`MPT-Calculator`: A Python-NGSolve Implementation of Magnetic Polarizabiltiy Tensor Accelerated by POD

B. A. Wilson[†], J. Elgy[‡], and P. D. Ledger[‡]
[†]Zienkiewicz Centre for Computational Engineering, College of Engineering, Swansea University
[‡]School of Computing and Mathematics, Keele University
b.a.wilson@swansea.ac.uk, j.elgy@keele.ac.uk, p.d.ledger@keele.ac.uk

October 25, 2022

1 Introduction

The purpose of this document is to provide an overview on how to install and use the `MPT-Calculator`, which is a high order finite element method (FEM) implementation using `NGSolve` [15, 18, 14] for computing the magnetic polarizability tensor for object characterisation in metal detection. In the case of frequency sweeps, this is accelerated by the Proper Orthogonal Decomposition (POD) technique. We begin, in Section 2, with an overview of the underlying mathematical theory describing the eddy current model and forward and inverse problems of metal detection. The formulae for the explicit calculation of the magnetic polarizability description of conducting permeable objects are included in this section along with references to the technical details. The installation of the program is described in Section 3. Then, in Section 4, an overview of the structure of the code is provided. In Section 5 a description of how to create your own geometry file is provided and then in Section 6 a series of examples that can be obtained with the software are included.

2 The eddy-current model and asymptotic expansion

We briefly discuss the eddy-current model along with stating the asymptotic expansion that forms the basis of the magnetic polarizability description of conducting objects in metal detection.

2.1 Eddy-current model

The eddy current model is a low frequency approximation of the Maxwell system that neglects the displacement currents, which is valid when the frequency is small and the conductivity of the body is high. A rigorous justification of the model involves the topology of the conducting body [3]. The eddy current model is described by the system

$$\nabla \times \boldsymbol{E}_\alpha = \mathrm{i}\omega\mu \boldsymbol{H}_\alpha, \tag{1a}$$
$$\nabla \times \boldsymbol{H}_\alpha = \boldsymbol{J}_0 + \sigma \boldsymbol{E}_\alpha. \tag{1b}$$

where $\boldsymbol{E}_\alpha$ and $\boldsymbol{H}_\alpha$ are the electric and magnetic interaction fields, respectively, $\boldsymbol{J}_0$ is an external current source, $\mathrm{i} := \sqrt{-1}$, ω is the angular frequency, μ is the magnetic permeability and σ is the electric conductivity. We will use the eddy current model for describing the forward and inverse problems in the metal detection problem.

2.1.1 Metal Detection Forward Problem

In the forward (or direct) problem, the position and materials of the conducting body B_α are known. The object has a high conductivity, $\sigma = \sigma_*$, and a permeability, $\mu = \mu_*$. The conducting body is assumed to buried in soil, which is assumed to be of a much lower conductivity so that $\sigma \approx 0$ and have a permeability $\mu = \mu_0 := 4\pi \times 10^{-7}$H/m. A background field is generated by a solenodial current source $\boldsymbol{J}_0$ with support in the air above the soil, which has $\sigma = 0$ and $\mu = \mu_0$. The region around the object is $B_\alpha^c := \mathbb{R}^3 \backslash B_\alpha$as shown in Figure 1.

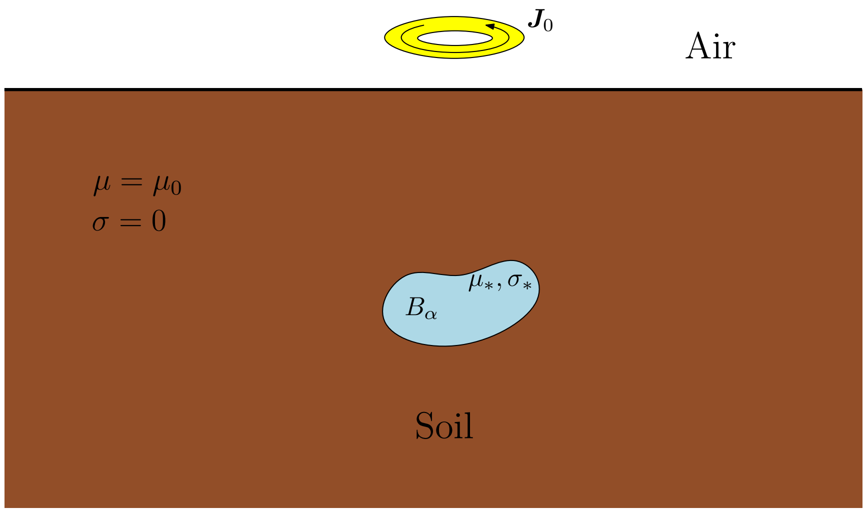


Figure 1: A diagram showing a hidden conducting object B_α surrounded by it's compliment B_α^c which is made up of soil and air.

The forward model is described by the system (1), which hold in $\mathbb{R}^3$, with

$$\mu(\boldsymbol{x}) = \begin{cases} \mu_* & \boldsymbol{x} \in B_\alpha \\ \mu_0 & \boldsymbol{x} \in B_\alpha^c \end{cases}, \sigma(\boldsymbol{x}) = \begin{cases} \sigma_* & \boldsymbol{x} \in B_\alpha \\ 0 & \boldsymbol{x} \in B_\alpha^c \end{cases}, \tag{2}$$

and the regions B_α and B_α^c are coupled by the transmission conditions

$$[\boldsymbol{n} \times \boldsymbol{E}_\alpha]_{\Gamma_\alpha} = [\boldsymbol{n} \times \boldsymbol{H}_\alpha]_{\Gamma_\alpha} = \boldsymbol{0}, \tag{3}$$

which hold on $\Gamma_\alpha := \partial B_\alpha$. In the above, $[u]_{\Gamma_\alpha} := u|_+ - u|_-$ denotes the jump, the $+$ refers to just outside of B_α and the $-$ to just inside and $\boldsymbol{n}$ denotes a unit outward normal to Γ_α.

The electric interaction field in (1) is non-physical and, to ensure uniqueness of this field, the condition $\nabla \cdot \boldsymbol{E}_\alpha = 0$ is imposed in B_α^c. Furthermore, we also require that $\boldsymbol{E}_\alpha = O(1/|\boldsymbol{x}|)$ and $\boldsymbol{H}_\alpha = O(1/|\boldsymbol{x}|)$ as $|\boldsymbol{x}| \to \infty$, denoting that the fields go to zero at least as fast as $1/|\boldsymbol{x}|$, although, in practice, this can faster.

2.1.2 Metal Detection Inverse Problem

In the metal detection inverse problem, one wishes to determine the location, shape and material properties of the conducting object B_α, described above, from measurements of $(\boldsymbol{H}_\alpha - \boldsymbol{H}_0)(\boldsymbol{x})$ at locations $\boldsymbol{x}$ in the air. Here, $\boldsymbol{H}_0$ denotes the background magnetic and is the magnetic field that result from the solution of (1) without the presence of the object B_α, i.e. $\boldsymbol{E}_0$ and $\boldsymbol{H}_0$ are the solution of (1) with $\sigma = 0$ and $\mu = \mu_0$ in $\mathbb{R}^3$. Similar to above, we also require that $\boldsymbol{E}_0 = O(1/|\boldsymbol{x}|)$ and $\boldsymbol{H}_0 = O(1/|\boldsymbol{x}|)$ as $|\boldsymbol{x}| \to \infty$, denoting that the fields go to zero at least as fast as $1/|\boldsymbol{x}|$, although, in practice, this can faster.

Practical metal detectors measure a voltage perturbation, which corresponds to $\int_S \boldsymbol{n} \cdot (\boldsymbol{H}_\alpha - \boldsymbol{H}_0)(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}$ over an appropriate surface S [7]. For very small coils, this voltage perturbation corresponds to $\boldsymbol{m} \cdot (\boldsymbol{H}_\alpha - \boldsymbol{H}_0)(\boldsymbol{x})$ where $\boldsymbol{m}$ is magnetic dipole moment of the coil [7].

2.2 The asymptotic expansion

The forward problem described in Section 2.1.1, implies that if we know B_α ,σ_* and μ_* we can solve (1) to determine $\boldsymbol{E}_\alpha$ and $\boldsymbol{H}_\alpha$. However, to do this repeatably for each new object is computationally expensive. Instead, we seek an approximation in the form of trying approximate the perturbation $(\boldsymbol{H}_\alpha - \boldsymbol{H}_0)(\boldsymbol{x})$ at some point $\boldsymbol{x}$ exterior to B_α.

To do this, following [4, 5] we define $B_\alpha := \alpha B + \boldsymbol{z}$ where B is a unit size object, α is the object size and $\boldsymbol{z}$ is the object's translation from the object as shown in Figure 2.

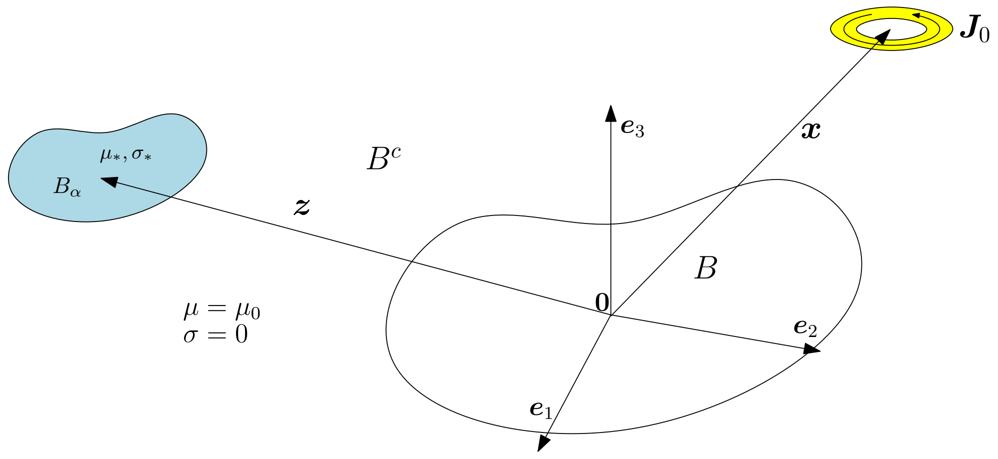


Figure 2: A diagram showing the physical description of B_α with respect to the coordinate axes.

Then, using results obtained by Ammari, Chen, Chen, Garnier and Volkov [4], Ledger and Lionheart [5] have derived the asymptotic expansion

$$(\boldsymbol{H}_\alpha - \boldsymbol{H}_0)(\boldsymbol{x})_i = (\boldsymbol{D}^2_{\boldsymbol{x}} G(\boldsymbol{x},\boldsymbol{z}))_{ij} (\mathcal{M})_{jk} (\boldsymbol{H}_0(\boldsymbol{z}))_k + O(\alpha^4), \tag{4}$$

which holds as $\alpha \to 0$. In the above, $G(\boldsymbol{x},\boldsymbol{z}) = 1/4\pi|\boldsymbol{x}-\boldsymbol{z}|$ is the free space Laplace Green's function, $\boldsymbol{D}^2_x G$ denotes the Hessian of G and Einstein summation convention of the indices is implied. The term $\mathcal{M}$ is the symmetric rank 2 magnetic polarizability tensor, which describes the shape and material properties of the object B_α and is independent of the object's position, but is frequency dependent. We will sometimes write $\mathcal{M}[\alpha B, \omega]$ to emphasise this. The above formulation, and the definition of $\mathcal{M}$ below, are presented for the case of a single homogenous object B, the extension to multiple inhomogeneous objects can be found in [9, 8].
Let us now turn our attention to the computation of the coefficients of $\mathcal{M}$ in (4), which describes the shape and material properties of B_α. From the following, will be able compute a library of such tensors for different choices of αB since $\mathcal{M}$ is independent of $\boldsymbol{z}$, this, in turn, will lend itself to the application of dictionary based classification algorithms [9] for the solution of the inverse problem stated in Section 2.1.2.

2.3 Calculating the Magnetic Polarizability Tensor

In the following, we state the explicit formulae for the computation of the coefficients of $\mathcal{M}$, which have been derived in [8]. Earlier, Ledger and Lionheart [5, 6, 7] have also derived other equivalent formulations, but those below lead to a more natural FEM implementation (using `NGSolve`).

Following [8], we write $\mathcal{M} = (\mathcal{M})_{ij} \boldsymbol{e}_i \otimes \boldsymbol{e}_j$ where $\boldsymbol{e}_i$ denotes the ith orthonormal unit vector and use the splitting $(\mathcal{M})_{ij} := (\mathcal{N}^0)_{ij} + (\mathcal{R})_{ij} + \mathrm{i}(\mathcal{I})_{ij}$ with

$$(\mathcal{N}^0[\alpha B])_{ij} := \alpha^3 \delta_{ij} \int_B (1-\mu_r^{-1}) \mathrm{d}\boldsymbol{\xi} + \frac{\alpha^3}{4} \int_{B\cup B^c} \tilde{\mu}_r^{-1} \nabla \times \tilde{\boldsymbol{\theta}}_i^{(0)} \cdot \nabla \times \tilde{\boldsymbol{\theta}}_j^{(0)} \, \mathrm{d}\boldsymbol{\xi}, \tag{5a}$$

$$(\mathcal{R}[\alpha B, \omega])_{ij} := -\frac{\alpha^3}{4} \int_{B\cup B^c} \tilde{\mu}_r^{-1} \nabla \times \boldsymbol{\theta}_j^{(1)} \cdot \nabla \times \overline{\boldsymbol{\theta}_i^{(1)}} \, \mathrm{d}\boldsymbol{\xi}, \tag{5b}$$

$$(\mathcal{I}[\alpha B, \omega])_{ij} := \frac{\alpha^3}{4} \int_B \nu \Big(\boldsymbol{\theta}_j^{(1)} + (\tilde{\boldsymbol{\theta}}_j^{(0)} + \boldsymbol{e}_j \times \boldsymbol{\xi}) \Big) \cdot \overline{\Big(\boldsymbol{\theta}_i^{(1)} + (\tilde{\boldsymbol{\theta}}_i^{(0)} + \boldsymbol{e}_i \times \boldsymbol{\xi}) \Big)} \, \mathrm{d}\boldsymbol{\xi}. \tag{5c}$$

In the above,

$$\tilde{\mu}_r(\boldsymbol{\xi}) := \begin{cases} \mu_r := \mu_*/\mu_0 & \boldsymbol{\xi} \in B \\ 1 & \boldsymbol{\xi} \in B^c \end{cases},$$

and $\nu := \alpha^2 \omega \mu_0 \sigma_*$, δ_{ij} is the Kronecker delta and the overbar denotes the complex conjugate. The computation of (5) rely on the solution of the transmission problems [8]

$$\nabla \times \tilde{\mu}_r^{-1} \nabla \times \boldsymbol{\theta}_i^{(0)} = \mathbf{0} \qquad \text{in } B \cup B^c, \tag{6a}$$
$$\nabla \cdot \boldsymbol{\theta}_i^{(0)} = 0 \qquad \text{in } B \cup B^c, \tag{6b}$$
$$[\boldsymbol{n} \times \boldsymbol{\theta}_i^{(0)}]_\Gamma = \mathbf{0} \qquad \text{on } \Gamma, \tag{6c}$$
$$[\boldsymbol{n} \times \tilde{\mu}_r^{-1} \nabla \times \boldsymbol{\theta}_i^{(0)}]_\Gamma = \mathbf{0} \qquad \text{on } \Gamma, \tag{6d}$$
$$\boldsymbol{\theta}_i^{(0)} - \boldsymbol{e}_i \times \boldsymbol{\xi} = \boldsymbol{O}(|\boldsymbol{\xi}|^{-1}) \qquad \text{as } |\boldsymbol{\xi}| \to \infty, \tag{6e}$$

and

$$\nabla \times \mu_r^{-1} \nabla \times \boldsymbol{\theta}_i^{(1)} - \mathrm{i}\nu(\boldsymbol{\theta}_i^{(0)} + \boldsymbol{\theta}_i^{(1)}) = \mathbf{0} \qquad \text{in } B, \tag{7a}$$
$$\nabla \times \nabla \times \boldsymbol{\theta}_i^{(1)} = \mathbf{0} \qquad \text{in } B^c, \tag{7b}$$
$$\nabla \cdot \boldsymbol{\theta}_i^{(1)} = 0 \qquad \text{in } B^c, \tag{7c}$$
$$[\boldsymbol{n} \times \boldsymbol{\theta}_i^{(1)}]_\Gamma = \mathbf{0} \qquad \text{on } \Gamma, \tag{7d}$$
$$[\boldsymbol{n} \times \tilde{\mu}_r^{-1} \nabla \times \boldsymbol{\theta}_i^{(1)}]_\Gamma = \mathbf{0} \qquad \text{on } \Gamma, \tag{7e}$$
$$\boldsymbol{\theta}_i^{(1)}(\boldsymbol{\xi}) = \boldsymbol{O}(|\boldsymbol{\xi}|^{-1}) \qquad \text{as } |\boldsymbol{\xi}| \to \infty. \tag{7f}$$

Note also that $\tilde{\boldsymbol{\theta}}_i^{(0)} := \boldsymbol{\theta}_i^{(0)} - \hat{\boldsymbol{e}}_i \times \boldsymbol{\xi}$. In order to obtain a discrete approximation using the FEM, we need to introduce a finite computational domain Ω such that $B \subset \Omega$, whose boundary $\partial\Omega$ is placed sufficiently far from the object B.

The tensor $\mathcal{N}^0[\alpha B]$ describes the magnetostatic characterisation of αB and is independent of ω. The frequency dependent $\mathcal{R}[\alpha B, \omega]$ tensor vanishes at low frequency and $\mathcal{N}^0[\alpha B] + \mathcal{R}[\alpha B, \omega] = \mathrm{Re}(\mathcal{M}[\alpha B, \omega])$ describes the frequency behaviour of the real part of the tensor. Similarly, $\mathcal{I}[\alpha B, \omega] = \mathrm{Im}(\mathcal{M}[\alpha B, \omega])$ describes the frequency behaviour of the imaginary part of the tensor, which vanishes at low frequencies and tends to 0 as the upper limiting frequency of the eddy current model is reached [8].

