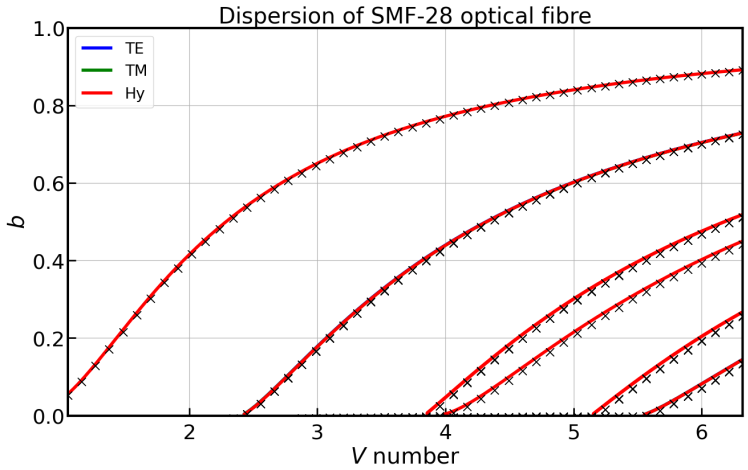


Fig. 42: Optical normalised waveguide parameter as a function of normalised frequency for SMF-28 fibre.

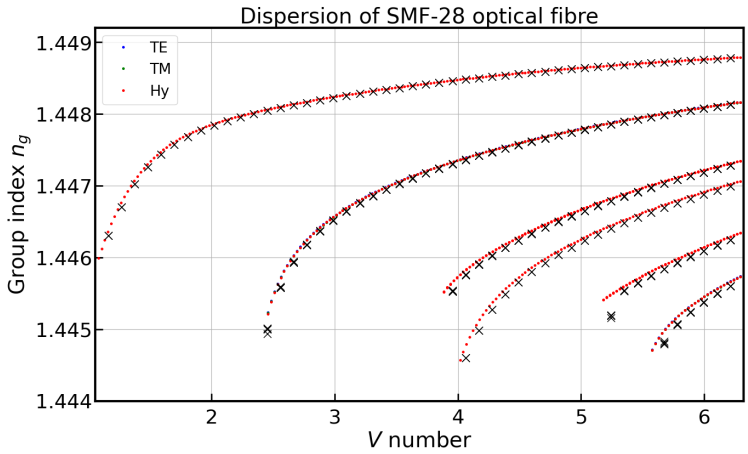


Fig. 43: Optical group index $n_g = \bar{n} + V\frac{d\bar{n}}{dV}$ as a function of normalised frequency for SMF-28 fibre.

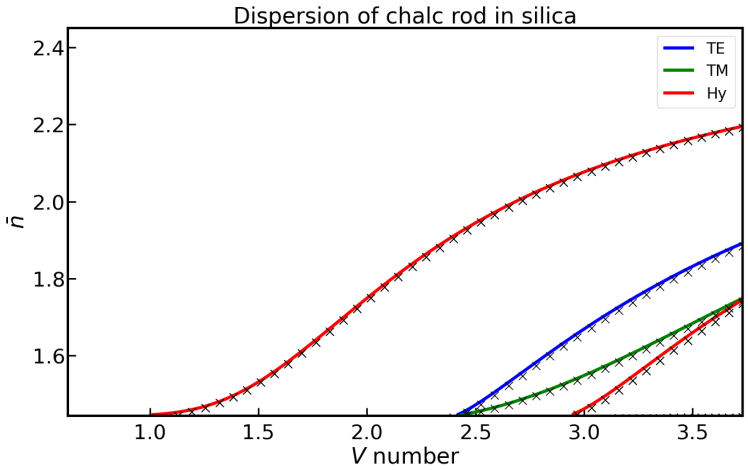


Fig. 44: Optical effective index as a function of normalised frequency for silicon rod in silica.

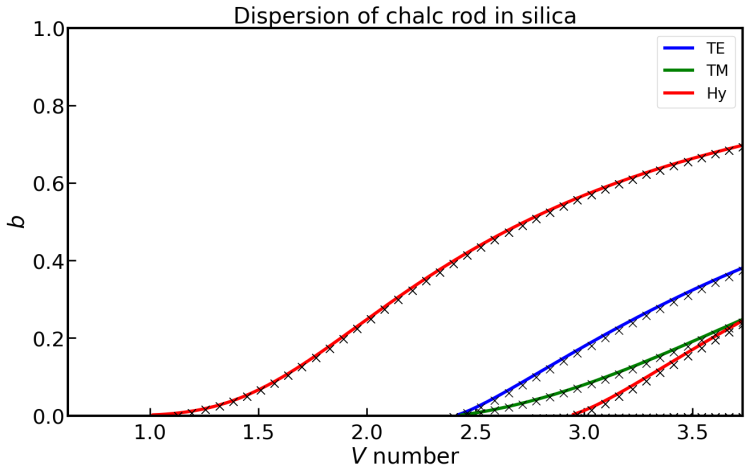


Fig. 45: Optical normalised waveguide parameter as a function of normalised frequency for silicon rod in silica.

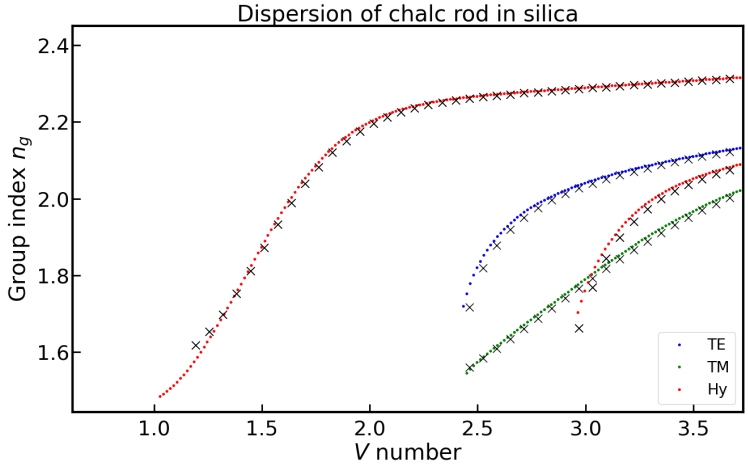


Fig. 46: Optical group index $n_g = \bar{n} + V\frac{d\bar{n}}{dV}$ as a function of normalised frequency for silicon rod in silica.

4.3.12 Tutorial 13 – Valdiating the calculation of the dispersion of an elastic rod in vacuum

The tutorial `simo-tut_13.py` performs the same kind of calculation as in the previous tutorial for the acoustic problem. In this case there is no simple analytic solution possible for the two-layer cylinder. Instead we create a structure of a single elastic rod surrounded by vacuum. NumBAT removes the vacuum region and imposes a free boundary condition at the boundary of the rod. The modes found are then compared to the analytic solution of a free homonegenous cylindrical rod in vacuum.

We find excellent agreement between the analytical (coloured lines) and numerical (crosses) results. Observe the existence of two classes of modes with azimuthal index $p = 0$, corresponding to the pure torsional modes, which for the lowest band propagate at the bulk shear velocity, and the so-called Pochammer hybrid modes, which are predominantly longitudinal, but must necessarily involve some shear motion to satisfy mass conservation.

It is instructive to examine the mode profiles in `tut_13-fields` and track the different field profiles and degeneracies found for each value of p. By basic group theory arguments, we know that every mode with $p \neq 0$ must come as a degenerate pair and this is satisfied to around 5 significant figures in the calculated results. It is interesting to repeat the calculation with a silicon (cubic symmetry) rod rather than chalcogenide (isotropic). In that case, the lower symmetry of silicon causes splitting of a number of modes, so that a larger number of modes are found to be singly degenerate, though invariably with a partner state at a nearby frequency.

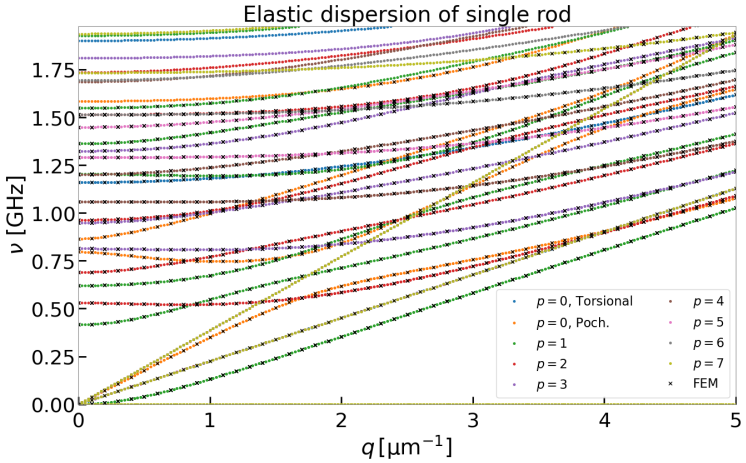


Fig. 47: Elastic frequency as a function of normalised wavenumber for a chalcogenide rod in vacuum.

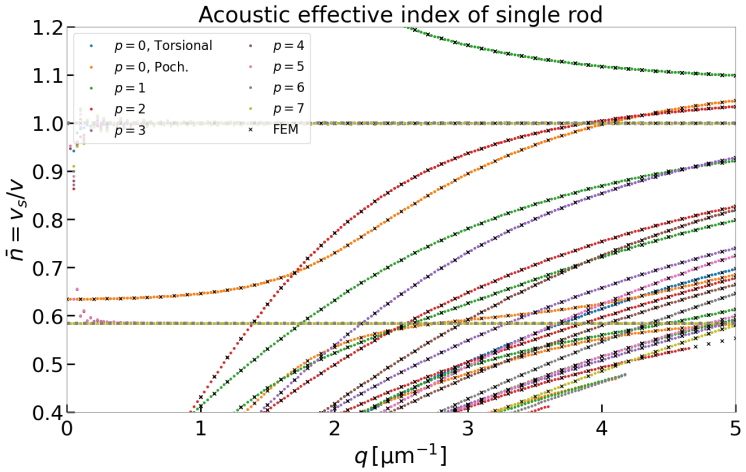


Fig. 48: Elastic "effective index" defined as the ratio of the bulk shear velocity to the phase velocity $n_{\text{eff}} = V_s/V$, for a chalcogenide rod in vacuum.

4.3.13 Tutorial 14 – Multilayered 'Onion'

This tutorial, contained in `simo-tut_14-multilayer-fibre.py` shows how one can create a circular waveguide with many concentric layers of different thickness. In this case, the layers are chosen to create a concentric Bragg grating of alternating high and low index layers. As shown in C. M. de Sterke, I. M. Bassett and A. G. Street, "Differential losses in Bragg fibers", J. Appl. Phys. **76**, 680 (1994), the fundamental mode of such a fibre is the fully azimuthally symmetric TE_0 mode rather than the usual HE_{11} quasi-linearly polarised mode that is found in standard two-layer fibres.

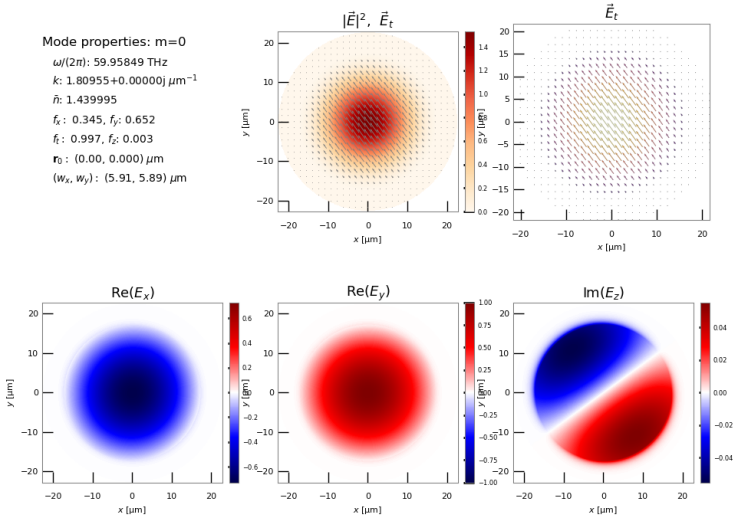


Fig. 49: Fundamental electromagnetic mode profile for the concentric Bragg fibre.

4.3.14 Tutorial 15 – Coupled waveguides

This tutorial, contained in `tutorials/simo-tut_14-coupled-wg.py` demonstrates the supermode behaviour of both electromagnetic and elastic modes in a pair of closely adjacent waveguides.

4.3.15 Tutorial 16 – Using NumBAT in a Jupyter notebook

For those who like to work in an interactive fashion, NumBAT works perfectly well inside a Jupyter notebook. This is demonstrated in the file `jup_16_smf28.ipynb` using the standard SMF-28 fibre problem as an example.

On a Linux system, you can open this at the command line with:

```
$ jupyter-notebook ``jup_16_smf28.ipynb``
```

or else load it directly in an already open Jupyter environment.

The notebook demonstrates how to run standard NumBAT routines step by step. The output is still written to disk, so the notebook includes some simple techniques for efficiently displaying mode profiles and spectra inside the notebook.

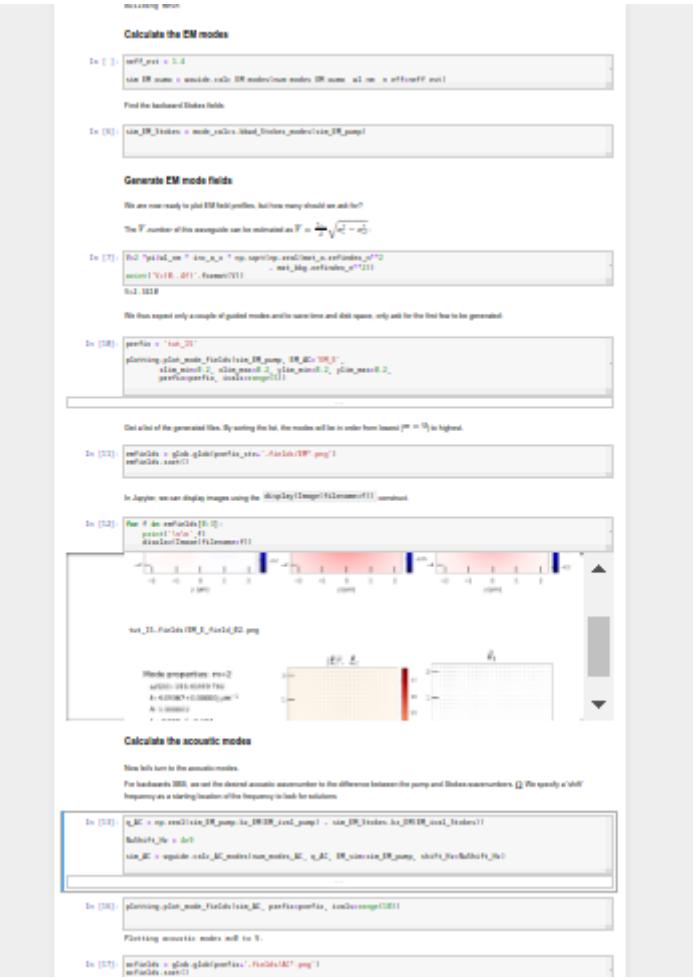


Fig. 50: NumBAT calculation of SMF-28 modes within a Jupyter notebook.

CHAPTER

FIVE

JOSA-B TUTORIAL PAPER

5.1 Introduction

Dr Christian Wolff and colleagues have used NumBAT throughout their 2021 SBS tutorial paper Brillouin scattering—theory and experiment: tutorial published in J. Opt. Soc. Am. B. This set of examples works through their discussions of backward SBS, forward Brillouin scattering, and intermodal forward Brillouin scattering.

As the calculations in this paper used NumBAT with essentially the same code in these tutorials, we do not bother to include the original figures.

5.1.1 Example 1 – Backward SBS in a circular silica waveguide

Figure 15 in the paper shows the fundamental optical field of a silica 1 micron waveguide, with the gain and other parameters shown in Table 2. The corresponding results generated with `simo-josab-01.py` are as follows:

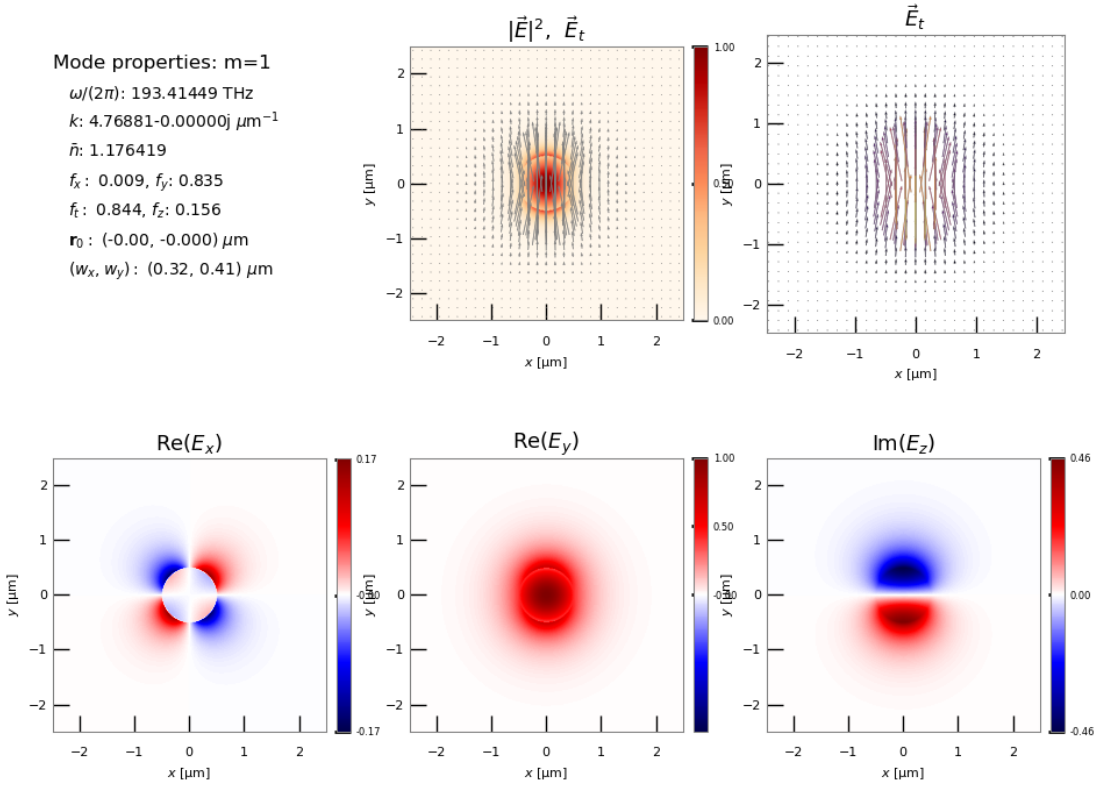


Fig. 1: Fundamental optical mode fields.

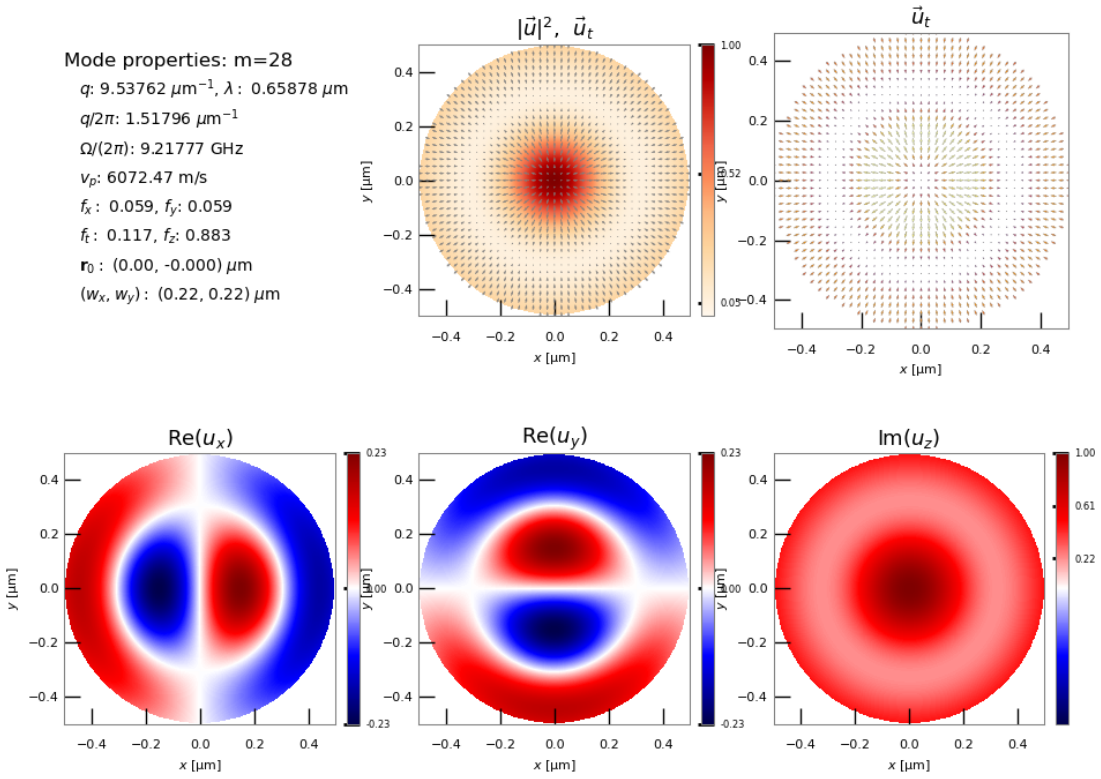


Fig. 2: Elastic mode with largest SBS gain.

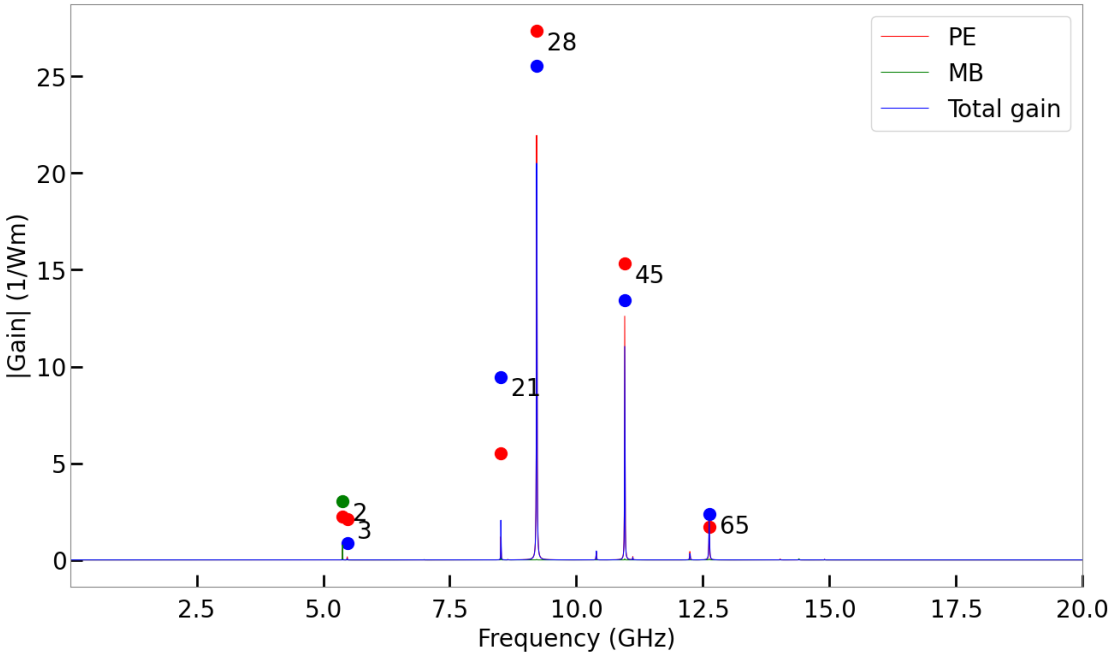
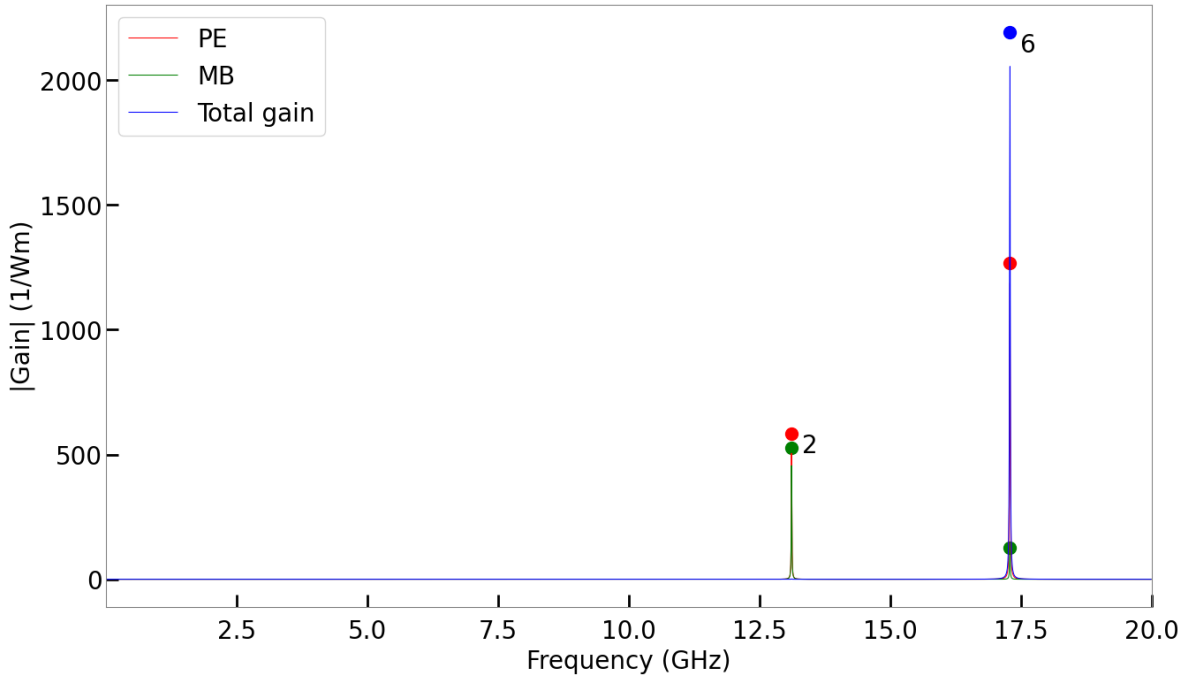


Fig. 3: Gain spectrum for the silica waveguide.

5.1.2 Example 2 – Backward SBS in a rectangular silicon waveguide

Figure 14 in the paper calculates the backwards SBS properties of a rectangular $450x200$ nm silicon waveguide. The corresponding results generated with `simo-josab-02.py` are as follows:



The fields and gain parameters are as follows:

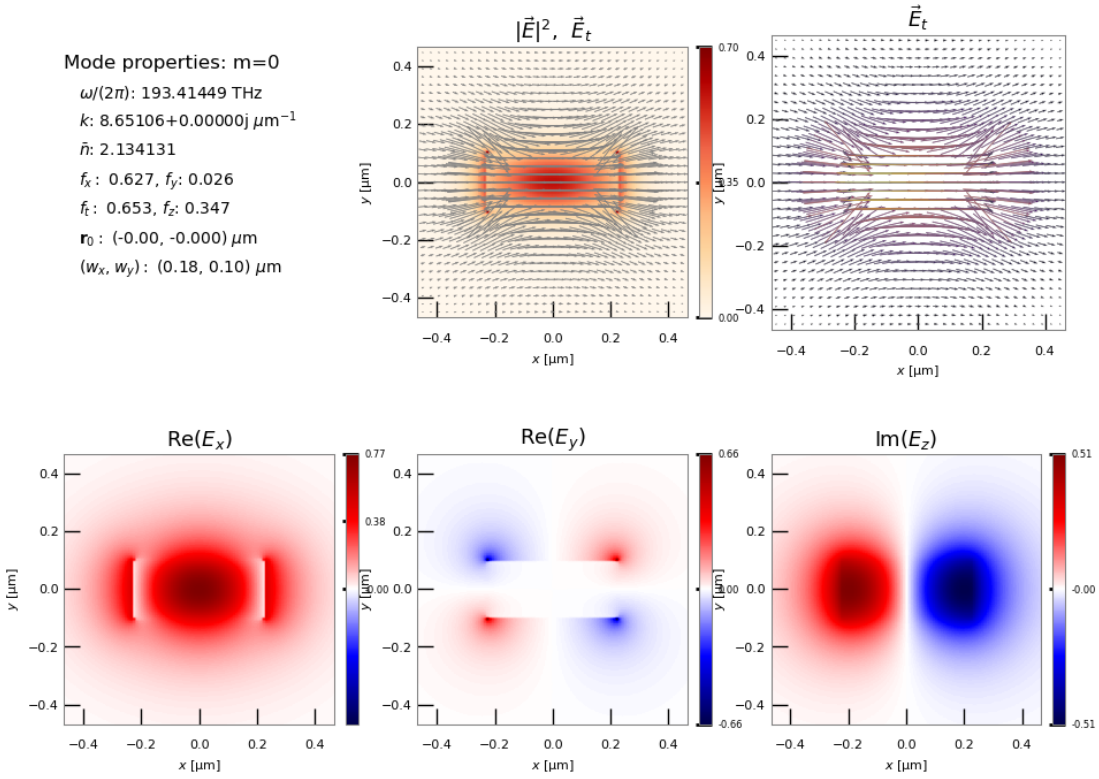


Fig. 4: Fundamental optical mode fields.

