PETGEM Documentation

Release 0.8

Octavio Castillo-Reyes

Jun 03, 2021

CONTENTS

CHAPTER

ONE

WHAT IS PETGEM?

Electromagnetic methods (EM) are an established tool in geophysics, with application in many areas such as hydrocarbon and mineral exploration, reservoir monitoring, CO_2 storage characterization, geothermal reservoir imaging and many others. In particular, the marine Controlled-Source Electromagnetic method (CSEM) and the 3D magnetotelluric (MT) method have become important techniques for reducing ambiguities in data interpretation in exploration geophysics. In order to be able to predict the EM signature of a given geological structure, modelling tools provide us with synthetic results which we can then compare to real data. In particular, if the geology is structurally complex, one might need to use methods able to cope with such complexity in a natural way by means of, e.g., an unstructured mesh representing its geometry. Among the modelling methods for EM based upon 3D unstructured meshes, the High-order Nédélec Finite Elements (FE), a type of Edge Finite Elements, offer a good trade-off between accuracy and number of degrees of freedom, i.e. size of the problem.

In the multi-core and many-core era, parallelization is a crucial issue. Nédélec FE offer good scalability potential. Its low DOF number make them potentially fast, which is crucial in the future goal of solving inverse problems which might involve over 100,000 realizations (e.g. within a inversion routine). However, the state of the art shows a relative scarcity of robust high-order edge-based codes to simulate these problems.

On top of that, **Parallel Edge-based Tool for Geophysical Electromagnetic Modelling** (PETGEM) is a Python tool for the scalable solution of the EM modeling on tetrahedral meshes, as these are the easiest to scale-up to very large domains or arbitrary shape. It supports distributed-memory paralelism through petsc4py package.

As a result, PETGEM tool allow users to specify high-order edge-based variational forms of H(curl) for the simulation of electromagnetic fields in realistic 3D active-source (e.g. CSEM) and 3D passive-source (e.g. MT) surveys with accuracy, reliability and efficiency.

PETGEM is developed as open-source under `BSD-3` license at Computer Applications in Science & Engineering (CASE) of the Barcelona Supercomputing Center (BSC). Requests and contributions are welcome.

CHAPTER

TWO

FEATURES

PETGEM use a code structure for the high-order Nédélec FE algorithm that emphasizes good parallel scalability, which is crucial in the multi-core era. Furthermore, it's modularity should simplify the process of reaching the best possible performance in terms of percentage of the peak amount of floating point operations provided by the architecture.

2.1 Software stack

An outline of the primary groups of modules in PETGEM design is given in *Figure 3.1*.

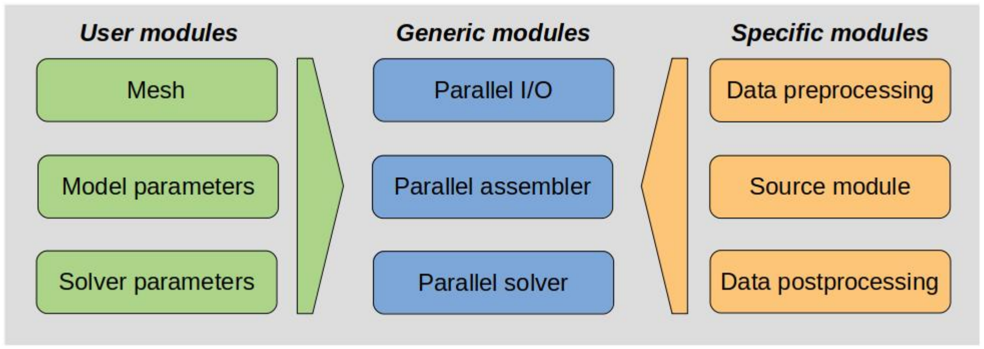

Fig. 1: Figure 3.1. Upper view of PETGEM software stack.

A more detailed explanation is the following:

- Modular and extensible EFEM kernel: The kernel is extensible in any direction. Therefore, the possibility of adding new features such as new boundary conditions, numerical algorithms, analysis modules, among others.
- Independent of problem formulation, numerical solution, and data storage: The kernel provides the independent abstractions for modeling, numerical methods, data storage and analysis.
- Parallel processing support: Based on distributed-memory parallelism (petsc4py) and static load balancing. Further, GPUs architectures are supported.
- Confidence and performance monitoring: Based on an simple and automatic profiling module.
- Efficient solvers & preconditioners: Direct as well as iterative solvers and preconditioners are supported through petsc4py package. As result, PETSc and MUMPs libraries are supported.
- Interface to mesh generator: Simple and light library to export nodal finite element meshes to edge elements data structures. Current version support Gmsh meshes.

- Edge FEM library: High-order Edge-based discretisations, vector basis functions, their geometry description, and generalized integration rules provides a generic support for implementing high-order edge-based solution algorithms.
- Linear systems library: Support to Compressed Row Storage (CSR) format for sparse matrices and their easy and efficient parallel assembly on distributed-memory platforms.
- CSEM/MT module: Ad-hoc design to meet specific requirements for the solution 3D CSEM/MT surveys, namely, conductivity model, physical parameters, transmitter and receiver lists.
- Pre-processing suite: Set of Python functions for pre-processing phase within PETGEM.
- Documentation: Available in HTML and PDF format through Sphinx and LaTeX (textlive).

2.2 Programming language

PETGEM is based on Python language programming because:

- It is open source and functional on a wide number of platforms, including HPC environments.
- It uses a high level and very expressive language.
- It is based on a sophisticated array manipulation in a Fortran-like manner.
- It uses a good body of bindings to common tools needed in scientific computing: plotting, numerical libraries, debugging and testing.

The code structure is modular, simple and flexible which allows exploiting not just our own modules but also third party libraries. Therefore, the software stack includes interfaces to external suites of data structures and libraries that contain most of the necessary building blocks needed for programming large scale numerical applications, e.g. sparse matrices, vectors, iterative and direct solvers. As a result, the code is compact and eliminates the need to write such libraries and thus speeds up development time by orders of magnitude[1],[2],[3],[4].

In order to meet the high computational cost of the modeling, PETGEM supports distributed-memory parallelism through Petsc4py package.

2.3 Target architecture

The HPC goal of the PETGEM involves using cutting-edge architectures. To that goal, the code is implemented in current state-of-the-art platforms such as Intel Xeon Platinum processors from the Skylake generation, Intel Haswell and Intel Xeon Phi processors, which offer high-performance, flexibility, and power efficiency. Nevertheless, PETGEM support older architectures such as SandyBridge, for the sake of usability and to be able to compare performance.

[1] Bangerth, W., Burstedde, C., Heister, T., and Kronbichler, M. (2011). Algorithms and data structures for massively parallel generic adaptive finite element codes. ACM Transactions on Mathematical Software (TOMS), 38(2):14.

[2] Heister, T., Kronbichler, M., and Bangerth, W. (2010). Massively parallel finite element programming. Recent Advances in the Message Passing Interface, pages 122–131.

[3] Key, K. and Ovall, J. (2011). A parallel goal-oriented adaptive finite element method for 2.5-d electromagnetic modelling. Geophysical Journal International, 186(1):137– 154.

[4] Osseyran, A. and Giles, M. (2015). Industrial Applications of High-Performance Computing: Best Global Practices. Chapman & Hall/CRC Computational Science. CRC Press, first edition.

CHAPTER

THREE

CSEM/MT FORWARD MODELING & HIGH-ORDER EDGE FINITE ELEMENT METHOD

The last decade has been a period of rapid growth for electromagnetic methods (EM) in geophysics, mostly because of their industrial adoption. In particular, the 3D marine controlled-source electromagnetic (3D CSEM) method and 3D magnetotelluric (3D MT) method have become important techniques for reducing ambiguities in data interpretation in exploration geophysics. In order to be able to predict the EM signature of a given geological structure, modeling tools provide us with synthetic results which we can then compare to measured data. In particular, if the geology is structurally complex, one might need to use methods able to cope with such complexity in a natural way by means of, e.g., an unstructured mesh representing its geometry. Among the modeling methods for EM based upon 3D unstructured meshes, the high-order Nédélec Edge Finite Element Method (HEFEM) offers a good trade-off between accuracy and number of degrees of freedom, e.g. size of the problem. Furthermore, its divergence-free basis is very well suited for solving Maxwell's equation. On top of that, we choose to support tetrahedral meshes, as these are the easiest to use for very large domains or complex geometries.

We refer to the following papers for a complete discussion of marine/land 3-D CSEM/MT modelling and its problem statement within PETGEM:

- Castillo-Reyes, O., Queralt, P., Marcuello, A., Ledo, J. (2021). Land CSEM Simulations and Experimental Test Using Metallic Casing in a Geothermal Exploration Context: Vall'es Basin (NE Spain) Case Study. IEEE Transactions on Geoscience and Remote Sensing.
- Castillo-Reyes, O., de la Puente, J., García-Castillo, L. E., Cela, J.M. (2019). Parallel 3-D marine controlled-source electromagnetic modelling using high-order tetrahedral Nédélec elements. Geophysical Journal International, Volume 219, Issue 1, October 2019, Pages 39–65.
- Castillo-Reyes, O., de la Puente, Cela, J. M. PETGEM: A parallel code for 3D CSEM forward modeling using edge finite elements. Computers & Geosciences, vol 119: 123-136. ISSN 0098-3004. Elsevier.

CHAPTER

FOUR

INSTALLATION

This section describe the requirements and installation steps for PETGEM.

4.1 Requirements

PETGEM is known to run on various flavors of Linux clusters. Its requirements are:

- PETSc (builded version for **COMPLEX-VALUED NUMBERS**) for the use of direct/iterative parallel solvers
- Python 3 (versions 3.5.2, 3.6.3 and 3.6.9 have been tested)
- Numpy for arrays manipulation
- Scipy for numerical operations
- Singleton-decorator
- Sphinx and LaTeX (textlive) to generate documentation
- Petsc4py for parallel computations on distributed-memory platforms. It allows the use of parallel direct/iterative solvers from PETSc
- Mpi4py for parallel computations on distributed-memory platforms.
- h5py for input/output tasks.

On Linux, consult the package manager of your preference. PETGEM can be used without any installation by running the kernel from the top-level directory of the distribution.

4.2 Install PETGEM

- Following commands may require root privileges
- Download PETSc (PETSc 3.7, 3.8, 3.9, 3.12, 3.14, and 3.17 have been tested)
- Uncompress the PETSc archive (in this example, using PETSc 3.17.0):

```
$ tar zxvf petsc-3.17.0.tar.gz
```

- Configure and build PETSc. The configuration options depend on the calculations you want to perform (complex- or real-valued) as well as your compiler/MPI/Blas/Lapack setup. For PETGEM executions, **PETSC MUST BE BUILD FOR COMPLEX-VALUED NUMBERS**. In order to avoid incompatibilities between PETSC, petsc4py and PETGEM, we highly recommend the following configuration lines. Please, visit PETSc website for advanced configuration options. If you have a clean environment (not working MPI/Blas/Lapack), then run:

```
$ cd petsc-3.17.0
$ export PETSC_DIR=$PWD
$ export PETSC_ARCH=arch-linux2-c-debug
```

- If you do not want support for MUMPS, run following configure line:

```
$ ./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran  --download-mpich --
↪download-fblaslapack --with-scalar-type=complex
```

- If you want support for MUMPS, please add following options to previous configure line:

```
$ --download-mumps --download-scalapack --download-parmetis --download-metis --
↪download-ptscotch --download-cmake
```

- Further, to activate GPUs support, please add following options to previous configure line:

```
$ --with-cuda=1 --with_cuda_dir=PATH
```

  where `PATH` is the directory of your CUDA libraries.

- Then, build and install PETSc:

```
$ make $PETSC_DIR $PETSC_ARCH all
$ make $PETSC_DIR $PETSC_ARCH test
$ make $PETSC_DIR $PETSC_ARCH streams
```

- Ensure your `mpicc` compiler wrapper is on your search path:

```
$ export PATH="${PETSC_DIR}/${PETSC_ARCH}/bin:${PATH}"
```

- Ensure you have a Numpy installed:

```
$ pip3 install numpy
```

- And finally, install PETGEM with its dependencies (Scipy , Singleton-decorator, Sphinx, Petsc4py, Mpi4py, h5py) by typing:

```
$ pip3 install petgem
```

4.3 Downloading and building PETGEM

The PETGEM package is available for download at Python Package Index (PyPI), at GitHub, and the *Download* section of this project website.

- Configure and install PETSc (see *Install PETGEM* section)
- Ensure you have a Numpy installed:

```
$ pip3 install numpy
```

- Download PETGEM (PETGEM 1.0.0 have been tested)
- Uncompress the PETGEM archive:

```
$ tar zxvf petgem-1.0.0.tar.gz
$ cd petgem-1.0.0
```

- After unpacking the release tarball, the distribution is ready for building. Some environment configuration is needed to inform the PETSc location. As in *Install PETGEM* section, you can set the environment variables `PETSC_DIR` and `PETSC_ARCH` indicating where you have built/installed PETSc:

```
$ export PETSC_DIR=/usr/local/petsc
$ export PETSC_ARCH=arch-linux2-c-debug
```

- Alternatively, you can edit the file `setup.cfg` and provide the required information below `[config]` section:

```
[config]
petsc_dir = /usr/local/petsc
petsc_arch = arch-linux2-c-debug
```

- Build the distribution by typing:

```
$ python3 setup.py build
```

- After building, the distribution is ready for installation (this option may require root privileges):

```
$ python3 setup.py install
```

4.4 Build documentation

PETGEM is documented in PDF and HTML format using Sphinx and LaTeX. The documentation source is in the `doc/` directory. The following steps summarize how to generate PETGEM documentation.

- Move to the PETGEM doc directory:

```
$ cd doc
```

- Generate the PETGEM documentation in HTML format by typing:

```
$ make html
```

- Or, if you prefer the PDF format by typing:

```
$ make latexpdf
```

- The previous steps will build the documentation in the `doc/build` directory. Alternatively, you can modify this path by editing the file `setup.cfg` and provide the required information below `[build_sphinx]` section:

```
[build_sphinx]
source-dir = doc/source
build-dir  = usr/local/path-build
all_files  = 1
```

CHAPTER

FIVE

TUTORIAL

The PETGEM tutorial contains a small collection of programs which demonstrate main aspects of the PETGEM work-flow. Each example has the following structure:

1. `Readme.txt`. Short description about what the example does.
2. `params.yaml`. Physical parameters for the 3D CSEM/MT survey.
3. `petsc.opts`. Parameters for PETSc solvers.
4. Data for modeling (mesh, conductivity model and receivers list)

The data for simple cases modelling are freely available in the *Download* section.

5.1 Basic notions

The use of PETGEM can be summarized in three steps: pre-processing, modelling and post-processing. The pre-processing and modelling phases requires parameter files where the physical conditions of the 3D CSEM/MT model are defined. In the `params.yaml` the 3D CSEM survey is described using several keywords that allow one to define the main physical parameters and necessary file locations. In sake of simplicity and in order to avoid a specific parser, the `params.yaml` file is defined as Python dictionaries. Furthermore, the syntax of the `params.yaml` file is very simple yet powerful. As consequence, the dictionary names and his key names MUST NOT BE changed. See Preprocessing parameters file description and Modelling parameters file description in *Manual* section for a full explanation of those keywords.

For a general 3D CSEM/MT survey, the PETGEM work-flow can be summarize as follows:

1. Following the contents of the `params.yaml` file, a set of data are preprocessed (mesh, conductivity model and receivers positions) by the `kernel.py`
2. A problem instance is created
3. The problem sets up its domain, sub-domains, source, solver. This stage include the computation of the main data structures
4. Parallel assembling of $Ax = b$.
5. The solution is obtained in parallel by calling a `ksp()` PETSc object.
6. Interpolation of electromagnetic responses & post-processing parallel stage
7. Finally the solution can be stored by calling `postprocess()` method. Current version support hdf5.

Based on previous work-flow, any 3D CSEM/MT modelling requires the following input files:

1. A mesh file (current version supports Gmsh meshes)
2. A conductivity/resistivity model associated with the materials defined in the mesh file

3. A list of receivers positions in hdf5 format for the electromagnetic responses post-processing
4. A `params.yaml` file where are defined the 3D CSEM/MT parameters
5. A `petsc.opts` file where are defined options for PETSc solvers
6. A `kernel.py` script which manage the pre-processing and modelling tasks respectively

5.2 Running a simulation

This section introduces the basics of running PETGEM on the command line. Following command should be run in the top-level directory of the PETGEM source tree.

PETGEM kernel is invoked as follows:

```
$ mpirun -n MPI_tasks python3 kernel.py -options_file path/petsc.opts path/params.yaml
```

where `MPI_tasks` is the number of MPI parallel tasks, `kernel.py` is the script that manages the PETGEM workflow, `petsc.opts` is the PETSc options file and `params.yaml` is the modelling parameters file for PETGEM.

A template of this file is included in `examples/` of the PETGEM source tree. Additionally, a freely available copy of this file is located at *Download* section.

See *Model parameters file* section for more details about `params.yaml` file.

5.3 Model parameters file

By definition, any 3D CSEM/MT survey should include: physical parameters, a mesh file, source and receivers files. These data are included in the `params.yanl` file. Additionally, options for PETSc solvers are defined in a `petsc.opts` file.

In order to avoid a specific parser, `params.yaml` file is imported by PETGEM as a Python dictionary. As consequence, the dictionary name and his key names MUST NOT BE changed.

A glance of `params.yaml` file is the following:

```
###############################################################################
# PETGEM parameters file
###############################################################################
# Model parameters
model:
  mode: csem    # Modeling mode: csem or mt
  csem:
    sigma:
      #file: 'my_file.h5'                # Conductivity model file
      horizontal: [1., 0.01, 1., 3.3333]  # Horizontal conductivity
      vertical: [1., 0.01, 1., 3.3333]    # Vertical conductivity
    source:
      frequency: 2.                     # Frequency (Hz)
      position: [1750., 1750., -975.]   # Source position (xyz)
      azimuth: 0.                       # Source rotation in xy plane (in degrees)
      dip: 0.                           # Source rotation in xz plane (in degrees)
      current: 1.                       # Source current (Am)
      length: 1.                        # Source length  (m)
  mt:
    sigma:
```

(continues on next page)

(continued from previous page)

```
      file: 'my_file.h5'                  # Conductivity model file
      #horizontal: [1.e-10, 0.01]         # Horizontal conductivity
      #vertical: [1.e-10, 0.01]           # Vertical conductivity
    frequency: 4.                       # Frequency (Hz)
    polarization: 'xy'                  # Polarization mode for mt (xyz)

  # Common parameters for all models
  mesh: examples/case_CSEM/DIPOLE1D.msh   # Mesh file (gmsh format v2)
  receivers: examples/case_CSEM/receiver_pos.h5 # Receiver positions file (xyz)

# Execution parameters
run:
  nord: 1       # Vector basis order (1,2,3,4,5,6)
  cuda: False   # Cuda support (True or False)

# Output parameters
output:
  vtk: True                               # Postprocess vtk file (EM fields,␣
↪conductivity)
  directory: examples/case_CSEM/out           # Directory for output (results)
  directory_scratch: examples/case_CSEM/tmp   # Directory for temporal files
```

A template of this file is included in `examples/` of the PETGEM source tree. Additionally, a freely available copy of this file is located at *Download* section. Furthermore, in *Running a simulation* section of the PETGEM Manual is included a deep description about this file.

5.4 Visualization of results

Once a solution of a 3D CSEM/MT survey has been obtained, it should be post-processed by using a visualization program. PETGEM does not do the visualization by itself, but it generates output file (hdf5 format is supported) with the electromagnetic responses (Ex, Ey, Ez, Hx, Hy, Hz) for CSEM and the apparent resistivities and phases for MT. It also gives timing values in order to evaluate the performance.

CHAPTER

SIX

MANUAL

Table of contents

This manual provides reference documentation to PETGEM from a user's and developer's perspective.

6.1 How generate documentation

PETGEM is documented in PDF and HTML format using Sphinx and LaTeX. The documentation source is in the `doc/` directory. The following steps summarize how to generate PETGEM documentation.

- Move to the PETGEM doc directory:

```
$ cd doc
```

- Generate the PETGEM documentation in HTML format by typing:

```
$ make html
```

- Or, if you prefer the PDF format by typing:

```
$ make latexpdf
```

- The previous steps will build the documentation in the `doc/build` directory. Alternatively, you can modify this path by editing the file `setup.cfg` and provide the required information below `[build_sphinx]` section:

```
[build_sphinx]
source-dir = doc/source
build-dir  = usr/local/path-build
all_files  = 1
```

6.2 Coding style

PETGEM's coding style is based on PEP-0008 guidelines. Main guidelines are the following:

- 79 characteres per line.

- 4 spaces per indentation level.
- Variables are lower case meanwhile constants are upper case.
- Comments convention for functions is as follows:

```
def function(arg1, arg2):
    ''' This is a function.

        :param int arg1: array of dimensions ...
        :param str arg2: string that ...
    '''
```

- The use of inline comments is sparingly.
- Use of lowercase to name functions. Furthermore, functions names have following form: `<actionSubject>()`, e.g. `computeMatrix()`.
- Use of whole words instead abbreviations, examples:
  - Yes: `solveSystem()`, `computeEdges()`, `computeMatrix()`.
  - No: `solve()`, `compEdges()`, `compMatrix()`.

6.3 PETGEM design

PETGEM use a code structure for the high-order Nédelec FE algorithm that emphasizes good parallel scalability, which is crucial in the multi-core era. Furthermore, it's modularity should simplify the process of reaching the best possible performance in terms of percentage of the peak amount of floating point operations provided by the architecture. The code structure is modular, simple and flexible which allows exploiting not just PETGEM modules but also third party libraries. Therefore, the software stack includes interfaces to external suites of data structures and libraries that contain most of the necessary building blocks needed for programming large scale numerical applications, i.e. sparse matrices, vectors, iterative and direct solvers. As a result, the code is compact and flexible whose main UML diagrams are described as follows.

6.3.1 PETGEM directory structure

This subsection is dedicated to list and describe the PETGEM directory structure.

Table 1: Top directory structure

Name	Description
doc/	Source files for PETGEM documentation
examples/	Templates of basic scripts for 3D CSEM/MT modelling
petgem/	Source code
kernel.py	Heart of the code. This file contains the entire work-flow for a 3D CSEM/MT survey
DESCRIPTION.rst	Summary of PETGEM features, requirements and installation instructions
LICENSE.rst	License file
MANIFEST.in	File with exact list of files to include in PETGEM distribution
README.rst	Readme file
setup.py	Main PETGEM setup script, it is based on Python setup-module
setup.cfg	Setup configuration file for PETGEM setup script

The PETGEM source code is *petgem/*, which has the following contents:

Table 2: *petgem/* directory structure

Name	Description
common.py	Common classes and functions for init process.
hvfem.py	General modules and classes for describing a 3D CSEM/MT survey by using HEFEM ($p = 1, 2, 3, 4, 5, 6$) in tetrahedral meshes
mesh.py	Interface to import mesh files
parallel.py	Modules for supporting parallel computations and solution of 3D CSEM/MT
preprocessing.py	Modules for PETGEM pre-processing
solver.py	Parallel functions within PETGEM and interface to PETSc solvers

6.4 Running a simulation

This section introduces the basics of running PETGEM on the command line. In order to solve a 3D CSEM/MT survey, the PETGEM kernel requires two files, namely, the `petsc.opts` file and the `params.yaml` file, which are described below.

Options for PETSc solvers/preconditioners are accessible via the `petsc.opts` file, a glance of this is the following:

```
# Solver options for PETSc
-ksp_type gmres
-pc_type sor
-ksp_rtol 1e-8
-ksp_converged_reason
-ksp_monitor
```

Please, see PETSc documentation for more details about available options. A template of this file is included in `examples/` of the PETGEM source tree. Additionally, a freely available copy of this file is located at *Download* section.

On the other hand, any 3D CSEM/MT survey should include: order for Nédélec elements discretizations, physical parameters, a mesh file, source and receivers files. These data are included in the `params.yaml` file. In order to avoid a specific parser, `params.yaml` file is imported by PETGEM as a Python dictionary. As consequence, the dictionary name and his key names MUST NOT BE changed.

A glance of `params.yaml` file is the following:

```
###############################################################################
# PETGEM parameters file
###############################################################################
# Model parameters
model:
  mode: mt    # Modeling mode: csem or mt
  csem:
    sigma:
      #file: 'my_file.h5'                # Conductivity model file
      horizontal: [1.e-10, 0.01]        # Horizontal conductivity
      vertical: [1.e-10, 0.01]          # Vertical conductivity
    source:
      frequency: 2.                      # Frequency (Hz)
      position: [1750., 1750., -975.]    # Source position (xyz)
      azimuth: 0.                        # Source rotation in xy plane (in degrees)
      dip: 0.                            # Source rotation in xz plane (in degrees)
      current: 1.                        # Source current (Am)
      length: 1.                         # Source length  (m)
```

(continues on next page)

(continued from previous page)

```
  mt:
    sigma:
      #file: 'my_file.h5'                # Conductivity model file
      horizontal: [1.e-10, 0.01]       # Horizontal conductivity
      vertical: [1.e-10, 0.01]         # Vertical conductivity
    frequency: 2.                      # Frequency (Hz)
    polarization: 'xy'                 # Polarization mode for mt (xyz)

  # Common parameters for all models
  mesh: examples/case_MT/mesh_trapezoidal.msh   # Mesh file (gmsh format v2)
  receivers: examples/case_MT/receivers.h5 # Receiver positions file (xyz)

# Execution parameters
run:
  nord: 1        # Vector basis order (1,2,3,4,5,6)
  cuda: False    # Cuda support (True or False)

# Output parameters
output:
  vtk: False                             # Postprocess vtk file (EM fields,␣
↪conductivity)
  directory: examples/case_MT/out         # Directory for output (results)
  directory_scratch: examples/case_MT/tmp   # Directory for temporal files
```

The `params.yaml` file is divided into 3 sections, namely, model, run and output.

A template of `params.yaml` file is included in `examples/` of the PETGEM source tree. Additionally, a freely available copy of this file is located at *Download* section.

Once the `petsc.opts` file and the `params.yaml` file are ready, the PETGEM kernel is invoked as follows:

```
$ mpirun -n MPI_tasks python3 kernel.py -options_file path/petsc.opts path/params.yaml
```

where `MPI_tasks` is the number of MPI parallel tasks, `kernel.py` is the script that manages the PETGEM workflow, `petsc.opts` is the PETSc options file and `params.yaml` is the modelling parameters file for PETGEM.

PETGEM solves the problem and outputs the solution to the `output.directory/` path. The output file will be in h5py format. By default, the file structure contains a `fields` dataset and a `receiver_coordinates` dataset. In example, if the model has 3 receivers, `fields` dataset would contain six components (Ex, Ey, Ez, Hx, Hy, Hz) of the electromagnetic field responses as follows (`fields` only for illustrative purposes):

```
      Ex-component                Ey-component               Ez-component            ␣
↪   Hx-component                Hy-component               Hz-component
-4.5574e-13+7.1233e-13j       4.6739e-14+2.5366e-14j     -3.1475e-13+3.7062e-13j      -
↪4.5574e-13+7.1233e-13j   4.6739e-14+2.5366e-14j   -3.1475e-13+3.7062e-13j
-6.3279e-13+7.1644e-13j       9.0092e-14+5.6392e-14j     -5.3662e-13+3.7808e-13j      -
↪6.3279e-13+7.1644e-13j   9.0092e-14+5.6392e-14j   -5.3662e-13+3.7808e-13j
-1.0010e-12+7.0402e-13j       1.0689e-13+3.6703e-14j     -7.8662e-13+3.6350e-13j      -
↪1.0010e-12+7.0402e-13j   1.0689e-13+3.6703e-14j   -7.8662e-13+3.6350e-13j
```

It is worth to mention the fields will be separated by real and imaginary component.On the other hand, the receiver coordinates as follows (`receiver_coordinates`):

```
  X         Y                   Z
1750.          1750.    -990
1800.          1750.    -990
1850.          1750.    -990
```

In summary, for a general 3D CSEM/MT survey, the `kernel.py` script follows the next work-flow:

1. `kernel.py` reads a `params.yaml`
2. Following the contents of the `params.yaml`, a problem instance is created
3. Runs the data preprocessing
4. The problem sets up its domain, sub-domains, source-type, solver. This stage include the computation of the main data structures
5. Parallel assembling of $Ax = b$.
6. The solution is obtained in parallel by calling a `ksp()` PETSc object.
7. Interpolation of electromagnetic (electric and magnetic) responses and post-processing parallel stage
8. Finally the solution can be stored by calling `solver.postprocess()` method. Current version support h5py format.

Based on previous work-flow, any 3D CSEM/MT modelling requires the following input files:

1. A mesh file (current version supports mesh files from Gmsh)
2. A conductivity model associated with the materials defined in the mesh file
3. A list of receivers positions in hdf5 format for the electric responses post-processing
4. A `params.yaml` file where are defined the 3D CSEM/MT parameters
5. A `petsc.opts` file where are defined options for PETSc solvers

During the preprocessing stage, the `kernel.py` generates the mesh file (Gmsh format) with its conductivity model and receivers positions in PETSc binary files.

For this phase, the supported/expected formats are the following:

- `basis_order`: nédélec basis element order for the discretization. Supported orders are $p = 1, 2, 3, 4, 5, 6$
- `mesh_file`: path to tetrahedral mesh file in format from Gmsh
- `sigma`: numpy array or hdf5 file where each value represents the horizontal and vertical conductivity for each material defined in `mesh_file`. Therefore, size of `sigma` must match with the number of materials in `mesh_file`
- `receivers_file`: floating point values giving the X, Y and Z coordinates of each receiver in hdf5 format. Therefore, the number of receivers in the modelling will defined the number of the rows in `receivers_file`, i.e. if the modelling has 3 receivers, `receivers_file` should be:

```
5.8333300e+01   1.7500000e+03  -9.9000000e+02
1.1666670e+02   1.7500000e+03  -9.9000000e+02
1.7500000e+02   1.7500000e+03  -9.9000000e+02
```

- `output_directory_scratch`: output scratch directory for pre-processing task

Once the previous information has been provided, the `kernel.py` script will generate all the input files required by PETGEM in order to solve a 3D CSEM/MT survey. The `kernel.py` output files list is the following:

- `nodes.dat`: floating point values giving the X, Y and Z coordinates of the four nodes that make up each tetrahedral element in the mesh. The dimensions of `nodes.dat` are given by (number-of-elements, 12)
- `meshConnectivity.dat`: list of the node number of the n-th tetrahedral element in the mesh. The dimensions of `meshConnectivity.dat` are given by (number-of-elements, 4)
- `dofs.dat`: list of degrees of freedom of the n-th tetrahedral element in the mesh. Since PETGEM is based on high-order Nédelec FE, the dimensions of `dofs.dat` are given by (number-of-elements, 6) for $p = 1$, by

(number-of-elements, 20) for $p = 2$, by (number-of-elements, 45) for $p = 3$, by (number-of-elements, 84) for $p = 4$, by (number-of-elements, 140) for $p = 5$, and by (number-of-elements, 216) for $p = 6$.

- `edges.dat`: list of the edges number of the n-th tetrahedral element in the mesh. The dimensions of `edges.dat` are given by (number-of-elements, 6)
- `faces.dat`: list of the faces number of the n-th tetrahedral element in the mesh. The dimensions of `faces.dat` are given by (number-of-elements, 4)
- `edgesNodes.dat`: list of the two nodes that make up each edge in the mesh. The dimensions of `edgesNodes.dat` are given by (number-of-edges, 2)
- `faces.dat`: list of the four faces that make up each tetrahedral element in the mesh. The dimensions of `faces.dat` are given by (number-of-elements, 4)
- `facesNodes.dat`: list of the three nodes that make up each tetrahedral face in the mesh. The dimensions of `facesNodes.dat` are given by (number-of-faces, 3)
- `boundaries.dat`: list of the DOFs indexes that belongs on domain boundaries. The dimensions of `boundaries.dat` are given by (number-of-boundaries, 1)
- `conductivityModel.dat`: floating point values giving the conductivity to which the n-th tetrahedral element belongs. The dimensions of `conductivityModel.dat` are given by (number-of-elements, 1)
- `receivers.dat`: floating point values giving the X, Y and Z coordinates of the receivers. Additionally, this file includes information about the tetrahedral element that contains the n-th receiver, namely, its X, Y and Z coordinates of the four nodes, its list of the node number and its list of DOFs.
- `nnz.dat`: list containing the number of nonzeros in the various matrix rows, namely, the sparsity pattern. According to the PETSc documentation, using the `nnz.dat` list to preallocate memory is especially important for efficient matrix assembly if the number of nonzeros varies considerably among the rows. The dimensions of `nnz.dat` are given by (number-of-DOFs, 1)

For each `*.dat` output file, the `kernel.py` script will generate `*.info` files which are PETSc control information.

For post-processing stage (interpolation of electric responses at receivers positions) a point location function is needed. For this task, PETGEM is based on `find_simplex` function by Scipy, which find the tetrahedral elements (simplices) containing the given receivers points. If some receivers positions are not found, PETGEM will print its indexes and will save only those located points in the `receivers.dat` file.

6.5 Mesh formats

Current PETGEM version supports mesh files from Gmsh. Aforementioned format must be pre-processed using the `run_preprocessing.py` script. The `run_preprocessing.py` script is included in the top-level directory of the PETGEM source tree.

6.6 Available solvers

This section describes solvers available in PETGEM from an user's perspective. Direct as well as iterative solvers and preconditioners are supported through an interface to PETSc library.

PETGEM uses petsc4py package in order to support the Krylov Subspace Package (KSP) from PETSc. The object KSP provides an easy-to-use interface to the combination of a parallel Krylov iterative method and a preconditioner (PC) or a sequential/parallel direct solver. As result, PETGEM users can set various solver options and preconditioner options at runtime via the PETSc options database. These parameters are defined in the `petsc.opts` file.

6.7 Simulations in parallel

In FEM or HEFEM simulations, the need for efficient algorithms for assembling the matrices may be crucial, especially when the DOFs is considerably large. This is the case for realistic scenarios of 3D CSEM surveys because the required accuracy. In such situation, assembly process remains a critical portion of the code optimization since solution of linear systems which asymptotically dominates in large-scale computing, could be done with linear solvers such as PETSc, MUMPs, PARDISO). In fact, in PETGEM V0.6, the system assembly takes around $15%$ of the total time.

The classical assembly in FEM or HEFEM programming is based on a loop over the elements. Different techniques and algorithms have been proposed and nowadays is possible performing these computations at the same time, i.e., to compute them in parallel. However, parallel programming is not a trivial task in most programming languages, and demands a strong theoretical knowledge about the hardware architecture. Fortunately, Python presents user friendly solutions for parallel computations, namely, mpi4py and petsc4py . These open-source packages provides bindings of the MPI standard and the PETSc library for the Python programming language, allowing any Python code to exploit multiple processors architectures.

On top of that, *Figure 7.4* depicts shows an upper view of the matrix assembly and solution using the mpi4py and petsc4py package in PETGEM. The first step is to partition the work-load into subdomains. This task can be done by PETSc library, which makes load over processes balanced. After domain partition, subdomains are assigned to MPI tasks and the elemental matrices are calculated concurrently. These local contributions are then accumulated into the global matrix system. The process for global vector assembly is similar.

Subsequently, the system is ready to be solved. PETGEM uses the Krylov Subspace Package (KSP) from PETSc through the petsc4py package. The KSP object provides an easy-to-use interface to the combination of a parallel Krylov iterative method and a preconditioner (PC) or a sequential/parallel direct solver. As result, PETGEM users can set various solver options and preconditioner options at runtime via the PETSc options database. Since PETSc knows which portions of the matrix and vectors are locally owned by each processor, the post-processing task is also completed in parallel.

All petsc4py classes and methods are called from the PETGEM kernel in a manner that allows a parallel matrix and parallel vectors to be created automatically when the code is run on many processors. Similarly, if only one processor is specified the code will run in a sequential mode. Although petsc4py allows control the way in which the matrices and vectors to be split across the processors on the architecture, PETGEM simply let petsc4py decide the local sizes in sake of computational flexibility. However, this can be modified in an easy way without any extra coding required.

6.8 Visualization of results

Once a solution of a 3D CSEM/MT survey has been obtained, it should be post-processed by using a visualization program. PETGEM does not do the visualization by itself, but it generates output files (hdf5 and VTK are supported) with the electric responses at receivers positions and nodes in the mesh. It also gives timing values in order to evaluate the performance.

Figure 7.5 shows an example of PETGEM output for the first modelling case (Canonical model of an off-shore hydrocarbon reservoir) described in *CSEM example* section. *Figure 7.5* was obtained using Paraview.

6.9 CSEM example

This section includes a step-by-step walk-through of the process to solve a simple 3D CSEM survey. The typical process to solve a problem using PETGEM is followed: a model is meshed, PETGEM input files are preprocessed and modeled by using `kernel.py` script to solve the modelling and finally the results of the simulation are visualised.

All necessary data to reproduce the following examples are available in the *Download* section.

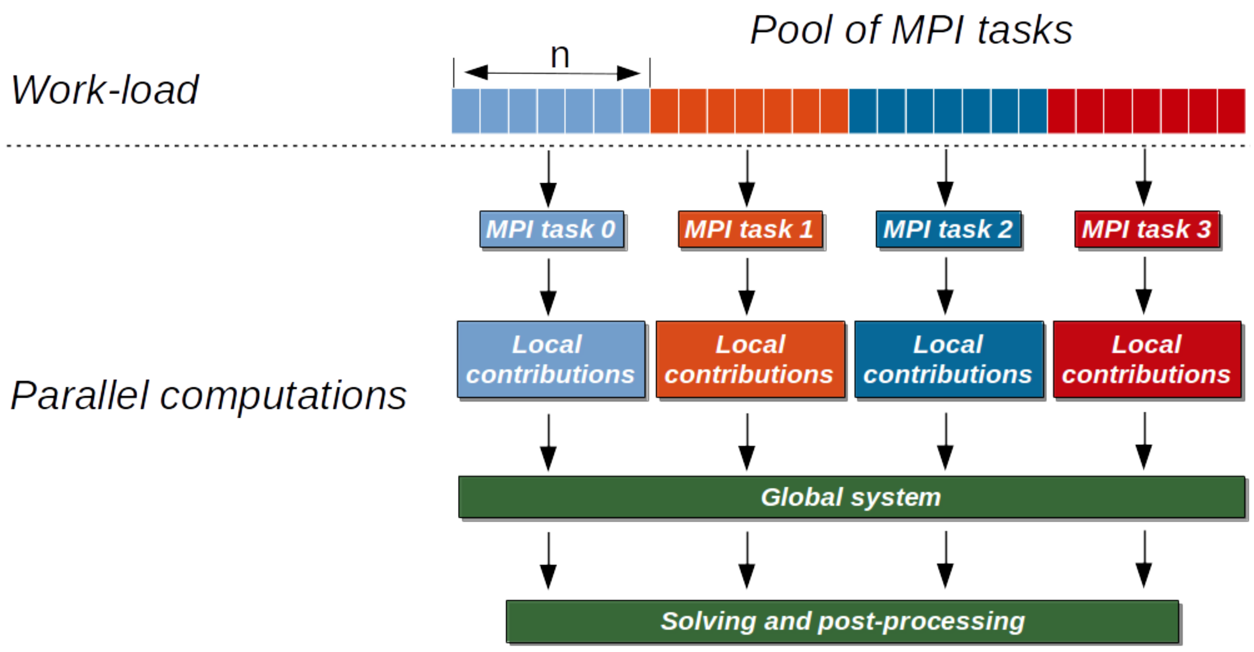

Fig. 1: Figure 7.4. Parallel scheme for assembly and solution in PETGEM using 4 MPI tasks. Here the elemental matrices computation is done in parallel. After calculations the global system is built and solved in parallel using the petsc4py and mpi4py packages.

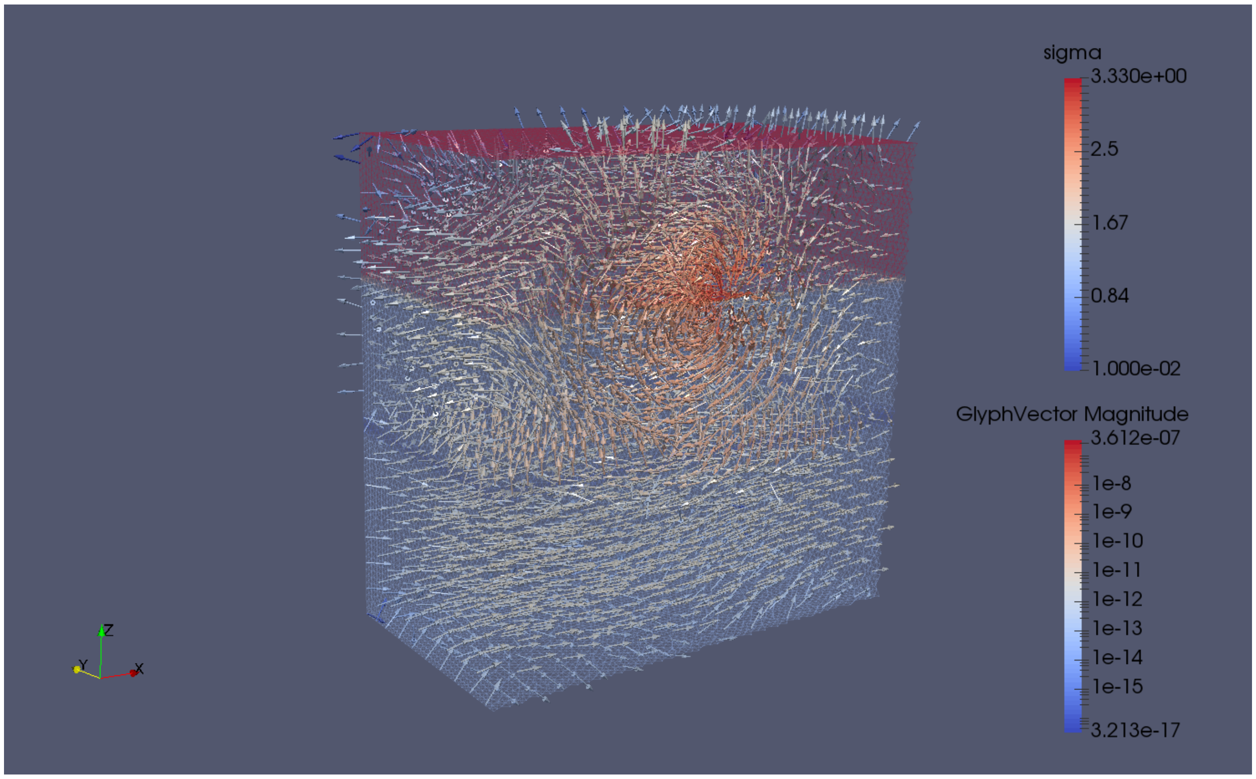

Fig. 2: Figure 7.5. PETGEM vtk output.

6.9.1 Example 1: Canonical model of an off-shore hydrocarbon reservoir

Model

In order to explain the 3D CSEM modelling using PETGEM, here is considered the canonical model by Weiss2006 which consists in four-layers: 1000 m thick seawater (3.3 S/m), 1000 m thick sediments (1 S/m), 100 m thick oil (0.01 S/m) and 1400 m thick sediments (1 S/m). The computational domain is a $[0, 3500]^3$ m cube. For this model, a 2 Hz x-directed dipole source is located at $z = 975$, $x = 1750$, $y = 1750$. The receivers are placed in-line to the source position and along its orientation, directly above the seafloor ($z = 990$). Further, the Nédélec element order is p=1 (first order)

Meshing

PETGEM V1.0 is based on tetrahedral meshes. Therefore, *Figure 7.8* shows a 3D view of the model with its unstructured tetrahedral mesh for the halfspace $y > 1750$, with the color scale representing the electrical conductivity σ for each layer. Mesh in *Figure 7.8* have been obtained using Gmsh.

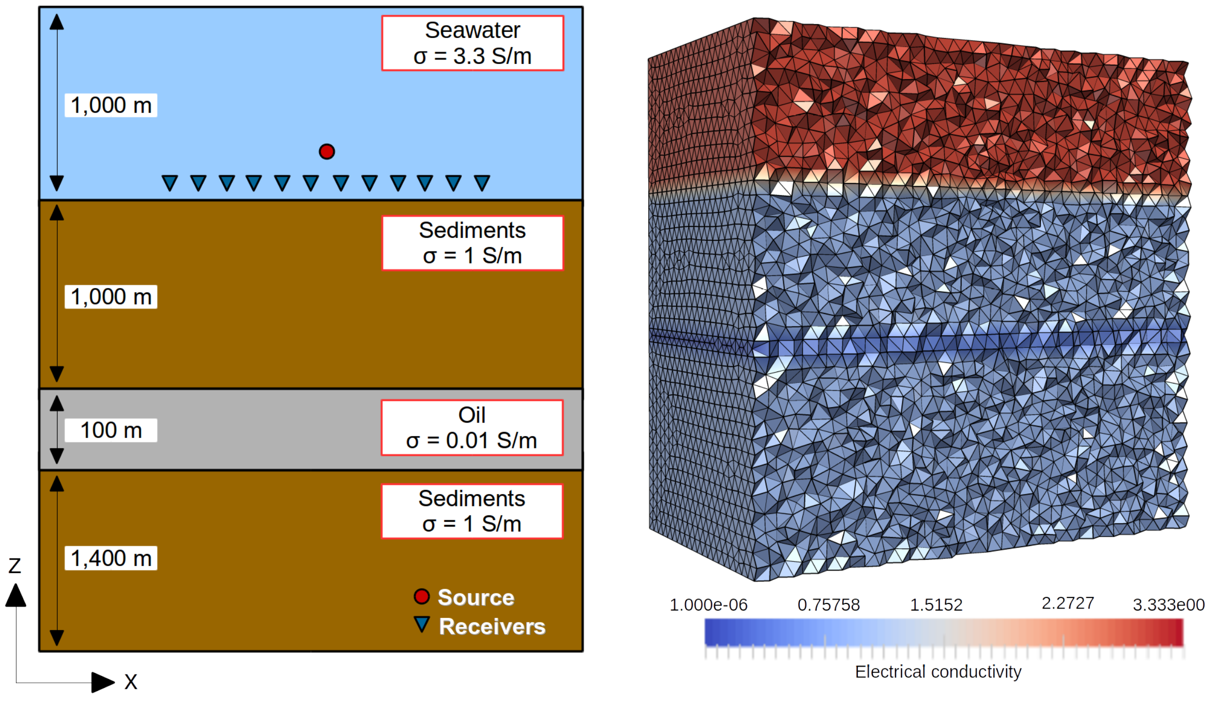

Fig. 3: Figure 7.8. In-line canonical off-shore hydrocarbon model with its unstructured tetrahedral mesh for $y > 1750$. The color scale represents the electrical conductivity σ for each layer.

In this case, the mesh is composed by four conductivity materials. Please, see Gmsh manual for details about the mesh creation process.

Parameters file for PETGEM

The parameters file used for this example follows:

```
###############################################################################
# PETGEM parameters file
###############################################################################
# Model parameters
model:
  mode: csem    # Modeling mode: csem or mt
  csem:
    sigma:
      #file: 'my_file.h5'                # Conductivity model file
      horizontal: [1., 0.01, 1., 3.3333]  # Horizontal conductivity
      vertical: [1., 0.01, 1., 3.3333]    # Vertical conductivity
    source:
      frequency: 2.                      # Frequency (Hz)
      position: [1750., 1750., -975.]    # Source position (xyz)
      azimuth: 0.                        # Source rotation in xy plane (in degrees)
      dip: 0.                            # Source rotation in xz plane (in degrees)
      current: 1.                        # Source current (Am)
      length: 1.                         # Source length  (m)

  # Common parameters for all models
  mesh: examples/case_1/DIPOLE1D.msh   # Mesh file (gmsh format v2)
  receivers: examples/case_1/receiver_pos.h5 # Receiver positions file (xyz)

# Execution parameters
run:
  nord: 1       # Vector basis order (1,2,3,4,5,6)
  cuda: False   # Cuda support (True or False)

# Output parameters
output:
  vtk: True                             # Postprocess vtk file (EM fields,␣
↪conductivity)
  directory: examples/case_1/out          # Directory for output (results)
  directory_scratch: examples/case_1/tmp  # Directory for temporal files
```

Note that you may wish to change the location of the input files to somewhere on your drive. The solver options have been configured in the `petsc.opts` file as follows:

```
# Solver options for
-ksp_type gmres
-pc_type sor
-ksp_rtol 1e-8
-ksp_converged_reason
-log_summary
```

That's it, we are now ready to solve the modelling.

Running PETGEM

To run the simulation, the following command should be run in the top-level directory of the PETGEM source tree:

```
$ mpirun -n 16 python3 kernel.py -options_file case1/petsc.opts case1/params.yaml
```

`kernel.py` solves the problem as follows:

1. Problem initialization
2. Preprocessing data

3. Import files
4. Parallel assembly system
5. Parallel solution system
6. Post-processing of electric responses

and outputs the solution to the output path (`case1/out/`). The output file will be in hdf5 format.

PETGEM post-processing

Once a solution of a 3D CSEM survey has been obtained, it should be post-processed by using a visualization program. PETGEM does not do the visualization by itself, but it generates output file (hdf5) with the electric responses and receivers positions. It also gives timing values in order to evaluate the performance.

The electric fields responses can be handled freely and plotted. The dimension of the array is determined by the number of receivers in the modelling (58 in this example). *Figure 7.9* shows a comparison of the x-component of total electric field between PETGEM results and the quasi-analytical solution obtained with the DIPOLE1D tool. The total electric field in *Figure 7.9* was calculated using a mesh with $\approx$ 2 millions of DOFs.

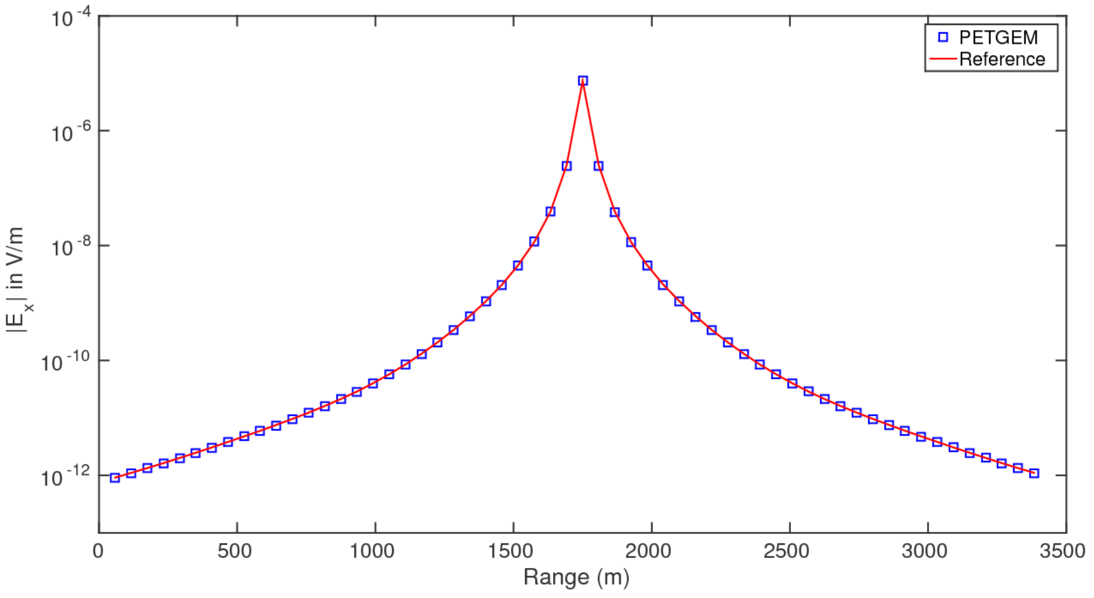

Fig. 4: Figure 7.9. Total electric field comparative for x-component between PETGEM V1.0 and DIPOLE1D.

6.9.2 Example 2: Use of high-order Nédélec elements

In order to show the potential of high-order Nédélec elements, here is considered the same model of previous example. In this case, the Nédélec element order is p=2 (second order).

Meshing

As in previous example, the mesh is composed by four conductivity materials. The resulting Gmsh script can be found in *Download* section. It is worth to mention that the use a Nédélec element order p=2 helps to reduce the number of tetrahedral elements to achieved a certain error level.

Parameters file for PETGEM

The parameters file used for this example follows:

```
###############################################################################
# PETGEM parameters file
###############################################################################
# Model parameters
model:
  mode: csem    # Modeling mode: csem or mt
  csem:
    sigma:
      #file: 'my_file.h5'                # Conductivity model file
      horizontal: [1., 0.01, 1., 3.3333]  # Horizontal conductivity
      vertical: [1., 0.01, 1., 3.3333]    # Vertical conductivity
    source:
      frequency: 2.                      # Frequency (Hz)
      position: [1750., 1750., -975.]    # Source position (xyz)
      azimuth: 0.                        # Source rotation in xy plane (in degrees)
      dip: 0.                            # Source rotation in xz plane (in degrees)
      current: 1.                        # Source current (Am)
      length: 1.                         # Source length  (m)
  mt:
    sigma:
      file: 'my_file.h5'                 # Conductivity model file
      #horizontal: [1.e-10, 0.01]        # Horizontal conductivity
      #vertical: [1.e-10, 0.01]          # Vertical conductivity
    frequency: 4.                        # Frequency (Hz)
    polarization: 'xy'                   # Polarization mode for mt (xyz)

  # Common parameters for all models
  mesh: examples/case_2/DIPOLE1D.msh    # Mesh file (gmsh format v2)
  receivers: examples/case_2/receiver_pos.h5 # Receiver positions file (xyz)

# Execution parameters
run:
  nord: 2       # Vector basis order (1,2,3,4,5,6)
  cuda: False   # Cuda support (True or False)

# Output parameters
output:
  vtk: True                               # Postprocess vtk file (EM fields,␣
↪conductivity)
  directory: examples/case_2/out           # Directory for output (results)
  directory_scratch: examples/case_2/tmp   # Directory for temporal files
```

Running PETGEM

To run the simulation, the following command should be run in the top-level directory of the PETGEM source tree:

```
$ mpirun -n 48 python3 kernel.py -options_file case2/petsc.opts case2/params.yaml
```

`kernel.py` solves the problem as follows:

1. Problem initialization
2. Preprocessing data

3. Import files
4. Parallel assembly system
5. Parallel solution system
6. Post-processing of electric responses

and outputs the solution to the output path (`case2/out`). The output files will be in hdf5 format.

Nédélec order comparison

Figure 7.10 shows a comparison of the x-component of total electric field between p=1 and p=2 against the quasi-analytical solution obtained with the DIPOLE1D tool. The total electric field in *Figure 7.10* was calculated using a mesh with ≈ 2 millions of DOFs for p=1 and a mesh with ≈ 600 thousands of DOFs.

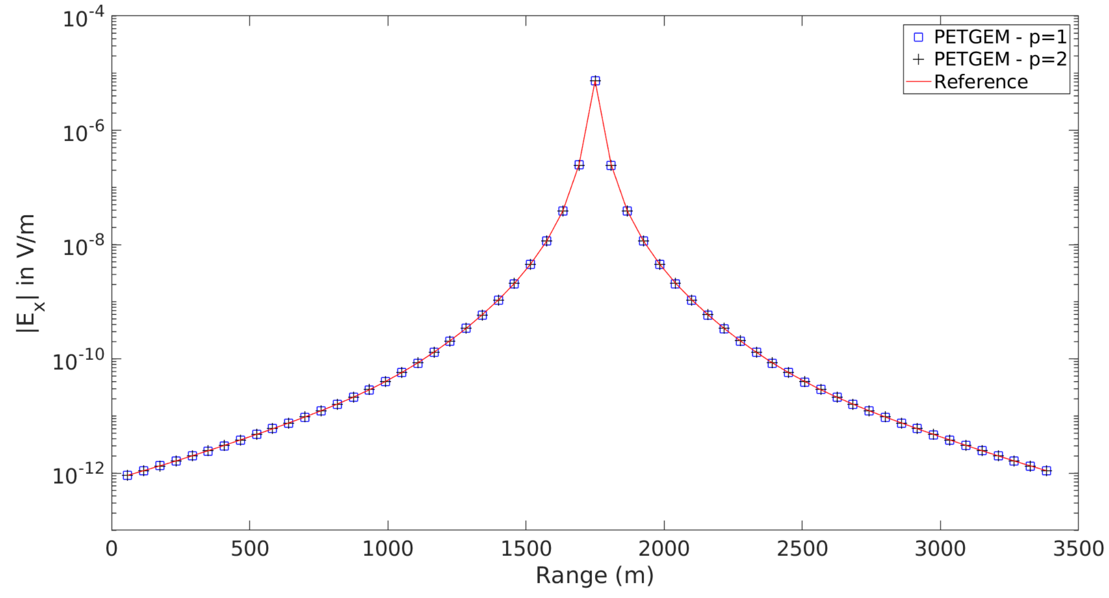

Fig. 5: Figure 7.10. Nédélec order comparative within PETGEM V1.0

6.9.3 Example 3: Use of PETSc solvers

In PETGEM, options for PETSc solvers/preconditioners are accessible via the `petsc.opts` file. Here is presented a short list of relevant configuration options for the solution of the problem under consideration.

- Direct solver by MUMPs via PETSc interface

```
# Solver options for PETSc
-pc_type lu
-pc_factor_mat_solver_package mumps
```

- Generalized minimal residual (GMRES) solver with a successive over-relaxation method (SOR) as preconditioner. Further, these options prints useful information: True residual norm, the preconditioned residual norm at each iteration and reason a iterative method was said to have converged or diverged.

```
# Solver options for PETSc
-ksp_type gmres
-pc_type sor
-ksp_rtol 1e-8
-ksp_monitor_true_residual
-ksp_converged_reason
```

- Generalized minimal residual (GMRES) solver with a geometric algebraic multigrid method (GAMG) as preconditioner.

```
# Solver options for PETSc
-ksp_type gmres
-pc_type gamg
-ksp_rtol 1e-8
```

- Generalized minimal residual (GMRES) solver with an additive Schwarz method (ASM) as preconditioner.

```
# Solver options for PETSc
-ksp_type gmres
-pc_type asm
-sub_pc_type lu
-ksp_rtol 1e-8
```

- Generalized minimal residual (GMRES) solver with a Jacobi method as preconditioner.

```
# Solver options for PETSc
-ksp_type gmres
-pc_type bjacobi
-ksp_rtol 1e-8
```

- Transpose free quasi-minimal residual (TFQMR) solver with a successive over-relaxation method (SOR) as preconditioner.

```
# Solver options for PETSc
-ksp_type tfqmr
-pc_type sor
-ksp_rtol 1e-8
```

Previous configuration are based on most used in the literature.

6.10 MT example

This section includes a step-by-step walk-through of the process to solve a simple 3D MT survey. The typical process to solve a problem using PETGEM is followed: a model is meshed, PETGEM input files are preprocessed and modeled by using `kernel.py` script to solve the modelling and finally the results of the simulation are visualised.

All necessary data to reproduce the following examples are available in the *Download* section.

6.10.1 Example 4: 3D trapezoidal hill model

Model

In order to explain the 3D MT modelling using PETGEM, here is considered a 3D trapezoidal hill model. This test case is a homogeneous half-space model with 1/100 S/m as host resistivity and 1e-9 S/m as the free-space resistivity.

The trapezoidal hill, centered at the computational domain, has a height of 0.45km with a hill-top square of 0.45km X 0.45km, and a hill-bottom square of 2km X 2km. A cross-section view of the model under consideration is depicted in *Figure 7.11*. By using the horizontal components of the EM fields at 2Hz, the apparent resistivities are computed. 41 sites at 0km and along the x-axis are arranged equidistant spacing over the interval x=[-2:2]km. Further, the Nédélec element order is p=2 (second order).

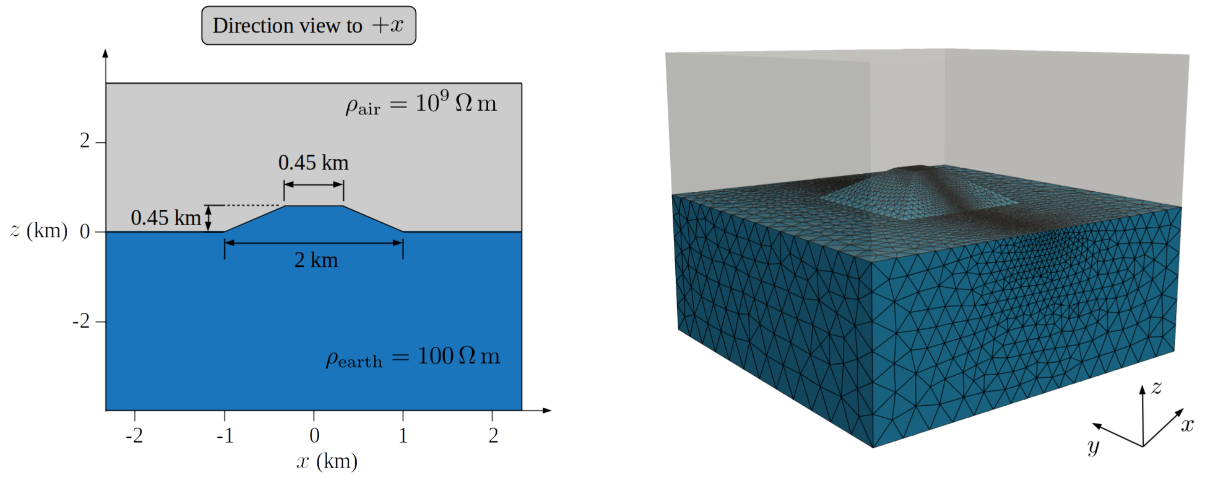
Direction view to $+x$
$\rho_{\text{air}} = 10^9\,\Omega\,\text{m}$
0.45 km
0.45 km
2 km
z (km)
$\rho_{\text{earth}} = 100\,\Omega\,\text{m}$
x (km)
z
x
y

Fig. 6: Figure 7.11. Cross-section view of the 3D trapezoidal hill model (left-panel) with its resulting computational unstructured and hp-adapted mesh for p = 2 (right-panel).

Parameters file for PETGEM

The parameters file used for this example follows:

```
###############################################################################
# PETGEM parameters file
###############################################################################
# Model parameters
model:
  mode: mt     # Modeling mode: csem or mt
  mt:
    sigma:
      #file: 'my_file.h5'                 # Conductivity model file
      horizontal: [1.e-10, 0.01]        # Horizontal conductivity
      vertical: [1.e-10, 0.01]          # Vertical conductivity
    frequency: 2.                       # Frequency (Hz)
    polarization: 'xy'                  # Polarization mode for mt (xyz)

  # Common parameters for all models
  mesh: examples/case_4/mesh_trapezoidal.msh    # Mesh file (gmsh format v2)
  receivers: examples/case_4/receivers.h5 # Receiver positions file (xyz)

# Execution parameters
run:
  nord: 2        # Vector basis order (1,2,3,4,5,6)
  cuda: False    # Cuda support (True or False)

# Output parameters
output:
```

(continues on next page)

(continued from previous page)

```
  vtk: False                               # Postprocess vtk file (EM fields,␣
↪conductivity)
  directory: examples/case_4/out           # Directory for output (results)
  directory_scratch: examples/case_4/tmp   # Directory for temporal files
```

Note that you may wish to change the location of the input files to somewhere on your drive. The solver options have been configured in the `petsc.opts` file as follows:

```
# Solver options for
-pc_type lu
-pc_factor_mat_solver_package mumps
```

That's it, we are now ready to solve the modelling.

Running PETGEM

To run the simulation, the following command should be run in the top-level directory of the PETGEM source tree:

```
$ mpirun -n 16 python3 kernel.py -options_file case4/petsc.opts case4/params.yaml
```

`kernel.py` solves the problem as follows:

1. Problem initialization
2. Preprocessing data
3. Import files
4. Parallel assembly system (twice, 1 for each polarization mode)
5. Parallel solution system (twice, 1 for each polarization mode)
6. Post-processing of electric responses, apparent resistivities and phases

and outputs the solution to the output path (`case4/out/`). The output file will be in hdf5 format.

PETGEM post-processing

Once a solution of a 3D MT survey has been obtained, it should be post-processed by using a visualization program. PETGEM does not do the visualization by itself, but it generates output file (hdf5) with the electric responses (apparent resistivities and phases) and receivers positions. It also gives timing values in order to evaluate the performance.

The electric fields responses can be handled freely and plotted. The dimension of the array is determined by the number of receivers in the modelling (41 in this example). *Figure 7.12* shows a comparison of the obtained apparent resistivity and phase.

6.11 Code documentation

Following sub-sections are dedicated to code documentation of PETGEM.

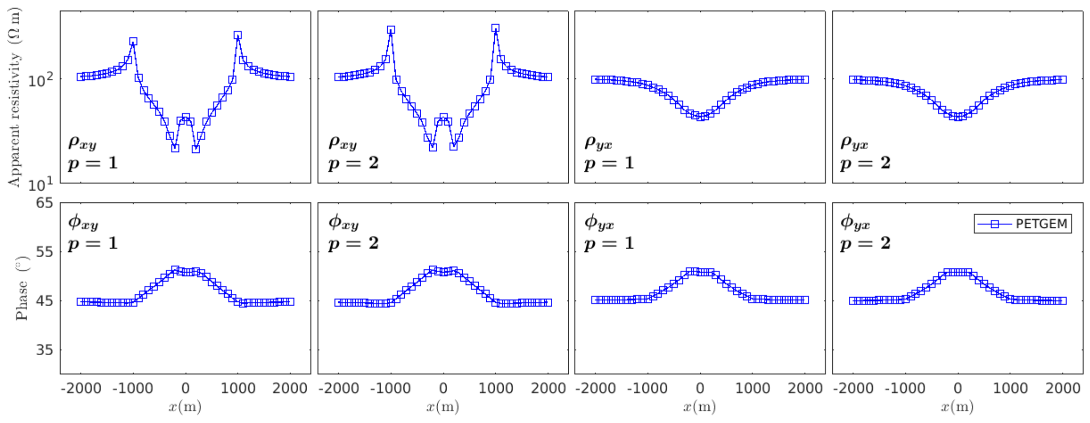

Fig. 7: Figure 7.12. Apparent resistivities (left-panel) and phases (right-panel) for the 3D trapezoidal hill model. The PETGEM solutions were calculated with p = 2.

6.11.1 kernel

kernel.py

PETGEM kernel for 3D CSEM/MT forward modelling using high order vector elements.

6.11.2 Common scripts

common.py

Define common operations for **PETGEM**.

class petgem.common.**InputParameters**(*params*, *parEnv*)
Method to import a yaml parameter file.

Parameters **object** (*dict*) – user params yaml file.

Returns user parameters as object view.

Return type object.

class petgem.common.**Print**(*text*, *color_code=None*)
This class provides methods for pretty print.

Parameters **str** (*object*) – string to be printed.

Returns None.

Return type None.

classmethod header()
Print the header.

Param None.

Returns None.

Return type None.

classmethod master(*text*, *color_code=None*)
If the caller is the master process, this method prints a message.

Param None.

Returns None.

Return type None.

class `petgem.common.`**`Timer`**(*elapsed=0*)
Definition of timer class.

`elapsed = None`

`reset`()
Reset timer.

`start`()
Start timer.

`stop`()
Stop timer.

`petgem.common.`**`measure_all_class_methods`**(*Cls*)
"Implement a decorator to measure execution time for each method.

Args: f: the decorated function

Returns: a function wrap

`petgem.common.`**`measure_time`**(*f=None*, *group=None*, *split=False*)
"Implement method to measure execution time.

Args: f: the decorated function group: the group name split: decides if all blocks in a group contribute to the same timer

Returns: a function wrap

`petgem.common.`**`unitary_test`**()
Unitary test for common.py script.

6.11.3 Pre-processing

preprocessing.py

Define data preprocessing operations for **PETGEM**.

`petgem.preprocessing.`**`unitary_test`**()
Unitary test for preprocessing.py script.

6.11.4 Mesh scripts

mesh.py

Define functions for mesh handling.

`petgem.mesh.`**`computeBoundaries`**(*dof_connectivity*, *dof_edges*, *dof_faces*, *bEdges*, *bFaces*, *Nord*)
Compute the indexes of dofs boundaries and internal dofs.

Parameters

- **`dof_connectivity`** (*`ndarray`*) – local/global dofs list for elements
- **`dof_edges`** (*`ndarray`*) – dofs index on edges

- **dof_faces** (*ndarray*) – dofs index on faces
- **bEdges** (*ndarray*) – boundary-edges connectivity with dimensions = (number_boundary_edges,1)
- **bfaces** (*ndarray*) – indexes of boundary-faces = (number_boundary_faces, 1)
- **Nord** (*int*) – polynomial order of nedelec basis functions

Returns indexes of internal dofs and indexes of boundary dofs

Return type ndarray

petgem.mesh.**computeBoundaryEdges**(*edgesN*, *bfacesN*)
Compute boundary edges of a tetrahedral mesh.

Parameters

- **edgesN** (*ndarray*) – edges-nodes connectivity.
- **bfacesN** (*ndarray*) – boundary-faces-nodes connectivity.

Returns boundary-edges connectivity.

Return type ndarray

petgem.mesh.**computeBoundaryElements**(*elemsF*, *bFaces*, *nFaces*)
Compute boundary elements.

Parameters

- **elemsF** (*ndarray*) – elements-faces connectivity.
- **bfaces** (*ndarray*) – indexes of boundary faces.
- **nFaces** (*int*) – number of faces in the mesh.

Returns indexes of boundary elements.

Return type ndarray

petgem.mesh.**computeBoundaryFaces**(*elemsF*, *facesN*)
Compute boundary faces of a tetrahedral mesh.

Parameters

- **elemsF** (*ndarray*) – elements-face connectivity.
- **facesN** (*ndarray*) – faces-nodes connectivity.

Returns nodal-connectivity and indexes of boundary-faces, number of boundary faces.

Return type ndarray

petgem.mesh.**computeEdges**(*elemsN*, *nElems*)
Compute edges of a 3D tetrahedral mesh.

Parameters

- **elemsN** (*ndarray*) – elements-nodes connectivity.
- **nElems** (*int*) – number of tetrahedral elements in the mesh.

Returns edges connectivity and edgesNodes connectivity.

Return type ndarray

petgem.mesh.**computeFacePlane**(*nodes*, *bFaces*, *bFacesN*)
Compute the plane id to which the boundary belongs.

Parameters

- **nodes** (*ndarray*) – nodes coordinates.
- **bFaces** (*ndarray*) – indexes of boundary-faces.
- **bFacesN** (*ndarray*) – boundary-faces-nodes connectivity.

Returns plane id to which the boundary belongs.

Return type ndarray

petgem.mesh.**computeFaces**(*elemsN*, *nElems*)

Compute the element's faces of a 3D tetrahedral mesh.

Parameters

- **matrix** (*ndarray*) – elements-faces connectivity.
- **nElems** (*int*) – number of elements in the mesh.

Returns element/faces connectivity.

Return type ndarray

Note: References:n Rognes, Marie E., Robert Cndarray. Kirby, and Anders Logg. "Efficient assembly of H(div) and H(curl) conforming finite elements." SIAM Journal on Scientific Computing 31.6 (2009): 4130-4151.

petgem.mesh.**computeFacesEdges**(*elemsF*, *elemsE*, *nFaces*, *nElems*)

Compute the edge's faces connectivity

Parameters

- **elemsF** (*ndarray*) – elements-faces connectivity.
- **elemsE** (*ndarray*) – elements-edges connectivity.
- **nFaces** (*int*) – number of faces in the mesh.
- **nElems** (*int*) – number of elements in the mesh.

Returns edges/faces connectivity.

Return type ndarray.

petgem.mesh.**unitary_test**()

Unitary test for mesh.py script.

6.11.5 HEFEM scripts

hvfem.py

Define functions for high-order vector finite element method.

petgem.hvfem.**AffineTetrahedron**(*X*)

Compute affine coordinates and their gradients.

Parameters **X** (*ndarray*) – point coordinates

Returns affine coordinates and gradients of affine coordinates

Return type ndarray

Note: References:n Fuentes, F., Keith, B., Demkowicz, L., & Nagaraj, S. (2015). Orientation embedded high order shape functions for the exact sequence elements of all shapes. Computers & Mathematics with applications, 70(4), 353-458.

`petgem.hvfem.`**`AncEE`**(*S*, *DS*, *Nord*, *Idec*, *N*)

Compute edge Hcurl ancillary functions and their curls.

Parameters

- **`S`** (*`ndarray`*) – affine coordinates associated to edge
- **`DS`** (*`ndarray`*) – derivatives of S in R^N
- **`Nord`** (*`int`*) – polynomial order
- **`Idec`** (*`bool`*) – Binary flag
- **`N`** (*`int`*) – spatial dimension

Returns edge Hcurl ancillary functions, curls of edge Hcurl ancillary functions

Return type ndarray

Note: References:n Idec: = FALSE s0+s1 != 1

= TRUE s0+s1 = 1

`petgem.hvfem.`**`AncETri`**(*S*, *DS*, *Nord*, *Idec*, *N*)

Compute triangle face Hcurl ancillary functions and their curls.

Parameters

- **`S`** (*`ndarray`*) – (s0,s1,s2) affine coordinates associated to triangle face
- **`DS`** (*`ndarray`*) – derivatives of S0,S1,S2
- **`Nord`** (*`int`*) – polynomial order
- **`Idec`** (*`boll`*) – Binary flag:
- **`N`** (*`int`*) – spatial dimension

Returns triangle Hcurl ancillary functions and curls of triangle Hcurl ancillary functions

Return type ndarray

`petgem.hvfem.`**`HomIJacobi`**(*S*, *DS*, *Nord*, *Minalpha*, *Idec*, *N*)

Compute values of integrated homogenized Jacobi polynomials and their gradients. Result is half of a matrix with each row associated to a fixed alpha. Alpha grows by 2 in each row.

Parameters

- **`S`** (*`ndarray`*) – (s0,s1) affine(like) coordinates
- **`DS`** (*`ndarray`*) – gradients of S in R(N)
- **`Nord`** (*`int`*) – max polynomial order
- **`Minalpha`** (*`int`*) – first row value of alpha (integer)
- **`Idec`** (*`bool`*) – decision flag to compute

Returns polynomial values and derivatives in x (Jacobi polynomials)

Return type ndarray

`petgem.hvfem.`**`HomLegendre`**`(`*S*, *Nord*`)`

Compute values of homogenized Legendre polynomials.

Parameters

- **`S`** (*`ndarray`*) – affine(like) coordinates
- **`Nord`** (*`int`*) – polynomial order

Returns polynomial values

Return type ndarray

`petgem.hvfem.`**`OrientE`**`(`*S*, *DS*, *Nori*, *N*`)`

Compute the local to global transformations of edges.

Parameters

- **`S`** (*`ndarray`*) – projection of affine coordinates on edges
- **`DS`** (*`ndarray`*) – projection of gradients of affine coordinates on edges
- **`Nori`** (*`ndarray`*) – edge orientation
- **`N`** (*`int`*) – number of dimensions

Returns global transformation of edges and global transformation of gradients of edges

Return type ndarray

`petgem.hvfem.`**`OrientTri`**`(`*S*, *DS*, *Nori*, *N*`)`

Compute the local to global transformations of edges.

Parameters

- **`S`** (*`ndarray`*) – projection of affine coordinates on faces
- **`DS`** (*`ndarray`*) – projection of gradients of affine coordinates on faces
- **`Nori`** (*`ndarray`*) – face orientation
- **`N`** (*`int`*) – number of dimensions

Returns global transformation of faces and global transformation of gradients of faces

Return type ndarray

`petgem.hvfem.`**`PolyIJacobi`**`(`*X*, *T*, *Nord*, *Minalpha*, *Idec*`)`

Compute values of integrated shifted scaled Jacobi polynomials and their derivatives starting with p=1.

Result is 'half' of a matrix with each row associated to a fixed alpha. Alpha grows by 2 in each row.

Parameters

- **`X`** (*`ndarray`*) – coordinate from [0,1]
- **`T`** (*`ndarray`*) – scaling parameter
- **`Nord`** (*`int`*) – max polynomial order
- **`Minalpha`** (*`int`*) – first row value of alpha
- **`Idec`** (*`bool`*) – decision flag to compute (= FALSE polynomials with x and t derivatives, = TRUE polynomials with x derivatives only)

Returns polynomial values, derivatives in x (Jacobi polynomials), derivatives in t `petgem.hvfem.`**`PolyJacobi`**(*X*, *T*, *Nord*, *Minalpha*)

Compute values of shifted scaled Jacobi polynomials P**alpha-i.

Result is a half of a matrix with each row associated to a fixed alpha. Alpha grows by 2 in each row.

Parameters

- **`X`** (*`ndarray`*) – coordinate from [0,1]
- **`T`** (*`float`*) – scaling parameter
- **`Nord`** (*`int`*) – max polynomial order
- **`Minalpha`** (*`int`*) – first row value of alpha (integer)

Returns polynomial values

Return type ndarray

`petgem.hvfem.`**`PolyLegendre`**(*X*, *T*, *Nord*)

Compute values of shifted scaled Legendre polynomials.

Parameters

- **`X`** (*`ndarray`*) – coordinate from [0,1]
- **`T`** (*`float`*) – scaling parameter
- **`Nord`** (*`int`*) – polynomial order

Returns polynomial values

Return type ndarray

`petgem.hvfem.`**`ProjectTetE`**(*Lam*, *DLam*)

Projection of tetrahedral edges in concordance with numbering of topological entities (vertices, edges, faces).

Parameters

- **`Lam`** (*`ndarray`*) – affine coordinates
- **`DLam`** (*`ndarray`*) – gradients of affine coordinates

Returns projection of affine coordinates on edges, projection of gradients of affine coordinates on edges

Return type ndarray

Note: References:n Fuentes, F., Keith, B., Demkowicz, L., & Nagaraj, S. (2015). Orientation embedded high order shape functions for the exact sequence elements of all shapes. Computers & Mathematics with applications, 70(4), 353-458.

`petgem.hvfem.`**`ProjectTetF`**(*Lam*, *DLam*)

Projection of tetrahedral faces in concordance with numbering of topological entities (vertices, edges, faces).

Parameters

- **`Lam`** (*`ndarray`*) – affine coordinates
- **`DLam`** (*`ndarray`*) – gradients of affine coordinates

Returns projection of affine coordinates on faces, projection of gradients of affine coordinates on faces

Return type ndarray

`petgem.hvfem.`**`compute2DGaussPoints`**(*Nord*)

Compute 2D gauss points for high-order nédélec elements.

Parameters **`Nord`** (*`int`*) – polynomial order of nedelec basis functions

Returns coordinates of gauss points and its weights

Return type ndarray.

`petgem.hvfem.`**`compute3DGaussPoints`**(*Nord*)

Compute 3D gauss points for high-order nédélec elements.

Parameters **`Nord`** (*`int`*) – polynomial order of nedelec basis functions

Returns coordinates of gauss points and its weights

Return type ndarray.

`petgem.hvfem.`**`computeBasisFunctions`**(*edge_orientation*, *face_orientation*, *jacobian*, *invjacob*, *Nord*, *points*)

Compute the basis function for a given element.

Parameters

- **`edges_orientation`** (*`ndarray`*) – orientation for edges
- **`faces_orientation`** (*`ndarray`*) – orientation for faces
- **`jacobian`** (*`ndarray`*) – jacobian matrix
- **`invjacob`** (*`ndarray`*) – inverse of jacobian matrix
- **`Nord`** (*`int`*) – polynomial order of nedelec basis functions
- **`points`** (*`ndarray`*) – spatial points at which basis functions will be computed

Returns basis functions and its curl for p-order=Nord

Return type ndarray

`petgem.hvfem.`**`computeBasisFunctionsReferenceElement`**(*edge_orientation*, *face_orientation*, *Nord*, *points*)

Compute the basis function for the reference element.

Parameters

- **`edges_orientation`** (*`ndarray`*) – orientation for edges
- **`faces_orientation`** (*`ndarray`*) – orientation for faces
- **`Nord`** (*`int`*) – polynomial order of nedelec basis functions
- **`points`** (*`ndarray`*) – spatial points at which basis functions will be computed

Returns basis functions on reference element

Return type ndarray

`petgem.hvfem.`**`computeConnectivityDOFS`**(*elemsE*, *elemsF*, *Nord*)

Compute the degrees of freedom connectivity for a given list of edges, faces and elements.

Parameters

- **`elemsE`** (*`ndarray`*) – elements-edge connectivity with dimensions = (number_elements, 6)
- **`elemsF`** (*`ndarray`*) – element/faces connectivity with dimensions = (number_elements, 4)

- **Nord** (*int*) – polynomial order of nedelec basis functions

Returns local/global dofs list for elements, dofs index on edges, dofs index on faces, dofs index on volumes, total number of dofs

Return type ndarray and int

Note: References:n Amor-Martin, A., Garcia-Castillo, L. E., & Garcia-Doñoro, D. D. (2016). Second-order Nédélec curl-conforming prismatic element for computational electromagnetics. IEEE Transactions on Antennas and Propagation, 64(10), 4384-4395.

`petgem.hvfem.`**`computeElementOrientation`**(*edgesEle*, *nodesEle*, *edgesNodesEle*, *globalEdgesInFace*)

Compute the orientation for the computation of hierarchical basis functions of high-order (High-order nédélec basis functions).

:param ndarray edgesEle:list of element's edges :param ndarray nodesEle: list of element's nodes :param ndarray edgesNodesEle: list of nodes for each edge in edgesEle :param ndarray globalEdgesInFace: list of edges for each face :return: orientation for edges and orientation for faces :rtype: ndarray.

Note: References:n Amor-Martin, A., Garcia-Castillo, L. E., & Garcia-Doñoro, D. D. (2016). Second-order Nédélec curl-conforming prismatic element for computational electromagnetics. IEEE Transactions on Antennas and Propagation, 64(10), 4384-4395.

`petgem.hvfem.`**`computeElementalMatrices`**(*edge_orientation*, *face_orientation*, *jacobian*, *invjacob*, *Nord*, *sigmaEle*)

Compute the elemental mass matrix and stiffness matrix based ons high-order vector finite element.

Parameters

- **edges_orientation** (*ndarray*) – orientation for edges
- **faces_orientation** (*ndarray*) – orientation for faces
- **jacobian** (*ndarray*) – jacobian matrix
- **invjacob** (*ndarray*) – inverse of jacobian matrix
- **Nord** (*int*) – polynomial order of vector basis functions
- **sigmaEle** (*ndarray*) – element conductivity with dimensions (1, 2), (horizontal and vertical)

Returns elemental mass matrix and elemental stiffness matrix

Return type ndarray

Note: References:n Fuentes, F., Keith, B., Demkowicz, L., & Nagaraj, S. (2015). Orientation embedded high order shape functions for the exact sequence elements of all shapes. Computers & Mathematics with applications, 70(4), 353-458.

`petgem.hvfem.`**`computeJacobian`**(*eleNodes*)

Compute the jacobian and its inverse.

Parameters **eleNodes** (*ndarray*) – spatial coordinates of the nodes with dimensions = (4,3)

Returns jacobian matrix and its inverse.

Return type ndarray

`petgem.hvfem.`**`computeSourceVectorRotation`**(*azimuth*, *dip*)

Compute the weigths vector for source rotation in the xyz plane.

Parameters

- **`azimuth`** (*`float`*) – degrees for x-y plane rotation
- **`dip`** (*`float`*) – degrees for x-z plane rotation

Returns weigths for source rotation

Return type ndarray.

`petgem.hvfem.`**`get2DJacobDet`**(*coordEle*, *faceNumber*)

Compute the determinant of the jacobian for 2D integrals (when 3D basis functions are used)

Parameters

- **`coordEle`** (*`ndarray`*) – coordinates of the tetrahedron
- **`faceNumber`** (*`int`*) – local face number

Returns determinant of the 2D jacobian

Return type ndarray

`petgem.hvfem.`**`getFaceByLocalNodes`**(*faceNumber*)

Get local nodes ordering for a given face

Parameters **`faceNumber`** (*`int`*) – local face number

Returns list of local nodes for faceNumber

Return type ndarray

`petgem.hvfem.`**`getNeumannBCface`**(*face_flag*, *polarization*, *ud*)

Get Neumann boundary condition for boundary face.

Parameters

- **`face_flag`** (*`int`*) – face flag (side on which face belongs)
- **`polarization`** (*`int`*) – polarization mode (x-mode or y-mode)
- **`ud`** (*`ndarray`*) – magnetic field vector

Returns value of Neumann boundary condition for boundary face

Return type ndarray

`petgem.hvfem.`**`getNormalVector`**(*faceNumber*, *invJacobMatrix*)

This function computes the normal vector for a given tetrahedral face.

Parameters

- **`faceNumber`** (*`int`*) – local face number
- **`invJacobMatrix`** (*`int`*) – inverse of jacobian matrix

Returns face normal vector

Return type ndarray

`petgem.hvfem.`**`getRealFromReference`**(*rRef*, *verticesReal*)

Translate a point defined in the reference element (rRef) to the real element (rReal) defined by verticesReal

Parameters

- **`rRef`** (*`ndarray`*) – reference coordinates

- **verticesReal** (*ndarray*) – real coordinates of element (4x3) = (4 nodes x 3 coordinates)

Returns points in real element

Return type ndarray

`petgem.hvfem.`**`shape3DETet`**(*X*, *Nord*, *NoriE*, *NoriF*)

Compute values of 3D tetrahedron element H(curl) shape functions and their derivatives.

Parameters

- **X** (*ndarray*) – master tetrahedron coordinates from (0,1)^3
- **Nord** (*int*) – polynomial order
- **NoriE** (*ndarray*) – edge orientation
- **NoriF** (*ndarray*) – face orientation

Returns number of dof, values of the shape functions at the point, curl of the shape functions

Return type ndarray.

Note: References:n Fuentes, F., Keith, B., Demkowicz, L., & Nagaraj, S. (2015). Orientation embedded high order shape functions for the exact sequence elements of all shapes. Computers & Mathematics with applications, 70(4), 353-458.

`petgem.hvfem.`**`tetrahedronXYZToXiEtaZeta`**(*eleNodes*, *points*)

Compute the reference tetrahedron coordinates from xyz global tetrahedron coordinates.

Parameters

- **eleNodes** (*ndarray*) – spatial coordinates of the nodes with dimensions = (4,3)
- **points** (*ndarray*) – xyz points coordinates to be transformed

Returns xietazeta points coordinates

Return type ndarray

`petgem.hvfem.`**`transform2Dto3DInReferenceElement`**(*points2D*, *faceNumber*)

Transforms 2D points defined on some face into its 3D representation.

Parameters

- **points2D** (*ndarray*) – 2D points to be transformed
- **faceNumber** (*int*) – local face number

Returns resulting 3D points

Return type ndarray

`petgem.hvfem.`**`unitary_test`**()

Unitary test for hvfem.py script.

vectors.py

Define standard vector and matrix functions.

`petgem.vectors.`**`deleteDuplicateRows`**(*matrix*)

Delete duplicate rows in a matrix.

Parameters **`matrix`** (*`ndarray`*) – input matrix to be processed.

Returns matrix without duplicate rows

Return type ndarray

`petgem.vectors.`**`findUniqueRows`**(*array*, *return_index=False*, *return_inverse=False*)

Find unique rows of a two-dimensional numpy array.

Parameters

- **`ndarray`** – array to be processed.
- **`return_index`** (*`bool`*) – the indices of array that result in the unique array.
- **`return_inverse`** (*`bool`*) – indices of the unique array that can be used to reconstruct array.

Returns unique rows.

Return type ndarray.

`petgem.vectors.`**`invConnectivity`**(*M*, *nP*)

Compute the opposite connectivity matrix of M.

Parameters

- **`M`** (*`ndarray`*) – connectivity matrix with dimensions = (nElems,eleOrder)
- **`nP`** (*`int`*) – number of nodes/edges/faces in matrix M.

Returns connectivity matrix with dimensions = (nNodes,S), (nEdges,S) or (nFaces,S)

Return type ndarray.

Note: eleOrder determines the number of entities per element in matrix M, therefore 4 is the nodal element order, 6 is the edge element order and 3 is the element order of faces. S in the output is the maximum number of elements sharing a given node/edge/face

If M represents a element/nodes connectivity, the function computes a node/elements connectivity.

If M represents a element/edges connectivity, the function computes a edge/elements connectivity.

If M represents a element/faces connectivity, the function computes a faces/elements connectivity.

`petgem.vectors.`**`is_duplicate_entry`**(*x*)

Compute number of duplicate entries in a vector.

Parameters **`x`** (*`int-array`*) – matrix.

Returns number of duplicate entries.

Return type int.

`petgem.vectors.`**`unitary_test`**()

Unitary test for vectors.py script.

6.11.6 Solver

solver.py

Define functions a 3D CSEM/MT solver using high-order vector finite element method (HEFEM).

`petgem.solver.`**`unitary_test`**`()`

Unitary test for solver.py script.

6.11.7 Parallel

parallel.py

Define classes and functions for parallel computations within **PETGEM**.

`petgem.parallel.`**`createParallelDenseMatrix`**`(`*dimension1*, *dimension2*, *communicator=None*`)`

Create a parallel dense matrix in petsc format.

Parameters

- **`dimension1`** (*`int`*) – matrix dimension (rows).
- **`dimension2`** (*`int`*) – matrix dimension (columns).
- **`communicator`** (*`str`*) – mpi communicator.

Returns parallel matrix.

Return type petsc parallel and dense matrix.

`petgem.parallel.`**`createParallelMatrix`**`(`*dimension1*, *dimension2*, *nnz*, *matrix_type*, *communicator=None*`)`

Create a parallel sparse matrix in petsc format.

Parameters

- **`dimension1`** (*`int`*) – matrix dimension (rows).
- **`dimension2`** (*`int`*) – matrix dimension (columns).
- **`nnz`** (*`int`*) – not zero pattern for allocation.
- **`matrix_type`** (*`int`*) – matrix type for parallel computations.
- **`communicator`** (*`str`*) – mpi communicator.

Returns parallel matrix.

Return type petsc AIJ parallel matrix.

`petgem.parallel.`**`createParallelVector`**`(`*size*, *vector_type*, *communicator=None*`)`

Create a parallel vector in petsc format.

Parameters

- **`size`** (*`int`*) – vector size.
- **`vector_type`** (*`int`*) – vector type for parallel computations.
- **`communicator`** (*`str`*) – mpi communicator.

Returns parallel vector.

Return type petsc parallel vector.

`petgem.parallel.`**`createSequentialDenseMatrixWithArray`**`(`*dimension1*, *dimension2*, *data*`)`

Given an input array, create a sequential dense matrix in petsc format.

Parameters

- **`dimension1`** (*`int`*) – matrix dimension (rows).

- **`dimension2`** (*`int`*) – matrix dimension (columns).
- **`data`** (*`ndarray`*) – data to be exported.

Returns parallel matrix.

Return type petsc parallel and dense matrix.

`petgem.parallel.`**`createSequentialVector`**`(`*size*, *vector_type*, *communicator=None*`)`
Create a sequential vector in petsc format.

Parameters

- **`size`** (*`int`*) – vector size.
- **`vector_type`** (*`int`*) – vector type for parallel computations.
- **`communicator`** (*`str`*) – mpi communicator.

Returns sequential vector.

Return type petsc sequential vector.

`petgem.parallel.`**`createSequentialVectorWithArray`**`(`*data*`)`
Given an input array, create a sequential vector in petsc format.

Parameters **`data`** (*`ndarray`*) – data to be exported.

Returns parallel matrix.

Return type petsc parallel and dense matrix.

`petgem.parallel.`**`readPetscMatrix`**`(`*input_file*, *communicator=None*`)`
Read a Petsc matrix which format is defined by two files: input_file.dat and input_file.info.

Parameters

- **`input_file`** (*`str`*) – file name to be readed.
- **`communicator`** (*`str`*) – mpi communicator.

Returns petsc_matrix.

Return type petsc sparse matrix.

`petgem.parallel.`**`readPetscVector`**`(`*input_file*, *communicator=None*`)`
Read a Petsc vector which format is defined by two files: input_file.dat and input_file.info.

Parameters

- **`input_file`** (*`str`*) – file name to be readed.
- **`communicator`** (*`str`*) – mpi communicator.

Returns petsc_vector.

Return type petsc vector.

`petgem.parallel.`**`unitary_test`**`()`
Unitary test for parallel.py script.

`petgem.parallel.`**`writeDenseMatrix`**`(`*output_file*, *data*, *communicator=None*`)`
Write a Petsc dense matrix which format is defined by two files: output_file.dat and output_file.info.

Parameters

- **`output_file`** (*`str`*) – file name to be saved.
- **`matrix data`** (*`petsc`*) – dense matrix to be saved.

- **communicator** (*str*) – mpi communicator.

Returns None.

petgem.parallel.**writeParallelDenseMatrix**(*output_file*, *data*, *communicator=None*)
Write a Petsc parallel dense matrix which format is defined by two files: output_file.dat and output_file.info.

Parameters

- **output_file** (*str*) – file name to be saved.
- **matrix data** (*petsc*) – dense matrix to be saved.
- **communicator** (*str*) – mpi communicator.

Returns None.

petgem.parallel.**writePetscVector**(*output_file*, *data*, *communicator=None*)
Write a Petsc vector which format is defined by two files: output_file.dat and output_file.info.

Parameters

- **output_file** (*str*) – file name to be saved.
- **vector data** (*petsc*) – array to be saved.
- **communicator** (*str*) – mpi communicator.

Returns None.

6.11.8 Post-processing

preprocessing.py

Define data postprocessing operations for **PETGEM**.

petgem.postprocessing.**computeImpedance**(*fields*, *omega*, *mu*)
Compute apparent resistiviy, phase, tipper and impedance for MT mode.

Parameters

- **fields** (*list*) – list of numpy arrays with electromagnetic fields with dimensions = (number_polarizations, number_receivers, number_EM_components)
- **omega** (*float*) – angular frequency
- **mu** (*float*) – medium permeability.

Returns apparent resistivity, phase, tipper and impedance.

Return type ndarray

petgem.postprocessing.**fieldInterpolator**(*solution_vector*, *nodes*, *elemsN*, *elemsE*, *edgesN*, *elemsF*, *facesE*, *dof_connectivity*, *points*, *inputSetup*)
Interpolate electromagnetic field for a set of 3D points.

Parameters

- **solution_vector** (*ndarray-petsc*) – vector field to be interpolated
- **nodes** (*ndarray*) – nodal coordinates
- **elemsN** (*ndarray*) – elements-node connectivity with dimensions = (number_elements, 4)

- **elemsE** (*ndarray*) – elements-edge connectivity with dimensions = (number_elements, 6)
- **edgesN** (*ndarray*) – edge-node connectivity with dimensions = (number_edges, 2)
- **elemsF** (*ndarray*) – element-faces connectivity with dimensions = (number_elements, 4)
- **facesE** (*ndarray*) – face-edges connectivity with dimensions = (number_faces, 3)
- **dof_connectivity** (*ndarray*) – local/global dofs list for elements, dofs index on edges, dofs index on faces, dofs index on volumes, total number of dofs
- **points** (*ndarray*) – point coordinates
- **inputSetup** (*obj*) – inputSetup object.

Returns electromagnetic fields for a set of 3D points

Return type ndarray and int

`petgem.postprocessing.`**`unitary_test`**`()`

Unitary test for postprocessing.py script.

CHAPTER

SEVEN

PUBLICATIONS

Papers:

- Castillo-Reyes, O., Queralt, P., Marcuello, A., Ledo, J. (2021). Land CSEM Simulations and Experimental Test Using Metallic Casing in a Geothermal Exploration Context: Vall‘es Basin (NE Spain) Case Study. IEEE Transactions on Geoscience and Remote Sensing.
- Castillo-Reyes, O., de la Puente, J., García-Castillo, L. E., Cela, J.M. (2019). Parallel 3-D marine controlled-source electromagnetic modelling using high-order tetrahedral Nédélec elements. Geophysical Journal International, Volume 219, Issue 1, October 2019, Pages 39–65.
- Castillo-Reyes, O., de la Puente, Cela, J. M. PETGEM: A parallel code for 3D CSEM forward modeling using edge finite elements. Computers & Geosciences, vol 119: 123-136. ISSN 0098-3004. Elsevier.
- Castillo-Reyes, O., de la Puente, Cela, J. M. Three-Dimensional CSEM modelling on unstructured tetrahedral meshes using edge finite elements. In: Barrios Hernández C., Gitler I., Klapp J. (eds) High Performance Computing. CARLA 2016. Communications in Computer and Information Science, vol 697: 247-256. ISBN 978-3-319-57971-9 Springer, Cham. 2017.
- Castillo-Reyes, O., de la Puente, Cela, J. M. Improving edge finite element assembly for geophysical electromagnetic modelling on shared-memory architectures. 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference – UEMCON. 2016.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Edge-based parallel framework for the simulation of 3D CSEM surveys. ICE Barcelona-AAPG/SEG International Conference & Exhibition. 2016.
- Castillo-Reyes, O., de la Puente, J., Barucq, H., Diaz, J., and Cela, J. M., Parallel and vectorized code for CSEM surveys in geophysics: An edge-based approach. ECCOMAS. 2016.
- Castillo-Reyes, O., de la Puente, J., Modesto, D., Puzyrev, V., and Cela, J. M., A parallel tool for numerical approximation of 3D electromagnetic surveys in geophysics. Computación y Sistemas: Topic trends in computing research, vol. 20, no. 1, pp 29-39. 2016.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Towards an HPC tool for simulation of 3D CSEM surveys: an edge-based approach. PRACEdays16 Conference. 2016.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Assessment of edge-based finite element technique for geophysical electromagnetic problems: efficiency, accuracy and reliability. Proceedings of the 1st Pan-American Congress on Computational Mechanics and XI Argentine Congress on Computational Mechanics. CIMNE, pp. 984-995, 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Edge-based electric field formulation in 3D CSEM simulations: a parallel approach. Proceedings of the 6th International Conference and Workshop on Computing and Communication. IEEE, 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Edge-elements for geophysical electromagnetic problems: a new implementation challenge. PRACEdays15 Conference. 2015.

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., HPC and edge elements for geophysical electromagnetic problems: an overview. BSC Doctoral Symposium (2nd: 2015: Barcelona). 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Parallel and numerical issues of the edge finite element method for 3D controlled-source electromagnetic surveys. Proceedings of the International Conference on Computing Systems and Telematics. IEEE, 2015.

Conferences:

- Castillo-Reyes, O. Modeling and Inversion 3D Electromagnetic Datasets on HPC platforms. ADMOS 2021 10th International Conference On Adaptive Modeling and Simulation. June 2021.
- Castillo-Reyes, O. Reservoir characterization using geophysics electromagnetic and HPC. SIAM Conference on Mathematical & Computational Issues in the Geosciences. June 2021.
- Castillo-Reyes, O., Queralt, P., Marcuello, A., Ledo, J., Amor-Martin, A., García-Castillo, L.E. 14:00 - 15:00 3D Electromagnetic Modeling and Inversion using an Open-Source Paradigm: Experiences and Perspectives. SIAM Conference on Mathematical & Computational Issues in the Geosciences. June 2021.
- Castillo-Reyes, O. Electromagnetic modeling using steel casing in a geothermal exploration context Online AGU Fall Meeting. 8o Congreso Metropolitano de Modelado y Simulación Numérica 2021. May 2021.
- Co-author. Open-source lanscape for Three-Dimensional Controlled-Source Electromagnetic Modeling. Online AGU Fall Meeting. December 2020.
- Castillo-Reyes, O. Control Source Electromagnetic test in the Vallès Basin (Spain) for geothermal characterization: experiment setup and numerical simulations. Online AGU Fall Meeting. December 2020.
- Castillo-Reyes, O. Land CSEM simulations and experimental test using metallic casing in a geothermal exploration context: Vallés Basin (NE Spain) case study. Online MATHROCKS Workshop on Simulation and Inversion Methods in Geophysics. Workshop virtual organized by the Institut National de Recherche en Informatique et en Automatique – INRIA (Pau University, France) and University of the Basque Country (Spain). November 2020.
- Castillo-Reyes, O. Electromagnetic modeling for geothermal resources exploration. PIXIL Virtual Workshop. Virtual conference organized by the Institut National de Recherche en Informatique et en Automatique – INRIA. Pau University, France. May 2020.
- Castillo-Reyes, O. HPC with python for numerical modeling in geophysics. Workshop on mathematical and computational modelling on the Earth. LaCaN, Polytechnic University of Catalonia, Barcelona, Spain. May 2019.
- Castillo-Reyes, O. HPC for numerical simulation in geophysics. 7° Congreso metropolitano de modelado y simulación numérica. National Autonomous University of Mexico – UNAM. CDMX, Mexico. May 2019.
- Castillo-Reyes, O. Electromagnetic modeling using Nédélec elements of high-order and HPC. XXXIX Ibero-Latin American Congress on Computational Methods in Engineering – CILAMCE. Sorbonne Universités. Paris / Compiègne, France. November 2018.
- Castillo-Reyes, O. HPC code development using Python. Mexican Supercomputing Network annual meeting – RedMexSu. University of Guadalajara. Guadajalara, Mexico. October 2018.
- Castillo-Reyes, O. HPC geophysical electromagnetic modeling. Fifth International Congress on Multiphysics, Multiscale, and Optimization problems. University of the Basque Country. Bilbao, Spain. May 2018.
- Castillo-Reyes, O. Python performance for HPC geophysical applications. 9th International Supercomputing Conference in Mexico – ISUM 2018. Red Mexicana de Supercómputo. Mérida, Yucatán, Mexico. March 2018.
- Castillo-Reyes, O. Overview of numerical techniques and applications for CSEM/MT geophysical surveys. SIAM Conference on Mathematical and Computational Issues in the Geosciences. University Erlangen-Nürnberg. Erlangen, Germany. September 2017.

- Castillo-Reyes, O. PETGEM: Parallel Edge-based Tool for Geophysical Electromagnetic Modelling. Congress on Numerical Methods in Engineering. Technical University of Valencia. Valencia, Spain. July 2017.
- Castillo-Reyes, O. PETGEM: potential of 3D CSEM modelling using a new HPC tool for exploration geophysics. 10th International Marine Electromagnetics conference – MARELEC. University of Liverpool. Liverpool, United Kingdom. June 2017.
- Castillo-Reyes, O. High performance computing using python: advances in geophysical electromagnetic modelling. Computing and Electromagnetics International Workshop. Polytechnic University of Catalonia. Barcelona, Spain. June 2017.
- Castillo-Reyes, O. Python code for CSEM modelling in geophysics and HPC architectures: advances and challenges. 2do Foro Internacional de Talento Mexicano – Innovation Match MX 2016-2017. Mexico, D.F. May 2017.
- Castillo-Reyes, O. Python for HPC geophysical applications. GeoPython 2017. University of Applied Sciences and Arts Northwestern Switzerland. Basel, Switzerland. May 2017.
- Castillo-Reyes, O. Python for HPC geophysical electromagnetic applications: experiences and perspectives. 4th BSC International Doctoral Symposium. Barcelona, Spain. May 2017.
- Castillo-Reyes, O. See underneath. High Performance Computing, geophysics and electromagnetic methods. Interdisciplinary Meeting of Predoctoral Researchers – JIPI 2017. University of Barcelona. Barcelona, Spain. February 2017.
- Castillo-Reyes, O. Supercomputing and electromagnetic modelling in geophysics: advances and challenges. Centro de Ciencias de la tierra. University of Veracruz. Xalapa, Veracruz, Mexico. December 2016.
- Castillo-Reyes, O. Improving edge finite element assembly for geophysical electromagnetic modelling on shared-memory architectures. 7th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference – UEMCON 2016. New York, USA. October 2016.
- Castillo-Reyes, O. Three-dimensional CSEM modelling on unstructured tetrahedral meshes using edge finite elements. Latin American High Performance Computing Conference – CARLA 2016. Mexico, D.F. August 2016.
- Castillo-Reyes, O. Edge-based parallel code for CSEM surveys in geophysics: performance and accuracy improvements. 12th World Congress on Computational Mechanics – WCCM XII. Seúl, Corea. July 2016.
- Castillo-Reyes, O. Towards an HPC tool for 3D CSEM forward modelling in geophysics. Fourth International Congress on Multiphysics, Multiscale, and Optimization problems. Bilbao, España. May 2016
- Castillo-Reyes, O. High performance computing, geophysics and numerical methods: a symbiotic relation. 1er Foro Internacional de Talento Mexicano – Innovation Match MX 2015-2016. Guadalajara, Jalisco, México. April 2016.
- Castillo-Reyes, O. Edge-based electric field formulation in 3D CSEM simulations: a parallel approach. 6th International Conference and Workshop on Computing and Communication – IEMCON – 2015. University of British Columbia. Vancouver, Canada. October 2015.
- Castillo-Reyes, O. High Performance Computing and electromagnetic modeling in geophysics: from concepts to application. Research Center in Computing. National Polytechnic Institute. Mexico, D.F. October 2015.
- Castillo-Reyes, O. Parallel and numerical issues of the edge finite element method for 3D controlled-source electromagnetic surveys. IEEE International Conference on Computing Systems and Telematics. University of Veracruz. Xalapa, Veracruz, Mexico. October 2015.
- Castillo-Reyes, O. "Your Thesis in 3 Minutes" (3TM) with the topic: Edge-elements formulation of CSEM in geophysics: a parallel approach. Jornadas de Cooperación CONACyT – Cataluña 2015. Polytechnic University of Catalonia – National Council of Science and Technology of Mexico. Barcelona, Spain. June 2015.

- Castillo-Reyes, O. Edge-elements for geophysical electromagnetic problems: A new implementation challenge. PRACE Scientific and Industrial Conference 2015 – PRACEDays15. Dublin, Ireland. April 2015.
- Castillo-Reyes, O. HPC and edge elements for geophysical electromagnetic problems: an overview. 2nd BSC International Doctoral Symposium. Barcelona, Spain. April 2015.
- Castillo-Reyes, O. Assessment of edge-based finite element technique for geophysical electromagnetic problems: efficiency, accuracy and reliability. 1st. Pan-American Congress on Computational Mechanics – PANACM 2015. IACM. Buenos Aires, Argentina. April 2015.
- Castillo-Reyes, O. HPC solutions for oil industry: trends and challenges. Centro de Ciencias de la Tierra. University of Veracruz. Xalapa, Veracruz, Mexico. December 2014.
- Castillo-Reyes, O. High Performance Computing, Science and Engineering. Master in Telematic. School of Accounting and Management. University of Veracruz. Xalapa, Veracruz, Mexico. December 2014.
- Castillo-Reyes, O. HPC solutions for oil industry: trends and challenges. IV Simposio de Becarios CONACyT en Europa. Strasbourg, France. November 2014.

CHAPTER

EIGHT

SUPPORT

Work on PETGEM has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 644202. The research leading to these results has received funding from the European Union's Horizon 2020 Programme (2014-2020) and from Brazilian Ministry of Science, Technology and Innovation through Rede Nacional de Pesquisa (RNP) under the HPC4E Project , grant agreement No. 689772. Further, this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 777778. Moreover, the development of PETGEM has received funding from the European Union's Horizon 2020 programme, grant agreement N○828947, and from the Mexican Department of Energy, CONACYT-SENER Hidrocarburos grant agreement N○B-S-69926. Finally, this project has been 65% cofinanced by the European Regional Development Fund (ERDF) through the Interreg V-A Spain-France-Andorra program (POCTEFA2014-2020). POCTEFA aims to reinforce the economic and social integration of the French-Spanish-Andorran border. Its support is focused on developing economic, social and environmental cross-border activities through joint strategies favouring sustainable territorial development

CHAPTER

NINE

DOWNLOAD

9.1 Conditions of use

PETGEM is developed as open-source under `BSD-3` license.

9.2 Downloads

9.2.1 Scripts

- PETGEM is available for download at the project website generously hosted by PyPi and GitHub. Download here or here.
- `kernel.py`: python script that manages the PETGEM work-flow. Download `here`.
- `petsc.opts`: options file for PETSc solvers/preconditioners. Download `here`.
- `params.yaml`: file that defines physical parameters of the 3D CSEM survey. Download `here`.

9.2.2 CSEM examples

- Example 1: Dataset for the Canonical model of an off-shore hydrocarbon reservoir (Nédélec elements of first-order). Download `here`.
- Example 2: Dataset for the Canonical model of an off-shore hydrocarbon reservoir (Nédélec elements of second-order). Download `here`.
- Example 3: Dataset for the use of PETSc solvers. Download `here`.

9.2.3 MT examples

- Example 1: Dataset for the 3D trapezoidal hill model (Nédélec elements of second-order). Download `here`.

CHAPTER

TEN

CONTACT

Octavio Castillo-Reyes
Tel: +34 934137992
email: octavio.castillo@bsc.es
Location: Nexus II building - third floor C/ Jordi Girona, 29. Barcelona 08034

CHAPTER

ELEVEN

VIEW MAP

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX