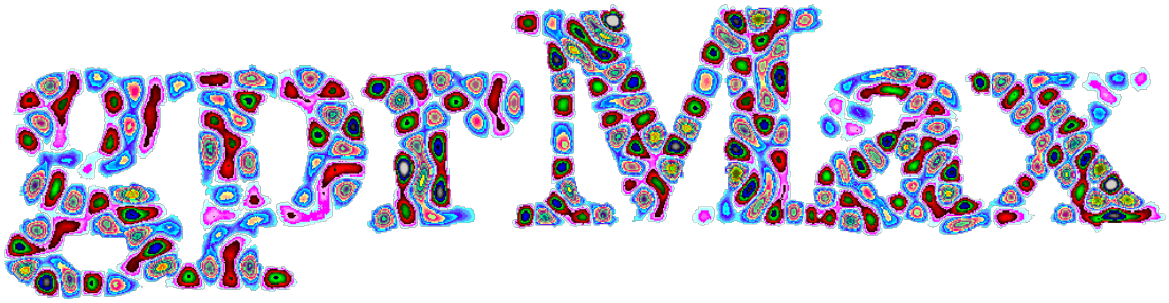


gprMax User Guide

Release 3.1.4

Craig Warren and Antonis Giannopoulos

Dec 21, 2018

Introduction

1 Getting Started **3**
- 1.1 What is gprMax? . . . 3
- 1.2 Package overview . . . 3
- 1.3 Installation . . . 4
- 1.4 Running gprMax . . . 6
- 1.5 Updating gprMax . . . 7

2 Software Features **9**
- 2.1 What's new/changed? . . . 9
- 2.2 Key features . . . 11

3 Guidance on GPR modelling **15**
- 3.1 Basic concepts . . . 15
- 3.2 Coordinate system and conventions . . . 17
- 3.3 Spatial discretization . . . 18
- 3.4 Absorbing boundary conditions . . . 19

4 Input file commands **21**
- 4.1 Essential commands . . . 22
- 4.2 General commands . . . 23
- 4.3 Material commands . . . 24
- 4.4 Object construction commands . . . 27
- 4.5 Source and output commands . . . 35
- 4.6 PML commands . . . 41

5 Output data **43**
- 5.1 Field(s) output . . . 43
- 5.2 Geometry output . . . 46

6 Plotting **49**
- 6.1 A-scans . . . 49
- 6.2 B-scans . . . 49
- 6.3 Antenna parameters . . . 50
- 6.4 Built-in waveforms . . . 50

7 File utilities **57**
- 7.1 inputfile_old2new.py . . . 57
- 7.2 outputfiles_merge.py . . . 57
- 7.3 convert_png2h5.py . . . 57

8 Scripting the input file **59**
- 8.1 Constants/variables . . . 59

8.2 Functions for input commands . . . 59

9 OpenMP, MPI, and HPC **61**
9.1 OpenMP . . . 61
9.2 MPI . . . 61
9.3 HPC job scripts . . . 61

10 GPGPU **67**
10.1 Extra installation steps for GPU usage . . . 67
10.2 Running gprMax using GPU(s) . . . 67

11 GPR antenna models **69**
11.1 Information . . . 69
11.2 Module overview . . . 70
11.3 How to use the module . . . 70

12 Antenna patterns **75**
12.1 Information . . . 75
12.2 Module overview . . . 75
12.3 How to use the module . . . 76

13 AustinMan/AustinWoman **79**
13.1 Information . . . 79
13.2 Package overview . . . 79
13.3 How to use the models . . . 81

14 Materials **83**
14.1 Information . . . 83
14.2 How to use the module . . . 83
14.3 Eccosorb . . . 83

15 Optimisation - Taguchi's method **93**
15.1 Information . . . 93
15.2 Package overview . . . 94
15.3 How to use the package . . . 95

16 Introductory (2D) models **101**
16.1 A-scan from a metal cylinder . . . 101
16.2 B-scan from a metal cylinder . . . 104

17 Antenna models **107**
17.1 Wire dipole antenna model . . . 107
17.2 Bowtie antenna model . . . 108
17.3 B-scan with a bowtie antenna model . . . 111

18 Advanced features **115**
18.1 Building a heterogeneous soil . . . 115

19 FAQs **119**

20 Screencasts & videos **121**
20.1 Installation . . . 121
20.2 Plotting . . . 121
20.3 Visualise EM wave propagation . . . 121

21 Code Overview **123**

22 Analytical comparisons **125**
22.1 Hertzian dipole in free space . . . 125
22.2 Half-wave dipole in free space . . . 127

23 Numerical comparisons **129**
23.1 FDTD/MoM 129

24 Performance benchmarking **131**
24.1 How to benchmark? 131
24.2 Results 132

25 References **137**

Bibliography **139**

(http://docs.gprmax.com/en/latest/?badge=latest)

(http://www.gprmax.com)

CHAPTER 1

Getting Started

1.1 What is gprMax?

gprMax (http://www.gprmax.com) is open source software that simulates electromagnetic wave propagation. It solves Maxwell's equations in 3D using the Finite-Difference Time-Domain (FDTD) method. gprMax was designed for modelling Ground Penetrating Radar (GPR) but can also be used to model electromagnetic wave propagation for many other applications.

gprMax is currently released under the GNU General Public License v3 or higher (http://www.gnu.org/copyleft/gpl.html).

gprMax is principally written in Python (https://www.python.org) 3 with performance-critical parts written in Cython (http://cython.org). It includes a CPU-based solver parallelised using OpenMP (http://www.openmp.org), and a GPU-based solver written using the NVIDIA CUDA (https://developer.nvidia.com/cuda-zone) programming model.

1.1.1 Using gprMax? Cite us

If you use gprMax and publish your work we would be grateful if you could cite our work using:

- Warren, C., Giannopoulos, A., & Giannakis I. (2016). gprMax: Open source software to simulate electromagnetic wave propagation for Ground Penetrating Radar, *Computer Physics Communications* (http://dx.doi.org/10.1016/j.cpc.2016.08.020)

For further information on referencing gprMax visit the Publications section of our website (http://www.gprmax.com/publications.shtml).

1.2 Package overview

```
gprMax/
    conda_env.yml
    CONTRIBUTORS
    docs/
    gprMax/
    LICENSE
```

(continues on next page)

(continued from previous page)

```
README.rst
setup.cfg
setup.py
tests/
tools/
user_libs/
user_models/
```

- `conda_env.yml` is a configuration file for Anaconda (Miniconda) that sets up a Python environment with all the required Python packages for gprMax.
- `CONTRIBUTORS` contains a list of names of people who have contributed to the gprMax codebase.
- `docs` contains source files for the User Guide. The User Guide is written using reStructuredText (http://docutils.sourceforge.net/rst.html) markup, and is built using Sphinx (http://sphinx-doc.org) and Read the Docs (https://readthedocs.org).
- `gprMax` is the main package. Within this package the main module is `gprMax.py`
- `LICENSE` contains information on the GNU General Public License v3 or higher (http://www.gnu.org/copyleft/gpl.html).
- `README.rst` contains getting started information on installation, usage, and new features/changes.
- `setup.cfg` is used to set preference for code formatting/styling using flake8.
- `setup.py` is used to compile the Cython extension modules.
- `tests` is a sub-package which contains test modules and input files.
- `tools` is a sub-package which contains scripts to assist with viewing and post-processing output from models.
- `user_libs` is a sub-package where useful modules contributed by users are stored.
- `user_models` is a sub-package where useful input files contributed by users are stored.

1.3 Installation

The following steps provide guidance on how to install gprMax:

1. Install Python, required Python packages, and get the gprMax source code from GitHub
2. Install a C compiler which supports OpenMP
3. Build and install gprMax

You can watch screencasts (http://docs.gprmax.com/en/latest/screencasts.html) that demonstrate the installation and update processes.

1.3.1 1. Install Python, required Python packages, and get gprMax source

We recommend using Miniconda to install Python and the required Python packages for gprMax in a self-contained Python environment. Miniconda is a mini version of Anaconda which is a completely free Python distribution (including for commercial use and redistribution). It includes more than 300 of the most popular Python packages for science, math, engineering, and data analysis.

- Download and install Miniconda (http://conda.pydata.org/miniconda.html). Choose the Python 3.6 version for your platform (see the Quick Install page (http://conda.pydata.org/docs/install/quick.html) for help installing Miniconda)
- Open a Terminal (Linux/macOS) or Command Prompt (Windows) and run the following commands:

```
$ conda update conda
$ conda install git
$ git clone https://github.com/gprMax/gprMax.git
$ cd gprMax
$ conda env create -f conda_env.yml
```

This will make sure conda is up-to-date, install Git, get the latest gprMax source code from GitHub, and create an environment for gprMax with all the necessary Python packages.

If you prefer to install Python and the required Python packages manually, i.e. without using Anaconda/Miniconda, look in the `conda_env.yml` file for a list of the requirements.

1.3.2 2. Install a C compiler which supports OpenMP

Linux

- gcc (https://gcc.gnu.org) should be already installed, so no action is required.

macOS

- Xcode (the IDE for macOS) comes with the LLVM (clang) compiler, but it does not currently support OpenMP, so you must install gcc (https://gcc.gnu.org). That said, it is still useful to have Xcode (with command line tools) installed. It can be downloaded from the App Store. Once Xcode is installed, download and install the Homebrew package manager (http://brew.sh) and then to install gcc, run:

```
$ brew install gcc
```

Microsoft Windows

- Download and install Microsoft Visual C++ 2015 Build Tools (http://download.microsoft.com/download/5/F/7/5F7ACAEB-8363-451F-9425-68A90F98B238/visualcppbuildtools_full.exe) (currently you must use the 2015 version, not 2017). Use the default installation options.

Alternatively if you are using Windows 10 and feeling adventurous you can install the Windows Subsystem for Linux (https://msdn.microsoft.com/en-gb/commandline/wsl/about) and then follow the Linux install instructions for gprMax. Note however that currently WSL does not aim to support GUI desktops or applications, e.g. Gnome, KDE, etc....

1.3.3 3. Build and install gprMax

Once you have installed the aforementioned tools follow these steps to build and install gprMax:

- Open a Terminal (Linux/macOS) or Command Prompt (Windows), navigate into the top-level gprMax directory, and if it is not already active, activate the gprMax conda environment `conda activate gprMax`. Run the following commands:

```
(gprMax)$ python setup.py build
(gprMax)$ python setup.py install
```

You are now ready to proceed to running gprMax.

1.4 Running gprMax

gprMax is designed as a Python package, i.e. a namespace which can contain multiple packages and modules, much like a directory.

Open a Terminal (Linux/macOS) or Command Prompt (Windows), navigate into the top-level gprMax directory, and if it is not already active, activate the gprMax conda environment `conda activate gprMax`.

Basic usage of gprMax is:

```
(gprMax)$ python -m gprMax path_to/name_of_input_file
```

For example to run one of the test models:

```
(gprMax)$ python -m gprMax user_models/cylinder_Ascan_2D.in
```

When the simulation is complete you can plot the A-scan using:

```
(gprMax)$ python -m tools.plot_Ascan user_models/cylinder_Ascan_2D.out
```

Your results should like those from the A-scan from the metal cylinder example in introductory/basic 2D models section (http://docs.gprmax.com/en/latest/examples_simple_2D.html#view-the-results)

When you are finished using gprMax, the conda environment can be deactivated using `conda deactivate`.

1.4.1 Optional command line arguments

Argument name	Type	Description
`-n`	integer	number of times to run the input file. This option can be used to run a series of models, e.g. to create a B-scan with 60 traces: `(gprMax)$ python -m gprMax user_models/cylinder_Bscan_2D.in -n 60`
`-gpu`	integer	NVIDIA CUDA device ID for a specific GPU card. If not specified will default to device ID 0.
`-restart`	integer	model number to start/restart simulation from. It would typically be used to restart a series of models from a specific model number, with the `-n` argument, e.g. to restart from A-scan 45 when creating a B-scan with 60 traces: `(gprMax)$ python -m gprMax user_models/cylinder_Bscan_2D.in -n 15 -restart 45`
`-task`	integer	task identifier (model number) when running simulation as a job array on Open Grid Scheduler/Grid Engine (http://gridscheduler.sourceforge.net/index.html). For further details see the parallel performance section of the User Guide (http://docs.gprmax.com/en/latest/openmp_mpi.html)
`-mpi`	integer	number of Message Passing Interface (MPI) tasks, i.e. master + workers, for MPI task farm. This option is most usefully combined with `-n` to allow individual models to be farmed out using a MPI task farm, e.g. to create a B-scan with 60 traces and use MPI to farm out each trace: `(gprMax)$ python -m gprMax user_models/cylinder_Bscan_2D.in -n 60 -mpi 61`. For further details see the parallel performance section of the User Guide (http://docs.gprmax.com/en/latest/openmp_mpi.html)
`-benchmark`	flag	switch on benchmarking mode. This can be used to benchmark the threading (parallel) performance of gprMax on different hardware. For further details see the benchmarking section of the User Guide (http://docs.gprmax.com/en/latest/benchmarking.html)
`--geometry-only`	flag	build a model and produce any geometry views but do not run the simulation, e.g. to check the geometry of a model is correct: `(gprMax)$ python -m gprMax user_models/heterogeneous_soil.in --geometry-only`
`--geometry-fixed`	flag	run a series of models where the geometry does not change between models, e.g. a B-scan where *only* the position of simple sources and receivers, moved using `#src_steps` and `#rx_steps`, changes between models.
`--opt-taguchi`	flag	run a series of models using an optimisation process based on Taguchi's method. For further details see the user libraries section of the User Guide (http://docs.gprmax.com/en/latest/user_libs_opt_taguchi.html)
`--write-processed`	flag	write another input file after any Python code and include commands in the original input file have been processed. Useful for checking that any Python code is being correctly processed into gprMax commands.
`-h` or `--help`	flag	used to get help on command line options.

1.5 Updating gprMax

- Open a Terminal (Linux/macOS) or Command Prompt (Windows), navigate into the top-level gprMax directory, and if it is not already active, activate the gprMax conda environment `conda activate gprMax`. Run the following commands:

```
(gprMax)$ git pull
(gprMax)$ python setup.py cleanall
(gprMax)$ python setup.py build
(gprMax)$ python setup.py install
```

This will pull the most recent gprMax source code form GitHub, remove/clean previously built modules, and then build and install the latest version of gprMax.

1.5.1 Updating conda and Python packages

Periodically you should update conda and the required Python packages. With the gprMax environment deactivated and from the top-level gprMax directory, run the following commands:

```
$ conda update conda
$ conda env update -f conda_env.yml
```

CHAPTER 2

Software Features

This section begins with an overview, primarily for previous users, about what is new, what has changed, and what has been retired in this version of gprMax. It is then followed by more general descriptions some of the key features of gprMax that are useful for GPR modelling as well as more general electromagnetic simulations.

2.1 What's new/changed?

A brief summary of what each input command does is given. Please refer to the *Input file commands* section for a detailed description of the syntax of each command.

The code has been completely re-written in Python/Cython. In the process a lot of changes have been made to improve efficiency, speed, and usability. Many new features have been implemented, which have been focussed on the following areas:

- Scripting in the input file
- Built-in library of antenna models
- Anisotropic material modelling
- Dispersive material modelling using multiple pole Debye, Lorentz or Drude formulations
- Building heterogeneous objects using fractal distributions
- Building objects with rough surfaces
- Modelling soils with realistic dielectric and geometric properties
- Improved PML (RIPML) performance

2.1.1 New commands

- `#python` and `#end_python` are used to define blocks of the input file where Python code will be executed. This allows the user to use scripting directly in the input file.
- `#material` replaces `#medium` with a new syntax
- `#add_dispersion_debye` is used to add Debye dispersive properties to a `#material`
- `#add_dispersion_lorentz` is used to add Lorentz dispersive properties to a `#material`

- `#add_dispersion_drude` is used to add Drude dispersive properties to a `#material`
- `#soil_peplinski` is a soil mixing model that can be used with `#fractal_box` to generate soil(s) with more realistic dielectric and geometric properties
- `#cylindrical_sector` is a new object building command
- `#geometry_view` replaces `#geometry_file` or `#geometry_vtk` and is used to create views of the geometry of the model in open source Visualization Toolkit (VTK) (http://www.vtk.org) format which can be viewed in many free readers, such as Paraview (http://www.paraview.org)
- `#fractal_box` is used to create a volume with a fractal distribution of properties
- `#add_surface_roughness` is used to add a rough surface to a `#fractal_box`
- `#add_surface_water` is used to add surface water to a `#fractal_box` that has a rough surface
- `#add_grass` is used to add grass to a `#fractal_box`
- `#waveform` is used to specify a waveform shape, and works in conjunction with the changed source commands
- `#magnetic_dipole` is used to introduce a magnetic dipole source, i.e. current on a small loop
- `#pml_cfs` is an advanced command for adjusting the CFS parameters used in the new PML

2.1.2 Changed commands

- All object building commands support anisotropy via additional material identifiers, e.g. `#box: 0.0 0.0 0.0 0.1 0.1 0.1 matX matY matZ`
- Dielectric smoothing can be turned on (the default setting) or off for any volumetric object building command by specifying a character `y` (on) or `n` (off) after the material identifier, e.g. `#sphere: 0.5 0.5 0.5 0.25 sand n`
- `#triangle` can now create both triangular patches (2D) and triangular prisms (3D) via a thickness parameter
- `#pml_cells` replaces `#pml_layers` and can now be used to control the number of cells of PML on the six faces of the model domain. The number of cells can be set to zero on any of the faces to turn that PML off if desired. The default behaviour (if this command is not specified) is to use 10 cells.
- `#hertzian_dipole` and `#voltage_source` now specify polarisation, location, any additional parameters, and an identifier to link to a `#waveform` command
- `#snapshot` no longer requires a type, as only VTK snapshot files are now produced
- `#num_of_procs` is now called `#num_threads`
- `#tx_steps` is now called `#src_steps`

2.1.3 Retired commands

- `#analysis` and `#end_analysis` are no longer required as: sources and receivers can be specified anywhere is the input file; the output file automatically has the same name as the input file with a `.out` extension; the format of the output file is now HDF5 (http://www.hdfgroup.org/HDF5/); gprMax can be run with the syntax `gprMax my_input_file -n number_of_runs`, where `number_of_runs` can be used to rerun the model for creating scans and/or moving geometry between runs.
- `#tx` is no longer required as the polarisation and position of a source is now specified in the source command, e.g. `#hertzian_dipole: y 0.05 0.05 0.05 myPulse`
- `#cylinder_new` has become `#cylinder`
- `#cylindrical_segment` was under-used and its effect can be created by cutting a `#cylinder` with a `#box`

- `#bowtie` can be created using the new behaviour of `#triangle`
- `#number_of_media` is not required with the new Python code
- `#nips_number` is not required with the new Python code
- `#media_file` was under-used
- `#geometry_file` has been replaced with the `#geometry_view` command
- `#medium` has been replaced with the `#material` command
- `#abc_type`, `#abc_order`, `#abc_stability_factors`, `#abc_optimization_angles`, and `#abc_mixing_parameters` are not required as the Higdon ABCs have been removed
- `#huygens_surface` will become part of the new `#plane_wave` command which is yet to be implemented

2.1.4 Commands yet to be implemented

There are commands from previous versions of gprMax that are planned for this version, but are yet to be implemented. These will be introduced in a future update. They are `#thin_wire` and `#plane_wave`.

2.1.5 Migrating old input files

gprMax includes a Python module (in the `tools` package) to help you migrate old input files, written for previous versions of gprMax, to the syntax of the new commands. The module will do its best to convert the old file and write a new one, however, you should still carefully check the new file to make sure it is what you intended! Usage (from the top-level gprMax directory) is: `python -m tools.inputfile_old2new my_old_inputfile.in`.

2.2 Key features

2.2.1 Python scriptable input files

The input file has now been made scriptable by permitting blocks of Python code to be specified between `#python` and `#end_python` commands. The code is executed when the input file is read by gprMax. You don't need any external tools, such as MATLAB, to generate larger, more complex input files for building intricate models. Python scripting means that gprMax now includes *libraries of more complex objects, such as antennas*, that can be easily inserted into a model. You can also access a number of built-in constants from your Python code. For further details see the *Python section*.

2.2.2 Dispersive materials

gprMax has always included the ability to represent dispersive materials using a single-pole Debye model. Many materials can be adequately represented using this approach for the typical frequency ranges associated with GPR. However, multi-pole Debye, Drude and Lorentz functions are often used to simulate the electric susceptibility of materials such as: water *[PIE2009]*, human tissue *[IRE2013]*, cold plasma *[LI2013]*, gold *[VIA2005]*, and soils *[BER1998]*, *[GIAK2012]*, *[TEI1998]*. Electric susceptibility relates the polarization density to the electric field, and includes both the real and imaginary parts of the complex electric permittivity variation. In the new version of gprMax a recursive convolution based method is used to express dispersive properties as apparent current density sources *[GIA2014]*. A major advantage of this implementation is that it creates an inclusive susceptibility function that holds, as special cases, Debye, Drude and Lorentz materials. For further details see the *material commands section*.

2.2.3 Realistic soils, heterogeneous objects and rough surfaces

The inclusion of improved models of soils is important for many GPR simulations. gprMax can now be used to create soils with more realistic dielectric and geometrical properties. A semi-empirical model, initially suggested by *[DOB1985]*, is used to describe the dielectric properties of the soil. The model relates relative permittivity of the soil to bulk density, sand particle density, sand fraction, clay fraction and water volumetric fraction. Using this approach, a more realistic soil with a stochastic distribution of the aforementioned parameters can be modelled. The real and imaginary parts of this semi-empirical model can be approximated using a multi-pole Debye function plus a conductive term. This can now be achieved in gprMax using the new dispersive material functionality. For further details see the *material commands section*.

Fractals are scale invariant functions which can express the topography of the earth for a wide range of scales with sufficient detail *[TUR1987]*. For this reason fractals have been chosen to represent the topography of soils. Fractals can be generated by the convolution of Gaussian noise with an inverse Fourier transform of $\frac{1}{kb}$, where k is the wavenumber and b is a constant related to the fractal dimension *[TUR1997]*. gprMax can now generate heterogeneous volumes (boxes) with realistic soil properties that can have rough surfaces applied. For further details see the *fractal object building commands section*.

Fractal correlated noise *[TUR1997]* is used to describe the stochastic distribution of the properties of soils. This approach has been chosen because it has been shown that soil-related environmental properties frequently obey fractal laws *[BUR1981]*, *[HILL1998]*. For further details see the *material commands section* and the *fractal object building commands section*.

2.2.4 Library of antenna models

gprMax now includes Python modules with pre-defined models of antennas that behave similarly to commercial antennas *[WAR2011] [STA2017]*. Currently models of antennas similar to Geophysical Survey Systems, Inc. (GSSI) (http://www.geophysical.com) 1.5 GHz (Model 5100) antenna, and 400 MHz antenna, as well as MALA Geoscience (http://www.malags.com/) 1.2 GHz antenna are included. By taking advantage of Python scripting in input files, using such complex structures in a model is straightforward without having to be built step-by-step by the user. For further details see the *Python section*.

2.2.5 Anisotropic materials

It is possible to specify objects that have diagonal anisotropy which allows materials such as wood and fibre-reinforced composites, often imaged with GPR, to be more accurately modelled. Standard isotropic objects specify one material identifier that defines the same properties in x, y, and z directions. However, every volumetric object building command can also be specified with three material identifiers, which allows properties for the x, y, and z directions to be separately defined.

2.2.6 Dielectric smoothing

At the boundaries between different materials in a model there is the question of what electric and magnetic material properties to use?

- Should the last object to be defined at that location dictate the electric and magnetic properties?
- Should an average set of electric and magnetic properties of the materials of the objects that share that location be used?

This latter option is often referred to as dielectric smoothing and has been shown to result in more accurate simulations *[LUE1994] [BOU1996] [WHI2009]*. To address this question gprMax includes an option to turn dielectric smoothing on or off for volumetric object building commands. The default behaviour (if no option is specified) is for dielectric smoothing to be on. The option can be specified with a single character `y` (on) or `n` (off) given after the material identifier in each object command. When dielectric smoothing is on gprMax calculates the arithmetic mean of the electric and magnetic properties of the surrounding Yee cells, to use for the single Yee cell edge (boundary) of interest.

2.2.7 Perfectly Matched Layer (PML) boundary conditions

With increased research into quantitative information from GPR, it has become necessary for models to be able to have more efficient and better-performing Perfectly Matched Layer (PML) absorbing boundary conditions. Since 2005 gprMax has featured PML absorbing boundary conditions based on the uniaxial PML (UPML) *[GED1998]* formulation. A PML based on a recursive integration approach to the complex frequency shifted (CFS) PML *[GIA2012]* has been adopted in the new version of gprMax. A general formulation of this RIPML, which can be used to develop any order of PML, has been used to implement first and second order CFS stretching functions. One of the attractions of the RIPML is that it is easily applied as a correction to the field quantities after the complete FDTD grid has been updated using the standard FDTD update equations. gprMax now offers the ability (for advanced users) to customise the parameters of the PML which allows its performance to be better optimised for specific applications. Additionally, since the RIPML is media agnostic it can be used without change to problems involving dispersive and anisotropic materials. For further details see the *PML commands section*.

2.2.8 Open source, robust, file formats

Alongside improvements to the input file there is a new output file format – HDF5 (http://www.hdfgroup.org/HDF5/) – to manage the larger and more complex data sets that are being generated. HDF5 is a robust, portable and extensible format with a number of free readers available. For further details see the *output file section*.

In addition, the Visualization Toolkit (VTK) (http://www.vtk.org) is being used for improved handling and viewing of the detailed 3D FDTD geometry meshes. The VTK is an open-source system for 3D computer graphics, image processing and visualisation. It also has a number of free readers available including Paraview (http://www.paraview.org). For further details see the *geometry view command*.

CHAPTER 3

Guidance on GPR modelling

In order to make the most of gprMax for modelling GPR you should be familiar with the Finite-Difference Time-Domain (FDTD) method method on which the software is based.

This section discusses some basic concepts of the FDTD method and GPR modelling. There is a large amount of further information available in the relevant literature. Good starting points are *[KUN1993]* and *[TAF2005]*, and the specific application of FDTD to the GPR forward problem is described in *[GIA1997]*.

3.1 Basic concepts

All electromagnetic phenomena, on a macroscopic scale, are described by the well-known Maxwell's equations. These are first order partial differential equations which express the relations between the fundamental electromagnetic field quantities and their dependence on their sources.

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$
$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J_c} + \mathbf{J_s}$$
$$\nabla \cdot \mathbf{B} = 0$$
$$\nabla \cdot \mathbf{D} = q_v$$

where t is time (seconds) and q_v is the volume electric charge density (coulombs/cubic metre). In Maxwell's equations, the field vectors are assumed to be single-valued, bounded, continuous functions of position and time. In order to simulate the GPR response from a particular target or set of targets the above equations have to be solved subject to the geometry of the problem and the initial conditions.

The nature of the GPR forward problem classifies it as an *initial value – open boundary* problem. This means that in order to obtain a solution you have to define an initial condition (i.e. excitation of the GPR transmitting antenna) and allow for the resulting fields to propagate through space reaching a zero value at infinity since, there is no specific boundary which limits the geometry of the problem and where the electromagnetic fields can take a predetermined value. Although the first part is easy to accommodate (i.e. specification of the source), the second part cannot be easily tackled using a finite computational space.

The FDTD approach to the numerical solution of Maxwell's equations is to discretize both the space and time continua. Thus the discretization spatial Δx, Δy and Δz and temporal Δt steps play a very significant role – since the smaller they are the closer the FDTD model is to a real representation of the problem. However, the values of the discretization steps always have to be finite, since computers have a limited amount of storage and

finite processing speed. Hence, the FDTD model represents a discretized version of the real problem and is of limited size. The building block of this discretized FDTD grid is the Yee cell *[YEE1966]* named after Kane Yee who pioneered the FDTD method. This is illustrated for the 3D case in Fig. 3.1.

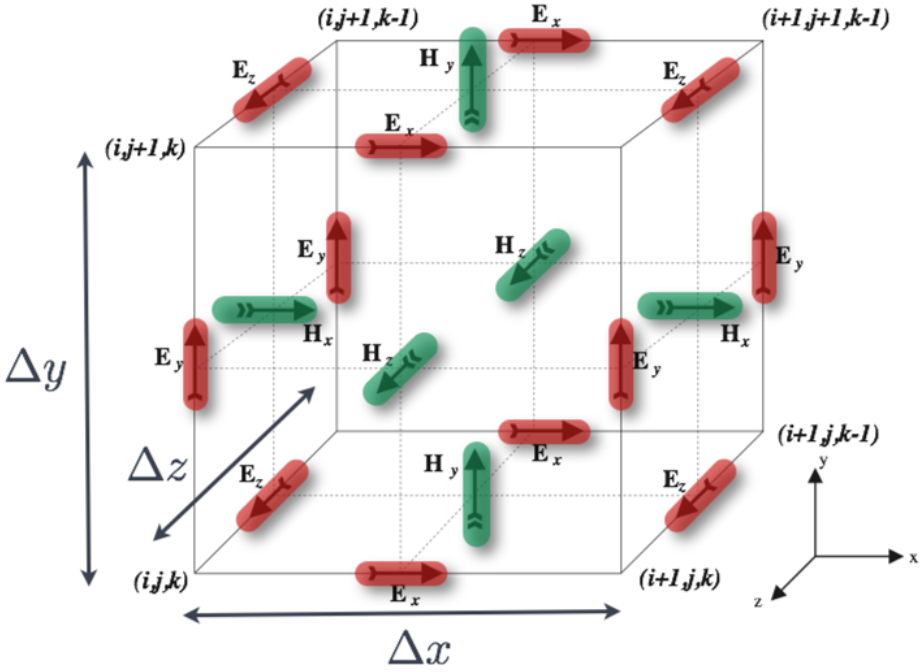


Fig. 3.1: Single FDTD Yee cell showing electric (red) and magnetic (green) field components.

By assigning appropriate constitutive parameters to the locations of the electromagnetic field components complex shaped targets can be included easily in the models. However, objects with curved boundaries are represented using a staircase approximation.

gprMax is fundamentally based on solving Maxwell's equations in 3D using the FDTD method - transverse electromagnetic (TEM) mode. However, it can also be used to carry out simulations in 2D using the transverse magnetic (TM) mode. This is achieved through specifying a single cell slice of the domain, i.e. one dimension of the domain must be equal to the spatial discretization in that direction. When this occurs the electric and magnetic field components on the two faces of single cell slice in the invariant direction are set to zero. This is illustrated for the 2D TMz case in Fig. 3.2.

Using this approach means that Maxwell's equations in 3D, shown in (3.1) as six coupled partial differential equations, reduce to the corresponding 2D form - in this case 2D TMz, shown in (3.2).

$$
\begin{aligned}
\frac{\partial E_x}{\partial t} &= \frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - J_{Sx} - \sigma E_x\right) \\
\frac{\partial E_y}{\partial t} &= \frac{1}{\epsilon}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - J_{Sy} - \sigma E_y\right) \\
\frac{\partial E_z}{\partial t} &= \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - J_{Sz} - \sigma E_z\right) \\
\frac{\partial H_x}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - M_{Sx} - \sigma^* H_x\right) \\
\frac{\partial H_y}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} - M_{Sy} - \sigma^* H_y\right) \\
\frac{\partial H_z}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - M_{Sz} - \sigma^* H_z\right)
\end{aligned}
\tag{3.1}
$$

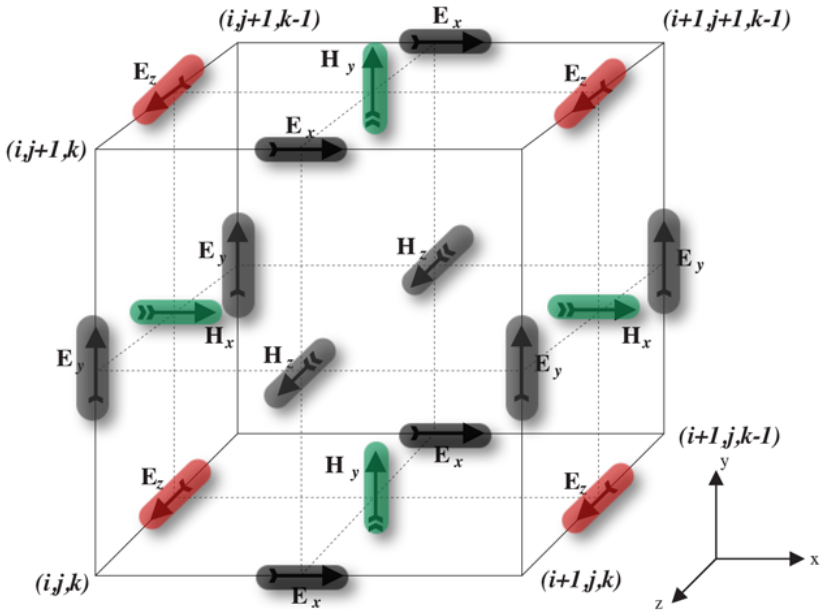


Fig. 3.2: Single FDTD Yee cell showing electric (red), magnetic (green), and zeroed out (grey) field components for 2D transverse magnetic (TM) z-direction mode.

$$
\begin{aligned}
\frac{\partial E_z}{\partial t} &= \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - J_{Sz} - \sigma E_z\right) \\
\frac{\partial H_x}{\partial t} &= \frac{1}{\mu}\left(-\frac{\partial E_z}{\partial y} - M_{Sx} - \sigma^* H_x\right) \\
\frac{\partial H_y}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - M_{Sy} - \sigma^* H_y\right)
\end{aligned}
\tag{3.2}
$$

These equations are discretized in both space and time and applied in each FDTD cell. The numerical solution is obtained directly in the time domain in an iterative fashion. In each iteration the electromagnetic fields advance (propagate) in the FDTD grid and each iteration corresponds to an elapsed simulated time of one Δt. Hence by specifying the number of iterations you can instruct the FDTD solver to simulate the fields for a given time window.

The price you have to pay for obtaining a solution directly in the time domain using the FDTD method is that the values of Δx, Δy, Δz and Δt can not be assigned independently. FDTD is a conditionally stable numerical process. The stability condition is known as the CFL condition after the initials of Courant, Freidrichs and Lewy and is given by,

$$
\Delta t \leq \frac{1}{c\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2}}},
$$

where c is the speed of light. Hence Δt is bounded by the values of Δx, Δy and Δz. The stability condition for the 2D case is easily obtained by letting $\Delta z \longrightarrow \infty$.

3.2 Coordinate system and conventions

A right-handed Cartesian coordinate system is used with the origin of space coordinates in the *lower left corner* at (0,0,0). Fig. 3.3 illustrates the coordinate system of gprMax. Only one row of cells in the x direction is depicted. The space coordinates range from the left edge of the first cell to the right edge of the last one. Assuming that $\Delta x = 1$ metre, if you wanted to allocate a rectangle with its x dimension equal to 3 metres and its lower x

coordinate at 1 then the x range would be [1..4]. The 3D cells allocated by gprMax would be [1..3]. In the 3D FDTD cell there are no field components located at the centre of the cell. Electric field components are tangential to, and magnetic field components normal to the interfaces between cells. The field components depicted in Fig. 3.3 correspond to space coordinate 1. Source and output points defined in space coordinates are directly converted to cell coordinates and the corresponding field components.

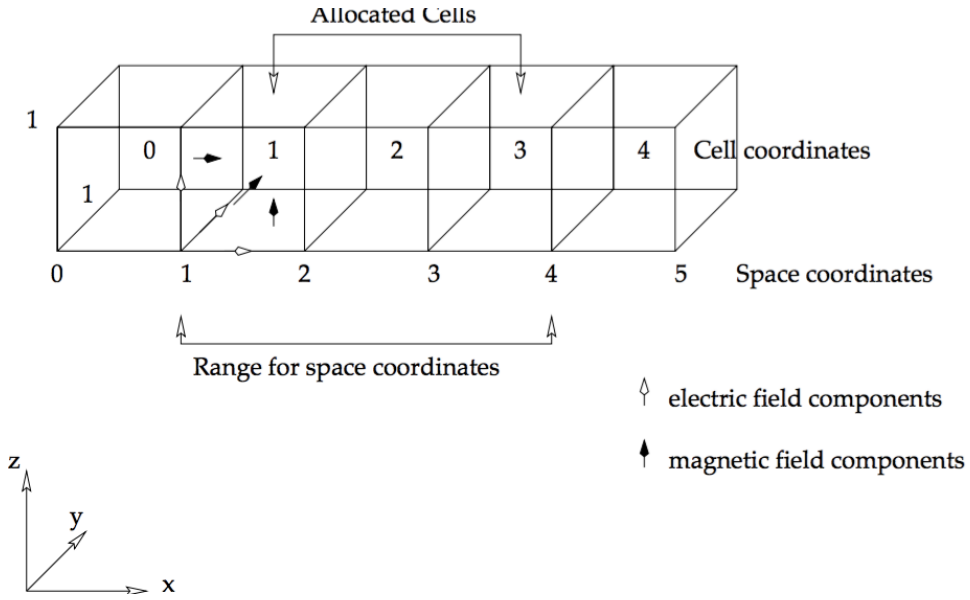


Fig. 3.3: gprMax coordinate system and conventions.

The actual positions of field components for a given set of space coordinates (x, y, z) are:

$$E_x\ (x + \frac{\Delta x}{2}, y, z)$$
$$E_y\ (x, y + \frac{\Delta y}{2}, z)$$
$$E_z\ (x, y, z + \frac{\Delta z}{2})$$
$$H_x\ (x, y + \frac{\Delta y}{2}, z + \frac{\Delta z}{2})$$
$$H_y\ (x + \frac{\Delta x}{2}, y, z + \frac{\Delta z}{2})$$
$$H_z\ (x + \frac{\Delta x}{2}, y + \frac{\Delta y}{2}, z)$$

Hertzian dipole sources as well as other electric field excitations (i.e. voltage sources, transmission lines) are located at the corresponding electric field components.

3.3 Spatial discretization

There is no specific guideline for choosing the right spatial discretization for a given problem. In general, it depends on the required accuracy, the frequency content of the source pulse and the size of the targets. Obviously, all targets present in a model must be adequately resolved. This means, for example, that a cylinder with radius equal to one or two spatial steps does not really look like a cylinder!

An other important factor which influences the spatial discretization is the errors associated with numerically induced dispersion. This means that contrary to the real world where electromagnetic waves propagate with the same velocity irrespectively of their direction and frequency (assuming no dispersive media and far-field conditions) in the discrete one this is not the case. This error (details can be found in *[GIA1997]* and *[KUN1993]*) can be kept in a minimum if the following *rule-of-thumb* is satisfied:

The discretization step should be at least ten times smaller than the smallest wavelength of the propagating electromagnetic fields.

$$\Delta l = \frac{\lambda}{10}$$

Note that in general low-loss media wavelengths are much smaller compared to free space.

3.4 Absorbing boundary conditions

One of the most challenging issues in modelling *open boundary* problems, such as GPR, is the truncation of the computational domain at a finite distance from sources and targets where the values of the electromagnetic fields can not be calculated directly by the numerical method applied inside the model. Hence, an approximate condition known as *absorbing boundary condition (ABC)* is applied at a sufficient distance from the source to truncate and therefore limit the computational space. The role of this ABC is to absorb any waves impinging on it, hence simulating an unbounded space. The computational space (i.e the model) limited by the ABCs should contain all important features of the model such as sources and output points and targets. Fig. 3.4 illustrates this basic difference between the problem to be modelled and the actual FDTD modelled space.