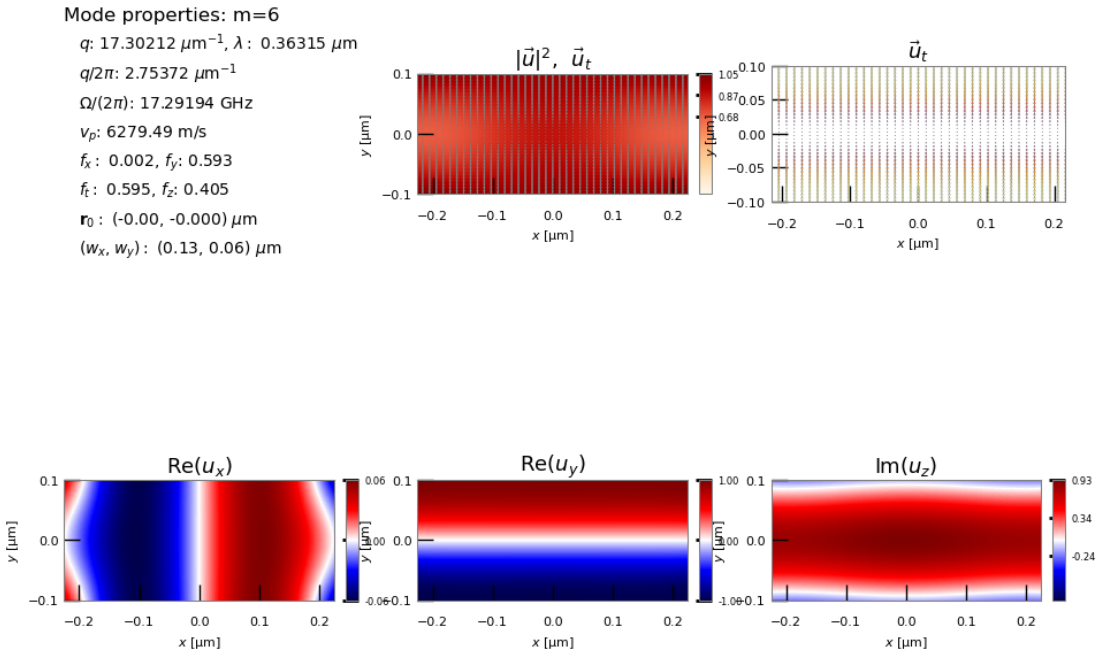


Fig. 5: Fundamental elastic mode fields.

We can reproduce Fig. 13 showing the elastic dispersion of this waveguide silicon waveguide using `simo-josab-02b-acdisp.py`. .. figure:: ../../josab_tutorial/josab_02b-disp-qnu.png

width
15cm

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`

5.1.3 Example 3 – Forward Brillouin scattering in a circular silica waveguide

Figure 16 and Table 3 examine the same waveguides in the case of forward Brillouin scattering.

These results can be generated with `simo-josab-03.py` and `simo-josab-04.py`.

Let's see the results for the silica cylinder first:

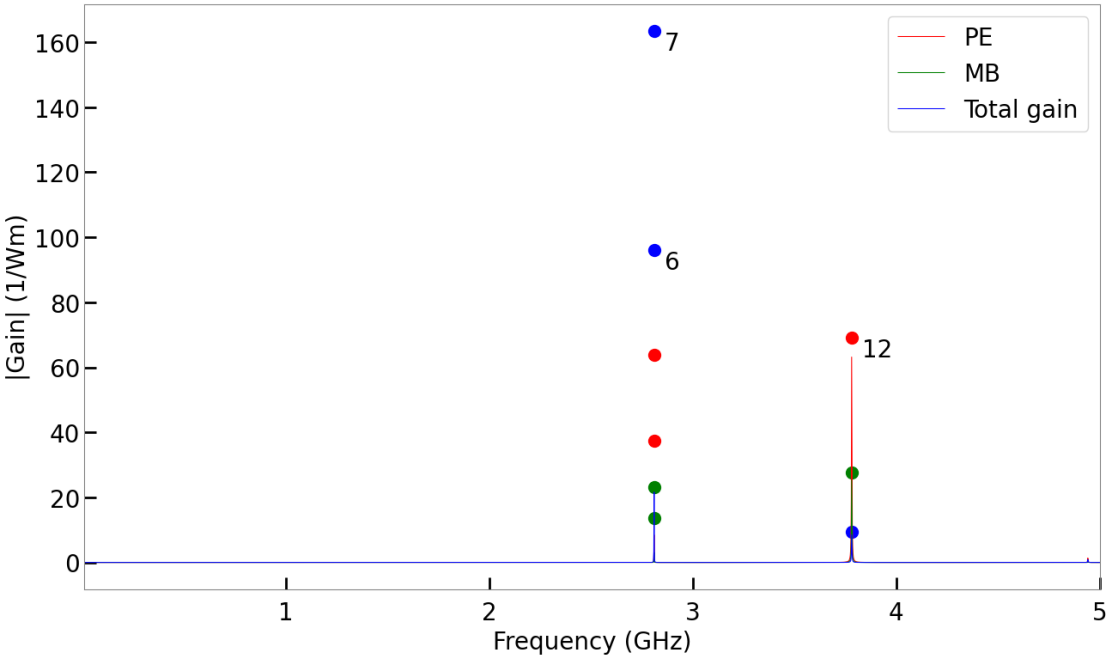


Fig. 6: Gain spectrum for forward SBS of the silica cylinder.

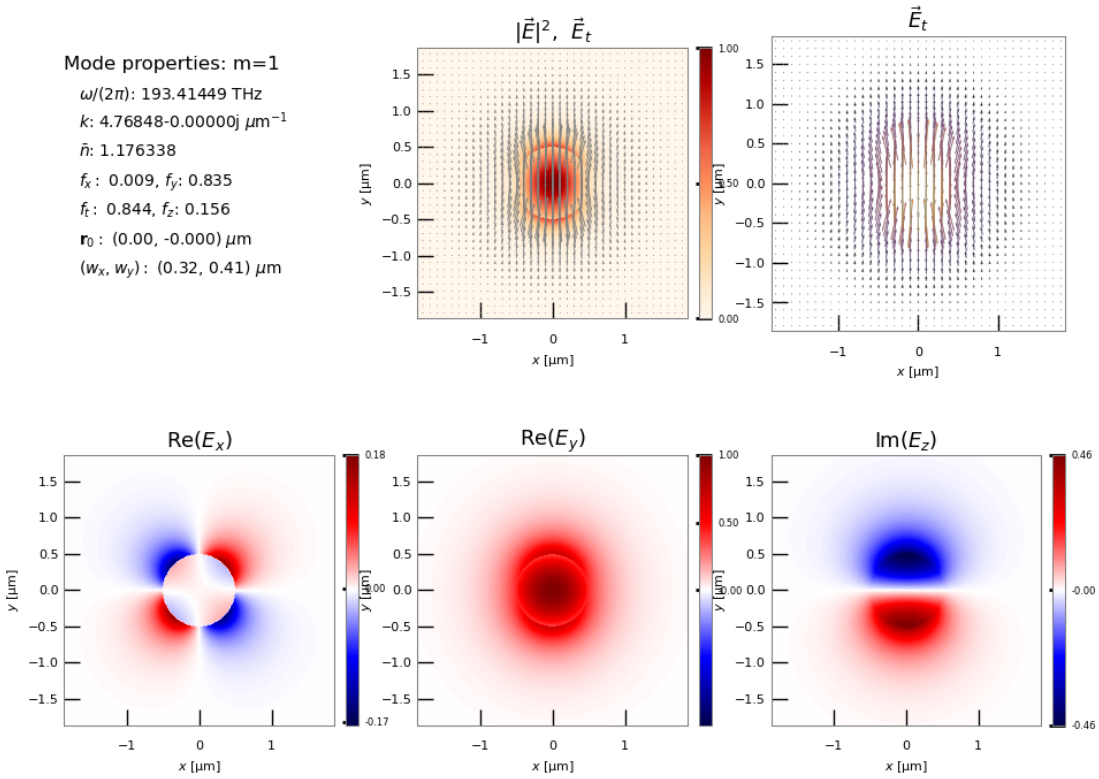


Fig. 7: Fundamental optical mode field.

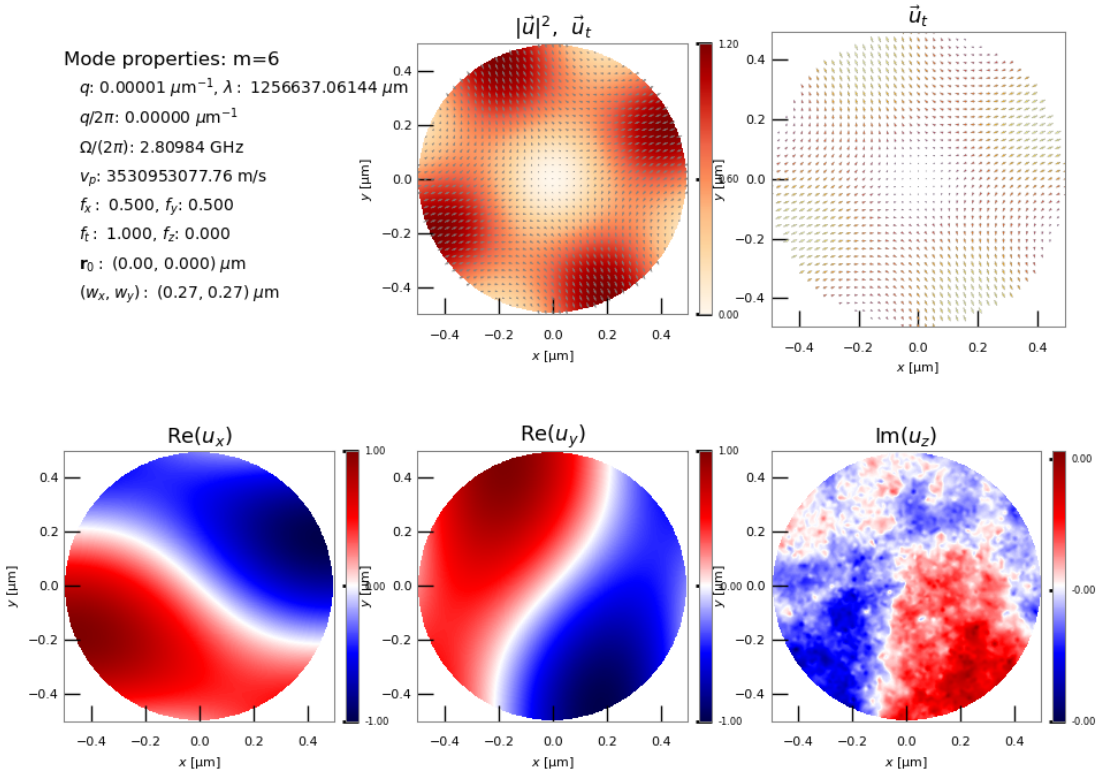


Fig. 8: Elastic mode of maximum gain.

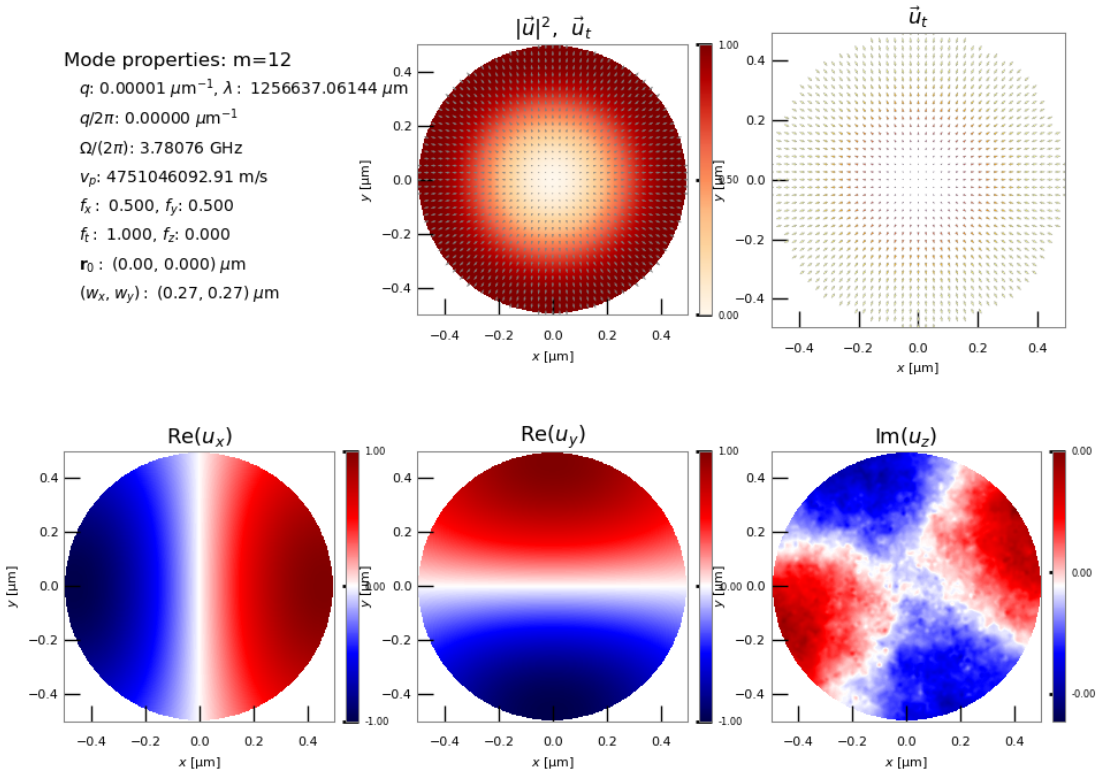


Fig. 9: Elastic mode of second highest gain.

5.1.4 Example 4 – Forward Brillouin scattering in a rectangular silicon waveguide

The corresopnding results for the silicon waveguide can be generated with `simo-josab-04.py`:

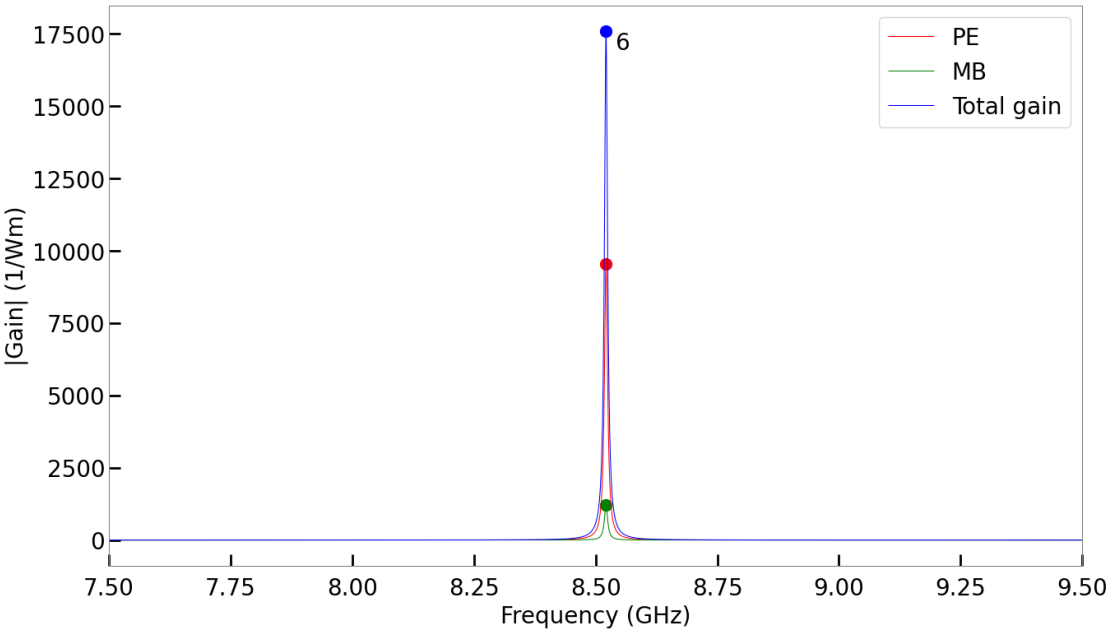


Fig. 10: Gain spectrum for forward SBS of the silicon waveguide.

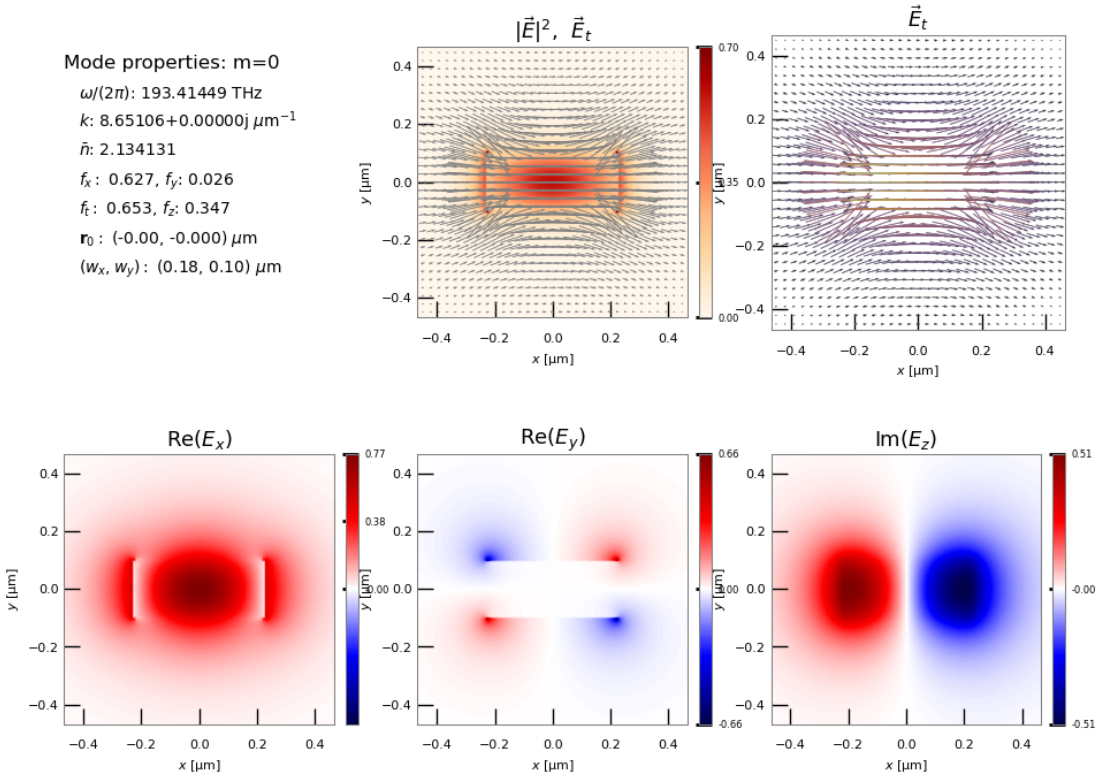


Fig. 11: Fundamental optical mode field.

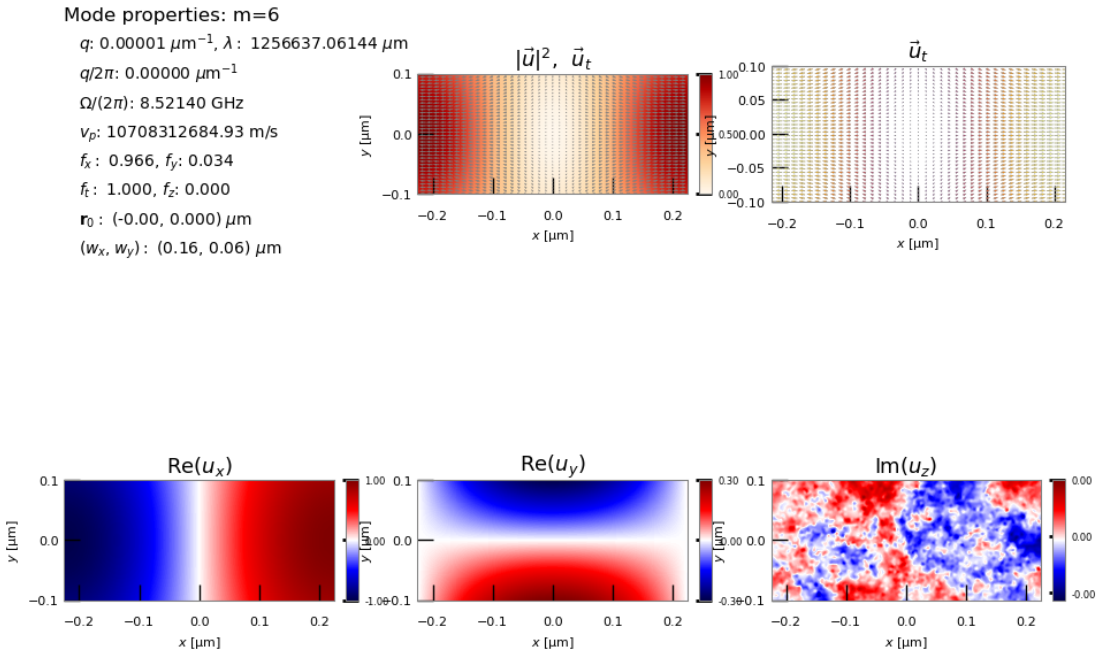


Fig. 12: Elastic mode of maximum gain.

5.1.5 Example 5 – Intermodal Forward Brillouin scattering in a circular silica waveguide

For the problem of intermodal FBS, the paper considers coupling between the two lowest optical modes. The elastic mode of highest gain is actually a degenerate pair:

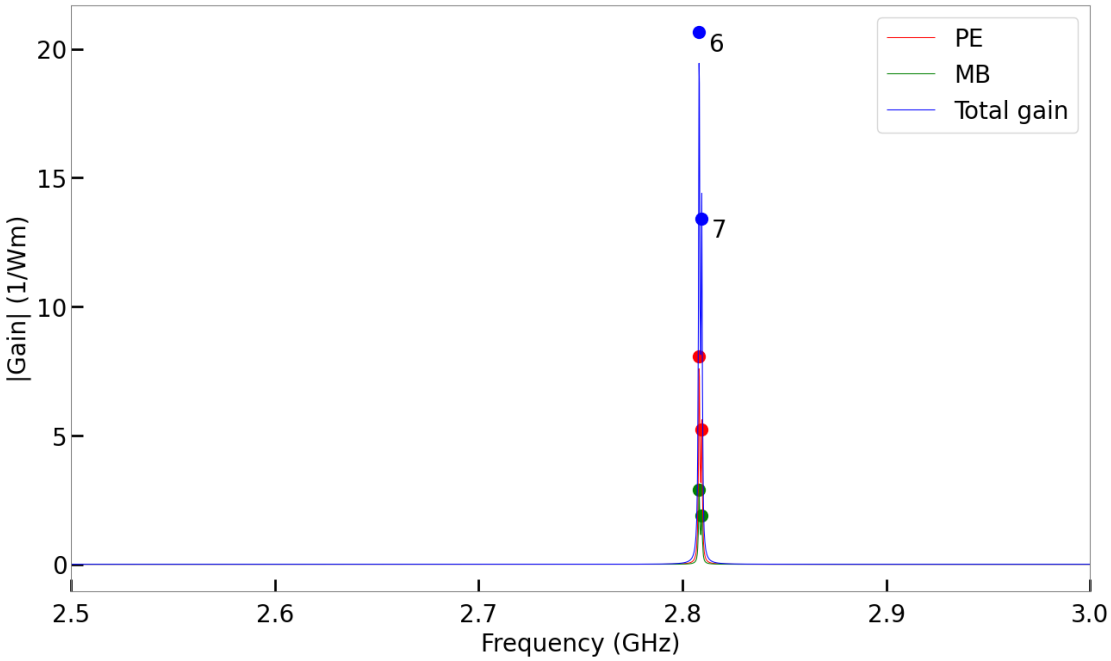


Fig. 13: Gain spectrum for intermodal forward SBS of the silica waveguide.

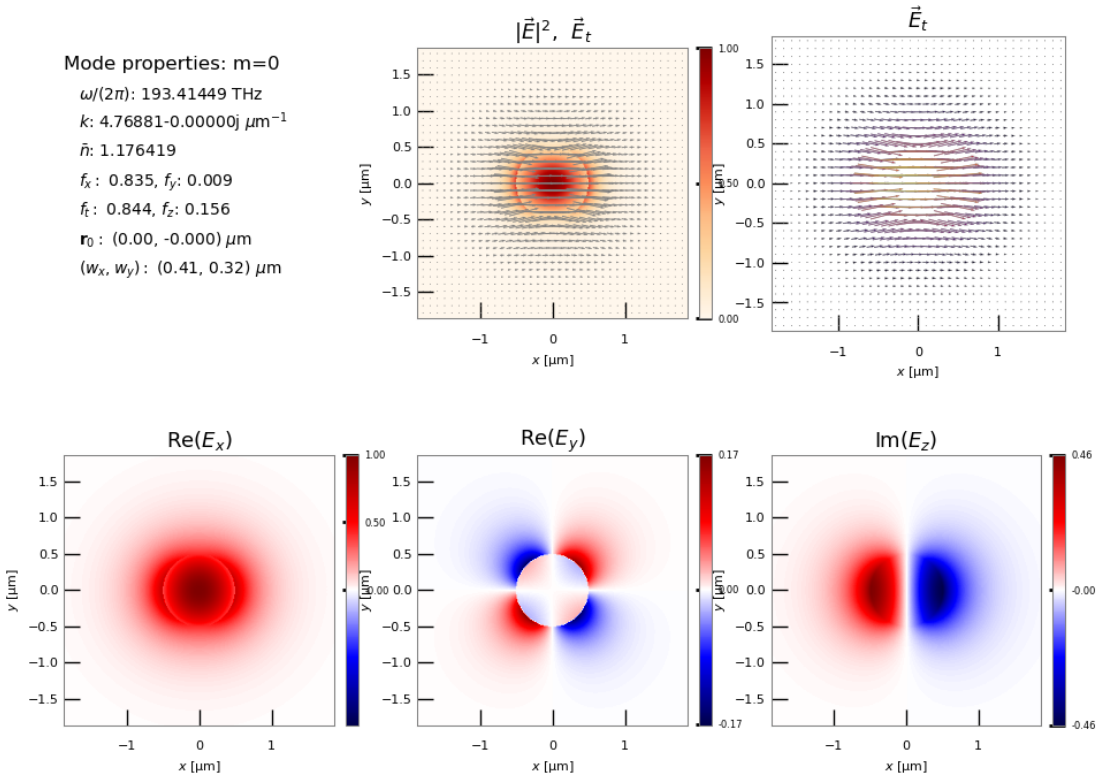


Fig. 14: Fundamental optical mode field.

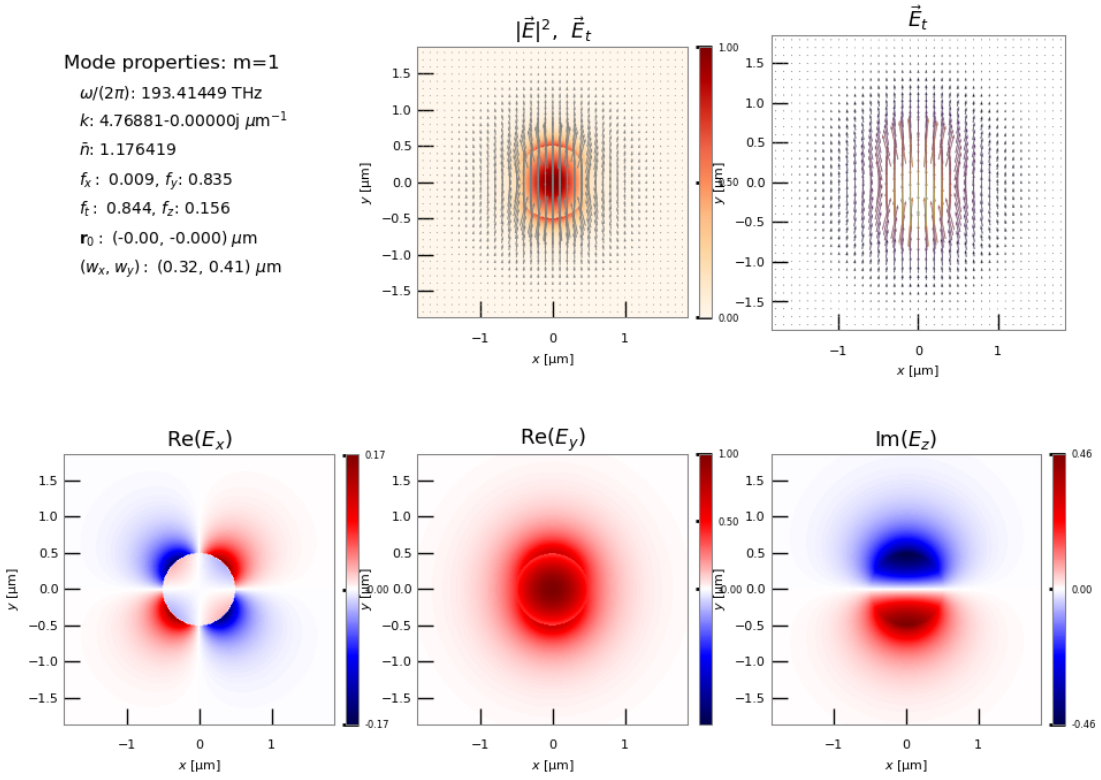


Fig. 15: Second order optical mode field.

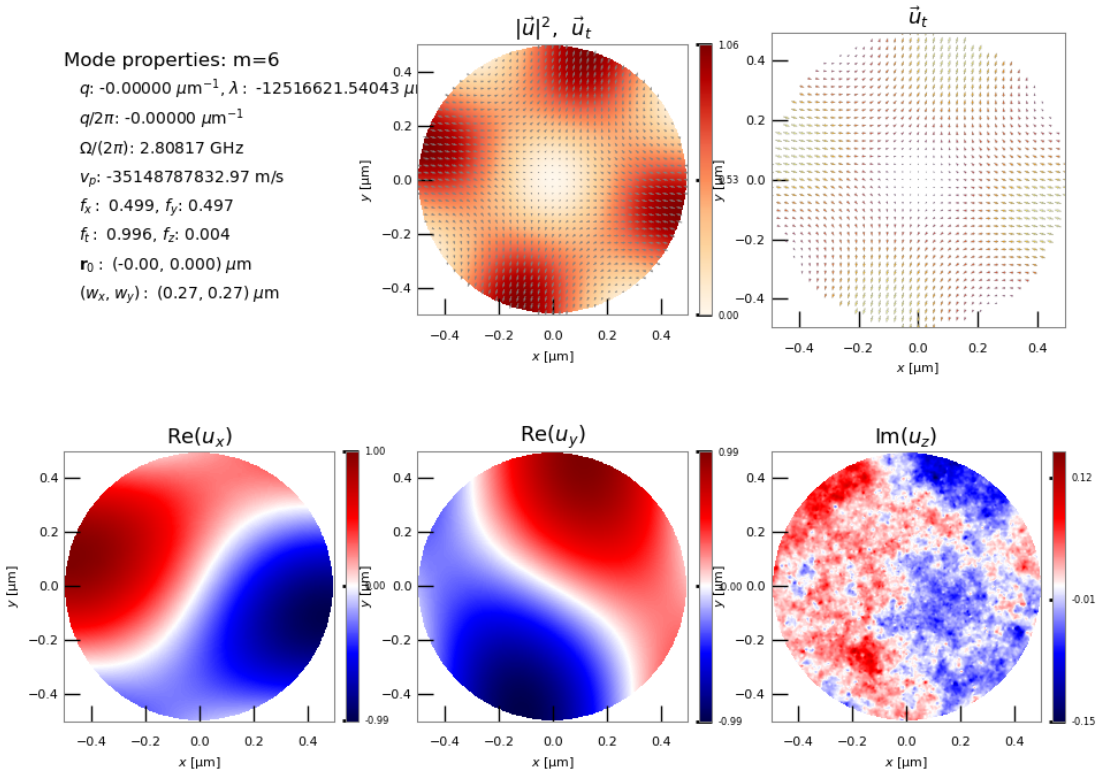


Fig. 16: Elastic mode field of maximum gain.

5.1.6 Example 6 – Intermodal Forward Brillouin scattering in a rectangular silicon waveguide

Finally, the silicon waveguide generates extraordinarily high gain when operated in an intermodal configuration:

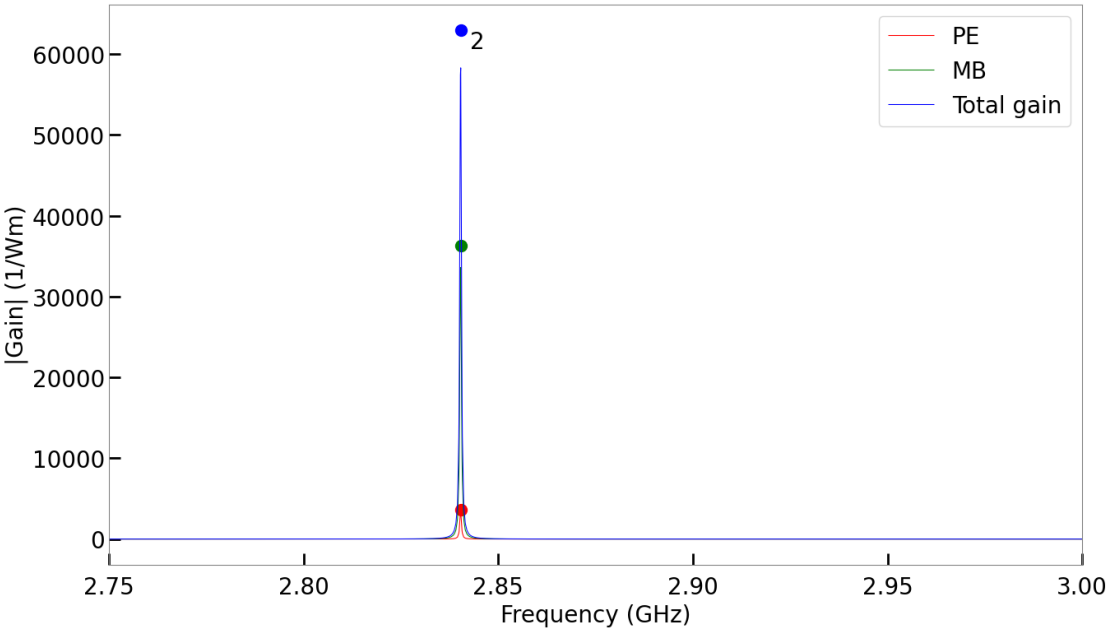


Fig. 17: Gain spectrum for intermodal forward SBS of the silicon waveguide.

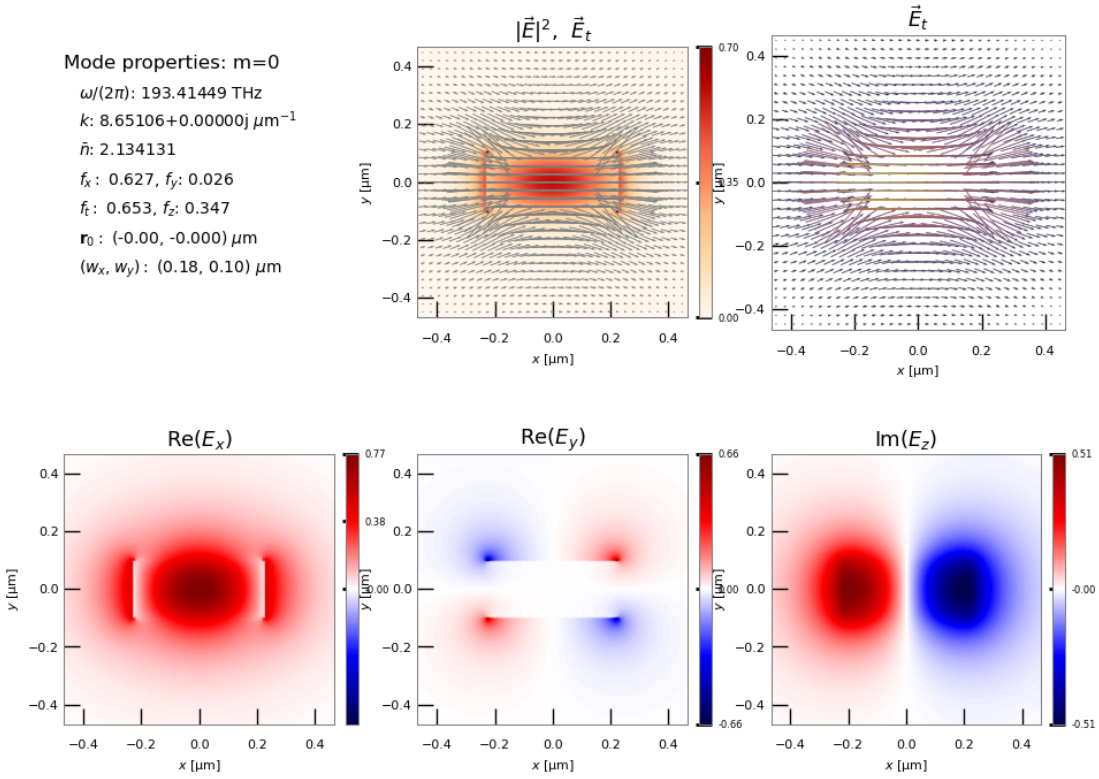


Fig. 18: Fundamental optical mode field.

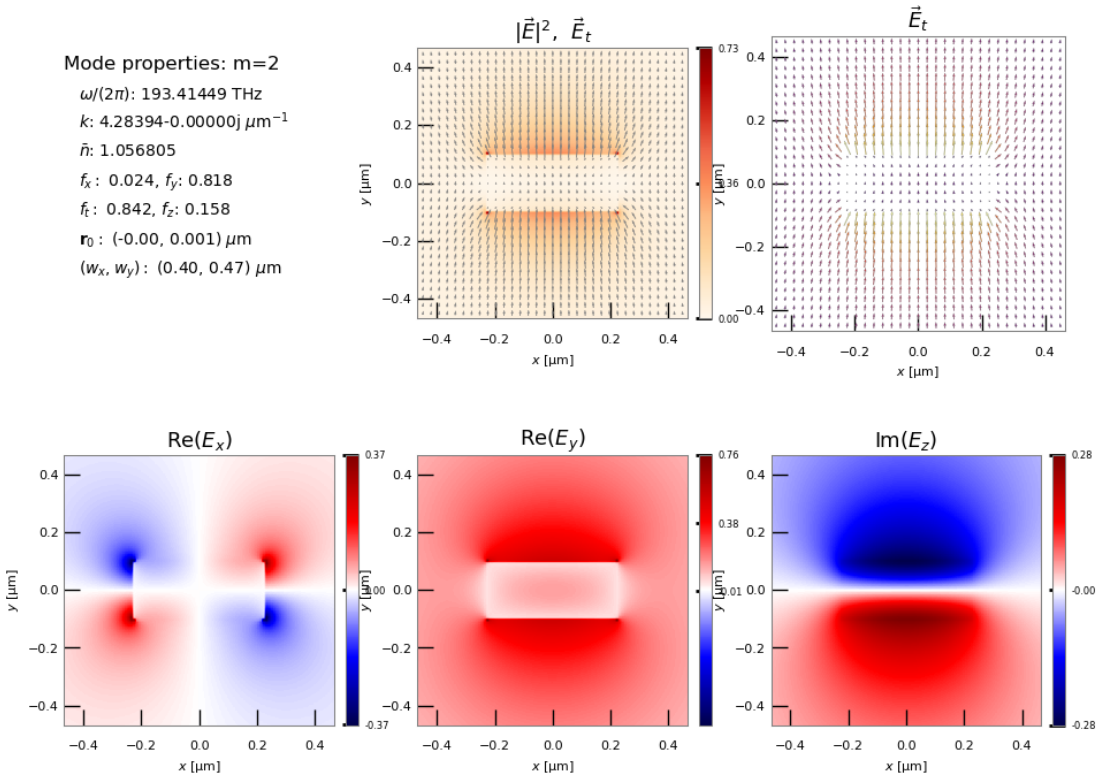


Fig. 19: Second order optical mode field.

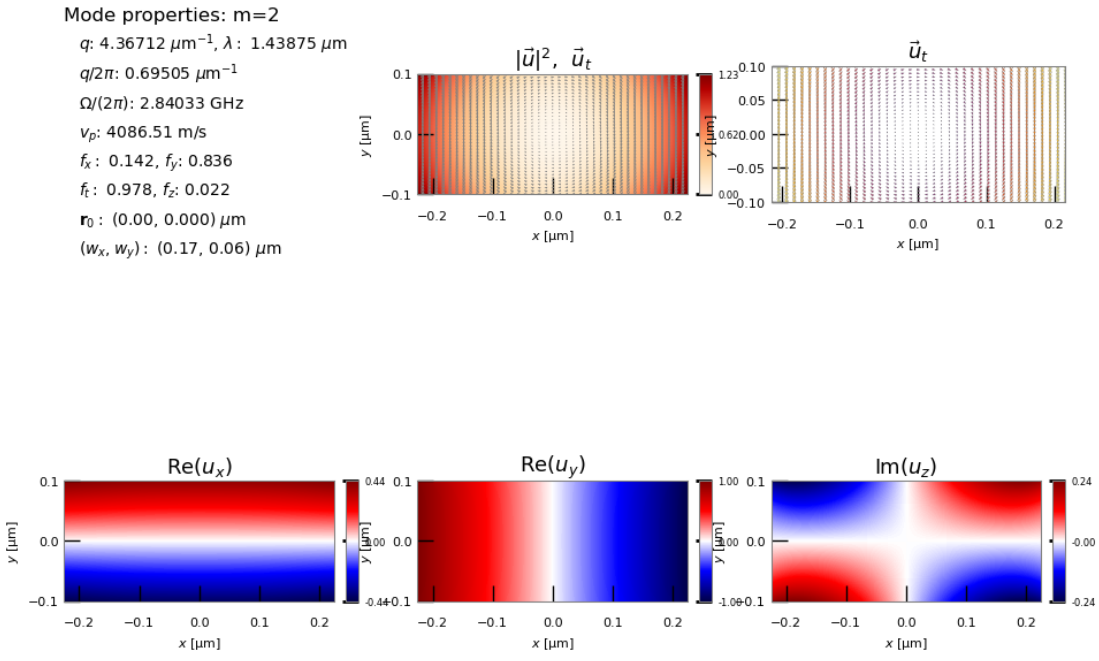


Fig. 20: Elastic mode field of maximum gain.

CHAPTER

SIX

LITERATURE EXAMPLES

6.1 Introduction

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.

6.1.1 Example 1 – 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 `SiO2_2013_Laude` specifically modelled on the parameters in this paper. This naming scheme for materials allows several different versions of nominally the same material to be selected,so that users can easily compare calculations to other authors's exact parameter choices, without changing their preferred material values for their own samples and experiments.

In this paper, Laude and Beugnot plot a spectrum of the elastic energy density, which is not directly measureable. However the spectral peaks show close alignment with the NumBAT gain spectrum for the same structure as is apparent in the second plot. We attribute the remaining difference in the location of the spectral peaks to the choice of elastic material properties in the paper. The paper reports a bulk shear velocity of 3400 m/s, which is around 8% smaller than the usual value of approximately 3760 m/s, but the full set of material properties used in the calculations are not provided. Hence we have used a more standard set of values.

The modal profiles for the peaks marked C and D can also be compared with those from the paper in the subsequent plots. The most direct comparison comes from looking at the NumBAT contour plots for the transverse and longitudinal elastic fields. Note that the transverse and axial plots for mode D in the paper are inconsistent. The axial plot is the correct one for this mode.

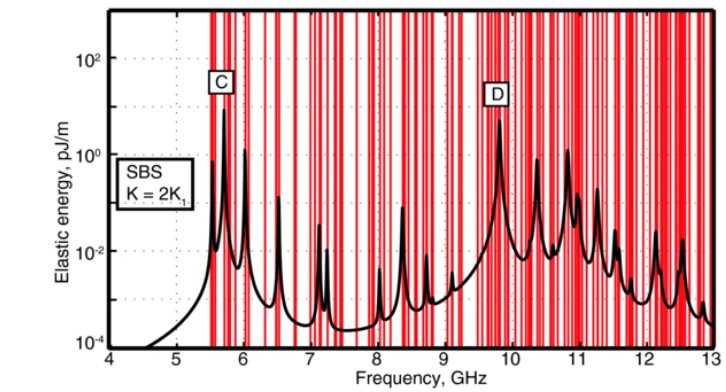


Fig. 1: Spectrum of elastic mode energy calculated in Laude and Beugnot for backward SBS in a silica waveguide.

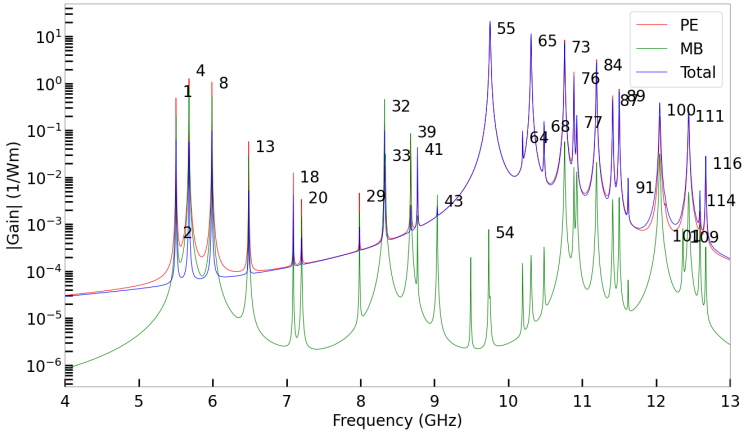


Fig. 2: NumBAT gain spectrum on semilogy axis.

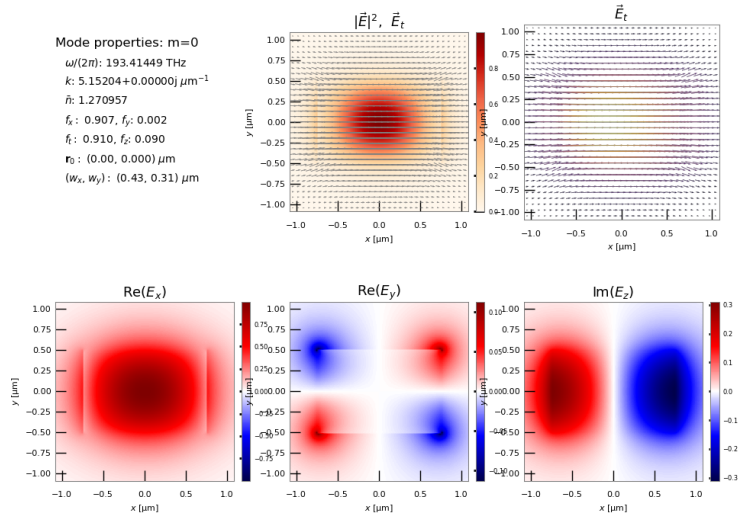


Fig. 3: Fundamental optical mode profiles calculated in NumBAT.

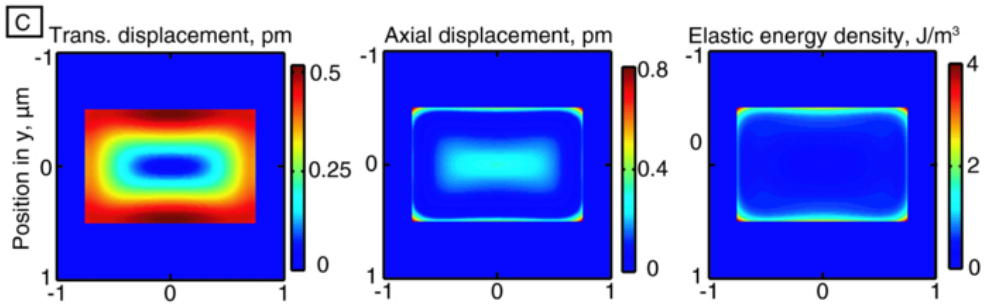


Fig. 4: Elastic mode profiles calculated in Laude and Beugnot for backward SBS in a silica waveguide of diameter 1.05 micron. Mode C corresponds to the peak marked in the spectrum above.

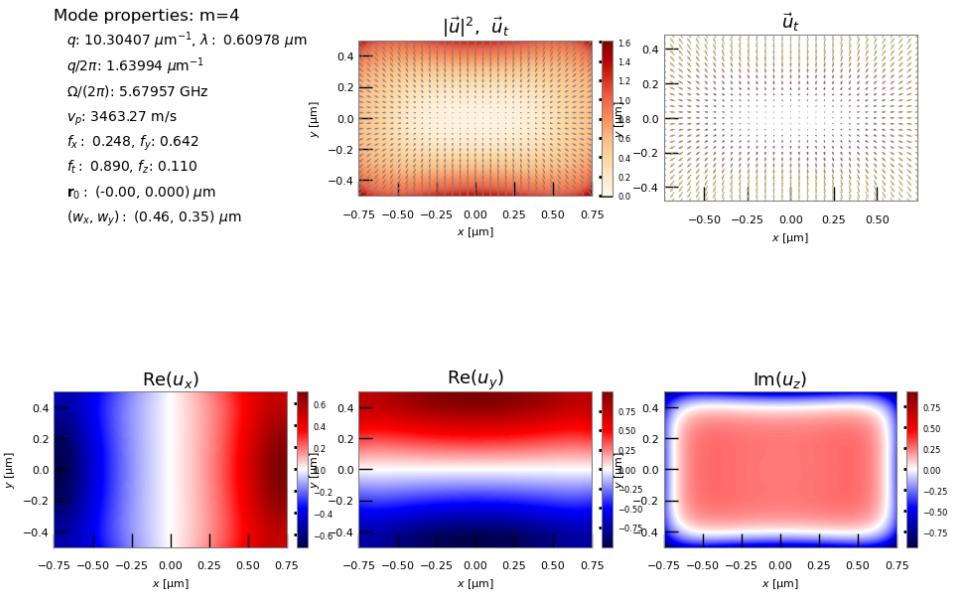


Fig. 5: NumBAT calculation of high gain elastic mode, marked as C in paper.

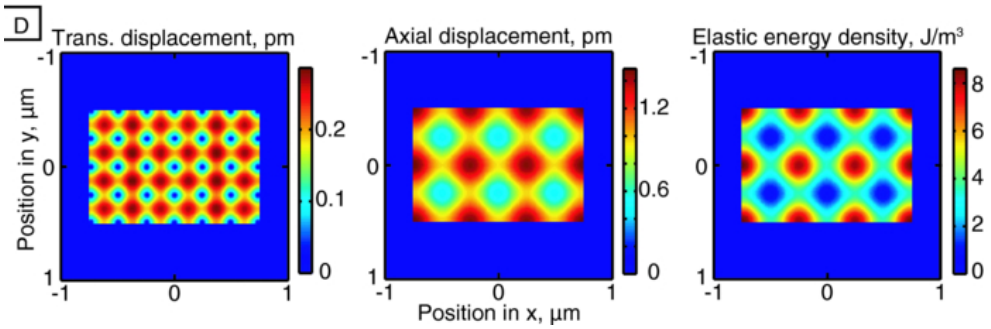


Fig. 6: Elastic mode profiles calculated in Laude and Beugnot for backward SBS in a silica waveguide of diameter 1.05 micron. Mode D corresponds to the peak marked in the spectrum above.

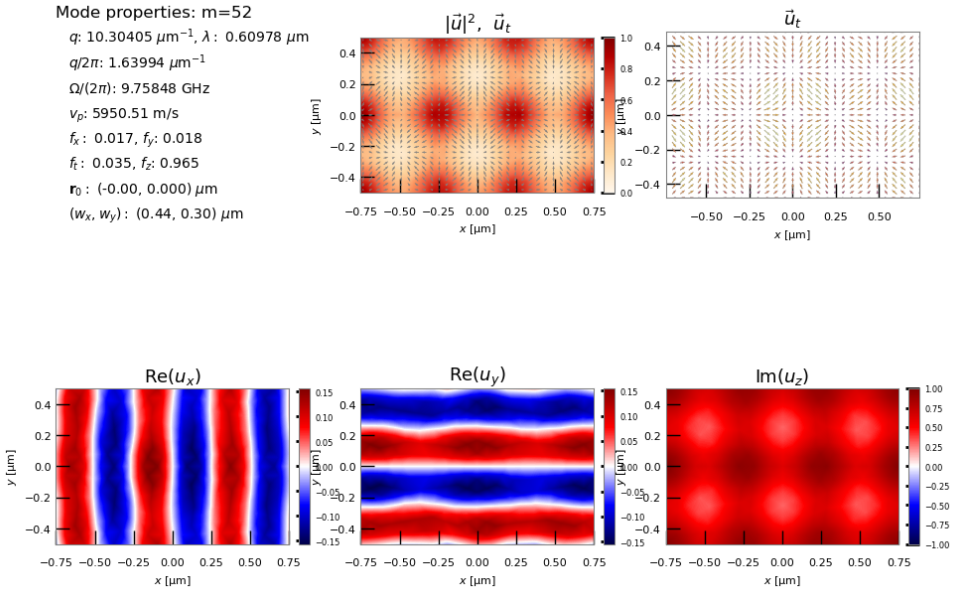


Fig. 7: NumBAT calculation of high gain elastic mode, marked as D in paper.

6.1.2 Example 2 – 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.

Once again, the plots from the orignal paper are shown in the first figure. In this case, with a very large number of modes of different shear wave order, the precise mode profile for highest gain depends very sensitively on the simulation parameters.

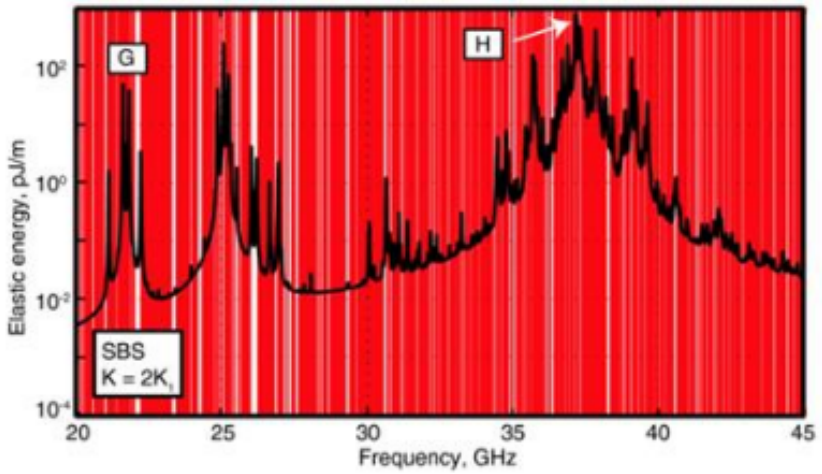


Fig. 8: Spectrum of elastic mode energy calculated in Laude and Beugnot for backward SBS in a silicon waveguide.

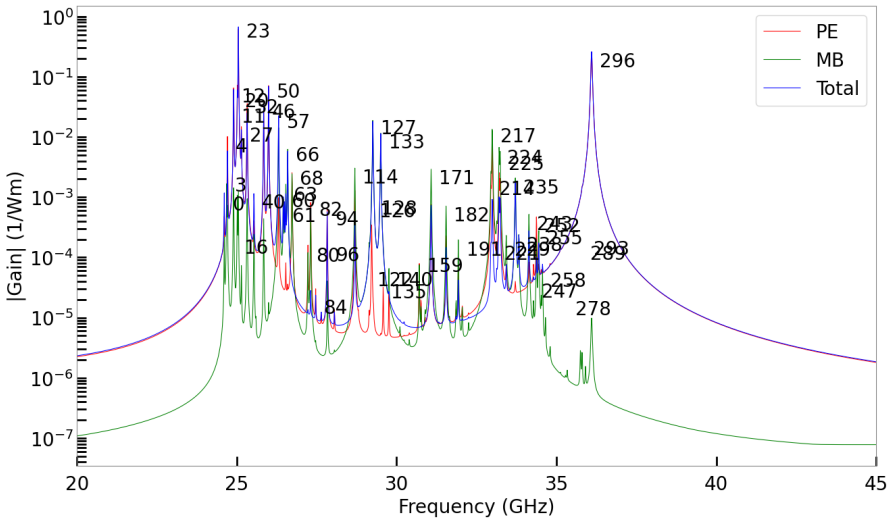


Fig. 9: NumBAT gain spectrum on semilogy axis.

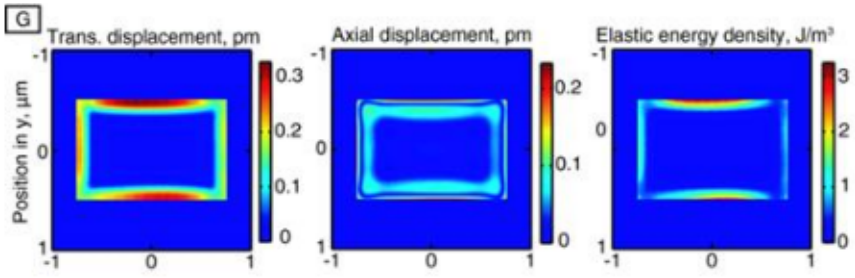


Fig. 10: Field profiles for mode G calculated in Laude and Beugnot for backward SBS in a silicon waveguide.

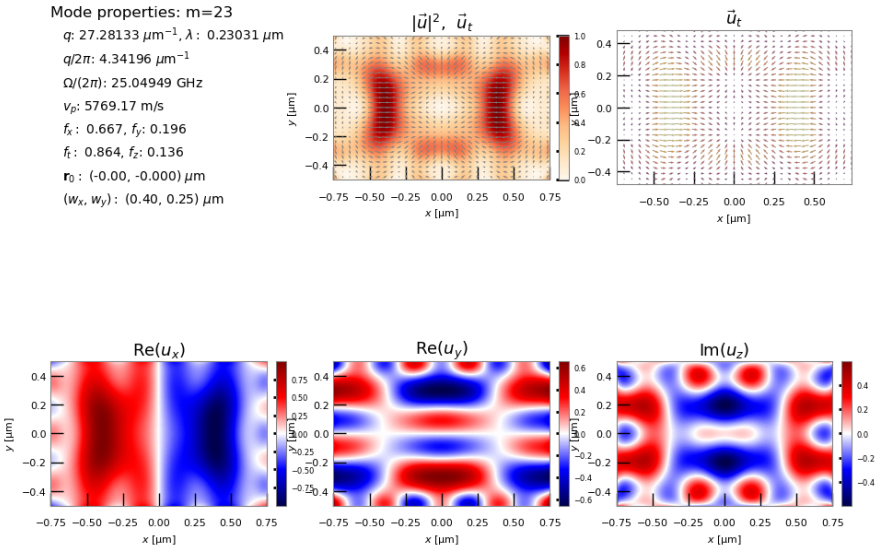


Fig. 11: High gain elastic mode calculated by NumBAT, marked as G in paper.

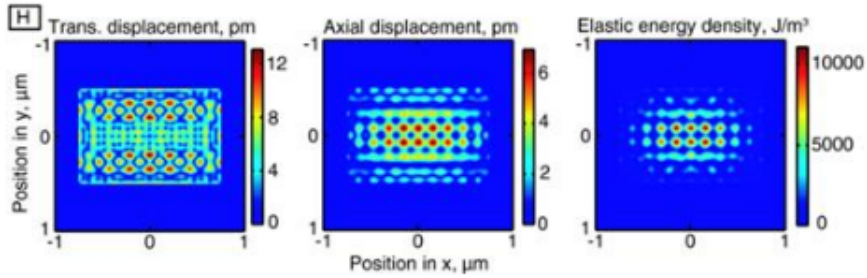


Fig. 12: Field profiles for mode H calculated in Laude and Beugnot for backward SBS in a silicon waveguide.