In order to compute the tensor coefficients in (5) we need to solve (6) and (7) (repeatably for each ω) and this will be achieved by computing approximate solutions using a range of different numerical schemes

1. A *hp* FEM discretisation of the transmission problems (6) and (7) using `NGSolve` [15, 18, 14] to compute $\mathcal{M}[\alpha B, \omega]$ for a single frequency.

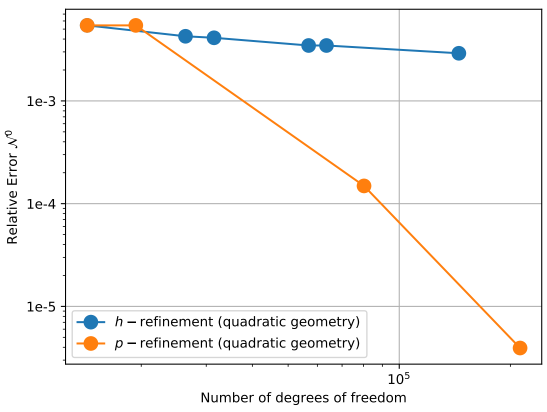
2. A *hp* FEM discretisation of the transmission problems (6) and (7) using `NGSolve` [15, 18, 14] for performing the computation of $\mathcal{M}[\alpha B, \omega]$ over a range of frequencies.

3. A Proper Orthogonal Decomposition (POD) reduced order model, which greatly accelerates the computation of the full order model in 2. for computing $\mathcal{M}[\alpha B, \omega]$ over a range of frequencies.

4. Certificates computed at run time, which indicate the accuracy of the POD outputs with respect to the full order solution over a range of frequencies.

A detailed description of the numerical implementation of these schemes can be found in [17].

In particular, we advocate a *hp* FEM discretisation using $\boldsymbol{H}(\mathrm{curl})$ conforming elements for the transmission problems (6) and (7) due to the superior performance it overs over a traditional h-version of the FEM, in which only the mesh is refined. In the *hp*-version, the polynomial order of the elements can be increased, as well as refining the finite element grid, in order to obtain accurate solutions. In practice, it is often sufficient to generate a suitable mesh, which has local refinement around sharp edges and corners and increase the polynomial degree in order to obtain accurate solutions for the magnetic polarizability tensor coefficients. As an illustration, we present in Figure 3 convergence curves of $\|\mathcal{N}^0_{hp} - \mathcal{N}^0\|_F / \|\mathcal{N}^0\|_F$ and $\|\mathcal{M}_{hp} - \mathcal{M}\|_F / \|\mathcal{M}\|_F$ computed for a conducting sphere of radius $\alpha = 0.01$m, $\sigma_* = 6 \times 10^6$S/m, $\mu_* = 1.5\mu_0$, which has an analytical solution [16], where *hp* denotes the tensor coefficients obtained by a *hp* FEM discretisation. The solutions obtained on a sequence of meshes with 12074, 22392, 26751, 48418, 54092, 123788 unstructured tetrahedral elements of order $p = 0$ (h-refinement) are compared with those obtained on a fixed mesh of 12074 unstructured tetrahedral elements using of order $p = 0, 1, 2, 3$, in turn (p-refinement). We observe the downward sloping behaviour of the p-refinement convergence curve, which illustrates that exponential convergence is being obtained, compared to the slower algebraic rate with h-refinement.



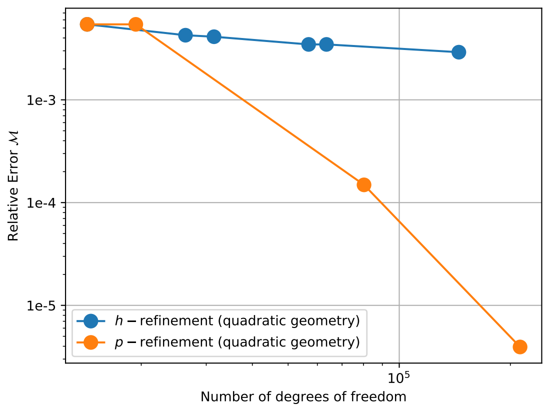


Figure 3: Conducting sphere of radius $\alpha = 0.01$m, $\sigma_* = 6\times10^6$S/m, $\mu_* = 1.5\mu_0$: Convergence curves of $\|\mathcal{N}^0_{hp} - \mathcal{N}^0\|_F/\|\mathcal{N}^0\|_F$ and $\|\mathcal{M}_{hp} - \mathcal{M}\|_F/\|\mathcal{M}\|_F$ for h and p refinement

The following describes the installation and use `MPT-Calculator`, which implements the above schemes.

3 Installation

Due to the code being written in python and using `NGSolve` the user is required to install both of these in order to use the `MPT-Calculator`. Please follow the instructions available at `https://ngsolve.org/` in order to install the high order finite element and meshing library `NGSolve` [15, 18, 14] released under the LPLG license and ensure a compatible version of Python 3 is installed which is available at `https://www.python.org`. Note that the compatible versions of Python 3 and NGSolve are different on Linux and Mac OS and one should check the NGSolve webpage for up to date information. The code is compatible with `NGSolve` 6.2.1907 and after. The examples in section 6 have been run using version 6.2.2004 and version 3.8.2 of `Python 3` [2] and meshes for this have also been provided for users runnings different version of NGSolve. The code has been tested on version 10.14.6 of `MAC OS` and 22.04 of `Ubuntu`.

Along with these installations, the `MPT-Calculator` relies on a number of python packages, which the user is required to install they are as follows `sys`, `numpy`, `os`, `time`, `multiprocessing_on_dill`, `cmath`, `subprocess`, `matplotlib`. On a MAC or Linux they can be installed from the command line using the command

```
pip3 install "package to be installed"
```

where `"package to be installed"` is replaced with the appropriate package name. The user is then required to download or clone the repository of this `MPT-Calculator` from github. Finally users on Ubuntu are required to enter the following lines into their `.bashrc` file,

```
export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1
export MKL_THREADING_LAYER=sequential
```

This is due to the code calling multiple instances of `NGSolve` in multiprocessing mode.

3.1 Jupyter Notebook Support

The current version of `NGSolve` and `Netgen` offers support for `Jupyter Notebooks` and offers support for web based visualisation. Currently, to install `Jupyter Notebooks` on MAC or Linux systems enter

```
pip3 install jupyter
```

into the terminal. On Windows systems enter

```
pip install jupyter
```

into the command prompt. To further enable the `NGSolve` visualisation tools, the user is required to also install `webgui_jupyter_widgets` and `widgetsnbextension`. Similarly to installing the software via `pip`, we enter

```
pip3 install webgui_jupyter_widgets
jupyter nbextensions install --user --py widgetnbextension
jupyter nbextension enable --user --py widgetnbextension
jupyter nbextension install --user --py webgui_jupyter_widgets
jupyter nbextension enable --user --py webgui_jupyter_widgets
```

into the command prompt or terminal. Further information can be found on the `NGSolve` website, here.

4 Overview and structure of the code

Let us now discuss the layout of the code and the files which are designed to be edited. The user is expected interact with 3 files `main.py`, `Settings.py` and `PlotterSettings.py`, these files along with a geometry file (`.geo` file or `OCC .py` file) (see Section 5) allow the user to produce an array of different frequency sweeps for many different objects. In this section we discuss the layout of the folder system in place, how each of the input files can used and edited by the user to produce a frequency sweep along with how and where the results are saved. The structure of the code with respect to the project root directory can be seen in the folder tree below where examples have been included for `.geo` and `OCC` geometry descriptions and the results folder layout is illustrated by means of a sphere example.

```
├── Documentation
│   └── PythonCodeDocumentation.pdf
├── Functions
│   ├── Helper_Functions
│   │   ├── count_prismatic_elements.py
│   │   ├── exact_sphere.py
│   │   └── step_to_vol_mesher.py
│   ├── Checkvalid.py
│   ├── FullSolvers.py
│   ├── MeshCreation.py
│   ├── ML_MPT_Predictor.py
│   ├── MPTFunctions.py
│   ├── MultiPermeability.py
│   ├── PlotEditor.py
│   ├── PlotEditorWithErrorBars.py
│   ├── Plotters.py
│   ├── PODFunctions.py
│   ├── PODPlotEditor.py
│   ├── PODPlotEditorWithErrorBars.py
│   ├── PODSolvers.py
│   ├── ResultsFunctions.py
│   ├── SingleSolve.py
├── GeoFiles
│   ├── Claw_wodden_handle.geo
│   ├── Coin.geo
│   ├── Cylinder.geo
│   ├── DualBar.geo
│   ├── dualBox.geo
│   └── ...
├── OCC_Geometry
│   ├── OCC_bottle.py
│   ├── OCC_cylinder.py
│   └── ...
├── Results
│   └── sphere
│       └── al_0.001_mu_1_sig_1e6
│           └── 1e1-1e10_40_el_57698_ord_2
│               ├── Data
│               │   ├── Eigenvalues.csv
│               │   ├── Frequencies.csv
│               │   ├── N0.csv
│               │   └── Tensors.csv
│               ├── Functions
│               │   └── Plotters.py
│               ├── Graphs
│               │   ├── ImaginaryEigenvalues.pdf
│               │   ├── ImaginaryTensorCoeficients.pdf
│               │   ├── RealEigenvalues.pdf
│               │   └── RealTensorCoeficients.pdf
│               ├── Input_files
│               │   ├── main.py
│               │   ├── Settings.py
│               │   ├── sphere.geo
│               │   └── sphere.zip
│               ├── PlotEditor.py
│               └── PlotterSettings.py
├── Settings
│   ├── PlotterSettings.py
│   └── Settings.py
├── VolFiles
│   ├── Claw_wodden_handle.vol
│   ├── Knife_Cheap_Chef.vol
│   ├── OCC_cylinder.vol
│   └── ...
├── Results_2d
├── Changelog_for_MPT-Calculator
├── LICENSE
├── main_2d.py
├── README.md
└── main.py
```

4.1 User input files

The files (along with their file paths) the user is expected to interact with are

```
main.py
Settings/Settings.py
Settings/PlotterSettings.py.
```

Once the files have the desired inputs the simulation is run using the default parameters by entering the command

```
python3 main.py
```

possibly replacing `python3` with `python3.8` here and throughout depending on your setup and version installed to the command line from the main directory. The `MPT-Calculator` then runs the default frequency sweep and saves the outputs in an output folder (see Section 4.2). The following sections cover the Python interface with `MPT-Calculator`. We start this explanation with `main.py`.

4.1.1 `main.py`

The file `main.py` is the file with which the user is expected to have most interaction, it contains the starting function for `MPT-Calculator` which calls functions to generate a mesh, perform frequency sweeps and finally post process the data produced in the frequency sweep. Inside `main.py` the user will find the function

```
main(h='coarse', order=2, curve_degree=5, start_stop=(), alpha='',
           geometry='default', frequency_array='default',
           use_OCC=False, use_POD=False, use_parallel=True)
```

To run a simulation, the user is required to import and call `main`. For example

```
from main import main
main()
```

In this section we shall briefly explain what each of these optional input arguments refer to and how to provide an input for each. Note that there is no ordering requirement for passing keyword arguments to a function in Python nor does every argument need to be provided. As such calling

```
main(order=2, geometry='sphere.geo')
```

is equivalent to

```
main(geometry='sphere.geo', order=2)
```

For the first argument, `h`, the user can provide a (string) flag for the `Netgen` mesher to control the density of the mesh. The available options are `'verycoarse'`, `'coarse'`, `'moderate'`, `'fine'`, and `'veryfine'` that take into account the geometry of the object under test. Alternatively, the user can provide a specific numerical value, e.g. 0.5, that will be used to produce a quasi-uniform mesh over the entire computational domain. Note that supplying a numerical value will often result in a fine discretisation outside of the object.

```
h='coarse'
```

Another key option when it comes to defining a mesh is the order of the elements (how complex the functions describing the solution are) this requires an integer input which is greater that 0 (0 being the fastest to run but least accurate and higher orders being slower but more accurate) when running full simulations (not small tests) it is recommended that an order of 3 be used, this can be defined as follows

```
order=3
```

Finally, for curved geometries, e.g. a sphere, `NGSolve` approximates the true geometry of the object via curved edge elements. `NGSolve` calculates the curvature of the edge elements via fitting an n^{th} order polynomial to the true geometry, with higher order providing a more accurate representation. The user has control of the order of polynomial via the `curve_degree` argument, which can be set by

```
curve_degree=5
```

For practical applications, `curve_degree=5` is sufficient.

For control over the true geometry, we can control 3 arguments, `geometry`, `alpha`, and `use_OCC`. For the first input, `geometry`, the user is enabled to define a `.geo` or `.py` (geometry) file which is to be used in the sweep (see Section 5 for information on how to create geometry files) this input is a string and can be defined as follows

```
geometry='sphere.geo'
```

or

```
geometry='OCC_sphere.py'
```

If using an `OCC` description, then `use_OCC` must be `True`. Finally, we can control the scaling of the unit sized objects in the `.geo` or `.py` file via the parameter α (meters), the input for this is a float and can be defined as follows

```
alpha=0.001
```

This then defines how the object in the geometry file should be scaled. In this example a scaling of $\alpha = 1 \times 10^{-3}$ m is shown.

The next 3 arguments allow the user to control the frequencies [rad/s] used in the calculation of the frequency sweep. First, to control the logarithmically spaced frequency samples, the user can provide the argument

```
start_stop=(Start, Finish, Points)
```

as a tuple, which sets the smallest, largest, and number of samples used in the construction of the logarithmically spaced frequency sweep. n.b. these are the powers base 10 of the frequencies i.e. we create a frequency sweep for $10^{\texttt{Start}} - 10^{\texttt{Finish}}$ rad/s. For example if we passed the following as an argument to the main function

```
start_stop=(1, 8, 8)
```

We would run a frequency sweep for 8 logarithmically spaced points between 10^1 and 10^8 i.e. for the following frequencies

$$\omega = 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8 \text{ rad/s}.$$

Alternatively, the user can specify exact frequencies at which to evaluate the MPT. This is done via the `Frequency_Array` argument. This is passed as a list of frequencies, in rad/s, on a linear scale. For example

```
frequency_array=[10,15,300,53245,10**7]
```

will calculate the MPT at $10, 15, 300, 53\,245$, and 10^7 rad/s. A single frequency can be obtained by specifying at as the only item in the list.

Finally the user will find some options which affect how the frequency sweep will be produced. The first of these is whether or not to create a reduced order model (ROM) using the method of proper orthogonal decomposition (POD) the user may define whether to use this by editing the variable `use_POD` (boolean), if the user would like the frequency sweep to be produced using POD they set

```
use_POD=True
```

The POD works by producing a frequency sweep for a small number of frequencies (snapshots), it then uses the solutions to create an ROM which can then be used to produce a full frequency sweep containing many more points at a fraction of the computational cost. It is useful to note at this point that although POD works very well as the relative permeability μ_r of the object being simulated increases the POD becomes less accurate, see the spheroid example. This brings us to our final user input in the `main` function which is the variable `use_parallel` this option decides whether to produce the frequency sweep using multiple cores, by setting

```
use_parallel=True
```

The wall clock time will be reduced but the frequency sweep then requires more of the machines resources since it is running simulations on multiple cores. The default setting for the number of cores to be used is 4 but can be edited in the default settings section of `Settings.py`.

4.1.2 Return Values

The `main` function additionally returns variables to the Python console. These outputs are stored as a Python dictionary where each dictionary key corresponds to a specific output. By default, these outputs are:

- `TensorArray` is a complex $N \times 9$ array containing each of the tensor coefficients for the 3×3 rank 2 MPT for each frequency of interest. This is stored in a row-major format as (row 1, row 2, row 3).
- `EigenValues` is a $N \times 3$ array containing the eigenvalues for each frequency. Eigenvalues for `MPT-Calculator` are stored in ascending order for each frequency of interest with $\lambda_i(\mathcal{R} + \mathcal{N}^0)$ and $\lambda_i(\mathcal{I})$ treated independently.
- `N0` is the real 3×3 $\mathcal{N}^0$ coefficients.
- `NElements` is the integer number of total elements in the mesh.
- `FrequencyArray` is the $N \times 1$ array containing the frequencies (rad/s) used for the sweep.
- `NDOF` is a tuple containing the number of degrees of freedom used for the $\boldsymbol{\theta}^{(0)}$ and $\boldsymbol{\theta}^{(1)}$ finite element spaces. This is stored as `(NDOF(`$\boldsymbol{\theta}^{(0)}$`), NDOF(`$\boldsymbol{\theta}^{(1)}$`))`.

As an example, running

```
ReturnDict = main()
ReturnDict['N0']
```

will return the real 3×3 array of $\mathcal{N}^0$ tensor coefficients for the default settings of `MPT-Calculator`.

In addition optional return arguments are added to the dictionary depending on the settings used. `EddyCurrentTest` will contain an estimated max frequency where the eddy current model holds if `EddyCurentTest` is set to True in `Settings.py`.

If the `use_POD` option is used then additional outputs corresponding to the POD snapshot solutions are also returned. `PODTensorArray`, `PODEigenValues`, and `PODFrequencyArray` contain the POD snapshot equivalents for `TensorArray`, `EigenValues`, and `FrequencyArray`. Finally, the $N \times 6$ array of POD error certificates is stored as `PODErrorBars` if `PODErrorBars=True` in `Settings.py`. Note that since the rank 2 `MPT` is symmetric we only store the diagonal and lower triangular coefficients. We store the error certificates, e, as $[e_{11}, e_{22}, e_{33}, e_{12}, e_{13}, e_{23}]$.

4.1.3 `Settings/Settings.py`

The second section which the users is able to interact with is the `Settings.py` file. This file contains settings related to how the frequency sweep is to be run, any addition outputs the user would like, how the data should be saved and how `NGSolve` solves each of the finite element problems. The file is subdivided into the following sections, `DefaultSettings`, `AdditionalOutputs`, `SaverSettings` and `SolverParameters`.

We start by discussing the inputs in `DefaultSettings`, the definition of which is printed in the code snippet Listing 1.

```
def DefaultSettings():
    #How many cores to be used (monitor memory consumption)
    CPUs = 4
    #(int)

    #Is it a big problem (more memory efficiency but slower)
    BigProblem = False
    #(boolean)

    #How many snapshots should be taken
    PODPoints = 13
    #(int)

    #Tolerance to be used in the TSVD
    PODTol = 10**-6
    #(float)

    #Use an old mesh
    OldMesh = False
    #(boolean) Note that this still requires the relevant .geo file to
   obtain
    #information about the materials in the mesh

    return CPUs,BigProblem,PODPoints,PODTol,OldMesh
```

Listing 1: DefaultSettings Definition

This section defines variables relating about how to carry out the simulation. The number of cores is set by editing the variable `CPUs` (integer) an example of this would be setting

```
CPUs = 4
```

which would produce a frequency sweep using 4 of the machines cores. It is useful to mention at this point that when using multiple cores the user is recommended to monitor memory usage especially when producing a frequency sweep for an object with a fine mesh on a machine with limited resources. For larger problems, which are more memory intensive, we have an option for the simulation to use less memory (this option is for POD sweeps only). This is more memory efficient, however, it is both slower and effects performance of both the POD output and POD error bars. This option is engaged by editing the variable `BigProblem` (boolean), setting

```
BigProblem=True
```

This works by reducing the accuracy and therefore size of the snapshot solutions, saving each complex coefficient using the data type `np.complex64` (32-bit floats for both the real and imaginary parts). Although the use of this varies for different machines, for reference, using a 2015 mac with 8GB of ram it is advisable to set `BigProblem = True` for problem larger than 50,000 elements with $p = 3$ with 13 snapshots or greater. The next two variables relate to the use of POD the first is `PODPoints` (integer), this defines how many snapshots should be taken when using the POD method, and can be set as

```
PODPoints = 13
```

This then creates an ROM using 13 snapshots (if `POD = True` in `main.py`). The other setting in the the `DefaultSettings` section is the variable `PODTol` (float) this variable sets at which point to truncate the singular value decomposition setting

```
PODTol = 10**-4
```

sets the truncation to the point at which the singular values drop below the 10^{-4} (for more explanation see [17]). Suggested values for `PODPoints` and `PODTol` are 13 and 10^{-4} respectively. Lastly the setting `OldMesh` can be used to produce a frequency sweep with a precomputed mesh, saved in a `.vol` file. Since different versions of NGSolve produce different meshes this option allows you to run new examples using meshes produced on older versions of NGSolve. This is done by specifying the `.geo` file which contains the material information about the relevant mesh e.g. `geometry='myshape.geo'`, as well as supplying the `.vol` file e.g. `myshape.vol` to the `VolFiles` folder (this requires the `.geo` and `.vol` files to have the same name). Then by setting

```
OldMesh = True
```

the “old mesh” is used rather than producing a new mesh.

Next we will discuss the section `AdditionalOutputs`, which defines the Additional outputs that the code returns. The definition of `AdditionalOutputs`, including the default variables is shown in Listing 2.

```python
def AdditionalOutputs():
    #Plot the POD points
    PlotPod = True
    #(boolean) do you want to plot the snapshots (This requires
   additional
    #calculations and will slow down sweep by around 2% for default
   settings)

    #Produce certificate bounds for POD outputs
    PODErrorBars = True
    #(boolean)

    #Test where the eddy-current model breaks for the object
    EddyCurrentTest = False
    #(boolean)

    #Produce a vtk outputfile for the eddy-currents (outputs a large file
   !)
    vtk_output = False
    #(boolean) do you want to produce a vtk file of the eddy currents in
   the
    #object (single frequency only)

    #Refine the vtk output (extremely large file!)
    Refine_vtk = False
    #(boolean) do you want ngsolve to refine the solution before
   exporting
    #to the vtk file (single frequency only)
    #(not compatable with all NGSolve versions)

    return PlotPod, PODErrorBars, EddyCurrentTest, vtk_output, Refine_vtk
```

Listing 2: AdditionalOutputs definition

The first variable `PlotPod` (boolean) defines whether or not to calculate and plot points where the POD has taken snapshots. These points can be plotted by setting

```
PlotPod = True
```

Having these points plotted can be very useful as it gives a visual representation of the accuracy of the frequency sweep produced by POD compared with a full order frequency sweep. We also give the user a way of producing the error certificates by editing the the variable `PODErrorBars` (boolean) which is selected by setting

```
PODErrorBars = True
```

This then produces error certificates for the POD outputs. The next option `EddyCurrentTest` (boolean) is to calculate the frequency at which the eddy current model is estimated to break down for the selected object, which may be lower than the prescribed maximum frequency and, for complex objects, may be a considerable restriction. Note this is based on a computation using the object’s geometry rather than the standard engineering rules of thumb of saying the object is small compared to the wavelength and the conductivity is high $\epsilon\omega < \sigma_*$ [12]. This is selected by setting

```
EddyCurrentTest=True
```

These three outputs can be produced and saved then chosen to be shown or hidden after the sweep is run. The next two option are for the single frequency solve, with the solution of the single frequency problem the user may export the solution from NGSolve so that it can later be used by programs such as Paraview. This is done by editing the variables `vtk_output` (boolean) and `Refine_vtk` (boolean), by setting

```
vtk_output=True
```

a `.vtk` file will be produced in the `vtk_output` section of the `Results` folder, which contains the eddy-currents in the object. The user may also refine the solution before exporting by setting

```
Refine_vtk=True
```

It is useful to take note at this point of the size of the files produced by these options an example is of a 2605 element mesh which when exported with no refinement produced a `.vtk` file of size 956 KB and with refinement produces a `.vtk` file of size 81.9 MB.

We next discuss the section `SaverSettings`, this section allows the user to define a file path within the folder Results to save the outputs of the frequency sweep. A copy of the section can be seen in Listing 3.

```
def SaverSettings():
    #Place to save the results to
    FolderName = "Default"
    #(string) This defines the folder (and potentially subfolders) the
    #data will be saved in (if "Default" then a predetermined the data
    #will be saved in a predetermined folder structure)
    #Example input "MyShape/MyFrequencySweep"

    return FolderName
```

Listing 3: SaverSettings Definition

This is done by changing the variable `FolderName` (string) to the desired filepath within which save the outputs. An example of this would be

```
FolderName = 'MyShape/MyFrequencySweep'
```

which would then store the output in `Results/MyShape/MyFrequencySweep`. Although this is an option, we recommend that `FolderName = "Default"`, which, when set, produces a series of subfolders containing information relating to material properties scaling and frequencies in the sweep. This makes it very hard for the user to overwrite (and lose) existing data from other frequency sweeps, which will be done if the user forgets to change `FolderName` for a new frequency sweep.

The final section in `Settings.py` relates to how `NGSolve` solves each of the finite element problems. The variables of this can be seen in the code Listing 4.

```
def SolverParameters():
    #Parameters associated with solving the problem can edit this
    #preconditioner to be used
    Solver = "bddc"
    #(string) "bddc"/"local"

    #regularisation
    epsi = 10**-10
    #(float) regularisation to be used in the problem

    #Maximum iterations to be used in solving the problem
    Maxsteps = 2500
    #(int) maximum number of iterations to be used in solving the problem
    #the bddc will converge in most cases in less than 200 iterations
    #the local will take more

    #Relative tolerance
    Tolerance = 10**-8
    #(float) the amount the redsidual must decrease by relatively to
   solve
    #the problem

    #print convergence of the problem
    ngsglobals.msg_level = 0
    #(int) Do you want information about the solving of the problems
    #Suggested inputs
    #0 for no information, 3 for information of convergence
    #Other useful options 1,6
    return Solver,epsi,Maxsteps,Tolerance
```

Listing 4: SolverParameters Definition

The default options work well for the frequency sweeps that are presented in the examples section. The default settings are as follows

```
Solver = 'bddc'
epsi = 10**-10
Maxsteps = 1500
Tolerance = 10**-8
ngsglobals.msg_level = 0
```

These settings are optimised for meshes containing more than 20,000, 3rd order elements, see the examples section for further details. The options for the settings is as follows: The variable `Solver` (string) can be set to either `"bddc"` or `"local"` this changes the preconditioner used by the iterative CG solver in `NGSolve`. For simulations using very coarse meshes or simulations with 0th order elements the `"local"` preconditioner is faster but for larger simulations setting

```
Solver = "bddc"
```

is recommended. The variable `tolerance` (float) sets the required relative residual for the iterative CG solver and controls the accuracy of the linear solve. For coarse discretisations, this does not need to be too small. For example, for full order sweeps, setting

```
Tolerance = 10**-6
```

often leads to satisfactory results, however, upon testing, we found that for the POD the solution needs to be calculated more accurately to capture the underlying behaviour of the solution. The variables `epsi` (float) (regularisation) and `Maxsteps` (integer) (the maximum number of iterations performed) are less important since provided that `epsi` is 1 order of magnitude smaller than `tolerance` a solution will be reached within the maximum number of iterations. Finally, the variable `ngsglobals.msg_level` (integer) relates to the information provided by `NGSolve` about solving the problems. We recommend that this be set to either `0` (for no information) or `3` (for information relating to the assembly and solving of the finite element problems). The next section explains how to use the `PlotterSettings.py` file.

4.1.4 `Settings/PlotterSettings.py`

To display the results of the frequency sweep a simple plotting function has been created[1], the settings of this function can be edited by changing the inputs of the `PlotterSettings.py` file. The `PlotterSettings` function within the `PlotterSettings.py` file can be seen in Listing ??.

,

```
def PlotterSettings():

    #Line settings
    EigsToPlot = [1,2,3]
    #(list) Which Eigenvalues should be plotted smallest to largest (this
    is
    #used for both the main lines and snapshots)
    TensorsToPlot = [1,4,6,2,3,5]
    #(list) Which Tensor coefficients to plot leading diagonals are
   [1,4,6]
    #and tensor layout can be seen below (this is used for both the main
    #lines and the snapshots)
    #
    #              (1,2,3)
    # Tensor ref =(_,4,5)
    #              (_,_,6)

    #Line styles
    MainLineStyle = "-"
    #(string) Linestyle of the eigenvalue/Tensor plots (string, see
    #matplotlib for availible linestyles)
    MainMarkerSize = 4
    #(float) markersize of eigenvalue/Tensor plots (if applicable
   linestyle
    #is chosen)

    #Snapshot styles
    SnapshotLineStyle = "x"
    #(string) Linestyle of snapshots (if plotted)
    SnapshotMarkerSize = 8
    #(float) markersize of snapshots (if plotted)

    #ErrorBars
    ErrorBarLineStyle = "--"
    #(string) Linestyle of the error bars (string, see matplotlib for
    #availible linestyles)
    ErrorBarMarkerSize = 4
    #(float) markersize of the error bars (if applicable linestyle is
   chosen)

    #Eddy-current model breakdown
    EddyCurrentLine = True
    #(boolean) display where the eddy-current model breaks down (if value
    #has been calculated)

    #Title
    Title = False
    #(boolean)

    #Display graph?
    Show = False
```

[1]This is a basic visualisation tool, which allows the user to view results of the sweep. For a graph tailored to the users specification, use the data stored in the `.csv` files saved in the `Data` folder to produce their own plot.

```
    #(boolean) if false then graph is only saved


    return Title, Show, EigsToPlot, TensorsToPlot, MainLineStyle,\
     MainMarkerSize, SnapshotLineStyle, SnapshotMarkerSize,\
     ErrorBarLineStyle, ErrorBarMarkerSize, EddyCurrentLine
```

Listing 5: PlotterSettings Definition

The first two inputs of the file `EigsToPlot` (list) and `TensorsToPlot` (list) relate to which lines are to be plotted on the graphs. With these options the user may change both the lines and the order in which the lines are to be plotted. The variable `EigsToPlot` defines which eigenvalues, sorted smallest to largest, are plotted. For example, setting

```
EigsToPlot = [3,2,1]
```

will plot all of the eigenvalues largest to smallest. The variable `TensorsToPlot` allows the user to choose which coefficients of $\mathcal{M}$ are to be plotted. Taking account of the symmetry of the tensor, the user has a choice of 6 independent coefficients to plot as referenced by the numbers in the following matrix

$$\text{reference matrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}. \tag{8}$$

For example, we may plot just diagonal coefficients of the matrix by setting

```
TensorsToPlot = [1,4,6]
```

For simplicity, a legend is created automatically for each of the lines in the plot, without additional inputs required from the user. The next 6 variables relate the style of lines being plotted. The variables `MainLineStyle` (string), `SnapshotLineStyle` (string) and `ErrorBarLineStyle` (string) define the line styles of the full frequency sweep and the line styles of the snapshots and certificate bounds (if `PlotPod = True` and/ or `PODErrorBars = True`), respectively. The plots are created using the matplotlib module of python and, thus, the available line styles are those supported by matplotlib [1]. The other variables relating to line style are `MainMarkerSize` (float), `SnapshotMarkerSize` (float) and `ErrorBarMarkerSize` (float), which define the size of the markers used in the plot for the full sweep, snapshots and certificate bounds, respectively. The recommended settings of these vary on the markers chosen, but the default settings for the line styles and markers are

```
MainLineStyle = '-'
MainMarkerSize = 4
SnapshotLineStyle = 'x'
SnapshotMarkerSize = 8
ErrorBarLineStyle = '--'
ErrorBarMarkerSize = 4
```

The next variable, `EddyCurrentLine` (boolean), defines whether or not to plot the point at which the eddy-current model breaks down. This can be enabled by setting

```
EddyCurrentLine = True
```

The next variable, `Title` (boolean), defines whether or not the plot includes a title and, again, the title is automatically created for each of the different plots with no user input required. The default setting is

```
Title = True
```

The final variable is `Show` (boolean), which defines whether or not to display the produced graphs once the frequency sweep is complete[2]. The default setting is

```
Show = True
```

With this we conclude the section on how to interact with each of the input files. We next discuss how and where the output of the code is saved.

[2]The graphs are automatically saved this option is whether to display them as well.

4.2 Output files

In this section, we discuss how the outputs for each of the different versions of the code are saved. As mentioned briefly in Section 4.1.3, the user may define which folder (or filepath) the outputs are saved in. However, when the code is run, it will generate subfolders to organise the results below this folder (or filepath), as described below. We start with the case of a single frequency simulation.

4.2.1 Single frequency output

In the case of a simulation with only one frequency, there is no need for a graphical representation, due to this the output folder has the structure shown below

```
OutputFolder
    ├── Data
    │   ├── Eddy_current-breakdown.txt
    │   ├── Eigenvalues.csv
    │   ├── MPT.csv
    │   └── N0.csv
    └── Input_files
        ├── Settings.py
        ├── main.py
        ├── geometry.geo
        └── geometry.zip
```

Notice the inclusion of the folder `Input_files`, this folder contains a copy of the inputs files used to run the simulation. This allows the user to reproduce their results, if they so wish. We note the inclusion of the `sphere.zip` file which is a compressed version of the mesh used for the simulation (`sphere.vol` file).The folder `Data`, contains the results for the computed $\mathcal{M}$, $\mathcal{N}^0$, stored in `MPT.csv` and `N0.csv`, respectively, and their eigenvalues $\lambda_i(\mathcal{N}^0 + \mathcal{R})$ and $\lambda_i(\mathcal{I})$, stored in the file `Eigenvalues.csv` as the sum $\lambda_i(\mathcal{N}^0+\mathcal{R})+\mathrm{i}\lambda_i(\mathcal{I})$ to reduce the number of output files. If the option `EddyCurrentTest = True` then we also have the `Eddy-current_breakdown.txt` file, which states the frequency in rad/s at which the eddy current model breaks down.

If the variable `vtk_output = True`, we have the added output in the `vtk_output` folder of the results folder which can be seen in below for the example of a sphere evaluated at the single frequency of $\omega = 133$ rad/s.

```
vtk_output
    └── sphere
        └── om_1.33e2
            ├── sphere.geo
            └── sphere.vtk
```

Here we store the `.vtk` file along with the associated `.geo` file which contains information about the materials used to construct the object. Let us next discuss the case of a full frequency sweep without using POD.

4.2.2 Full order frequency sweep output

In the case of the full order frequency sweep, the folder structure for the output folder can be seen in the tree below.

```
OutputFolder
    ├── Data
    │   ├── Eddy_current-breakdown.txt
    │   ├── Eigenvalues.csv
    │   ├── Frequencies.csv
    │   ├── N0.csv
    │   └── Tensors.csv
    ├── Functions
    │   └── Plotters.py
    ├── Graphs
    │   ├── ImaginaryEigenvalues.pdf
    │   ├── ImaginaryTensorCoefficients.pdf
    │   ├── RealEigenvalues.pdf
    │   └── RealTensorCoefficients.pdf
    ├── Input_files
    │   ├── Settings.py
    │   ├── main.py
    │   ├── geometry.geo
    │   └── geometry.zip
    ├── PlotEditor.py
    └── PlotterSettings.py
```

The `Functions` folder contains the file `Plotters.py`, which contains the functions used for producing the plots. The folder `Graphs` contains the graphs displaying the results of the frequency sweep. The file `Frequencies.csv` contains a list of the frequencies for which the tensors have been calculated and `Eigenvalues.csv` contains the sum $\lambda_i(\mathcal{N}^0 + \mathcal{R}) + \mathrm{i}\lambda_i(\mathcal{I})$. The file `Tensors.csv` stores the MPTs coefficient as row vectors and stacks them so that each row corresponds to the same corresponding frequency in the row of the `Frequencies.csv` file and the eigenvalues in the row of the `Eigenvalues.csv` file. Finally, the files `PlotEditor.py` and `PlotterSettings.py` allow the user to replot and edit the graphs by editing the inputs in `PlotterSettings.py` in the same manner as in 4.1.4 and then running the file `PlotEditor.py` by entering the command

```
python3 PlotEditor.py
```

in the command line from the output folder. Next we discuss the case of a frequency sweep produced using POD.

4.2.3 POD frequency sweep output

In the case of a frequency sweep produced using POD there are two possible options for the structure of the output folder. If `PlotPod = True`, `PODErrorBars = True` and `EddyCurrentTest = True`, the folder will have the structure shown in Figure **??**.

```
OutputFolder
    ├── Data
    │   ├── Eddy_current-breakdown.txt
    │   ├── Eigenvalues.csv
    │   ├── Frequencies.csv
    │   ├── N0.csv
    │   ├── PODEigenvalues.csv
    │   ├── PODFrequencies.csv
    │   ├── PODTensors.csv
    │   └── Tensors.csv
    ├── Functions
    │   └── Plotters.py
    ├── Graphs
    │   ├── ImaginaryEigenvalues.pdf
    │   ├── ImaginaryTensorCoefficients.pdf
    │   ├── RealEigenvalues.pdf
    │   └── RealTensorCoefficients.pdf
    ├── Input_files
    │   ├── Settings.py
    │   ├── main.py
    │   ├── geometry.geo
    │   └── geometry.zip
    ├── PODPlotEditor.py
    ├── PODPlotEditorWithErrorBars.py
    ├── PlotEditor.py
    ├── PlotEditorWithErrorBars.py
    └── PlotterSettings.py
```

The key differences to the full order solve are the inclusion of the data files `PODFrequencies.csv`, `PODEigenvalues.csv`, `PODTensors.csv` and `ErrorBars.csv`. These files contain the frequencies for the snapshots, the eigenvalues of the tensors calculated at the snapshots, the tensors calculated at the snapshots and the certificate bounds of the outputs, respectively. The other difference between this and the full order sweep is the inclusion of the extra plot editor files `PODPlotEditor.py`, `PODPlotEditorWithErrorBars.py` and `PlotEditorWithErrorBars.py`, which enables plots of both the snapshots and full frequency sweep to be reproduced.

In the case where `PlotPod = False`, the structure is as shown in folder tree shown above except that the files `PODEigenvalues.csv`, `PODTensors.csv` and `PODPlotEditor.py` are not included in the folder. Similarly when `PODErrorBars = False`, we find the exclusion of the files `ErrorBars.csv`, `PODPlotEditorWithErrorBars.py` and `PlotEditorWithErrorBars.py` from the output folder. Finally when `PlotPod = False` and `PlotEditorWithErrorBars.py = False` we have the exclusion of `PODEigenvalues.csv`, `PODTensors.csv` and `ErrorBars.csv` along with `PODPlotEditor.py`, `PODPlotEditorWithErrorBars.py` and `PlotEditorWithErrorBars.py`.

The user may replot the graphs by entering the commands

```
python3 PlotEditorWithErrorBars.py
python3 PODPlotEditor.py
python3 PODPlotEditorWithErrorBars.py
```

in the output folder, to show a plot with the snapshots and/ or certificate bounds being displayed on the graph. Or if the command

```
python3 PlotEditor.py
```

is entered in the output folder, the graphs without the snapshots included is reproduced. With this we conclude our section on the overview and structure of the code. In the next section, we discuss how to create an object using a `.geo` file.

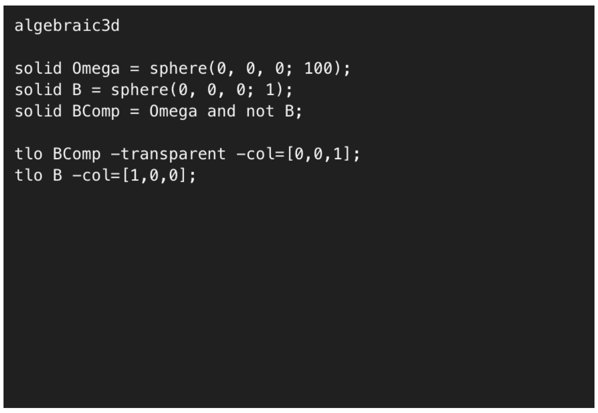
5 Creating your own geometry file

In this section we discuss how to create a geometry file (`.geo`), with the geometry file being interpreted by `Netgen` [13] (the meshing tool in `NGSolve`) we first explain how to create shapes and geometries using a `.geo` file.

Due to `Netgen` being the underlying interpreter of the `.geo` file we direct the user to the relevant Constructive Solid Geometry (CSG) documentation available at `http://netgen-mesher.sourceforge.net/docs/ng4.pdf`. With this knowledge of this, we next discuss how the requirements outlined in Section 2.3, relating to the domain Ω, which contains the object B, can be specified. Finally we explain how to define material properties of the objects being simulated.

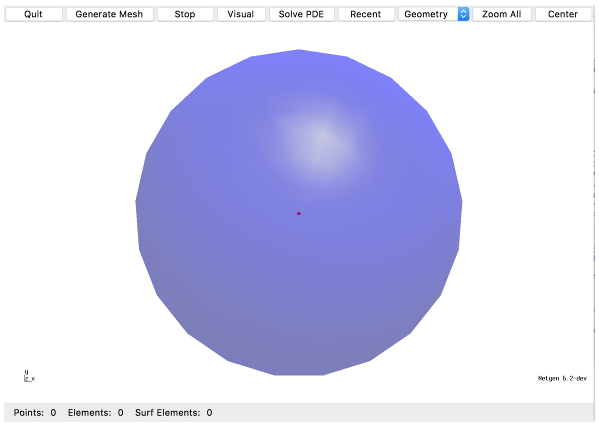
5.1 Truncating the domain

Since it is impossible to simulate an infinite computational domain we are required to create a truncated domain Ω whose boundary $\partial\Omega$ is sufficiently far from the object B. In order to do this in the `.geo` file we simply define a region which is much larger than the object and place the object at its centre. An example of this is presented in the `.geo` file shown in Figure 4 along with a visualisation of the of the geometry obtained in `Netgen`.



```
algebraic3d

solid Omega = sphere(0, 0, 0; 100);
solid B = sphere(0, 0, 0; 1);
solid BComp = Omega and not B;

tlo BComp -transparent -col=[0,0,1];
tlo B -col=[1,0,0];
```

(a) Example of a `.geo` file describing a sphere in a sphere



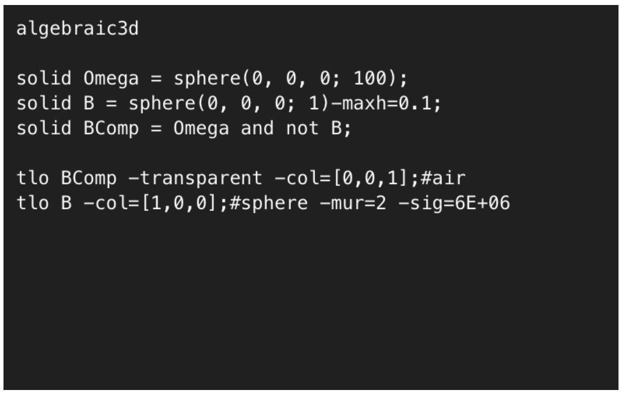
(b) Visulisation of the `.geo` file in (a)

Figure 4: Example of a `.geo` file (a) describing a sphere of unit radius contained in a domain consisting of a sphere with a radius of 100, with (b) a visualisation of of the constructed geometry using negten.

This example defines a unit sphere B contained in a domain Ω, which consists of a sphere with a radius of 100. Truncating the domain at a distance approximately 100 times the object size is sufficient for most cases. Next we discuss how to define the material properties of the different regions in the domain.

5.2 Defining material properties

To set the material parameters, the user is required to label each region defined as a Top Level Object (tlo). This is done by inserting a `#` after each toplevel object followed by a label for the material, it's relative permeability μ_r and it's conductivity σ_*. We define the material properties for the example presented in the Section 5.1 in the Figure 5.



```
algebraic3d

solid Omega = sphere(0, 0, 0; 100);
solid B = sphere(0, 0, 0; 1)-maxh=0.1;
solid BComp = Omega and not B;

tlo BComp -transparent -col=[0,0,1];#air
tlo B -col=[1,0,0];#sphere -mur=2 -sig=6E+06
```

Figure 5: Image displaying an example of a `.geo` file with material properties defined for it's regions.

In Figure 5, `#sphere -mur=2 -sig=6e+06` labels the object B as `"sphere"` and defines it's material properties to be $\mu_r = 2$ and $\sigma_* = 6 \times 10^6$S/m. The inclusion of `#air` labels the non conducting region as `"air"`, which is a protected material and, therefore, does not require the user to input a value for μ_r or σ_*. Lastly, note the inclusion of `-maxh=0.1` when defining the geometry of B. This is an inbuilt function of `Netgen` for meshing purposes allowing the user to specify a maximum element size for each region in the domain. This is useful since it allows the user to refine the mesh in the conducting region of the domain.

Next, we specify the restrictions of the syntax for defining material properties. Any line which starts with `tlo` (which must have no spaces before) must also contain a material label defined as `#material`. On the same line, the user must define both μ_r and σ_* (unless the material is defined to be `#air`), to do this the user must provide the flag `-mur=***` and `-sig=***` with at least one space between the two flags, but no spaces before or either of the equal signs, and with `***` replaced with value of the parameter. Note that σ_* should be specified in S/m. Some examples of this can be seen below, first the following examples are accepted,

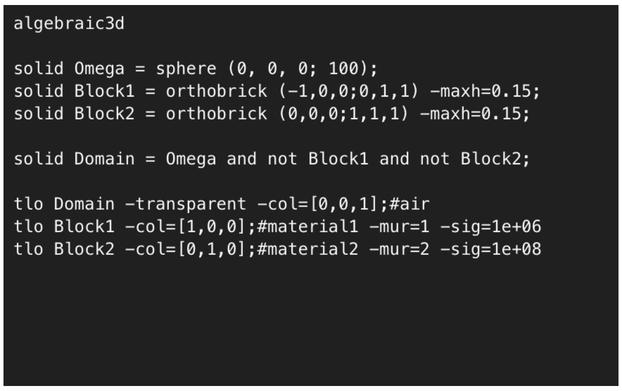
```
tlo B -col=[1,0,0];#sphere -mur=2 -sig=6E+06
tlo B -col=[1,0,0];    #sphere-mur=2 -sig=6E+06
tlo B -col=[1,0,0];#sphere    -mur=2    -sig=6E+06
```

However, the following examples will not work

```
tlo B -col=[1,0,0];#sphere -mur = 2 -sig = 6E+06
 tlo B -col=[1,0,0];#sphere-mur=2 -sig=6E+06
tlo B -col=[1,0,0];#sphere -mur=2-sig=6E+06
tlo B -col=[1,0,0];#sphere
-mur=2-sig=6E+06
```

Due to there being a space before or after the equal signs in the first; to a space before the `tlo` in the second; to the lack of a space between the definition of `-mur` and `-sig` in the third and due to the split two lines in the last.

We next consider the case of an object made up of multiple conducting regions follow the examples in Ledger, Lionheart and Amad [9]. We define a conducting rectangular bar, B which is a $2 \times 1 \times 1$ block made up 2 distinct sections, contained in a domain bounded by a sphere with radius of 100, as defined in the `.geo` file shown in Figure 6. Note that B does not have units, the object αB has units with α being the size parameter specified in m and α is specified later.

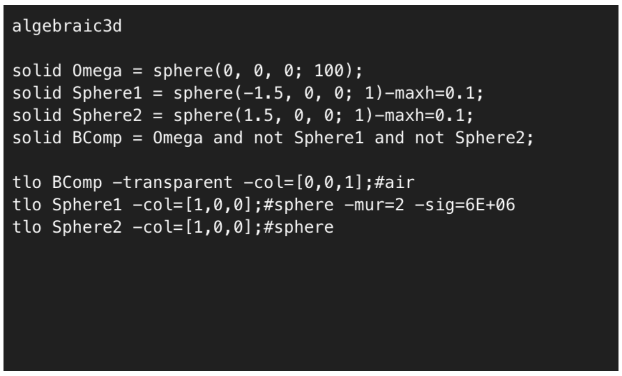


```
algebraic3d

solid Omega = sphere (0, 0, 0; 100);
solid Block1 = orthobrick (-1,0,0;0,1,1) -maxh=0.15;
solid Block2 = orthobrick (0,0,0;1,1,1) -maxh=0.15;

solid Domain = Omega and not Block1 and not Block2;

tlo Domain -transparent -col=[0,0,1];#air
tlo Block1 -col=[1,0,0];#material1 -mur=1 -sig=1e+06
tlo Block2 -col=[0,1,0];#material2 -mur=2 -sig=1e+08
```

Figure 6: Image displaying an example of a `.geo` file which defines a bar constructed of 2 regions with different material properties contained in a domain consisting of a sphere with a radius of 100.

In Figure 6 we have defined two distinct regions `Block1` and `Block2` with materials `"Material1"` and `"Material2"` having different material properties, respectively.

We finish this section with an example of two unit spheres which are constructed using the same material. An example of the `.geo` file can be seen in Figure 7.



```
algebraic3d

solid Omega = sphere(0, 0, 0; 100);
solid Sphere1 = sphere(-1.5, 0, 0; 1)-maxh=0.1;
solid Sphere2 = sphere(1.5, 0, 0; 1)-maxh=0.1;
solid BComp = Omega and not Sphere1 and not Sphere2;

tlo BComp -transparent -col=[0,0,1];#air
tlo Sphere1 -col=[1,0,0];#sphere -mur=2 -sig=6E+06
tlo Sphere2 -col=[1,0,0];#sphere
```

Figure 7: Image displaying an example of a `.geo` file which defines a bar constructed of 2 regions with different material properties contained in a domain consisting of a sphere with a radius of 100.

This example demonstrates how we only need to define the material properties for `#sphere` once even though there are two regions to which it will be applied. This makes it easier to change material properties for a whole geometry when made up of multiple `tlo`s. We expect this to be useful when working with more complex geometries such as that in the case of a rifle shell, which will be presented in Section 6.5.

5.3 OCC Geometry

The current version of `Netgen` (version 6.2.2203) supports defining object geometries via the Open Cascade Technology (OCC) format. Unlike the `.geo` file format, the OCC format is purely Python based. I.e. geometries are defined in `.py` files. Consequently, the user can use standard Python arguments and syntax. For example importing libraries and using iterative loops.

In the same way as the `.geo` file format, `Netgen` constructs complex shapes out of simpler geometric primitives. Currently, `NGSolve` recommends that OCC geometries be used rather than `.` files, but there is little documentation available about available primitives and methods. For documentation, we refer the user to the `NGSolve` documentation and the Open Cascade Technology bottle example:, here and here.

An example OCC geometry file to define a unit radius sphere is presented in Listing 6. In this example we illustrate introducing a unit radius sphere into a larger non-conducting truncated region. This example is included as `OCC_Geometry/OCC_sphere.py`.

```
from netgen.occ import *

# Setting mur, sigma, and defining the top level object name:
object_name = 'sphere'
mur = 1
sigma = 1e6

# setting radius
r = 1

# Generating OCC primative sphere centered at [0,0,0] with radius r:
sphere = Sphere(Pnt(0,0,0), r=r)

# Generating surrounding non-conducting region as [-1000,1000]^3 box:
box = Box(Pnt(-1000, -1000, -1000), Pnt(1000,1000,1000))

# setting material and bc names:
# For comparability, we want the non-conducting region to have the 'outer
    ' boundary condition and be labelled as 'air'
sphere.mat(object_name)
sphere.bc('default')
box.mat('air')
box.bc('outer')

# Setting maxh:
sphere.maxh = 0.5
box.maxh = 1000

# Joining the two meshes:
# Glue joins two OCC objects together without interior elemements
joined_object = Glue([sphere, box])

# Generating Mesh:
geo = OCCGeometry(joined_object)
nmesh = geo.GenerateMesh()
nmesh.Save(r'VolFiles/OCC_sphere.vol')
```

Listing 6: OCC description for a unit radius sphere inside a larger truncated non-conducting region.

Similarly to the .geo file format, we make specific mandates for the format. Each OCC geometry file must include:

```
object_name = ['object 1', 'object 2', ... ]
sigma = [conductivity_1, conductivity_2, ...]
mur = [mur_1, mur_2, ...]
```

corresponding to the names (string) of the different sub-regions that make up the object and their conductivity (S/m) (float) and relative magnetic permeability (float) values respectively which, in the case of multiple sub-regions can be defined in the form of the lists. In the simplest case, where only one primitive is used, then these can be defined using a single entry. E.g.

```
object_name = 'sphere'
sigma = 1e6
mur = 1
```

would define a single object called 'sphere' with conductivity $\sigma_* = 10^6$ S/m and relative permeability $\mu_r = 1$.

In addition, we also mandate that each mesh generated using the OCC format must be save using the same name as the .py file that defines the geometry. For example in Listing **??**spherelst:OCC_sphereave the file name `OCC_sphere.py` and save the generated mesh as `OCC_sphere.vol`. In a similar way to the .geo file format, each OCC .py file must be stored in the `OCC_Geometry` folder and each mesh must be stored in the VolFiles folder.

Secondly, there must be a valid object description using the available OCC primitives and a defined non-conducting region. For example:

```
sphere = Sphere(Pnt(0,0,0), r=1)
box = Box(Pnt(-1000,-1000,-1000), Pnt(1000,1000,1000))
```

defines a sphere of radius 1 and a large cube, centred at (0,0,0) with side length 2000. Note the similarities between the .geo file syntax for a simple primitive and the OCC primitive. We also need to specify material names, boundary condition names, and any additional properties (e.g.). For compatibility with MPT-Calculator, we want the non-conducting region to have the 'outer' boundary condition and be labelled as 'air'. In addition, the material name for the conducting object must match one of the entries in object_name.

```
sphere.mat(object_name[0])
sphere.bc('default')
box.mat('air')
box.bc('outer')
```

Finally, we need to join the non-conducting region with the conducting object. To avoid interior elements, we use the method:

```
joined_object = Glue([sphere, box])
```

Once the geometric description has been defined, the user must generate and save a mesh. This is done using the OCCGeometry, GenerateMesh, and Save methods. E.g.

```
geo = OCCGeometry(joined_object)
nmesh = geo.GenerateMesh()
nmesh.Save(r'VolFiles/OCC_sphere.vol')
```

We recommend that the user maintains a similar style when defining OCC .py files to ensure compatibility. In addition, GenerateMesh admits multiple meshing arguments, such as meshsize.very_coarse, meshsize.coarse, meshsize.moderate, ect that have the same effect as the mesh size options discussed in Section 4.1.1.

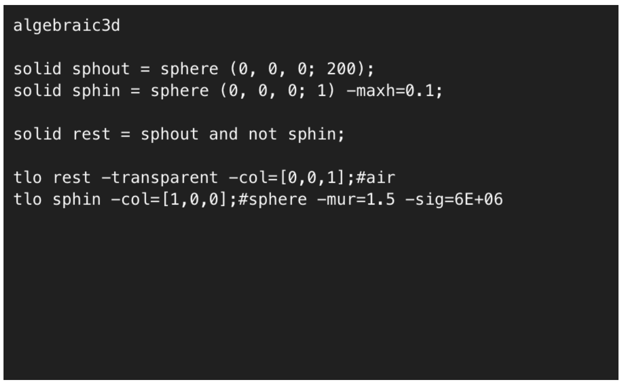
Further examples are provided in the OCC_Geometry folder.

6 Examples

In this section, we consider a set of examples in which the functionality of the MPT-Calculator is demonstrated. Note that these examples have been run with NGSolve 6.2.2004 if running with a different versions the user may obtain different meshes, due to this the meshes used to produce the results have been included in the VolFiles folder. We start with the case of a sphere.

6.1 Sphere

For the case where B is a sphere of unit radius and αB is a sphere of radius $\alpha = 0.01$m we shall create three different simulations, the first is a simulation consisting of a single frequency, the second a full order frequency sweep and the third a frequency sweep implementing the POD. All of these sphere examples are created using the sphere.geo file, shown in Figure 8, where Ω is chosen to be a ball of radius 200 containing the object B, $\sigma_* = 6 \times 10^6$ S/m and $\mu_* = 1.5\mu_0$.



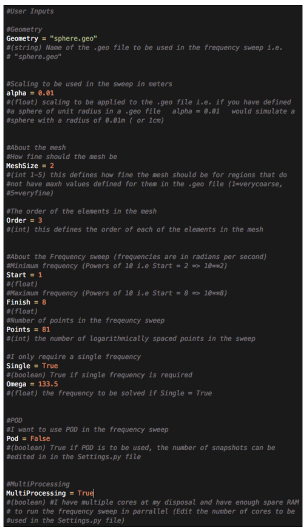
```
algebraic3d

solid sphout = sphere (0, 0, 0; 200);
solid sphin = sphere (0, 0, 0; 1) -maxh=0.1;

solid rest = sphout and not sphin;

tlo rest -transparent -col=[0,0,1];#air
tlo sphin -col=[1,0,0];#sphere -mur=1.5 -sig=6E+06
```

Figure 8: Image displaying the `Sphere.geo` file used in all of the sphere frequency sweep examples in Section 6.1.

We start our examples session with a simulation of the sphere described in Figure 8 for a single frequency.

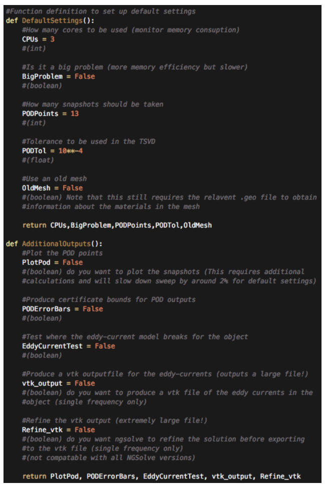
6.1.1 Single frequency sweep for a sphere

To set up this simulation, we used the `sphere.geo` file shown in Figure 8 along with the inputs displayed in Figure 9.



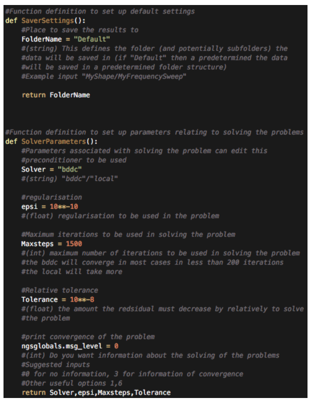
```
#User Inputs

#Geometry
Geometry = "sphere.geo"
#(string) Name of the .geo file to be used in the frequency sweep i.e.
# "sphere.geo"


#Scaling to be used in the sweep in meters
alpha = 0.01
#(float) scaling to be applied to the .geo file i.e. if you have defined
#a sphere of unit radius in a .geo file   alpha = 0.01   would simulate a
#sphere with a radius of 0.01m ( or 1cm)


#About the mesh
#How fine should the mesh be
MeshSize = 2
#(int 1-5) this defines how fine the mesh should be for regions that do
#not have maxh values defined for them in the .geo file (1=verycoarse,
#5=veryfine)

#The order of the elements in the mesh
Order = 3
#(int) this defines the order of each of the elements in the mesh


#About the Frequency sweep (frequencies are in radians per second)
#Minimum frequency (Powers of 10 i.e Start = 2 => 10**2)
Start = 1
#(float)
#Maximum frequency (Powers of 10 i.e Start = 8 => 10**8)
Finish = 8
#(float)
#Number of points in the freqeuncy sweep
Points = 81
#(int) the number of logarithmically spaced points in the sweep

#I only require a single frequency
Single = True
#(boolean) True if single frequency is required
Omega = 133.5
#(float) the frequency to be solved if Single = True


#POD
#I want to use POD in the frequency sweep
Pod = False
#(boolean) True if POD is to be used, the number of snapshots can be
#edited in in the Settings.py file


#MultiProcessing
MultiProcessing = True
#(boolean) #I have multiple cores at my disposal and have enough spare RAM
# to run the frequency sweep in parrallel (Edit the number of cores to be
#used in the Settings.py file)
```

(a) An image of `main.py`.



```
#Function definition to set up default settings
def DefaultSettings():
    #How many cores to be used (monitor memory consuption)
    CPUs = 3
    #(int)

    #Is it a big problem (more memory efficiency but slower)
    BigProblem = False
    #(boolean)

    #How many snapshots should be taken
    PODPoints = 13
    #(int)

    #Tolerance to be used in the TSVD
    PODTol = 10**-4
    #(float)

    #Use an old mesh
    OldMesh = False
    #(boolean) Note that this still requires the relavent .geo file to obtain
    #information about the materials in the mesh

    return CPUs,BigProblem,PODPoints,PODTol,OldMesh

def AdditionalOutputs():
    #Plot the POD points
    PlotPod = False
    #(boolean) do you want to plot the snapshots (This requires additional
    #calculations and will slow down sweep by around 2% for default settings)

    #Produce certificate bounds for POD outputs
    PODErrorBars = False
    #(boolean)

    #Test where the eddy-current model breaks for the object
    EddyCurrentTest = False
    #(boolean)

    #Produce a vtk outputfile for the eddy-currents (outputs a large file!)
    vtk_output = False
    #(boolean) do you want to produce a vtk file of the eddy currents in the
    #object (single frequency only)

    #Refine the vtk output (extremely large file!)
    Refine_vtk = False
    #(boolean) do you want ngsolve to refine the solution before exporting
    #to the vtk file (single frequency only)
    #(not compatable with all NGSolve versions)

    return PlotPod, PODErrorBars, EddyCurrentTest, vtk_output, Refine_vtk
```

(b) Top of `Settings.py`.



```
#Function definition to set up default settings
def SaverSettings():
    #Place to save the results to
    FolderName = "Default"
    #(string) This defines the folder (and potentially subfolders) the
    #data will be saved in (if "Default" then a predetermined the data
    #will be saved in a predetermined folder structure)
    #Example input "MyShape/MyFrequencySweep"

    return FolderName



#Function definition to set up parameters relating to solving the problems
def SolverParameters():
    #Parameters associated with solving the problem can edit this
    #preconditioner to be used
    Solver = "bddc"
    #(string) "bddc"/"local"

    #regularisation
    epsi = 10**-10
    #(float) regularisation to be used in the problem

    #Maximum iterations to be used in solving the problem
    Maxsteps = 1500
    #(int) maximum number of iterations to be used in solving the problem
    #the bddc will converge in most cases in less than 200 iterations
    #the local will take more

    #Relative tolerance
    Tolerance = 10**-8
    #(float) the amount the redsidual must decrease by relatively to solve
    #the problem

    #print convergence of the problem
    ngsglobals.msg_level = 0
    #(int) Do you want information about the solving of the problems
    #Suggested inputs
    #0 for no information, 3 for information of convergence
    #Other useful options 1,6
    return Solver,epsi,Maxsteps,Tolerance
```

(c) Bottom of `Settings.py`.

Figure 9: Images displaying the inputs for the single frequency sweep for a sphere (a) image of the inputs in `main.py` (b) image of the inputs in the top of `Settings.py` (c) image of the inputs in the bottom of `Settings.py`.

These inputs for `main.py` and `Settings.py` have been summarised in Table 1. As `Single = True` the following variables are not used: `Start = 1`, `Finish = 8`, `Points = 81`, `Pod = False`, `PODPoints = 13`, `PODTol = 10**-4`, `PlotPod = False` and `PODErrorBars = False`, hence, their values are arbitrary.

main.py

`Geometry = "sphere.geo"`	`alpha = 0.01`	`MeshSize = 2`
`Order = 3`	`Start = 1`	`Finish = 8`
`Points = 81`	`Single = True`	`Omega = 133.5`
`Pod = False`	`MultiProcessing = True`	

Settings.py

`CPUs = 3`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = False`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = False`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

Table 1: A table summarising the inputs for the simulation of the sphere with for a single frequency.

This means our interest lies in computing the characterisation for $\alpha B = 0.01B$ at a frequency of $\omega = 133.5$rad/s. Furthermore, the inputs led to a mesh of 29170 elements and a discretisation of $p = 3$. On a 2012 iMac with a 2.9 GHz quad core i5 processor with 16 GB 1600 MHz DDR3 memory the computation takes 1 minute and 37 seconds using 3 CPUs and results in the following for the $\mathcal{N}^0$ and $\mathcal{M}$,

$$\mathcal{N}^0_{hp} = \begin{pmatrix} 1.80\times10^{-6} & 6.06\times10^{-14} & 1.89\times10^{-15} \\ 6.06\times10^{-14} & 1.80\times10^{-6} & 9.55\times10^{-14} \\ 1.89\times10^{-15} & 9.55\times10^{-14} & 1.80\times10^{-6} \end{pmatrix},$$

$$\mathcal{M}_{hp} = \begin{pmatrix} 1.79\times10^{-6}+6.97\times10^{-8}\mathrm{i} & 5.85\times10^{-14}+6.61\times10^{-15}\mathrm{i} & -3.53\times10^{-15}+2.23\times10^{-14}\mathrm{i} \\ 5.85\times10^{-14}+6.61\times10^{-15}\mathrm{i} & 1.79\times10^{-6}+6.97\times10^{-8}\mathrm{i} & 9.06\times10^{-14}+3.18\times10^{-14}\mathrm{i} \\ -3.53\times10^{-15}+2.23\times10^{-14}\mathrm{i} & 9.06\times10^{-14}+3.18\times10^{-14}\mathrm{i} & 1.79\times10^{-6}+6.97\times10^{-8}\mathrm{i} \end{pmatrix}.$$

The exact tensors for this object are known to be diagonal and multiple of identity [7, 16]. Specifically $(\mathcal{N})_{ii} = 1.80\times10^{-6}$ and $(\mathcal{M})_{ii} = 1.79\times10^{-6} + 6.97\times10^{-8}\mathrm{i}$ to 2dp. By repeatably performing h or p refinement, the off diagonal terms tend to 0 and the diagonal entries of $\mathcal{M}_{hp}$ and $\mathcal{N}_{hp}$ to their exact values according to the convergence curves presented in Figure 3. In this case, the relative error $\|\mathcal{M} - \mathcal{M}_{hp}\|_F/\|\mathcal{M}\|_F = 2.72\times10^{-6}$. We also find the computed eigenvalues to be

$$\lambda(\mathcal{N}^0 + \mathcal{R}) = \begin{pmatrix} 1.79\times10^{-6} \\ 1.79\times10^{-6} \\ 1.79\times10^{-6} \end{pmatrix}, \quad \lambda(\mathcal{I}) = \begin{pmatrix} 6.97\times10^{-8} \\ 6.97\times10^{-8} \\ 6.97\times10^{-8} \end{pmatrix}.$$

We next consider the example of a full order frequency sweep once again for the sphere described by the `.geo` file in Figure 8.

6.1.2 Full order frequency sweep for a sphere

A frequency sweep is now performed for the sphere described by the `sphere.geo` file in Figure 8 along with the inputs represented in Table 2. As `Single = False` and `Pod = False` the following variables are not used `Omega = 133.5`, `PODPoints = 13`, `PODTol = 10**-4` and `PODErrorBars = False`, hence, their values are arbitrary.

main.py

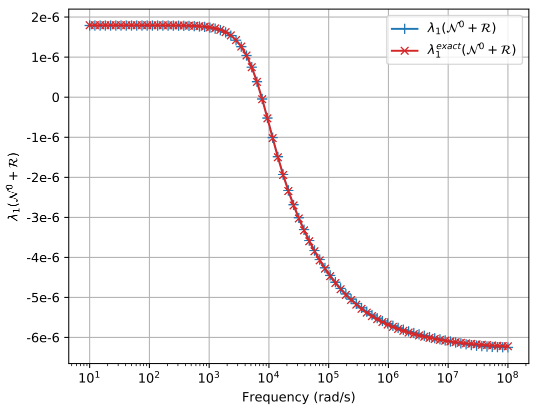
`Geometry = "sphere.geo"`	`alpha = 0.01`	`MeshSize = 2`
`Order = 3`	`Start = 1`	`Finish = 8`
`Points = 81`	`Single = False`	`Omega = 133.5`
`Pod = False`	`MultiProcessing = True`	

Settings.py

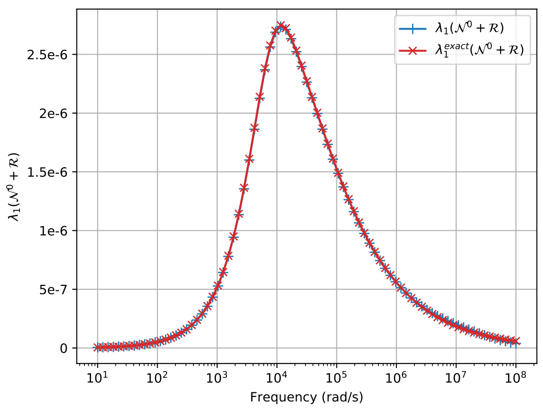
`CPUs = 4`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = False`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = False`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

Table 2: A table summarising the inputs for the simulation of a sphere for a full order frequency sweep.

This means our interest lies in computing the characterisation for $\alpha B = 0.01B$ at 81 frequencies in the range $10^1 \leqslant \omega \leqslant 10^8$rad/s. Furthermore, the inputs lead to a mesh of 29170 elements and a discretisation of $p = 3$. On a 2012 iMac with a 2.9 GHz quad core i5 processor with 16 GB 1600 MHz DDR3 memory the computation takes 1 hour 2 minutes 44 seconds using 4 CPUs, from this we obtain the results shown in Figure 10.



(a) A graph showing how $\lambda_1(\mathcal{N}^0 + \mathcal{R})$ changes with frequency.



(b) A graph showing how $\lambda_1(\mathcal{I})$ changes with frequency.

Figure 10: Graphs displaying the eigenvalues of both the real and imaginary parts for $\mathcal{M}$ calculated using a full order frequency sweep for a sphere compared with the exact values.

In Figure 10, we only show the first eigenvalues $\lambda_1(\mathcal{R}+\mathcal{N}^0)$ and $\lambda_1(\mathcal{I})$, this is due to the difference between $\lambda_1, \lambda_2, \lambda_3$ being negligible in each case. We also show the exact values of the eigenvalues, which corresponds to the diagonal coefficients of the tensors in this case [7, 16], in order to illustrate the performance of the sweep. Note that the code does not produce the exact value for $\lambda_1(\mathcal{R}+\mathcal{N}^0)$ and $\lambda_1(\mathcal{I})$ and these have been included afterwards for demonstration purposes. From a visual inspection, the simulation has obtained very good results and this is confirmed by a sweep of the relative error shown in Figure 11, which shows the maximum relative error is 0.006 over the sweep.

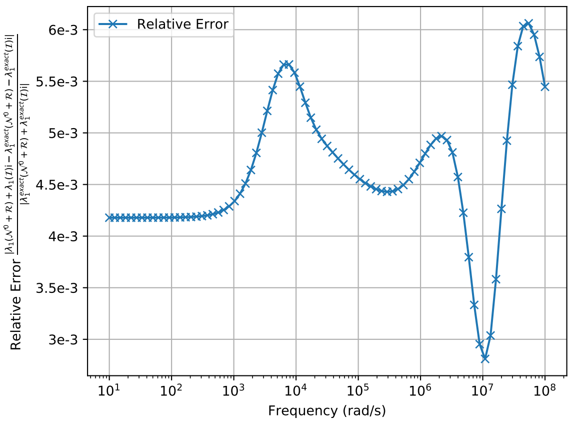


Figure 11: A graph showing how the relative error in the first eigenvalue changes due to frequency.

We next consider the case of a the reduced order frequency sweep using the POD.

6.1.3 POD frequency sweep for a sphere

For this frequency sweep , the sphere described by the `sphere.geo` file in Figure 8 is simulated with the inputs as shown in Table 3. As `Single = False` the following variables are not used: `Omega = 133.5` hence their values are arbitrary.

`main.py`

`Geometry = "sphere.geo"`	`alpha = 0.01`	`MeshSize = 2`
`Order = 3`	`Start = 1`	`Finish = 8`
`Points = 81`	`Single = False`	`Omega = 133.5`
`Pod = True`	`MultiProcessing = True`	

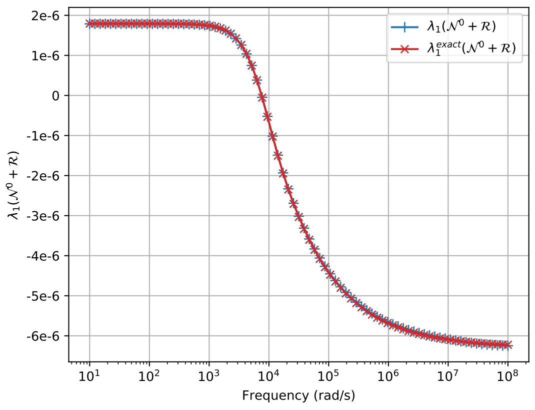
`Settings.py`

`CPUs = 4`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = True`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = False`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

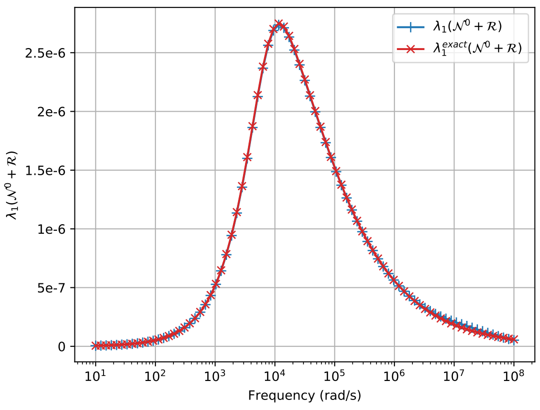
Table 3: A table summarising the inputs for the simulation of a sphere for a reduced order frequency sweep.

This means our interest lies in computing the characterisation for $\alpha B = 0.01B$ at 81 output frequencies in the range $10^1 \leqslant \omega \leqslant 10^8$rad/s using the same mesh of 29170 elements and a $p = 3$ discretisation. However, the result is now generated by using a POD technique in which 13 snapshots are chosen logarithmically (following the approach described in Section 4.1.1) in order to create to create the ROM. On a 2012 iMac with a 2.9 GHz quad core i5 processor with 16 GB 1600 MHz DDR3 memory the computation took 12 minutes and 28 seconds using 4 CPUs leading to the results shown in Figure 12[3].

[3]The POD frequency sweep in Figure 12 has been produced with `ImagTensorFullOrderCalc = True` see Section **??** for more details.



(a) A graph showing how $\lambda_1(\mathcal{N}^0 + \mathcal{R})$ changes with frequency.



(b) A graph showing how $\lambda_1(\mathcal{I})$ changes with frequency.

Figure 12: Graphs displaying the eigenvalues of both the real and imaginary parts for $\mathcal{M}$ calculated using a frequency sweep which implemented a POD for a sphere compared with the exact values.

Again only the first eigenvalues $\lambda_1(\mathcal{R} + \mathcal{N}^0)$ and $\lambda_1(\mathcal{I})$ are shown together with their exact values, added in a post-processing step. From Figure 12, we see that the sweep is very accurate and this is confirmed by showing the relative error as a function of frequency in Figure 13.

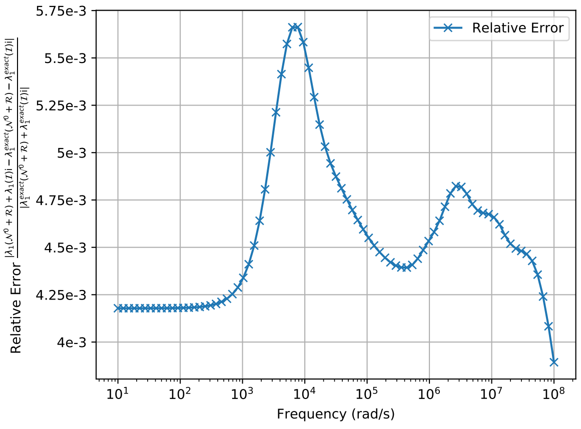
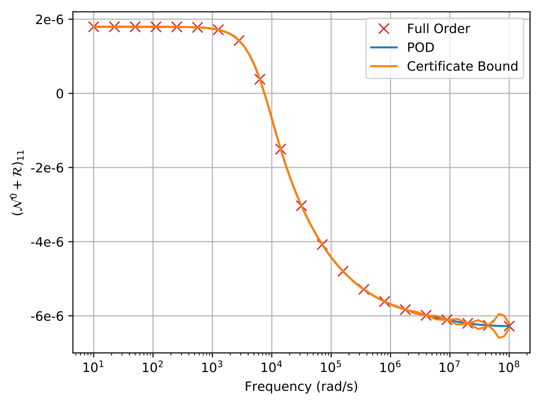
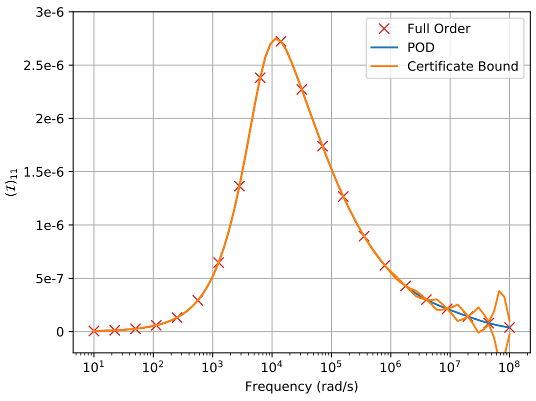


Figure 13: A graph showing how the relative error in the first eigenvalue changes due to frequency.

From Figure 13, we see that the performance is good, if not better, than the full order model, but only requires 13 full order model snapshots and took a quarter of the time of the full order model. However, if we set `PODErrorBars =True` then we obtain upper bounds on the difference between the full order and POD frequency sweeps. An example of this can be seen in Figure 14 this was run for 21 snapshots with a POD tolerance of 10^{-6}. Note that these certificate bounds are computed at run time with minimal additional computational cost. For further details of the cost break down and additional examples of certificate bounds see [17].



(a) A graph showing how $(\mathcal{N}^0 + \mathcal{R})_{11}$ changes with frequency.



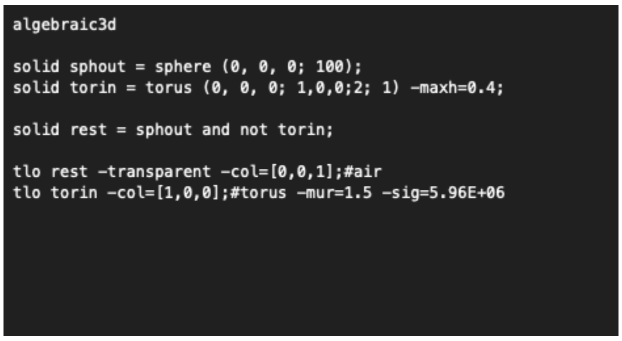
(b) A graph showing how $(\mathcal{I})_{11}$ changes with frequency.

Figure 14: A graph showing the certificate bounds produced for the sphere with 21 snapshots with a POD tolerance of 10^{-6}.

For more detail about how the bounds are computed along with other examples please see [17]. With this we conclude the examples of the sphere, we next consider a simulation of a conducting permeable torus.

6.2 Torus

In this section, we consider the case where B is a torus with a major and minor radii of of 2 and 1, respectively, and the physical object αB is created from this object using the scaling $\alpha = 0.01$m. We simulate a full order frequency sweep using the `Torus.geo` file shown in Figure 15, where Ω is chosen to be a ball of radius 100 containing the object B with material parameters $\sigma_* = 5.96 \times 10^6$ S/m and $\mu_* = 1.5\mu_0$.



```
algebraic3d

solid sphout = sphere (0, 0, 0; 100);
solid torin = torus (0, 0, 0; 1,0,0;2; 1) -maxh=0.4;

solid rest = sphout and not torin;

tlo rest -transparent -col=[0,0,1];#air
tlo torin -col=[1,0,0];#torus -mur=1.5 -sig=5.96E+06
```

Figure 15: An image showing the `Torus.geo` file used to simulate the torus.

To set up this simulation, we used the `Torus.geo` file shown in Figure 15 along with the inputs summarised in Table 4.

`main.py`

`Geometry = "Torus.geo"`	`alpha = 0.01`	`MeshSize = 3`
`Order = 3`	`Start = 1`	`Finish = 8`
`Points = 81`	`Single = False`	`Omega = 133.5`
`Pod = False`	`MultiProcessing = True`	

`Settings.py`

`CPUs = 4`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = True`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = False`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

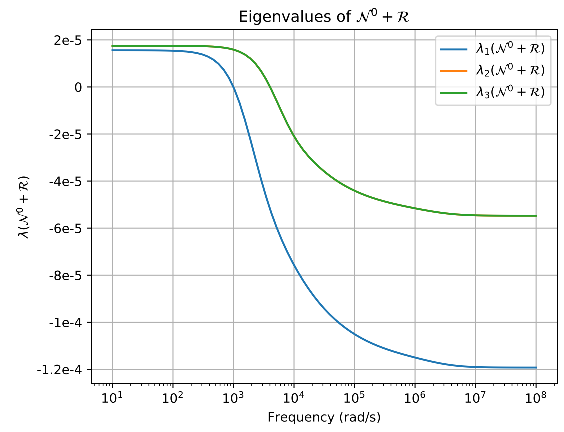
Table 4: A table summarising the inputs for the simulation of a torus for a reduced order frequency sweep.

This means our interest lies in computing the characterisation for $\alpha B = 0.01B$ at 81 frequencies in the range $10^1 \leqslant \omega \leqslant 10^8$ rad/s. Furthermore, the inputs lead to a mesh of 32008 elements and a discretisation of $p = 3$. On a 2012 iMac with a 2.9 GHz quad core i5 processor with 16 GB 1600 MHz DDR3 memory the computation takes 57 minutes 15 seconds using 4 CPUs (POD offers a considerable savings and comparable accuracy, see below). Using `Order = 0` and the above settings has a run time of only 22 minutes 57 seconds, but with lower accuracy. In this case, the inputs in `PlotterSettings.py` are chosen to be as summarised in Table 5[4], which leads to the results shown in Figure 16.

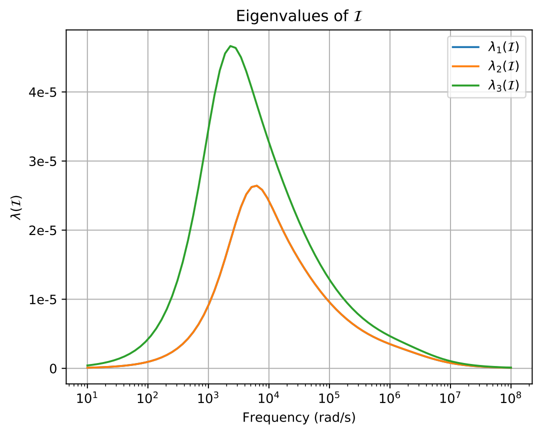
`EigsToPlot = [1,2,3]`	`TensorsToPlot = [1,4,6,2,3,5]`
`MainLineStyle = "-"`	`MainMarkerSize = 4`
`SnapshotLineStyle = "x"`	`SnapshotMarkerSize = 8`
`ErrorBarLineStyle = "--"`	`ErrorBarMarkerSize = 4`
`Title = True`	`EddyCurrentLine = False`

Table 5: A table summarising the inputs for the plots produced by the simulation of a torus using a reduced order frequency sweep.

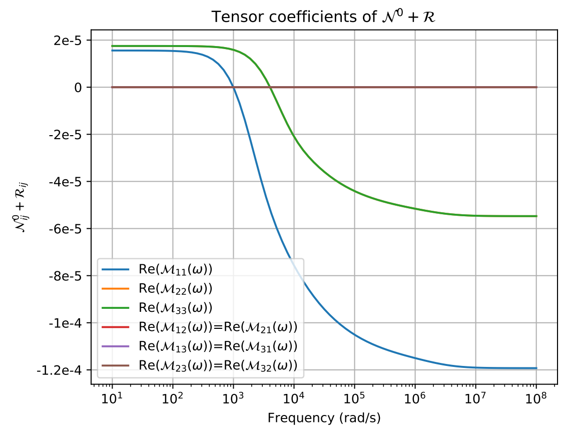
[4]We chosen to not list the `Show` variable as this determines whether the graphs are show at the time of making.



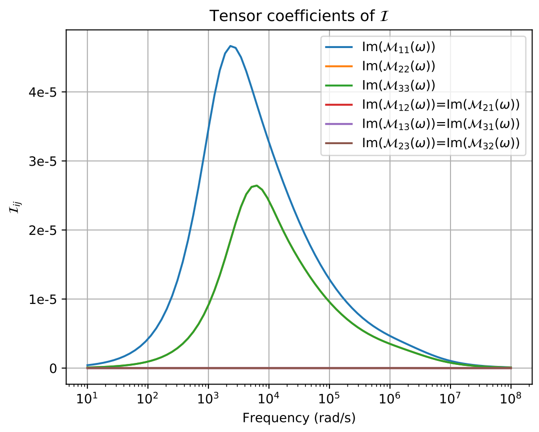
(a) A graph showing how $\lambda_i(\mathcal{N}^0 + \mathcal{R})$ changes with frequency.



(b) A graph showing how $\lambda_i(\mathcal{I})$ changes with frequency.



(c) A graph showing how $\mathcal{N}^0_{ij} + \mathcal{R}_{ij}$ change with frequency.



(d) A graph showing how $\mathcal{I}_{ij}$ change with frequency.

Figure 16: Graphs displaying the eigenvalues and tensor coefficients of both the real and imaginary parts for $\mathcal{M}$ calculated using a full order frequency sweep for a torus.

This means that the eigenvalues $\lambda_i(\mathcal{N}^0+\mathcal{R})$, $\lambda_i(\mathcal{I})$, $i = 1, 2, 3$ are computed and shown as a function of frequency along with tensor coefficients $\mathcal{N}^0_{ij}+\mathcal{R}_{ij}$, $\mathcal{I}_{ij}$ for the combinations $(i, j) = \{(1, 1), (2, 2), (3, 3), (1, 2) = (2, 1), (1, 3) = (3, 1), (2, 3) = (3, 2)\}$. We note that the object has rotational and reflectional symmetries and so it has only 2 independent coefficients corresponding to tensor indices $(1, 1)$ and $(2, 2) = (3, 3)$ [5, 6].

Repeating the same simulation with `Pod = True` in Table 4 results in similar results, but only takes 14 minutes 38 seconds, which is a substantial saving.

With this we conclude our example of the torus, we next consider the example of a tetrahedron.

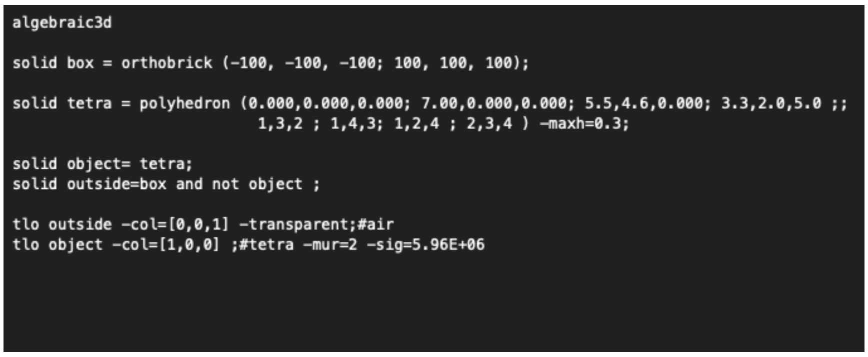
6.3 Tetrahedron

In this section, we consider the case where B is an irregular tetrahedron with vertices

$$(0, 0, 0), \qquad (7, 0, 0), \qquad (5.5, 4.6, 0), \qquad (3.3, 2.0, 0.5)$$

and αB is the irregular tetrahedron produced using the scaling $\alpha = 0.01$m.

The geometry is described by `Tetra.geo` file as shown in Figure 17, where Ω is chosen to be a cube with sides of length 200 containing the object B and its material parameters are $\sigma_* = 5.96 \times 10^6$ S/m and $\mu_* = 2\mu_0$.



```
algebraic3d

solid box = orthobrick (-100, -100, -100; 100, 100, 100);

solid tetra = polyhedron (0.000,0.000,0.000; 7.00,0.000,0.000; 5.5,4.6,0.000; 3.3,2.0,5.0 ;;
                          1,3,2 ; 1,4,3; 1,2,4 ; 2,3,4 ) -maxh=0.3;

solid object= tetra;
solid outside=box and not object ;

tlo outside -col=[0,0,1] -transparent;#air
tlo object -col=[1,0,0] ;#tetra -mur=2 -sig=5.96E+06
```

Figure 17: An image showing the `Tetra.geo` file used to simulate the torus.

The inputs for this simulation are summarised in Table 6.

`main.py`

`Geometry = "Tetra.geo"`	`alpha = 0.01`	`MeshSize = 3`
`Order = 3`	`Start = 2`	`Finish = 8`
`Points = 81`	`Single = False`	`Omega = 133.5`
`Pod = True`	`MultiProcessing = True`	

`Settings.py`

`CPUs = 4`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = True`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = False`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

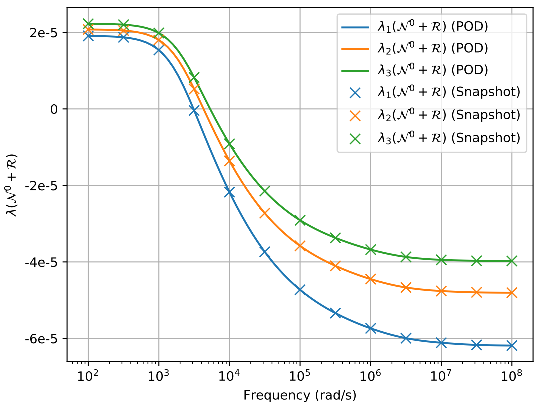
Table 6: A table summarising the inputs for the simulation of an irregular tetrahedron for a reduced order frequency sweep.

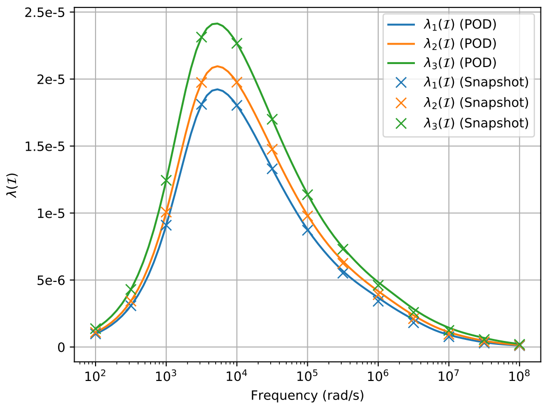
This means our interest lies in computing the characterisation for $\alpha B = 0.01B$ at 81 frequencies in the range $10^1 \leqslant \omega \leqslant 10^8$ rad/s. Furthermore, the inputs lead to a mesh of 22923 elements and a discretisation of $p = 3$. In this case, a POD technique using 13 snapshots chosen logarithmically (following the approach described in Section 4.1.1) is used to create to create the ROM. On a 2012 iMac with a 2.9 GHz quad core i5 processor with 16 GB 1600 MHz DDR3 memory the computation takes 7 minutes and 18 seconds using 4 CPUs.

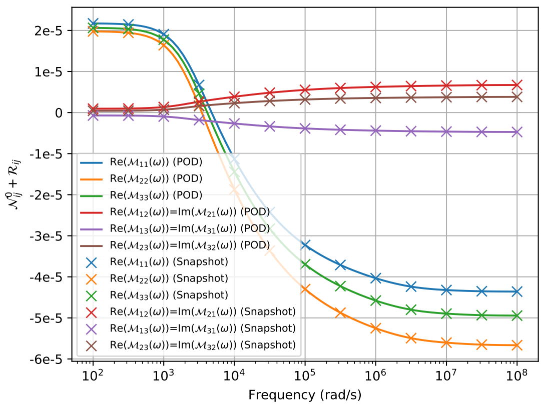
The inputs in `PlotterSettings.py` are summarised in Table 7 leading to the results shown in Figure 18.

`EigsToPlot = [1,2,3]`	`TensorsToPlot = [1,4,6,2,3,5]`
`MainLineStyle = "-"`	`MainMarkerSize = 4`
`SnapshotLineStyle = "x"`	`SnapshotMarkerSize = 8`
`ErrorBarLineStyle = "--"`	`ErrorBarMarkerSize = 4`
`Title = True`	`EddyCurrentLine = False`

Table 7: A table summarising the inputs for the plots produced by the simulation of an irregular tetrahedron using a reduced order frequency sweep.


(a) A graph showing how $\lambda(\mathcal{N}^0 + \mathcal{R})$ changes with frequency.


(b) A graph showing how $\lambda(\mathcal{I})$ changes with frequency.


(c) A graph showing how $\mathcal{N}^0_{ij} + \mathcal{R}_{ij}$ change with frequency.

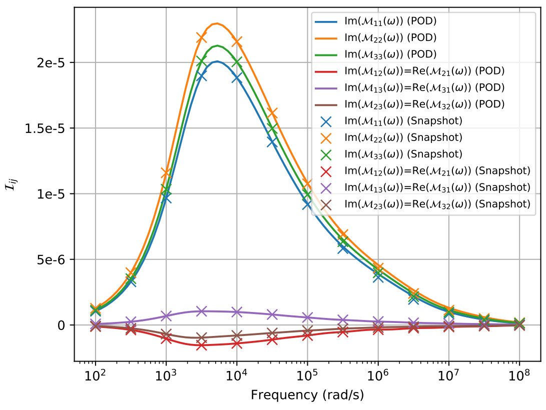

(d) A graph showing how $\mathcal{I}_{ij}$ change with frequency.

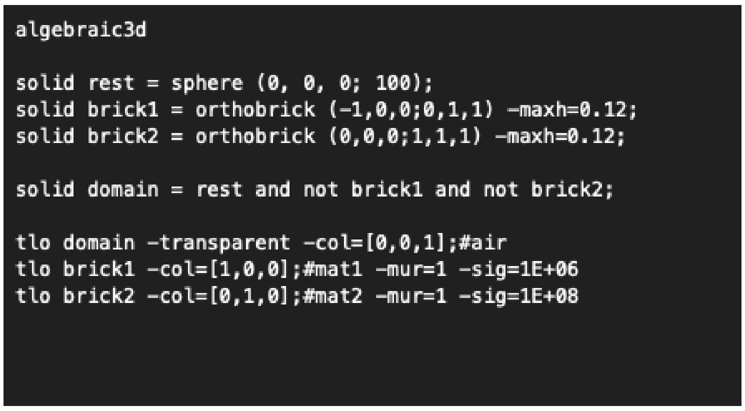
Figure 18: Graphs displaying the eigenvalues and tensor coefficients of both the real and imaginary parts for $\mathcal{M}$ calculated using a reduced order frequency sweep for a tetrahedron.

From Figure 18, we see that $\mathcal{M}$ has 6 independent non-zero coefficients, due to the lack of rotational and reflectional symmetries in the (irregular) tetrahedron. With this we conclude our example of the tetrahedron, we next consider the case of a bar created using two regions.

6.4 Dual Bar

In this section we consider $B = B^{(1)} \cup B^{(2)}$ to be a bar created by joining two rectangular blocks with different parameters. The physical object αB is obtained by scaling B by $\alpha = 0.01$m. The geometry described by the `DualBar.geo` file in Figure 19, where Ω is chosen to be a ball of radius 100 containing the object B. Following the notation in [9], the material parameters of $B^{(1)}$ and $B^{(2)}$ are

$$\begin{aligned} \sigma_*^{(1)} &= 10^6 \text{ S/m} \\ \sigma_*^{(2)} &= 10^8 \text{ S/m} \end{aligned} \qquad \text{and} \qquad \begin{aligned} \mu_*^{(1)} &= \mu_0 \\ \mu_*^{(2)} &= \mu_0 \end{aligned}$$



```
algebraic3d

solid rest = sphere (0, 0, 0; 100);
solid brick1 = orthobrick (-1,0,0;0,1,1) -maxh=0.12;
solid brick2 = orthobrick (0,0,0;1,1,1) -maxh=0.12;

solid domain = rest and not brick1 and not brick2;

tlo domain -transparent -col=[0,0,1];#air
tlo brick1 -col=[1,0,0];#mat1 -mur=1 -sig=1E+06
tlo brick2 -col=[0,1,0];#mat2 -mur=1 -sig=1E+08
```

Figure 19: An image showing the `DualBar.geo` file used to simulate the torus.

To set up this simulation, we used the `Dualbar.geo` file shown in Figure 19 along with the inputs summarised in Table 8

`main.py`

`Geometry = "DualBar.geo"`	`alpha = 0.01`	`MeshSize = 3`
`Order = 3`	`Start = 2`	`Finish = 7`
`Points = 81`	`Single = False`	`Omega = 133.5`
`Pod = True`	`MultiProcessing = True`	

`Settings.py`

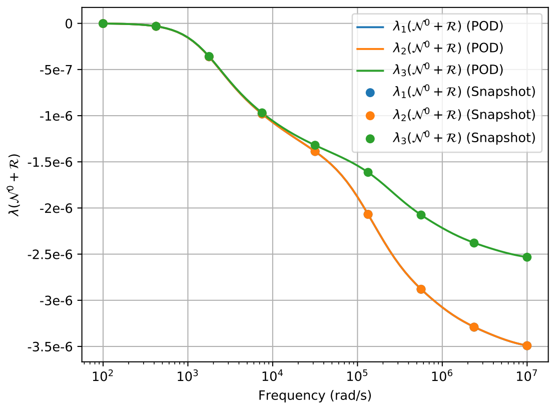
`CPUs = 4`	`BigProblem = False`	`PODPoints = 9`
`PODTol = 1e-05`	`OldMesh = False`	`PlotPod = True`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = False`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

Table 8: A table summarising the inputs for the simulation of a bar containing two regions for a reduced order frequency sweep.

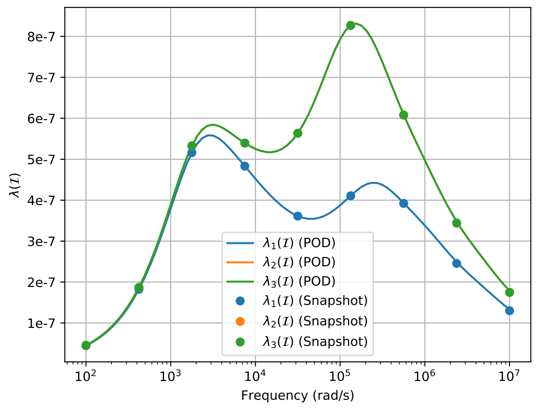
This means our interest lies in computing the characterisation for $\alpha B = 0.01B$ at 81 frequencies in the range $10^2 \leqslant \omega \leqslant 10^7$ rad/s. Furthermore, the inputs lead to a mesh of 36772 elements and a discretisation of $p = 3$. In this case, a POD technique using 9 snapshots chosen logarithmically (following the approach described in Section 4.1.1) is used to create to create the ROM. On a 2012 iMac with a 2.9 GHz quad core i5 processor with 16 GB 1600 MHz DDR3 memory the computation takes 8 minutes and 32 seconds using 4 CPUs. The inputs of `PLotterSettings.py` are summarised in Table 9 leading to the results shown in Figure 20.

`EigsToPlot = [1,2,3]`	`TensorsToPlot = [1,4,6]`
`MainLineStyle = "-"`	`MainMarkerSize = 4`
`SnapshotLineStyle = "o"`	`SnapshotMarkerSize = 6`
`ErrorBarLineStyle = "--"`	`ErrorBarMarkerSize = 4`
`Title = False`	`EddyCurrentLine = True`

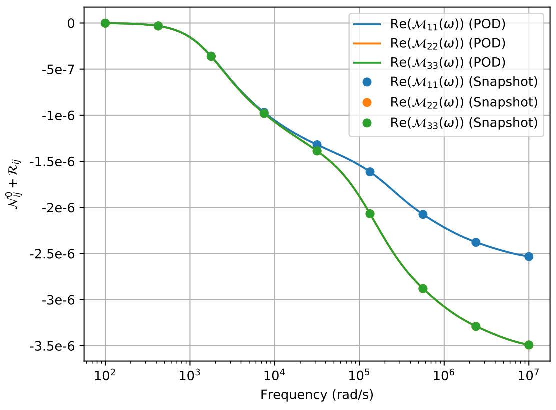
Table 9: A table summarising the inputs for the plots produced by the simulation of bar created using two regions using a reduced order frequency sweep.



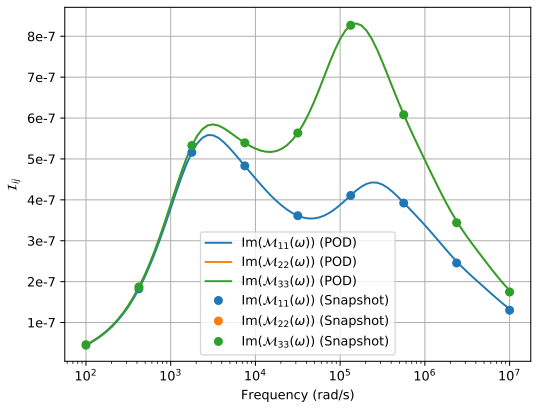
(a) A graph showing how $\lambda(\mathcal{N}^0 + \mathcal{R})$ changes with frequency.



(b) A graph showing how $\lambda(\mathcal{I})$ changes with frequency.



(c) A graph showing how $\mathcal{N}^0_{ij} + \mathcal{R}_{ij}$ change with frequency.



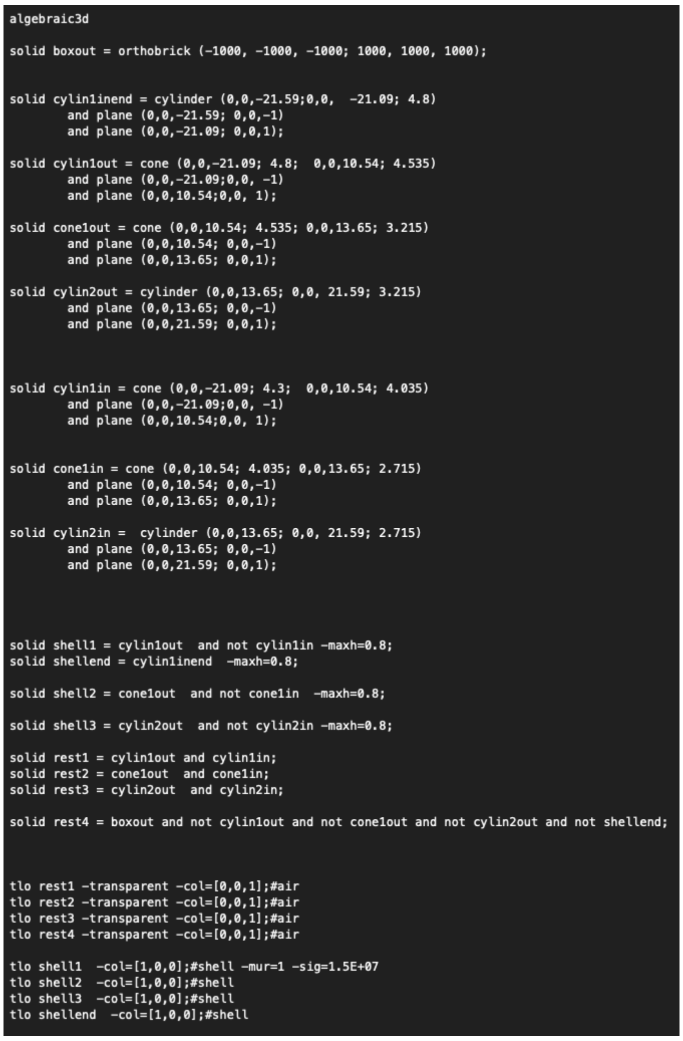
(d) A graph showing how $\mathcal{I}_{ij}$ change with frequency.

Figure 20: Graphs displaying the eigenvalues and tensor coefficients of both the real and imaginary parts for $\mathcal{M}$ calculated using a reduced order frequency sweep for a bar constructed of two regions.

In this case, we have chosen to plot only the non-zero coefficients of the tensors by changing the settings in `PlotterSettings.py`, also, note that even though we have `EddyCurrentLine = True` no line is produced, this is due to having `EddyCurrentTest = False` in `Settings.py`. With this we conclude our example of the bar of two regions, we next consider the case of a rifle shell.

6.5 Rifle shell casing

Finally, we consider αB to the be the rifle shell casing as defined in [11, 6] where B is defined such that $\alpha = 0.001$m. The geometry for B is defined by the `rifle.geo` file in Figure 21, where Ω is chosen to be a cube with sides of length 2000 containing the shell B, which has material properties $\sigma_* = 1.5 \times 10^7$ S/m and $\mu_* = \mu_0$.



```
algebraic3d

solid boxout = orthobrick (-1000, -1000, -1000; 1000, 1000, 1000);


solid cylin1inend = cylinder (0,0,-21.59;0,0,  -21.09; 4.8)
        and plane (0,0,-21.59; 0,0,-1)
        and plane (0,0,-21.09; 0,0,1);

solid cylin1out = cone (0,0,-21.09; 4.8;  0,0,10.54; 4.535)
        and plane (0,0,-21.09;0,0, -1)
        and plane (0,0,10.54;0,0, 1);

solid cone1out = cone (0,0,10.54; 4.535; 0,0,13.65; 3.215)
        and plane (0,0,10.54; 0,0,-1)
        and plane (0,0,13.65; 0,0,1);

solid cylin2out = cylinder (0,0,13.65; 0,0, 21.59; 3.215)
        and plane (0,0,13.65; 0,0,-1)
        and plane (0,0,21.59; 0,0,1);



solid cylin1in = cone (0,0,-21.09; 4.3;  0,0,10.54; 4.035)
        and plane (0,0,-21.09;0,0, -1)
        and plane (0,0,10.54;0,0, 1);


solid cone1in = cone (0,0,10.54; 4.035; 0,0,13.65; 2.715)
        and plane (0,0,10.54; 0,0,-1)
        and plane (0,0,13.65; 0,0,1);

solid cylin2in =  cylinder (0,0,13.65; 0,0, 21.59; 2.715)
        and plane (0,0,13.65; 0,0,-1)
        and plane (0,0,21.59; 0,0,1);




solid shell1 = cylin1out  and not cylin1in -maxh=0.8;
solid shellend = cylin1inend  -maxh=0.8;

solid shell2 = cone1out  and not cone1in  -maxh=0.8;

solid shell3 = cylin2out  and not cylin2in -maxh=0.8;

solid rest1 = cylin1out and cylin1in;
solid rest2 = cone1out  and cone1in;
solid rest3 = cylin2out  and cylin2in;

solid rest4 = boxout and not cylin1out and not cone1out and not cylin2out and not shellend;



tlo rest1 -transparent -col=[0,0,1];#air
tlo rest2 -transparent -col=[0,0,1];#air
tlo rest3 -transparent -col=[0,0,1];#air
tlo rest4 -transparent -col=[0,0,1];#air

tlo shell1  -col=[1,0,0];#shell -mur=1 -sig=1.5E+07
tlo shell2  -col=[1,0,0];#shell
tlo shell3  -col=[1,0,0];#shell
tlo shellend  -col=[1,0,0];#shell
```

Figure 21: An image showing the `rifle.geo` file used to simulate the rifle shell casing.

To set up this simulation, we used the `rfle.geo` file show in Figure 21 along with the inputs summarised in Table 10.

`main.py`

`Geometry = "rifle.geo"`	`alpha = 0.001`	`MeshSize = 3`
`Order = 3`	`Start = 1`	`Finish = 8`
`Points = 81`	`Single = True`	`Omega = 100000`
`Pod = False`	`MultiProcessing = True`	

`Settings.py`

`CPUs = 3`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = False`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = True`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

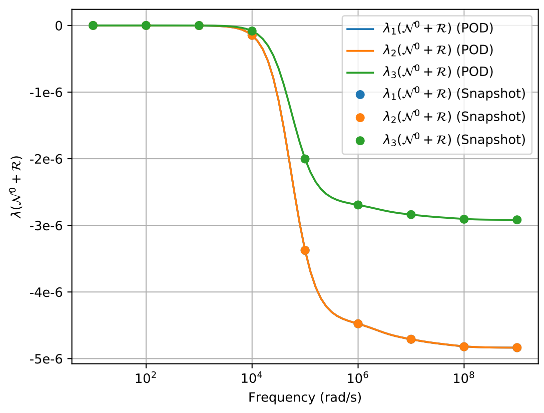
Table 10: A table summarising the inputs for the simulation of a rifle shell casing for a reduced order frequency sweep.

This means our interest lies in computing the charcterisation for $\alpha B = 0.001B$ at 81 frequencies in the range $10^1 \leqslant \omega \leqslant 10^9$ rad/s. Furthermore, the inputs lead to a mesh of 80013 elements and a discretisation of $p = 3$. In this case, a POD technique using 9 snapshots chosen logarithmically (following the approach described in Section 4.1.1) is used to create to create the ROM. On a 2012 iMac with a 2.9GHz quad core i5 processor with 16 Gb 1600 MHz DDR3 memory the computation takes 24 minutes and 46 seconds using 4 CPUs.

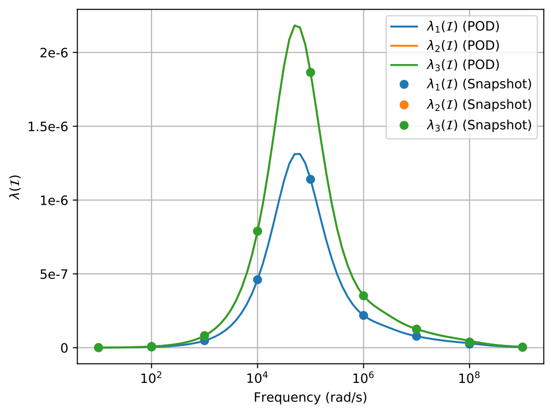
The inputs of `PlotterSettings.py` are summarised in Table 11 leading to the results shown in Figure 22.

`EigsToPlot = [1,2,3]`	`TensorsToPlot = [1,4,6]`
`MainLineStyle = "-"`	`MainMarkerSize = 4`
`SnapshotLineStyle = "o"`	`SnapshotMarkerSize = 6`
`ErrorBarLineStyle = "--"`	`ErrorBarMarkerSize = 4`
`Title = False`	`EddyCurrentLine = False`

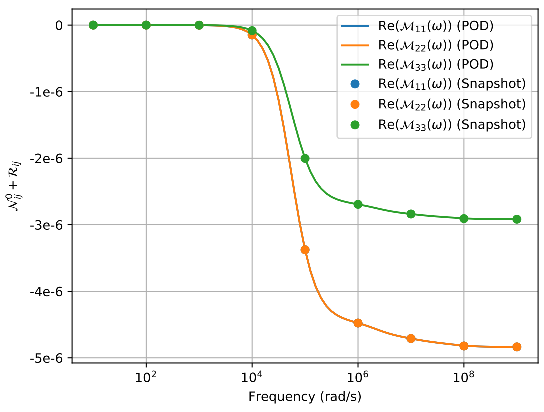
Table 11: A table summarising the inputs for the plots produced by the simulation of a rifle shell casing using a reduced order frequency sweep.



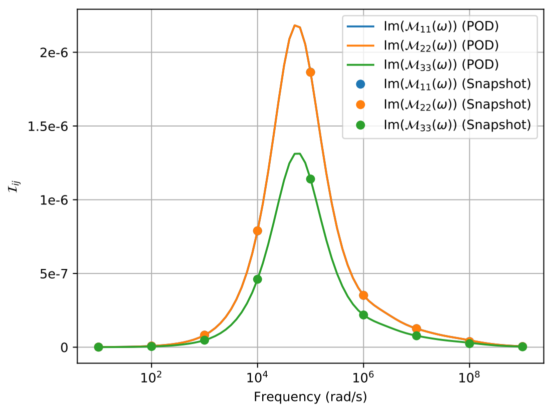
(a) A graph showing how $\lambda(\mathcal{N}^0 + \mathcal{R})$ changes with frequency.



(b) A graph showing how $\lambda(\mathcal{I})$ changes with frequency.



(c) A graph showing how $\mathcal{N}^0_{ij} + \mathcal{R}_{ij}$ change with frequency.



(d) A graph showing how $\mathcal{I}_{ij}$ change with frequency.

Figure 22: Graphs displaying the eigenvalues and tensor coefficients of both the real and imaginary parts for $\mathcal{M}$ calculated using a reduced order frequency sweep for a rifle shell casing.

Once again we have only plotted the non-zero tensor coefficients. In Figure 22 (c) we note that the curve could be improved around $\omega = 10^6$, although is still 'acceptable' if compared to the full order model. This suggests that we may wish to include some more additional snapshots in the reduced order model. Nevertheless, this 81 point frequency sweep for an object αB discretized by 80013 element mesh with $p = 3$ using the POD is computed in considerably less time than the corresponding full order model. We will finally show one more example of the rifle shell casing where a `.vtk` file is produced, the inputs for this can be seen in Figure 12.

`main.py`

`Geometry = "rifle.geo"`	`alpha = 0.001`	`MeshSize = 3`
`Order = 3`	`Start = 1`	`Finish = 8`
`Points = 81`	`Single = True`	`Omega = 100000`
`Pod = False`	`MultiProcessing = True`	

`Settings.py`

`CPUs = 3`	`BigProblem = False`	`PODPoints = 13`
`PODTol = 0.0001`	`OldMesh = False`	`PlotPod = False`
`PODErrorBars = False`	`EddyCurrentTest = False`	`vtk_output = True`
`Refine_vtk = False`	`FolderName = "Default"`	`Solver = "bddc"`
`epsi = 1e-10`	`Maxsteps = 1500`	`Tolerance = 1e-08`
	`ngsglobals.msg_level = 0`	

Table 12: A table summarising the inputs for the simulation of a rifle shell casing for a single frequency with a vtk output.

These inputs compute the characterisation for $\alpha B = 0.001B$, at $\omega = 10^5$. The inputs lead to a mesh of 80013 elements with $p = 3$ with a final vtk output file size of 33.8 MB which has been used to produce the following image of the eddy-currents in paraview displayed in Figure 24. The `rifle.vtk` file stores 7 fields, which can be seen in Table 13.

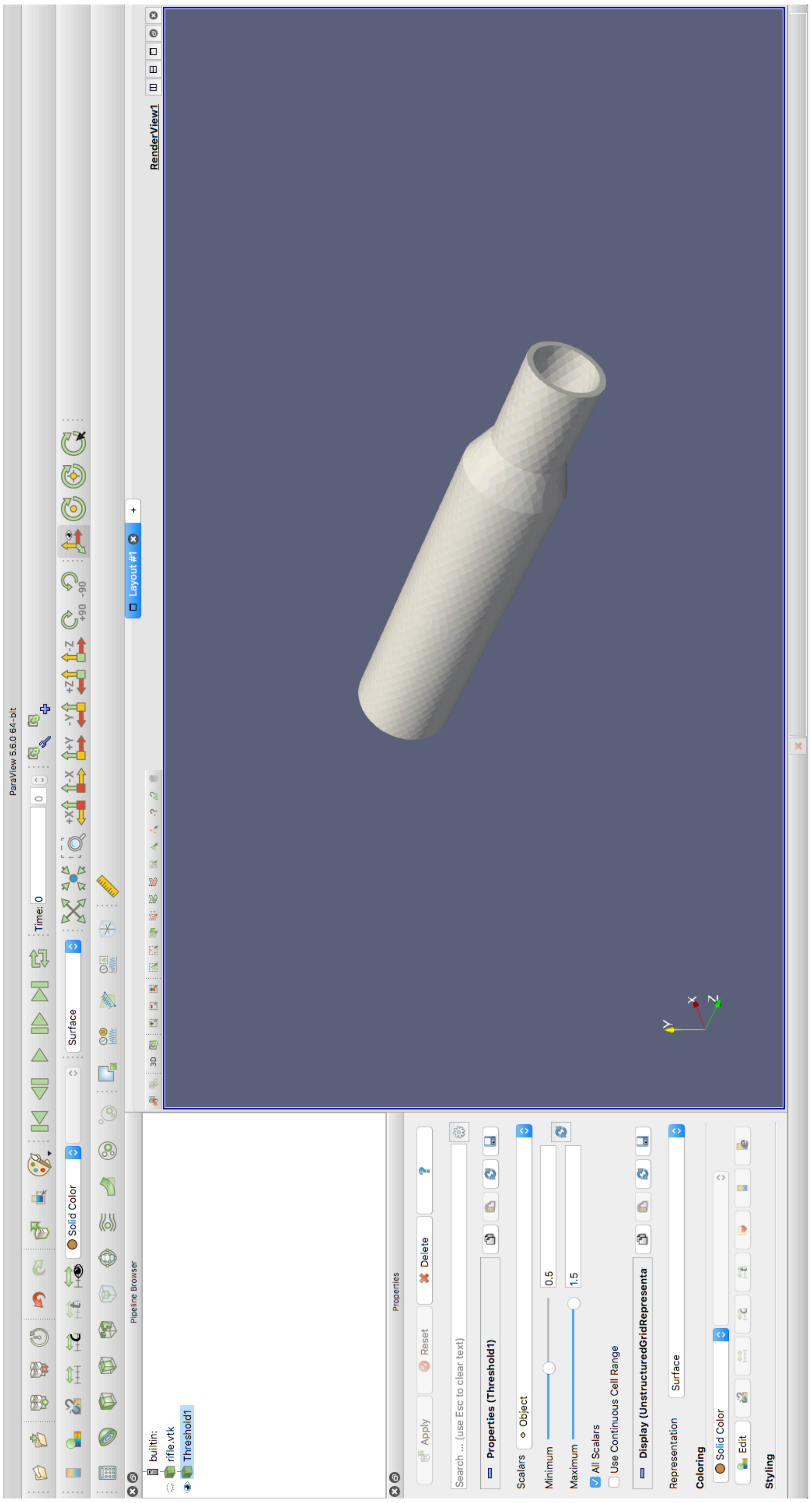
$\mathrm{Re}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_1^{(1)})$	$\mathrm{Re}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_2^{(1)})$	$\mathrm{Re}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_3^{(1)})$
$\mathrm{Im}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_1^{(1)})$	$\mathrm{Im}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_2^{(1)})$	$\mathrm{Im}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_3^{(1)})$
	Object	

Table 13: A table summarising the outputs saved in the `.vtk` output file.

These are the real and imaginary parts of the eddy-currents for each or the three solutions along with the parameter Object. The parameter Object corresponds to a cut-off parameter, which is defined as

$$\chi(\boldsymbol{\xi}) := \begin{cases} 1 & \boldsymbol{\xi} \in B \\ 0 & \text{otherwise.} \end{cases}$$

This allows the user to apply a threshold filter using Object to remove the outer domain an example of which can be seen in Figure 23. The user may use this to show the eddy currents corresponding to $\mathrm{Re}(\mathrm{i}\omega\sigma_\alpha\boldsymbol{\theta}_1^{(1)})$ for just B which can be seen in Figure 24.



Scalars: Object
Minimum: 0.5
Maximum: 1.5
All Scalars: ✓
Use Continuous Cell Range
Representation: Surface
Coloring: Solid Color

Figure 23: An image showing the use of the object parameter in a rifle shell casing.

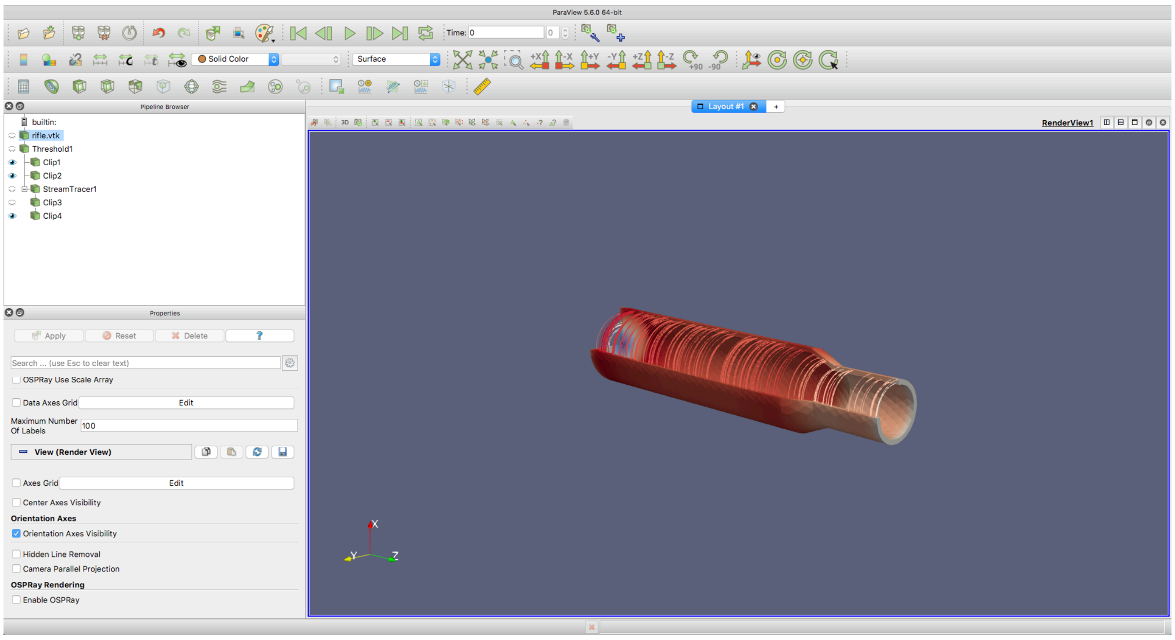


Figure 24: An image showing the eddy-currents in a rifle shell casing with $\omega = 10^5$ rad/s.

With this we conclude the examples section. Along with the `.geo` files for the examples which have been discussed in this article, there are a selection of other `.geo` files which are included with the download of the code.

7 Citation

If you use the tool, please refer to it in your work by citing the references

- [17] B. A. Wilson and P. D. Ledger, Efficient computation of the magnetic polarizabiltiy tensor spectral signature using pod. International Journal for Numerical Methods in Engineering 122, 1940-1963, 2021.
- [10] P. D. Ledger, B. A. Wilson, A. A. S. Amad, W. R. B. Lionheart Identification of meallic objects using spectral MPT signatures: Object characterisation and invariants. International Journal for Numerical Methods in Engineering. Accepted Author Manuscript, 2021.
- [8], P. D. Ledger and W. R. B. Lionheart, The spectral properties of the magnetic polarizability tensor for metallic object characterisation, Math Meth Appl Sci., 43, 78-113, 2020,
- [7] P. D. Ledger and W. R. B. Lionheart, An explicit formula for the magnetic polarizability tensor for object characterization, IEEE Trans Geosci Remote Sens., 56(6), 3520-3533, 2018.

as well as those of `NGSolve`:

- [15] J. Schöberl, C++11 Implementation of Finite Elements in NGSolve, ASC Report 30/2014, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2014.
- [18] S. Zaglmayr, High Order Finite Elements for Electromagnetic Field Computation, PhD Thesis, Johannes Kepler University Linz, 2006
- [14], J. Schöberl, NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules, Computing and Visualization in Science, 1(1), 41-52, 1997.

References

[1] matplotlib. `https://matplotlib.org`.

[2] python. `https://www.python.org`.

[3] H. Ammari, A. Buffa, and J.-C. Nédélec. A justification of eddy currents model for the maxwell equations. *SIAM Journal on Applied Mathematics*, 60(5):1805–1823, 2000.

[4] H. Ammari, J. Chen, Z. Chen, J. Garnier, and D. Volkov. Target detection and characterization from electromagnetic induction data. *J. Math. Pures Appl.*, 101(1):54–75, 2014.

[5] P. D. Ledger and W. R. B. Lionheart. Characterising the shape and material properties of hidden targets from magnetic induction data. *IMA J. Appl. Math.*, 80(6):1776–1798, 2015.

[6] P. D. Ledger and W. R. B. Lionheart. Understanding the magnetic polarizability tensor. *IEEE Trans Magn.*, 52(5):6201216, 2016.

[7] P. D. Ledger and W. R. B. Lionheart. An explicit formula for the magnetic polarizability tensor for object characterization. *IEEE Trans Geosci Remote Sens.*, 56(6):3520–3533, 2018.

[8] P. D. Ledger and W. R. B. Lionheart. The spectral properties of the magnetic polarizability tensor for metallic object characterisation. *Math Meth Appl Sci.*, 43:78–113, 2020.

[9] P. D. Ledger, W. R. B. Lionheart, and A.A.S. Amad. Characterisation of multiple conducting permeable objects in metal detection by polarizability tensors. *Math Meth Appl Sci.*, 42(3):830–860, 2019.

[10] PD Ledger, BA Wilson, AAS Amad, and WRB Lionheart. Identification of metallic objects using spectral mpt signatures: Object characterisation and invariants. *arXiv preprint arXiv:2012.10376*, 2020.

[11] O. A. Abdel Rehim, J. L. Davidson, L. A. Marsh, M. D. O'Toole, D. W. Armitage, and A. J. Peyton. Measurement system for determining the magnetic polarizability tensor of small metal targets. *in Proc. IEEE Sensor Appl. Symp.*, pages 1–5, 2015.

[12] Kersten Schmidt, Oliver Sterz, and Ralf Hiptmair. Estimating the eddy-current modeling error. *IEEE transactions on Magnetics*, 44(6):686–689, 2008.

[13] J. Schöberl. Netgen documentaion. `http://netgen-mesher.sourceforge.net/docs/ng4.pdf`.

[14] J. Schöberl. Netgen - an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, 1997.

[15] J. Schöberl. C++11 implementation of finite elements in ngsolve. Technical report, ASC Report 30/2014, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2014.

[16] J. R. Wait. A conducting sphere in a time varying magnetic field. *Geophsics*, 16(4):666–672, 1951.

[17] B. A. Wilson and P. D. Ledger. Efficient computation of the magnetic polarizabiltiy tensor spectral signature using pod. 2020. https://arxiv.org/abs/2001.07629.

[18] S. Zaglmayr. *High Order Finite Elements for Electromagnetic Field Computation.* PhD thesis, 2006.