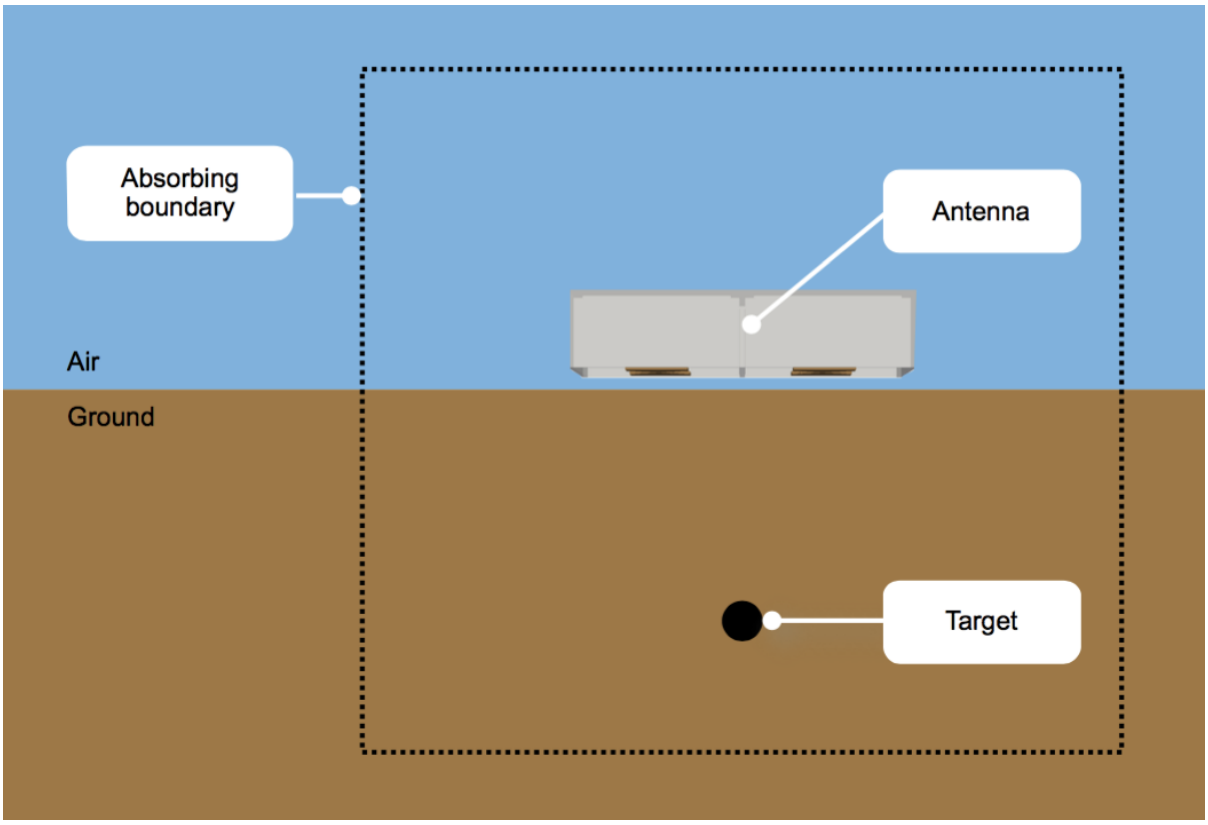


Fig. 3.4: GPR forward problem showing computational domain bounded by Absorbing Boundary Conditions (ABCs)

It is assumed that the half-space which contains the target(s) is of infinite extent. Therefore, the only reflected waves will be the ones originating from the target. In cases where the host medium is not of infinite extent (e.g. a finite concrete slab) the assumption of infinite extent can be made as far as the actual reflections from the slab termination are not of interest or its actual size is large enough that any reflected waves which will originate at its termination will not affect the solution for the required time window. In general, any objects that span the size of the computational domain (i.e. model) are assumed to extend to infinity. The only reflections which will originate from their termination at the truncation boundaries of the model are due to imperfections of the ABCs and in general are of a very small amplitude compared with the reflections from target(s) inside the model.

The ABCs employed in gprMax will, in general, perform well (i.e. without introducing significant artificial reflections) if all sources and targets are kept at least 15 cells away from them. gprMax uses Perfectly Matched Layer (PML) ABCs based on a recursive integration approach to the complex frequency shifted (CFS) PML *[GIA2012]*. A general formulation of this RIPML, which can be used to develop any order of PML, has been used to implement first and second order CFS stretching functions. One of the attractions of the RIPML is that it is easily applied as a correction to the field quantities after the complete FDTD grid has been updated using the standard FDTD update equations.

The cells of the RIPML, which have a user adjustable thickness, very efficiently absorb most waves that propagate in them. Although, source and output points can be specified inside these cells **it is wrong to do so** from the point of view of correct modelling. The fields inside these cells are not of interest to GPR modelling. Placing sources inside these cells could have effects that have not been studied and will certainly provide erroneous results from the perspective of GPR modelling. The requirement to keep sources and targets at least 15 cells away for the PML has to be taken into account when deciding the size of the model domain. Additionally, free space (i.e. air) should be always included above a source for at least 15-20 cells in GPR models. Obviously, the more cells there are between observation points, sources, targets and the absorbing boundaries, the better the results will be.

gprMax now offers the ability (for advanced users) to customise the parameters of the PML which allows its performance to be better optimised for specific applications. For further details see the *PML commands section*.

All other *boundary conditions* which apply at interfaces between different media in the FDTD model are automatically enforced in gprMax.

CHAPTER 4

Input file commands

An input file has to be supplied to gprMax which should contain all the necessary information to run a GPR model. The input file is an ASCII text file which can be prepared with any text editor or word-processing program. In the input file the hash character (#) is reserved and is used to denote the beginning of a command which will be passed to gprMax. The general syntax of commands is:

```
#command_name: parameter1 parameter2 parameter3 ...
```

A command and associated parameters should occupy a single line of the input file, and only one command per line is allowed. Hence, the first character of a line containing a command **must** be the hash character (#). If the line starts with **any other character** it is ignored by the program. Therefore, user comments or descriptions can be included in the input file. If a line starts with a hash character (#) the program will expect a valid command. If the name of the command is not correct the program will abandon execution and issue an error message. When a command requires more than one parameter then these should be separated using a white space character.

The order of commands in the input file is not important with the exception of object construction commands.

To describe the commands that can be used in the input file and their parameters the following conventions are used:

- `f` means a real number which can be entered using either a `[.]` separating the integral from the decimal part, e.g. 1.5, or in scientific notation, e.g. 15e-1 or 0.15e1.
- `i` means an integer number.
- `c` means a single character, e.g. `y`.
- `str` means a string of characters with **no** white spaces in between, e.g `sand`.
- `file` means a filename.
- `[ ]` square brackets are used to indicate optional parameters.

Unless otherwise specified, the SI system of units is used throughout gprMax:

- All parameters associated with simulated space (i.e. size of model, spatial increments, etc...) should be specified in **metres**.
- All parameters associated with time (i.e. total simulation time, time instants, etc...) should be specified in **seconds**.
- All parameters denoting frequency should be specified in **Hertz**.

- All parameters associated with spatial coordinates in the model should be specified in **metres**. The origin of the coordinate system **(0,0)** is at the lower left corner of the model.

It is important to note that gprMax converts spatial and temporal parameters given in **metres** and **seconds** to integer values corresponding to **FDTD cell coordinates** and **iteration number** respectively. Therefore, rounding to the nearest integer number of the user defined values is performed.

The fundamental spatial and temporal discretization steps are denoted as Δx , Δy, Δz and Δt respectively.

The commands have been grouped into six categories:

- **Essential** - required to run any model, such as the domain size and spatial discretization
- **General** - provide further control over the model
- **Material** - used to introduce different materials into the model
- **Object construction** - used to build geometric shapes with different constitutive parameters
- **Source and output** - used to place source and output points in the model
- **PML** - provide advanced customisation and optimisation of the absorbing boundary conditions

4.1 Essential commands

Most of the commands are optional but there are some essential commands which are necessary in order to construct any model. For example, none of the media and object commands are necessary to run a model. However, without specifying any objects in the model gprMax will simulate free space (air), which on its own, is not particularly useful for GPR modelling. If you have not specified a command which is essential in order to run a model, for example the size of the model, gprMax will terminate execution and issue an appropriate error message.

The essential commands are:

4.1.1 #domain:

Allows you to specify the size of the model. The syntax of the command is:

```
#domain: f1 f2 f3
```

where `f1 f2 f3` are the size of the model in the x, y, and z directions respectively. For example to specify a 500 x 500 x 1000mm model use: `#domain: 0.5 0.5 1.0`

4.1.2 #dx_dy_dz:

Allows you to specify the discretization of space in the x , y and z directions respectively (i.e. Δx , Δy, Δz). The syntax of the command is:

```
#dx_dy_dz: f1 f2 f3
```

where `f1` is the spatial step in the x direction (Δx), `f2` is the spatial step in the y direction (Δy) and `f3` is the spatial step in the z direction (Δz). The spatial discretization controls the maximum permissible time step Δt with which the solution advances in time in order to reach the required simulated time window. The relation between Δt and Δx , Δy, Δz is:

$$\Delta t \leq \frac{1}{c\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2}}},$$

where c is the speed of light. In gprMax the equality is used to determine Δt from Δx , Δy, and Δz. Small values of Δx , Δy, and Δz result in small values for Δt which means more iterations in order to reach a given simulated time. However, it is important to note that the smaller the values of Δx , Δy, Δz and Δt are the more accurate your model will be. See the *Guidance on GPR modelling* section for tips on choosing a spatial discretisation.

4.1.3 #time_window:

Allows you to specify the total required simulated time. The syntax of the command is:

```
#time_window: f1
```

or

```
#time_window: i1
```

In the first case the `f1` parameter determines the required simulated time in seconds. For example, if you want to simulate a GPR trace of 20 nanoseconds then `#time_window: 20e-9` can be used. gprMax will perform the necessary number of iterations in order to reach the required simulated time. Alternatively, if the command is specified with an `i1` gprMax will interpret this value as a total number of iterations. Hence the command `#time_window: 100` means that 100 iterations will be performed. The number of iterations and the total simulated time window are related by:

$$t_w = \Delta t N_{it},$$

where t_w is the time window in seconds, Δt the time step, and N_{it} the number of iterations. gprMax converts the specified time window in seconds to a number of iterations internally using the aforementioned equation. The result of the division is rounded to the nearest integer.

4.2 General commands

4.2.1 #python: and #end_python:

Allows you to write blocks of Python code between `#python` and `#end_python` in the input file. The code is executed when the input file is read by gprMax. For further details see the *Python section*.

4.2.2 #include_file:

Allows you to include commands from a file. It will insert the commands from the specified file at the location where the `#include_file` command is placed. The syntax of the command is:

```
#include_file: file1
```

`file1` can be the name of the file containing the commands in the same directory as the input file, or `file` can be the full path to the file containing the commands (allowing you to specify any location).

4.2.3 #time_step_stability_factor:

Allows you to alter the value of the time step Δt used by gprMax. gprMax uses the equality in the CFL condition, hence the maximum permissible time step. If a smaller time step is required then the syntax of the command is:

```
#time_step_stability_factor: f1
```

where `f1` can take values $0 < \text{f1} \leq 1$. Then the actual time step used will be $\text{f1} \times \Delta t$, where Δt is calculated using the equality from the CFL condition.

4.2.4 #title:

Allows you to include a title for your model. This title is saved in the output file(s). The syntax of the command is:

```
#title: str1
```

where `str1` can contain white space characters to separate individual words. The title has to be contained in a single line.

4.2.5 #messages:

Allows you to control the amount of information displayed on screen when gprMax is run. The syntax of the command is:

```
#messages: c1
```

where `c1` can be either y (yes) or n (no) which turns on or off the messages on the screen. The default value is y. When messages are on, gprMax will display on the screen information the translation of space and time values to cell coordinates, iteration number, material parameters etc... This information can be useful for error checking.

4.2.6 #output_dir:

Allows you to control the directory where output file(s) will be stored. The syntax of the command is:

```
#output_dir: str1
```

where `str1` can be either the absolute path to the directory for the output file(s) or a path relative to the directory of the input files. The default value is the same as the directory of the input files.

4.2.7 #num_threads:

Allows you to control how many OpenMP threads (usually the number of physical CPU cores available) are used when running the model. The most computationally intensive parts of gprMax, which are the FDTD solver loops, have been parallelised using OpenMP (http://openmp.org) which supports multi-platform shared memory multiprocessing. The syntax of the command is:

```
#num_threads: i1
```

where `i1` is the number of OpenMP threads to use. If `#num_threads` is not specified gprMax will firstly look to see if the environment variable `OMP_NUM_THREADS` exists, and if not will detect and use all available physical CPU cores on the machine.

4.3 Material commands

4.3.1 Built-in materials

gprMax has two builtin materials which can be used by specifying the identifiers `pec` and `free_space`. These simulate a perfect electric conductor and air, i.e. a non-magnetic material with $\epsilon_r = 1$, $\sigma = 0$, respectively. Additionally the identifiers `grass` and `water` are currently reserved for internal use and should not be used unless you intentionally want to change their properties.

4.3.2 #material:

Allows you to introduce a material into the model described by a set of constitutive parameters. The syntax of the command is:

```
#material: f1 f2 f3 f4 str1
```

- `f1` is the relative permittivity, ϵ_r
- `f2` is the conductivity (Siemens/metre), σ
- `f3` is the relative permeability, μ_r
- `f4` is the magnetic loss (Ohms/metre), σ_*
- `str1` is an identifier for the material.

For example `#material: 3 0.01 1 0 my_sand` creates a material called `my_sand` which has a relative permittivity (frequency independent) of $\epsilon_r = 3$, a conductivity of $\sigma = 0.01$ S/m, and is non-magnetic, i.e. $\mu_r = 1$ and $\sigma_* = 0$

4.3.3 #add_dispersion_debye:

Allows you to add dispersive properties to an already defined `#material` based on a multiple pole Debye formulation (see *Software Features* section). For example, the susceptability function for a single-pole Debye material is given by:

$$\chi_p(t) = \frac{\Delta\epsilon_{rp}}{\tau_p} e^{-t/\tau_p},$$

where $\Delta\epsilon_{rp} = \epsilon_{rsp} - \epsilon_{r\infty}$, ϵ_{rsp} is the zero-frequency relative permittivity for the pole, $\epsilon_{r\infty}$ is the relative permittivity at infinite frequency, and τ_p is the pole relaxation time.

The syntax of the command is:

```
#add_dispersion_debye: i1 f1 f2 f3 f4 ... str1
```

- `i1` is the number of Debye poles.
- `f1` is the difference between the zero-frequency relative permittivity and the relative permittivity at infinite frequency, i.e. $\Delta\epsilon_{rp1} = \epsilon_{rsp1} - \epsilon_{r\infty}$, for the first Debye pole.
- `f2` is the relaxation time (seconds), τ_{p1}, for the first Debye pole.
- `f3` is the difference between the zero-frequency relative permittivity and the relative permittivity at infinite frequency, i.e. $\Delta\epsilon_{rp2} = \epsilon_{rsp2} - \epsilon_{r\infty}$, for the second Debye pole.
- `f4` is the relaxation time (seconds), τ_{p2}, for the second Debye pole.
- ...
- `str1` identifies the material to add the dispersive properties to.

For example to create a model of water with a single Debye pole, $\epsilon_{rsp1} = 80.1$, $\epsilon_{r\infty} = 4.9$ and $\tau_{p1} = 9.231 \times 10^{-12}$ seconds use: `#material: 4.9 0 1 0 my_water` and `#add_dispersion_debye: 1 75.2 9.231e-12 my_water`.

Note:

- You can continue to add pairs of values for $\Delta\epsilon_{rp}$ and τ_p for as many Debye poles as you have specified with `i1`.
- The relative permittivity in the `#material` command should be given as the relative permittivity at infinite frequency, i.e. $\epsilon_{r\infty}$.
- Temporal values associated with pole frequencies and relaxation times should always be greater than the time step Δt used in the model.

4.3.4 #add_dispersion_lorentz:

Allows you to add dispersive properties to an already defined `#material` based on a multiple pole Lorentz formulation (see *Software Features* section). For example, the susceptability function for a single-pole Lorentz material is given by:

$$\chi_p(t) = \Re\left\{-j\gamma_p e^{(-\delta_p + j\beta_p)t}\right\},$$

where

$$\beta_p = \sqrt{\omega_p^2 - \delta_p^2} \quad \text{and} \quad \gamma_p = \frac{\omega_p^2 \Delta\epsilon_{rp}}{\beta_p},$$

where $\Delta\epsilon_{rp} = \epsilon_{rsp} - \epsilon_{r\infty}$, ϵ_{rsp} is the zero-frequency relative permittivity for the pole, $\epsilon_{r\infty}$ is the relative permittivity at infinite frequency, ω_p is the frequency (Hertz) of the pole pair, δ_p is the damping coefficient (Hertz) , and $j = \sqrt{-1}$.

The syntax of the command is:

```
#add_dispersion_lorentz: i1 f1 f2 f3 f4 f5 f6 ... str1
```

- `i1` is the number of Lorentz poles.
- `f1` is the difference between the zero-frequency relative permittivity and the relative permittivity at infinite frequency, i.e. $\Delta\epsilon_{rp1} = \epsilon_{rsp1} - \epsilon_{r\infty}$, for the first Lorentz pole.
- `f2` is the frequency (Hertz), ω_{p1}, for the first Lorentz pole.
- `f3` is the damping coefficient (Hertz), δ_{p1}, for the first Lorentz pole.
- `f4` is the difference between the zero-frequency relative permittivity and the relative permittivity at infinite frequency, i.e. $\Delta\epsilon_{rp2} = \epsilon_{rsp2} - \epsilon_{r\infty}$, for the second Lorentz pole.
- `f5` is the frequency (Hertz), ω_{p2}, for the second Lorentz pole.
- `f6` is the damping coefficient (Hertz), δ_{p2}, for the second Lorentz pole.
- ...
- `str1` identifies the material to add the dispersive properties to.

Note:

- You can continue to add triplets of values for $\Delta\epsilon_{rp}$, ω_p and δ_p for as many Lorentz poles as you have specified with `i1`.
- The relative permittivity in the `#material` command should be given as the relative permittivity at infinite frequency, i.e. $\epsilon_{r\infty}$.
- Temporal values associated with pole frequencies and relaxation times should always be greater than the time step Δt used in the model.

4.3.5 #add_dispersion_drude:

Allows you to add dispersive properties to an already defined `#material` based on a multiple pole Drude formulation (see *Software Features* section). For example, the susceptability function for a single-pole Drude material is given by:

$$\chi_p(t) = \frac{\omega_p^2}{\gamma_p}(1 - e^{-\gamma_p t}),$$

where ω_p is the frequency (Hertz) of the pole, and γ_p is the inverse of the pole relaxation time (Hertz).

The syntax of the command is:

```
#add_dispersion_drude: i1 f1 f2 f3 f4 ... str1
```

- `i1` is the number of Drude poles.
- `f1` is the frequency (Hertz), ω_{p1}, for the first Drude pole.
- `f2` is the inverse of the relaxation time (Hertz), γ_{p1}, for the first Drude pole.
- `f3` is the frequency (Hertz), ω_{p2}, for the second Drude pole.
- `f4` is the inverse of the relaxation time (Hertz), γ_{p2} for the second Drude pole.
- ...
- `str1` identifies the material to add the dispersive properties to.

Note:

- You can continue to add pairs of values for ω_p and γ_p for as many Drude poles as you have specified with `i1`.
- Temporal values associated with pole frequencies and relaxation times should always be greater than the time step Δt used in the model.

4.3.6 #soil_peplinski:

Allows you to use a mixing model for soils proposed by Peplinski (http://dx.doi.org/10.1109/36.387598), valid for frequencies in the range 0.3GHz to 1.3GHz. The command is designed to be used in conjunction with the `#fractal_box` command for creating soils with realistic dielectric and geometric properties. The syntax of the command is:

```
#soil_peplinski: f1 f2 f3 f4 f5 f6 str1
```

- `f1` is the sand fraction of the soil.
- `f2` is the clay fraction of the soil.
- `f3` is the bulk density of the soil in grams per centimetre cubed.
- `f4` is the density of the sand particles in the soil in grams per centimetre cubed.
- `f5` and `f6` define a range for the volumetric water fraction of the soil.
- `str1` is an identifier for the soil.

For example for a soil with sand fraction 0.5, clay fraction 0.5, bulk density 2 g/cm^3, sand particle density of 2.66 g/cm^3, and a volumetric water fraction range of 0.001 - 0.25 use: `#soil_peplinski: 0.5 0.5 2.0 2.66 0.001 0.25 my_soil`.

Note: Further information on the Peplinski soil model and our implementation can be found in 'Giannakis, I. (2016). Realistic numerical modelling of Ground Penetrating Radar for landmine detection. The University of Edinburgh. (http://hdl.handle.net/1842/20449)'

4.4 Object construction commands

Object construction commands are processed in the order they appear in the input file. Therefore space in the model allocated to a specific material using for example the `#box` command can be reallocated to another material using the same or any other object construction command. Space in the model can be regarded as a canvas in which objects are introduced and one can be overlaid on top of the other overwriting its properties in order to produce

the desired geometry. The object construction commands can therefore be used to create complex shapes and configurations.

4.4.1 Anisotropy

It is possible to specify objects that have diagonal anisotropy which allows materials such as wood and fibre-reinforced composites, often imaged with GPR, to be more accurately modelled.

$$\bar{\bar{\epsilon}} = \begin{bmatrix} \epsilon_{xx} & 0 & 0 \\ 0 & \epsilon_{yy} & 0 \\ 0 & 0 & \epsilon_{zz} \end{bmatrix}, \quad \bar{\bar{\sigma}} = \begin{bmatrix} \sigma_{xx} & 0 & 0 \\ 0 & \sigma_{yy} & 0 \\ 0 & 0 & \sigma_{zz} \end{bmatrix}$$

Standard isotropic objects specify one material identifier that defines the same properties in x, y, and z directions. However, every volumetric object building command can also be specified with three material identifiers, which allows properties for the x, y, and z directions to be separately defined. The `#plate` command, which defines a surface, can specify up to two material identifiers, and the `#edge` command, which defines a line, continues to take one material identifier. For example to create a box with different material properties in each of the x, y, and z directions use:

```
#material: 41 10 1 0 matX
#material: 35 10 1 0 matY
#material: 33 1 1 0 matZ
#box: 0 0 0 0.1 0.1 0.1 matX matY matZ
```

As another example, to create a cylinder of radius 10 mm that has the same properties in the x and y directions but different properties in the z direction use:

```
#material: 41 10 1 0 matXY
#material: 33 1 1 0 matZ
#cylinder: 0.1 0.1 0.1 0.5 0.1 0.1 0.01 matXY matXY matZ
```

4.4.2 Dielectric smoothing

At the boundaries between different materials in the model there is the question of which material properties to use. Should the last object to be defined at that location dictate the properties? Should an average set of properties of the materials of the objects that share that location be used? This latter option is often referred to as dielectric smoothing and has been shown to result in more accurate simulations *[LUE1994] [BOU1996]*. To address this question gprMax includes an option to turn dielectric smoothing on or off for volumetric object building commands. The default behaviour (if no option is specified) is for dielectric smoothing to be on. The option can be specified with a single character `y` (on) or `n` (off) given after the material identifier in each object command. For example to specify a sphere of material `sand` with dielectric smoothing turned off use: `#sphere: 0.5 0.5 0.5 0.1 sand n`.

Note:

- If a material has dispersive properties then dielectric smoothing is automatically turned off for that material.
- If an object is anistropic then dielectric smoothing is automatically turned off for that object.
- Non-volumetric object building commands, `#edge`, `#plate`, and `#triangle` (applies to triangular patch not triangular prism) cannot have dielectric smoothing.

4.4.3 #geometry_view:

Allows you output to file(s) information about the geometry of model. The file(s) use the open source Visualization ToolKit (VTK) (http://www.vtk.org) format which can be viewed in many free readers, such as Paraview

(http://www.paraview.org). The command can be used to create several 3D views of the model which are useful for checking that it has been constructed as desired. The syntax of the command is:

```
#geometry_view: f1 f2 f3 f4 f5 f6 f7 f8 f9 file1 c1
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the volume of the geometry view in metres.
- `f4 f5 f6` are the upper right (x,y,z) coordinates of the volume of the geometry view in metres.
- `f7 f8 f9` are the spatial discretisation of the geometry view in metres. Typically these will be the same as the spatial discretisation of the model but they can be courser if desired.
- `file1` is the filename of the file where the geometry view will be stored in the same directory as the input file.
- `c1` can be either n (normal) or f (fine) which specifies whether to output the geometry information on a per-cell basis (n) or a per-cell-edge basis (f). The fine mode should be reserved for viewing detailed parts of the geometry that occupy small volumes, as using this mode can generate geometry files with large file sizes.

Tip: When you want to just check the geometry of your model, run gprMax using the optional command line argument `--geometry-only`. This will build the model and produce any geometry view files, but will not run the simulation.

4.4.4 #edge:

Allows you to introduce a wire with specific properties into the model. A wire is an edge of a Yee cell and it can be useful to model resistors or thin wires. The syntax of the command is:

```
#edge: f1 f2 f3 f4 f5 f6 str1
```

- `f1 f2 f3` are the starting (x,y,z) coordinates of the edge, and `f4 f5 f6` are the ending (x,y,z) coordinates of the edge. The coordinates should define a single line.
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.

For example to specify a x-directed wire that is a perfect electric conductor, use: `#edge: 0.5 0.5 0.5 0.7 0.5 0.5 pec`. Note that the y and z coordinates are identical.

4.4.5 #plate:

Allows you to introduce a plate with specific properties into the model. A plate is a surface of a Yee cell and it can be useful to model objects thinner than a Yee cell. The syntax of the command is:

```
#plate: f1 f2 f3 f4 f5 f6 str1
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the plate, and `f4 f5 f6` are the upper right (x,y,z) coordinates of the plate. The coordinates should define a surface and not a 3D object like the `#box` command.
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.

For example to specify a xy oriented plate that is a perfect electric conductor, use: `#plate: 0.5 0.5 0.5 0.7 0.8 0.5 pec`. Note that the z coordinates are identical.

4.4.6 #triangle:

Allows you to introduce a triangular patch or a triangular prism with specific properties into the model. The patch is just a triangular surface made as a collection of staircased Yee cells, and the triangular prism extends the triangular patch in the direction perpendicular to the plane. The syntax of the command is:

```
#triangle: f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 str1 [c1]
```

- `f1 f2 f3` are the coordinates (x,y,z) of the first apex of the triangle, `f4 f5 f6` the coordinates (x,y,z) of the second apex, and `f7 f8 f9` the coordinates (x,y,z) of the third apex.
- `f10` is the thickness of the triangular prism. If the thickness is zero then a triangular patch is created.
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing. For use only when creating a triangular prism, not a triangular patch.

For example, to specify a xy orientated triangular patch that is a perfect electric conductor, use: `#triangle: 0.5 0.5 0.5 0.6 0.4 0.5 0.7 0.9 0.5 0.0 pec`. Note that the z coordinates are identical and the thickness is zero.

4.4.7 #box:

Allows you to introduce an orthogonal parallelepiped with specific properties into the model. The syntax of the command is:

```
#box: f1 f2 f3 f4 f5 f6 str1 [c1]
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the parallelepiped, and `f4 f5 f6` are the upper right (x,y,z) coordinates of the parallelepiped.
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing.

4.4.8 #sphere:

Allows you to introduce a spherical object with specific parameters into the model. The syntax of the command is:

```
#sphere: f1 f2 f3 f4 str1 [c1]
```

- `f1 f2 f3` are the coordinates (x,y,z) of the centre of the sphere.
- `f4` is its radius.
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing.

For example, to specify a sphere with centre at (0.5, 0.5, 0.5), radius 100 mm, and with constitutive parameters of `my_sand`, use: `#sphere:  0.5 0.5 0.5 0.1 my_sand`.

Note:

- Sphere objects are permitted to extend outwith the model domain if desired, however, only parts of object inside the domain will be created.

4.4.9 #cylinder:

Allows you to introduce a circular cylinder into the model. The orientation of the cylinder axis can be arbitrary, i.e. it does not have align with one of the Cartesian axes of the model. The syntax of the command is:

```
#cylinder: f1 f2 f3 f4 f5 f6 f7 str1 [c1]
```

- `f1 f2 f3` are the coordinates (x,y,z) of the centre of one face of the cylinder, and `f4 f5 f6` are the coordinates (x,y,z) of the centre of the other face.
- `f7` is the radius of the cylinder.
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing.

For example, to specify a cylinder with its axis in the y direction, a length of 0.7 m, a radius of 100 mm, and that is a perfect electric conductor, use: `#cylinder: 0.5 0.1 0.5 0.5 0.8 0.5 0.1 pec`.

Note:

- Cylinder objects are permitted to extend outwith the model domain if desired, however, only parts of object inside the domain will be created.

4.4.10 #cylindrical_sector:

Allows you to introduce a cylindrical sector (shaped like a slice of pie) into the model. The syntax of the command is:

```
#cylindrical_sector: c1 f1 f2 f3 f4 f5 f6 f7 str1 [c1]
```

- `c1` is the direction of the axis of the cylinder from which the sector is defined and can be `x`, `y`, or `z`.
- `f1 f2` are the coordinates of the centre of the cylindrical sector.
- `f3 f4` are the lower and higher coordinates of the axis of the cylinder from which the sector is defined (in effect they specify the thickness of the sector).
- `f5` is the radius of the cylindrical sector.
- `f6` is the starting angle (in degrees) for the cylindrical sector (with zero degrees defined on the positive first axis of the plane of the cylindrical sector).
- `f7` is the angle (in degrees) swept by the cylindrical sector (the finishing angle of the sector is always anti-clockwise from the starting angle).
- `str1` is a material identifier that must correspond to material that has already been defined in the input file, or is one of the builtin materials `pec` or `free_space`.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing.

For example, to specify a cylindrical sector with its axis in the z direction, radius of 0.25 m, thickness of 2 mm, a starting angle of 330 °, a sector angle of 60 °, and that is a perfect electric conductor, use: `#cylindrical_sector: z 0.34 0.24 0.500 0.502 0.25 330 60 pec`.

Note:

- Cylindrical sector objects are permitted to extend outwith the model domain if desired, however, only parts of object inside the domain will be created.

4.4.11 #fractal_box:

Allows you to introduce an orthogonal parallelepiped with fractal distributed properties which are related to a mixing model or normal material into the model. The syntax of the command is:

```
#fractal_box: f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 i1 str1 str2 [i2] [c1]
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the parallelepiped, and `f4 f5 f6` are the upper right (x,y,z) coordinates of the parallelepiped.
- `f7` is the fractal dimension which, for an orthogonal parallelepiped, should take values between zero and three.
- `f8` is used to weight the fractal in the x direction.
- `f9` is used to weight the fractal in the y direction.
- `f10` is used to weight the fractal in the z direction.
- `i1` is the number of materials to use for the fractal distribution (defined according to the associated mixing model). This should be set to one if using a normal material instead of a mixing model.
- `str1` is an identifier for the associated mixing model or material.
- `str2` is an identifier for the fractal box itself.
- `i2` is an optional parameter which controls the seeding of the random number generator used to create the fractals. By default (if you don't specify this parameter) the random number generator will be seeded by trying to read data from `/dev/urandom` (or the Windows analogue) if available or from the clock otherwise.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing. If `c1` is specified then a value for `i2` must also be present.

For example, to create an orthogonal parallelepiped with fractal distributed properties using a Peplinski mixing model for soil, with 50 different materials over a range of water volumetric fractions from 0.001 - 0.25, you should first define the mixing model using: `#soil_peplinski: 0.5 0.5 2.0 2.66 0.001 0.25 my_soil` and then specify the fractal box using `#fractal_box: 0 0 0 0.1 0.1 0.1 1.5 1 1 1 50 my_soil my_fractal_box`.

4.4.12 #add_surface_roughness:

Allows you to add rough surfaces to a `#fractal_box` in the model. A fractal distribution is used for the profile of the rough surface. The syntax of the command is:

```
#add_surface_roughness: f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 str1 [i1]
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of a surface on a `#fractal_box`, and `f4 f5 f6` are the upper right (x,y,z) coordinates of a surface on a `#fractal_box`. The coordinates must locate one of the six surfaces of a `#fractal_box` but do not have to extend over the entire surface.
- `f7` is the fractal dimension which, for an orthogonal parallelepiped, should take values between zero and three.
- `f8` is used to weight the fractal in the first direction of the surface.
- `f9` is used to weight the fractal in the second direction of the surface.
- `f10 f11` define lower and upper limits for a range over which the roughness can vary. These limits should be specified relative to the dimensions of the `#fractal_box` that the rough surface is being applied.
- `str1` is an identifier for the `#fractal_box` that the rough surface should be applied to.
- `i1` is an optional parameter which controls the seeding of the random number generator used to create the fractals. By default (if you don't specify this parameter) the random number generator will be seeded

by trying to read data from `/dev/urandom` (or the Windows analogue) if available or from the clock otherwise.

Up to six `#add_rough_surface commands` can be given for any `#fractal_box` corresponding to the six surfaces.

For example, if a `#fractal_box` has been specified using: `#fractal_box: 0 0 0 0.1 0.1 0.1 1.5 1 1 1 50 my_soil my_fractal_box` then to apply a rough surface that varys between 85 mm and 110 mm (i.e. valleys that are up to 15 mm deep and peaks that are up to 10 mm tall) to the surface that is in the positive z direction, use `#add_surface_roughness: 0 0 0.1 0.1 0.1 0.1 1.5 1 1 0.085 0.110 my_fractal_box`.

4.4.13 #add_surface_water:

Allows you to add surface water to a `#fractal_box` in the model that has had a rough surface applied. The syntax of the command is:

```
#add_surface_water: f1 f2 f3 f4 f5 f6 f7 str1
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of a surface on a `#fractal_box`, and `f4 f5 f6` are the upper right (x,y,z) coordinates of a surface on a `#fractal_box`. The coordinates must locate one of the six surfaces of a `#fractal_box` but do not have to extend over the entire surface.
- `f7` defines the depth of the water, which should be specified relative to the dimensions of the `#fractal_box` that the surface water is being applied.
- `str1` is an identifier for the `#fractal_box` that the surface water should be applied to.

For example, to add surface water that is 5 mm deep to an existing `#fractal_box` that has been specified using `#fractal_box: 0 0 0 0.1 0.1 0.1 1.5 1 1 1 50 my_soil my_fractal_box` and has had a rough surface applied using `#add_surface_roughness: 0 0 0.1 0.1 0.1 0.1 1.5 1 1 0.085 0.110 my_fractal_box`, use `#add_surface_water: 0 0 0.1 0.1 0.1 0.1 0.105 my_fractal_box`.

Note:

- The water is modelled using a single-pole Debye formulation with properties $\epsilon_{rs} = 80.1$, $\epsilon_{\infty} = 4.9$, and a relaxation time of $\tau = 9.231 \times 10^{-12}$ seconds (http://dx.doi.org/10.1109/TGRS.2006.873208). If you prefer, gprMax will use your own definition for water as long as it is named `water`.

4.4.14 #add_grass:

Allows you to add grass with roots to a `#fractal_box` in the model. The blades of grass are randomly distributed over the specified surface area and a fractal distribution is used to vary the height of the blades of grass and depth of the grass roots. The syntax of the command is:

```
#add_grass: f1 f2 f3 f4 f5 f6 f7 f8 f9 i1 str1 [i2]
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of a surface on a `#fractal_box`, and `f4 f5 f6` are the upper right (x,y,z) coordinates of a surface on a `#fractal_box`. The coordinates must locate one of three surfaces (in the positive axis direction) of a `#fractal_box` but do not have to extend over the entire surface.
- `f7` is the fractal dimension which, for an orthogonal parallelepiped, should take values between zero and three.
- `f8 f9` define lower and upper limits for a range over which the height of the blades of grass can vary. These limits should be specified relative to the dimensions of the `#fractal_box` that the grass is being applied.

- `i1` is the number of blades of grass that should be applied to the surface area.
- `str1` is an identifier for the `#fractal_box` that the grass should be applied to.
- `i2` is an optional parameter which controls the seeding of the random number generator used to create the fractals. By default (if you don't specify this parameter) the random number generator will be seeded by trying to read data from `/dev/urandom` (or the Windows analogue) if available or from the clock otherwise.

For example, to apply 100 blades of grass that vary in height between 100 and 150 mm to the entire surface in the positive z direction of a `#fractal_box` that had been specified using `#fractal_box: 0 0 0 0.1 0.1 0.1 1.5 1 1 50 my_soil my_fractal_box`, use `#add_grass: 0 0 0.1 0.1 0.1 0.1 1.5 0.2 0.25 100 my_fractal_box`.

Note:

- The grass is modelled using a single-pole Debye formulation with properties $\epsilon_{rs} = 18.5087$, $\epsilon_{\infty} = 12.7174$, and a relaxation time of $\tau = 1.0793 \times 10^{-11}$ seconds (http://dx.doi.org/10.1007/BF00902994). If you prefer, gprMax will use your own definition for grass if you use a material named `grass`. The geometry of the blades of grass are defined by the parametric equations: $x = x_c + s_x \left(\frac{t}{b_x}\right)^2$, $y = y_c + s_y \left(\frac{t}{b_y}\right)^2$, and $z = t$, where s_x and s_y can be -1 or 1 which are randomly chosen, and where the constants b_x and b_y are random numbers based on a Gaussian distribution.

4.4.15 #geometry_objects_read:

Allows you to insert pre-defined geometry into a model. The geometry is specified using a 3D array of integer numbers stored in a HDF5 file. The integer numbers must correspond to the order of a list of `#material` commands specified in a text file. The syntax of the command is:

```
#geometry_objects_read: f1 f2 f3 file1 file2
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates in the domain where the lower left corner of the geometry array should be placed.
- `file1` is the path to and filename of the HDF5 file that contains an integer array which defines the geometry.
- `file2` is the path to and filename of the text file that contains `#material` commands.
- `c1` is an optional parameter which can be `y` or `n`, used to switch on and off dielectric smoothing. Dielectric smoothing can only be turned on if the geometry objects that are being read were originally generated by gprMax, i.e. via the `#geometry_objects_write` command.

Note:

- The integer numbers in the HDF5 file must be stored as a NumPy array at the root named `data` with type `np.int16`.
- The integer numbers in the HDF5 file correspond to the order of material commands in the materials text file, i.e. if `#sand: 3 0 1 0` is the first material in the materials file, it will be associated with any integers that are zero in the HDF5 file.
- You can use an integer of -1 in the HDF5 file to indicate not to build any material at that location, i.e. whatever material is already in the model at that location.
- The spatial resolution of the geometry objects must match the spatial resolution defined in the model.
- The spatial resolution must be specified as a root attribute of the HDF5 file with the name `dx_dy_dz` equal to a tuple of floats, e.g. (0.002, 0.002, 0.002)
- If the geometry objects being imported were originally generated using gprMax, i.e. exported using #geometry_objects_write, then you can use dielectric smoothing as you like when generating the original geometry

objects. However, if the geometry objects being imported were generated by an external method then dielectric smoothing will not take place.

For example, to insert a 2x2x2mm^3 AustinMan model with the lower left corner 40mm from the origin of the domain, and using disperive material properties use `#geometry_objects_read: 0.04 0.04 0.04 ../user_libs/AustinManWoman/AustinMan_v2.3_2x2x2.h5 ../user_libs/AustinManWoman/AustinManWoman_materials_dispersive.txt`

4.4.16 #geometry_objects_write:

Allows you to write geometry generated in a model to file. The file can be read back into gprMax using the `#geometry_objects_read` command. This allows complex geometry that can take some time to generate to be saved to file and more quickly imported into subsequent models. The geometry information is saved as a 3D array of integer numbers stored in a HDF5 file, and corresponding material information is stored in a text file. The integer numbers correspond to the order of a list of `#material` commands specified in the text file. The syntax of the command is:

```
#geometry_objects_write: f1 f2 f3 f4 f5 f6 file1
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the parallelepiped, and `f4 f5 f6` are the upper right (x,y,z) coordinates of the parallelepiped.
- `file1` is the basename for the files where geometry and material information will be stored.

Note:

- The structure of the HDF5 file is the same as that described for the `#geometry_objects_read` command.
- Objects are stored using spatial resolution defined in the model.

4.5 Source and output commands

4.5.1 #waveform:

Allows you to specify waveforms to use with sources in the model. The syntax of the command is:

```
#waveform: str1 f1 f2 str2
```

- `str1` is the type of waveform which can be:
  - `gaussian` which is a Gaussian waveform.
  - `gaussiandot` which is the first derivative of a Gaussian waveform.
  - `gaussiandotnorm` which is the normalised first derivative of a Gaussian waveform.
  - `gaussiandotdot` which is the second derivative of a Gaussian waveform.
  - `gaussiandotdotnorm` which is the normalised second derivative of a Gaussian waveform.
  - `ricker` which is a Ricker (or Mexican hat) waveform, i.e. the negative, normalised second derivative of a Gaussian waveform.
  - `gaussianprime` which is the first derivative of a Gaussian waveform, directly derived from the aforementioned `gaussian` (see notes below).
  - `gaussiandoubleprime` which is the second derivative of a Gaussian waveform, directly derived from the aforementioned `gaussian` (see notes below).

  - `sine` which is a single cycle of a sine waveform.
  - `contsine` which is a continuous sine waveform. In order to avoid introducing noise into the calculation the amplitude of the waveform is modulated for the first cycle of the sine wave (ramp excitation).
- `f1` is the scaling of the maximum amplitude of the waveform (for a `#hertzian_dipole` the units will be Amps, for a `#voltage_source` or `#transmission_line` the units will be Volts).
- `f2` is the centre frequency of the waveform (Hertz). In the case of the Gaussian waveform it is related to the pulse width.
- `str2` is an identifier for the waveform used to assign it to a source.

For example, to specify the normalised first derivate of a Gaussian waveform with an amplitude of one and a centre frequency of 1.2GHz, use: `#waveform: gaussiandotnorm 1 1.2e9 my_gauss_pulse`.

Note:

- The functions used to create the waveforms can be found in the *tools section*.
- `gaussiandot`, `gaussiandotnorm`, `gaussiandotdot`, `gaussiandotdotnorm`, `ricker` waveforms have their centre frequencies specified by the user, i.e. they are not derived to the 'base' `gaussian`
- `gaussianprime` and `gaussiandoubleprime` waveforms are the first derivative and second derivative of the 'base' `gaussian` waveform, i.e. the centre frequencies of the waveforms will rise for the first and second derivatives.

4.5.2 #excitation_file:

Allows you to specify an ASCII file that contains columns of amplitude values that specify custom waveform shapes that can be used with sources in the model.

The first row of each column must begin with a identifier string that will be used as the name of each waveform. Optionally, the first column of the file may contain a time vector of values (which must use the identifier `time`) to interpolate the amplitude values of the waveform. If a time vector is not given, a vector of time values corresponding to the simulation time step and number of iterations will be used.

If there are less amplitude values than the number of iterations that are going to be performed, the end of the sequence of amplitude values will be padded with zero values up to the number of iterations. If extra amplitude values are specified than needed then they are ignored. The syntax of the command is:

```
#excitation_file: file1 [str1 str2]
```

- `file1` can be the name of the file containing the specified waveform in the same directory as the input file, or `file` can be the full path to the file containing the specified waveform (allowing you to specify any location).
- `str1` and `str2` are an optional parameter pair that allow values for `kind` and `fill_value` to be passed to the interpolation function (scipy.interpolate.interp1d (https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html)). If they are not given the default values for the function will be used.

For example, to specify the file `my_waves.txt`, which contains two custom waveform shapes, use: `#excitation_file: my_waves.txt`. The contents of the file `my_waves.txt` would take the form:

```
time my_pulse1 my_pulse2
0 0 0
1.926e-12 1.2e-6 0
3.852e-12 1.3e-6 1.0e-1
5.778e-12 5.0e-6 1.5e-1
...       ...    ...
```

(continues on next page)

(continued from previous page)

```
...       ...    ...
...       ...    ...
```

Then to use `my_pulse1` custom waveform shape with, for example, a z-polarised Hertzian dipole source:

```
#hertzian_dipole: z 0.5 0.5 0.5 my_pulse1
```

Note:

- The `#waveform` command is not necessary when using a custom waveform excitation, only the `#excitation_file` command and whatever source is going to be used with the custom waveform excitation.

4.5.3 #hertzian_dipole:

Allows you to specify a current density term at an electric field location - the simplest excitation, often referred to as an additive or soft source.

$$J_s = \frac{I\Delta l}{\Delta x \Delta y \Delta z},$$

where J_s is the current density, I is the current, Δl is the length of the infinitesimal electric dipole, and Δx, Δy, and Δz are the spatial resolution of the grid.

Note:

- Δl is set equal to Δx, Δy, or Δz depending on the specified polarisation.

The syntax of the command is:

```
#hertzian_dipole: c1 f1 f2 f3 str1 [f4 f5]
```

- `c1` is the polarisation of the source and can be `x`, `y`, or `z`.
- `f1 f2 f3` are the coordinates (x,y,z) of the source in the model.
- `f4 f5` are optional parameters. `f4` is a time delay in starting the source. `f5` is a time to remove the source. If the time window is longer than the source removal time then the source will stop after the source removal time. If the source removal time is longer than the time window then the source will be active for the entire time window. If `f4 f5` are omitted the source will start at the beginning of time window and stop at the end of the time window.
- `str1` is the identifier of the waveform that should be used with the source.

For example, to use a x-polarised Hertzian dipole with unit amplitude and a 600 MHz centre frequency Ricker waveform, use: `#waveform: ricker 1 600e6 my_ricker_pulse` and `#hertzian_dipole: x 0.05 0.05 0.05 my_ricker_pulse`.

Note:

- When a `#hertzian_dipole` is used in a 2D simulation it acts as a line source of current in the invariant (geometry) direction of the simulation.

4.5.4 #magnetic_dipole:

This will simulate an infinitesimal magnetic dipole. This is often referred to as an additive or soft source. The syntax of the command is:

```
#magnetic_dipole: c1 f1 f2 f3 str1 [f4 f5]
```

- `c1` is the polarisation of the source and can be `x`, `y`, or `z`.
- `f1 f2 f3` are the coordinates (x,y,z) of the source in the model.
- `f4 f5` are optional parameters. `f4` is a time delay in starting the source. `f5` is a time to remove the source. If the time window is longer than the source removal time then the source will stop after the source removal time. If the source removal time is longer than the time window then the source will be active for the entire time window. If `f4 f5` are omitted the source will start at the beginning of time window and stop at the end of the time window.
- `str1` is the identifier of the waveform that should be used with the source.

4.5.5 #voltage_source:

Allows you to introduce a voltage source at an electric field location. It can be a hard source if it's resistance is zero, i.e. the time variation of the specified electric field component is prescribed, or if it's resistance is non-zero it behaves as a resistive voltage source. It is useful for exciting antennas when the physical properties of the antenna are included in the model. The syntax of the command is:

```
#voltage_source: c1 f1 f2 f3 f4 str1 [f5 f6]
```

- `c1` is the polarisation of the source and can be `x`, `y`, or `z`.
- `f1 f2 f3` are the coordinates (x,y,z) of the source in the model.
- `f4` is the internal resistance of the voltage source in Ohms. If `f4` is set to zero then the voltage source is a hard source. That means it prescribes the value of the electric field component. If the waveform becomes zero then the source is perfectly reflecting.
- `f5 f6` are optional parameters. `f5` is a time delay in starting the source. `f6` is a time to remove the source. If the time window is longer than the source removal time then the source will stop after the source removal time. If the source removal time is longer than the time window then the source will be active for the entire time window. If `f5 f6` are omitted the source will start at the beginning of time window and stop at the end of the time window.
- `str1` is the identifier of the waveform that should be used with the source.

For example, to specify a y directed voltage source with an internal resistance of 50 Ohms, an amplitude of five, and a 1.2 GHz centre frequency Gaussian waveform use: `#waveform: gaussian 5 1.2e9 my_gauss_pulse` and `#voltage_source: y 0.05 0.05 0.05 50 my_gauss_pulse`.

4.5.6 #transmission_line:

Allows you to introduce a one-dimensional transmission line model *[MAL1994]* at an electric field location. The transmission line can have a specified resistance greater than zero and less than the impedance of free space (376.73 Ohms). It is useful for exciting antennas when the physical properties of the antenna are included in the model. The syntax of the command is:

```
#transmission_line: c1 f1 f2 f3 f4 str1 [f5 f6]
```

- `c1` is the polarisation of the transmission line and can be `x`, `y`, or `z`.
- `f1 f2 f3` are the coordinates (x,y,z) of the transmission line in the model.

- `f4` is the characteristic resistance of the transmission line source in Ohms. It can be any value greater than zero and less than the impedance of free space (376.73 Ohms).
- `f5 f6` are optional parameters. `f5` is a time delay in starting the excitation of the transmission line. `f6` is a time to remove the excitation of the transmission line. If the time window is longer than the excitation of the transmission line removal time then the excitation of the transmission line will stop after the excitation of the transmission line removal time. If the excitation of the transmission line removal time is longer than the time window then the excitation of the transmission line will be active for the entire time window. If `f5 f6` are omitted the excitation of the transmission line will start at the beginning of time window and stop at the end of the time window.
- `str1` is the identifier of the waveform that should be used with the source.

Time histories of voltage and current values in the transmission line are saved to the output file. These are documented in the *output file section*. These parameters are useful for calculating characteristics of an antenna such as the input impedance or S-parameters. gprMax includes a Python module (in the `tools` package) to help you view the input impedance and s11 parameter from an antenna model fed using a transmission line. Details of how to use this module is given in the *tools section*.

For example, to specify a z directed transmission line source with a resistance of 75 Ohms, an amplitude of five, and a 1.2 GHz centre frequency Gaussian waveform use: `#waveform: gaussian 5 1.2e9 my_gauss_pulse` and `#transmission_line: z 0.05 0.05 0.05 75 my_gauss_pulse`.

An example antenna model using a transmission line can be found in the *examples section*.

4.5.7 #rx:

Allows you to introduce output points into the model. These are locations where the values of the electric and magnetic field components over the number of iterations of the model will be saved to file. The syntax of the command is:

```
#rx: f1 f2 f3 [str1 str2]
```

- `f1 f2 f3` are the coordinates (x,y,z) of the receiver in the model.
- `str1` is the identifier of the receiver.
- `str2` is a list of outputs with this receiver. It can be any selection from `Ex`, `Ey`, `Ez`, `Hx`, `Hy`, `Hz`, `Ix`, `Iy`, or `Iz`.

Note:

- When the optional parameters `str1` and `str2` are not given all the electric and magnetic field components will be output with the receiver point.

4.5.8 #rx_array:

Provides a simple method of defining multiple output points in the model. The syntax of the command is:

```
#rx_array: f1 f2 f3 f4 f5 f6 f7 f8 f9
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the output line/rectangle/volume, and `f4 f5 f6` are the upper right (x,y,z) coordinates of the output line/rectangle/volume.
- `f7 f8 f9` are the increments (x,y,z) which define the number of output points in each direction. `f7`, `f8`, or `f9` can be set to zero to prevent any output points in a particular direction. Otherwise, the minimum value of `f7` is Δx, the minimum value of `f8` is Δy, and the minimum value of `f9` is Δz.

4.5.9 #src_steps: and #rx_steps:

Provides a simple method to allow you to move the location of all simple sources (`#src_steps`) or all receivers (`#rx_steps`) between runs of a model. The syntax of the commands is:

```
#src_steps: f1 f2 f3
#rx_steps: f1 f2 f3
```

`f1 f2 f3` are increments (x,y,z) to move all simple sources (`#hertzian_dipole` or `#magnetic_dipole`) or all receivers (created using either `#rx` or `#rx_array` commands).

Note:

- `#src_steps` and `#rx_steps` are not suitable for moving sources which have associated geometry, e.g. antenna models.
- Values for `#src_steps` and `#rx_steps` should not be changed between model runs using Python scripting.

4.5.10 #snapshot:

Allows you to obtain information about the electromagnetic fields within a volume of the model at a given time instant. The file(s) use the open source Visualization ToolKit (VTK) (http://www.vtk.org) format which can be viewed in many free readers, such as Paraview (http://www.paraview.org). The syntax of this command is:

```
#snapshot: f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 file1
```

or

```
#snapshot: f1 f2 f3 f4 f5 f6 f7 f8 f9 i1 file1
```

- `f1 f2 f3` are the lower left (x,y,z) coordinates of the volume of the snapshot in metres.
- `f4 f5 f6` are the upper right (x,y,z) coordinates of the volume of the snapshot in metres.
- `f7 f8 f9` are the spatial discretisation of the snapshot in metres.
- `f10` or `i1` are the time in seconds (float) or the iteration number (integer) which denote the point in time at which the snapshot will be taken.
- `file1` is the name of the file where the snapshot will be stored. Snapshot files are automatically stored in a directory with the name of the input file appended with '_snaps'. For multiple model runs each model run will have its own directory, i.e. '_snaps1', 'snaps2' etc...

For example to save a snapshot of the electromagnetic fields in the model at a simulated time of 3 nanoseconds use: `#snapshot: 0 0 0 1 1 1 0.1 0.1 0.1 3e-9 snap1`

Tip: You can take advantage of Python scripting to easily create a series of snapshots. For example, to create 30 snapshots starting at time 0.1ns until 3ns in intervals of 0.1ns, use the following code snippet in your input file. Replace `x1 y1 z1 x2 y2 z2 dx dy dz` accordingly.

```
#python:
for i in range(1, 31):
    print('#snapshot: x1 y1 z1 x2 y2 z2 dx dy dz {} snapshot{}'.format((i/10)*1e-9,
↪ i))
#end_python:
```

4.6 PML commands

The default behaviour is for gprMax to use a first order CFS PML that has a thickness of 10 cells on each of the six sides of the model domain. This can be altered by using the following commands.

4.6.1 #pml_cells:

Allows you to control the number of cells of PML that are used on the six sides of the model domain. The PML is defined within the model domain, i.e. it is not added to the domain size. The syntax of the command is:

```
#pml_cells: i1 [i2 i3 i4 i5 i6]
```

- `i1` is the number of cells of PML to use on all sides of the model domain (can be set to zero to completely switch off the PML), or `i1` is the number of cells of PML to use on the side of the model domain nearest the origin of the x-axis (x0).
- `i2` is the number of cells of PML to use on the side of the model domain nearest the origin of the y-axis (y0).
- `i3` is the number of cells of PML to use on the side of the model domain nearest the origin of the z-axis (z0).
- `i4` is the number of cells of PML to use on the side of the model domain furthest from the origin of the x-axis (xmax).
- `i5` is the number of cells of PML to use on the side of the model domain furthest from the origin of the y-axis (ymax).
- `i6` is the number of cells of PML to use on the side of the model domain furthest from the origin of the z-axis (zmax).
- `i1 i2 i3 i4 i5 i6` may be set to zero to turn off the PML on a specific side of the model domain.

For example to use a PML with 20 cells (thicker than the default 10 cells) on only the z-axis sides of the domain use:

```
#pml_cells: 10 10 20 10 10 20
```

4.6.2 #pml_cfs:

Allows you (advanced) control of the parameters that are used to build each order of the PML. Up to a second order PML can currently be specified, i.e. by using two `#pml_cfs` commands. The syntax of the command is:

```
#pml_cfs: str1 str2 f1 f2 str3 str4 f3 f4 str5 str6 f5 f6
```

- `str1` is the type of scaling to use for the CFS α parameter. It can be `constant`, `linear`, `quadratic`, `cubic`, `quartic`, `quintic` and `sextic`.
- `str2` is the direction of the scaling to use for the CFS α parameter. It can be `forward` or `reverse`.
- `f1 f2` are the minimum and maximum values for the CFS α parameter.
- `str3` is the type of scaling to use for the CFS κ parameter. It can be `constant`, `linear`, `quadratic`, `cubic`, `quartic`, `quintic` and `sextic`.
- `str4` is the direction of the scaling to use for the CFS κ parameter. It can be `forward` or `reverse`.
- `f3 f4` are the minimum and maximum values for the CFS κ parameter. The minimum value for the CFS κ parameter is one.
- `str5` is the type of scaling to use for the CFS σ parameter. It can be `constant`, `linear`, `quadratic`, `cubic`, `quartic`, `quintic` and `sextic`.

- `str6` is the direction of the scaling to use for the CFS σ parameter. It can be `forward` or `reverse`.
- `f5 f6` are the minimum and maximum values for the CFS σ parameter.

The CFS values (which are internally specified) used for the default standard first order PML are: `#pml_cfs: constant forward 0 0 constant forward 1 1 quartic forward 0 None`. Specifying 'None' for the maximum value of σ forces gprMax to calculate it internally based on the relative permittivity and permeability of the underlying materials in the model.

The parameters will be applied to all slabs of the PML that are switched on.

Tip: `forward` direction implies minimum parameter value at the inner boundary of the PML and maximum parameter value at the edge of computational domain, `reverse` is the opposite.

CHAPTER 5

Output data

5.1 Field(s) output

gprMax produces an output file that has the same name as the input file but with `.out` appended. The output file uses the widely-supported HDF5 (https://www.hdfgroup.org/HDF5/) format which was designed to store and organize large amounts of numerical data. There are a number of free tools available to read HDF5 files. Also MATLAB has high- and low-level functions for reading and writing HDF5 files, i.e. `h5info` and `h5disp` are useful for returning information and displaying the contents of HDF5 files respectively. gprMax includes some Python modules (in the `tools` package) to help you view output data. These are documented in the *tools section*.

5.1.1 File structure

The output file has the following HDF5 attributes at the root (`/`):

- `gprMax` is the version number of gprMax used to create the output
- `Title` is the title of the model
- `Iterations` is the number of iterations for the time window of the model
- `nx_ny_nz` is a tuple containing the number of cells in each direction of the model
- `dx_dy_dz` is a tuple containing the spatial discretisation, i.e. Δx, Δy, Δz
- `dt` is the time step of the model, i.e. Δt
- `srcsteps` is the spatial increment used to move all sources between model runs.
- `rxsteps` is the spatial increment used to move all receivers between model runs.
- `nsrc` is the total number of sources in the model.
- `nrx` is the total number of receievers in the model.

The output file contains HDF5 groups for sources (`srcs`), transmission lines (`tls`), and receivers (`rxs`). Within each group are further groups that correspond to individual sources/transmission lines/receivers, e.g. `src1`, `src2` etc...

```
/
    rxs/
        rx1/
```

(continues on next page)

(continued from previous page)

```
            Name
            Position
            Ex
            Ey
            Ez
            Hx
            Hy
            Hz
            Ix [optional]
            Iy [optional]
            Iz [optional]
        rx2/
            ...
    srcs/
        src1/
            Type
            Position
        src2/
            ...

    tls/
        tl1/
            Position
            Resistance
            dl
            Vinc
            Iinc
            Vtotal
            Itotal
        tl2/
            ...
```

Within each individual `rx` group are the following attributes:

- `Name` is the name of the receiver if specified. Otherwise 'Rx(x,y,z)', where x,y,z is the position of the receiver, is used.
- `Position` is the x, y, z position (in metres) of the receiver in the model.

Within each individual `rx` group can be the following datasets:

- `Ex` is an array containing the time history (for the model time window) of the values of the x component of the electric field at that receiver position.
- `Ey` is an array containing the time history (for the model time window) of the values of the y component of the electric field at that receiver position.
- `Ez` is an array containing the time history (for the model time window) of the values of the z component of the electric field at that receiver position.
- `Hx` is an array containing the time history (for the model time window) of the values of the x component of the magnetic field at that receiver position.
- `Hy` is an array containing the time history (for the model time window) of the values of the y component of the magnetic field at that receiver position.
- `Hz` is an array containing the time history (for the model time window) of the values of the z component of the magnetic field at that receiver position.
- `Ix` is an optional array containing the time history (for the model time window) of the values of the x component of current (calculated around a single cell loop) at that receiver position.
- `Iy` is an optional array containing the time history (for the model time window) of the values of the y component of current (calculated around a single cell loop) at that receiver position.

- `Iz` is an optional array containing the time history (for the model time window) of the values of the z component of current (calculated around a single cell loop) at that receiver position.

Within each individual `src` group are the following attributes:

- `Type` is the type of source, e.g. Hertzian dipole, voltage source etc...
- `Position` is the x, y, z position (in metres) of the source in the model.

Within each individual `tl` group are the following attributes:

- `Position` is the x, y, z position (in metres) of the source in the model.
- `Resistance` is the resistance of the transmission line.
- `dl` is the spatial discretisation of the transmission line.

Within each individual `tl` group are the following datasets:

- `Vinc` is an array containing the time history (for the model time window) of the values of the incident voltage in the transmission line.
- `Iinc` is an array containing the time history (for the model time window) of the values of the incident current in the transmission line.
- `Vtotal` is an array containing the time history (for the model time window) of the values of the total (field) voltage in the transmission line.
- `Itotal` is an array containing the time history (for the model time window) of the values of the total (field) current in the transmission line.

5.1.2 Snapshots

Snapshot files use the open source Visualization ToolKit (VTK) (http://www.vtk.org) format which can be viewed in many free readers, such as Paraview (http://www.paraview.org). Paraview is an open-source, multi-platform data analysis and visualization application. It is available for Linux, macOS, and Windows. The `#snapshot:` command produces an ImageData (.vti) snapshot file containing electric and magnetic field data and current data for each time instance requested.

Tip: You can take advantage of Python scripting to easily create a series of snapshots. For example, to create 30 snapshots starting at time 0.1ns until 3ns in intervals of 0.1ns, use the following code snippet in your input file. Replace `xs, ys, zs, xf, yf, zf, dx, dy, dz` accordingly.

```
#python:
from gprMax.input_cmd_funcs import *
for i in range(1, 31):
    snapshot(xs, ys, zs, xf, yf, zf, dx, dy, dz, (i/10)*1e-9, 'snapshot' + str(i))
#end_python:
```

The following are steps to get started with viewing snapshot files in Paraview:

1. **Open the file** either from the File menu or toolbar. Paraview should recognise the time series based on the file name and load in all the files.
2. Click the **Apply** button in the Properties panel. You should see an outline of the snapshot volume.
3. Use the **Coloring** drop down menu to select either **E-field** or **H-field**, and the further drop down menu to select either **Magnitude**, **x**, **y** or **z** component.
4. From the **Representation** drop down menu select **Surface**.
5. You can step through or play as an animation the time steps using the **time controls** in the toolbar.

Tip:

- Turn on the Animation View (View->Animation View menu) to control the speed and start/stop points of the animation.
- Use the Color Map Editor to adjust the Color Scaling.
- Adjust the default lighting: In the Properties panel click on the gear icon to turn on the advanced properties. Go to the Lights section and click edit. Uncheck the Light Kit check box and click Close.

5.2 Geometry output

Geometry files use the open source Visualization ToolKit (VTK) (http://www.vtk.org) format which can be viewed in many free readers, such as Paraview (http://www.paraview.org). Paraview is an open-source, multi-platform data analysis and visualization application. It is available for Linux, Mac OS X, and Windows.

The `#geometry_view:` command produces either ImageData (.vti) for a per-cell geometry view, or PolygonalData (.vtp) for a per-cell-edge geometry view. The per-cell geometry views also show the location of the PML regions and any sources and receivers in the model. The following are steps to get started with viewing geometry files in Paraview:

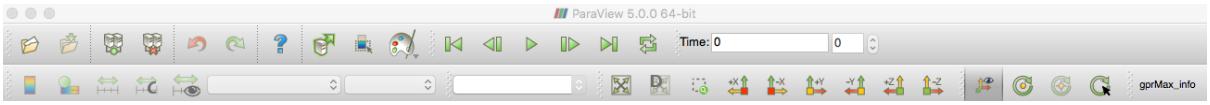


Fig. 5.1: Paraview toolbar showing `gprMax_info` macro button.

1. **Open the file** either from the File menu or toolbar.
2. Click the **Apply** button in the Properties panel. You should see an outline of the volume of the geometry view.
3. Install the `gprMax_info.py` Python script, that comes with the gprMax source code (in the `tools/Paraview macros` directory), as a macro in Paraview. This script makes it quick and easy to view the different materials in a geometry file. To add the script as a macro in Paraview choose the file from the Macros->Add new macro menu. It will then appear as a shortcut button in the toolbar as shown in Fig. 5.1. You only need to do this once, the macro will be kept in Paraview for future use.
4. Click the `gprMax_info` shortcut button. All the materials in the model should appear in the Pipeline Browser as Threshold items as shown in Fig. 5.2.

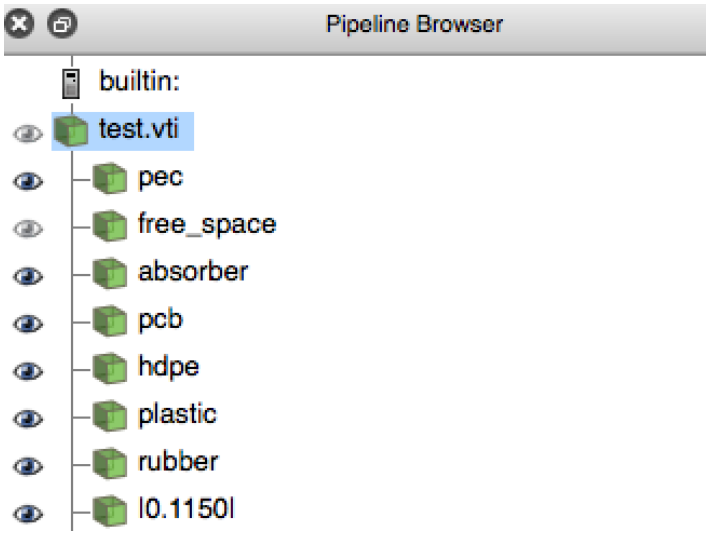


Fig. 5.2: Paraview Pipeline Browser showing list of materials in an example model.

Tip:

- You can turn on and off the visibility of materials using the eye icon in the Pipeline Browser. You can select multiple materials using the Shift key, and by shift-clicking the eye icon, turn the visibility of multiple materials on and off.
- You can set the Color and Opacity of materials from the Properties panel.

CHAPTER 6

Plotting

6.1 A-scans

6.1.1 plot_Ascan.py

This module uses matplotlib to plot the time history for the electric and magnetic field components, and currents for all receivers in a model (each receiver gets a separate figure window). Usage (from the top-level gprMax directory) is:

```
python -m tools.plot_Ascan outputfile
```

where `outputfile` is the name of output file including the path.

There are optional command line arguments:

- `--outputs` to specify a subset of the default output components (`Ex`, `Ey`, `Ez`, `Hx`, `Hy`, `Hz`, `Ix`, `Iy` or `Iz`) to plot. By default all electric and magnetic field components are plotted.
- `-fft` to plot the Fast Fourier Transform (FFT) of a single output component

For example to plot the `Ez` output component with it's FFT:

```
python -m tools.plot_Ascan my_outputfile.out --outputs Ez -fft
```

6.2 B-scans

6.2.1 plot_Bscan.py

gprMax produces a separate output file for each trace (A-scan) in the B-scan. These must be combined into a single file using the `outputfiles_merge.py` module (described in the *other utilities section*). This module uses matplotlib to plot an image of the B-scan. Usage (from the top-level gprMax directory) is:

```
python -m tools.plot_Bscan outputfile rx-component
```

where:

- `outputfile` is the name of output file including the path

- `rx-component` is the name of the receiver output component (`Ex`, `Ey`, `Ez`, `Hx`, `Hy`, `Hz`, `Ix`, `Iy` or `Iz`) to plot

6.3 Antenna parameters

6.3.1 plot_antenna_params.py

This module uses matplotlib to plot the input impedance (resistance and reactance) and s11 parameter from an antenna model fed using a transmission line. It also plots the time history of the incident and reflected voltages in the transmission line and their frequency spectra. The module can optionally plot the s21 parameter if another transmission line or a receiver output (`#rx`) is used on the receiver antenna. Usage (from the top-level gprMax directory) is:

```
python -m tools.plot_antenna_params outputfile
```

where `outputfile` is the name of output file including the path.

There are optional command line arguments:

- `--tltx-num` is the number of the transmission line (default is one) for the transmitter antenna. Transmission lines are numbered (starting at one) in the order they appear in the input file.
- `--tlrx-num` is the number of the transmission line (default is None) for the receiver antenna (for a s21 parameter). Transmission lines are numbered (starting at one) in the order they appear in the input file.
- `--rx-num` is the number of the receiver output (default is None) for the receiver antenna (for a s21 parameter). Receivers are numbered (starting at one) in the order they appear in the input file.
- `--rx-component` is the electric field component (`Ex`, `Ey` or `Ez`) of the receiver output for the receiver antenna (for a s21 parameter).

For example to plot the input impedance, s11 and s21 parameters from a simulation with transmitter and receiver antennas that are attached to transmission lines (the transmission line feeding the transmitter appears first in the input file, and the transmission line attached to the receiver antenna appears after it).

```
python -m tools.plot_antenna_params outputfile --tltx-num 1 --tlrx-num 2
```

6.4 Built-in waveforms

This section describes the definitions of the functions that are used to create the built-in waveforms, and how to plot them.

6.4.1 plot_source_wave.py

This module uses matplotlib to plot one of the built-in waveforms and it's FFT. Usage (from the top-level gprMax directory) is:

```
python -m tools.plot_source_wave type amp freq timewindow dt
```

where:

- `type` is the type of waveform, e.g. gaussian, ricker etc...
- `amp` is the amplitude of the waveform
- `freq` is the centre frequency of the waveform (Hertz). In the case of the Gaussian waveform it is related to the pulse width.

- `timewindow` is the time window (seconds) to view the waveform, i.e. the time window of the proposed simulation
- `dt` is the time step (seconds) to view waveform, i.e. the time step of the proposed simulation

There is an optional command line argument:

- `-fft` to plot the Fast Fourier Transform (FFT) of the waveform

6.4.2 Definitions

Definitions of the built-in waveforms and example plots are shown using the parameters: amplitude of one, centre frequency of 1GHz, time window of 6ns, and a time step of 1.926ps.

gaussian

A Gaussian waveform.

$$W(t) = e^{-\zeta(t-\chi)^2}$$

where $\zeta = 2\pi^2 f^2$, $\chi = \frac{1}{f}$ and f is the frequency.

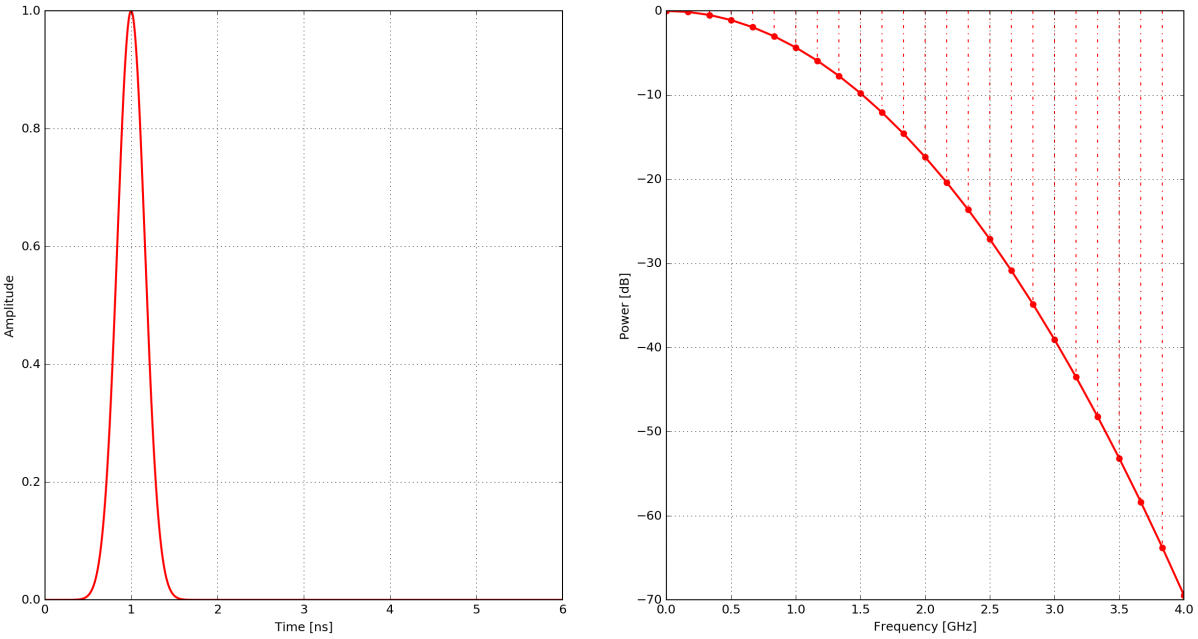


Fig. 6.1: Example of the `gaussian` waveform - time domain and power spectrum.

gaussiandot

First derivative of a Gaussian waveform.

$$W(t) = -2\zeta(t-\chi)e^{-\zeta(t-\chi)^2}$$

where $\zeta = 2\pi^2 f^2$, $\chi = \frac{1}{f}$ and f is the frequency.

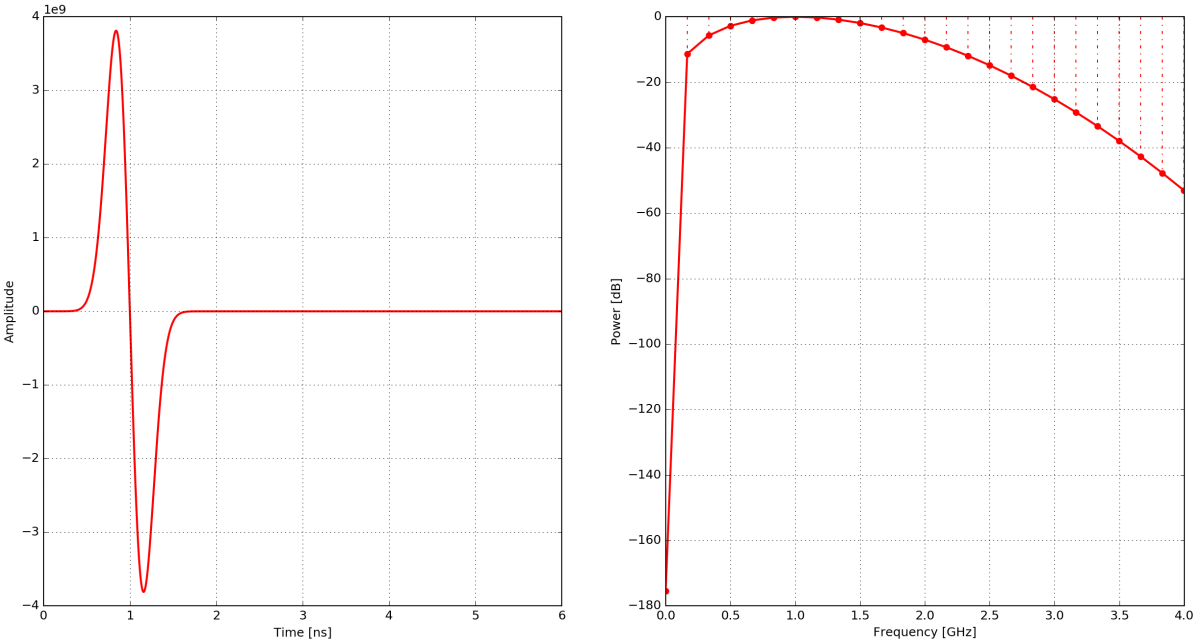


Fig. 6.2: Example of the `gaussiandot` waveform - time domain and power spectrum.

gaussiandotnorm

Normalised first derivative of a Gaussian waveform.

$$W(t) = -2\sqrt{\frac{e}{2\zeta}}\zeta(t-\chi)e^{-\zeta(t-\chi)^2}$$

where $\zeta = 2\pi^2 f^2$, $\chi = \frac{1}{f}$ and f is the frequency.

gaussiandotdot

Second derivative of a Gaussian waveform.

$$W(t) = 2\zeta\left(2\zeta(t-\chi)^2 - 1\right)e^{-\zeta(t-\chi)^2}$$

where $\zeta = \pi^2 f^2$, $\chi = \frac{\sqrt{2}}{f}$ and f is the frequency.

gaussiandotdotnorm

Normalised second derivative of a Gaussian waveform.

$$W(t) = \left(2\zeta(t-\chi)^2 - 1\right)e^{-\zeta(t-\chi)^2}$$

where $\zeta = \pi^2 f^2$, $\chi = \frac{\sqrt{2}}{f}$ and f is the frequency.

ricker

A Ricker (or Mexican Hat) waveform which is the negative, normalised second derivative of a Gaussian waveform.

$$W(t) = -\left(2\zeta(t-\chi)^2 - 1\right)e^{-\zeta(t-\chi)^2}$$

where $\zeta = \pi^2 f^2$, $\chi = \frac{\sqrt{2}}{f}$ and f is the frequency.

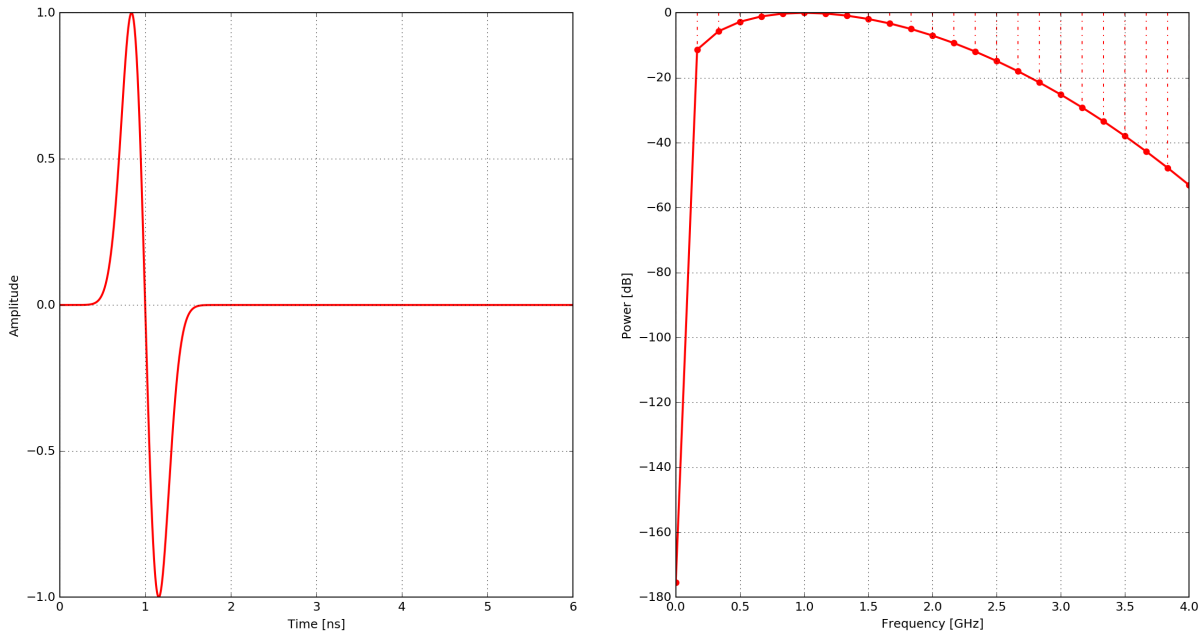


Fig. 6.3: Example of the `gaussiandotnorm` waveform - time domain and power spectrum.

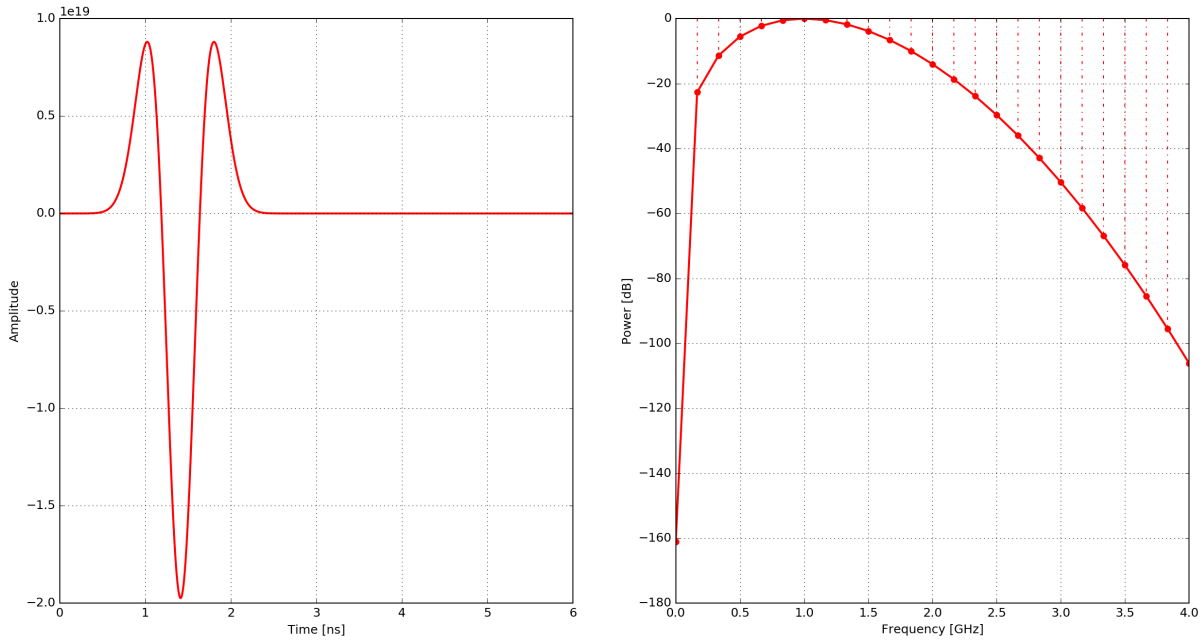


Fig. 6.4: Example of the `gaussiandotdot` waveform - time domain and power spectrum.

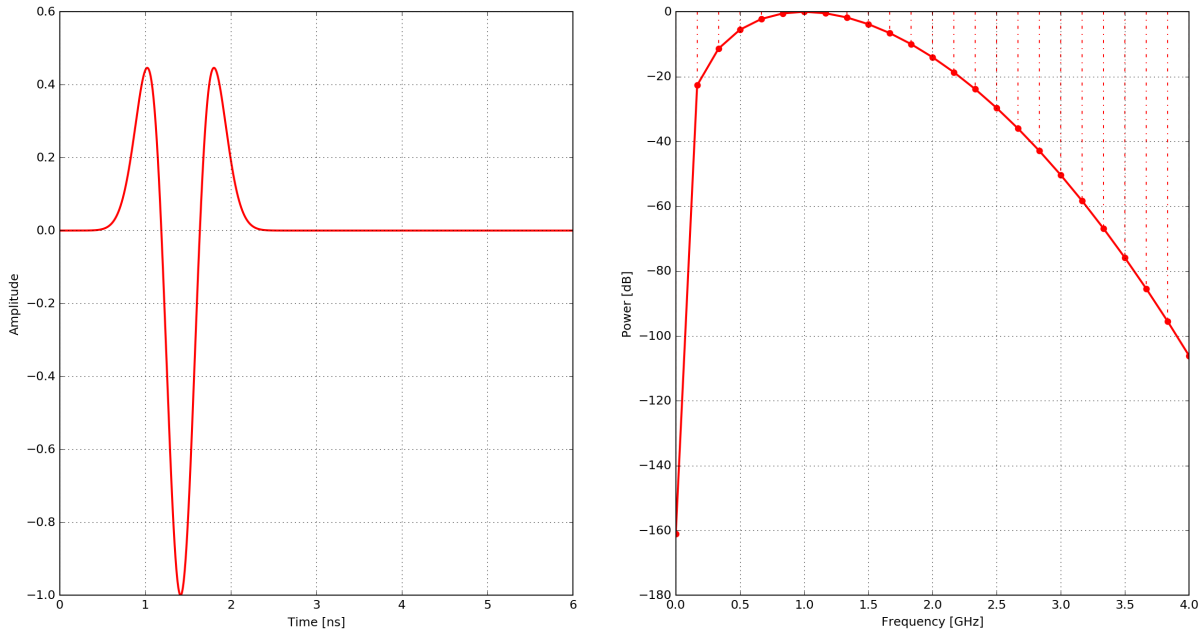


Fig. 6.5: Example of the `gaussiandotdotnorm` waveform - time domain and power spectrum.

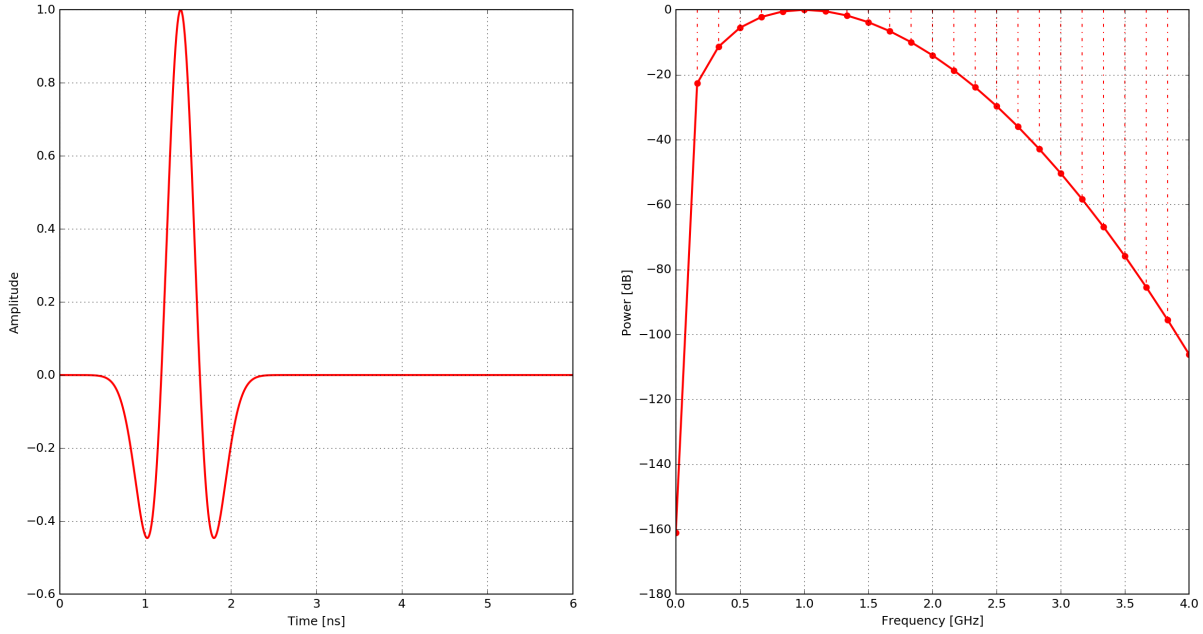


Fig. 6.6: Example of the `ricker` waveform - time domain and power spectrum.

sine

A single cycle of a sine waveform.

$$W(t) = R\sin(2\pi ft)$$

and

$$R = \begin{cases} 1 & \text{if } ft \leq 1, \\ 0 & \text{if } ft > 1. \end{cases}$$

f is the frequency

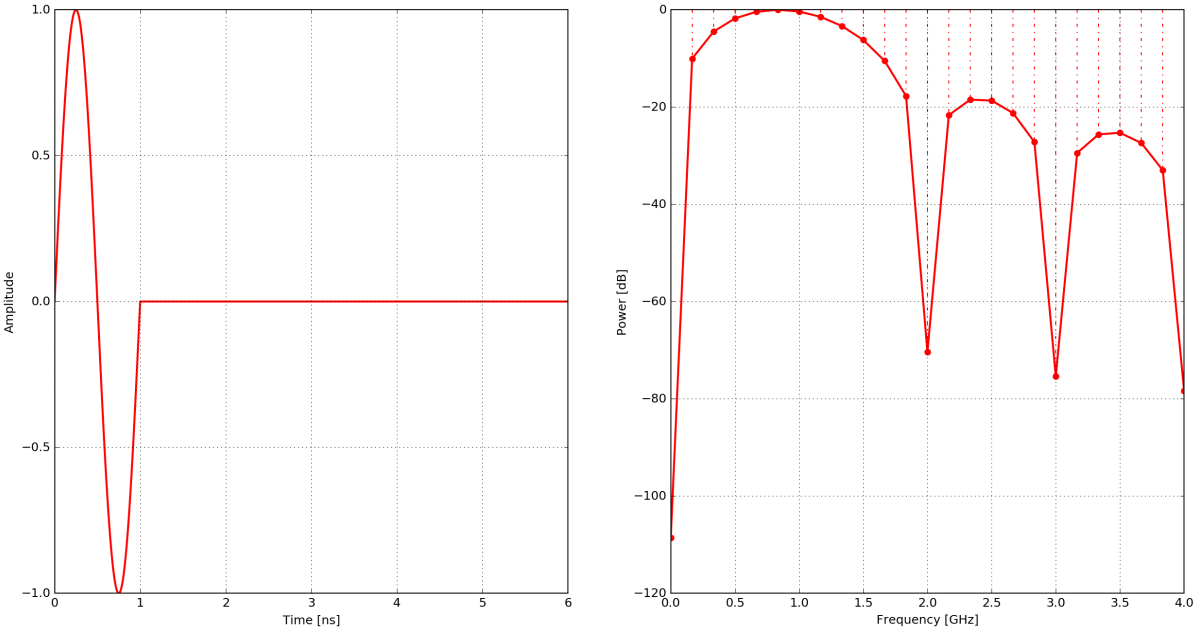


Fig. 6.7: Example of the `sine` waveform - time domain and power spectrum.

contsine

A continuous sine waveform. In order to avoid introducing noise into the calculation the amplitude of the waveform is modulated for the first cycle of the sine wave (ramp excitation).

$$W(t) = R\sin(2\pi ft)$$

and

$$R = \begin{cases} R_c ft & \text{if } R \leq 1, \\ 1 & \text{if } R > 1. \end{cases}$$

where R_c is set to 0.25 and f is the frequency.

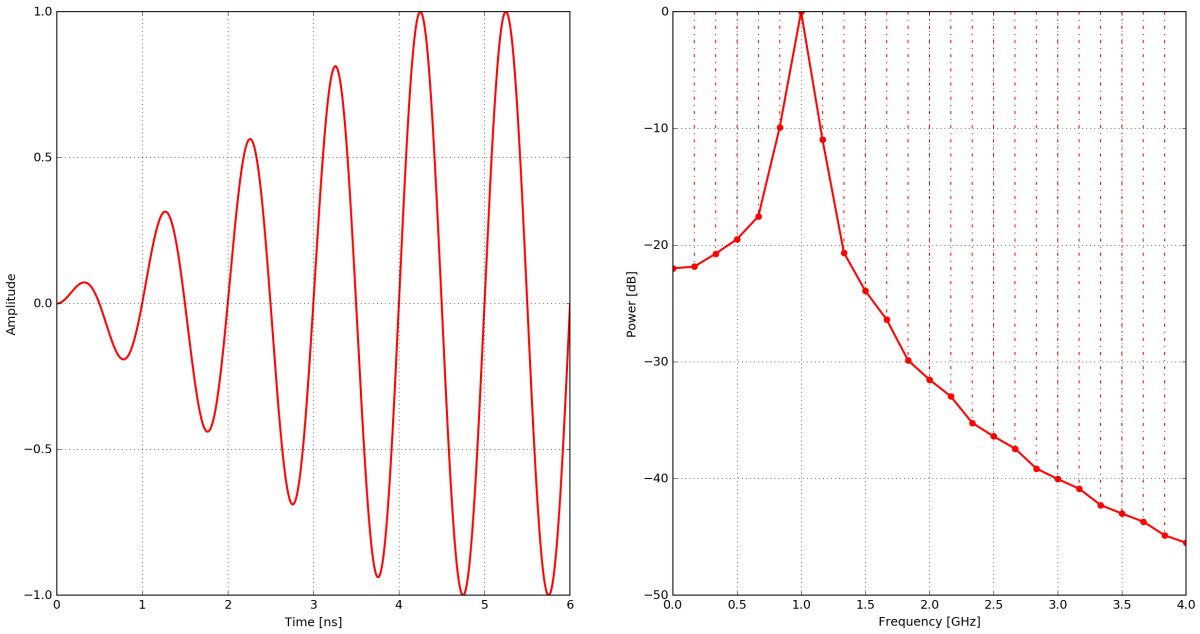


Fig. 6.8: Example of the `contsine` waveform - time domain and power spectrum.

CHAPTER 7

File utilities

This section provides information on how to use the other Python modules in the `tools` package to help manage gprMax files.

7.1 inputfile_old2new.py

This modules assists with the process of migrating input files from the syntax of the old (pre v.3) version of gprMax to the new syntax. It will produce a new input file with the old syntax (attempted to be!) translated to the new syntax. Usage (from the top-level gprMax directory) is:

```
python -m tools.inputfile_old2new inputfile
```

where `inputfile` is the name of input file including the path.

7.2 outputfiles_merge.py

gprMax produces a separate output file for each trace (A-scan) in a B-scan. This module combines the separate output files into a single file, and can remove the separate output files afterwards. Usage (from the top-level gprMax directory) is:

```
python -m tools.outputfiles_merge basefilename --remove-files
```

where:

- `basefilename` is the base name file of the output file series, e.g. for `myoutput1.out`, `myoutput2.out` the base file name would be `myoutput`
- `remove-files` is an optional argument (flag) that when given will remove the separate output files after the merge.

7.3 convert_png2h5.py

This module enables a Portable Network Graphics (PNG) image file to be converted into a HDF5 file that can then be used to import geometry into gprMax (see the `#geometry_objects_read` command for information

on how to use the HDF5 file with a materials file to import the geometry). The resulting geometry will be 2D but maybe extended in the z-(invariate) direction if a 3D model was desired. Usage (from the top-level gprMax directory) is:

```
python -m tools.convert_png2h5 imagefile dxdydz
```

where:

- `imagefile` is the name of the PNG image file including the path
- `dxdydz` is the spatial discretisation to be used in the model

There is an optional command line argument:

- `-zcells` is the number of cells to extend the geometry in the z-(invariate) direction of the model

For example create a HDF5 geometry objects file from the PNG image `my_layers.png` with a spatial discretisation of $\Delta x = \Delta y = \Delta z = 0.002$ metres, and extending 150 cells in the z-(invariate) direction of the model:

```
python -m tools.convert_png2h5 my_layers.png 0.002 0.002 0.002 -zcells 150
```

The module will display the PNG image and allow the user to select colours that will be used to define discrete materials in the model. When the user has finished selecting colours the window should be closed, whereupon the HDF5 file will be written.

CHAPTER 8

Scripting the input file

The input file has now been made scriptable by permitting blocks of Python code to be specified between `#python` and `#end_python` commands. The code is executed when the input file is read by gprMax. You don't need any external tools, such as MATLAB, to generate larger, more complex input files for building intricate models. Python scripting means that gprMax now includes *libraries of more complex objects, such as antennas*, that can be easily inserted into a model. You can also access a number of built-in constants from your Python code.

8.1 Constants/variables

You can access the following built-in constants from your Python code:

- `c` is the speed of light in vacuum $c = 2.9979245 \times 10^8$ m/s
- `e0` is the permittivity of free space $\epsilon_0 = 8.854187 \times 10^{-12}$ F/m
- `m0` is the permeability of free space $\mu_0 = 1.256637 \times 10^{-6}$ H/m
- `z0` is the impedance of free space $z_0 = 376.7303134$ Ohms

You can access the following built-in variables from your Python code:

- `current_model_run` is the current run number of the model that is been executed.
- `inputfile` is the path and name of the input file.
- `number_model_runs` is the total number of runs specified when the model was initially executed, i.e. from `python -m gprMax my_input_file -n number_of_model_runs`

8.2 Functions for input commands

To make it easier to create commands within a block of Python code, there is a built-in module which contains some of the most commonly used input commands in functional form. For example, to use Python to generate a series of cylinders in a model:

```
#python:
from gprMax.input_cmd_funcs import *
domain = domain(0.2, 0.2, 0.2)
```

(continues on next page)

(continued from previous page)

```
for x in range(0, 8):
    cylinder(0.02 + x * 0.02, 0.05, 0, 0.020 + x * 0.02, 0.05, domain[2], 0.005,
↪'pec')
#end_python:
```

The `domain` function will print the `#domain` command to the input file and return a variable with the extent of the domain that can be used elsewhere in a Python code block, e.g. in this case with the `cylinder` function. The `cylinder` function is just a functional version of the `#cylinder` command which prints it to the input file.

CHAPTER 9

OpenMP, MPI, and HPC

9.1 OpenMP

The most computationally intensive parts of gprMax, which are the FDTD solver loops, have been parallelised using OpenMP (http://openmp.org) which supports multi-platform shared memory multiprocessing.

By default gprMax will try to determine and use the maximum number of OpenMP threads (usually the number of physical CPU cores) available on your machine. You can override this behaviour in two ways: firstly, gprMax will check to see if the `#num_threads` command is present in your input file; if not, gprMax will check to see if the environment variable `OMP_NUM_THREADS` is set. This can be useful if you are running gprMax in a High-Performance Computing (HPC) environment where you might not want to use all of the available CPU cores.

9.2 MPI

The Message Passing Interface (MPI) has been utilised to implement a simple task farm that can be used to distribute a series of models as independent tasks. This can be useful in many GPR simulations where a B-scan (composed of multiple A-scans) is required. Each A-scan can be task-farmed as a independent model. Within each independent model OpenMP threading will continue to be used (as described above). Overall this creates what is know as a mixed mode OpenMP/MPI job.

By default the MPI task farm functionality is turned off. It can be switched on using the `-mpi` command line flag. MPI requires an installation of the `mpi4py` Python package, which itself depends on an underlying MPI installation, usually OpenMPI (http://www.open-mpi.org). On Microsoft Windows `mpi4py` requires Microsoft MPI 6 (https://www.microsoft.com/en-us/download/details.aspx?id=47259).

9.3 HPC job scripts

HPC environments usually require jobs to be submitted to a queue using a job script. The following are examples of job scripts for a HPC environment that uses Open Grid Scheduler/Grid Engine (http://gridscheduler.sourceforge.net/index.html), and are intended as general guidance to help you get started. Using gprMax in an HPC environment is heavily dependent on the configuration of your specific HPC/cluster, e.g. the names of parallel environments (`-pe`) and compiler modules will depend on how they were defined by your system administrator.

9.3.1 OpenMP example

`gprmax_omp.sh`

Here is an example of a job script for running models, e.g. A-scans to make a B-scan, one after another on a single cluster node. This is not as beneficial as the OpenMP/MPI example, but it can be a helpful starting point when getting the software running in your HPC environment. The behaviour of most of the variables is explained in the comments in the script.

```
#!/bin/sh
#############################################################################
##
### Change to current working directory:
#$ -cwd

### Specify runtime (hh:mm:ss):
#$ -l h_rt=01:00:00

### Email options:
#$ -m ea -M joe.bloggs@email.com

### Parallel environment ($NSLOTS):
#$ -pe sharedmem 16

### Job script name:
#$ -N gprmax_omp.sh
#############################################################################
##

### Initialise environment module
. /etc/profile.d/modules.sh

### Load and activate Anaconda environment for gprMax, i.e. Python 3 and required␣
↪packages
module load anaconda
source activate gprMax

### Set number of OpenMP threads for each gprMax model
export OMP_NUM_THREADS=16

### Run gprMax with input file
cd $HOME/gprMax
python -m gprMax mymodel.in -n 10
```

In this example 10 models will be run one after another on a single node of the cluster (on this particular cluster a single node has 16 cores/threads available). Each model will be parallelised using 16 OpenMP threads.

9.3.2 OpenMP/MPI example

`gprmax_omp_mpi.sh`

Here is an example of a job script for running models, e.g. A-scans to make a B-scan, distributed as independent tasks in a HPC environment using MPI. The behaviour of most of the variables is explained in the comments in the script.

```
#!/bin/sh
#############################################################################
##
### Change to current working directory:
#$ -cwd

```

(continues on next page)

(continued from previous page)

```
### Specify runtime (hh:mm:ss):
#$ -l h_rt=01:00:00

### Email options:
#$ -m ea -M joe.bloggs@email.com

### Resource reservation:
#$ -R y

### Parallel environment ($NSLOTS):
#$ -pe mpi 176

### Job script name:
#$ -N gprmax_omp_mpi.sh
################################################################################
##

### Initialise environment module
. /etc/profile.d/modules.sh

### Load and activate Anaconda environment for gprMax, i.e. Python 3 and required␣
packages
module load anaconda
source activate gprMax

### Load OpenMPI
module load openmpi

### Set number of OpenMP threads per MPI task (each gprMax model)
export OMP_NUM_THREADS=16

### Run gprMax with input file
cd $HOME/gprMax
python -m gprMax mymodel.in -n 10 -mpi 11
```

In this example 10 models will be distributed as independent tasks in a HPC environment using MPI.

The `-mpi` flag is passed to gprMax which takes the number of MPI tasks to run. This should be the number of models (worker tasks) plus one extra for the master task.

The `NSLOTS` variable which is required to set the total number of slots/cores for the parallel environment `-pe mpi` is usually the number of MPI tasks multiplied by the number of OpenMP threads per task. In this example the number of MPI tasks is 11 and number of OpenMP threads per task is 16, so 176 slots are required.

9.3.3 Job array example

`gprmax_omp_jobarray.sh`

Here is an example of a job script for running models, e.g. A-scans to make a B-scan, using the job array functionality of Open Grid Scheduler/Grid Engine. A job array is a single submit script that is run multiple times. It has similar functionality, for gprMax, to using the aforementioned MPI task farm. The behaviour of most of the variables is explained in the comments in the script.

```
#!/bin/sh
################################################################################
##
### Change to current working directory:
#$ -cwd

### Specify runtime (hh:mm:ss):
```

(continues on next page)

(continued from previous page)

```
#$ -l h_rt=01:00:00

### Parallel environment ($NSLOTS):
#$ -pe sharedmem 16

### Job array and task IDs
#$ -t 1-11

### Job script name:
#$ -N gprmax_omp_jobarray.sh
################################################################################
##

### Initialise environment module
. /etc/profile.d/modules.sh

### Load and activate Anaconda environment for gprMax, i.e. Python 3 and required␣
packages
module load anaconda
source activate gprMax

### Set number of OpenMP threads for each gprMax model
export OMP_NUM_THREADS=16

### Run gprMax with input file
cd $HOME/gprMax
python -m gprMax mymodel.in -n 10 -task $SGE_TASK_ID
```

The `-t` tells Grid Engine that we are using a job array followed by a range of integers which will be the IDs for each individual task (model). Task IDs must start from 1, and the total number of tasks in the range should correspond to the number of models you want to run, i.e. the integer with the `-n` flag passed to gprMax. The `-task` flag is passed to gprMax to tell it we are using a job array, along with the specific number of the task (model) with the environment variable `$SGE_TASK_ID`.

A job array means that exactly the same submit script is going to be run multiple times, the only difference between each run is the environment variable `$SGE_TASK_ID`.

9.3.4 Eddie

Eddie is the Edinburgh Compute and Data Facility (ECDF) (http://www.ed.ac.uk/information-services/research-support/research-computing/ecdf/high-performance-computing) run by the University of Edinburgh (http://www.ed.ac.uk). The following are useful notes to get gprMax installed and running on eddie3 (the third iteration of the cluster):

- Git is already installed on eddie3, so you don't need to install it through Anaconda, you can proceed directly to cloning the gprMax GitHub repository with `git clone https://github.com/gprMax/gprMax.git`
- Anaconda is already installed as an application module on eddie3. You should follow these instructions (https://www.wiki.ed.ac.uk/display/ResearchServices/Anaconda) to ensure Anaconda environments will be created in a suitable location (not your home directory as you will rapidly run out of space). Before you proceed to create the Anaconda environment for gprMax you must make sure the OpenMPI module is loaded with `module load openmpi`. This is neccessary so that the `mpi4py` Python module is correctly linked to OpenMPI. You can then create the Anaconda environment with `conda env create -f conda_env.yml`
- You should then activate the gprMax Anaconda environment, and build and install gprMax according the standard installation procedure.
- The previous job submission example scripts for OpenMP and OpenMP/MPI should run on eddie3.

- The `NSLOTS` variable for the total number of slots/cores for the parallel environment `-pe mpi` must be specified as a multiple of 16 (the total number of cores/threads available on a single node), e.g. 61 MPI tasks each using 4 threads would require a total 244 slots/cores. This must be rounded up to the nearest multiple of 16, e.g. 256.

CHAPTER 10

GPGPU

The most computationally intensive parts of gprMax, which are the FDTD solver loops, can optionally be executed using General-purpose computing on graphics processing units (GPGPU). This has been achieved through use of the NVIDIA CUDA programming environment, therefore a NVIDIA CUDA-Enabled GPU (https://developer.nvidia.com/cuda-gpus) is required to take advantage of the GPU-based solver.

10.1 Extra installation steps for GPU usage

The following steps provide guidance on how to install the extra components to allow gprMax to run on your GPU:

1. Install the NVIDIA CUDA Toolkit (https://developer.nvidia.com/cuda-toolkit). You can follow the Installation Guides in the NVIDIA CUDA Toolkit Documentation (http://docs.nvidia.com/cuda/index.html#installation-guides)
2. Install the pycuda Python module. Open a Terminal (Linux/macOS) or Command Prompt (Windows), navigate into the top-level gprMax directory, and if it is not already active, activate the gprMax conda environment `source activate gprMax` (Linux/macOS) or `activate gprMax` (Windows). Run `pip install pycuda`

10.2 Running gprMax using GPU(s)

Open a Terminal (Linux/macOS) or Command Prompt (Windows), navigate into the top-level gprMax directory, and if it is not already active, activate the gprMax conda environment `source activate gprMax` (Linux/macOS) or `activate gprMax` (Windows)

Run one of the test models:

```
(gprMax)$ python -m gprMax user_models/cylinder_Ascan_2D.in -gpu
```

Note: If you want to select a specific GPU card on your system, you can specify an integer after the `-gpu` flag. The integer should be the NVIDIA CUDA device ID for a specific GPU card. If it is not specified it defaults to device ID 0.

10.2.1 Combining MPI and GPU usage

Message Passing Interface (MPI) has been utilised to implement a simple task farm that can be used to distribute a series of models as independent tasks. This is described in more detail in the *OpenMP, MPI, HPC section*. MPI can be combined with the GPU functionality to allow a series models to be distributed to multiple GPUs on the same machine (node). For example, to run a B-scan that contains 60 A-scans (traces) on a system with 4 GPUs:

```
(gprMax)$ python -m gprMax user_models/cylinder_Bscan_2D.in -n 60 -mpi 5 -gpu
```

Note: The argument given with *-mpi* is number of MPI tasks, i.e. master + workers, for MPI task farm. So in this case, 1 master (CPU) and 4 workers (GPU cards).

User libraries is a sub-package where useful Python modules contributed by users are stored.

CHAPTER 11

GPR antenna models

11.1 Information

The module features models of antennas similar to commercial GPR antennas. The following antenna models are included:

Manufacturer/Model	Dimensions	Resolution(s)	Author/Contact	Attribution/Cite
GSSI 1.5GHz (Model 5100)	170x108x145mm	1, 2mm	Craig Warren (craig.warren@northumbria.ac.uk), Northumbria University, UK	1,2
MALA 1.2GHz	184x109x146mm	1, 2mm	Craig Warren (craig.warren@northumbria.ac.uk), Northumbria University, UK	1
GSSI 400MHz	300x300x170mm	1, 2mm	Sam Stadler (Sam.Stadler@liag-hannover.de), Leibniz Institute for Applied Geophysics (https://www.leibniz-liag.de/en/research/methods/electromagnetic-methods/ground-penetrating-radar/guided-gpr-waves.html), Germany	3

Attributions/citations:

1. Warren, C., Giannopoulos, A. (2011). Creating finite-difference time-domain models of commercial ground-penetrating radar antennas using Taguchi's optimization method. *Geophysics*, 76(2), G37-G47. (http://dx.doi.org/10.1190/1.3548506)

2. Giannakis, I., Giannopoulos, A., & Warren, C. (2018). Realistic FDTD GPR antenna models optimised using a novel linear/non-linear Full Waveform Inversion. *IEEE Transactions on Geoscience and Remote Sensing*, . ()

3. Stadler, S. (2017). A Forward Modeling Study for the Investigation of the Vertical Water-Content Distribution of Using Guided GPR Waves (Master's Thesis, Freiberg University of Mining and Technology, Germany). (https://tinyurl.com/y6vdab22)

11.2 Module overview

- `GSSI.py` is a module containing models of antennas similar to those manufactured by Geophysical Survey Systems, Inc. (GSSI) (http://www.geophysical.com).
- `MALA.py` is a module containing models of antennas similar to those manufactured by MALA Geoscience (http://www.malags.com/).

Descriptions of how the models were created can be found in the aforementioned attributions.

11.3 How to use the module

The antenna models can be accessed from within a block of Python code in an input file. The models are inserted at location x,y,z. The coordinates are relative to the geometric centre of the antenna in the x-y plane and the bottom of the antenna skid in the z direction. The models must be used with cubic spatial resolutions of either 0.5mm (GSSI 400MHz antenna only), 1mm (default), or 2mm by setting the keyword argument, e.g. `resolution=0.002`. The antenna models can be rotated 90 degrees counter-clockwise (CCW) in the x-y plane by setting the keyword argument `rotate90=True`.

Note: If you are moving an antenna model within a simulation, e.g. to generate a B-scan, you should ensure that the step size you choose is a multiple of the spatial resolution of the simulation. Otherwise when the position of antenna is converted to cell coordinates the geometry maybe altered.

11.3.1 Example

To include an antenna model similar to a GSSI 1.5 GHz antenna at a location 0.125m, 0.094m, 0.100m (x,y,z) using a 2mm cubic spatial resolution:

```
#python:
from user_libs.antennas.GSSI import antenna_like_GSSI_1500
antenna_like_GSSI_1500(0.125, 0.094, 0.100, resolution=0.002)
#end_python:
```

User libraries is a sub-package where useful Python modules contributed by users are stored.

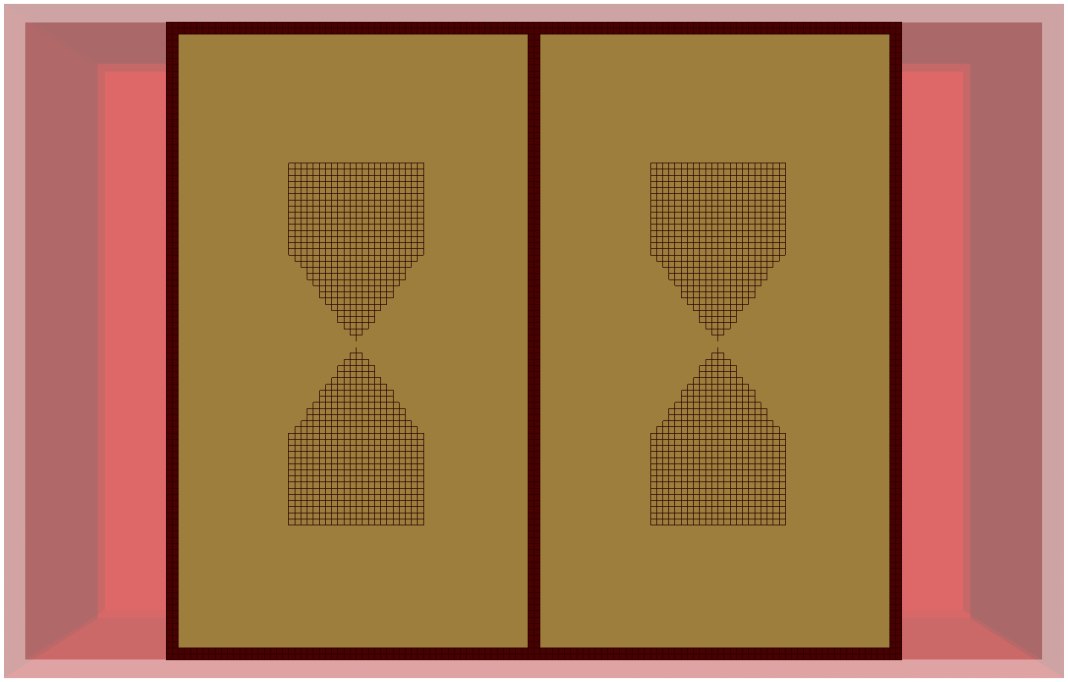


Fig. 11.1: FDTD geometry mesh showing an antenna model similar to a GSSI 1.5 GHz antenna (skid removed for illustrative purposes).

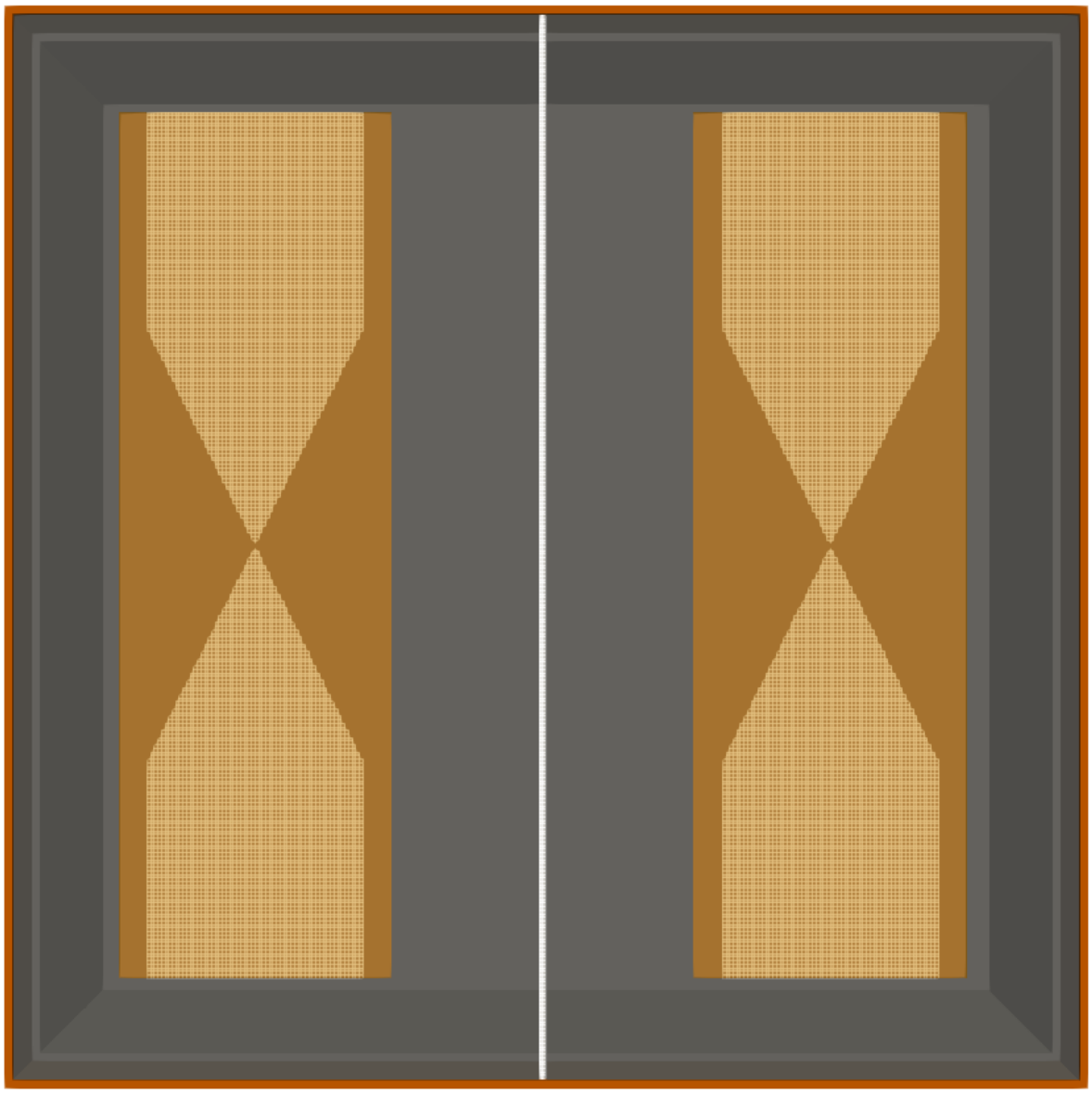


Fig. 11.2: FDTD geometry mesh showing an antenna model similar to a GSSI 400 MHz antenna (skid removed for illustrative purposes).

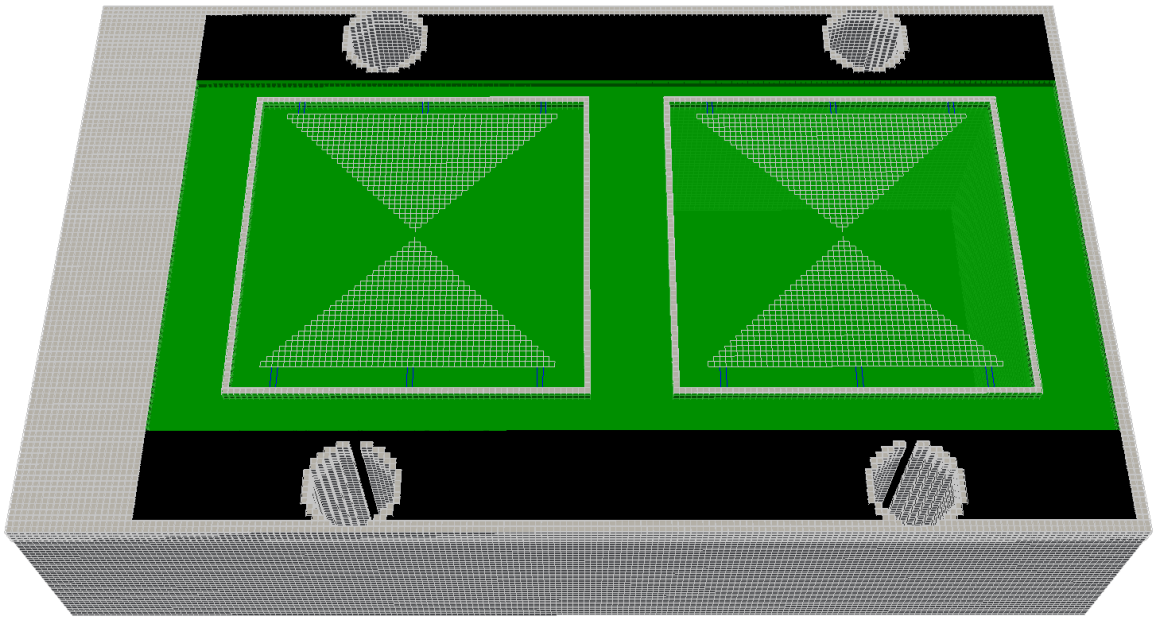


Fig. 11.3: FDTD geometry mesh showing an antenna model similar to a MALA 1.2GHz antenna (skid removed for illustrative purposes).

CHAPTER 12

Antenna patterns

12.1 Information

Author/Contact: Craig Warren (craig.warren@northumbria.ac.uk), Northumbria University, UK

Attribution/cite: Warren, C., Giannopoulos, A. (2016). Characterisation of a Ground Penetrating Radar Antenna in Lossless Homogeneous and Lossy Heterogeneous Environments. *Signal Processing* (http://dx.doi.org/10.1016/j.sigpro.2016.04.010)

The package features contains modules to help calculate, process, and visualise field patterns from simulations that contain models of antennas.

Warning: Although the principals of calculating and visualising field patterns are straightforward, this package should be used with care. The package:

- Does not calculate/plot conventional field patterns, i.e. at a single frequency. It uses a measure of the total energy of the electric field at a certain angle and radius, see http://dx.doi.org/10.1016/j.jappgeo.2013.08.001
- Requires knowledge of Python to contruct input files with antenna models and positioning of receivers, as well as to edit/modify the saving and processing modules
- Can require simulations that demand significant computational resource depending on the distance from the antenna at which the field patterns are observed, e.g. the example models, set with a maximum observation distance of 0.6m, require ~30GB of RAM

12.2 Module overview

- `initial_save.py` is a module that calculates and stores (in a Numpy file) the field patterns from the output file of a simulation.
- `plot_fields.py` is a module that plots the field patterns. It should be used after the field pattern data has been processed and stored using the `initial_save.py` module.

The package has been designed to work with input files that follow examples found in the `user_models` directory:

- `antenna_like_GSSI_1500_patterns_E.in` is an input file that includes an antenna model similar to a GSSI 1.5 GHz antenna and receivers to calculate a field pattern in the principal E-plane of the antenna
- `antenna_like_GSSI_1500_patterns_H.in` is an input file that includes an antenna model similar to a GSSI 1.5 GHz antenna and receivers to calculate a field pattern in the principal H-plane of the antenna

12.3 How to use the module

- Firstly you should familiarise yourself with the example model input file. Edit the input file as desired and run one of the simulations for either E-plane or H-plane patterns.
- Whilst the simulation is running edit the 'user configurable paramters' sections of the `initial_save.py` and `plot_fields.py` modules to match the setup of the simulation.
- Once the simulation has completed, run the `initial_save.py` module on the output file, e.g. for the E-plane `python -m user_libs.antenna_patterns.initial_save user_models/antenna_like_GSSI_1500_patterns_E_Er5.out`. This will produce a Numpy file containing the field pattern data.
- Plot the field pattern data by running the `plot_fields.py` module on the Numpy file, e.g. for the E-plane `python -m user_libs.antenna_patterns.plot_fields user_models/antenna_like_GSSI_1500_patterns_E_Er5.npy`

Tip: If you want to create different plots you just need to edit and re-run the `plot_fields.py` module on the Numpy file, i.e. you don't have to re-process the output file.

User libraries is a sub-package where useful Python modules contributed by users are stored.

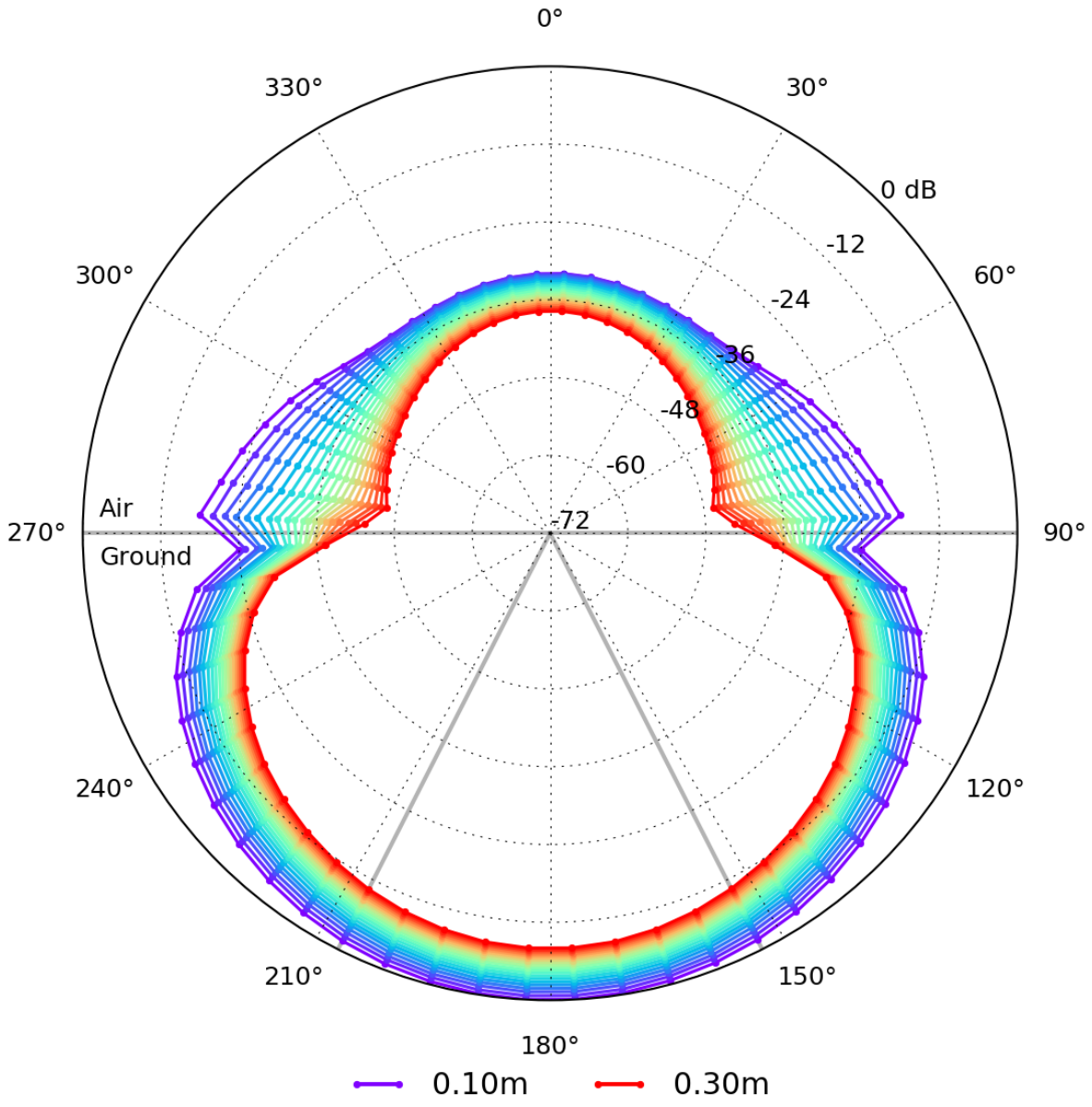


Fig. 12.1: Example of the E-plane pattern from a simulation containing an antenna model similar to a GSSI 1.5 GHz antenna over a homogeneous, lossless half-space with a relative permittivity of five.

CHAPTER 13

AustinMan/AustinWoman

13.1 Information

Authors: Jackson W. Massey, Cemil S. Geyik, Jungwook Choi, Hyun-Jae Lee, Natcha Techachainiran, Che-Lun Hsu, Robin Q. Nguyen, Trevor Latson, Madison Ball, and Ali E. Yılmaz

Contact: Ali E. Yılmaz (ayilmaz@mail.utexas.edu), The University of Texas at Austin

License: Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License (http://creativecommons.org/licenses/by-nc-nd/3.0/)

Attribution/cite: Please follow the instructions at http://web.corral.tacc.utexas.edu/AustinManEMVoxels/AustinMan/citing_the_model/index.html

AustinMan and AustinWoman (http://bit.ly/AustinMan) are open source electromagnetic voxel models of the human body, which are developed by the Computational Electromagnetics Group (http://www.ece.utexas.edu/research/areas/electromagnetics-acoustics) at The University of Texas at Austin (http://www.utexas.edu). The models are based on data from the National Library of Medicine's Visible Human Project (https://www.nlm.nih.gov/research/visible/visible_human.html).

The following whole body models are available.

Model	Resolution (mm ^{3})	Dimensions (cells)
AustinMan	8x8x8	86 x 47 x 235
AustinMan	4x4x4	171 x 94 x 470
AustinMan	2x2x2	342 x 187 x 939
AustinMan	1x1x1	683 x 374 x 1877
AustinWoman	8x8x8	86 x 47 x 217
AustinWoman	4x4x4	171 x 94 x 433
AustinWoman	2x2x2	342 x 187 x 865
AustinWoman	1x1x1	683 x 374 x 1730

13.2 Package overview

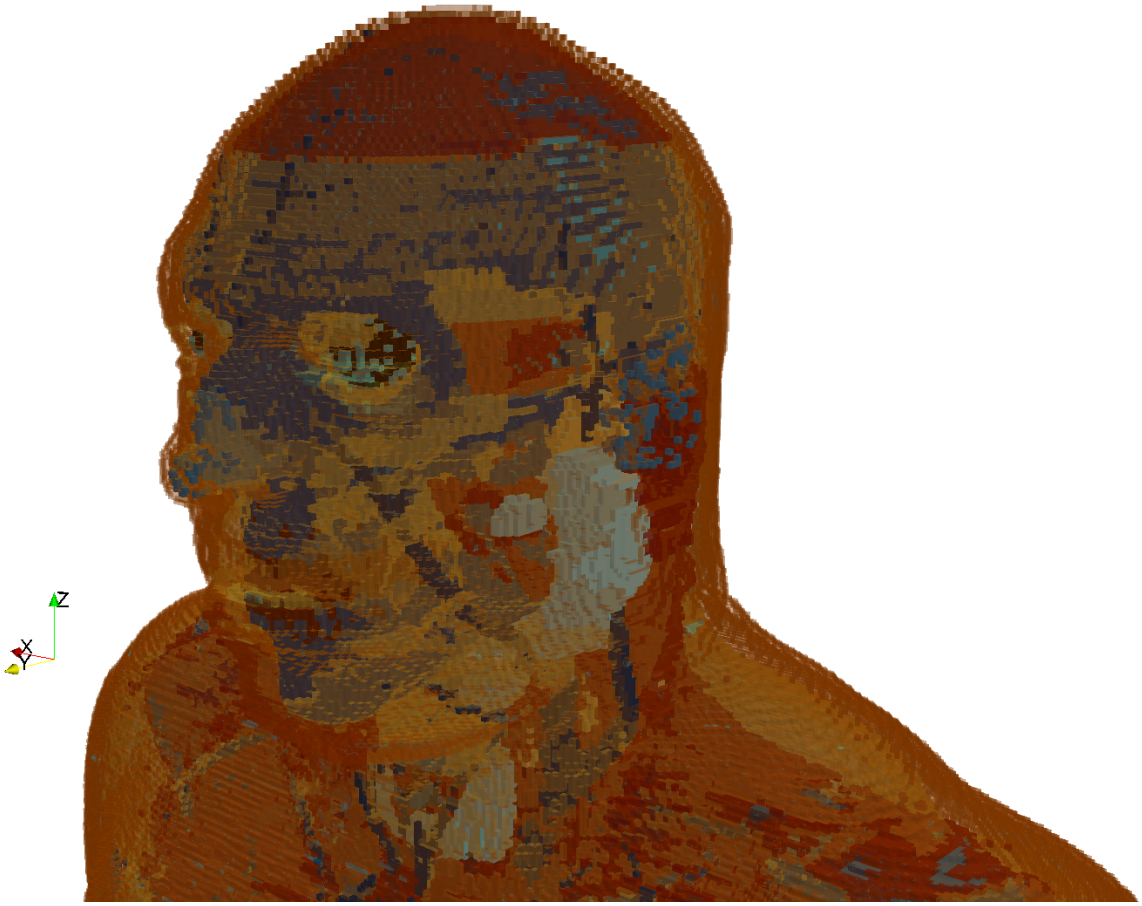


Fig. 13.1: FDTD geometry mesh showing the head of the AustinMan model (2x2x2mm ^{3}).

```
AustinManWoman_materials.txt
AustinManWoman_materials_dispersive.txt
head_only_h5.py
```

- `AustinManWoman_materials.txt` is a text file containing non-dispersive material properties at 900 MHz (http://niremf.ifac.cnr.it/tissprop/).
- `AustinManWoman_materials_dispersive.txt` is a text file containing dispersive material properties using a 3-pole Debye model (http://dx.doi.org/10.1109/LMWC.2011.2180371).

Note:

- The main body tissues are described using a 3-pole Debye model, but not all materials have a dispersive description.
- The dispersive material properties can only be used with the 1x1x1mm or 2x2x2mm AustinMan/Woman models. This is because the time step of the model must always be less than any of the relaxation times of the poles of the Debye models used for the dispersive material properties.

- `head_only_h5.py` is a script to assist with creating a model of only the head from a full body AustinMan/Woman model.

13.3 How to use the models

The AustinMan and AustinWoman models themselves are not included in the user libraries sub-package.

- Download a HDF5 file (.h5) of AustinMan or AustinWoman (http://bit.ly/AustinMan) at the resolution you wish to use

To insert either AustinMan or AustinWoman models into a simulation use the `#geometry_objects_read`.

13.3.1 Example

To insert a 2x2x2mm ^{3} AustinMan with the lower left corner 40mm from the origin of the domain, using disperive material properties, and with no dielectric smoothing, use the command:

```
#geometry_objects_read: 0.04 0.04 0.04 ../user_libs/AustinManWoman/AustinMan_v2.3_
↪2x2x2.h5 ../user_libs/AustinManWoman/AustinManWoman_materials_dispersive.txt
```

For further information on the `#geometry_objects_read` see the section on object contruction commands in the *Input commands section*.

User libraries is a sub-package where useful Python modules contributed by users are stored.

Fig. 13.2: FDTD geometry mesh showing the AustinMan body model (2x2x2mm ^{3}).

CHAPTER 14

Materials

14.1 Information

The module is intended to provide a database of electromagnetic properties of different materials. It currently includes the following material libraries:

- `eccosorb.txt` contains information on some of the Eccosorb LS series (http://www.eccosorb.com/products-eccosorb-ls.htm) of electromagnetic absorber materials manufactured by Laird NV (http://www.eccosorb.eu) (formerly Emerson & Cuming Microwave Products NV). LS 14, 16, 18, 20, 22, 26, 28, and 30 are included. They are simulated using a 3-pole Debye model.

14.2 How to use the module

14.2.1 Example

The simplest way to access any of the material libraries is to use the `#include_file` command, after which the name of the material can then be used with any object construction command:

```
#include_file: user_libs/materials/eccosorb.txt
#box: 0 0 0 0.5 0.5 0.5 eccosorb_ls22
```

14.3 Eccosorb

Eccosorb (http://www.eccosorb.eu) are electromagnetic absorber materials manufactured by Laird NV (http://www.eccosorb.eu) (formerly Emerson & Cuming Microwave Products NV). Currently models for some of the LS series (14, 16, 18, 20, 22, 26, 28, and 30) are included in this library. The models were created by fitting a 3-pole Debye model to the real and imaginary parts of the relative permittivity taken from the manufacturers datasheet (http://www.eccosorb.com/Collateral/Documents/English-US/Electrical%20Parameters/ls%20parameters.pdf). The following figures show the fitting.

User libraries is a sub-package where useful Python modules contributed by users are stored.

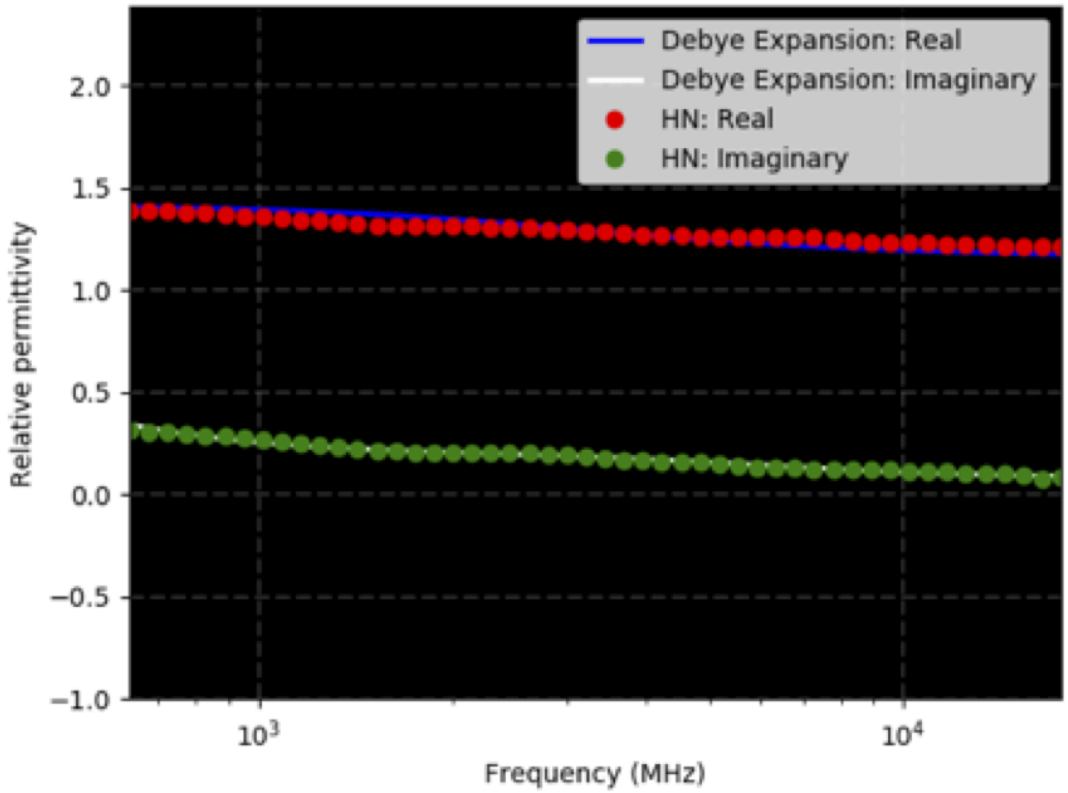


Fig. 14.1: 3-pole Debye fit for Eccosorb LS14 absorber (HN indicates data from manufacturer datasheet)

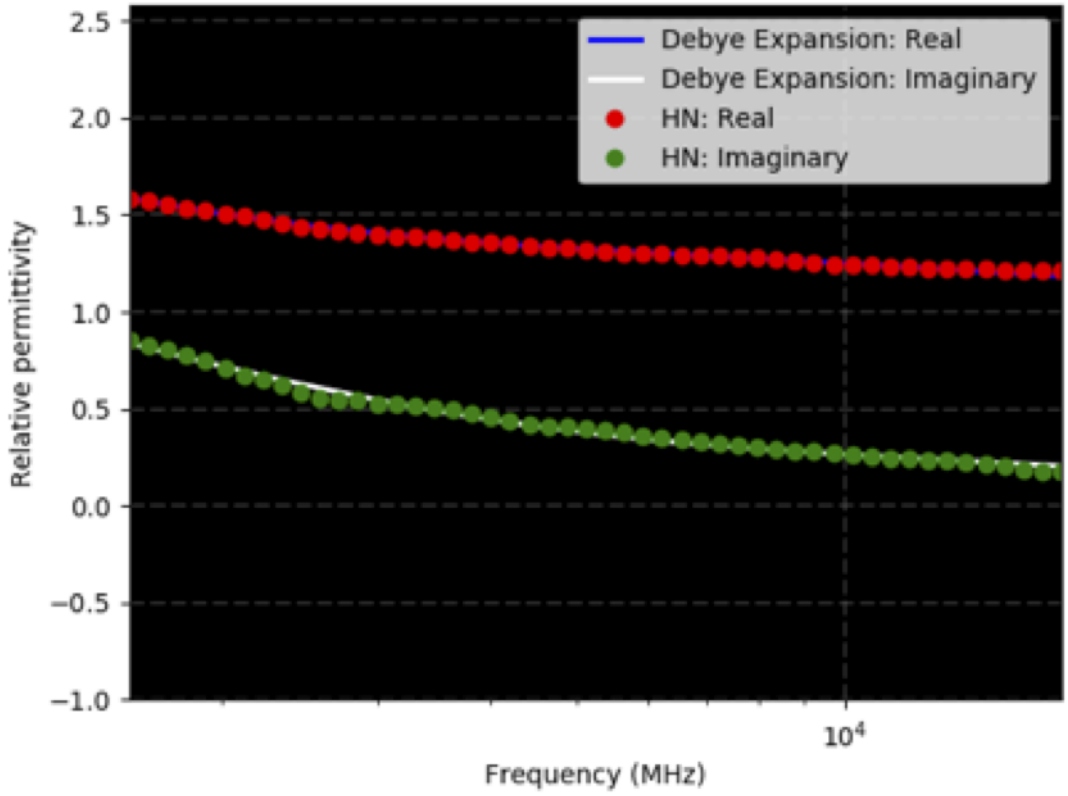


Fig. 14.2: 3-pole Debye fit for Eccosorb LS16 absorber (HN indicates data from manufacturer datasheet)

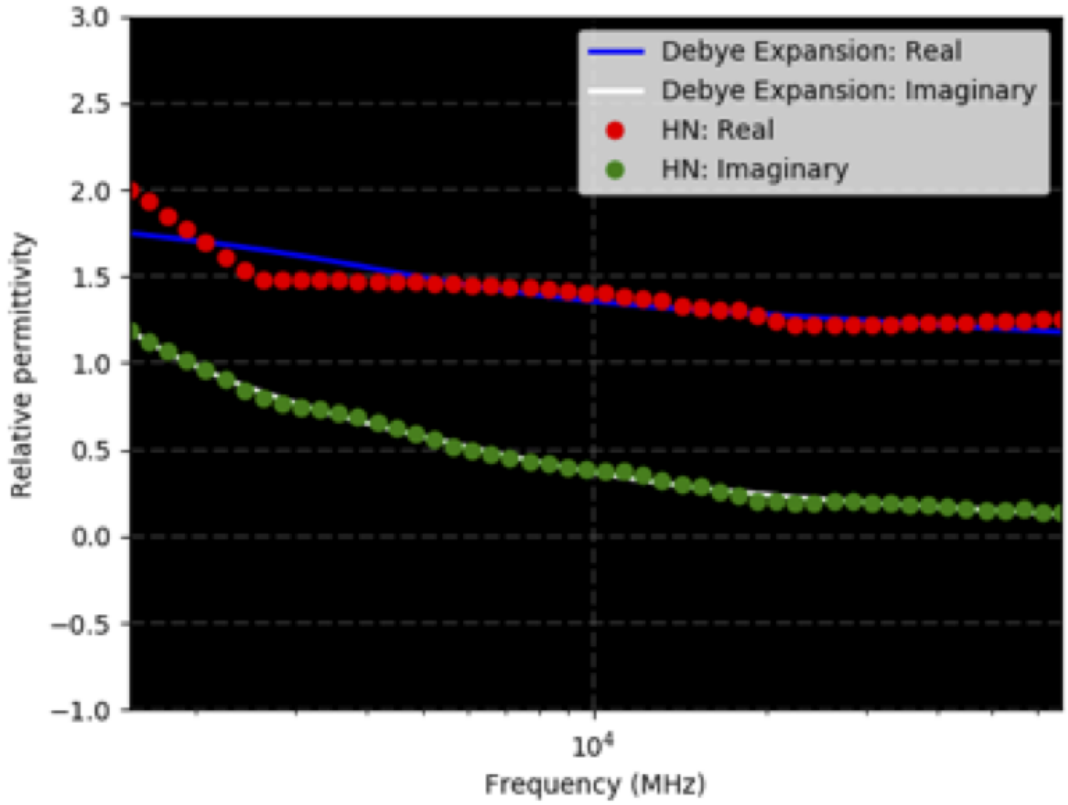


Fig. 14.3: 3-pole Debye fit for Eccosorb LS18 absorber (HN indicates data from manufacturer datasheet)

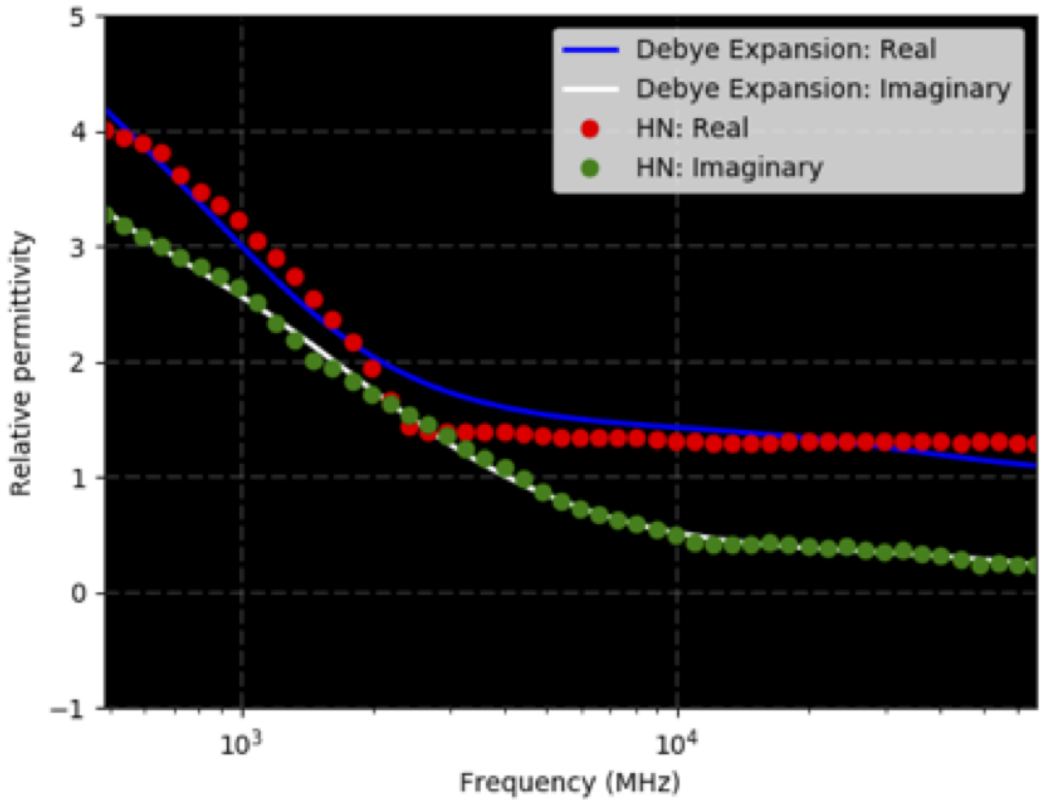


Fig. 14.4: 3-pole Debye fit for Eccosorb LS20 absorber (HN indicates data from manufacturer datasheet)

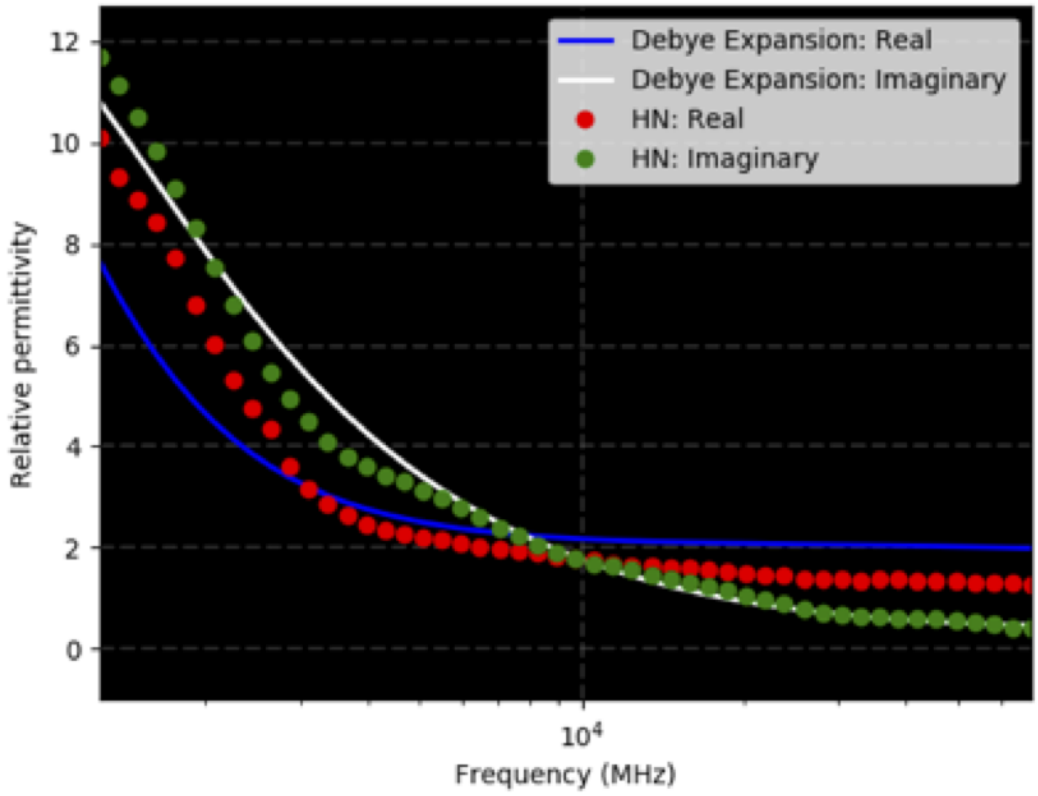


Fig. 14.5: 3-pole Debye fit for Eccosorb LS22 absorber (HN indicates data from manufacturer datasheet)

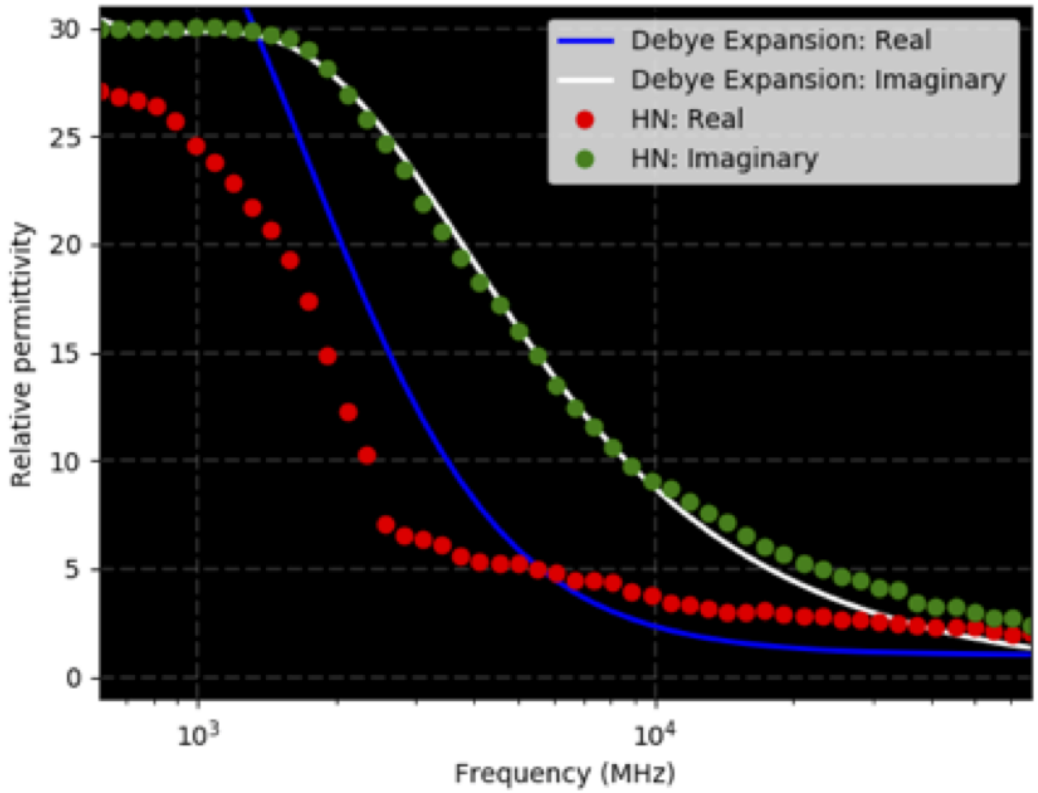


Fig. 14.6: 3-pole Debye fit for Eccosorb LS26 absorber (HN indicates data from manufacturer datasheet)

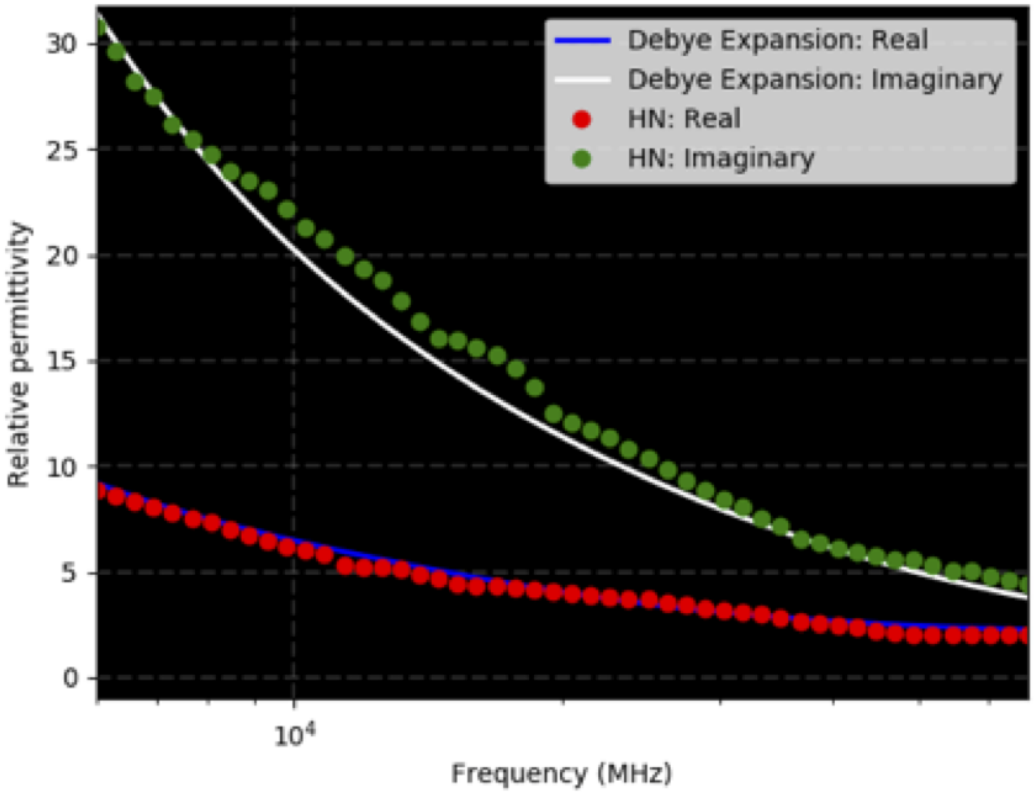


Fig. 14.7: 3-pole Debye fit for Eccosorb LS28 absorber (HN indicates data from manufacturer datasheet)

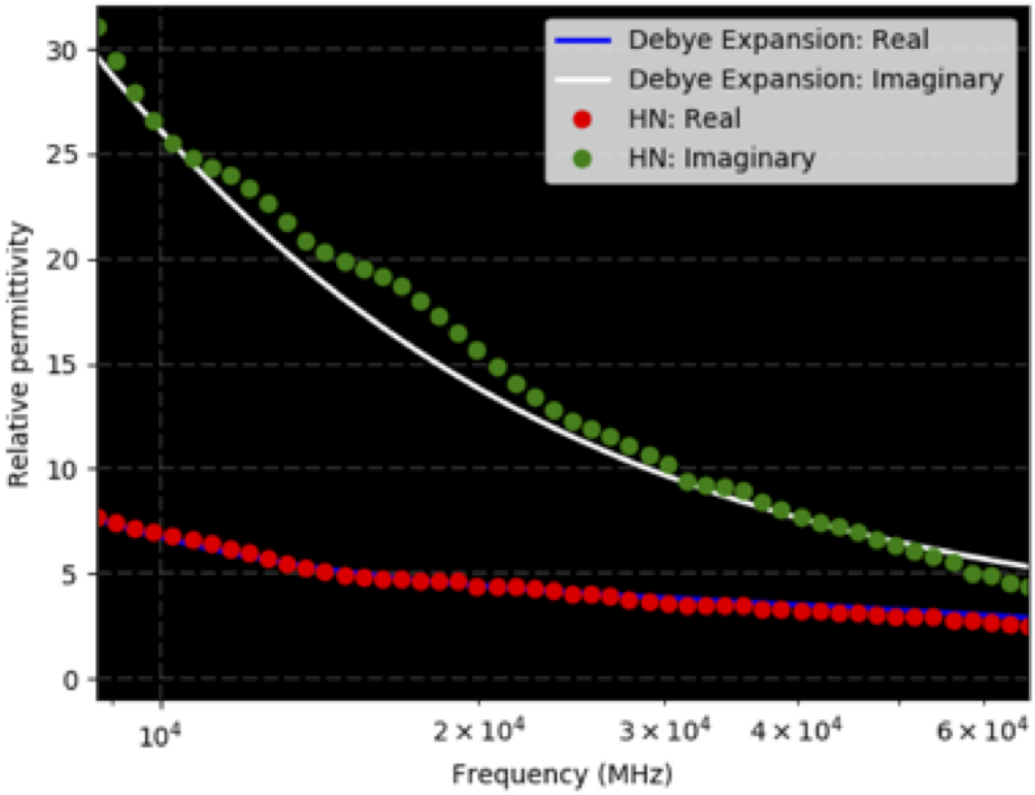


Fig. 14.8: 3-pole Debye fit for Eccosorb LS30 absorber (HN indicates data from manufacturer datasheet)

CHAPTER 15

Optimisation - Taguchi's method

15.1 Information

Author/Contact: Craig Warren (craig.warren@northumbria.ac.uk), Northumbria University, UK

Attribution/cite: Warren, C., Giannopoulos, A. (2011). Creating finite-difference time-domain models of commercial ground-penetrating radar antennas using Taguchi's optimization method. *Geophysics*, 76(2), G37-G47. (http://dx.doi.org/10.1190/1.3548506)

The package features an optimisation technique based on Taguchi's method. It allows users to define parameters in an input file and optimise their values based on a fitness function, for example it can be used to optimise material properties or geometry in a simulation.

Warning: This package combines a number of advanced features and should not be used without knowledge and familiarity of the underlying techniques. It requires:

- Knowledge of Python to contruct an input file to use with the optimisation
- Familiarity of optimisation techniques, and in particular Taguchi's method
- Careful sanity checks to be made throughout the process

15.1.1 Taguchi's method

Taguchi's method is based on the concept of the Orthogonal Array (OA) and has the following advantages:

- Simple to implement
- Effective in reduction of experiments
- Fast convergence speed
- Global optimum results
- Independence from initial values of optimisation parameters

Details of Taguchi's method in the context of electromagnetics can be found in *[WEN2007a]* and *[WEN2007b]*.

15.2 Package overview

```
antenna_bowtie_opt.in
fitness_functions.py
OA_9_4_3_2.npy
OA_18_7_3_2.npy
plot_results.py
```

- `antenna_bowtie_opt.in` is a example model of a bowtie antenna where values of loading resistors are optimised.
- `fitness_functions.py` is a module containing fitness functions. There are some pre-built ones but users should add their own here.
- `OA_9_4_3_2.npy` and `OA_18_7_3_2.npy` are NumPy archives containing pre-built OAs (http://neilsloane.com/oadir/)
- `plot_results.py` is a module for plotting the results, such as parameter values and convergence history, from an optimisation process when it has completed.

15.2.1 Implementation

The process by which Taguchi's method optimises parameters is illustrated in the following figure:

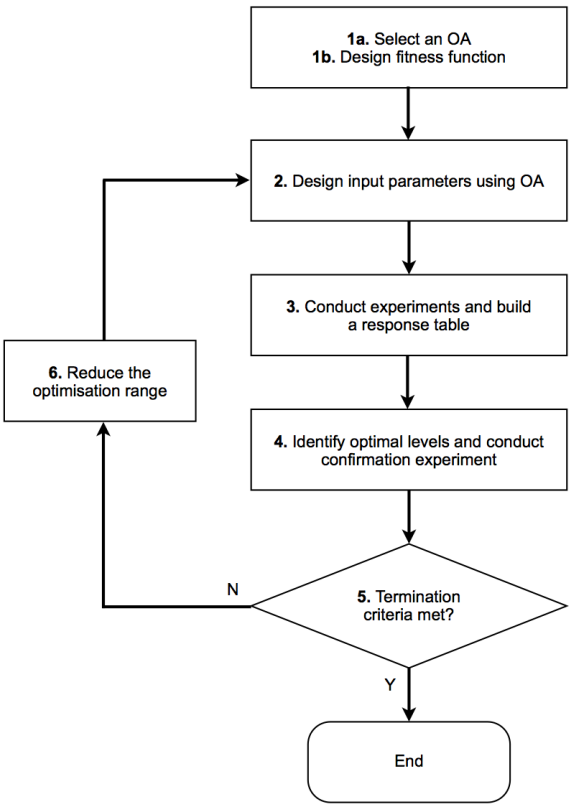


Fig. 15.1: Process associated with Taguchi's method.

In stage 1a, one of the 2 pre-built OAs will automatically be chosen depending on the number of parameters to optimise. Currently, up to 7 independent parameters can be optimised, although a method to construct OAs of any size is under testing.

In stage 1b, a fitness function is required to set a goal against which to compare results from the optimisation process. A number of pre-built fitness functions can be found in the `fitness_functions.py` module, e.g.

`minvalue`, `maxvalue` and `xcorr`. Users can also easily add their own fitness functions to this module. All fitness functions must take two arguments:

- `filename` a string containing the full path and filename of the output file
- `args` a dictionary which can contain any number of additional arguments for the function, e.g. names of outputs (rxs) in the model

Additionally, all fitness functions must return a single fitness value which the optimsation process will aim to maximise.

Stages 2-6 are iterated through by the optimisation process.

Parameters and settings for the optimisation process are specified within a special Python block defined by `#taguchi` and `#end_taguchi` in the input file. The parameters to optimise must be defined in a dictionary named `optparams` and their initial ranges specified as lists with lower and upper values. The fitness function, it's parameters, and a stopping value are defined in dictionary named `fitness` which has keys for:

- `name` a string that is the name of the fitness function to be used
- `args` a dictionary containing arguments to be passed to the fitness function. Within `args` there must be a key called `outputs` which contains a string or list of the names of one or more outputs (rxs) in the model
- `stop` a value from the fitness function which when exceeded the optimisation should stop

Optionally a variable called `maxiterations` maybe specified which will set a maximum number of iterations after which the optimisation process will terminate irrespective of any other criteria. If it is not specified it defaults to a maximum of 20 iterations.

There is also a builtin criterion to terminate the optimisation process is successive fitness values are within 0.1% of one another.

15.3 How to use the package

The package requires `#python` and `#end_python` to be used in the input file, as well as `#taguchi` and `#end_taguchi` for specifying parameters and setting for the optimisation process. A Taguchi optimisation is run using the command line option `--opt-taguchi`.

15.3.1 Example

The following example demonstrates using the Taguchi optimisation process to optimise values of loading resistors used in a bowtie antenna. The example is slighty contrived as the goal is simply to find values for the resistors that produces a maximum absolute amplitude in the response from the antenna. We already know this should occur when the values of the resistors are at a minimum. Nevertheless, it is useful to illustrate the optimisation process and how to use it.

The bowtie design features three vertical slots (y-direction) in each arm of the bowtie. Each slot has different loading resistors, but within each slot there are four resistors of the same value. A resistor is modelled as two parallel edges of a cell. The bowtie is placed on a lossless substrate of relative permittivity 4.8. The antenna is modelled in free space, and an output point of the electric field (named `Ex60mm`) is specified at a distance of 60mm from the feed of the bowtie (red coloured cell).

```
#taguchi:
optparams['resinner'] = [0.1, 1000]
optparams['resmiddle'] = [0.1, 1000]
optparams['resouter'] = [0.1, 1000]
fitness = {'name': 'min_max_value', 'stop': 10, 'args': {'type': 'absmax', 'outputs
↪': 'Ex60mm'}}
#end_taguchi:

#python:
```

(continues on next page)

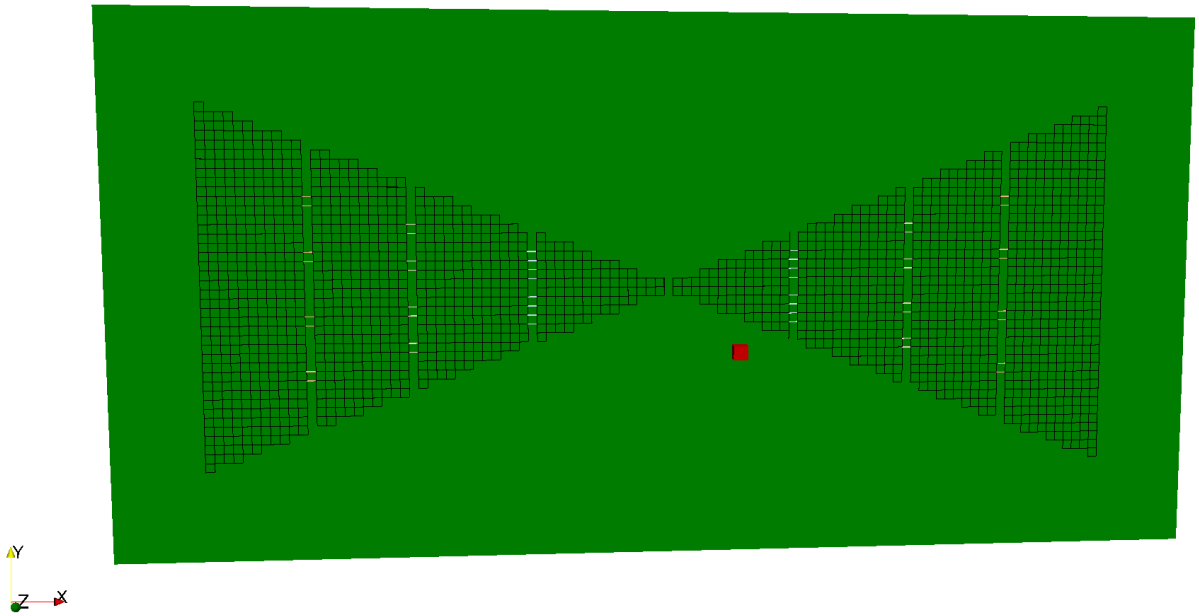


Fig. 15.2: FDTD geometry mesh showing bowtie antenna with slots and loading resistors.

(continued from previous page)

```
9
10  import numpy as np
11  from gprMax.input_cmd_funcs import *
12
13  title = 'antenna_bowtie_opt'
14  print('#title: {}'.format(title))
15
16  domain = domain(0.180, 0.120, 0.160)
17  dxdydz = dx_dy_dz(0.001, 0.001, 0.001)
18  timewindow = time_window(5e-9)
19  fr4_dims = (0.120, 0.060, 0.002)
20  bowtie_dims = (0.050, 0.040) # Length, height
21  flare_angle = np.arctan((bowtie_dims[1]/2) / bowtie_dims[0])
22  tx_pos = (domain[0]/2, domain[1]/2, 0.050)
23
24  # Vertical slot positions, relative to feed position, i.e. txpos[0]
25  vcut_pos = (0.014, 0.027, 0.038)
26
27  # Loading resistor values
28  res = np.array([optparams['resinner'], optparams['resmiddle'], optparams['resouter
    ↪']])
29  rescond = ((1 / res) * (dxdydz[1] / (dxdydz[0] * dxdydz[2]))) / 2 # Divide by␣
    ↪number of parallel edges per resistor
30
31  # Materials
32  material(4.8, 0, 1, 0, 'fr4')
33  for i in range(len(res)):
34      material(1, rescond[i], 1, 0, 'res' + str(i + 1))
35
36  # Source excitation and type
37  print('#waveform: gaussian 1 2e9 mypulse')
38  print('#transmission_line: x {:g} {:g} {:g} 50 mypulse'.format(tx_pos[0], tx_
    ↪pos[1], tx_pos[2]))
39
40  # Output point - distance from tx_pos in z direction
41  print('#rx: {:g} {:g} {:g} Ex60mm Ex'.format(tx_pos[0], tx_pos[1], tx_pos[2] + 0.
    ↪060))
```

(continues on next page)

(continued from previous page)

```
42
43  # Bowtie - upper x half
44  triangle(tx_pos[0], tx_pos[1], tx_pos[2], tx_pos[0] + bowtie_dims[0], tx_pos[1] -␣
    ↪bowtie_dims[1]/2, tx_pos[2], tx_pos[0] + bowtie_dims[0], tx_pos[1] + bowtie_
    ↪dims[1]/2, tx_pos[2], 0, 'pec')
45
46  # Bowtie - upper x half - vertical cuts
47  for i in range(len(vcut_pos)):
48          for j in range(int(bowtie_dims[1] / dxdydz[2])):
49                  edge(tx_pos[0] + vcut_pos[i], tx_pos[1] - bowtie_dims[1]/2 + j *␣
    ↪dxdydz[1], tx_pos[2], tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] - bowtie_
    ↪dims[1]/2 + j * dxdydz[1], tx_pos[2], 'free_space')
50
51  # Bowtie - upper x half - vertical cuts - loading
52  for i in range(len(vcut_pos)):
53      gap = ((vcut_pos[i] * np.tan(flare_angle) * 2) - 4*dxdydz[1]) / 5
54      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] - (1.5 * gap) - dxdydz[1], tx_pos[2],␣
    ↪tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] - (1.5 * gap) - dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
55      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] - (1.5 * gap) - 2*dxdydz[1], tx_pos[2],
    ↪ tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] - (1.5 * gap) - 2*dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
56      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] - (0.5 * gap), tx_pos[2], tx_pos[0] +␣
    ↪vcut_pos[i] + dxdydz[0], tx_pos[1] - (0.5 * gap), tx_pos[2], 'res' + str(i + 1))
57      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] - (0.5 * gap) - dxdydz[1], tx_pos[2],␣
    ↪tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] - (0.5 * gap) - dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
58      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] + (0.5 * gap), tx_pos[2], tx_pos[0] +␣
    ↪vcut_pos[i] + dxdydz[0], tx_pos[1] + (0.5 * gap), tx_pos[2], 'res' + str(i + 1))
59      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] + (0.5 * gap) + dxdydz[1], tx_pos[2],␣
    ↪tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] + (0.5 * gap) + dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
60      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] + (1.5 * gap) + dxdydz[1], tx_pos[2],␣
    ↪tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] + (1.5 * gap) + dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
61      edge(tx_pos[0] + vcut_pos[i], tx_pos[1] + (1.5 * gap) + 2*dxdydz[1], tx_pos[2],
    ↪ tx_pos[0] + vcut_pos[i] + dxdydz[0], tx_pos[1] + (1.5 * gap) + 2*dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
62
63  # Bowtie - lower x half
64  triangle(tx_pos[0] + dxdydz[0], tx_pos[1], tx_pos[2], tx_pos[0] - bowtie_dims[0],␣
    ↪tx_pos[1] - bowtie_dims[1]/2, tx_pos[2], tx_pos[0] - bowtie_dims[0], tx_pos[1] +␣
    ↪bowtie_dims[1]/2, tx_pos[2], 0, 'pec')
65
66  # Bowtie - lower x half - cuts for loading
67  for i in range(len(vcut_pos)):
68          for j in range(int(bowtie_dims[1] / dxdydz[2])):
69                  edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] - bowtie_
    ↪dims[1]/2  + j * dxdydz[1], tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] -␣
    ↪bowtie_dims[1]/2  + j * dxdydz[1], tx_pos[2], 'free_space')
70
71  # Bowtie - lower x half - vertical cuts - loading
72  for i in range(len(vcut_pos)):
73      gap = ((vcut_pos[i] * np.tan(flare_angle) * 2) - 4*dxdydz[1]) / 5
74      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] - (1.5 * gap) - dxdydz[1],␣
    ↪tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] - (1.5 * gap) - dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
75      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] - (1.5 * gap) -␣
    ↪2*dxdydz[1], tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] - (1.5 * gap) -␣
    ↪2*dxdydz[1], tx_pos[2], 'res' + str(i + 1))
76      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] - (0.5 * gap), tx_pos[2],␣
    ↪tx_pos[0] - vcut_pos[i], tx_pos[1] - (0.5 * gap), tx_pos[2], 'res' + str(i + 1))
```

(continues on next page)

(continued from previous page)

```
77      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] - (0.5 * gap) - dxdydz[1],␣
    ↪tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] - (0.5 * gap) - dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
78      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] + (0.5 * gap), tx_pos[2],␣
    ↪tx_pos[0] - vcut_pos[i], tx_pos[1] + (0.5 * gap), tx_pos[2], 'res' + str(i + 1))
79      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] + (0.5 * gap) + dxdydz[1],␣
    ↪tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] + (0.5 * gap) + dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
80      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] + (1.5 * gap) + dxdydz[1],␣
    ↪tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] + (1.5 * gap) + dxdydz[1], tx_
    ↪pos[2], 'res' + str(i + 1))
81      edge(tx_pos[0] - vcut_pos[i] - dxdydz[0], tx_pos[1] + (1.5 * gap) +␣
    ↪2*dxdydz[1], tx_pos[2], tx_pos[0] - vcut_pos[i], tx_pos[1] + (1.5 * gap) +␣
    ↪2*dxdydz[1], tx_pos[2], 'res' + str(i + 1))
82
83  # PCB
84  box(tx_pos[0] - fr4_dims[0]/2, tx_pos[1] - fr4_dims[1]/2, tx_pos[2] - fr4_dims[2],␣
    ↪tx_pos[0] + fr4_dims[0]/2, tx_pos[1] + fr4_dims[1]/2, tx_pos[2], 'fr4')
85
86  # Detailed geometry view of PCB and bowtie
87  #geometry_view(tx_pos[0] - fr4_dims[0]/2, tx_pos[1] - fr4_dims[1]/2, tx_pos[2] -␣
    ↪fr4_dims[2], tx_pos[0] + fr4_dims[0]/2, tx_pos[1] + fr4_dims[1]/2, tx_pos[2] +␣
    ↪dxdydz[2], dxdydz[0], dxdydz[1], dxdydz[2], title + '_tx', type='f')
88
89  # Geometry view of entire domain
90  #geometry_view(0, 0, 0, domain[0], domain[1], domain[2], dxdydz[0], dxdydz[1],␣
    ↪dxdydz[2], title)
91
92  #end_python:
```

The first part of the input file (lines 1-6) contains the parameters to optimise, their initial ranges, and fitness function information for the optimisation process. Three parameters representing the resistor values are defined with ranges between 0.1 Ω and 1 $k\Omega$. A pre-built fitness function called `min_max_value` is specified with a stopping criterion of 10V/m. Arguments for the `min_max_value` function are `type` given as `absmax`, i.e. the maximum absolute values, and the output point in the model that will be used in the optimisation is specified as having the name `Ex60mm`.

The next part of the input file (lines 8-92) contains the model. For the most part there is nothing special about the way the model is defined - a mixture of Python, NumPy and functional forms of the input commands (available by importing the module `input_cmd_funcs`) are used. However, it is worth pointing out how the values of the parameters to optimise are accessed. On line 29 a NumPy array of the values of the resistors is created. The values are accessed using their names as keys to the `optparams` dictionary. On line 30 the values of the resistors are converted to conductivities, which are used to create new materials (line 34-35). The resistors are then built by applying the materials to cell edges (e.g. lines 55-62). The output point in the model in specifed with the name `Ex60mm` and as having only an `Ex` field output (line 42).

The optimisation process is run on the model using the `--opt-taguchi` command line flag.

```
python -m gprMax user_libs/optimisation_taguchi/antenna_bowtie_opt.in --opt-taguchi
```

Results

When the optimisation has completed a summary will be printed showing histories of the parameter values and the fitness metric. These values are also saved (pickled) to file and can be plotted using the `plot_results.py` module, for example:

```
python -m user_libs.optimisation_taguchi.plot_results user_libs/optimisation_
    ↪taguchi/antenna_bowtie_opt_hist.pickle
```

```
Optimisations summary for: antenna_bowtie_opt_hist.pickle
Number of iterations: 4
History of fitness values: [4.2720928, 5.68856, 5.7023263, 5.7023263]
History of parameter values:
resinner [250.07498, 0.87031555, 0.1, 0.1]
resmiddle [250.07498, 0.87031555, 0.1, 0.1]
resouter [250.07498, 0.87031555, 0.1, 0.1]
```

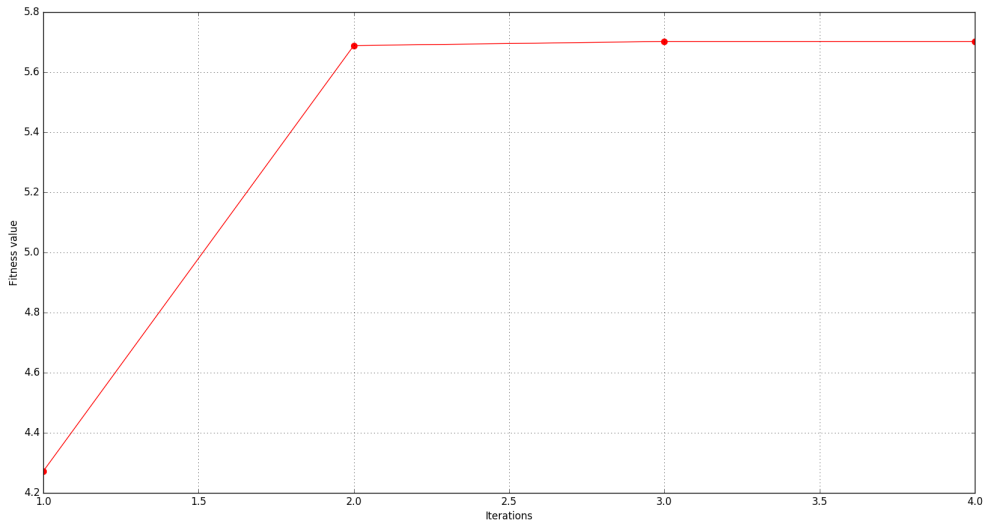


Fig. 15.3: History of values of fitness metric (absolute maximum).

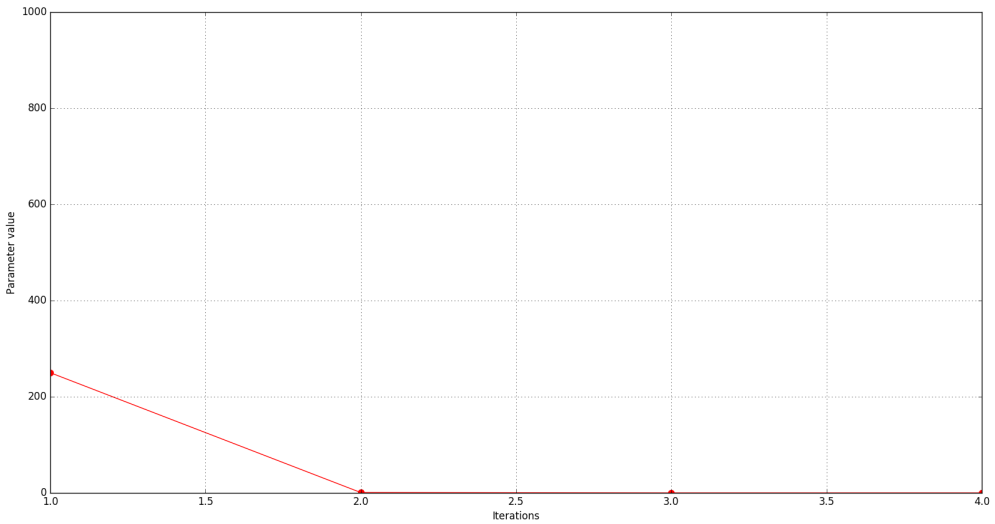


Fig. 15.4: History of values of parameters - `resinner`, `resmiddle`, and `resouter` (in this case they are all identical).

The optimisation process terminated after 4 iterations because succcessive fitness values were within 0.1% of one another. A maximum absolute amplitude value of 5.7 V/m was achieved when the three resistors had values of 0.1 Ω.

CHAPTER 16

Introductory (2D) models

This section provides some introductory example models in 2D that demonstrate basic features of gprMax. Each example comes with an input file which you can download and run.

16.1 A-scan from a metal cylinder

`cylinder_Ascan_2D.in`

This example is the GPR modelling equivalent of 'Hello World'! It demonstrates how to simulate a single trace (A-scan) from a metal cylinder buried in a dielectric half-space.

```
#title: A-scan from a metal cylinder buried in a dielectric half-space
#domain: 0.240 0.210 0.002
#dx_dy_dz: 0.002 0.002 0.002
#time_window: 3e-9

#material: 6 0 1 0 half_space

#waveform: ricker 1 1.5e9 my_ricker
#hertzian_dipole: z 0.100 0.170 0 my_ricker
#rx: 0.140 0.170 0

#box: 0 0 0 0.240 0.170 0.002 half_space
#cylinder: 0.120 0.080 0 0.120 0.080 0.002 0.010 pec

#geometry_view: 0 0 0 0.240 0.210 0.002 0.002 0.002 0.002 cylinder_half_space n
```

The geometry of the scenario is straightforward and an image created from the geometry view is shown in Fig. 16.1. The transparent area around the boundary of the domain represents the *PML region*. The red cell is the source and the blue cell is the receiver.

For this initial example a detailed description of what each command in the input file does and why each command was used is given. The following steps explain the process of building the input file:

16.1.1 Determine the constitutive parameters for the materials

There will be three different materials in the model representing air, the dielectric half-space, and the metal cylinder. Air (free space) already exists as a built-in material in gprMax which can be accessed using the `free_space`

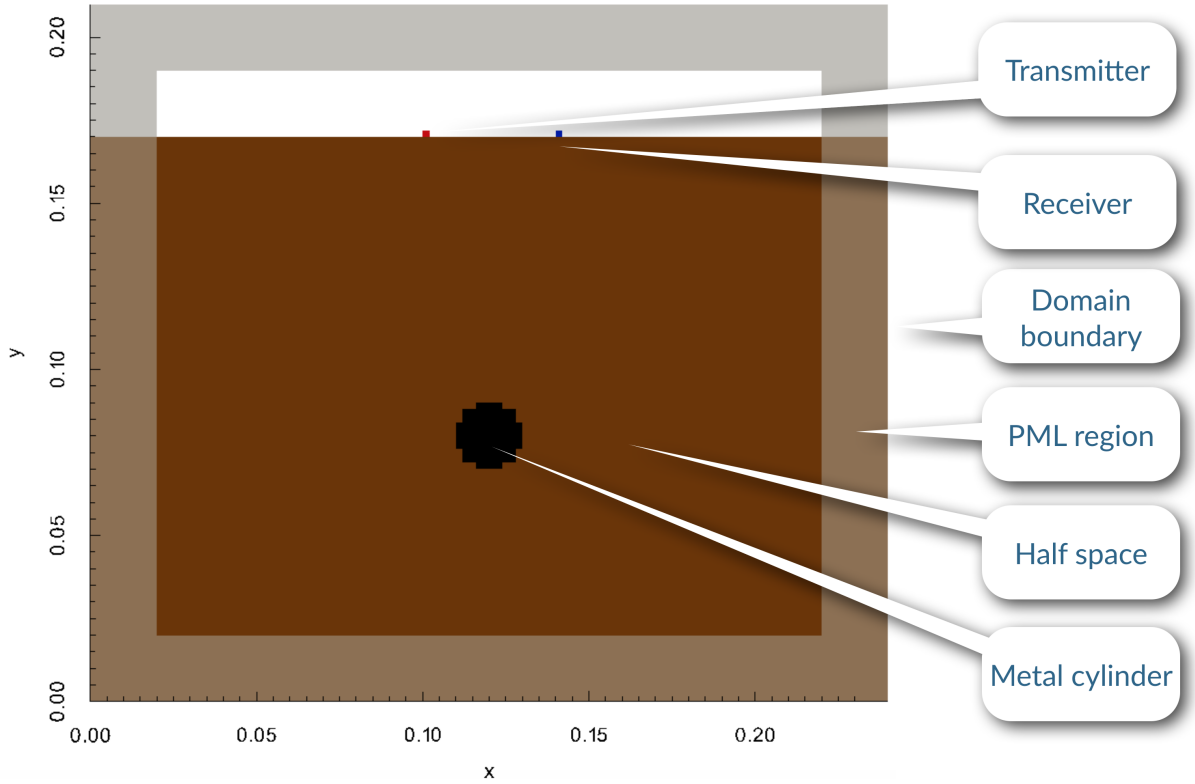


Fig. 16.1: Geometry of a 2D model of a metal cylinder buried in a dielectric half-space.

identifier. The metal cylinder will be modelled as a Perfect Electric Conductor, which again exists as a built-in material in gprMax and can be accessed using the `pec` identifier. So the only material which has to be defined is for the dielectric half-space. It is a non-magnetic material, i.e. $\mu_r = 1$ and $\sigma_* = 0$ and with a relative permittivity of six, $\epsilon_r = 6$, and zero conductivity, $\sigma = 0$. The identifier `half_space` will be used.

```
#material: 6 0 1 0 half_space
```

16.1.2 Determine the source type and excitation frequency

These should generally be known, often based on the GPR system or scenario being modelled. Low frequencies are used where significant penetration depth is important, whereas high frequencies are used where less penetration and better resolution are required. In this case a theoretical Hertzian dipole source fed with a Ricker waveform with a centre frequency of $f_c = 1.5$ GHz will be used to simulate the GPR antenna (see the *bowtie antenna example model* for how to include a model of the actual GPR antenna in the simulation).

```
#waveform: ricker 1 1.5e9 my_ricker
#hertzian_dipole: z 0.100 0.170 0 my_ricker
```

The Ricker waveform is created with the `#waveform` command, specifying an amplitude of one, centre frequency of 1.5 GHz and picking an arbitrary identifier of `my_ricker`. The Hertzian dipole source is created using the `#hertzian_dipole` command, specifying a z direction polarisation (perpendicular to the survey direction if a B-scan were being created), location on the surface of the slab, and using the Ricker waveform already created.

16.1.3 Calculate a spatial resolution and domain size

In the *Guidance on GPR modelling* section it was stated that a good *rule-of-thumb* was that the spatial resolution should be one tenth of the smallest wavelength present in the model. To determine the smallest wavelength, the highest frequency and lowest velocity present in the model are required. The highest frequency is not the centre frequency of the Ricker waveform! *By examining the spectrum of a Ricker waveform* it is evident much higher frequencies are present, i.e. at a level -40dB from the centre frequency, frequencies 2-3 times as high are present. In this case the highest significant frequency present in the model is likely to be around 4 GHz. The wavelength at 4 GHz in the half-space (which has the lowest velocity) would be:

$$\lambda = \frac{c}{f\sqrt{\epsilon_r}} = \frac{299792458}{4 \times 10^9\sqrt{6}} \approx 31 \text{ mm}$$

This would give a minimum spatial resolution of 3 mm. However, the diameter of the cylinder is 20 mm so would be resolved to 7 cells. Therefore a better choice would be 2 mm which resolves the diameter of the rebar to 10 cells.

```
#dx_dy_dz: 0.002 0.002 0.002
```

The domain size should be enough to enclose the volume of interest, plus allow 10 cells (if using the default value) for the PML absorbing boundary conditions and approximately another 10 cells of between the PML and any objects of interest. In this case the plan is to take a B-scan of the scenario (in the next example) so the domain should be large enough to do that. Although this is a 2D model one cell must be specified in the infinite direction (in this case the z direction) of the domain.

```
#domain: 0.240 0.210 0.002
```

16.1.4 Choose a time window

It is desired to see the reflection from the cylinder, therefore the time window must be long enough to allow the electromagnetic waves to propagate from the source through the half-space to the cylinder and be reflected back to the receiver, i.e.

$$t = \frac{0.180}{\frac{c}{\sqrt{6}}} \approx 1.5 \text{ ns}$$

This is the minimum time required, but the source waveform has a width of 1.2 ns, to allow for the entire source waveform to be reflected back to the receiver an initial time window of 3 ns will be tested.

```
#time_window: 3e-9
```

The time step required for the model is automatically calculated using the *CFL condition in 2D*.

16.1.5 Create the objects

Now physical objects can be created for the half-space and the cylinder. First the `#box` command will be used to create the half-space and then the `#cylinder` command will be given which will overwrite the properties of the half-space with those of the cylinder at the location of the cylinder.

```
#box: 0 0 0 0.240 0.170 0.002 half_space
#cylinder: 0.120 0.080 0 0.120 0.080 0.002 0.010 pec
```

16.1.6 Run the model

You can now run the model:

```
python -m gprMax user_models/cylinder_Ascan_2D.in
```

Tip:

- You can use the `--geometry-only` command line argument to build a model and produce any geometry views but not run the simulation. This option is useful for checking the geometry of the model is correct.

16.1.7 View the results

You should have produced an output file `cylinder_Ascan_2D.out`. You can view the results (see *Output data* section) using the command:

```
python -m tools.plot_Ascan user_models/cylinder_Ascan_2D.out
```

Fig. 16.2 shows the time history of the electric and magnetic field components and currents at the receiver location. The E_z field component can be converted to voltage which represents the A-scan (trace). The initial part of the signal (~0.5-1.5 ns) represents the direct wave from transmitter to receiver. Then comes the reflected wavelet (~1.8-2.6 ns), which has opposite polarity, from the metal cylinder.

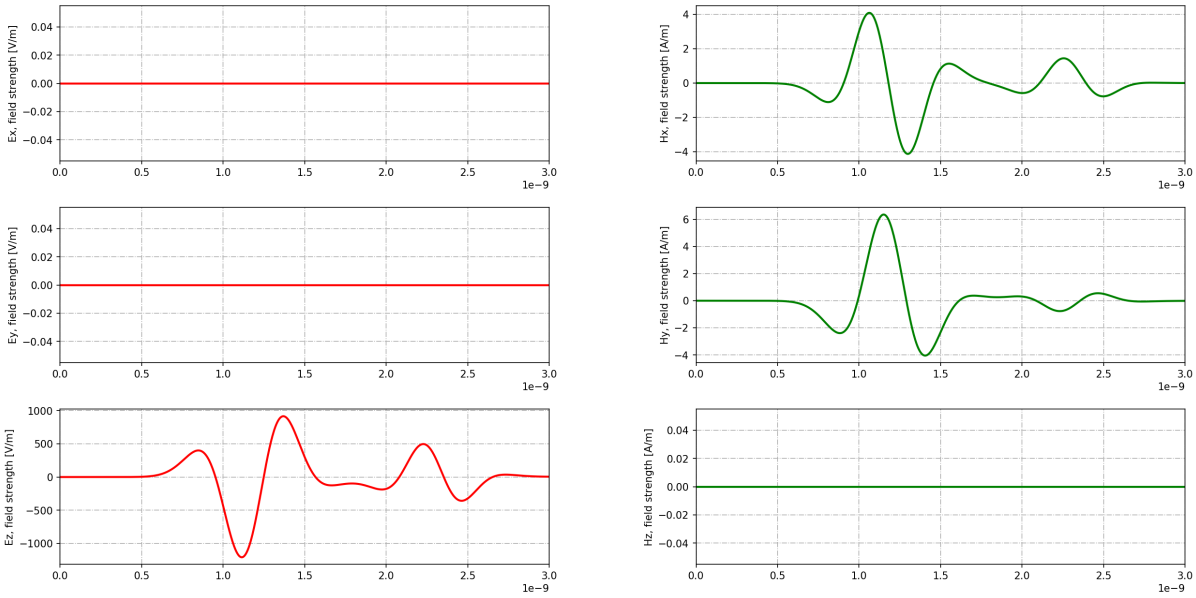


Fig. 16.2: Electric and magnetic field component time histories from the receiver in the model of a metal cylinder buried in a dielectric half-space.

Check out a video of the field propagation in this example (https://youtu.be/BpBo0-SFda4). More videos are screencasts can be found in the *screencasts section*.

16.2 B-scan from a metal cylinder

`cylinder_Bscan_2D.in`

This example uses the same geometry as the previous example but this time a B-scan is created. A B-scan is composed of multiple traces (A-scans) recorded as the source and receiver are moved over the target, in this case the metal cylinder.

```
1  #title: B-scan from a metal cylinder buried in a dielectric half-space
2  #domain: 0.240 0.210 0.002
3  #dx_dy_dz: 0.002 0.002 0.002
4  #time_window: 3e-9
5
6  #material: 6 0 1 0 half_space
7
8  #waveform: ricker 1 1.5e9 my_ricker
9  #hertzian_dipole: z 0.040 0.170 0 my_ricker
10 #rx: 0.080 0.170 0
11 #src_steps: 0.002 0 0
12 #rx_steps: 0.002 0 0
13
14 #box: 0 0 0 0.240 0.170 0.002 half_space
15 #cylinder: 0.120 0.080 0 0.120 0.080 0.002 0.010 pec
```

The differences between this input file and the one from the A-scan are the x coordinates of the source and receiver (lines 11 and 12), and the commands needed to move the source and receiver (lines 13 and 14). As before, the source and receiver are offset by 40mm from each other as before but they are now shifted to a starting position for the scan. The `#src_steps` command is used to move every source in the model by specified steps each time the model is run. Similarly, the `#rx_steps` command is used to move every receiver in the model by specified steps each time the model is run. Note, the same functionality can be achieved by using a block of Python code in the input file to move the source and receiver individually (for further details see the *Python section*).

To run the model to create a B-scan you must pass an optional argument to specify the number of times the model should be run. In this case this is the number of A-scans (traces) that will comprise the B-scan. For a B-scan over a distance of 120mm with a step of 2mm that is 60 A-scans.

```
python -m gprMax user_models/cylinder_Bscan_2D.in -n 60
```

16.2.1 Results

You should have produced 60 output files, one for each A-scan, with names `cylinder_Bscan_2D1.out`, `cylinder_Bscan_2D2.out` etc... These can be combined into a single file using the command:

```
python -m tools.outputfiles_merge user_models/cylinder_Bscan_2D
```

You should see a combined output file `cylinder_Bscan_2D_merged.out`. You can add the optional argument `--remove-files` if you want to automatically delete the original single A-scan output files.

You can now view an image of the B-scan using the command:

```
python -m tools.plot_Bscan user_models/cylinder_Bscan_2D_merged.out Ez
```

Fig. 16.3 shows the B-scan (of the E_z field component). Again, the initial part of the signal (~0.5-1.5 ns) represents the direct wave from transmitter to receiver. Then comes the refelected wave (~2-3 ns) from the metal cylinder which creates the hyperbolic shape.

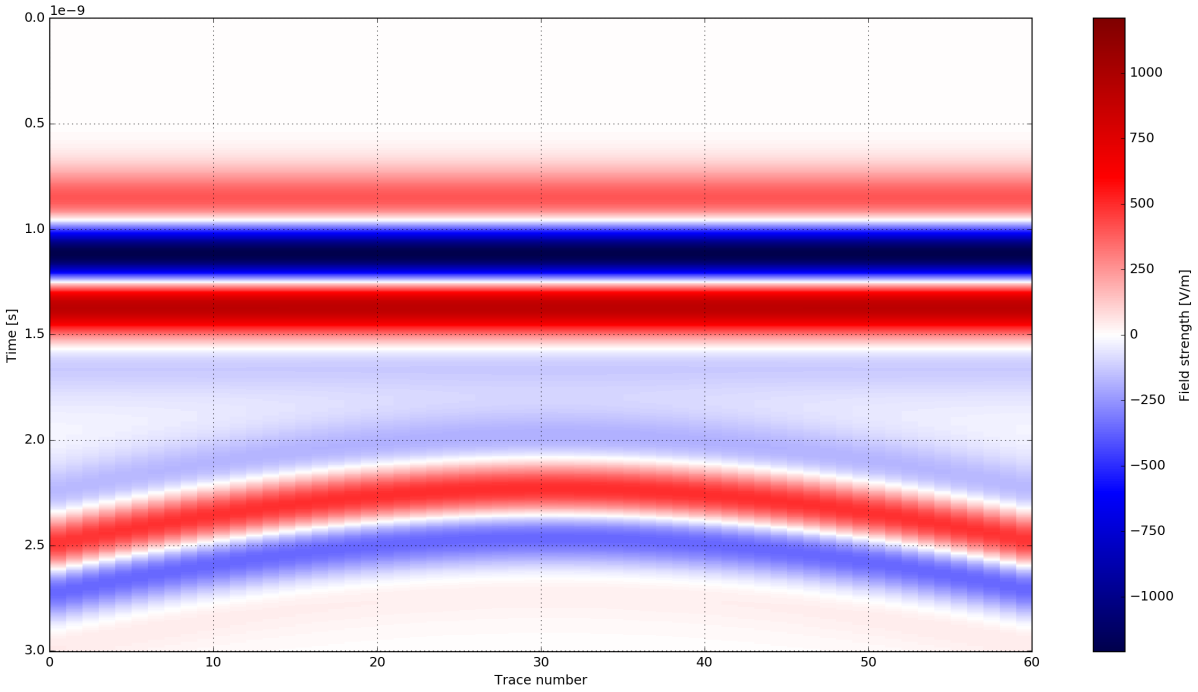


Fig. 16.3: B-scan of model of a metal cylinder buried in a dielectric half-space.

CHAPTER 17

Antenna models

This section provides some example models of antennas. Each example comes with an input file which you can download and run.

17.1 Wire dipole antenna model

`antenna_wire_dipole_fs.in`

This example demonstrates a model of a half-wavelength wire dipole antenna in free space. It is a balanced antenna and it's characteristics are well known from theory *[BAL2005]*. The length of the dipole is 150mm with a 1mm gap between the arms.

```
#title: Wire antenna - half-wavelength dipole in free-space
#domain: 0.050 0.050 0.200
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 60e-9

#waveform: gaussian 1 1e9 mypulse
#transmission_line: z 0.025 0.025 0.100 73 mypulse

## 150mm length
#edge: 0.025 0.025 0.025 0.025 0.025 0.175 pec

## 1mm gap at centre of dipole
#edge: 0.025 0.025 0.100 0.025 0.025 0.101 free_space

#geometry_view: 0.020 0.020 0.020 0.030 0.030 0.180 0.001 0.001 0.001 antenna_wire_
↪dipole_fs f
```

The wire is modelled using the `#edge` command which specifies properties of the edge of the Yee cell. The antenna is fed using the `#transmission_line` command. The one-dimensional transmission line model virtually attaches to the dipole at the gap between the arms. The antenna has an input resistance $Z_{in} = 73\ \Omega$ specified in the `#transmission_line` command, and uses a Gaussian waveform with a centre frequency of 1GHz. A time window of 60ns is used: firstly, to give enough time for the response to decay down to zero; and secondly, to allow a reasonable resolution (17MHz) for calculating antenna parameters that involve taking a FFT ($\Delta f = 1/T$ where Δf is the frequency bin spacing and T is the time window).

Time histories of voltage and current values in the transmission line are saved to the output file. These are documented in the *output file section*. These parameters are useful for calculating characteristics of the antenna such as the input impedance or S-parameters. gprMax includes a Python module (in the `tools` package) to help you view the input impedance and admittance and s11 parameter from an antenna model fed using a transmission line. Details of how to use this module is given in the *tools section*.

17.1.1 Results

You can view the results (see *Output data* and *tools* sections) using the command:

```
python -m tools.plot_antenna_params user_models/antenna_wire_dipole_fs.out
```

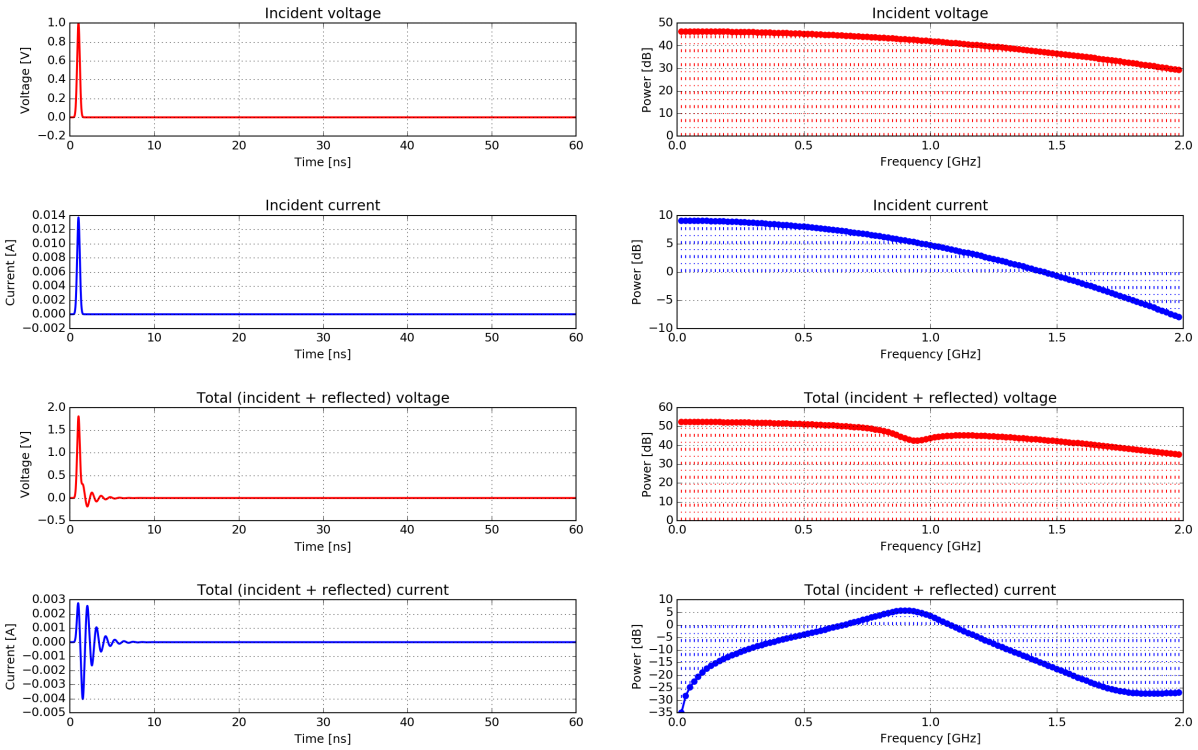


Fig. 17.1: Time and frequency domain plots of the incident and total (incident + reflected) voltages and currents in the transmission line ($\Delta f = 17\ MHz$).

Fig. 17.1 shows time histories and frequency spectra of the incident and total (incident + reflected) voltages and currents in the transmission line. Fig. 17.2 shows the input admittance and impedance (resistance and reactance), and s11 parameter of the half-wavelength wire dipole. Fig. 17.3 shows a more detailed view of these parameters. The s11 parameter shows that the first resonance of the antenna is at 950MHz. Depending on the radius of the wire, the length of the dipole for first resonance is about $l = 0.47\lambda$ to 0.48λ. The thinner the wire the closer the resonance is to 0.48λ *[BAL2005]*. In this case, with a first resonance of 950MHz and a length of 150mm, $l/\lambda = 0.475$. The input impedance is $Z_{in} = 72.8 + j1\ \Omega$. If $l/\lambda = 0.5$ then the theoretical input impedance would be $Z_{in} = 73 + j42.5\ \Omega$. The reactive (imaginery) part associated with the input impedance of a dipole is a function of its length.

Fig. 17.4 demonstrates the increased frequency resolution ($\Delta f = 1.4\ MHz$) when an even longer time window (700ns) is used.

17.2 Bowtie antenna model

`antenna_like_MALA_1200_fs.in`

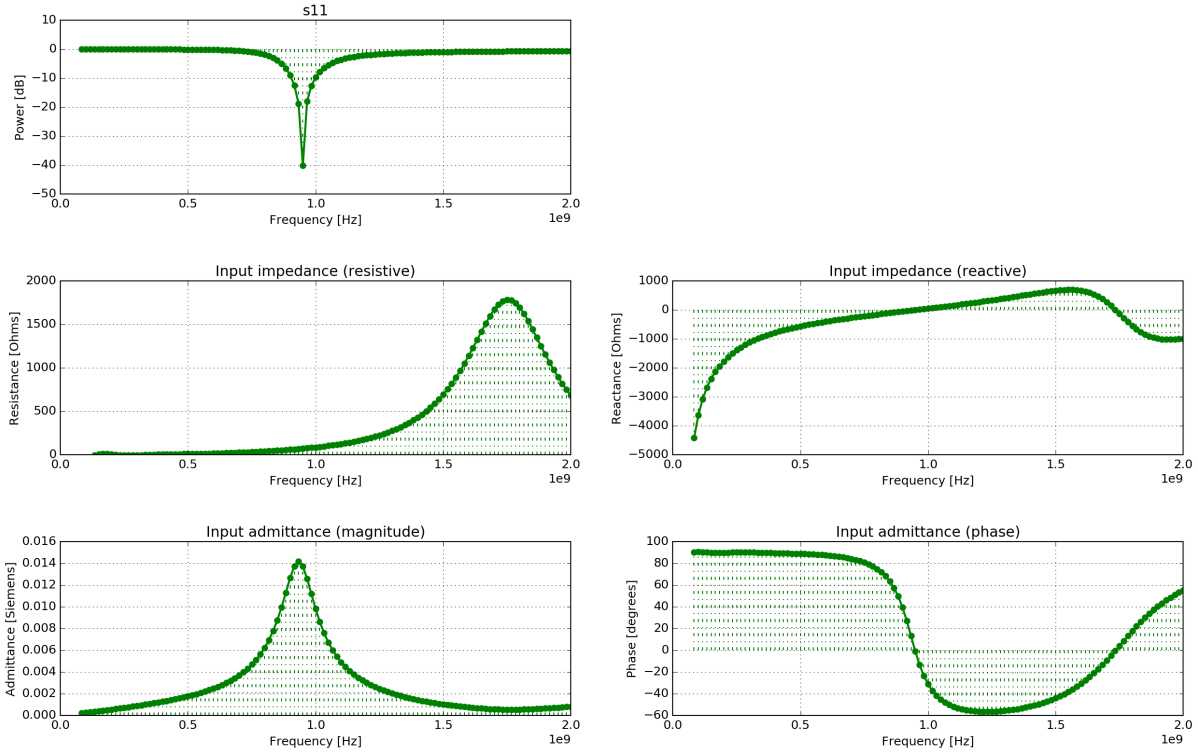


Fig. 17.2: Input admittance and impedance (resistance and reactance) and s11 parameter values of the antenna ($\Delta f = 17\ MHz$).

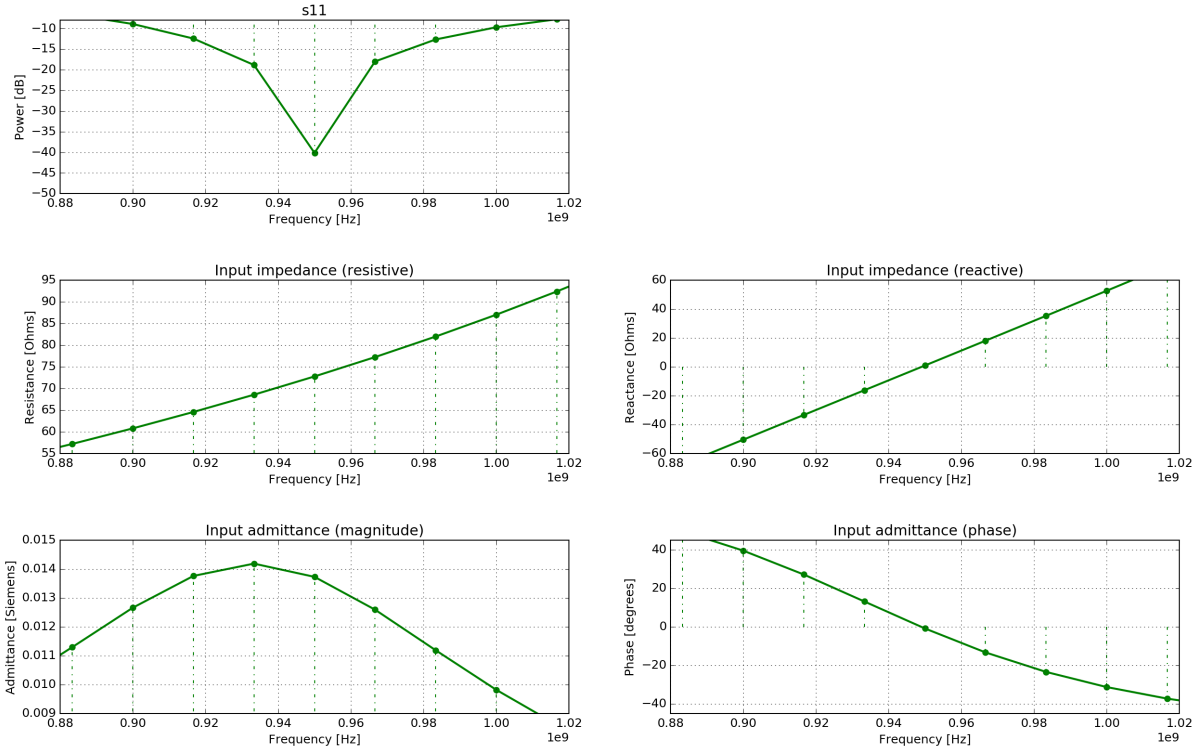


Fig. 17.3: Detailed view of input admittance and impedance (resistance and reactance) and s11 parameter values of the antenna ($\Delta f = 17\ MHz$).

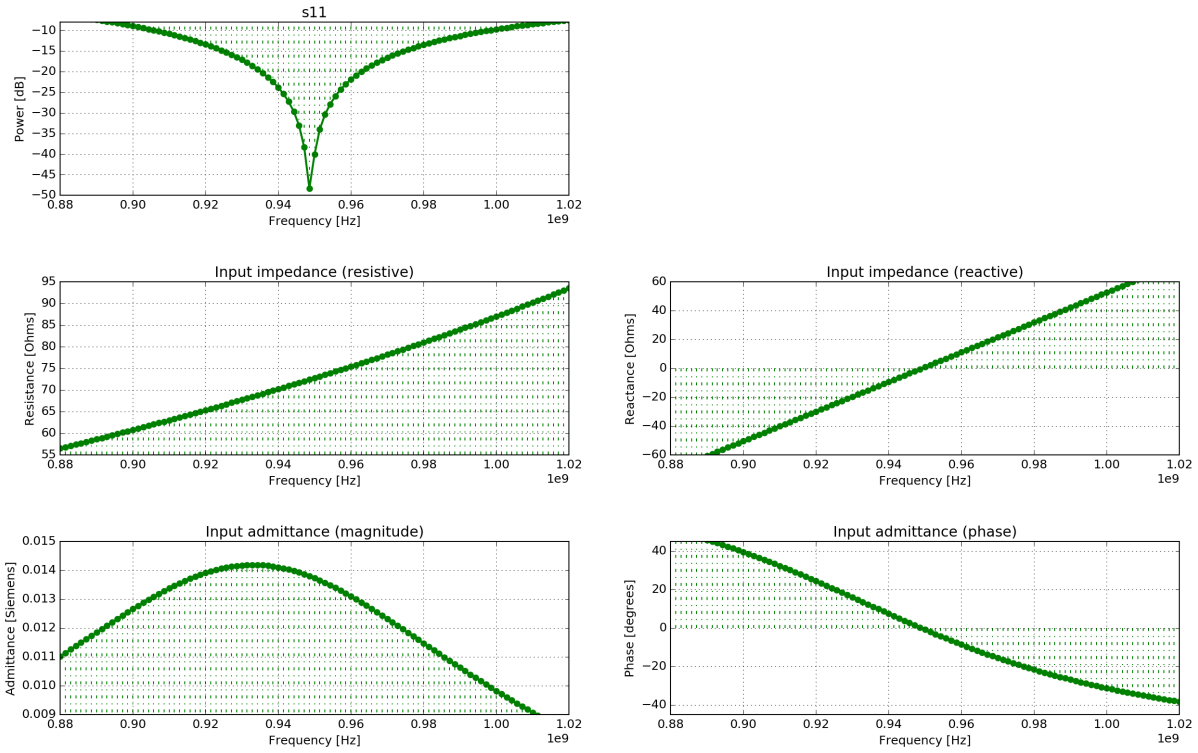


Fig. 17.4: Detailed view of input admittance and impedance (resistance and reactance) and s11 parameter values of the antenna ($\Delta f = 1.4\ MHz$)

This example demonstrates how to use one of the built-in antenna models in a simulation. Using a model of an antenna rather than a simple source, such as a Hertzian dipole, can improve the accuracy of the results of a simulation for many situations. It is especially important when the target is in the near-field of the antenna and there are complex interactions between the antenna and the environment. The simulation uses the model of an antenna similar to a MALA 1.2GHz antenna.

```
#title: MALA 1.2GHz 'like' antenna in free-space
#domain: 0.264 0.189 0.220
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 6e-9

#python:
from user_libs.antennas.MALA import antenna_like_MALA_1200
antenna_like_MALA_1200(0.132, 0.095, 0.100, 0.001)
#end_python:
```

The antenna model is loaded from a Python module and inserted into the input file just like another geometry command. The arguments for the `antenna_like_MALA_1200` function specify its (x, y, z) location as 0.132m, 0.095m, 0.100m using a 1mm spatial resolution. In this example the antenna is the only object in the model, i.e. the antenna is in free space. More information on using the built-in antenna models can be found in the *Python section*.

17.2.1 Results

When the simulation is run two geometry files for the antenna are produced along with an output file which contains a single receiver (the antenna output). You can view the results (see *Output data* and *tools* sections) using the command:

```
python -m tools.plot_Ascan user_models/antenna_like_MALA_1200_fs.out --outputs Ey
```

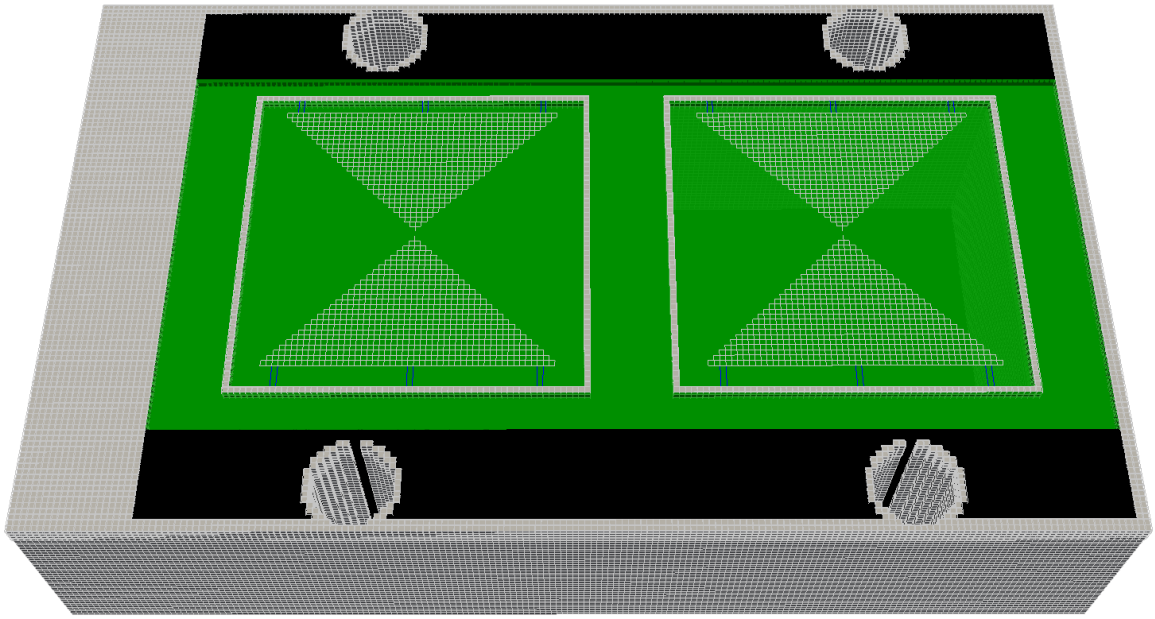


Fig. 17.5: FDTD geometry mesh showing an antenna model similar to a MALA 1.2GHz antenna (skid removed for illustrative purposes).

Fig. 17.6 shows the time history of the y-component of the electric field from the receiver bowtie of the antenna model (the antenna bowties are aligned with the y axis).

17.3 B-scan with a bowtie antenna model

`cylinder_Bscan_GSSI_1500.in`

This example demonstrates how to create a B-scan with an antenna model. The scenario is purposely simple to illustrate the method. A metal cylinder of diameter 20mm is buried in a dielectric half-space which has a relative permittivity of six. The simulation uses the model of an antenna similar to a GSSI 1.5GHz antenna.

```
#title: B-scan from a metal cylinder buried in a dielectric half-space with a GSSI 1.5GHz 'like' antenna
#domain: 0.480 0.148 0.235
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 6e-9

#material: 6 0 1 0 half_space

#box: 0 0 0 0.480 0.148 0.170 half_space
#cylinder: 0.240 0 0.080 0.240 0.148 0.080 0.010 pec

#python:
from user_libs.antennas.GSSI import antenna_like_GSSI_1500
antenna_like_GSSI_1500(0.105 + current_model_run * 0.005, 0.074, 0.170, 0.001)
#end_python:

geometry_view: 0 0 0 0.480 0.148 0.235 0.001 0.001 0.001 cylinder_GSSI_1500 n
```

The antenna must be moved to a new position for every single A-scan (trace) in the B-scan. In this example the B-scan distance will be 270mm with a trace every 5mm, so 54 model runs will be required.

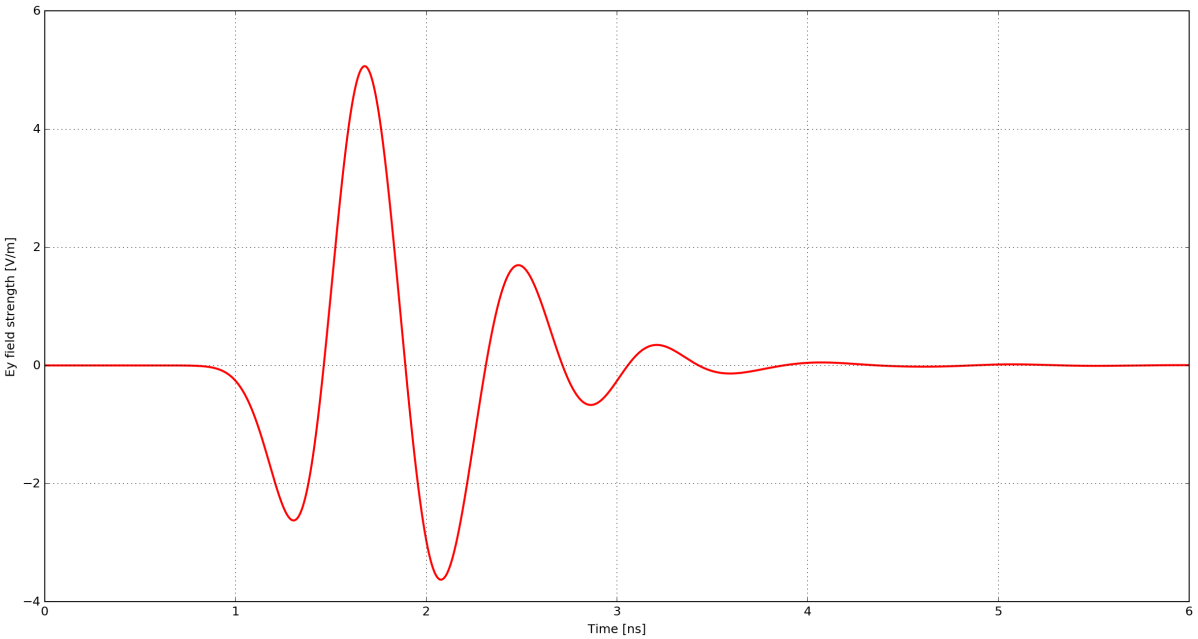


Fig. 17.6: Ey field output from the receiver bowtie of a model of an antenna similar to a MALA 1.2GHz antenna.

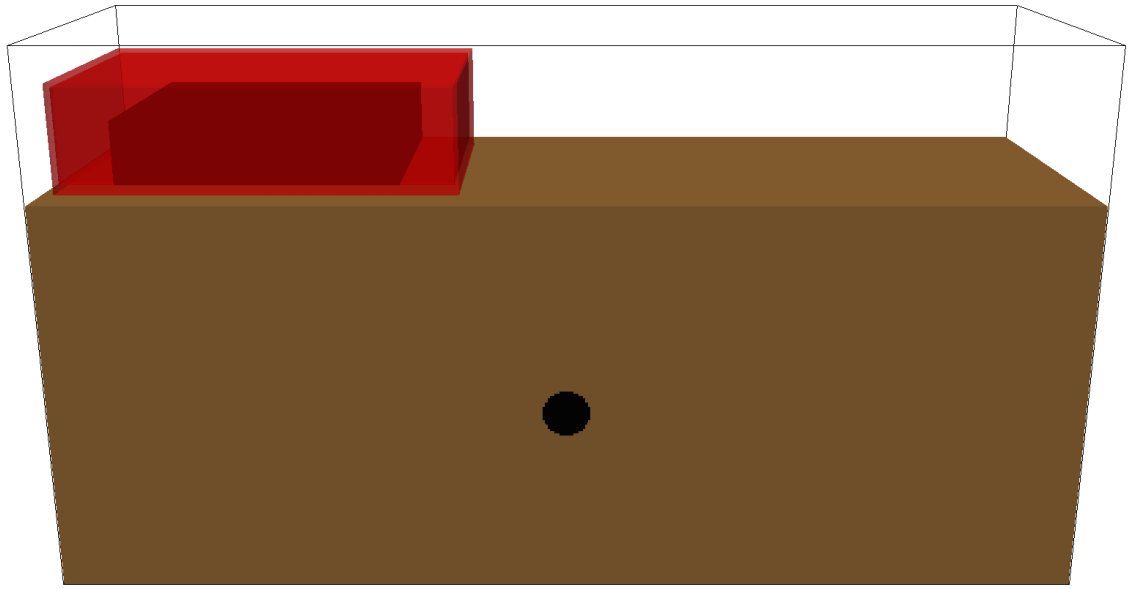


Fig. 17.7: FDTD geometry mesh showing a metal cylinder buried in a half-space and an antenna model similar to a GSSI 1.5GHz antenna.

```
python -m gprMax cylinder_Bscan_GSSI_1500.in -n 54
```

The total number of runs for a model as well as the number of the current run of the model are stored and can be accessed in Python as `number_model_runs` and `current_model_run`. The `current_model_run` can be used to move the position of the antenna for every run of the model as shown in Line 13. The antenna will be moved 5mm in the x direction for every new run of the model.

Note: If you are moving an antenna model within a simulation, e.g. to generate a B-scan, you should ensure that the step size you choose is a multiple of the spatial resolution of the simulation. Otherwise when the position of antenna is converted to cell coordinates the geometry maybe altered.

17.3.1 Results

After merging the A-scans into a single file you can now view an image of the B-scan using the command see *Output data* and *tools* sections:

```
python -m tools.plot_Bscan user_models/cylinder_Bscan_GSSI_1500_merged.out Ey
```

Fig. 17.8 shows the B-scan (of the Ey field component). The initial part of the signal (~1-2 ns) represents the direct wave from transmitter to receiver. Then comes a hyperbolic response from the metal cylinder.

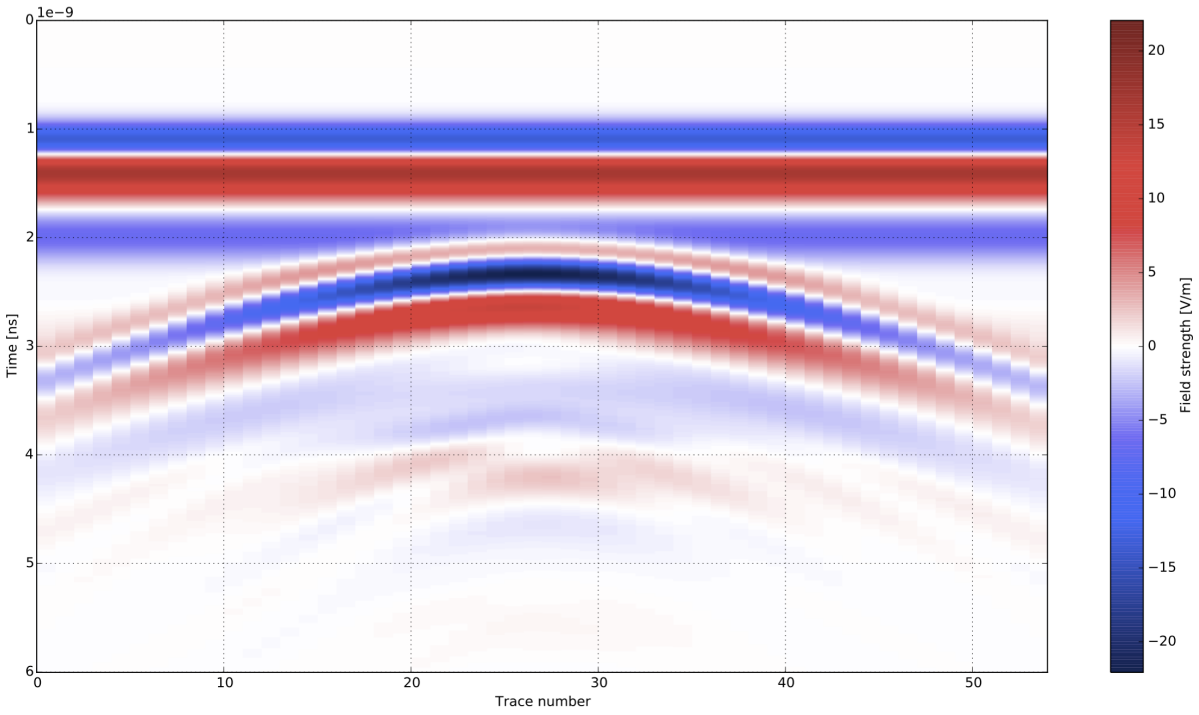


Fig. 17.8: B-scan of model of a metal cylinder buried in a dielectric half-space with a model of an antenna similar to a GSSI 1.5GHz antenna.

CHAPTER 18

Advanced features

This section provides example models of some of the more advanced features of gprMax. Each example comes with an input file which you can download and run.

18.1 Building a heterogeneous soil

`heterogeneous_soil.in`

This example demonstrates how to build a more realistic soil model using a stochastic distribution of dielectric properties. A mixing model for soils proposed by Peplinski (http://dx.doi.org/10.1109/36.387598) is used to define a series of dispersive material properties for the soil.

```
1  #title: Heterogeneous soil using a stochastic distribution of dielectric␣
   ↪properties given by a mixing model from Peplinski
2  #domain: 0.15 0.15 0.1
3  #dx_dy_dz: 0.001 0.001 0.001
4  #time_window: 6e-9
5
6  #waveform: ricker 1 1.5e9 my_ricker
7  #hertzian_dipole: y 0.045 0.075 0.085 my_ricker
8  #rx: 0.105 0.075 0.085
9
10 #soil_peplinski: 0.5 0.5 2.0 2.66 0.001 0.25 my_soil
11 #fractal_box: 0 0 0 0.15 0.15 0.070 1.5 1 1 1 50 my_soil my_soil_box
12 #add_surface_roughness: 0 0 0.070 0.15 0.15 0.070 1.5 1 1 0.065 0.080 my_soil_box
13
14 #geometry_view: 0 0 0 0.15 0.15 0.1 0.001 0.001 0.001 heterogeneous_soil n
```

Line 10 defines a series of dispersive materials to represent a soil with sand fraction 0.5, clay fraction 0.5, bulk density 2 g/cm^3, sand particle density of 2.66 g/cm^3, and a volumetric water fraction range of 0.001 - 0.25. The volumetric water fraction is given as a range which is what defines a series of dispersive materials.

These materials can then be distributed stochastically over a volume using the `#fractal_box` command. Line 11 defines a volume, a fractal dimension, a number of materials, and a mixing model to use. The fractal dimension, 1.5, controls how the materials are stochastically distributed. The fractal weightings, 1, 1, 1, weight the fractal in the x, y, and z directions. The number of materials, 50, specifies how many dispersive materials to create using the mixing model (`my_soil`).

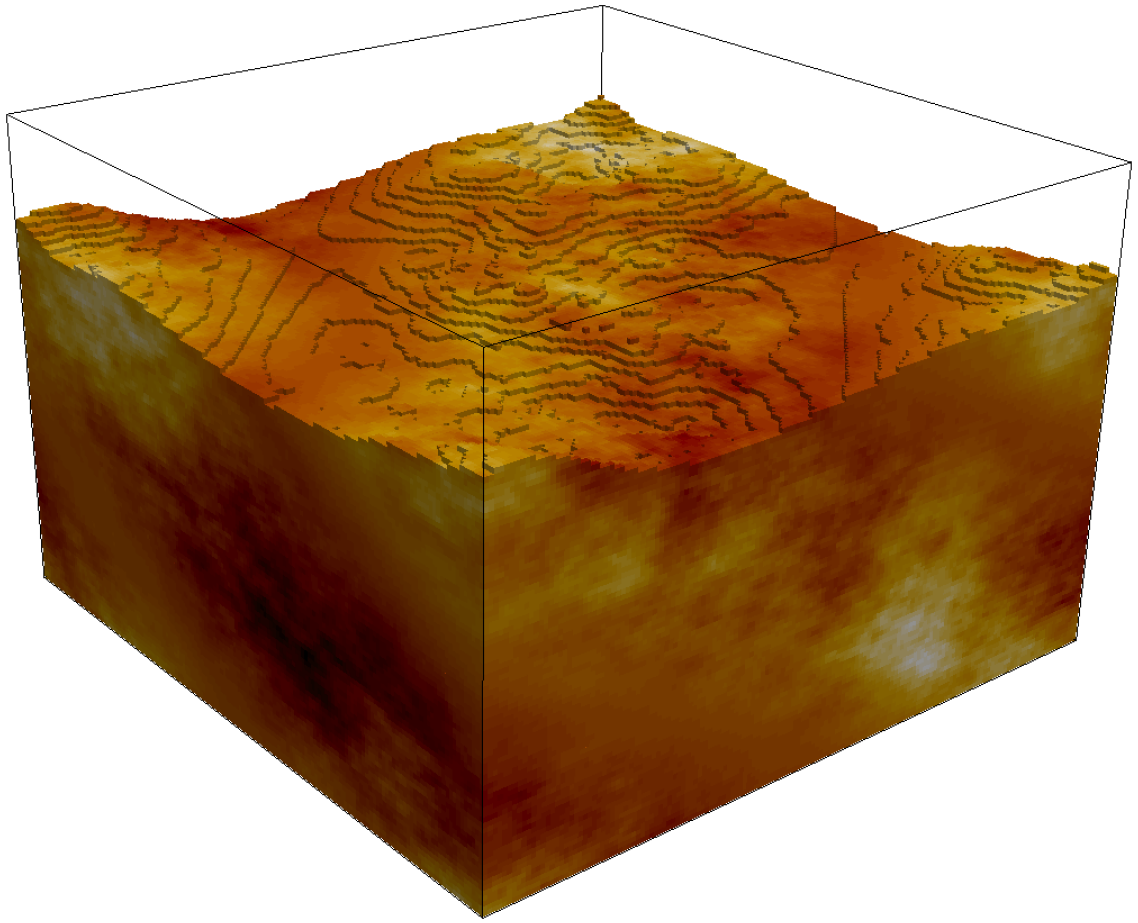


Fig. 18.1: FDTD geometry mesh showing a heterogeneous soil model with a rough surface.

18.1.1 Adding rough surfaces

A rough surface can be added to any side of `#fractal_box` using,

```
#add_surface_roughness: 0 0 0.070 0.15 0.15 0.070 1.5 1 1 0.065 0.080 my_soil_box
```

which defines one of the surfaces of the `#fractal_box`, a fractal dimension, and minimum and maximum values for the height of the roughness (relative to the original `#fractal_box` volume). In this example the roughness will be stochastically distributed with troughs up to 5mm deep, and peaks up to 10mm high.

More information, including adding surface water and vegetation, can be found in the *section on using the fractal box command*.

CHAPTER 19

FAQs

This section provides answers to frequently asked questions about gprMax and its uses.

What applications can gprMax simulate? gprMax is electromagnetic wave simulation software that is based on the Finite-Difference Time-Domain (FDTD) method. Many of its features have been designed to benefit simulating Ground Penetrating Radar (GPR), however it can be used to simulate many other applications in areas such as engineering, geophysics, archaeology, and medicine.

Why does gprMax not have a GUI? We considered developing a CAD-based graphical user interface (GUI) but, for now, decided against it. There were two guiding principals behind this design decision: firstly, users most often perform a series of related simulations with varying parameters to solve or optimize a particular problem; and secondly, we decided the limited resources we had were best concentrated on developing advanced modelling features for GPR within software that could easily be interfaced with other tools. Although a CAD-based GUI is useful for creating single simulations it becomes increasingly cumbersome for a series of simulations or where simulations contain heterogeneities, e.g. a model of a soil with stochastically varying electrical properties.

How is gprMax licensed? gprMax is released under the GNU General Public License v3 or higher (http://www.gnu.org/copyleft/gpl.html). This means when distributing derived works, the source code of the work must be made available under the same license.

Where does the name gprMax come from? The name gprMax comes from the joining of the acroymn for Ground Penetrating Radar - **gpr** - and the name of the Scottish scientist who formulated the classical theory of electromagnetic radiation, James Clerk Maxwell (https://en.wikipedia.org/wiki/James_Clerk_Maxwell) - **Max**.

Do I need to learn Python to use gprMax? No, but it can be beneficial to know a little Python. We have made it easier to create more complex simulations in gprMax through scripting in the input file. This is achieved by allowing blocks of Python code to be written in the input file which are executed when the file is read by gprMax.

Can I still do all my pre/post-processing for gprMax in MATLAB? Yes, MATLAB has built-in functions to read HDF5 files (http://uk.mathworks.com/help/matlab/high-level-functions.html).

Can I convert my output file to a text file, e.g. to import into Microsoft Excel Yes, we recommend you download HDFView (https://support.hdfgroup.org/products/java/hdfview/) which is free viewer for HDF files. You can then export any of datasets in the output file to a text (ASCII) file that can be imported into Microsoft Excel. To do so right-click on the dataset in HDFView and choose Export Dataset -> Export Data to Text File.

But converting my input file from the old version of gprMax will be really painful Hopefully not! We have provided a Python module to help you convert input files from the old version of gprMax to use syntax introduced in version 3.

How do I choose a spatial resolution for my simulation? Spatial resolution should be chosen to mitigate numerical dispersion and to adequately resolve geometry in your simulation. *A 2D example of modelling a metal*

cylinder in a dielectric provides guidance on how to determine spatial resolution.

I specified a certain piece of geometry but I don't see it when I view my geometry file. gprMax builds objects in a model in the order the objects were specified in the input file, using a layered canvas approach. This means, for example, a cylinder object which comes after a box object in the input file will overwrite the properties of the box object at any locations where they overlap. This approach allows complex geometries to be created using basic object building blocks.

Can I run gprMax on my HPC/cluster? Yes. gprMax has been parallelised using OpenMP and features a task farm based on MPI. For more information read the *parallel performance section of the User Guide*

CHAPTER 20

Screencasts & videos

This section provides links to screencasts and videos that explain how to install gprMax, demonstrate some of its key features, and give example models showing applications.

20.1 Installation

These screencasts are demonstrated on Microsoft Windows 7, but the installation and updating procedure is the same for other versions of Windows, and quite similar for Linux and macOS also. Detailed written installation and updating instructions are provided in the Getting Started section.

- How to install gprMax on Microsoft Windows (https://youtu.be/YkPWMmJILcI)
- How to update gprMax on Microsoft Windows (https://youtu.be/e0ROY792s9o)

20.2 Plotting

How to plot time and frequency domain representations of source waveforms and field outputs.

- Plotting source waveforms using a Jupyter notebook (https://youtu.be/zaf0w8Np2cU)

20.3 Visualise EM wave propagation

You can use the `#snapshot` command and Paraview to visualise how electromagnetic waves propagate in a simulation.

- PEC cylinder buried in a lossless half-space (2D) (https://youtu.be/BpBo0-SFda4)
- PEC cylinder buried in different half-spaces (2D) (https://youtu.be/g744O_wb14I)

CHAPTER 21

Code Overview

This section aims to provide information and advice for developers who want to get started using and modifying the gprMax code.

The code has been written in Python (3.x) with performance-critical parts, i.e. the FDTD solver, written using Cython (for CPU) or the NVIDIA CUDA programming model (for GPU). Cython allows the CPU-based solver to be parallelised using OpenMP which enables it to run on multi-core CPUs. gprMax also features a Messaging Passing Interface (MPI) task farm, which can operate with CPU nodes or multiple GPUs.

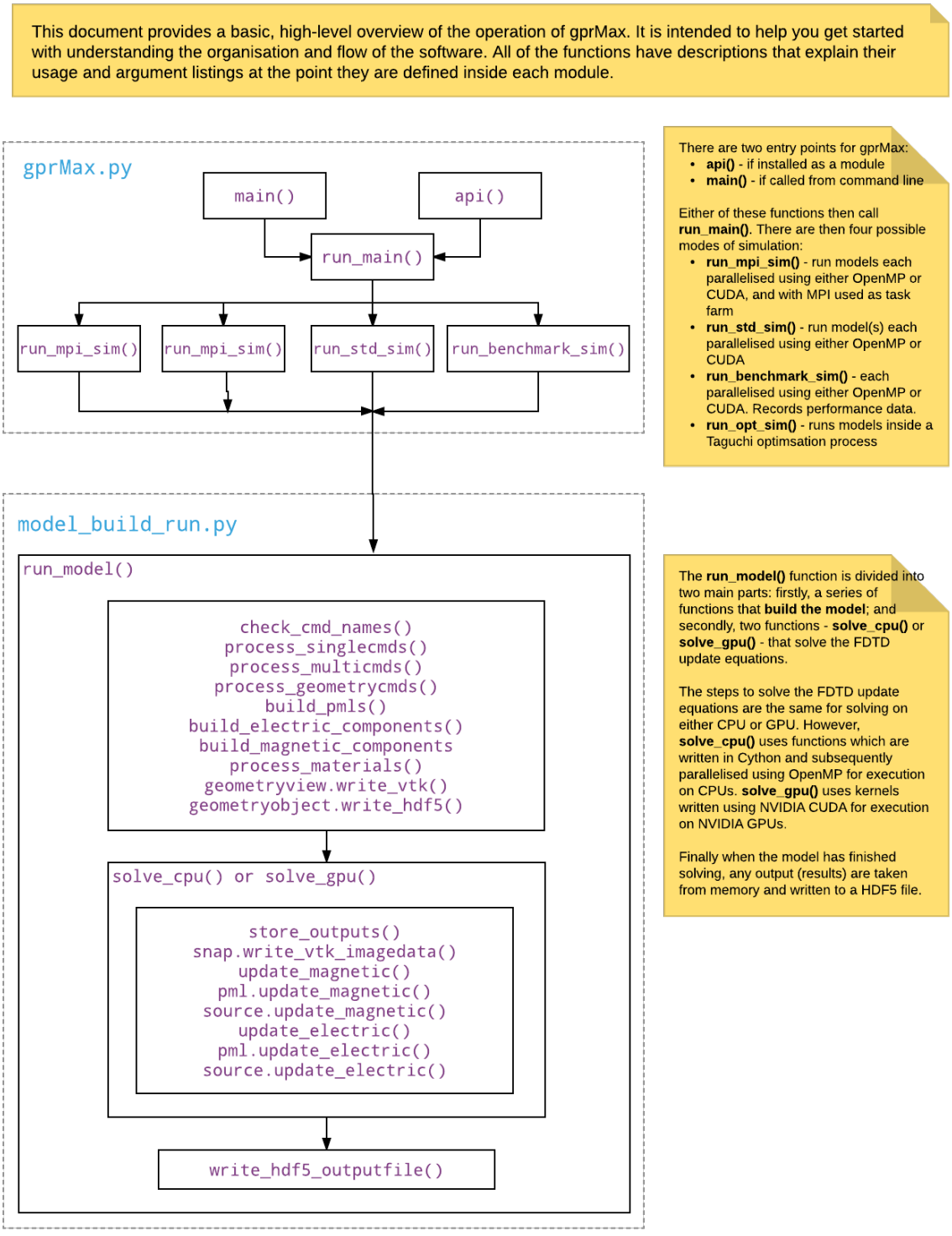


Fig. 21.1: Basic, high-level overview of the flow of control (operation) of the code.

CHAPTER 22

Analytical comparisons

This section presents comparisons between analytical solutions and modelled solutions using gprMax.

22.1 Hertzian dipole in free space

`hertzian_dipole_fs.in`

This example is of a Hertzian dipole, i.e. an additive source (electric current density), in free space.

```
#title: Hertzian dipole in free-space
#domain: 0.100 0.100 0.100
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 3e-9

#waveform: gaussiandot 1 1e9 myWave
#hertzian_dipole: z 0.050 0.050 0.050 myWave
#rx: 0.070 0.070 0.070
```

The function `hertzian_dipole_fs`, which can be found in the `analytical_solutions` module in the `tests` sub-package, computes the analytical solution.

22.1.1 Results

Fig. 22.1 shows the time history of the electric and magnetic field components of the modelled and analytical solutions. The responses completely overlap one another due to their similarity. Therefore, Fig. 22.2 shows the percentage differences between the modelled and analytical solutions.

The match between the analytical and numerically modelled solutions is excellent. The maximum difference is approximately 1%, which is observed in the Ez field component (the same direction as the Hertzian dipole source). The other electric field components exhibit maximum differences of approximately 0.5%, and the magnetic field components 0.25%.

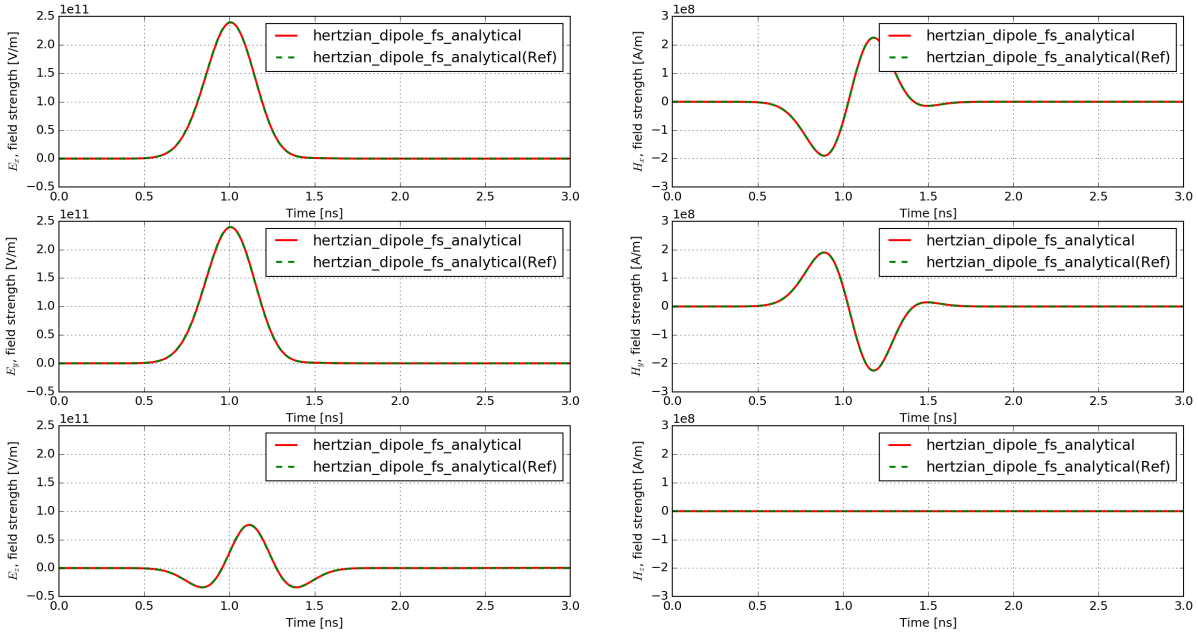


Fig. 22.1: Time history of the electric and magnetic field components of the modelled and analytical solutions ('Ref', in this case, indicates solution calculated from theory).

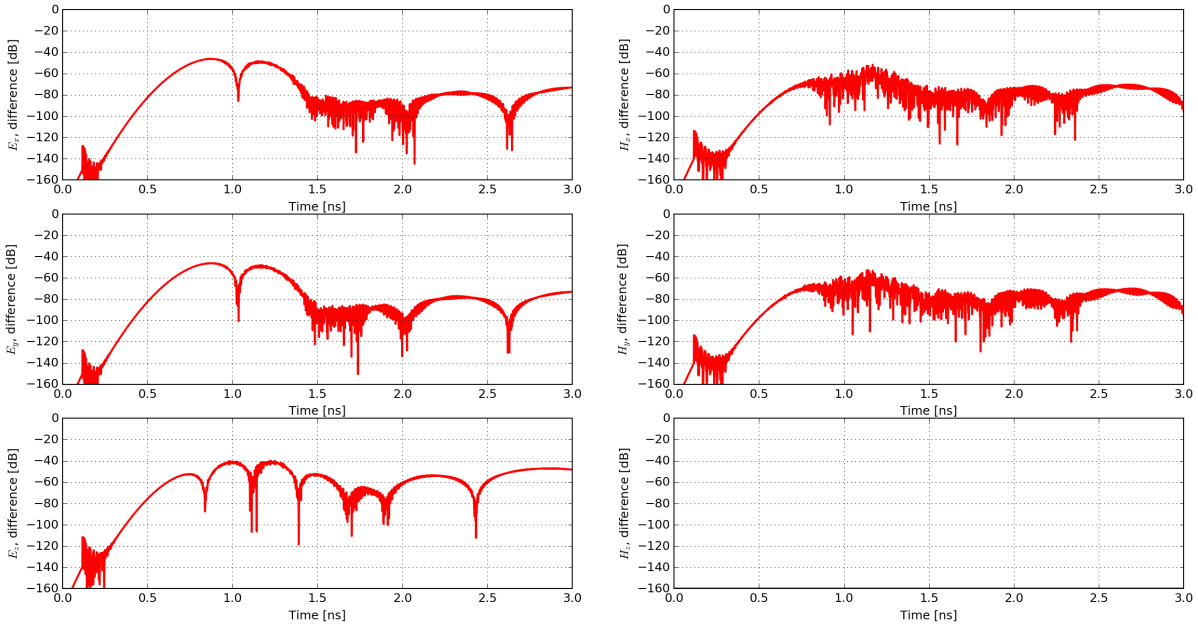


Fig. 22.2: Percentage differences between the modelled and analytical solutions.

22.2 Half-wave dipole in free space

See the *section on antenna example models* for the simulated s11 parameter and input impedance of a half-wave dipole antenna in free space. The resonant frequency and input impedance from the model agree very well with the theoretical predictions for a half-wave dipole antenna.

CHAPTER 23

Numerical comparisons

This section presents comparisons of models using different numerical modelling techniques.

23.1 FDTD/MoM

The Finite-Difference Time-Domain (FDTD) method from gprMax is compared with the Method of Moments (MoM) from the MATLAB antenna toolbox (http://uk.mathworks.com/products/antenna/).

23.1.1 Bowtie antenna in free space

This example considers the input impedance of a planar bowtie antenna in free space. The length and height of the bowtie are 100mm, giving a flare angle of 90°.

FDTD model

antenna_bowtie_fs.in

```
#python:

from gprMax.input_cmd_funcs import *

title = 'antenna_bowtie_fs'
print('#title: {}'.format(title))

domain = domain(0.200, 0.200, 0.100)
dxdydz = dx_dy_dz(0.001, 0.001, 0.001)
time_window = time_window(30e-9)
bowtie_dims = (0.050, 0.100) # Length, height
tx_pos = (domain[0]/2, domain[1]/2, domain[2]/2)

# Source excitation and type
print('#waveform: gaussian 1 1.5e9 mypulse')
print('#transmission_line: x {:g} {:g} {:g} 50 mypulse'.format(tx_pos[0], tx_
↪pos[1], tx_pos[2]))

```

(continues on next page)

(continued from previous page)

```
# Bowtie - upper x half
triangle(tx_pos[0], tx_pos[1], tx_pos[2], tx_pos[0] + bowtie_dims[0] + 2 *␣
↪dxdydz[0], tx_pos[1] - bowtie_dims[1]/2, tx_pos[2], tx_pos[0] + bowtie_dims[0] +␣
↪2 * dxdydz[0], tx_pos[1] + bowtie_dims[1]/2, tx_pos[2], 0, 'pec')

# Bowtie - lower x half
triangle(tx_pos[0] + dxdydz[0], tx_pos[1], tx_pos[2], tx_pos[0] - bowtie_dims[0],␣
↪tx_pos[1] - bowtie_dims[1]/2, tx_pos[2], tx_pos[0] - bowtie_dims[0], tx_pos[1] +␣
↪bowtie_dims[1]/2, tx_pos[2], 0, 'pec')

# Detailed geometry view around bowtie and feed position
geometry_view(tx_pos[0] - bowtie_dims[0] - 2*dxdydz[0], tx_pos[1] - bowtie_dims[1]/
↪2  - 2*dxdydz[1], tx_pos[2] - 2*dxdydz[2], tx_pos[0] + bowtie_dims[0] +␣
↪2*dxdydz[0], tx_pos[1] + bowtie_dims[1]/2  + 2*dxdydz[1], tx_pos[2] +␣
↪2*dxdydz[2], dxdydz[0], dxdydz[1], dxdydz[2], title + '_pcb', type='f')

#end_python:
```

A Gaussian waveform with a centre frequency of 1.5GHz was used to excite the antenna, which was fed by a transmission line with a characteristic impedance of 50 Ohms.

The module `plot_antenna_params` from the `tools` subpackage was used to calculate and plot the input impedance from the FDTD model.

MoM model

The bowtie antenna was created using the antenna toolbox in MATLAB, and the `bowtieTriangular` class.

```
bowtie = bowtieTriangular('Length', 0.1)
zin = impedance(bowtie, 33.33e6:33.33e6:6e9)
```

23.1.2 Results

Fig. 23.1 shows the input impedance (resistive and reactive) for the FDTD (gprMax) and MoM (MATLAB) models. The frequency resolution for the FFT used in both models was $\Delta f = 33.33\ MHz$.

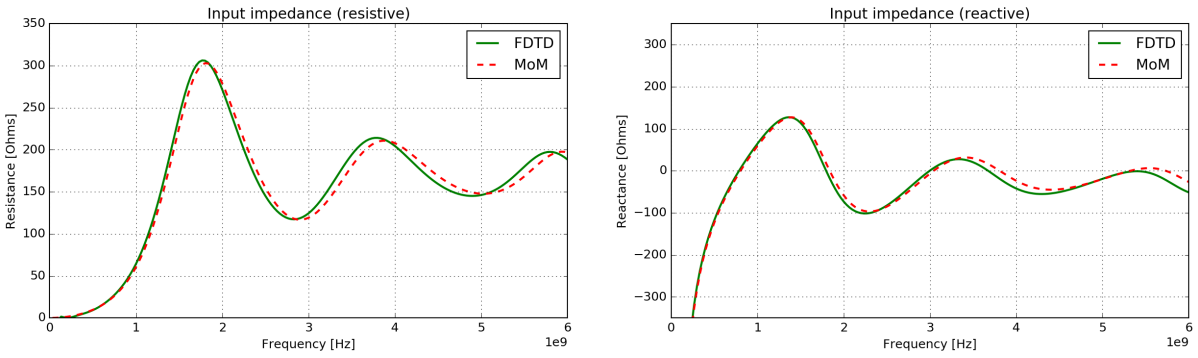


Fig. 23.1: Input impedance (resistive and reactive) of a bowtie antenna in free space using FDTD (gprMax) and MoM (MATLAB) models.

The results from the FDTD and MoM modelling techniques are in very good agreement. The biggest mismatch occurs in the resistive part of the input impedance at frequencies above 3GHz.

CHAPTER 24

Performance benchmarking

This section provides information and results from performance benchmarking of gprMax.

24.1 How to benchmark?

The following simple models (found in the `tests/benchmarking` sub-package) can be used to benchmark gprMax on your own system. The models feature different domain sizes and contain a simple source in free space.

```
#domain: 0.1 0.1 0.1
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 3e-9

#waveform: gaussiandotnorm 1 900e6 MySource
#hertzian_dipole: x 0.05 0.05 0.05 MySource
#rx: 0.05 0.05 0.05
```

```
#domain: 0.15 0.15 0.15
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 3e-9

#waveform: gaussiandotnorm 1 900e6 MySource
#hertzian_dipole: x 0.05 0.05 0.05 MySource
#rx: 0.05 0.05 0.05
```

```
#domain: 0.2 0.2 0.2
#dx_dy_dz: 0.001 0.001 0.001
#time_window: 3e-9

#waveform: gaussiandotnorm 1 900e6 MySource
#hertzian_dipole: x 0.05 0.05 0.05 MySource
#rx: 0.05 0.05 0.05
```

Using the following steps to collect and report benchmarking results for each of the models:

1. Run gprMax in benchmarking mode, e.g. `python -m gprMax tests/benchmarking/bench_100x100x100.in -benchmark`

2. Use the `plot_benchmark` module to create plots of the execution time and speed-up, e.g. `python -m tests.benchmarking.plot_benchmark tests/benchmarking/bench_100x100x100.npz`. You can combine results into a single plot, e.g. e.g. `python -m tests.benchmarking.plot_benchmark tests/benchmarking/bench_100x100x100.npz --otherresults tests/benchmarking/bench_150x150x150.npz`.
3. Share your data by emailing us your Numpy archives and plot files to info@gprmax.com

24.2 Results

24.2.1 Mac OS X

iMac15,1

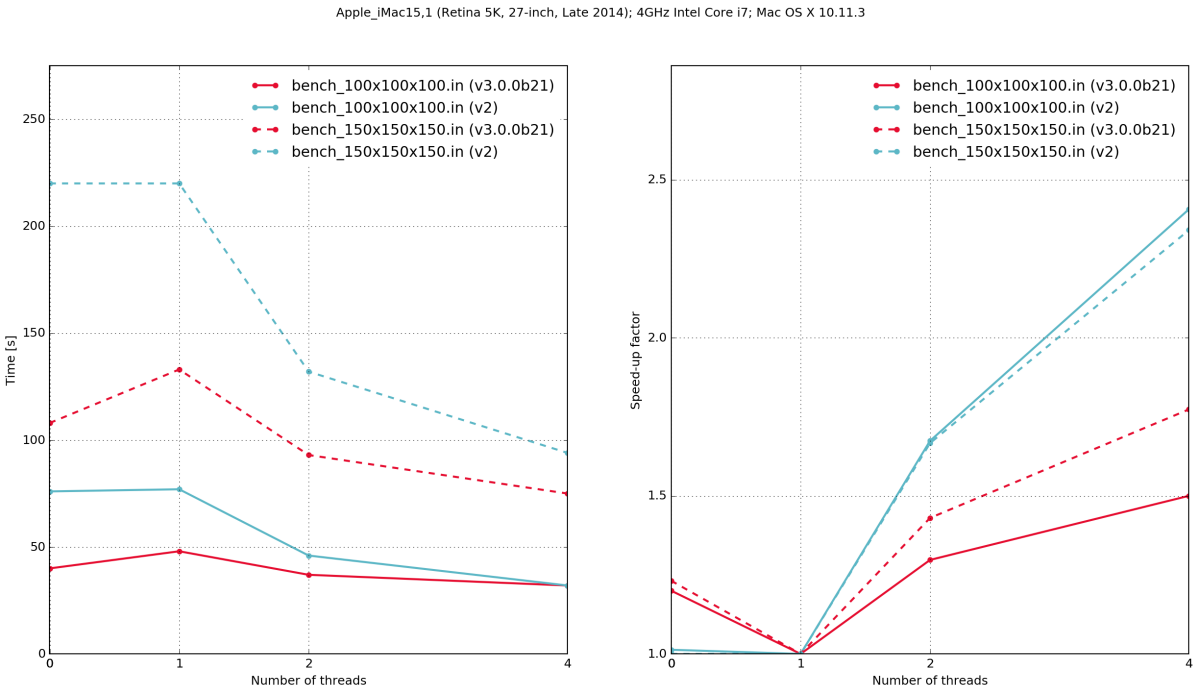
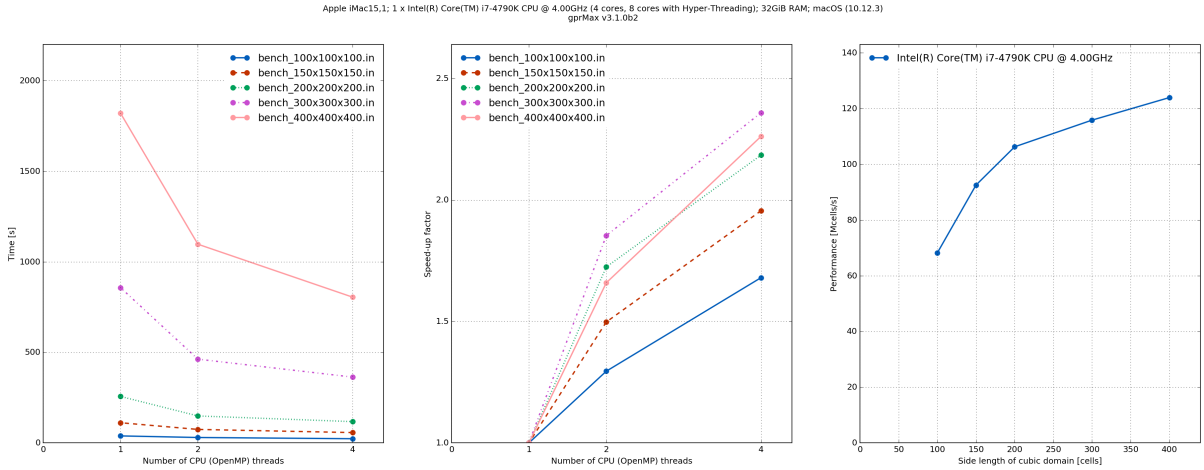
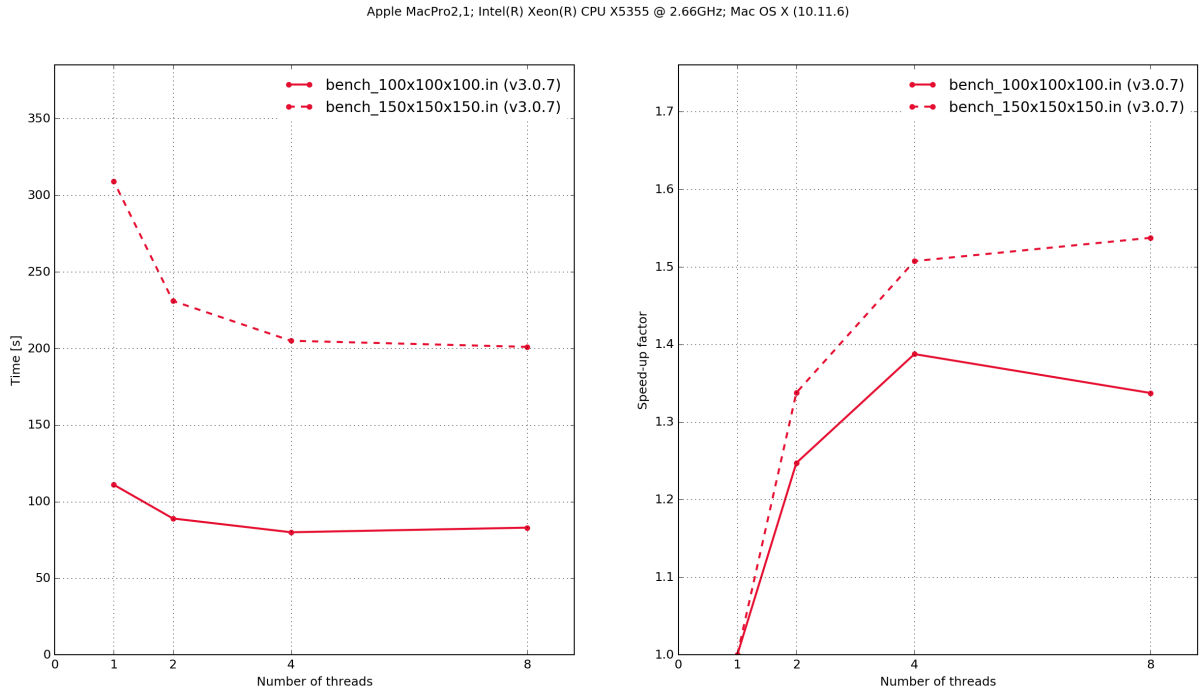


Fig. 24.1: Execution time and speed-up factor plots for Python/Cython-based gprMax and previous (v.2) C-based code.

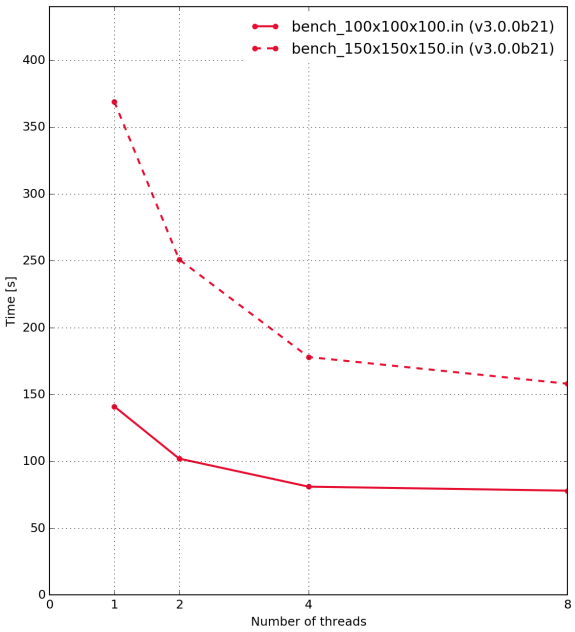
Note: Zero threads indicates that the code was compiled serially, i.e. without using OpenMP.

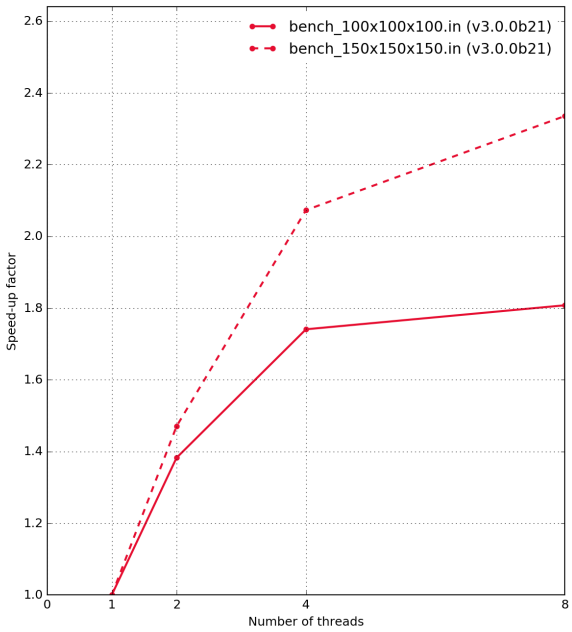
The results demonstrate that the Python/Cython-based code is faster, in these two benchmarks, than the previous version which was written in C. It also shows that the performance scaling with multiple OpenMP threads is better with the C-based code. Results from the C-based code show that when it is compiled serially the performance is approximately the same as when it is compiled with OpenMP and run with a single thread. With the Python/Cython-based code this is not the case. The overhead in setting up and tearing down the OpenMP threads means that for a single thread the performance is worse than the serially-compiled version.


Apple iMac15,1; 1 x Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz (4 cores, 8 cores with Hyper-Threading); 32GiB RAM; macOS (10.12.3)
gprMax v3.1.0b2
bench_100x100x100.in
bench_150x150x150.in
bench_200x200x200.in
bench_300x300x300.in
bench_400x400x400.in
Time [s]
Number of CPU (OpenMP) threads
Speed-up factor
Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz
Performance [Mcells/s]
Side length of cubic domain [cells]


Apple MacPro2,1; Intel(R) Xeon(R) CPU X5355 @ 2.66GHz; Mac OS X (10.11.6)
bench_100x100x100.in (v3.0.7)
bench_150x150x150.in (v3.0.7)
Time [s]
Number of threads
Speed-up factor

Apple_MacPro3,1 (2008); 2 x 2.8GHz Quad-Core Intel Xeon; Mac OS X 10.11.2





iMac15,1

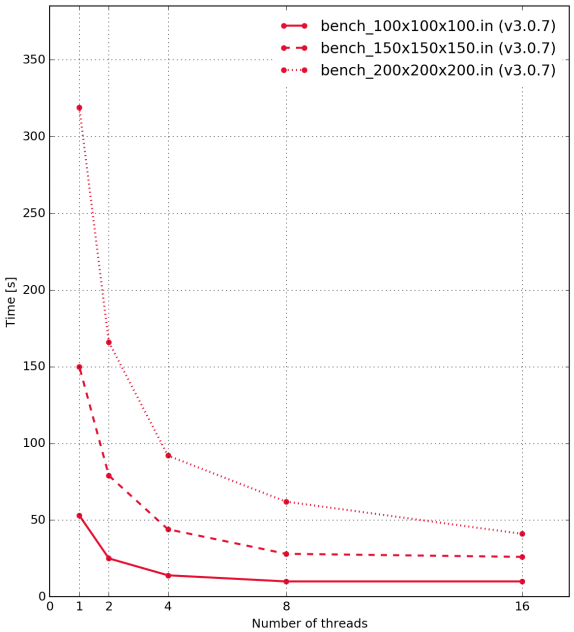
MacPro1,1

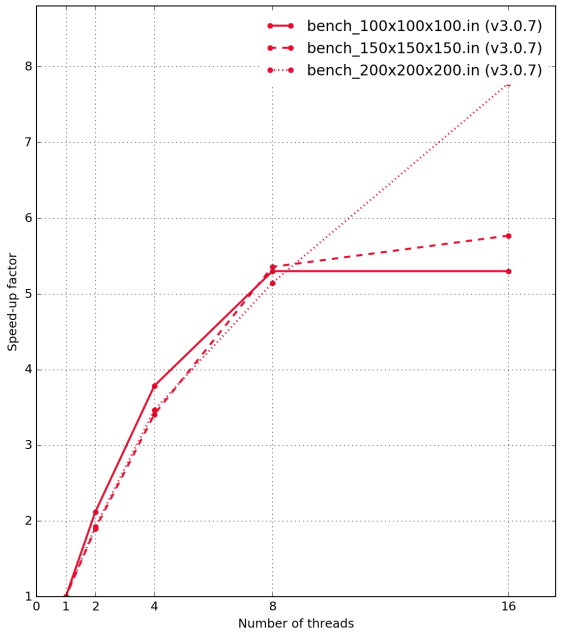
MacPro3,1

24.2.2 Linux

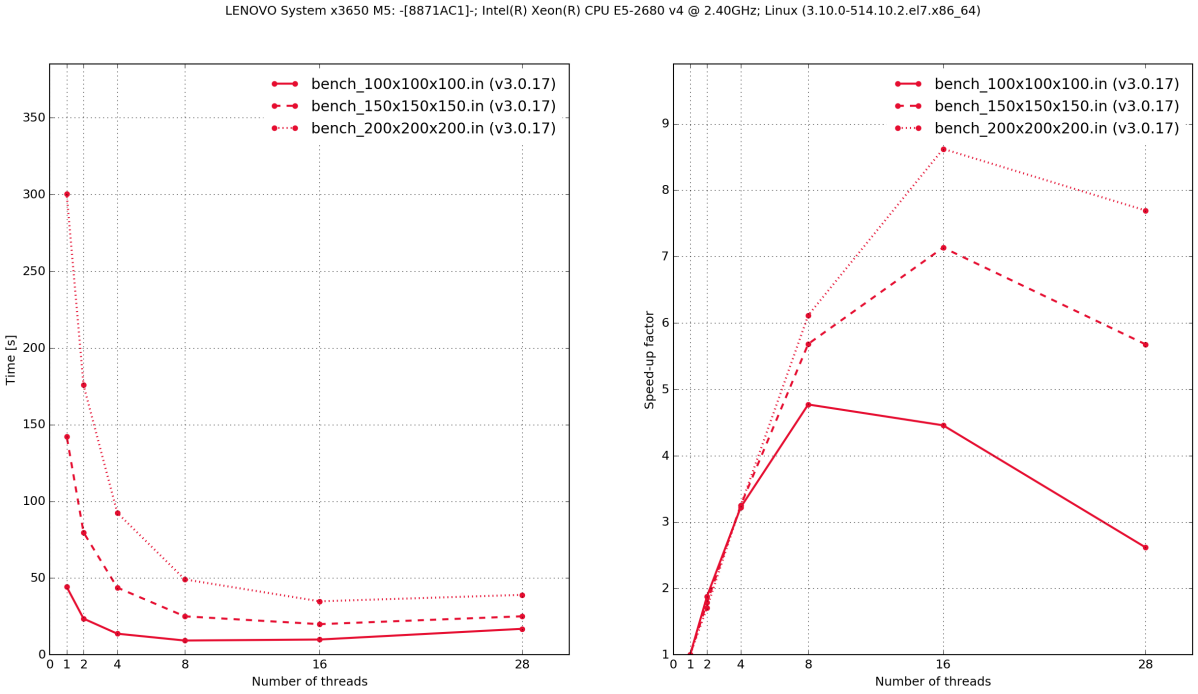
Dell PowerEdge R630

Dell PowerEdge R630; Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz; Linux (3.10.0-327.18.2.el7.x86_64)

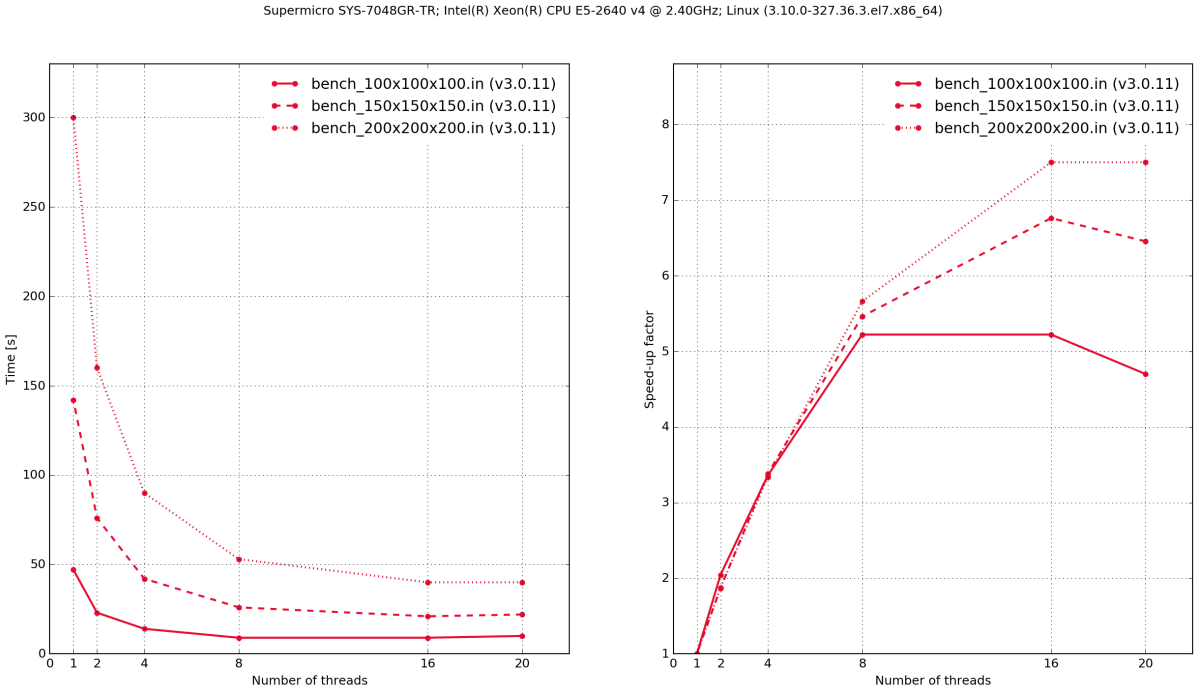




Lenovo System x3650 M5



SuperMicro SYS-7048GR-TR

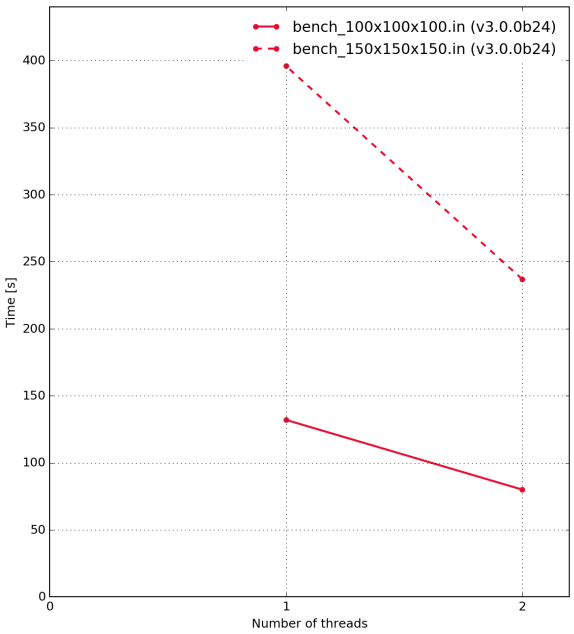


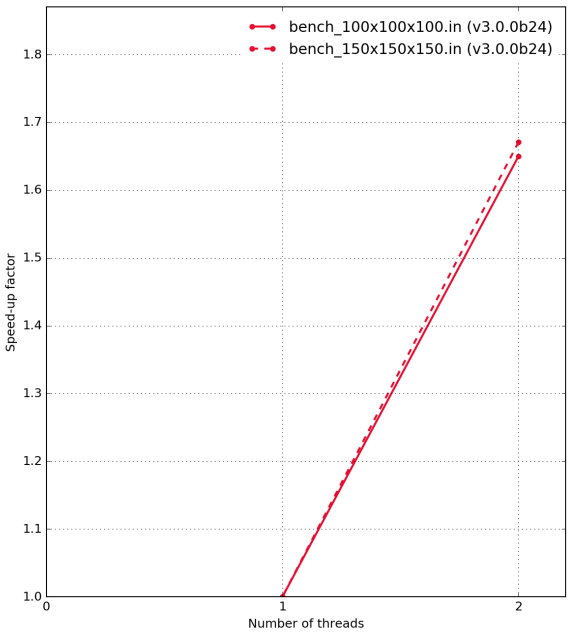
24.2.3 Windows

Lenovo T430

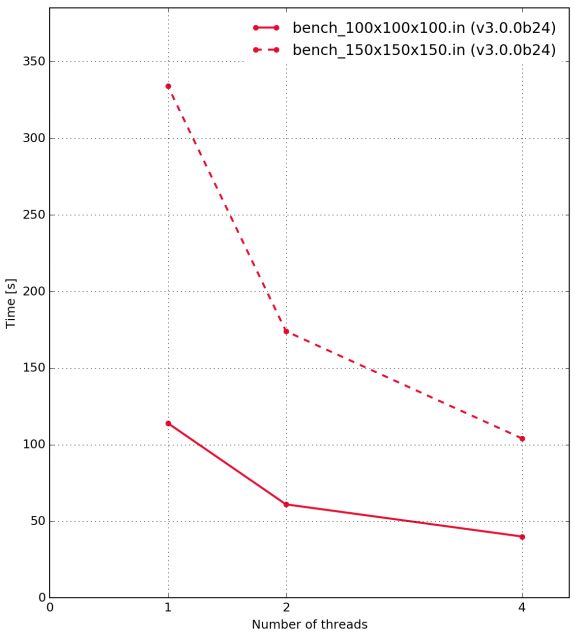
Dell Z420

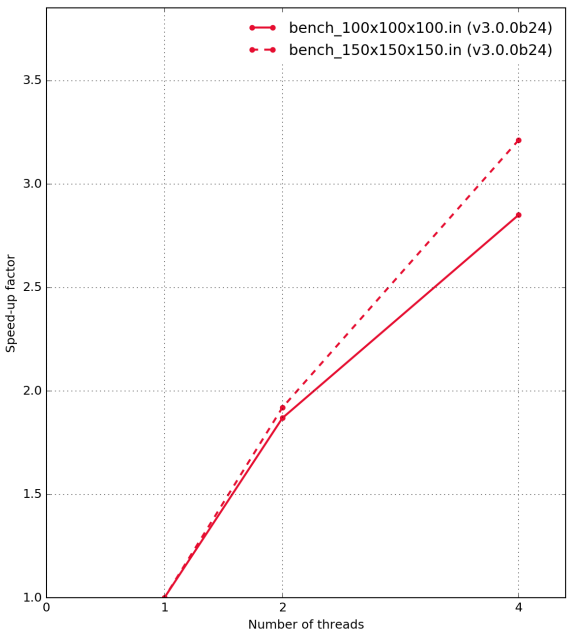
Lenovo T430; Intel(R) Core(TM) i5-3210M CPU @ 2.5GHz; Windows 7 (64-bit)





Dell Z420; Intel(R) Xeon(R) E5-1620 CPU @ 3.6GHz; Windows 7 (64-bit)





CHAPTER 25

References

Bibliography

[BAL2005] Balanis, C. A. (2005). Antenna theory: analysis and design (Vol. 1). John Wiley & Sons.

[BER1998] Bergmann, T., Robertsson, J. O., & Holliger, K. (1998). Finite-difference modeling of electromagnetic wave propagation in dispersive and attenuating media. Geophysics, 63(3), 856-867. (http://dx.doi.org/10.1190/1.1444396)

[BOU1996] Bourgeois, J. M., & Smith, G. S. (1996). A fully three-dimensional simulation of a ground-penetrating radar: FDTD theory compared with experiment. Geoscience and Remote Sensing, IEEE Transactions on, 34(1), 36-44. (http://dx.doi.org/10.1109/36.481890)

[BUR1981] Burrough, P. A. (1981). Fractal dimensions of landscapes and other environmental data. Nature, 294(5838), 240-242. (http://dx.doi.org/10.1038/294240a0)

[DOB1985] Dobson, M. C., Ulaby, F. T., Hallikainen, M. T., & El-Rayes, M. (1985). Microwave dielectric behavior of wet soil-Part II: Dielectric mixing models. Geoscience and Remote Sensing, IEEE Transactions on, (1), 35-46. (http://dx.doi.org/10.1109/tgrs.1985.289498)

[HILL1998] Hillel, D. (1998). Environmental soil physics: Fundamentals, applications, and environmental considerations. Academic press. (http://dx.doi.org/10.1016/b978-012348525-0/50030-6)

[GED1998] Gedney, S. D. (1998). The perfectly matched layer absorbing medium. Advances in Computational Electrodynamics: The Finite-Difference Time-Domain Method, 263-344.

[GIAK2012] Giannakis, I., Giannopoulos, A., & Davidson, N. (2012). Incorporating dispersive electrical properties in FDTD GPR models using a general Cole-Cole dispersion function. In 2012 14th International Conference on Ground Penetrating Radar (GPR). (http://dx.doi.org/10.1109/icgpr.2012.6254866)

[GIA2014] Giannakis, I., & Giannopoulos, A. (2014). A Novel Piecewise Linear Recursive Convolution Approach for Dispersive Media Using the Finite-Difference Time-Domain Method. Antennas and Propagation, IEEE Transactions on, 62(5), 2669-2678. (http://dx.doi.org/10.1109/tap.2014.2308549)

[GIA1997] Giannopoulos, A. (1997). The investigation of Transmission-Line Matrix and Finite-Difference Time-Domain Methods for the Forward Problem of Ground Probing Radar, D.Phil thesis, Department of Electronics, University of York, UK. (http://etheses.whiterose.ac.uk/id/eprint/2443)

[GIA2012] Giannopoulos, A. (2012). Unsplit implementation of higher order PMLs. Antennas and Propagation, IEEE Transactions on, 60(3), 1479-1485. (http://dx.doi.org/10.1109/tap.2011.2180344)

[IRE2013] Ireland, D., & Abbosh, A. (2013). Modeling human head at microwave frequencies using optimized Debye models and FDTD method. Antennas and Propagation, IEEE Transactions on, 61(4), 2352-2355. (http://dx.doi.org/10.1109/tap.2013.2242037)

[KUN1993] Kunz, K. S., & Luebbers, R. J. (1993). The finite difference time domain method for electromagnetics. CRC press.

[LI2013] Li, J., Guo, L. X., Jiao, Y. C., & Wang, R. (2013). Composite scattering of a plasma-coated target above dispersive sea surface by the ADE-FDTD method. Geoscience and Remote Sensing Letters, IEEE, 10(1), 4-8. (http://dx.doi.org/10.1109/lgrs.2012.2189751)

[LUE1994] Luebbers, R., Steich, D., & Kunz, K. (1993). FDTD calculation of scattering from frequency-dependent materials. Antennas and Propagation, IEEE Transactions on, 41(9), 1249-1257. (http://dx.doi.org/10.1109/8.247751)

[MAL1994] Maloney, J. G., Shlager, K. L., & Smith, G. S. (1994). A simple FDTD model for transient excitation of antennas by transmission lines. Antennas and Propagation, IEEE Transactions on, 42(2), 289-292.

[PIE2009] Pieraccini, M., Bicci, A., Mecatti, D., Macaluso, G., & Atzeni, C. (2009). Propagation of large bandwidth microwave signals in water. Antennas and Propagation, IEEE Transactions on, 57(11), 3612-3618. (http://dx.doi.org/10.1109/tap.2009.2025674)

[STA2017] Stadler, S. (2017). A Forward Modeling Study for the Investigation of the Vertical Water-Content Distribution of Using Guided GPR Waves (Master's Thesis, Freiberg University of Mining and Technology, Germany) (https://app.box.com/s/h2n2ytsdllcm77du1o9lknumngmq8vq8)

[TAF2005] Taflove, A., & Hagness, S. C. (2005). Computational electrodynamics. Artech house.

[TEI1998] Teixeira, F. L., Chew, W. C., Straka, M., Oristaglio, M. L., & Wang, T. (1998). Finite-difference time-domain simulation of ground penetrating radar on dispersive, inhomogeneous, and conductive soils. Geoscience and Remote Sensing, IEEE Transactions on, 36(6), 1928-1937. (http://dx.doi.org/10.1109/36.729364)

[TUR1987] Turcotte, D. L. (1987). A fractal interpretation of topography and geoid spectra on the Earth, Moon, Venus, and Mars. Journal of Geophysical Research: Solid Earth (1978–2012), 92(B4), E597-E601. (http://dx.doi.org/10.1029/jb092ib04p0e597)

[TUR1997] Turcotte, D. L. (1997). Fractals and chaos in geology and geophysics. Cambridge university press. (http://dx.doi.org/10.1017/cbo9781139174695)

[VIA2005] Vial, A., Grimault, A. S., Macías, D., Barchiesi, D., & de La Chapelle, M. L. (2005). Improved analytical fit of gold dispersion: Application to the modeling of extinction spectra with a finite-difference time-domain method. Physical Review B, 71(8), 085416. (http://dx.doi.org/10.1103/physrevb.71.085416)

[WAR2011] Warren, C., & Giannopoulos, A. (2011). Creating finite-difference time-domain models of commercial ground-penetrating radar antennas using Taguchi's optimization method. Geophysics, 76(2), G37-G47. (http://dx.doi.org/10.1190/1.3548506)

[WEN2007a] Weng, W. C., Yang, F., & Elsherbeni, A. (2007). Electromagnetics and antenna optimization using Taguchi's method. Synthesis Lectures on Computational Electromagnetics, 2(1), 1-94.

[WEN2007b] Weng, W. C., Yang, F., & Elsherbeni, A. Z. (2007). Linear antenna array synthesis using Taguchi's method: A novel optimization technique in electromagnetics. Antennas and Propagation, IEEE Transactions on, 55(3), 723-730.

[WHI2009] Whittow, W. G., & Edwards, R. M. (2009). Effects of averaging procedures for electrical properties at the interface of dissimilar tissues in the human head with finite-difference time-domain modelling. Science, Measurement & Technology, IET, 3(1), 51-58.

[YEE1966] Yee, K. S. (1966). Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. IEEE Trans. Antennas Propag, 14(3), 302-307. (http://dx.doi.org/10.1109/TAP.1966.1138693)