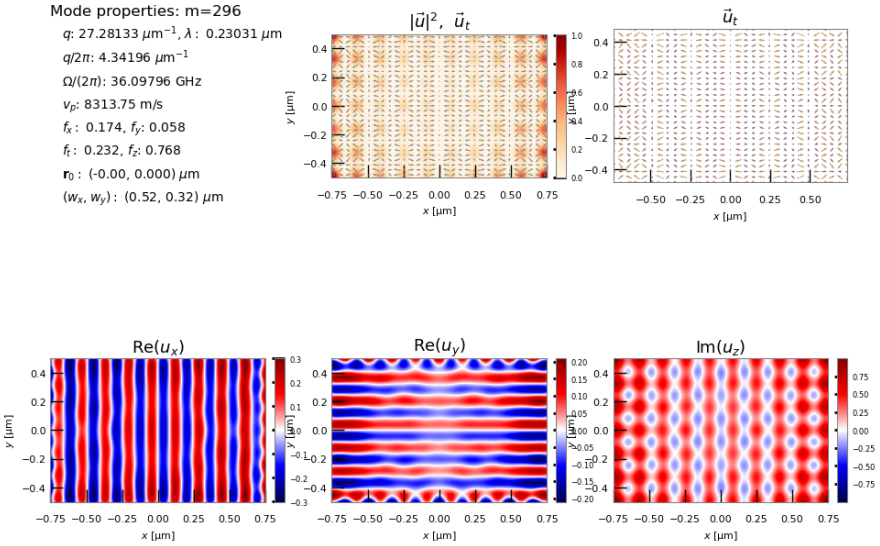


Fig. 13: High gain elastic mode calculated by NumBAT, marked as H in paper.

6.1.3 Example 3 – 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 elastic waves in a subwavelength-diameter optical fibre, *Nature Communications* **5**, 5242 (2014).

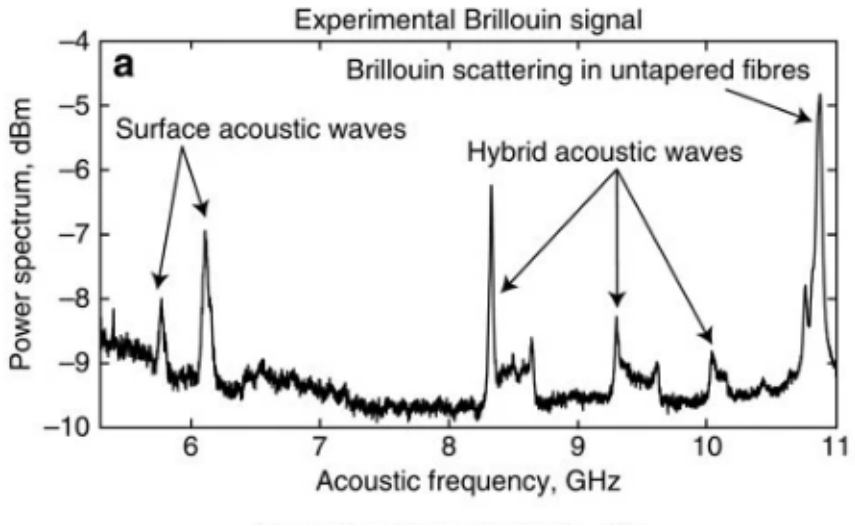


Fig. 14: Measured gain spectrum for a 1 micron silica nanowire in J.-C. Beugnot *et al.*

../../lit_examples/lit_03-gain_spectra-logy_w1000.png

Fig. 15: NumBAT calculated gain spectrum for the 1 micron silica nanowire.

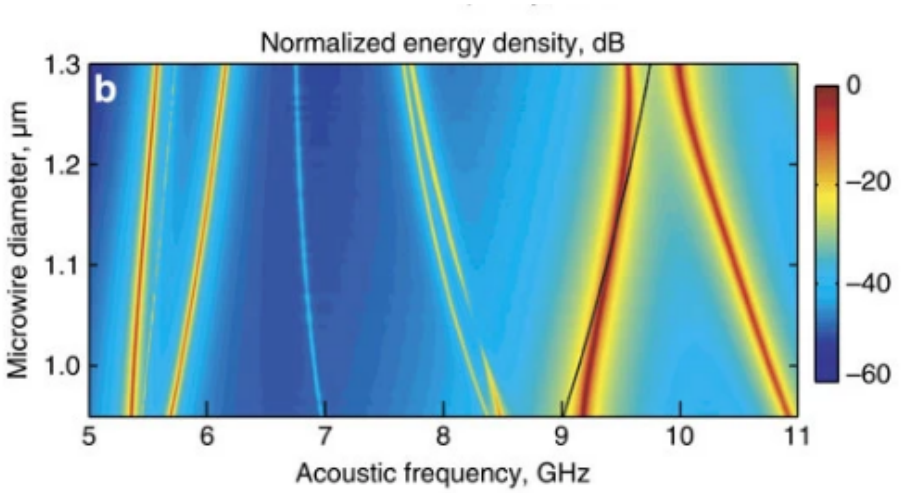


Fig. 16: Calculated dispersion of gain spectrum with nanowire width in J.-C. Beugnot *et al.*

../../lit_examples/lit_03-gain_tot-diam_scan.png

Fig. 17: Full elastic wave spectrum for silica microwire, as per Fig. 4a in paper.

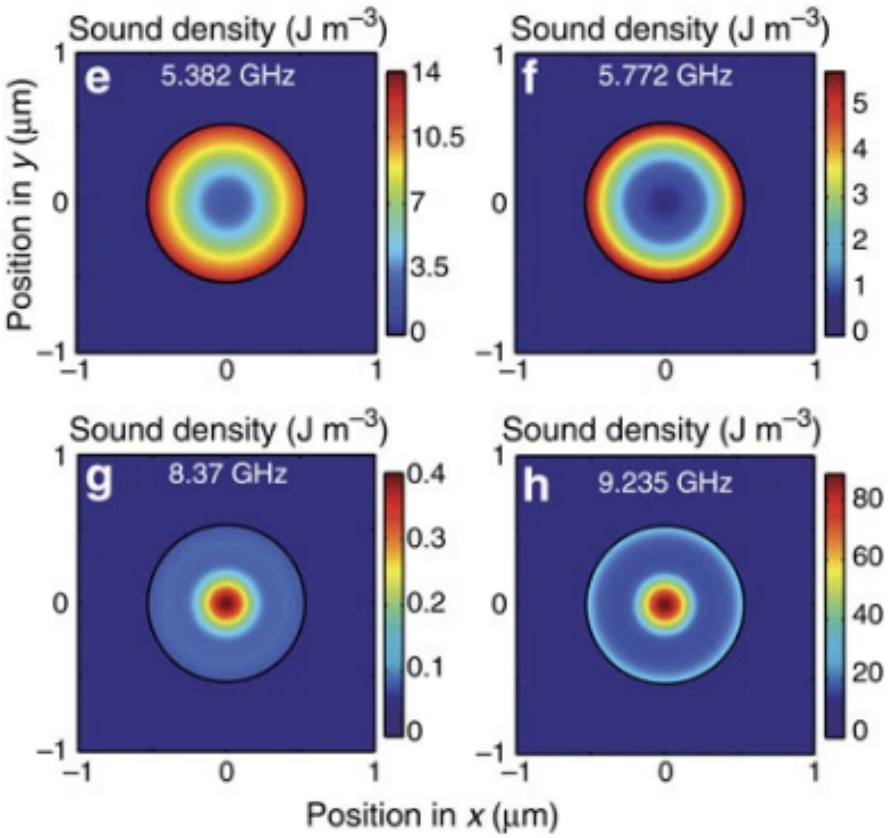
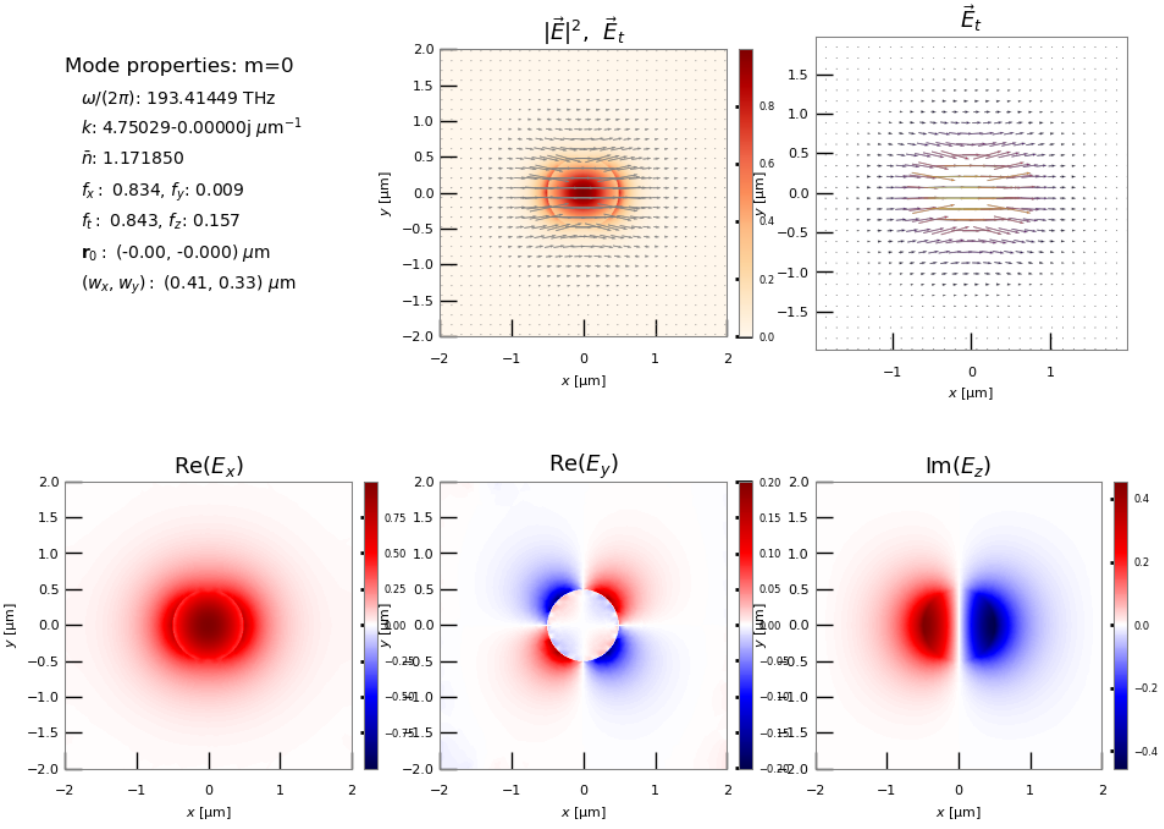


Fig. 18: Calculated elastic mode profiles for the 1 micron nanowire in J.-C. Beugnot *et al.*



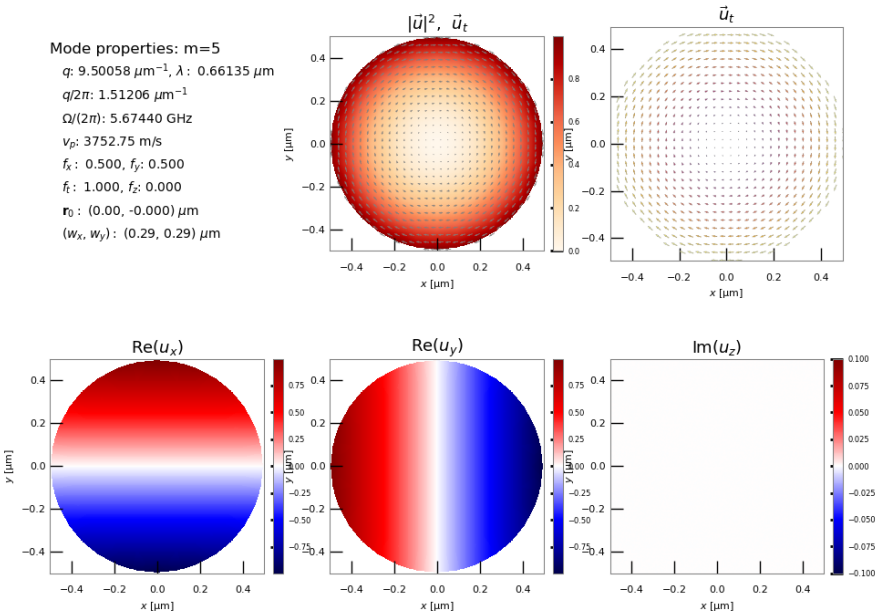


Fig. 19: Azimuthal shear mode.

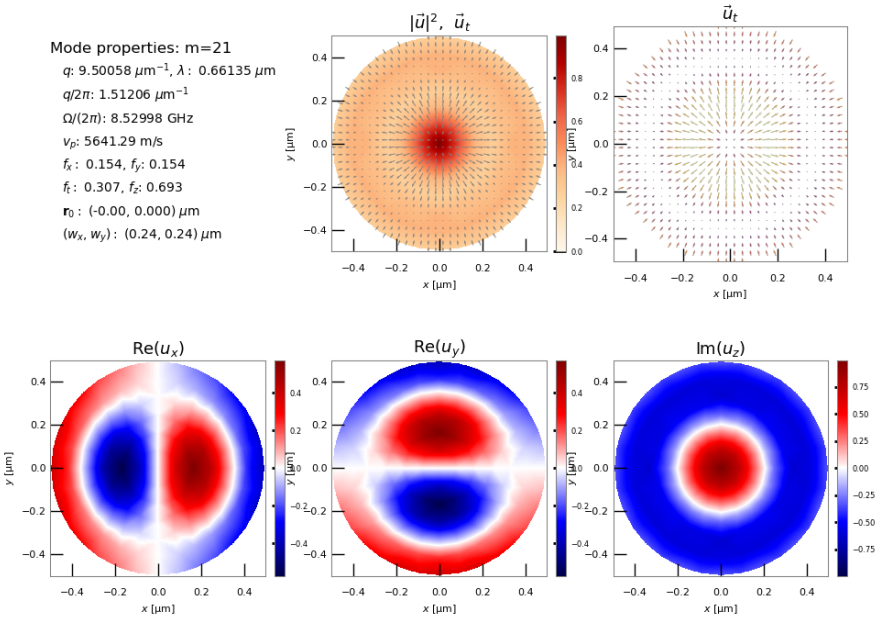


Fig. 20: Azimuthal shear mode.

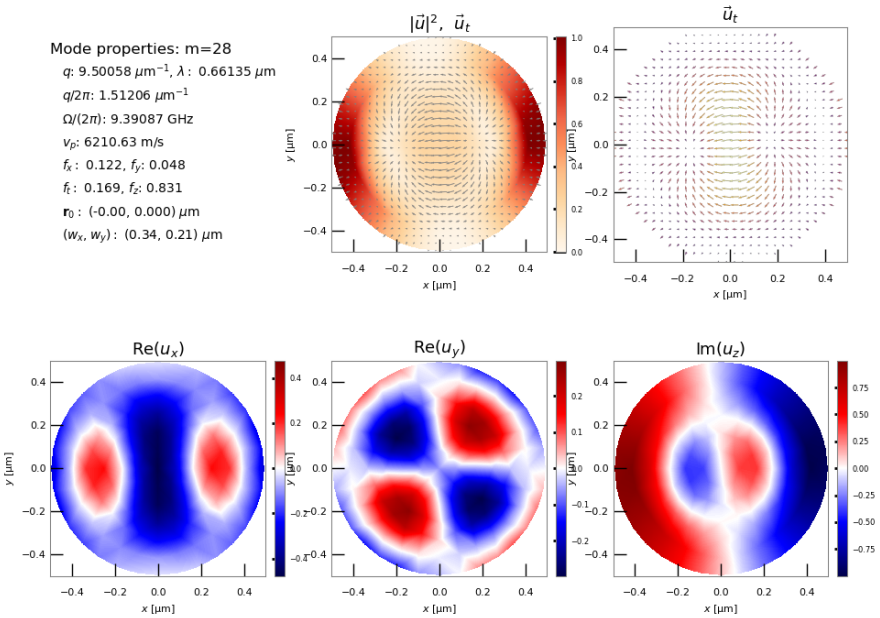


Fig. 21: Azimuthal shear mode.

6.1.4 Example 4 – 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 elastic model causes a problem where the slab layer significantly distorting elastic 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.

We may also choose to study the simplified situation where the pedestal is removed, which gives good agreement for the gain spectrum:

6.1.5 Example 5 – 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 elastic loss calculation incorrect but avoids the problem of elastic energy being excessively concentrated in the substrate.

6.1.6 Example 6 – BSBS self-cancellation in a tapered fibre (small fibre)

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).

6.1.7 Example 6b – BSBS self-cancellation in a tapered fibre (large fibre)

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.

6.1.8 Example 7 – FSBF in a silicon rib waveguide

This example, in `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).

6.1.9 Example 8 – 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.

6.1.10 Example 9 – 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.

6.1.11 Example 10 – SBS in the mid-infrared

This example, in `simo-lit_10-Wolff-OptExpress-2014.py` and `simo-lit_10a-Wolff-OptExpress-2014.py`, by C. Wolff and collaborators examines backward SBS in the mid-infrared using germanium as the core material in a rectangular waveguide with a silicon nitride cladding.

The second of these two files illustrates the rotation of the core material from the [100] orientation to the [110] orientation. The second file prints out the full elastic properties of the germanium material in both orientations which are seen to match the values in the paper by Wolff et al.

CHAPTER

SEVEN

PYTHON INTERFACE API

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.

7.1 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.make_material()*.

exception materials.**BadMaterialFileError**

Bases: `Exception`

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]

Vac_longitudinal()

For an isotropic material, returns the longitudinal (P-wave) elastic phase velocity.

Vac_shear()

For an isotropic material, returns the shear (S-wave) elastic phase velocity.

elastic_properties()

Returns a string containing key elastic properties of the material.

full_str()

has_elastic_properties()

Returns true if the material has at least some elastic properties defined.

is_isotropic()

is_vacuum()

Returns True if the material is the vacuum.

load_cubic_crystal()

load_general_crystal()

load_tensors()

load_trigonal_crystal()

rotate_axis(*theta*, *rotate_axis*, *save_rotated_tensors=False*)

Rotate crystal axis by theta radians.

Args:

theta (float): Angle to rotate by in radians.

rotate_axis (str): Axis around which to rotate.

Keyword Args:

save_rotated_tensors (bool): Save rotated tensors to csv.

Returns:

`Material` object with rotated tensor values.

set_refractive_index(*nr*, *ni=0.0*)

class materials.**VoigtTensor4**(*sym*, *src_dict=None*, *src_file=None*)

Bases: `object`

A class for representing rank 4 tensors in the compact Voigt representation.

dump()

load_isotropic()

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

rotate(*theta*, *rotation_axis*)

set_isotropic(*m11*, *m12*, *m44*)

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

Args:

E (float): Youngs modulus

v (float): Poisson ratio

materials.**make_material**(*s*)

materials.**rotate_tensor_elt**(*i*, *j*, *k*, *l*, *T_pqrs*, *mat_R*)

Calculates the element ijkl of the rotated tensor T' from the original rank-4 tensor T_PQ in 6x6 Voigt notation under the rotation specified by the 3x3 matrix R.

7.2 objects module

The `objects` module provides functions for defining and constructing waveguides. The diagrams in Chapter 2 can be used to identify which parameters (`slab_a_x, slab_c_y, maerial_d` etc) correspond to each region.

class objects.**Structure**(*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.

Args:

unitcell_x (float):
The horizontal period of the unit cell in nanometers.

inc_a_x (float):
The horizontal diameter of the primary inclusion in nm.

Keyword Args:

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 primary 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`. ectangular 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 two inclusions in nm.

incs_y_offset (float):
Vertical offset between centers of two inclusions in nm.

coat_x (float): The width of the first coat layer around
the inclusion.

coat_y (float):
The thickness of the first coating layer around the inclusion.

coat2_x (float):
The width of the second coating layer around the inclusion.

coat2_y (float):
The thickness of the second coating 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 (Material):
The outer background material.

material_a (Material):
The primary inclusion material.

material_b-r (Material):
Materials of additional layers.

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*, *q_AC*, *shift_Hz=None*, *EM_sim=None*, *bcs=None*, *debug=False*, ***args*)

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

Args:
num_modes (int): Number of AC modes to solve for.

Keyword Args:
q_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 (`Simulation` object): Typically an acoustic
simulation follows on from an optical one. Supply the EM `Simulation` object so the AC FEM mesh can be constructed from this. This is done by removing vacuum regions.

Returns:
`Simulation` 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.

Args:
num_modes (int): Number of EM modes to solve for.

wl_nm (float): Wavelength of EM wave in vacuum in nanometres.

n_eff (float): Guesstimated effective index of
fundamental mode, will be origin of FEM search.

Returns:
`Simulation` object

check_mesh()

Visualise geometry and mesh with gmsh.

get_mail_data()

Returns list of lines in the NumBAT mesh format .mail file

get_material(*k*)

make_mesh(*d_materials*)

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

plot_mesh(*outpref*)

Visualise mesh with gmsh and save to a file.

set_threadsafe()

set_xyshift_ac(*x*, *y*)

set_xyshift_em(*x*, *y*)

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

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

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

7.3 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**(*sim*, *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(*n_points=501*, *EM_field=FieldType.EM_E*)

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

clear_mode_plot_data()

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)

interpolate_mode(*mode_helper*)

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

plot_mode(*comps*, *EM_field=FieldType.EM_E*, *ax=None*, *n_points=501*, *decorator=None*)

plot_mode_H(*comps*)

plot_strain()

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**(*sim*, *m*)

Bases: *Mode*

Class representing a single acoustic (AC) mode.

class mode_calcs.**ModeEM**(*sim*, *m*)

Bases: *Mode*

Class representing a single electromagnetic (EM) mode.

class mode_calcs.**ModePlotHelper**(*sim*)

Bases: object

cleanup()

init_arrays()

interpolate_mode_i(*ival*, *field_type*)

plot_strain_mode_i(*ival*)

set_plot_params(*xlim_min=0*, *xlim_max=0*, *ylim_min=0*, *ylim_max=0*, *EM_AC='EM_E'*, *quiver_points=30*, *num_ticks=None*, *ticks=False*, *colorbar=True*, *contours=False*, *contour_lst=None*, *suppress_imimre=True*, *pdf_png='png'*, *prefix='tmp'*, *suffix=''*, *decorator=<plotting.Decorator object>*)

setup_plot_grid(*n_points=501*)

class mode_calcs.**Simulation**(*structure*, *num_modes=20*, *wl_nm=1*, *n_eff=None*, *shift_Hz=None*, *q_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()

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_modes(*n_points=501*)

analyse_symmetries(*ptgrp*)

calc_AC_modes(*bcs=None*)

Run a Fortran FEM calculation to find the acoustic modes.

Returns a `Simulation` 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 num on element, Eig value, el num]

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 `Simulation` 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*)

clean_for_save()

get_mode_helper()

get_modes()

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

Return type
numarray(*Mode*)

get_xyshift()

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()`

`static load_simulation(prefix)`

`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)

save_simulation(*prefix*)

set_r0_offset(*rx*, *ry*)

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 `Simulation` 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 Simulation object that has these key values:

mode_calcs.**load_simulation**(*prefix*)

7.4 integration module

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

class integration.**Gain**

Bases: object

Q_factor_all()

Q_factor_raw()

alpha_all()

alpha_raw()

gain_MB(*m_AC*)

gain_MB_all()

gain_MB_raw()

gain_PE(*m_AC*)

gain_PE_all()

gain_PE_raw()

gain_total(*m_AC*)

gain_total_all()

gain_total_raw()

linewidth_Hz_all()

linewidth_Hz_raw()

plot_spectra(*freq_min=0.0*, *freq_max=50000000000.0*, *num_interp_pts=3000*, *dB=False*, *dB_peak_amp=10*, *mode_comps=False*, *semilogy=False*, *pdf_png='png'*, *save_txt=False*, *prefix=''*, *suffix=''*, *decorator=None*, *show_gains='All'*, *mark_modes_thresh=0.02*)

set_EM_modes(*mP*, *mS*)

set_allowed_AC(*m_allow*)

set_allowed_EM_Stokes(*m_allow*)

set_allowed_EM_pumps(*m_allow*)

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*, *q_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

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

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

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

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

Args:

sim_EM_pump (`Simulation` object): Contains all info on pump EM modes

sim_EM_Stokes (`Simulation` object): Contains all info on Stokes EM modes

sim_AC (`Simulation` object): Contains all info on AC modes

q_AC (float): Propagation constant of acoustic modes.

Keyword Args:

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:

SBS_gain
[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 = [n_modes_EM_Stokes,n_modes_EM_pump,n_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 = [n_modes_EM_Stokes,n_modes_EM_pump,n_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 = [n_modes_EM_Stokes,n_modes_EM_pump,n_modes_AC].

alpha : The acoustic power loss for each mode in [1/s]. Dimensions = [n_modes_AC].

integration.**gain_python**(*sim_EM_pump*, *sim_EM_Stokes*, *sim_AC*, *q_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.**get_gains_and_qs**(*sim_EM_pump*, *sim_EM_Stokes*, *sim_AC*, *q_AC*, *EM_ival_pump=0*, *EM_ival_Stokes=0*, *AC_ival=0*, *fixed_Q=None*, *typ_select_out=None*)

integration.**grad_u**(*dx*, *dy*, *u_mat*, *q_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*, *q_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.

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

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

7.5 plotting module

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

class plotting.**Decorator**

Bases: `object`

add_frame(*ax*)

add_title(*ax*)

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_frame_drawer(*fd*)

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.

set_title(*t*)

plotting.**delme_plot_one_component_axes**(*ax*, *m_X*, *m_Y*, *l_fields*, *plps*, *cc*)

plotting.**gain_spectra**(*sim_AC*, *SBS_gain*, *SBS_gain_PE*, *SBS_gain_MB*, *linewidth_Hz*, *q_AC*, *EM_ival_pump*, *EM_ival_Stokes*, *AC_ival*, *freq_min=0.0*, *freq_max=50000000000.0*, *num_interp_pts=3000*, *dB=False*, *dB_peak_amp=10*, *mode_comps=False*, *semilogy=False*, *pdf_png='png'*, *save_txt=False*, *prefix=''*, *suffix=''*, *decorator=None*, *show_gains='All'*)

plotting.**get_quiver_skip_range**(*npts*, *skip*)

plotting.**plot_all_components**(*v_x*, *v_y*, *m_X*, *m_Y*, *v_plots*, *plps*, *sim_wguide*, *ival*)

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

plotting.**plot_gain_spectra**(*sim_AC*, *SBS_gain*, *SBS_gain_PE*, *SBS_gain_MB*, *linewidth_Hz*, *EM_ival_pump*, *EM_ival_Stokes*, *AC_ival='All'*, *freq_min=0.0*, *freq_max=50000000000.0*, *num_interp_pts=3000*, *dB=False*, *dB_peak_amp=10*, *mode_comps=False*, *semilogy=False*, *pdf_png='png'*, *save_txt=False*, *prefix=''*, *suffix=''*, *decorator=None*, *show_gains='All'*, *mark_modes_thresh=0.02*)

Construct the SBS gain spectrum, built from Lorentzian peaks of the individual modes.

Args:

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].

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 Args:

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.

pdf_png (str): Save figures as 'png' or 'pdf'.

save_txt (bool): Save spectra data to txt file.

prefix (str): String to be appended to start of file name.

suffix (str): String to be appended to end of file name.

plotting.**plot_mode_data**(*ax*, *v_fields*, *plps*, *sim_wguide*, *ival*, *v_x*, *v_y*)

plotting.**plot_mode_fields**(*sim_wguide*, *ivals=None*, *n_points=501*, *quiver_points=30*, *xlim_min=0*, *xlim_max=0*, *ylim_min=0*, *ylim_max=0*, *EM_AC='EM_E'*, *num_ticks=None*, *colorbar=True*, *contours=False*, *contour_lst=None*, *stress_fields=False*, *pdf_png='png'*, *prefix=''*, *suffix=''*, *ticks=True*, *comps=[]*, *decorator=None*, *suppress_imimre=True*, *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.

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

Keyword Args:
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): Add a string to start of file name

suffix (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**(*mesh_xy*, *prefix=''*, *suffix=''*)

Plot EM mode fields.

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

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

plotting.**plot_one_component**(*m_X*, *m_Y*, *v_fields*, *plps*, *sim_wguide*, *ival*, *cc*, *axis=None*)

plotting.**plot_one_component_axes_contour_and_quiver**(*ax*, *m_X*, *m_Y*, *l_fields*, *plps*, *cc_cont=None*, *cc_quiver=None*)

plotting.**plot_one_component_quiver**(*ax*, *m_X*, *m_Y*, *v_fields*, *plps*, *cc*)

plotting.**plot_set_axes_style**(*ax*, *plps*, *decorator*)

plotting.**plot_set_ticks**(*ax*, *plps*, *decorator*)

plotting.**plot_set_title**(*ax*, *comp_label*, *plps*, *decorator*)

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=''*, *suffix=''*, *ticks=True*, *comps=[]*, *decorator=None*, *suppress_imimre=True*, *modal_gains_PE=None*, *modal_gains_MB=None*, *modal_gains=None*)

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

plotting.**savefig**(*fig*, *fname*)

CHAPTER

EIGHT

THE NUMBAT FEM MODE SOLVERS

8.1 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
```

8.1.1 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;
```

(continues on next page)

(continued from previous page)

```
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};
```

8.1.2 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.

8.1.3 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.

8.2 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

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX