custEM Documentation

Release 01.03.2022, custEM version: 1.2.

R. Rochlitz

Mar 03, 2022

Contents

Version: 1.2. ~ Date: Mar 03, 2022

The Python toolbox **custEM** is an open-source development for customizable 3D finite-element modeling of controlled-source, transient, and natural-source electromagnetic data. The toolbox is based on the finite-element library FEniCS, see https://fenicsproject.org/ . Different total and secondary electric or magnetic field as well as gauged-potential approaches are implemented. In addition, **custEM** contains a mesh generation submodule for the straightforward generation of tetrahedral meshes supporting a large number of marine, land-based, airborne or mixed model scenarios. Interpolation and visualization tools simplify the post-processing workflow.

Mesh generation runs on any laptop or desktop PC. Also minimalistic examples can be tested on such machines. Because of using direct solvers, we recommend to get access to a cluster with >= 32 cores/threads and >=256 GB RAM for calculating sufficiently accurate models for more complex 3D setups. However, quite good results for simpler 3D modeling studies often require only 32-64 GB.

Please note that custEM is under continuous development. For information about planned updates and previous development steps we refer to the *Development* section. Do not hesitate to contact raphael.rochlitz@leibniz-liag.de for any kind of question about custEM or discovered issues in the code, examples, or the documentation.

CHAPTER 1

Getting started

To guide you through setting up the toolbox and making third party dependencies available, please follow the *Installation* notes. After A successful installation, user could run any of the test files provided in the corresponding directory. In addition to these initial tests for the overall functionality of **custEM**, we refer to the provided examples. The first four frequency-domain and first three time-domain examples correspond to the ones presented by (Rochlitz et al., 2019) & (Rochlitz et al., 2021), respectively. Further advise to develop solution strategies for user-specific modeling problems is provided by the tutorials.

In addition, we recommend to read the *Tips&Tricks* section, which contains further practical advise based on our modeling experiences.

CHAPTER 2

Installation

For instructions, it is referred to the *Installation* section.

CHAPTER 3

Documentation

The python API documentation is available online on ReadtheDocs. In, addition, the complete API documentation is available as “hmtl_doc” zip-file in the *docs* directory of the custEM repository and can be accessed via opening the “index.html” file in the zip directory with your favorite browser.

The *Source code* documentation content is generated automatically when publishing new custEM versions. Detailed information about specific functionalities might be accessed with help of the genindex or using the search.

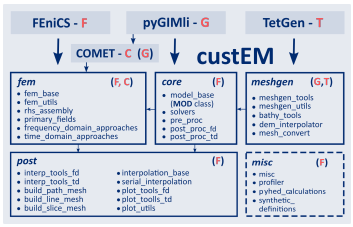

Fig. 1: Overview of Third-party dependencies and modules in custEM

CHAPTER 4

Tutorials

Tutorials are available in the *tutorials* directory in the **custEM** repository as *jupyter notebook*, *python* script, and in *html* format. The *html* documentation of the *Tutorials* can be also accessed in the corresponding section. Please note that the usage of *jupyter notebooks* is not straightforward in combination with multiprocessing. For running the *run_tutorial*, please use the provided *run_tutorial.py* script and call it from the command line with the *mpirun* syntax:

–> mpirun -n x python run_tutorial.py

with x, the number of MPI processes.

CHAPTER 5

Examples

A selection of CSEM modeling examples can be found *here*.

CHAPTER 6

License

Copyright 2016-2020 by R. Rochlitz

The **custEM** toolbox is licensed under the GNU GENERAL PUBLIC LICENSE (GPL). The terms are listed here *License*.

CHAPTER 7

Authors

Contributing authors and contacts are listed in the *Authors* section.

CHAPTER 8

References

Rochlitz, R., Skibbe, N. and Günther, T. (2019), *custEM: customizable finite element simulation of complex controlled-source electromagnetic data*, GEOPHYSICS Software and Algorithms, 84(2), F17-F33, doi: 10.1190/geo2018-0208.1.

Rochlitz, R. (2020), *Analysis and open-source Implementation of Finite Element Modeling techniques for Controlled-Source Electromagnetics*, PhD thesis, Westfälische Wilhelms-Universität Münster, Germany.

Rochlitz, R., Seidel, M. and Börner, R.-U. (2021), *Evaluation of three approaches for simulating time-domain electromagnetic data using the open-source software custEM*, Geophysical Journal International, 227(3), 1980-1995., doi: 10.1093/gji/ggab302.

Werthmüller, D., Rochlitz, R., Castillo-Reyes, O., and Heagy, L. (2021), *Towards an open-source landscape for 3-D CSEM modelling*, Geophysical Journal International, 227(1), 644-659, doi: 10.1093/gji/ggab238.

If required, please contact raphael.rochlitz@leibniz-liag.de for access.

Next to us, don't forget to give credits to the authors of FEniCS, pyGIMLi, TetGen, COMET and empymod if sub-modules based on their developments are used.

8.1 Installation

Currently, FEniCS is only working properly on Linux (and maybe Mac) systems. Therefore, custEM was developed and is restricted to Linux up to now. There are two recommended and quite robust ways to install **FEniCS**, **pyGIMLi** and **TetGen** as requirements for custEM:

1. Installation via conda on Linux systems - !strongly recommended!
2. Installation on Ubuntu or maybe further (not tested) Linux systems

Alternative installation procedures for FEniCS, i.e., the Docker environment, can be tested by users. However, this is not straightforward and making pyGIMLi available at the same time is tricky.

GitLab repository: https://gitlab.com/Rochlitz.R/custEM/

For installation issues and questions please contact the authors:

raphael.rochlitz@leibniz-liag.de

8.1.1 Conda installation:

- custEM conda package available for straightforward installation
- Successfully tested on: Ubuntu 16.04 and newer, Debian 9 Stretch, Arch Linux, Gentoo, Scientific Linux.
- Complete support of all features with FEniCS versions higher than 2018.1

1. **Install Anaconda or miniconda:**

   - https://conda.io/docs/user-guide/install/index.html)
   - https://conda.io/miniconda.html)

   If not done automatically during installation, don't forget to add conda to your PATH variable via adding a line in your bashrc file (something like):

   –> `export PATH=$HOME/anaconda3/bin:$PATH`

   or

   —> `export PATH=$HOME/miniconda/bin:$PATH`

2. Add required CONDA channels:

   –> `conda config --add channels conda-forge --add channels gimli`

   Note: It is possible to skip global setting of channels. Instead, add `-c gimli -c conda-forge` to the *create* or *install* commands (see 3.).

3. Install custEM and requirements (FEnICS, pyGIMLi, TetGen):

   OPTION 1:

   This option is recommended for users working with existing functionalities of custEM. If your work could likely require minor custEM updates from time to time, we recommend OPTION 2.

   —> `conda create -n custEM custem`

   –> `conda activate custEM`

   Here, *custem* is the conda package provided via the gimli channel and *custEM* is the environment name, which can be modified by users. No setting of paths is required and a stable version of custEM is ensured!

   Examples and tutorials can be downloaded separately from the GitLab repository and placed anywhere on the comptuer architecture.

   OPTION 2:

   This option allows to account for smaller code updates by installing a fixed version of the Third Party dependecies, but using custEM from the GitLab repository and being able to pull most recent versions.

   Download custEM from the GitLab repository by running

   –> `git clone https://gitlab.com/Rochlitz.R/custEM.git`

   or by donwloading as zip file from

   https://gitlab.com/Rochlitz.R/custEM.git

To build a conda environement with all the required dependencies for custEM, navigate to the custEM main directory and run

```
—> conda env create

-> conda activate custEM

—> conda develop .
```

With this option, examples and tutorials were downloaded automatically and are available in the custEM main directory.

4. Depending on your computer architecture, it might be necessary to set the following flag to prevent the MUMPS internal OpenMP parallelization via running the command:

   ```
   -> export OMP_NUM_THREADS=1
   ```

   If you experience that significantly more processes are used than set in the *mpirun* call and the computation lasts unreasonably long, not setting this flag is the reason. It might be reasnoable to use a mix of MPI and OpenMP parallelization, in that case test using more than 1 *OMP_NUM_THREADS*.

8.1.2 Ubuntu (Debian) installation

- Successfully tested on: Ubuntu 16.04 LTS and 18.04 LTS, probably works on Debian Linux systems as well.
- The complete custEM toolbox with all features is supported.
- **FEniCS**, **pyGIMLi** and **TetGen** need to be installed manually.
- Only python 3 is supported as third party libraries do not maintain python 2 compatible versions anymore.

1. Get custEM from zip-file or GitLab repository and add the custEM main directory to the PYTHONPATH. If the custEM repository was installed in the home directory ($HOME), it would be:

   ```
   -> export PYTHONPATH=${PYTHONPATH}:$HOME/custEM
   ```

   Please note that the correct path must not point to the main repository but the **custEM** directory in the repository which contains the submodules!

2. Install FEniCS on Ubuntu (https://fenicsproject.org/download/) via:

   ```
   -> sudo apt-get install software-properties-common -> sudo
   add-apt-repository ppa:fenics-packages/fenics -> sudo apt-get
   update -> sudo apt-get install --no-install-recommends fenics ->
   sudo apt-get dist-upgrade
   ```

3. Install pyGIMLi on Ubuntu (https://www.pygimli.org/installation.html):

   ```
   -> curl -Ls install.pygimli.org | bash
   ```

4. Make your system Python find pyGIMLi by editing the following path variables. In this case, pyGIMLi was installed in the directory: (Note the line breaks!) *$HOME/custEM/ThirdParty/gimli_stable*:

   ```
   -> export PYTHONPATH=${PYTHONPATH}:$HOME/custEM/ThirdParty/
      gimli_stable/gimli/gimli/python

   -> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/custEM/
      ThirdParty/gimli_stable_170817/gimli/build/lib
   ```

5. Get TetGen for mesh generation (http://wias-berlin.de/software/tetgen/) and add *tetgen* to your PATH variable via:

–> `export PATH=$PATH:$HOME/custEM/ThirdParty/tetgen`

6. Depending on your computer architecture, it might be necessary to set the following flag to prevent the MUMPS internal OpenMP parallelization via running the command:

   –> `export OMP_NUM_THREADS=1`

   If you experience that significantly more processes are used than set in the *mpirun* call and the computation lasts unreasonably long, not setting this flag is the reason. It might be reasnoable to use a mix of MPI and OpenMP parallelization, in that case test using more than 1 *OMP_NUM_THREADS*.

8.1.3 Further notes

- It is recommended to add all the *export* commands to your *bashrc* file.
- If custEM was installed via conda, it is recommended to add the '-u' flag after 'python' or 'python3' in the commmand promt calls to force all prints to appear in time and not delayed, e.g.:

  –> mpirun -n 12 python -u run_script.py

- In order to use the provided *jupyter notebook* tutorials, jupyter needs to be installed **after** all other steps listed below:

  –> pip install jupyter

- Computation times might be speed up with reducing the number of mpirun processes (e.g., 8 instead of 32) and enabling OpenMP parallelization during the solution of the system of equations via MUMPS with adjusting the flag *OMP_NUM_THREADS*, e.g., *export OMP_NUM_THREADS=4* instead of 1.

8.2 Source code

8.2.1 custEM package

8.2.1.1 Subpackages

custEM.core package

Submodules

custEM.core.model_base module

@author: Rochlitz.R

class custEM.core.model_base.**MOD**(*mod_name*, *mesh_name*, *approach*, *debug_level=20*, *fenics_debug_level=50*, *mute=False*, ***main_kwargs*)

Bases: `object`

Main class for 3D CSEM modeling using FEniCS. After an instance of this mainclass is created, all modeling steps are conducted within this environement. Instances within the mainclass 'MOD' are:

- FS: *custEM.fem.fem_base.FunctionSpaces*
- FS.PF: *custEM.fem.primary_fields.PrimaryField*
- FS.DOM: *custEM.fem.fem_base.Domains*

- MP: *custEM.fem.fem_base.ModelParameters*
- Solver: *custEM.core.solvers.Solver*
- **FE - depending on the time- or frequency-domain modeling approach:**
  - *custEM.fem.fem_utils.ApproachBaseTD*
  - *custEM.fem.time_domain_approaches.ImplicitEuler*
  - custEM.fem.time_domain_approaches.FourierTransformBased.
  - *custEM.fem.time_domain_approaches.RationalArnoldi*
  - *custEM.fem.fem_utils.ApproachBaseFD*
  - *custEM.fem.frequency_domain_approaches.E_vector*
  - *custEM.fem.frequency_domain_approaches.H_vector*
  - *custEM.fem.frequency_domain_approaches.A_V_nodal*
  - *custEM.fem.frequency_domain_approaches.A_V_mixed*
  - *custEM.fem.frequency_domain_approaches.F_U_mixed*
  - *custEM.fem.fem_utils.DC*
- **IB - depending on time- or frequency-domain modeling:**
  - *custEM.post.interp_tools_td.TimeDomainInterpolator*
  - *custEM.post.interp_tools_fd.FrequencyDomainInterpolator*
- **PP - depending on post-processing or pre-processing:**
  - custEM.core.post_proc.PostProcessing
  - *custEM.core.pre_proc.PreProcessing*

- **solve_main_problem()** function to solve the weak formulation of the main system, defined in the **FE** instance; note that iterative solvers are theoretically considered but not working properly yet
- **init_third_party_parameters()** adjust a few dolfin and numpy parameters; so far, no need for customization
- **init_default_MOD_parameters()** initialize default parameters which can be overwritten by specifying keyword arguments when calling the **MOD** instance
- **init_instances()** initialize all required 'sub'-classes of the **MOD** class
- **handle_exception()** overwrite excepthook from sys module for customized logger

handle_exception(*exc_type*, *exc_value*, *exc_traceback*)
Overwrite *excepthook* from *sys* module to enable raising uncaught exceptions by the customized *self.logger* from the *logging* module and write them to the "*mod_name* + _debug.log" file for debugging.

- see description of Python exceptions and corresponding traceback

init_default_model_parameters()
Initalizes default parameters of the **MOD** class which will be updated if keyword arguments are set when calling the MOD instance. The parameters are described in the **MOD** class documentation.

init_instances()
Initalizes instances within MOD (**FS, MP, PP, FE, Solver, IB**). For more information, we refer to the main documentation of the sub-modules.

init_third_party_parameters(*fenics_debug_level*)

Set some dolfin and print parameters. Might contain more specifications with custom choices in future.

- **fenics_debug_level, type int** specified log level in FEniCS, default is *50*, most levels refer to the ones of the standard Python library *logging*:

  10 or "DEBUG", subdry 13 or "TRACE", what's happening (in detail) 16 or "PROGRESS", what's happening (broadly) 20 or "INFO", information of general interest 30 or "WARNING", things that may go boom later 40 or "ERROR", things that go boom 50 or "CRITICAL", ciritcal errors between go boom and beyond

solve_main_problem(*solver='MUMPS', bc=None, sym=True, out_of_core=False, convert=True, method='cg', pc='ilu', delete_factorization=True, auto_interpolate=False, calculation_frequencies=[], secondary_potential=False, interpolation_quantities=['E_t', 'H_t'], **post_proc_kwargs*)

Assemble and solve linear systems of equations. Some arguments are piped to the **PostProcessingTD** or **PostProcessingFD** classes with the *post_proc_kwargs* dictionary during the post-processing phase, if automated postprocessing is enabled (*convert=True*). For the documentation of valid keyword arguments for the *PP.convert_results()* method, it is referred to the post-processing classes.

- **solver = 'MUMPS', type int** use direct solver MUMPS, no alternatives so far
- **bc = None, type str** default boundary conditions (BC) are implicit Neumann bc; alternatively, **'ZeroDirichlet'**, short **'ZD'**, can be choosen or inhomogeneous Dirichlet conditions **'ID'** in case of the frequency-domain *E-field* approach
- **sym = 1, 2 or True, type int or bool** so far, all implemented systems of equations are symmetric, this parameter may be **False** in other FE formulations 1 stands for positive definite symmetric matrices. 2 stands for gerneral structurally symmetric matrices
- **out_of_core = False, type bool** set **True** to enable *'out_of_core'* option of direct solver MUMPS if main memory is exceeded. WARNING! may be REALY slow
- **convert = True, type bool** automatically converts solution function *U* to real and imarginary parts of electric and magnetic fields and saves 'xml' and 'pvd' files. If **False**, functions *convert_results()* and *export_all_results* in the postprocessing instance *PP* can be called manually with custom conversion and export parameters
- **delete_factorization = True, type bool** if **False**, keep solver instance with factorization of main system for later usage; usually not required
- **calculation_frequencies = [], type list of int** only compute results for frequencies specified in this list; useful to recompute reminaing results after a random MUMPS crash
- **auto_interpolate = False, type bool** set **True** to automatically interpolate solutions in parallel after the solution and conversion phase has been finished for a frequency (useful in multi-frequency simulations)
- **interpolate_quantities = ['E_t', 'H_t'], type list of str** list of quantities for parallel interpolation on the fly, choose from *'E_t'*, *'H_t'*, *'E_s'*, or *'H_s'*
- **method = tfqmr, type str** if iterative solvers are considered in future, choose solver here
- **pc = ilu, type str** if iterative solvers are considered in future, choose preconditioner here
- **secondary_potential = False, type bool** set True to use secondary potential formulation for calculating DC fields, only relevant for **DC** approach, can also be specified before by build_var_form() call
- **post_proc_kwargs** pipe post-processing keyword arugments to post-processing instance if automated conversion and export is enabled

custEM.core.post_proc_fd module

@author: Rochlitz.R

class custEM.core.post_proc_fd.**PostProcessingFD**(*FE*, *export_domains*)
Bases: object

Frequency-domain PostProcessing class for computing and exporting electric and magnetic fields, called from MOD instance.

- **convert_results()** convert main solution to other field quantities, e.g., E to H
- **export_E_fields()** export total and/or secondary electric field data
- **export_H_fields()** export total and/or secondary magnetic field data
- **export_A_fields()** export total and/or secondary potential 'field' data
- **export_all_results()** export all available EM-field and/or potential data
- **save_field()** utility function for writing data on hard drive
- **convert_H_to_E()** convert magnetic fields obtained with magnetic field or potential approaches to electric fields
- **dc_post_proc()** conduct post-porcessing for direct current DC modeling approach

convert_H_to_E(*from_potential=False*)
Compute electric on basis of magnetic fields if enabled.

convert_results(*fi=0*, *convert_to_H=True*, *convert_to_E=False*, *export_cg=False*, *export_pvd=True*, *export_nedelec=True*, ***dummy_kwargs*)
Automatically convert results from E-fields to H-fields or vice versa or from potentials to E or H, depending on the utilized approach.

- **fi = 0, type int** integer specifiying to which frequency the current solution belongs
- **convert_to_H = True, type bool** calculate H-fields from E-field or potential approaches
- **convert_to_E = True, type bool** calculate E-fields, if H-field or F-U approach was used
- **export_cg = False, type bool** set **True** for exporting calculated data on Lagrange spaces
- **export_pvd = True, type bool** set **True** to export fields in *.pvd* format for *Paraview*
- **export_nedelec = True, type bool** set **True** for exporting calculated data on a Nedelec spaces

dc_post_proc(*export_cg=True*, *export_pvd=True*, *export_nedelec=True*)
Post-processing for direct-current (DC) approach.

- **export_cg, type bool** specify if fields in data format should be exported directly after conversion
- **export_pvd, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a Nedelec space

export_A_fields(*export_pvd*, *export_cg*, *export_nedelec*)
Export potentials using specified data (*xml/h5*) and/or *pvd* format from either Nedelec space or Lagrange VectorFunctionSpace.

- **export_cg, type bool** specify if fields in data format should be exported directly after conversion
- **export_pvd, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a Nedelec space

export_E_fields(*export_pvd*, *export_cg*, *export_nedelec*)
Export electric fields using specified data (*xml/h5*) and/or *pvd* format from either Nedelec space or Lagrange VectorFunctionSpace.

- **export_pvd, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_cg, type bool** specify if fields in data format should be exported directly after conversion
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a NedelecSpace

export_H_fields(*export_pvd*, *export_cg*, *export_nedelec*)
Export magnetic fields using specified data (*xml/h5*) and/or *pvd* format from either Nedelec space or Lagrange VectorFunctionSpace.

- **export_pvd, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_cg, type bool** specify if fields in data format should be exported directly after conversion
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a Nedelec space

export_all_results(*quantities='EAH'*, *export_cg=False*, *export_pvd=True*, *export_nedelec=True*)
Export a selection of calculated electric, magnetic or potential fields using specified data (*xml/h5*) and/or *pvd* format.

- **quantities = None, type str** if **export_all** is False, specify which data should be exported, use a combination of **E**, **H** and/or **A** combined in one string
- **export_cg = False, type bool** specify if fields in data format should be exported directly after conversion
- **export_pvd = True, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a Nedelec space

save_field(*quant*, *q1*, *q2*, *export_pvd*, *export_cg*, *export_nedelec*)
Store fields on hard disc drive.

- **quant, type str** string specifying the field quantitiy to be exported
- **q1, q2, type dolfin Function** functions containing the results
- **export_cg, type bool** specify if fields in data format should be exported directly after conversion
- **export_pvd, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a Nedelec space

custEM.core.post_proc_td module

@author: Rochlitz.R

class custEM.core.post_proc_td.**PostProcessingTD**(*FE*, *export_domains*)
Bases: object

Time-domain PostProcessing class for computing and exporting electric and magnetic fields, called from MOD instance. Note that interpolated results are currently only stored as *numpy* arrays. A method to export time-dependent solutions as *.pvd* file for *Paraview* might be implemented in future if required.

- **convert_results()** convert frequency-domain solutions of **FT** approach to time domain, derive *H* from *E* in **IE** and **RA** approaches
- **export_td_data()** save total electric and/or magnetic fields on hard disc drive
- **interpolate_frequency_solutions()** interpolate frequency-domain results in **FT** approach

- **reorder_results()** reorder interpolated frequency-domain results in **FT** approach
- **merge_td_results()** combine all interpolated time-domain results in a common data structure and export all time-domain results of an interpolation mesh as *numpy* arrays (pvd files not supported yet)

convert_results(*convert_to_H=True*, *export_cg=False*, *export_pvd=False*, *export_nedelec=True*, *interp_mesh=None*, *interpolate_fd=True*, *reorder_fd=True*, *transform_td=True*)
Automatically convert results from E-fields to H-fields.

- **convert_to_H = True, type bool** calculate H-fields from E-fields
- **export_cg = False, type bool** set **True** for exporting calculated data on Lagrange spaces
- **export_pvd = True, type bool** set **True** to export fields in *.pvd* format for *Paraview*
- **export_nedelec = True, type bool** set **True** for exporting calculated data on a Nedelec spaces
- **interp_mesh = None, type str** specify interpolation mesh for automated **FT** approach post-processing
- **interpolate_fd = True, type bool** enable automated interpolation of frequency-domain results in **FT** approach
- **reorder_fd = True, type bool** enable automated reordering of interpolated frequency-domain results in **FT** approach
- **transform_td = True, type bool** enable automated transformation of reordered frequency-domain results in **FT** approach

export_td_data(*export_cg*, *export_pvd*, *export_nedelec*)
Export electric and or magnetic fields using specified data (*xml/h5*) and/or *pvd* format from Nedelec spaces.

- **export_pvd, type bool** specify if fields are also exported in *.pvd* format for *Paraview*
- **export_cg, type bool** specify if fields in data format should be exported directly after conversion
- **export_nedelec = True, type bool** set **True** for exporting caluclated data on a NedelecSpace

interpolate_frequency_solutions(*interp_mesh*, *EH*)
Interpolate frequency-domain data on specified interpolation mesh.

- **interp_mesh, type str** target interpolation mesh
- **EH, type str** by default, EH='EH', that means that both, electric and magnetic field results, are interpolated

merge_td_results(*interp_mesh*, *EH='EH'*, *interp_dir=None*)
Resort time-dependent results on interpolation meshes in a common data format for all time-domain modeling approaches and export data as *numpy* arrays.

- **interp_mesh, type str** target interpolation mesh
- **EH, type str** by default, EH='EH', that means that both, electric and magnetic field results, are sorted and exported

reorder_results(*interp_mesh*, *EH*)
Reorder frequency-domain data station-wise by importing all results and storing them in one file per station, containing the values for all chosen frequencies.

- **interp_mesh, type str** target interpolation mesh
- **EH, type str** by default, EH='EH', that means that both, electric and magnetic field results, are reordered

custEM.core.pre_proc module

@author: Rochlitz.R

class custEM.core.pre_proc.**PreProcessing**(*FS*, *MP*, *import_freq=None*, *field_selection='all'*, *self_mode=False*, *fs_type='None'*)

Bases: object

PreProcessing class called from MOD instance.

- **import_all_results()** import results of an existing model - all quantities
- **import_selected_results()** import results of an existing model - selected quantities
- **load()** basic import function for data files
- **read_h5()** utility import function for **h5** data files

import_all_results(*file_format*)

Load all existing quantities from already calculated solutions.

- **file_format, type str** either 'h5' or 'xml', set within model paramers in **MOD** instance

import_selected_results(*selection*, *file_format*)

Load selected quantities from already calculated solutions.

- **selection, type str** string that must contain a combination of **E_t**, **E_s**, **H_t**, **H_s**, **A_t** or **A_s** added to a continuous string
- **file_format, type str** either 'h5' or 'xml', set within model paramers in **MOD** instance

load(*quantity*, *file_format*, *switch*, *ri='data'*)

Import data in either *xml* or *h5* format.

- **quantitiy, type str** quantitiy, e.g., **E_t_real_cg**, which should be imported
- **file_format, type str** either 'h5' or 'xml', set within model paramers in **MOD** instance
- **switch, type int** internally used switch, either 1 to use Lagrange VectorFunctionSpace or 2 for Nedelec space

-ri = 'data', type str specify a function to be imported from the HDF5 data container, if it contains multiple results; used internally to distinguish between *real* and *imag* fields

read_h5(*f_name*, *switch*, *ri*)

Read data files in 'h5' format.

- **fname, type str** file name to import from, containing also the export path
- **switch, type int** internally used switch, either 1 to use Lagrange VectorFunctionSpace or 2 for Nedelec space

-ri = 'data', type str specify a function to be imported from the HDF5 data container, if it contains multiple results; used internally to distinguish between *real* and *imag* fields

custEM.core.solvers module

@author: Rochlitz.R

class custEM.core.solvers.**Solver**(*FS*, *FE*, *mumps_debug=False*, *serial_ordering=False*)
Bases: object

Solver class called internally from MOD instance.

- **solve_var_form_default()** solve variaitonal formulation using default FEniCS solver, no support of symmetry, memory limits
- **solve_system_default()** solve assembled system of equations with default FEniCS solver, no support of symmetry, memory limits
- **solve_system_mumps()** use MUMPS solver to solve system of equations, no memory limits and symmetry support
- **call_mumps()** call MUMPS solver within *solve_system_mumps()* method
- **solve_var_form_iter()** use iteative solvers to solve variational formulation, in development
- **solve_system_iter()** use iteative solvers to solve system of equations, in development

- default choice of boundary conditions are implicit Neumann BC
- for zero Dirichlet BC, change *bc* argument in the 'solve' functions or during assembly for all total field approaches

call_mumps(*A*, *b*, *u*, *sym*, *out_of_core*, *first_call=True*)
Call MUMPS solver.

- **A = left-hand-side matrix, type PETScMatrix** assembled LHS matrix
- **b = right-hand-side vector, type PETScVector** assembled RHS vector
- **u = solution vector, type PETScVector** empty solution vector to be filled
- see the definitions of the *solve_system_mumps()* method for the description of the other input parameters

solve_system_default(*A*, *b*)
Use default petsc_lu solver to solve an assembled system of equations.

- **A = left-hand-side matrix, type PETScMatrix** assembled LHS matrix
- **b = right-hand-side vector, type PETScVector** assembled RHS vector

solve_system_iter(*A*, *b*, *sym=False*, *method='gmres'*, *pc='petsc_amg'*, *abs_tol=1e-06*, *rel_tol=1e-06*, *maxiter=2000*)
Use Krylov solver to solve an assembled linear system of equations.

- **A = left-hand-side matrix, type PETScMatrix** assembled LHS matrix
- **b = right-hand-side vector, type PETScVector** assembled RHS vector

- **sym = True, type bool** set to **False** if an unsymmetric system of equations is solved
- **method = 'gmres', type str** iterative solution method
- **pc = 'petsc_amg', type str** prconditioning type
- **abs_tol = 1e-6, type float** absolute tolereance
- **rel_tol = 1e-6, type float** relative tolerance
- **maxiter = 2000, type int** maximum number of iterations

solve_system_mumps(*A*, *b*, *fs=None*, *sym=False*, *out_of_core=False*, *first_call=True*, *tx_selection=None*)

Use MUMPS solver to solve an assembled linear system of equations.

- **A = left-hand-side matrix, type PETScMatrix** assembled LHS matrix
- **b = right-hand-side vector, type PETScVector** assembled RHS vector
- **fs = None, type str or dolfin FunctionSpace** define FunctionSpace for solution function(s)
- **sym = False, type int or bool** set to **1** for positive definite symmetric matrix set to **2** or **True** for general structurally symmetric matrix
- **out_of_core = False, type bool** set to **True** if MUMPS should be allowed to use disc space for the solution process if the memory is completely exploited
- **tx_selection = None, type list of int** specify if system should be solved for not all but only selected right-hand sides; so far used to solve DC system only for grounded Tx if there are also ungrounded ones in the same model
- **first_call = True, type bool** specify if this is the first call of a solution based on the same system matrix, re-use factorization if first_call=**False**

solve_var_form_default(*L*, *R*, *U*, *bc=None*)

Use default petsc_lu solver to solve a LinearVariationProblem.

- **L = left-hand-side, type UFL form** not assembled! UFL form of the left-hand-side of the system of equations
- **R = right-hand-side, type UFL form** not assembled! UFL form of the right-hand-side of the system of equations
- **U = Solution space, type dolfin FunctionSpace** mixed solution FunctionSpace
- **bc = None, type str** so far *ZeroDirichlet* or short *ZD* bc and implicit Neumann bc by using *None* are supported

solve_var_form_iter(*L*, *R*, *U*, *bc*, *sym=False*, *method='gmres'*, *pc='petsc_amg'*, *abs_tol=1e-06*, *rel_tol=1e-06*, *maxiter=2000*)

Use Krylov solver to solve a LinearVariationProblem.

- **L = left-hand-side, type UFL form** not assembled! UFL form of the left-hand-side of the system of equations
- **R = right-hand-side, type UFL form** not assembled! UFL form of the right-hand-side of the system of equations
- **U = Solution space, type dolfin FunctionSpace** mixed solution FunctionSpace
- **bc = boundary conditions, type str** so far **'ZeroDirichlet'** or short **'ZD'** bc and implicit Neumann bc by using **None** are supported
- **sym = False, type bool** set to **True** if a symmetric system of equations is solved
- **method = 'gmres', type str** iterative solution method
- **pc = 'petsc_amg', type str** prconditioning type
- **abs_tol = 1e-6, type float** absolute tolereance
- **rel_tol = 1e-6, type float** relative tolerance
- **maxiter = 2000, type int** maximum number of iterations

Module contents

core

Submodules:

- **model_base** for defining main model class
- **pre_proc** for defining pre-processing class
- **post_proc_fd** for defining frequency-domain post-processing class
- **post_proc_td** for defining time-domain post-processing class
- **solver** for solving linear systems of equations

---

custEM.fem package

Submodules

custEM.fem.assembly module

@author: Rochlitz.R

class custEM.fem.assembly.**MTAssembler**(*FE*, *bc*)
Bases: object

Class that assembles the system matrix and the right-hand-sides (two source polarizations implemented as boundary conditions on the top

of the model domain) for natural-source modeling.

- **assemble()** assembly function of left- and right-hand sides for secondary-field formulations

assemble(*fi=0*)

class custEM.fem.assembly.**SecondaryFieldAssembler**(*FE*, *bc*)
Bases: object

Class that assembles the system matrix and the right-hand-sides for secondary-field formulations based on primary fields and delta_sigma.

- **assemble()** assembly function of left- and right-hand sides for secondary-field formulations

assemble(*fi=0*)
Let dolfin assemble the system matrix and right-hand side vectors.

- **fi = 0, type int** iteration index over frequencies for specifying the left-hand-side

class custEM.fem.assembly.**TotalFieldAssembler**(*FE*, *bc*, *td_dc=False*, *force_p1=False*)
Bases: object

Class for setting the assembled contribution of the current density on transmitter dof in the right-hand side vector for total field formulations.

- **assemble()** general assembly function of left- and right-hand sides for total-field formulations; calls all the other methods

- **find_dof_coords()** utility function searching for all Nedelec dofs and whose coordinates which belong to the source defined on edges
- **find_cells()** for each dof which belongs to the source, search for an element which contains the former
- **find_directions()** evaluate the local direction of the Nedelec dof in the element which contains it and switch the signto match the global direction
- **find_and_fill_start_stop_dofs()** required for the A_Phi approach on mixed elements to set divergence term on the start and stop point of grounded wire sources
- **find_for_hed(), find_for_vmd()** utility functions called by **find_dof_coords()**
- **find_for_line(), find_for_loop_r(), find_for_path()** utility functions called by **find_dof_coords()**
- **append_lists()** utility function to collect the transmitter dof
- **eval_target_dir()** utility function to find the global direction of an edge
- **test_An_total(),** method for testing total A-V formulation on nodal elements
- **add_topo(),** modify z-values of coordinates according to specified topography

add_topo(*tx*, *tx_topo=None*)

Apply topography information to the z-values of the transmitter coordinates.

- **tx, type list of coordinates** list of transmitter coordinates
- **tx_topo = None, type str or function** user might specify a custom topography reference here, not recommended, only implemented for tests

append_lists(*idx*, *tdir=1.0*, *comp=0*)

Utility function appending dof, coordinates, directions and component switches to lists during the dof-coordinate search.

- **idx, type list of int** integers specifying the transmitter dof
- **tdir, type float** either positive or negative *1*
- **comp = 0, type int** either 0 or 1 for x or y component

assemble(*fi=0*)

Conduct right-hand side assembly of source currents manually and let dolfin assemble the system matrix.

- **fi = 0, type int** iteration index over frequencies for specifying the left-hand-side

eval_target_dir(*a*, *b*)

Evaluate global direction of each transmitter segment.

- **a, b, type array/list of len 3** start and stop coordinates to find the global positive or negative target direction, counter-clockwise in a right-handed coordinate system

find_and_fill_start_stop_dofs(*bb*, *current*, *ti=0*, *fi=0*)

Required for the A-V approach on mixed elements and the DC approach to set divergence term on the start and stop point of grounded wire sources.

- **bb, dolfin function** assembled right-hand side function
- **current, type float,** source current for the corresponsing tx
- **ti = 0, type int** index of the corresponding tx

find_cells()
For each dof which belongs to the source, search for an element which contains this dof.

find_directions()
Evaluate the local direction of the Nedelec dof in the element, which contains it and switch the sign to match the global direction.

find_dof_coords()
Utility function searching for all Nedelec dofs on edges and the corresponding coordinates which belong to the transmitter.

find_for_hed()
Find source dof and coordinates for HED source.

find_for_line()
Find source dof and coordinates for "line" source.

find_for_loop_r()
Find source dof and coordinates for "loop_r" source.

find_for_path()
Find source dof and coordinates for arbitrary grounded or ungrounded transmitters described by a path of coordinates.

find_for_vmd()
Find source dof and coordinates for VMD source.

test_An_total(*fi*)
Experimental total A-Phi nodal approach, use with caution and contact the authors for advise.

custEM.fem.calc_primary_boundary_fields module

@author: Rochlitz.R

class custEM.fem.calc_primary_boundary_fields.**Pyhed_calc**(*mesh_name*, *results_dir*, *pf_type*, *file_format*)

Bases: object

Primary field calculation class which starts calculating primary fields with *pyhed*-internal multiprocessing from a serial process to ensure correct dof ordering.

calc_fields()

convertCoordinates(*gimli*, *dolfin*)
Input: arr1, arr2: two coordinate lists of same shape (n, 3) which contains the same coordinates but in a diffrent order. output: arr1_arr2, arr2_arr1: index arrays which converts coordinates from input1 to input2 and from input2 to input1.

read_h5(*var*, *fname*)

read_serial_calculation_parameters()

write_h5(*var*, *fname*)

custEM.fem.calc_primary_fields module

@author: Rochlitz.R

class custEM.fem.calc_primary_fields.**Pyhed_calc**(*mesh_name*, *results_dir*, *pf_type*, *file_format*)

Bases: object

Primary field calculation class which starts calculating primary fields with *pyhed*-internal multiprocessing from a serial process.

calc_fields()

convertCoordinates(*gimli*, *dolfin*)
input: arr1, arr2: two coordinate lists of same shape (n, 3) which contains the same coordinates but in a diffrent order. output: arr1_arr2, arr2_arr1: index arrays which converts coordinates from input1 to input2 and from input2 to input1.

read_h5(*var*, *fname*)

read_serial_calculation_parameters()

write_h5(*var*, *fname*)

custEM.fem.fem_base module

@author: Rochlitz.R

class custEM.fem.fem_base.**Domains**(*mesh*, *n_dom*, *domain_func=None*, *domain_vals=None*)
Bases: object

Domain class called internally from FunctionSpace instance.

- **add_anomaly()** DEPRECATED - should not be used anymore! Add an anomaly domain in the 'FEniCS-way', e.g. predefined anomaly instances from the file *misc/anomaly_expressions.py* or user-defined dolfin expressions can be added to the model, each anomaly gets a new marker

add_anomaly(*instance*)
DEPRECATED! Do not use anymore without contacting the authors. (from custEM.misc import anomaly_expressions as ae) (MOD.FS.DOM.add_anomaly(ae.Plate(origin=[0.0, -100.0, -400.0])))

class custEM.fem.fem_base.**FunctionSpaces**(*MP*, *p*, *test_mode*, *self_mode*, *ned_kind*, *dg_interp*, *load_existing*, *reuse_fs=None*, *overwrite_mesh=False*, *mesh_only=False*)

Bases: object

FunctionSpaces class called from MOD instance. Used for initializing the required function spaces for the FE assembly, boundary condtions, and primary fields for all secondary field formulations.

- **init_mesh()** import mesh and domain information from *.h5* file (default) in parallel or from *.xml* file in serial
- **add_primary_fields()** initialize PrimaryField instance, calculate or import primary fields
- **add_dirichlet_bc()** initalize dirichlet boundary conditions on the model boundary
- **build_dg_func()** build element-wise scalar parameter function (isotropic parameters)
- **build_dg_tensor_func()** build element-wise tensor parameter function (anisotropic parameters)
- **build_complex_sigma_func()** build complex-valued sigma functions (only isotropic) for modeling induced polarization effects

1. import mesh (see *init_mesh* documentation)
2. build FEniCS FunctionSpaces depending on the mesh, polynomial order **p** and the chosen approach

- **S = Nodal (LaGrange) FunctionSpace** order p and dimension 1
- **S_dg = element-wise scalar discontinuous FunctionSpace** order 0, 1 value per cell
- **S_dgt = element-wise tensor-valued discontinuous FunctionSpace** order 0, 9 values per cell
- **V = Nedelec Space** order p and dimension 3
- **V_cg = Nodal (LaGrange) VectorFunctionSpace** order p and dimension 3
- **W = mixed Nedelec Space** order p and dimension 6 for coupled real and imaginary parts
- **W_cg = mixed (LaGrange) VectorFunctionSpace** order p and dimension 6 for coupled real and imaginary parts
- **M = mixed solution FunctionSpace** order p and dimension 6 or 8 (dependent on appraoch)

3. initalize empty solution Function **U** on mixed solution space
4. if a secondary field formulation is used, a PrimaryField instance **PF** is added to the FunctionSpace class later on by calling the **add_Primary_Field()** method automatically during the **FE.build_var_form()** call

add_dirichlet_bc(*MP=None*, *dof_re=None*, *dof_im=None*, *bc_H=False*)
Initalize Dichichlet BoundaryConditions on the solution space, only for internal usage. The keyword arguments are set automatically to handle different cases.

- **MP = None, type class** only used to apply inhomogeneous Dirichlet conditions in A-V approach, which requires E-field conversion to A depending on *MP.omega*
- **dof_re = None, type list** list of boundary dof of "real" part of FE system to apply inhomogeneous Dirichlet conditions
- **dof_im = None, type list** list of boundary dof of "imaginary" part of FE system to apply inhomogeneous Dirichlet conditions
- **bc_H = False, type bool** flag for testing inhomogeneous Dirichlet conditions for H-field conversion, provides no benefits so far but might be re-considered for other purposes in future

add_primary_fields(*FE*, *MP*, *calc=True*, ***kwargs*)
Add a PrimaryField instance to the FunctionSpace instance, called by **FE** if the 'Secondary' field formulation is used.

- **FE, type class** FE approach instance (e.g., **E_vector**)
- **MP, type class** ModelParameters instance
- **calc = True, type bool** enable primary fields calculation or **PF** class initialization only
- **further kwargs** see full list in the description of the **PrimaryField** class

build_complex_sigma_func(*omega*, *sigma*, *tau*, *m*, *c*)
Calculate and assign complex-valued conductivities for simulating induced polarization effects with a Cole-Cole model discontinous function spaces. This only works for isotropic conductivities for now.

- **omega, type float** angular frequency
- **sigma, type list** list of conductivity values
- **tau, type list** list of *ip_tau* values, see **MP** class documentation
- **m, type list** list of *ip_m* values, see **MP** class documentation
- **c, type list** list of *ip_c* values, see **MP** class documentation

build_dg_func(*vals*, *inverse=False*)
Assign isotropic conductivity values to discontinous function spaces.

- **vals, type list (of lists)** conductivity values specified in **MP** instance
- **inverse = False, type bool** return either sigma or 1/sigma (=inv(sigma))

build_dg_tensor_func(*vals*, *inverse=False*, *logger=None*)
Assign anisotropic conductivity values to discontinous tensor function spaces.

- **vals, type list (of lists)** conductivity values specified in **MP** instance
- **inverse = False, type bool** return either sigma or inv(sigma)

init_mesh(*test_mode*, *MP*, *self_mode*, *overwrite_mesh*)
Import the requested mesh and identify domain markers, if available.

- see documentation of **MOD** class

class custEM.fem.fem_base.**ModelParameters**(*logger*, *str_args*, *test_mode*)
Bases: object

ModelParameter class called internally from MOD instance. This class contains, holds, and updates all physical and modeling parameters during the FE simulation. All model parameters can be updated with the **update_model_parameters()** method. A description of all available parameters of the **ModelParameters** class is listed below.

- **update_model_parameters()** update any supported physical or modeling parameter
- **calc_delta_sigma()** calculate *delta_sigma* as *sigma_ground - sigma_0* and take reshape lists to account for any anisotropic value descriptions
- **build_dg_parameter_functions()** build discontinuous Parameter functions for all physical parameters, single-valued functions for isotropic values and tensor-valued functions for anisotropic values
- **print_model_parameters()** print updated list of model parameters
- **load_mesh_parameteres()** update mesh-related model parameters from 'mesh-parameters' *json* file
- **check_parameter_dx_conformity()** check conformity of number of domains and number of physical parameter values
- **f = 10., type float** frequency (Hz)
- **omega = 1e1, type float** angular frequency, omega = 2 * pi * f
- **frequencies = [], type list** specify multiple frequencies (Hz)
- **omegas = [], type list** specify multiple angular frequencies, omegas = 2 * pi * frequencies
- **n_freqs = 1, type int** number of frequencies, calculated automatically
- **current = 1., type float** source current I (A) in the current carrying wire, variable *J* is deprecated but still used internally, *current* is the new alias for it
- **currents = [], type 1D or 2D array or list of lists** unique source currents I (A) for multiple Tx or multiple frequencies; In time-domain and for DC modeling, the first dimension of the current data array corresponds to n_tx; in frequency-domain modeling, the first dimension of the current data array corresponds to n_freqs and the second dimension of the current data array correponds to n_tx; if not specifed, current = 1. is used for all frequencies and Tx
- **m_moment = 1., type float** magnetic dipole moment for calculating background fields of a vmd source in a fullspace analytically

- **n_tx = 1, type int** number of transmitters (right-hand sides of the system of linear equations), calculated automatically
- **sigma_air = 1e-8, type float or list** homogeneous air-space conductivity, for modeling a fullspace, set sigma_air to the same value as sigma_ground
- **sigma_ground = 1e-2, type list of float or list of list of floats** subsurface conductivities corresponding to all domain markers (except 0 which corresponds to *sigma_air*); anisotropy can be specified by a list of three values (main diagonal) or a list of six values (upper triangle of 3x3 conductivity tensor) for each domain; a list of one value is handled identical to providing this single value as float instead of embedding it into a list
- **sigma_0 = [], type list of float or list of list of floats** background condcutivities for all *sigma_ground*, only required for secondary-field/potential formulations; arbitrary general anisotropic values are supported for *sigma_ground* and *sigma_0*; *delta_sigma* is evaluated internally
- **sigma = [1e-2, 1e-8], type list of float** list of all conductivities, the first two values in sigma are always **sigma_ground** (first layer) and **sigma_air**; if these values are updated later on, sigma will be updated internally; further values are related to subsurface layers or anomaly-domains in the model (see **sigma_anom**)
- **delta_sigma = [0., 0.], type list of float** conductivity differences, needed for secondary field formulations, updated internally
- **anomaly_layer_markers = [], type list of int** define the reference subsurface layer (from top to depth) for each anomaly to calculate **delta_sigma**. For instance, if two anomalies are located in the first and third layer of a three-layered earth, the correct choice would be *[1, 3]*; used for secondary field approaches only, this functionality will be updated by implementing an automated mapping in future
- **mu = mu_0 = 4 * pi * 1e-7, type float** magentic permeability (mu_r * mu_0), default is vacuum permeability
- **mu_r = 1., type float or list of floats** relative magnetic permeability
- **eps = eps_0 = 8.85 * 1e-12, type float** electric permittivity (eps_r * eps_0), default is vacuum permittivity
- **eps_r = 1., type float or list of floats** values of relative electric permittivity
- **ip_tau = 1., type float or list of floats** *tau* values of Cole-Cole model for modeling induced polarization effects
- **ip_c = 1., type float or list of floats** *c* values of Cole-Cole model for modeling induced polarization effects
- **ip_m = 0., type float or list of floats** *m* values of Cole-Cole model for modeling induced polarization effects
- **n_layers = 1, type int** number of subsurface layers, automatically imported from mesh-parameter file
- **layer_depths = [], type list of float** depths of subsurface-layer interfaces, automatically imported from mesh-parameter file
- **layer_thicknesses = [], type list of float** subsurface layer thicknesses, calculated automatically from **layer_depths** attribute
- **topo = None, type str** definition of DEM- or synthetic topography, imported automatically from mesh-parameter file.

build_dg_parameter_functions(*FS*, *pf_EH=None*, *eps=False*)

Build discontinuous isotropic or anisotropic parameter functions.

calc_delta_sigma()
Calculate delta sigma using *sigma_ground* and the corresponding background conductivities *sigma_0*.

check_parameter_dx_conformity()
Evaulate, if number of domains matches number of given physical parameter values.

load_mesh_parameters()
Import mesh parameters such as the number of domains or topography information.

print_model_parameters()
Print model parameters with adjusted style, e.g., no lengthly lists.

update_model_parameters(***mod_kwargs*)
Updates all non-default physical parameters if called. A list of available parameters is given in the documentation of the **ModelingParameters** class.

custEM.fem.fem_utils module

@author: Rochlitz.R

class custEM.fem.fem_utils.**ApproachBaseFD**
Bases: object

Initialize FE variables for all frequency-domain approaches. All variables can be updated by specifying the corresponding keyword arguments in the *build_var_form()* method.

- **update_fem_parameters()** update valid parameters of the **FE** and **PF** classes and print updated class attributes
- **update_tx_parameters()** if *s_type='auto'*, update Tx information from imported mesh-parameter file in the **MP** instance
- **s_type = 'auto', type str** alternatives: **hed** (HED source, treated identical to **line** tx in halfspace and as infinitesimal source in fullspace with analytical solution)

  vmd (VMD source, treated as infinitesimal source in fullspace with analytical solution)

  heds (multiple HED sources, treated as infinitesimal sources in fullspace with analytical solution; specify origins with a list of 3D coordinated in the *tx* variable)

  vmds (multiple VMD sources, treated as infinitesimal sources in fullspace with analytical solution; specify origins with a list of 3D coordinated in the *tx* variable)

  line (real finite length dipole source) **loop_c** (circular loop source) **loop_r** (rectangular loop source) **path** (arbitrary sources)
- **tx = None, type list** internal usage only; if *s_type=auto*, this variable will be updated by the information in the mesh-parameter file; otherwise, the geometry parameters will be used to define the tx
- **n_tx = 1, type int** number of transmitters (right-hand sides of system of linear equations)
- **grounding = False,** internal usage only and only relevant if *s_type=auto* (this variable will be updated by the information in the mesh-parameter file)
- **origin = [0.0, 0.0, 0.0], type list of len (3)** origin of *hed* or *vmd* Tx; only valid if *s_type* is not *auto* and *n_tx=1*
- **length = 1., type float** length of *hed* Tx; only valid if *s_type* is not *auto* and *n_tx=1*

- **self.azimuth = 0., type float** azimuth of *hed* or *vmd* Tx in degree; only valid if *s_type* is not *auto* and *n_tx=1*
- **start = [-100, 0.0, 0.0], type list of len(3)** start coordinate of *line* Tx or lower left corner of *loop_r* Tx; only valid if *s_type* is not *auto* and *n_tx=1*; z_value is ignored and always set to zero
- **stop = [100.0, 0.0, 0.0], type list of len(3)** stop coordinate of *line* Tx or upper right corner of *loop_r* Tx; only valid if *s_type* is not *auto* and *n_tx=1*; z_value is ignored and always set to zero
- **radius = 100., type float** radius of *loop_c* Tx; only valid if *s_type* is not *auto* and *n_tx=1*
- **pf_EH_flag = 'E', type str** set to 'H' for using primary magentic fields to calculate the right-hand side source contributions in secondary field formulations
- **bc = None, type str** specify alternative boundary conditions (see **FS** class)
- **quasi_static = True, type bool** use diffusive (quasi static) approximation of the Helmholtz equation without displacement currents, *False* only implemented in the *E_vector* approach
- **ip = False, type bool** set *True* to enable modeling of induced polarization effects by specifying Cole-Cole model parameters (see **MP** class), only implemented in the *E_vector* approach
- **tx_topo = None, type str or function** specify topography (from file or as function z=f(x, y)) for searching the vertical Tx coordinates on a manually chosen topography for the total field approaches, not recommendend to be changed
- **a_tol = 1e-2, type float** find correct source dof locations for total field formulations; might be changed but the default choice is almost always sufficient
- **assembly_time = 0., type float** used internally, adds all assembly times to this variable during the to be stored by the *profiler* later on

update_fem_parameters(*fem_kwargs*)

Update and print **FE** and **PF** class attributes for frequency-domain approaches by overwriting the corresponding keyword arguments.

fem_kwargs, type dict dictionary of keyword arguments for *build_var_form()* method

update_tx_parameters()

Update Tx information from imported mesh-parameter file in the **MP** instance if *s_type='auto'*.

class custEM.fem.fem_utils.**ApproachBaseTD**

Bases: object

Initialize FE variables for all time-domain approaches. All variables can be updated by specifying the corresponding keyword arguments in the *build_var_form()* method.

- **update_fem_parameters()** update valid parameters of the **FE** and **PF** classes and print updated class attributes
- **update_tx_parameters()** if *s_type='auto'*, update Tx information from imported mesh-parameter file in the **MP** instance
- **calc_static_magnetic_field()** calculate static magnetic field with the Biot-Savart law
- **biot_savart()** vectorized implementation of Biot-Savart law
- **interpolate_static()** interpolate static magentic field on specified interpolation mesh for independent post-processing purposes
- **calc_dc_field()** calculate direct current electric field with either total or secondary potential formulation of the common potential approach

- **interpolate_dc()** interpolate direct current field on specified interpolation mesh for independent post-processing purposes
- **export_interpolation_data()** export interpolated static magentic field or direct current electric field results as numpy files and, if specified, in *.pvd* file format

Here, only parameters relevant for the time-domain approaches are listed. For all other parameters, it is referred to the documentation of the **ApproachBaseFD** class.

Note that further relevant parameters that can be updated with the *build_var_form_method()* are initialized by each time-domain approach class individually. It is referred to the corresponding documentation in the *time_domain_approaches.py* file.

- **log_t_min = -6, type int/float** specify start observation time of logarithmically divided interval for computing transients; defines the start time as 10^log_t_min
- **log_t_max = 0, type int/float** specify stop observation time of logarithmically divided interval for computing transients; defines the stop time as 10^log_t_max
- **n_log_steps = 61, type int** number of steps to logarithmically divide the time interval of the transient, equal to the number of results exported at the corresponding times
- **times = None, type list** manually define export times of the transients
- **shut_off = True, type bool** calculate either transient transmitter shut-off of shut-on response

biot_savart(*rx*, *tx*)
Vectorized implementation of the Biot-Savart law.

- **rx, type list of coordinates** receiver coordinates
- **tx, type list of coordinates** list of transmitter coordinates describing the transmitter path

calc_dc_field(*dc_mesh=None*, *secondary_potential=False*, *store=False*, *td_dc=True*)
Calculate direct current electric fields with either secondary or total field formulation of potential approach for obtaining shut-off transients.

- **secondary_potential = False, type bool** use either total or secondary potential formulation
- **store = False, type bool** save static magnetic fields as HDF5 file

calc_static_magnetic_field(*store=False*)
Calculate static magentic fields with the Biot-Savart law.

- **store = False, type bool** save static magnetic fields as HDF5 file

export_interpolation_data(*field*, *V_serial*, *ti*, *export_pvd*, *interp_mesh*)
Export interpolated results as complex numpy arrays and, if specified, as *.pvd* files to be viewed in Paraview.

- **field, type dolfin Function** dolfin function containing the field values
- **V_serial, type dolin FunctionSpace** non-distributed solution function space
- **ti, type int** number of the Tx to specify output file name
- **export_pvd, type bool** aside from numpy array, save interpolated fields also as *.pvd* file for Paraview
- **interp_mesh, type str** interpolation mesh name

interpolate_dc(*interp_mesh*, *interp_p=None*, *export_pvd=True*, *interp_dir=None*)
Customized interpolation function to interpolate direct current fields on specified interpolation mesh.

- **interp_mesh, type str** name of target interpolation mesh
- **interp_p = None, type int** default is *interp_p=1*, not reasonable to be changed

- **export_pvd = True, type bool** save fields as *.pvd* file for Paraview or not
- **interp_dir = None, type str** specify custom interpolation directory, not recommended

interpolate_static(*interp_mesh*, *interp_p=None*, *export_pvd=True*, *interp_dir=None*)
Customized interpolation function to interpolate static magnetic fields on specified interpolation mesh.

- **interp_mesh, type str** name of target interpolation mesh
- **interp_p = None, type int** default is *interp_p=1*, not reasonable to be changed
- **export_pvd = True, type bool** save fields as *.pvd* file for Paraview or not
- **interp_dir = None, type str** specify custom interpolation directory, not recommended

update_fem_parameters(*fem_kwargs*)
Update and print **FE** class attributes for time-domain approaches by overwriting the corresponding keyword arguments.

- **fem_kwargs, type dict** dictionary of keyword arguments for *build_var_form()* method

update_tx_parameters()
Update Tx information from imported mesh-parameter file in the **MP** instance if *s_type='auto'*.

class custEM.fem.fem_utils.**DC**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

Implementation of common potential approach for caluclating direct current (DC) electric fields.

- **build_var_form()** build variational formulation with FEniCS
- **lhs_form()** build left-hand side of variational formulation
- **rhs_form_total()** build right-hand side of variational formulation for secondary potential approach
- **rhs_form_secondary()** build right-hand side of variational formulation for total potential approach
- **calc_primary_potential()** calculate primary potential for 1D backround resistivity distribution analytically
- **calc_dc_field()** customoized method for solving the DC FE problem

Most parameters are equivalent to the parameters for the frequency-domain approaches in the **ApproachBaseFD**. Please refer to the corresponding documentaion. The only **DC** class relevant keyword argument to be updated with the *build_var_form()* method is listed below.

- **secondary_potential = False, type bool** use either total or secondary potential formulation

build_var_form(***fem_kwargs*)
Update FE parameters and build variational formulations for the matrix assembly. For the description of valid keyword arguments, it is referred to the **ApproachBaseFD** and **DC** class documentations.

calc_dc_field(*PP*, *profiler*, *secondary_potential=None*, *delete_factorization=True*)
Assemble the system matrix and right-hand sides and solve the resulting linear system of equations for either the total or secondary potential approach. The electric DC field is also derived after the solution of the potential FE problem.

- **secondary_potential = False, type bool** use either total or secondary potential formulation, can be updated during calling this method is required, but should be already specified before

calc_primary_potential()
Calculate primary potential of static current analytically.

lhs_form()
Build left-hand side of variational formulation for DC approach.

rhs_form_secondary()
Build right-hand side of variational formulation for secondary DC potential approach.

rhs_form_total()
Build right-hand side of variational formulation for total potential approach.

custEM.fem.fem_utils.**add_sigma_vals_to_list**(*vals*, *inverse=False*)
Rearrange conductivity values to appropriate list format, e.g., convert isotropic values given as float to list of length 1.

- **vals, type list (of lists)** conductivity values specified in **MP** instance
- **inverse = False, type bool** return either sigma or 1/sigma (inv(sigma))

custEM.fem.fem_utils.**check_sigma_vals**(*MP*)
Evaulate, if anomalies are included or halfspace or fullspace model is chosen.

- **MP, type class** ModelParameters instance

custEM.fem.fem_utils.**check_source**(*s_type*, *tx*, *pf_flag='E'*, *approach=None*)
Evaulate, which source type is chosen.

- **s_type, type int** source type (see **ApproachBaseFD** documentation)
- **tx, type list of arrays** list of transmitter coordinates for each Tx
- **pf_flag = 'E', type str** either *'E'* or *'H'*

custEM.fem.frequency_domain_approaches module

@author: Rochlitz.R

class custEM.fem.frequency_domain_approaches.**A_V_mixed**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

FE implementation of mixed potential approach, see (Ansari, 2014) or a modified formulation by (Schwarzbach, 2009).

build_var_form(***fem_kwargs*)
Print FE parameters and initialize variational formulations for assembly. Valid keyword arguments are described in the **ApproachBaseFD** and **PrimaryFields** classes.

lhs_forms()
Left-hand side formulation as the basis for the system matrix assembly.

rhs_form_secondary(*fi=0*)
Right-hand side formulation for secondary potential field approach.

rhs_form_total()
Right-hand side formulation for total field approach. Right-hand side vector will be filled later by the **TotalFieldAssembler** instance.

class custEM.fem.frequency_domain_approaches.**A_V_nodal**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

FE implementation of nodal potential approach, see (Badea, 2001) or (Puzyref, 2013).

build_var_form(***fem_kwargs*)
Print FE parameters and initialize variational formulations for assembly. Valid keyword arguments are described in the **ApproachBaseFD** and **PrimaryFields** classes.

lhs_form()
Left-hand side formulation as the basis for the system matrix assembly.

rhs_form_secondary(*fi=0*)
Right-hand side formulation for secondary field approach.

rhs_form_total()
Right-hand side formulation for total field approach. Right-hand side vector will be filled later by the **TotalFieldAssembler** instance.

class custEM.fem.frequency_domain_approaches.**E_vector**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

FE implementation of E-field approach (Grayver, 2014, Schwarzbach 2009).

build_var_form(***fem_kwargs*)
Print FE parameters and initialize variational formulations for assembly. Valid keyword arguments are described in the **ApproachBaseFD** and **PrimaryFields** classes.

lhs_forms()
Left-hand side formulation as the basis for the system matrix assembly.

lhs_forms_full()
Left-hand side formulation as the basis for the system matrix assembly. Here, electric permittivities and induced polarization parameters are considered in addition to the conductivity values.

rhs_form_secondary(*fi=0*)
Right-hand side formulation for secondary electric field approach.

rhs_form_total()
Right-hand side formulation for total field approach. Right-hand side vector will be filled later by the **TotalFieldAssembler** instance.

class custEM.fem.frequency_domain_approaches.**F_U_mixed**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

FE implementation of Tau-Omega approach on mixed elements (Mitsuhata, 2004), called F-U potential approach in custEM.

build_var_form(***fem_kwargs*)
Print FE parameters and initialize variational formulations for assembly. Valid keyword arguments are described in the **ApproachBaseFD** and **PrimaryFields** classes.

lhs_forms()
Left-hand side formulation as the basis for the system matrix assembly.

rhs_form_secondary(*fi=0*)
Right-hand side formulation for secondary potential field approach.

rhs_form_total()
Right-hand side formulation for total field approach. Right-hand side vector will be filled later by the **TotalFieldAssembler** instance.

class custEM.fem.frequency_domain_approaches.**F_U_nodal**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

FE implementation of F-U (Tau-Omega) approach on nodal elements. This approach was not implemented yet as it requires a homogeneous conductivity distribution (at least in our derivation of the equations) in the complete computational domain. Hence, it is irrelevant for geo-EM applications.

class custEM.fem.frequency_domain_approaches.**H_vector**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

FE implementation of H-fild approach (Rochlitz, 2019).

build_var_form(*add_primary=True*, ***fem_kwargs*)
Print FE parameters and initialize variational formulations for assembly. Valid keyword arguments are described in the **ApproachBaseFD** and **PrimaryFields** classes.

lhs_form_full()
Left-hand side contributions for full (not quasi-static) H-field approach, in development, not fully tested yet.

lhs_forms()
Left-hand side formulation as the basis for the system matrix assembly.

rhs_form_secondary(*fi=0*)
Right-hand side formulation for secondary magentic field approach.

rhs_form_total()
Right-hand side formulation for total field approach. Right-hand side vector will be filled later by the **TotalFieldAssembler** instance.

custEM.fem.primary_fields module

@author: Rochlitz.R

class custEM.fem.primary_fields.**PrimaryField**(*FS*, *FE*, *MP*, ***pf_kwargs*)
Bases: *custEM.fem.fem_utils.ApproachBaseFD*

PrimaryField class for computing or loading existing primary fields for secondary field formulations. Called by *FE.build_var_form()* via the **FunctionSpace** class method *add_Primary_Field()*.

For the documentation of the the ***comet** toolbox (**pyhed** module), it is referred to the documentation of this software.

- **export_primary_fields()** save calcualted primary fields
- **load_primary_fields()** import existing primary fields; if not existing, start calculation
- **calc_hed_fullspace_field()** calculate analytic primary-field solution for a HED in a fullspace
- **calc_vmd_fullspace_field()** calculate analytic primary-field solution for a VMD in a fullspace
- **calc_layered_earth_field()** calculate semi-analytic primary-field solutions for layered earth geometries with *pyhed*
- **init_primary_field_name()** generate an encrypted primary field data file name based on the Tx configuration to avoid recomputation of exisiting indentical setups (same mesh, same Tx geometry, same physical parameters)
- **write_serial_calculation_parameters()** store **PF** class parameters in a file for the reimport of serial calculation scripts

Note that further relevant parameters are initialized by the **ApproachBaseFD** class. They are also valid keyword arguments of the *build_var_form()* method and can be changed accordingly.

- **max_length = None, type float** maximum length of the single HED segments used to approximate a finite-length bipole or loop transmitter; overwrites *n_segs* if both parameters are specified
- **n_segs = None, type int** total number of segments (single HED transmitters) for approximating finite-length bipole or loop transmitters; overwritten by *max_length* if both parameters are specified

- **pf_type = 'default', type str** primary field type, alternatives: **fullspace** or **custom**
- **pf_name = None, type str** an alternative name for the primary fields of the model can be defined; it is not recommended to change the default name
- **export_pvd = False, type bool** flag controlling if 'pvd' - files are exported (**'True'**) or not
- **eval_tx = False, type bool** evaulate Tx coordinates on corresponding edge-midpoints between nodes and use them for the primary field calculation instead of the node coordinates, also provides edge lengths and orientations
- **force_primary = False, type bool** force re-caculation of exisiting primary field with an identical parametrization
- **pyhed_interp = False, type bool** set True, if the alternative interpolation-based primary field calculation technique of the *pyhed*-module should be used
- **pf_EH_flag = 'E', type str** either **E** or **H** to specify, which type of primary fields should be used for the FE caluclations later on
- **procs_per_proc = 1, type int** change, if more processes should be used to calculate primary fields with 'pyhed' than specified via the mpirun command (e.g., -n 12) for sloving the main FEm problem. For instance, a value of **3** would lead to 36 processes used for the primary field calculation.
- **pf_p = 1, type int** polynomials order of Lagrange VectorFunctionSpace which is used to calculate the primary fields; could be changes to *2*, but this only leads to significantly increased primary field computation times and barely more accurate results
- **ph_log_level = 0, type int** log level of the pyhed software
- **pyhed_drop_tol = 1., type bool** internal drop tolerance for approximating the semi-analytical results at positions with singular fields values
- **dipole_z = 0., type float** shift z-value of source to the subsurface to enable calculations of primary fields for marine setups; use with caution as this option was not tested sufficiently
- **Assembler = None, type class** *Assembler* class that might be passed to the **PF** class for custom purposes

calc_hed_fullspace_field(*V_cg*)
Calculate analytic primary field solution for a HED in a fullspace.

- **V_cg, type dolfin VectorFunctionSpace** dolfin Lagrange vector function space

calc_layered_earth_field(*V_cg*, *boundary_fields*, *ti*)
Calculate semi-analytic primary field solution with the *pyhed* software for layered-earth geometries.

- **V_cg, type dolfin VectorFunctionSpace** dolfin Lagrange vector function space
- **boundary_fields, type bool** only for internal usage, used to calculate primary fields only on the domain boundaries to apply inhomogeneous Dirichlet BC
- **ti, type int** number of the Tx if multiple Tx are specified

calc_mt_fullspace_field(*V_cg*)
Calculate analytic primary field solution for a HED in a fullspace.

- **V_cg, type dolfin VectorFunctionSpace** dolfin Lagrange vector function space

calc_vmd_fullspace_field(*V_cg*)
Calculate analytic primary field solution for a VMD in a fullspace.

- **V_cg, type dolfin VectorFunctionSpace** dolfin Lagrange vector function space

export_primary_fields()
Saves calculated primary fields in dedicated HDF5 or *.pvd* files.

init_primary_field_name(*boundary_fields*)
Initialize an encoded, unique name for the files to store primary fields based on the mesh, transmitter characteristics and others.

- **boundary_fields = False, type bool** only for internal usage, used to calculate primary fields only on the domain boundaries to apply inhomogeneous Dirichlet BC

load_primary_fields(*boundary_fields=False*, *fi=0*)
Import existing or start calculation of primary fields.

- **boundary_fields = False, type bool** only for internal usage, used to calculate primary fields only on the domain boundaries to apply inhomogeneous Dirichlet BC
- **fi = 0, type int** iteration index over frequencies

write_serial_calculation_parameters()
Export geometry and transmitter setup for seperate primary field calculations in serial.

- **path, type str** current working directory, switch to this directory after the export of the parameter file

custEM.fem.time_domain_approaches module

@author: Rochlitz.R

class custEM.fem.time_domain_approaches.**FourierTransformBased**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseTD*

FE implementation of inverse Fourier-transform based approach. It calculates several tens of frequency-domain solutions with the total E-field approach and converts them to time domain afterwards.

- **build_var_form()** set up attributes required in the frequency-domain variational formulations
- **initialize frequencies()** initialize frequencies for the frequency-domain computations automatically if not given explicitly as keyword argument
- **calc_frequency_solutions()** calculate frequency-domain solutions with total E-field approach
- **transform_to_time_domain()** transform frequency-domain solutions to time domain
- **fht_hahnstein_80()** use default set of 80 filter coefficients to perform the fast Hankel transformation
- **n_mult = 6, type int** multiplyer of poles, do not change
- **n_poles = 2, type int** number of poles to be used
- **pole_shift = None, type float** shift poles in time; the default covered range is about 4 to 5 decades from the earliest times on; this values is adjusted automatically and it is not recommended to change it
- **dc_field = None, type dolfin Function** the DC field is automatically calculated on the fly; for special purposes, it might be computed externally and passed here

build_var_form(***fem_kwargs*)
Initialize parameters for the Fourier-transform based approach and print FE parameters. A description of the keyword arguments is given in the **ApproachBaseTD** and **FourierTransformBased**classes.

calc_frequency_solutions(*auto_interpolate*, *repeat_frequencies*, ***solve_kwargs*)
Calculate frequency-domain solutions. Keyword arguments are piped to this method by the *build_var_form()* method.

fht_hanstein_80(*fd_data*, *modus*, *dc_level=None*)
Default fast Hankel transformation routine with 80 filter coefficients.

- **fd_data, type numpy array** frequency-domain data, only one component
- **modus, type int** modus switch to calculate either shut-on, shut-off or impulse response results

dc_level = None, type float used to apply explicitly a calculated DC level for shut-off E-field transients calculations; otherwise, the DC level will be approximated by the fields values of the lowest frequency

initalize_frequencies()
Initialize frequencies for the frequency-domain solutions automatically based on the default 80 FHT filter coefficients.

transform_to_time_domain(*interp_mesh*, *EH*, *shut_on=True*, *shut_off=True*, *impulse=True*)
Transform frequency-domain solutions to time domain using the fast Hankel transformation routine.

- **interp_mesh, type str** interpolation mesh for the chosen frequency-domain results which should be converted to time domain
- **EH, type str** default is *'EH'*, that means both, electric and magentic fields are converted. Users can choose only *'E'* or *'H'*.
- **shut_on = True, type str** transform to shut-on field transients or not if *False*
- **shut_off = True, type str** transform to shut-off field transients or not if *False*
- **impulse = True, type str** transform to impulse (dB/dt) repsonse transients or not if *False*

class custEM.fem.time_domain_approaches.**ImplicitEuler**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseTD*

FE implementation of Implicit Euler time-stepping method, e.g., (Um, 2010). Users can use either a first- or second-order discretization in time and, with cautioan, a formulation considering displacement currents which needs some more tests.

- **time_stepping()** runs the complete time-stepping solution process as well as H-field conversion and export methods, FE modeling and time stepping related keyword arguments can be passed to this method.
- **build_var_form()** set up variational formulation and assemble mass and curl-curl matrices as well right-hand side (source) vector
- **init_initial_condition()** initialize initial condition for time-stepping approach
- **first_order_IE()** perform first-order implicit Euler time-stepping approach
- **second_order_IE()** perform second-order implicit Euler time-stepping approach
- **first_order_IE_full()** perform first-order implicit Euler time-stepping approach including displacement currents
- **init_times()** initalize internally used time vector for the time stepping
- **gaussian()** define gaussian source pulse for impulse response modeling
- **print_stepping_info()** print some information before starting the time-stepping process
- **source_type = 'djdt', type str** alternatively, use *'j'* to calculate B-fields instead of dB/dt
- **source_func = None, type numpy array** specify custom source (transmitter) function over time; needs to correspond to suitable *source_times*
- **source_times = None, type numpy array** specify custom *source_times* if no default ramp-free source functions are used

- **stepping_times = None, type numpy array** specify custom calculation times for the time-stepping after the source pulse
- **be_order = 1, type int** order of discretization in time for backward Euler method, alterntively set to *2*
- **n_lin_steps = 30, type int** number of linear steps between each logarithmic time step, each linear step requires only an inexpensive back-substitution for solving the system of linear equations
- **n_on = 101, type int** number of linear time steps for creating source signal, not recommended to be changed
- **t0_j = -10., type float** specify start time (negative) for specifying the duration of the source signal ramp if the source type is *j*
- **pulse_width = 1e-2, type float** relative width of dirac function, not recommended to be changed
- **t_tol = 1e-2, type float** tolerance to check if a calculated time step is linear or not (if not linear, a new matrix factorization will be computed)

build_var_form(***fem_kwargs*)
Build variational formulation for E-field time-stepping approach and print FE parameters. A description of the keyword arguments is given in the **ApproachBaseTD** and **ImplicitEuler**classes.

first_order_IE(*u_0*)
First-order backward Euler time-stepping scheme using diffusive approximation (no displacement currents).

- **u_0, type dolfin function** initial solution at first time step

first_order_IE_full(*u_0*, *u_1*)
First-order backward Euler time-stepping scheme for full formulation considering displacement currents.

- **u_0, type dolfin function** initial solution at first time step
- **u_1, type dolfin function** intital solution at second time step

gaussian(*x*, *x0*, *alpha*)
Calculate gaussian pulse.

- **x, type numpy array** time values
- **x0, type float** central value of times values
- **alpha, type float** stretching parameter of Dirac pulse function

init_initial_condition(*ini_con*, *order=1*)
Initialize the initial fields u_0 and u1 (if second order scheme or full formulation) for the time-stepping.

- **ini_con, type dolfin function** use customized initial fields, not supported yet
- **order = 1, type int** specify first or second order time-stepping scheme

init_times()
Initialize times for time-stepping. Depending on the *source_type*, initial time steps before are added automatically to incorporate the source signal before the observation times are calculated.

print_stepping_info(*dt*)
Print some information before starting the time-stepping process.

- **dt, type array** vector of time-steps

second_order_IE(*u_0*, *u_1*)
Second-order backward Euler time-stepping scheme.

- **u_0, type dolfin function** initial solution at first time step
- **u_1, type dolfin function** intital solution at second time step

time_stepping()
Initialize time-discfretization and variational formulations for assembly.

class custEM.fem.time_domain_approaches.**RationalArnoldi**(*FS*, *MP*)
Bases: *custEM.fem.fem_utils.ApproachBaseTD*

FE implementation of rational Arnoldi (Krylov-subspace) method after Börner (2015).

- **rational_arnoldi()** runs the complete rational-arnoldi solution process as well as H-field conversion and export methods, FE modeling and time stepping related keyword arguments can be passed to this method
- **build_var_form()** set up variational formulation and assemble mass and curl-curl matrices as well as right-hand side (source) vector
- **get_poles_from_table()** define poles used in rational Arnoldi method
- **build_rational_arnoldi_basis()** build Krylov subspace basis
- **expm_ra()** utility function to calculate matrix exponentials
- **project_solution()** calculate solution on Krylov subspace and project it back

- **n_mult = 6, type int** multiplyer of poles, do not change
- **n_poles = 2, type int** number of poles to be used
- **pole_shift = None, type float** shift poles in time; the default covered range is about 4 to 5 decades from the earliest times on; this values is adjusted automatically and it is not recommended to change it
- **dc_field = None, type dolfin Function** the DC field is automatically calculated on the fly; for special purposes, it might be computed externally and passed here

build_krylov_subspace_basis()
Build the Krylov subspace basis.

build_var_form(***fem_kwargs*)
Build variational formulation for rational Arnoldi approach and print FE parameters. A description of the keyword arguments is given in the **ApproachBaseTD** and **RationalArnoldi** classes.

expm_ra(*A*, *E*)
Calculate matrix exponentials with eigenvalues and eigenvectors.

- **A, E, type numpy ndarray** input matrices for the calculation

get_poles_from_table()
Define *poles* for ration Arnoldi method.

project_solution()
Project subspace solution back to original solution space.

rational_arnoldi()
Conduct rational Arnoldi modeling.

Module contents

fem

Submodules:

- ** frequency_domain_approaches**: E-field, H-field, A-V-mixed, A-V-nodal, F-U-mixed
- ** time_domain_approaches**: Implicit Euler, Inverse Fourier-Transform based, Rational Arnoldi
- **fem_base** for setting up the finite element kernel
- **fem_utils** for defining approach base class and related utility functions
- **primary_fields** for setting up primary fields
- **assembly** of left-/right-hand-sides using total field approaches

---

custEM.meshgen package

Submodules

custEM.meshgen.bathy_tools module

@author: Rochlitz.R

class custEM.meshgen.bathy_tools.**Bathy**(*bathy_file*, *x=None*, *y=None*, *centering=False*, *easting_shift=None*, *northing_shift=None*, *min_dist=50.0*, *min_points=10*, *coast_refinement=None*, *do_nothing=False*)

Bases: object

Coastline identification class for digital elevation models. Please note that the basic methods are working properly for a couple of specific models, but a more general application of this class needs some debugging and further extensions.

add_point_to_coast_line(*idx*, *direction*, *clockwise*, *tol=None*)

Add a coordinate to a coastline path if the distance to the previous position on the coastline is larger than *min_dist*.

- **idx, type list of [int, int]** x and y index of the DEM grid
- **direction, type str** direction string, either ‘w’, ‘a’, ‘s’, or ‘d’
- **clockwise, type bool** go in clock- or counterclockwise direction
- **tol = None, type float** custom search tolerance if required, not recommended to be changed

add_point_towards_east(*idx*)

Add a coordinate to a coastline path in eastern direction.

- **idx, type list of [int, int]** x and y index of the DEM grid

add_point_towards_north(*idx*)

Add a coordinate to a coastline path in northern direction.

- **idx, type list of [int, int]** x and y index of the DEM grid

add_point_towards_south(*idx*)

Add a coordinate to a coastline path in southern direction.

- **idx, type list of [int, int]** x and y index of the DEM grid

add_point_towards_west(*idx*)
Add a coordinate to a coastline path in western direction.

- **idx, type list of [int, int]** x and y index of the DEM grid

build_closing_frame(*idx*, *clockwise*)
Continue coastline path in clockwise or counterclockwise direction if the horizontal mesh boundary is reached.

- **idx, type list of [int, int]** x and y index of the DEM grid
- **clockwise, type bool** go in clock- or counterclockwise direction

check_distance(*x*, *y*)
Calculate distance between a coordinate [x, y] and the previous position on the coastline.

- **x, y, typefloat** x and y coordinates for the horizontal distance evaluation

check_neighbours(*kk*, *jj*)
Check z-values of the eight neighbors around a position in the DEM. Utility function that needs an update.

- **kk, jj, type int** x/y indices of the DEM grid

eval_coast_line()
Main method to find intersection of topo and zero-height level.

find_direction(*idx*, *from_boundary=False*)
Identifiy clock- or counterclockwise search path direction based on regular grid DEM data automatically. Does not work in all cases so far.

- **idx, type list of [int, int]** x and y index of the DEM grid to start the direction search from
- **from_boundary = False, type bool** internally used flag to check if a coastline path was following the horizontal mesh boundaries before entering the grid again

find_last_dir(*pre_idx*, *idx*)
Evaluate previous direction during the coastline search.

- **pre_idx, type list of [int, int]** previous x and y index of the DEM grid
- **idx, type list of [int, int]** x and y index of the DEM grid

find_start_point(*jjj=0*)
Evaluate start indices for a new coastline search in the DEM.

- **jjj, type int** counter to continute until all indices in the DEM-grid were considered

go_clockwise(*side*, *idx*, *from_corner=False*)
Continue coastline path in clockwise direction if the horizontal mesh boundary is reached.

- **side, type str** string specifying one of the four sides
- **idx, type list of [int, int]** x and y index of the DEM grid
- **from_corner = False, type bool** internally used flag to continue a path after a corner of the horizontal mesh extent was reached

go_counter_clockwise(*side*, *idx*, *from_corner=False*)
Continue coastline path in counterclockwise direction if the horizontal mesh boundary is reached.

- **side, type str** string specifying one of the four sides
- **idx, type list of [int, int]** x and y index of the DEM grid

- **from_corner = False, type bool** internally used flag to continue a path after a corner of the horizontal mesh extent was reached

loadASC(*topo_file*, *centering*, *easting_shift*, *northing_shift*)
Load *asc* file (DEM matrix with location header). This function is redudant to an identical function in the **DEM** class and will be removed in a major update of the bathy-tools.

- see **Bathy** class description

loadTXT(*topo_file*, *centering*, *easting_shift*, *northing_shift*)
Load *txt* file with column-based list of DEM coordinates. This function is redudant to an identical function in the **DEM** class and will be removed in a major update of the bathy-tools.

- see **Bathy** class description

translate(*x*, *y*, *centering=False*, *easting_shift=None*, *northing_shift=None*, *back=False*)
Function to translate coordinates in horizontal direction.

- **x, y, type list/array of coordinates** coordinates to be translated horizontally
- see **Bathy** class description

custEM.meshgen.dem_interpolator module

@authors: Rochlitz.R & Günther.T

class custEM.meshgen.dem_interpolator.**DEM**(*demfile*, *x=None*, *y=None*, *centering=False*, *easting_shift=None*, *northing_shift=None*, *revert_z=False*)
Bases: object

Interpolation class for digital elevation models (DEM). Automatically creates interpolation objects during the class initialization, based on DEM information stored as list of coordinates in *txt* format or as matrix in *asc* format.

loadASC(*ascfile*, *centering*, *easting_shift*, *northing_shift*)
Load ASC file (DEM matrix with location header).

- see **DEM** class description

loadTXT(*demfile*, *centering*, *easting_shift*, *northing_shift*)
Load *txt* file with column-based list of DEM coordinates.

- see **DEM** class description

show(*cmap='terrain'*, *cbar=True*, *ax=None*, ***kwargs*)
Show digital elevation model. Needs an update.

- **cmap = "terrain", type str ()** matplotlib colormap definiton
- **cbar = True, type bool** add colorbar to the plot or not
- **ax = None, type matplotlib figure axes object** add the plot to a given axes object or create a new one
- ****kwargs, type keyword arguments** add additional keyword arguments for the plot style (e.g., *lw*)

translate(*centering=False*, *easting_shift=None*, *northing_shift=None*, *back=False*)
Function to translate coordinates in horizontal direction.

- see **DEM** class description

- **back = False, type bool** set True to translate back in the opposite direction; useful for rotating coordinates around a specific origin

custEM.meshgen.invmesh_tools module

@author: Rochlitz.R

class custEM.meshgen.invmesh_tools.**PrismWorld**(*name*, *x_extent*, *x_reduction=100.0*, *y_depth=300.0*, *z_depth=300.0*, *prism_quality=32.0*, *prism_area=10000.0*, *n_prisms=50.0*, *orthogonal_txs=None*, *txs=[]*, *surface_rxs=[]*, ***blankworld_kwargs*)

Bases: object

Specific class for automated generation of pseudo-2D / 3D inversion meshes discretized by prisms.

add_prisms(*back_nodes*, *front_nodes*)
Add nodes and triangle faces to include horizontal prisms as conformal PLC.

add_triangles(*back_nodes*, *front_nodes*)
Add front and back triangle faces of the prisms. All the if statements are required to account for potential nodes on the x-directed boundary edges of the rectangular prism outcrops at the surface.

create_2d_mesh(*name*, *prism_quality*, *prism_area*)
Create and store 2D mesh in x-z slice with triangle, which will be the basis for elongating the triangles to prisms in y-direction later on.

create_surface_rectangles()
Build rectangular frames of the prisms at the surface to be included in the surface mesh later on for avoiding intersections. Check also if transmitter paths are intersecting these rectangles, and if yes, find the intersections and add the corresponding points to the frames.

sort_surface_node_ids(*ids*)
Sort nodes at the surface for conformal triangle insertion.

custEM.meshgen.mesh_convert module

@author: modified by Rochlitz.R

custEM.meshgen.mesh_convert.**mesh2xml2h5**(*mesh_name*, *m_dir*, *rot*)
Converts TetGen meshes in Medit (*.mesh*) format to *xml* and *h5* files for the usage in FEniCS. This function is called automatically during the intialization of the **MOD** class and converts a mesh if it does not already exist as *xml* and *h5* file.

- **mesh_name, type str** mesh name
- **m_dir, type str** path to mesh directory

Convert between .mesh and .xml, parser implemented as a state machine:

0 = read 'Dimension' 1 = read dimension 2 = read 'Vertices' 3 = read number of vertices 4 = read next vertex 5 = read 'Triangles' or 'Tetrahedra' 6 = read number of cells 7 = read next cell 8 = read next domains 9 = done

custEM.meshgen.mesh_convert.**write_cell_interval**(*ofile*, *cell*, *n0*, *n1*)
Write interval (1D) information to domain file.

custEM.meshgen.mesh_convert.**write_cell_tetrahedron**(*ofile*, *cell*, *n0*, *n1*, *n2*, *n3*)
Write tetrahedron 3D information to file.

custEM.meshgen.mesh_convert.**write_cell_triangle**(*ofile*, *cell*, *n0*, *n1*, *n2*)
Write triangle 2D information to domain file.

custEM.meshgen.mesh_convert.**write_domain_marker**(*ofile*, *cell*, *marker*, *dfile=False*)
Write domain markers to domain file.

custEM.meshgen.mesh_convert.**write_footer_cells**(*ofile*)
Write footer *cells* to file.

custEM.meshgen.mesh_convert.**write_footer_domains**(*ofile*, *dfile=False*)
Write footer to domain file.

custEM.meshgen.mesh_convert.**write_footer_edges**(*ofile*)
Write footer *edges* to file.

custEM.meshgen.mesh_convert.**write_footer_graph**(*ofile*)
Write footer graph to file.

custEM.meshgen.mesh_convert.**write_footer_mesh**(*ofile*)
Write footer of mesh file.

custEM.meshgen.mesh_convert.**write_footer_vertices**(*ofile*)
Write footer *vertices* to file.

custEM.meshgen.mesh_convert.**write_graph_edge**(*ofile*, *v1*, *v2*, *weight=1*)
Write edge graph to file.

custEM.meshgen.mesh_convert.**write_graph_vertex**(*ofile*, *vertex*, *num_edges*, *weight=1*)
Write vertex graph to file.

custEM.meshgen.mesh_convert.**write_header_cells**(*ofile*, *num_cells*)
Write header *cells* to file.

custEM.meshgen.mesh_convert.**write_header_domains**(*ofile*, *num_domains*, *dfile=False*)
Write header to domain file.

custEM.meshgen.mesh_convert.**write_header_edges**(*ofile*, *num_edges*)
Write header *edges* to file.

custEM.meshgen.mesh_convert.**write_header_graph**(*ofile*, *graph_type*)
Write header graph to file.

custEM.meshgen.mesh_convert.**write_header_mesh**(*ofile*, *cell_type*, *dim*)
Write header of mesh file.

custEM.meshgen.mesh_convert.**write_header_vertices**(*ofile*, *num_vertices*)
Write header *vertices* to file.

custEM.meshgen.mesh_convert.**write_vertex**(*ofile*, *vertex*, *x*, *y*, *z*)
Write vertex to file.

custEM.meshgen.mesh_convert.**write_xml_domain_file**(*ofile_name*, *num_domains*, *marker*)
Write domain marker to mesh function file.

custEM.meshgen.mesh_convert.**xml_to_hdf5**(*xml*, *h5*)
Convert *xml* to HDF5 (*h5*) file.

- **xml, type str** name of *xml* file
- **h5, type str** name of *h5* file

custEM.meshgen.meshgen_tools module

@author: Rochlitz.R

class custEM.meshgen.meshgen_tools.**BlankWorld**(***raw_kwargs*)
Bases: object

Toolbox for mesh generation using pyGIMLi functionalities to generate a world **Omega**, add structures such as interfaces, anomalies, transmitters etc. to this world and finally call TetGen to built a mesh based on the exported polyfile.

This class was devloped over four years and provides useful functionalities for the mesh generation, but there are many parts which need an overhaul in the future. Maybe an updated version of these tools will be moved to pyGIMLi in future and updated to work with TetGen 1.6

add_area_marker(*poly*, *pos*, *face_size=0.0*, *marker=777*)
Add region marker to the surface or interfaces for 2D meshing, solely used internally to define area-constraints for cooresponding meshes.

- **poly, type pyGIMLi Polygon** polygon to which the marker is defined.
- **pos, type list or array of length (3,)** x, y, z coordinates where the marker should be placed, note that this point must be located inside the correct domain!
- **face_size = 0., type int** maximum facet-area contstraint, default (0.) means no constraint.
- **marker = 777, type int** region marker to define sub-areas within the *poly*.

add_brick(*start*, *stop*, *cell_size=0.0*, *marker=None*)
Add rectangular *brick* anomaly to **Omega**. Note that this is a special case of the **add_plate()** method, which can be used alternatively.

- **start, type list** list of back lower left corner [x_min,, y_min, z_min].
- **stop, type list** list of front upper right corner [x_max,, y_max, z_max].
- **cell_size = 0., type float** maximum cell-volume contstraint, default (0.) means no constraint.

add_cylinder(*r, h, origin=[0.0, 0.0, -200.0], n_segs=16, dip=0.0, dip_azimuth=0.0, cell_size=0.0, marker=None, marker_position=None*)
Add cylinder-anomaly to **Omega**.

- **r, type float** radius of the cylinder
- **h, type float** height of the cylinder
- **origin = [0., 0., -200.], type list/array of three floats** center (of mass) coordinate of the cylinder; it is rotated in case of non-zero dip or dip azimuth values around this point
- **n_segs, type int** number of segments to approximate the circumference
- **dip = 0., type float,** dip angle of the plate in degree
- **dip_azimuth = 0., type float** dip azimuth angle in degree, zero points towards positive x-direction
- **cell_size = 0., type float** maximum cell-volume contstraint, default (0.) means no constraint.
- **marker = None, type int** specify custom marker for plate if required

add_hollow_cylinder(*r1, r2, h=100.0, origin=[0.0, 0.0, 0.0], rotation=None, cell_size=0.0, n_segs=16, preserve=[], close=False, mark=False, close1st=False*)
Add hollow cylinder-anomaly to **Omega**. In progress!

- **r1, type float** inner radius of cylinder.
- **r2, type float** outer radius of cylinder.
- **cell_size = 0., type float** maximum cell-volume contstraint, default (0.) means no constraint.

add_interface(*frame=None*, *topo=None*, *id_offset=None*, *eval_zl=False*)
Add more or less horizontal 2D interface (layer interfaces) with or without topography to Omega.

add_intersecting_anomaly(*surface_intersection=False*, *surface_path=[]*, *intersection_paths=[]*, *intersecting_layers=None*, *cell_size=None*, *bottom=None*, *top=None*, *marker=None*, *marker_position=None*)

add_inv_domains(*depth*, *poly=None*, *rxs=None*, *cell_size=0.0*, *h_fact=1.3*, *v_fact=1.2*, *d=100.0*, *marker_positions=None*, *overwrite_markers=None*)

add_marker(*label*, *pos*, *marker=None*, *cell_size=0.0*)
Add region marker to domains. This function is called internally for each layer or anomaly added. By default, domains are numbered consequtively. It is usually not recommended not to maunaly set domain marker, even though the possibility is given by overwriting the keyword argument *marker=None* when adding, e.g., anomalies.

- **label, type str** label for defining the domain information print output after calling TetGen, which is enabled by default.
- **pos, type list or array of length (3,)** x, y, z coordinates where the marker should be placed, note that this point must be located inside the correct domain!
- **marker = None, type int** set manually defined region marker for this domain.
- **cell_size = 0., type float** maximum cell-volume contstraint, default (0.) means no constraint.

add_paths(*paths*, *closed_paths=None*, *marker=None*)
Add (observation) lines to **Omega**.

- **paths, type list** *paths* is a list of pointsets (arrays with shape (n, 3)) which will be connected from point to point.
- **closed_paths = None, type list** a list of same length as *paths* is required with *True* or *False* values defining which path should be closed and which not.

add_plate(*dx=200.0, dy=200.0, dz=50.0, origin=[0.0, 0.0, -200.0], dip=0.0, dip_azimuth=0.0, cell_size=0.0, marker=None*)
Add rectangular *plate* anomaly to **Omega**. Rectangular bodys can be added by using the method *add_brick()*.

- **dx, dy = 200., type float** dimensions in x/y-directions (before rotation)
- **dz = 50., type float** thickness of the plate
- **origin = [0., 0., -200.], type list/array of three floats** center (of mass) coordinate of the plate; plate is rotated in case of non-zero dip or dip azimuth values around this point
- **dip = 0., type float,** dip angle of the plate in degree
- **dip_azimuth = 0., type float** dip azimuth angle in degree, zero points towards positive x-direction
- **cell_size = 0., type float** maximum cell-volume contstraint, default (0.) means no constraint.
- **marker = None, type int** specify custom marker for plate if required

add_points (*points*, *marker=None*)
Add points to **Omega**.

- **points, type array with shape (n, 3)** arrqay of points to be added
- **marker = None, type int** specify custom marker for points if required

add_prism (*poly*, *top*, *bottom*, *cell_size=0.0*, *marker=None*, *marker_pos=None*)
Add rectangular *brick* anomaly to **Omega**. Note that this is a special case of the **add_plate()** method, which can be used alternatively.

- **start, type list** list of back lower left corner [x_min,, y_min, z_min].
- **stop, type list** list of front upper right corner [x_max,, y_max, z_max].
- **cell_size = 0., type float** maximum cell-volume contstraint, default (0.) means no constraint.

add_quadrangle_face (*n1*, *n2*, *n3*, *n4*)
Add quadrangle facet to Omega defined by nodes n1 - n4.

- **n1, n2, n3 n4, type int** four node id's to build a quandrangle.

add_refinement_area (*frame*, *quality*, *area*, *z=0.0*, *topo=False*)

add_rx (*points*, *marker=None*)
Add recevier points for automated interpolation to **Omega**.

- **points, type list or array with shape (n, 3)** list of receiver points to add

add_surface_anomaly (*insert_paths=[]*, *depths=None*, *cell_sizes=None*, *marker=None*, *dips=None*, *dip_azimuths=None*, *split_depths=None*, *marker_positions=None*, *split_coords=None*)

add_topo (*node_positions*, *topo=None*, *eval_zl=False*)
Apply topography, either DEM information or synthetic descriptions as z = f(x, y) to the z-values of an interface mesh.

- **node_positions, type array of shape (n, 3)** positions of all nodes of the interface mesh

topo = None, type str or function if None, **self.topo** will be used for the earth's surface, for further interfaces, either additional DEM information can be used or a synthetic definition via a function f(x, y)

add_tx (*tx*, *grounded=True*, *marker=None*)
Add transmitter paths to **Omega**.

- **tx, type list** list of pointsets (arrays with shape (n, 3)) which will be connected from point to point.
- **grounded = False, type bool** close first and last point of path to build a grounded loop or not.

build_fullspace_mesh (*cell_size=None*, *boundary_marker=99*)
Create a fullspace mesh.

- **cell_size = 0., type float** create a fullspace mesh with cells of maximum volume ([m^3]) of **max_cell_size**

build_halfspace_mesh (*boundary_marker=99*, ***mesh_build_kwargs*)
Create halfspace mesh.

- **q = 34., type float** quality (see *triangle* documentation) of the surface mesh

- **outer_area_cell_size = None, type float** if not specified when initializing the **BlankWorld** instance, set or owerwrite the maximum area of cells on the surface for the "outer area" here
- **inner_area_cell_size = None, type float** if not specified when initializing the **BlankWorld** instance, set or owerwrite the maximum area of cells on the surface for the "inner area" here

build_layered_earth_mesh(*n_layers*, *layer_depths*, *insert_struct=None*, *closed_struct=None*, *tx_struct=None*, *struct_interface=0*, *boundary_marker=99*, ***le_kwargs*)

Create layered earth mesh.

- **n_layers, type int** number of subsurface layers
- **layer_depths, type list of floats** list with len == n_layers - 1, specifying the depths of 1D subsurface layer interfaces. the z-axis is pointing upwards, so negative values correspond to increasing depths. This argument will be overwritten, if subsurface topography is applied

build_surface(*insert_line_tx=[]*, *insert_loop_tx=[]*, *insert_points=[]*, *insert_lines=[]*, *insert_loops=[]*, *insert_paths=[]*, *closed_path=False*, *subsurface_interface=False*, *line_marker=None*, *loop_marker=None*, *path_marker=None*, *line_tx_marker=None*, *loop_tx_marker=None*, *point_marker=None*, *add_to_existing=False*)

Create pyGIMLi surface polygon for 2D meshing with "triangle" later on. Note! Structures at the surface such as CSEM transmitters or observation lines / points have to incorporated here.

- **insert_line_tx = [], type list** list of grounded transmitter paths, numbered successively
- **insert_loop_tx = [], type list** list of ungrounded transmitter paths; note, loop_tx will be always added and numbered after all line tx
- **insert_points = [], type list** list of meshgen_utils "pointSet" definitions
- **insert_lines = [], type list** list of meshgen_utils "line" definitions
- **insert_loops = [], type list** list of meshgen_utils "loop" definitions
- **insert_paths = [], type list** list of paths descriped by coordinates
- **closed_path = False, type bool or list of len == len(insert_paths)** close the segsment between the first and last point of a given "PATH" to either generate a grounded (line) or ungrounded (loop) CSEM source or in general, open or closed polygons
- **subsurface_interface = False, type bool** for internal usage only, specifies if the 2D polygon will be on the surface or a subsurface interface

call_tetgen(*tet_param='default'*, *export_before=True*, *export_format=' -k'*, *name=None*, *copy=True*, *print_infos=True*, *suppress='NEF'*)

Wrapper-function to call TetGen version 1.5 and copy the resulting mesh into the default *"m_dir"/_mesh* directory.

- **tet_param = 'default', type str** specify your parameters which should be passed to the TetGen call, e.g., '**'-pq1.4aYAO2/3'** without including flags for the export. Alternative keyword arguments are 'raw' ('**'-p') **or 'faces' ('**'-pYA'**). It is also possible to directly pass the TetGen-options (aside from export), e.g., *tet_param='-pq1.2aMAO2/3'*.
- **export_before = True, type bool** automatically export the pyGIMLi mesh *Omega* as ".poly"-file in the working directory **self.w_dir** before calling TetGen.
- **export_vtk = True, type bool** set False for not exporting *.vtk*-file to be viewed in Paraview.
- **name = None, type str** specify custom export name for the mesh, defualt is **self.name**.
- **copy = True, type bool** automatically copy the mesh in *.mesh* (Medit)-format to the save- directory **self.s_dir** for automated conversion on the first computation run.

- **print_infos = True, type bool** set *False* if no domain information should be printed at the end.

create_box()
Create bounding box from given limits and initialize **inner_area** and **outer_area** properties, if a separation into a fine-discretized inner (observation) area at the surface and outer area (boundary-area) was specified during intitializing the **BlankWorld** class.

eval_unique_markers()
extract unique markers specified in world *Omega*

extend_anomaly_outcrops()
Function for extending anomaly-bodies into a layer. Completely automated for internal usage. For more details, it is referred to the **add_surface_anomaly()** method, where users can define the anomalies!

extend_intersecting_anomaly()
Function for extending anomaly-bodies which interesect layers.

extend_world(*x_fact=10.0, y_fact=10.0, z_fact=10.0, marker=[0, 1], cell_sizes=None, boundary_marker=99, initial=True*)
alias for *there_is_always_a_bigger_world()* method

get_topo_vals(*node_positions*, *topo=None*, *z=0.0*)
Apply topography, either DEM information or synthetic descriptions as z = f(x, y) to the z-values of an interface mesh.

- **node_positions, type array of shape (n, 3)** positions of all nodes of the interface mesh

topo = None, type str or function if None, **self.topo** will be used for the earth's surface, for further interfaces, either additional DEM information can be used or a synthetic definition via a function f(x, y)

init_dem(*surface_mesh_coords*, *topo=None*)
Initialize DEM instance for interpolating real-world topography from digital-elevation-model-files and check if the extent of the dem-file coordinates is large enough for the specified mesh dimensions.

- **surface_mesh_coords, type array of shape (n, 3)** array containing all coordinates of the surface mesh for comparing the spatial extent in x- and y-direction.

print_region_info()
Print the domain information of the final mesh after calling TetGen.

remove_duplicate_poly_faces()
Check which *self.Omega.n_poly* faces were added multiple times if subsurface structures touch each other and remove the duplicates.

there_is_always_a_bigger_world(*x_fact=10.0, y_fact=10.0, z_fact=10.0, marker=[0, 1], cell_sizes=None, boundary_marker=99, initial=True*)
Appends a bigger world (halfspace) to the existing world *Omega*. This bigger world is synonymous to a "tetrahedron boundary".

- **x_fact, y_fact, z_fact = 10., type float** factor in x-, y- and z-direction multiplied to the original domain size specified by the **x/y/zlim* class attributes.
- **marker = [0, 1], list of two int** specify your own marker values for the bounding halfspace mesh.

update_kwargs(*kwargs*)

write_mesh_parameters()
Write topography and layered-earth structure information to a para- meter file to preserve these information for following computations. Also, tx and rx information is stored.

custEM.meshgen.meshgen_utils module

@author: Rochlitz.R

custEM.meshgen.meshgen_utils.**add_edges**(*surface*, *edges*, *nn*, *size*, *marker*, *closed=False*)
Add edge to polygon.

custEM.meshgen.meshgen_utils.**assign_topography**(*nodes*, *t_dir=None*, *topo=None*, *z=0.0*, *centering=True*, *easting_shift=None*, *northing_shift=None*, *revert_z=False*, *rotation=None*)
Assign topography on nodes.

- **nodes, type list/array with shape (:, 3)** list or array of 3D coordinates
- **t_dir = None, type str** directory of topography file
- **topo = None, type function or str** topography function or file
- **z = 0., type float** add constant offset to topography in z-direction
- **centering = True, type bool** Set **True** if extent of topography file should be centered horizontally around origin (0, 0)
- **easting_shift = None, type float** add specified shift to x-coordinates of topography file
- **northing_shift = None, type float** add specified shift to y-coordinates of topography file
- **rotation = None, type float** rotate complete topography file about specified angle(DEG) horizontally

custEM.meshgen.meshgen_utils.**closest_node**(*node*, *nodes*)
Find the closest node in a list of 2D/3D corrdinates.

- **node, list/array with length 3** reference node
- **nodes, list/array with shape (:, 2) or (:, 3)** list or array of nodes to be checked

custEM.meshgen.meshgen_utils.**create_circle**(*r*, *n*, *h*)
Create a list of **n** (number of) 3D coordinates on the circumference of a circle with radius **r** and height **h**.

- **r, type float** the radius of the circle
- **n, type int** number of nodes on the cirumference of the circle
- **h, type float** z-coordinates of the circle

custEM.meshgen.meshgen_utils.**create_ellipse**(*a*, *b*, *n*, *h*)
Create a list of **n** (number of) 3D coordinates on the circumference of an ellipsum with 'radii' **a** and **b**, and height **h**.

- **a, b, type float** the two radius of the ellipse
- **n, type int** number of nodes on the cirumference of the ellipse
- **h, type float** z-coordinates of the ellipse

custEM.meshgen.meshgen_utils.**export_tetgen_edge_file**(*preserve_edges*, *edge_marker*, *filename*)
Write a given set of edges which should be preserved into a Ascii file in Tetgen *.edge* format.

- **preserve_edges, type list** edges which should be exported to the *.edge* file
- **edge_marker, type list** list of markers assigned to the edges
- **filename, type str** name of output file

custEM.meshgen.meshgen_utils.**export_tetgen_node_file**(*preserve_nodes*, *filename*)
Write a given set of nodes which should be preserved into an Ascii file in Tetgen *.edge* format.

- **preserve_nodes, type list** edges which should be exported to the *.node* file
- **filename, type str** name of output file

custEM.meshgen.meshgen_utils.**export_tetgen_poly_file**(*poly*, *filename*, *float_format='.3f'*, *poly_marker=None*, ***kwargs*)
Write a given piecewise linear complex (mesh/poly) into an Ascii file in Tetgen *.poly* format.

- **poly, type pyGIMLi mesh** piecewise linear complex which should be exported to the *.poly* file
- **filename, type str** name of output file

float_format = '.12e', type string format that will be used to write float values in the Ascii file, default is the exponential float form with a precision of 12 digits

custEM.meshgen.meshgen_utils.**find_closest**(*l1*, *l2*)
Find the closest entry in a list of 1D corrdinates in comparison to all entries of another list.

- **l1, type list** reference list of coordinates
- **l2, type list** another list of coordinate, usually with a different length

custEM.meshgen.meshgen_utils.**find_edge_node_ids**(*frame*, *mesh*, *id_offset*)
Internal utility function for meshgen tools to find the edge node ids of a horizontal 2D mesh for the extension of the mesh in z-direction.

- **frame, type array with shape (:, 3)** references frame defined by coordinates
- **mesh, type pyGIMLi mesh** mesh, whose coordinates should be checked
- **id_offset, type int** offset of ids between 2D surface meshes and 3D mesh (should be removed if the meshgen_tools get an overhaul)

custEM.meshgen.meshgen_utils.**find_nodes_on_segment**(*p0*, *p1*, *mesh*, *topo_f*)
Find nodes on a segment.

- mesh

custEM.meshgen.meshgen_utils.**find_on_frame**(*frame_coords*, *coords*)
Check if coordinates are located on a frame.

- **frame_coords, type array with shape (:, 3)** array of frame coordinates
- **coords, type array with shape (:, 3)** array of coordinates to be checked

custEM.meshgen.meshgen_utils.**get_intersect**(*a1*, *a2*, *b1*, *b2*)
Return intersection coordinate of two segments *a* and *b*.

custEM.meshgen.meshgen_utils.**get_unique_poly_ids**(*polys*, *max_poly_length*)
Get unique ids of nodes of multiple polygons.

- **polys, type list** list of polygons
- **max_poly_length** maximum number of nodes of within all polygons

custEM.meshgen.meshgen_utils.**inside_poly**(*x*, *y*, *path*)
Return True if a coordinate (x, y) is inside a polygon defined by a list of verticies [(x1, y1), (x2, x2), ... , (xN, yN)].

Reference: http://www.ariel.com.au/a/python-point-int-poly.html

- **x, type float** x coordinate of point to be checked
- **y, type float** y coordinate of point to be checked
- **path, type list/array with shape (:, 3)** list or array of 3D coordinates

custEM.meshgen.meshgen_utils.**is_between**(*a*, *b*, *c*, *a_tol=0.01*, *print_points=False*)
Search if a Nedelec-dof is located on the edge between two points in 3D, needed for crooked sources defined as path. Search if point c is in between a and b.

- **a, b, c, type list/array of length 3** three points to be evaluated
- **a_tol = 1e-2, type float** absolute tolerance to account for numerical inaccuracies
- **print_points = False, type bool** set **True** to enable print of coordinates

custEM.meshgen.meshgen_utils.**is_between_1D**(*a*, *b*, *c*, *a_tol=0.01*, *print_points=False*)
Search if a x-coordinate is located in an interval in in horizontal direction. Search if point c is in between a and b.

- **a, b, c, type list/array of length 3** three points to be evaluated
- **a_tol = 1e-2, type float** absolute tolerance to account for numerical inaccuracies
- **print_points = False, type bool** set **True** to enable print of coordinates

custEM.meshgen.meshgen_utils.**is_between_2D**(*a*, *b*, *c*, *a_tol=0.01*, *print_points=False*)
Search if a Nedelec-dof is located on the edge between two points in 2D in horizontal direction. Search if point c is in between a and b.

- **a, b, c, type list/array of length 3** three points to be evaluated
- **a_tol = 1e-2, type float** absolute tolerance to account for numerical inaccuracies
- **print_points = False, type bool** set **True** to enable print of coordinates

custEM.meshgen.meshgen_utils.**is_between_2D_exact**(*a*, *b*, *c*, *a_tol=0.01*, *print_points=False*)
Search if a Nedelec-dof is located on the edge between two points in 2D in horizontal direction. Search if point c is in between a and b.

- **a, b, c, type list/array of length 3** three points to be evaluated
- **a_tol = 1e-2, type float** absolute tolerance to account for numerical inaccuracies
- **print_points = False, type bool** set **True** to enable print of coordinates

custEM.meshgen.meshgen_utils.**is_between_2Dvert**(*a*, *b*, *c*, *a_tol=0.01*, *print_points=False*)
Search if a Nedelec-dof is located on the edge between two points in 2D in vertical direction. Search if point c is in between a and b.

- **a, b, c, type list/array of length 3** three points to be evaluated
- **a_tol = 1e-2, type float** absolute tolerance to account for numerical inaccuracies
- **print_points = False, type bool** set **True** to enable print of coordinates

custEM.meshgen.meshgen_utils.**line**(*start=[0.0, 0.0, 0.0], stop=[0.0, 0.0, 0.0], n_segs=100, z=0.0, topo=None, t_dir=None, **topo_kwargs*)
Return 3D coordinates of points on a line in arbitrary direction.

- **start, stop = [0., 0., 0.], type list or array** start and stop points
- **n_segs = 100, type int** number of segments on the line
- **topo_kwargs** listed below
- **z = 0., type float** offset of the line in z-direction
- **topo = None, type str or python function** DEM-topography file or synthetic topography (python) function.
- **t_dir = None, type str** directory of topography (DEM) files
- **centering = False, type bool** set **True**, if the given DEM should be centered relative to the computational domain coordinate frame
- **easting_shift, northing_shift = None, type float** if centering is **False**, a custom offset of a given DEM relaive to the computational domain in x- and y- direction can be set
- **revert_z = False, type bool** revert z-values of the DEM if the original height information was provided in a donward-oriented coorindate system
- **rotation (RAD) = None, type float** rotation of the DEM around the center coordinate (set via the *centering* flag or *easting/northing_shift*) relative to the computational domain, given in rad, not degree

custEM.meshgen.meshgen_utils.**line_x**(*start=-100.0*, *stop=100.0*, *n_segs=100*, *y=0.0*, *z=0.0*, *topo=None*, *t_dir=None*, ***topo_kwargs*)

Return 3D coordinates of points on a line in x-direction.

- **start, stop = +-1e2, type float** start and stop values in x-direction
- **n_segs = 100, type int** number of segments on the line
- **y, z = 0., type float** offset of the line in y- and z-direction
- **topo_kwargs** see description of *line* method or *meshgen_tools/dem_interpolator*

custEM.meshgen.meshgen_utils.**line_y**(*start=-100.0*, *stop=100.0*, *n_segs=100*, *x=0.0*, *z=0.0*, *topo=None*, *t_dir=None*, ***topo_kwargs*)

Return 3D coordinates of points on a line in y-direction.

- **start, stop = +-1e2, type float** start and stop values in x-direction
- **n_segs = 100, type int** number of segments on the line
- **x, z = 0., type float** offset of the line in x- and z-direction
- **topo_kwargs** see description of *line* method or *meshgen_tools/dem_interpolator*

custEM.meshgen.meshgen_utils.**line_z**(*start=-100.0*, *stop=100.0*, *n_segs=100*, *x=0.0*, *y=0.0*)

Return 3D coordinates of points on a line in z-direction.

- **start, stop = +-1e2, type float** start and stop values in x-direction
- **n_segs = 100, type int** number of segments on the line
- **x, y = 0., type float** offset of the line in x- and y-direction

custEM.meshgen.meshgen_utils.**loop_c**(*origin=[0.0, 0.0, 0.0], r=50.0, n_segs=100, z=0.0, topo=None, t_dir=None, **topo_kwargs*)

Return 3D coordinates of points on a circular loop.

- **origin = [0., 0., 0.], type list of floats** x-, y-, z- coordinates of the center of the loop
- **r = 5e1, type float** radius of the loop
- **n_segs = 100, type int** number of segments on the loop circumference

- **topo_kwargs** see description of *line* method or *meshgen_tools/dem_interpolator*

custEM.meshgen.meshgen_utils.**loop_e**(*origin=[0.0, 0.0, 0.0], a=50.0, b=50.0, n_segs=100, z=0.0, topo=None, t_dir=None, **topo_kwargs*)

Return 3D coordinates of points on a ellipsoidal loop.

- **origin = [0., 0., 0.], type list of floats** x-, y-, z- coordinates of the center of the loop
- **a, b = 5e1, type float** radius of the loop
- **n_segs = 100, type int** number of segments on the loop circumference
- **topo_kwargs** see description of *line* method or *meshgen_tools/dem_interpolator*

custEM.meshgen.meshgen_utils.**loop_r**(*start=[-100.0, -100.0], stop=[100.0, 100.0], n_segs=4, nx=None, ny=None, z=0.0, topo=None, t_dir=None, loop_rotation=None, **topo_kwargs*)

Return 3D coordinates of points on a rectangular loop.

- **start, stop = +-[1e2, 1e2], type list: [float, float]** opposite corner coordinates of rectangular loop
- **n_segs = 4, type int** number of overall segments of the loop, this keyword argument is ignored if *nx* and *ny* are specified
- **nx, ny = 100, type int** number of segments on the x- and y-directed edges of the loop
- **loop_rotation = None,** rotate loop horizontally around angle in (RAD)
- **topo_kwargs** see description of *line* method or *meshgen_tools/dem_interpolator*

custEM.meshgen.meshgen_utils.**npoly_to_triangles**(*world*, *ids*)

Add polygones with more than three edges in form of triangles to *Omega*.

custEM.meshgen.meshgen_utils.**order_list**(*ref*, *points*, *tmp*)

Sort values in list in increasing order. Deprecated and not used anymore.

custEM.meshgen.meshgen_utils.**pointset**(*pointset*)

Return a given set of 3D points, either list or array, as numpy array.

- **pointset, type list** list of coordinates to be converted to an array of shape (:, 3)

custEM.meshgen.meshgen_utils.**poly_area**(*path*)

Caluclate enclosed area of polygon.

- **path, type array with shape (:, 2) or (:, 3)** array of coordinates describing the polygon

custEM.meshgen.meshgen_utils.**poly_peri**(*path*)

Calculate perimeter of polygon.

- **path, type array with shape (:, 2) or (:, 3)** array of coordinates describing the polygon

custEM.meshgen.meshgen_utils.**refine_adaptive**(*coords*, *txs*, *r=10.0*, *r2=5.0*, *r3=1.0*, *d3=200.0*, *d2=500.0*, *min_tx_dist=100.0*, *rot=30.0*, *n_segs=3*)

Decription to add

custEM.meshgen.meshgen_utils.**refine_path**(*path*, *n_segs=None*, *length=None*)

Refine each section of a 3D path into n_segs equal segments.

- **path, type list/array with shape (:, 3)** list or array of 3D coordinates
- **n_segs, type int** number of nodes to be inserted within each segment of the path
- **length, type float** maximum length of any segment of the path

custEM.meshgen.meshgen_utils.**refine_rx**(*coords*, *r=5.0*, *rot=None*, *n_segs=3*)
Refine rx position by enclosing them with hexagons of radius *r*. The smaller *r*, the better the refinement. Note that each hexagon will be subdivided into 6 equilateral triangles, which will support optimum tetrahedra quality during the meshing process.

- **coords, type list/array with shape (:, 3)** list or array of 3D coordinates
- **r = 5., type float** radius of surrounding equilateral (triangle if *n_segs=3*) shape
- **rot = None, type float** horizontal rotation angle(DEG) between subsequent shapes
- **n_segs = 3, type int** number of corners of surrounding equilateral polygon shape

custEM.meshgen.meshgen_utils.**resolve_rx_overlaps**(*rxs*, *refinement_size=10.0*, *ignore_z=True*)

custEM.meshgen.meshgen_utils.**rotate**(*array*, *alpha*, *axis='z'*, *tensor=False*)
Rotate array about angle alpha(RAD) around given axis Default rotation axis is z, alternatively x or y.

- **array, type array of shape (:, 3)** array of coordinates to be rotated
- **alpha, type float** angle(RAD) specifying the rotation
- **axis = 'z', type str** rotation axis

custEM.meshgen.meshgen_utils.**rotate_around_point**(*array*, *shift*, *angles*, *axes*, *pre_rotate=False*)
Rotate array around arbitrary point defined by shift, not around centroid.

- **array, type array of shape (:, 3)** array of coordinates to be rotated
- **shift, type list or array of length 3** x-, y-, and z- values defining the shift
- **angles, type list of floats** list of angles(RAD) specifying the rotation
- **axes, type list of str** list of rotation axes
- **pre_rotate = False, type bool** conduct an initial roation using the second entry of axes/angles

custEM.meshgen.meshgen_utils.**translate**(*array*, *shift*, *back=False*)
Translate array about given shift

- **array, type array with shape (:, 3)** array of coordinates to ne translated
- **shift, type list or array of length 3** x-, y-, and z- values defining the shift
- **back = False, type bool** set "True" is shift in opposite direction is required

custEM.meshgen.meshgen_utils.**write_synth_topo_to_asc**(*t_dir*, *file_name*, *topo*, *spacing=25.0*, *x_min=-10000.0*, *y_min=-10000.0*)
Write an array of topography data to an *.asc* file.

- **t_dir, type str** name of output directory
- **file_name, type str** name of exported topography file
- **topo, type array with shape (:, 3)** 2D array containing only z_coordinates with topography information
- **spacing = 25., type float** horizontal grid spacing in x- and y-direction
- **x_min = -1e4, type float** x-coordinate of "lower left" corner

- **y_min = -1e4, type float** y-coordiante of "lower_left" corner

custEM.meshgen.meshgen_utils.**write_synth_topo_to_xyz**(*t_dir*, *file_name*, *topo*)
Write an array of topography data to an *.xyz* file.

- **t_dir, type str** name of output directory
- **file_name, type str** name of exported topography file
- **topo, type array with shape (:, 3)** array containing coordinates with topography information

custEM.meshgen.vtk_convert module

@author: modified by Rochlitz.R

custEM.meshgen.vtk_convert.**vtk2xml2h5**(*mesh_name*, *m_dir*, *rot*, *overwrite_markers*)
Converts TetGen meshes in VTK (*.vtk*) format to *xml* and *h5* files for the usage in FEniCS. This function is called automatically during the intialization of the **MOD** class and converts a mesh if it does not already exist as *xml* and *h5* file.

- **mesh_name, type str** mesh name
- **m_dir, type str** path to mesh directory

Convert between .vtk and .xml, parser implemented as a state machine:

0 = read number of vertices 1 = read vertices 2 = read number of cells 3 = read cells 4 = read number of markers 5 = read cell markers 6 = done

custEM.meshgen.vtk_convert.**write_cell_tetrahedron**(*ofile*, *cell*, *n0*, *n1*, *n2*, *n3*)
Write tetrahedron 3D information to file.

custEM.meshgen.vtk_convert.**write_domain_marker**(*ofile*, *cell*, *marker*, *dfile=False*)
Write domain markers to domain file.

custEM.meshgen.vtk_convert.**write_footer_cells**(*ofile*)
Write footer *cells* to file.

custEM.meshgen.vtk_convert.**write_footer_domains**(*ofile*, *dfile=False*)
Write footer to domain file.

custEM.meshgen.vtk_convert.**write_footer_mesh**(*ofile*)
Write footer of mesh file.

custEM.meshgen.vtk_convert.**write_footer_vertices**(*ofile*)
Write footer *vertices* to file.

custEM.meshgen.vtk_convert.**write_header_cells**(*ofile*, *num_cells*)
Write header *cells* to file.

custEM.meshgen.vtk_convert.**write_header_domains**(*ofile*, *num_domains*, *dfile=False*)
Write header to domain file.

custEM.meshgen.vtk_convert.**write_header_mesh**(*ofile*, *cell_type*, *dim*)
Write header of mesh file.

custEM.meshgen.vtk_convert.**write_header_vertices**(*ofile*, *num_vertices*)
Write header *vertices* to file.

custEM.meshgen.vtk_convert.**write_vertex**(*ofile*, *vertex*, *x*, *y*, *z*)
Write vertex to file.

custEM.meshgen.vtk_convert.**write_xml_domain_file**(*ofile_name*, *num_domains*, *markers*)
Write domain marker to mesh function file.

custEM.meshgen.vtk_convert.**xml_to_hdf5**(*xml*, *h5*)
Convert *xml* to HDF5 (*h5*) file.

- **xml, type str** name of *xml* file
- **h5, type str** name of *h5* file

Module contents

meshgen

Submodules:

- **meshgen_tools** for tetrahedral mesh generation
- **meshgen_utils** providing utility functions for mesh generation
- **invmesh_tools** providing specific classes for building inversion meshes
- **mesh_convert** for automated conversion of *.mesh* files to *.xml* and *.h5*
- **dem_interpolator** for interpolating real digital elevation data
- **bathy_tools** for identifying coastline paths

---

custEM.misc package

Submodules

custEM.misc.anomaly_expressions module

Created on Mon Jun 13 15:14:20 2016

@author: Rochlitz.R

class custEM.misc.anomaly_expressions.**Plate**(*thick=50.0, origin=[0.0, 0.0, -200.0], l_x=200.0, l_y=200.0, x_dip=0.0, y_dip=0.0, tol=0.01, **df_kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Define plate anomaly with FEniCS syntax, deprecated - do not use.

inside(*x*, *on_boundary*)

custEM.misc.misc module

@author: Rochlitz.R

class custEM.misc.misc.**Air**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

inside(*x*, *on_boundary*)

class custEM.misc.misc.**DirichletBoundary**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

inside(*x*, *on_boundary*)

class custEM.misc.misc.**Ground**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

inside(*x*, *on_boundary*)

custEM.misc.misc.**block_print**()
Supress all prints in the command prompt until *enable_print()* is called.

custEM.misc.misc.**check_approach_and_file_format**(*approach*, *file_format*, *path*, *logger*)
Utility function that checks if a valid modeling approach is chosen (avoid typos) and if HFD5 format is supported.

- **approach, type str** approach specification (see **MOD** class)
- **file_format, type str or None** if None, used *h5* if supported, otherwise xml for data export
- **path, type str** path to *custEM.misc* directory, forwarded internally
- **'h5 or 'xml', type str** file format specification

custEM.misc.misc.**check_if_model_exists**(*out_name*, *overwrite*, *load_existing*, *logger=None*)
Utility function that checks if a finite element model already exists, which can either be overwritten or the calculation can be aborted.

- See **'MOD'** class description

custEM.misc.misc.**dump_csr**(*fname*, *values*, *column_indices*, *row_pointers*, *size*)
Utility function that dumps a PETSCMatrix as a sparse csr-matrix to the hard disc.

- **fname, type str** export name of the matrix without the suffix **'.csr'**
- **values, type array** values of matrix elements
- **column_indices, type array** column indices
- **row_pointers, type array** row_pointers
- **size, type int** size (length) of the square matrix

custEM.misc.misc.**enable_print**()
Enable prints in the command prompt after *block_print()* was called.

custEM.misc.misc.**get_coordinates**(*FS*)
Return coordinates of a lagrange VectorFunctionSpace as numpy array.

- **FS, type class** FunctionSpaces instance
- **xyz, type numpy array** array or dof coordinates with shape (n, 3)

custEM.misc.misc.**logger_print**(*logger*, *lvl*, *string*, *val=None*, *pre_dash=True*, *post_dash=False*, *barrier=True*, *root_only=True*)
Utility function that prints only from root process in mpi mode.

- **logger, type logger from logging module** default *custEM* logger
- **lvl, type int** debug_level for custEM code
- **string, type str** string that should be printed
- **val = None, various types** a value that should be printed after the "message"

- **pre_dash = True, type bool** flag that controls if a dashed line shoul be printed before the "message"
- **post_dash = False, type bool** flag that controls if a dashed line shoul be printed before the "message"
- **barrier = True, type bool** enable or disable multi-processing barrier (synchronize mpi processes)
- **root_only = True, type bool** log "message" only with root process or each single one

custEM.misc.misc.**make_directories**(*r_dir*, *m_dir*, *approach*, *para_dir=None*, *m_dir_only=False*)

Initialize the export directories if not existing.

- see **MOD** class description
- **para_dir = None, type str** if *None*, the default para_dir is '*m_dir*/_para'; otherwise, the parameter directory will be initializied according to specified path
- **m_dir_only = False, type bool** initialize only the main mesh directory or also the main results directory

custEM.misc.misc.**mpi_print**(*string*, *val=None*, *pre_dash=True*, *post_dash=False*, *barrier=True*)

Utility function that prints only from root process in mpi mode.

- **string, type str** string that should be printed
- **val = None, various types** a value that should be printed after the "message"
- **pre_dash = True, type bool** Flag that controls if a dashed line shoul be printed before the "message"
- **post_dash = False, type bool** Flag that controls if a dashed line shoul be printed before the "message"
- **barrier = True, type bool** Enable or disable multi-processing barrier (synchronize all mpi processes)

custEM.misc.misc.**petsc_transfer_function**(*source_func*, *target_func*, *cache=False*, *q=None*)

Build Petsc transfer matrix to interpolate between different meshes in parallel.

Note, this function will crash without error notice if there are more mpi processes used than there are dof in the target mesh.

- **source_func, type dolfin function** data function on source interpolation mesh
- **target func, tpye dolfin function** data function on target interpolation mesh
- **fi = 0, type int** iteration index over frequencies

custEM.misc.misc.**read_h5**(*mpi_comm*, *var*, *fname*, *counter=None*, *ri='data'*)

Utility function to write data files in parallel in *h5* format.

- **mpi_cw, type MPI communicator** communicator attributes for internal usage, FEniCS version dependent
- **var, type dolfin function** data which should be imported, e.g., the E-field solution *E_t_r*
- **fname, type str** destination file name, containing also the export path
- **counter = None, type int** specify the index of the data functions if more than one solution (e.g., multiple Tx or times) should be imported from the same *h5* file
- **ri = 'data', type str** specify a tag to name the data function in the *h5* file container; e.g., *'real'* and *'imag'* can be used to access both parts of the complex field in the same file, reducing the I/O overhead

custEM.misc.misc.**read_paths**(*file_name*)
Utility function that imports the default **'r_dir'** and **'m_dir'** attributes for the **'MOD'** instance from file.

Note, these attributes are always overwritten if the keyword arguments **'r_dir'** or **'m_dir'** are set manually by initializing the **'MOD'** instance with theses keyword arguments.

file_name, type str file name of the file which contains the paths, forwarded internally

- **results_dir, type str** main results directory
- **mesh_dir, type str** main mesh directory

custEM.misc.misc.**resort_coordinates**(*c1*, *c2*)
Resort two arrays with overall identical row entries but different order.

- **c1/c2, type numpy_arrays** two data arrays of shape (n, 3)
- **c1_sort and c2_sort, type numpy_arrays** index arrays which can be used to resort data from array 1 to array 2 and vice versa.

custEM.misc.misc.**root_write**(*writing_function*)
Utility function that stores data to file only from root process in mpi mode.

- **writing_function, type python function** a function for writing e.g. data or json files

custEM.misc.misc.**run_multiple_serial**(*script_name*, **serial_args*)
Utility function which is nearly identical to **run_serial**. The only difference is that a communicator object **ic** is returned that can be used for multi-threading of several serial runs.

A synchronisation with all communicators can be foreced from the root process in mpi mode by collecting all communicator objects in a list and by using the "Barrier" and "Disconnect" mpi4py methods before continuing.

- **script_name, type str** name of the python script that should be called in serial
- ***serial_args, type python list** list of input console arguments for the python script
- **ic, type MPI intercommunicator object** communicator object to communicate with spawned process

custEM.misc.misc.**run_serial**(*script_name*, **serial_args*)
Utility function that calls a script in serial from root process in mpi mode.

- **script_name, type str** name of the python script that should be called in serial
- ***serial_args, type python list** list of input console arguments for the python script

custEM.misc.misc.**smape**(*d1*, *d2*)
Calculate symmetric normalized

custEM.misc.misc.**specify_td_export_strings**(*FE*)
Utility function for specifying time-domain export names.

- **FE, type class** FE approach instance (e.g., **E_vector**)
- **E_str, H_str, mode, type str** data name suffixes to differ between shut-on and shut-off processes and impulse or step-response modeling with the implicite Euler method

custEM.misc.misc.**write_h5**(*mpi_comm*, *var*, *fname*, *counter=None*, *new=True*, *append=False*, *close=True*, *f=None*, *ri='data'*)
Utility function to write data files in parallel in 'h5' format.

- **mpi_cw, type MPI communicator** communicator attributes for internal usage, FEniCS version dependent

- **var, type dolfin function** data which should be exported, e.g., the E-field solution *E_t_r*
- **fname, type str** destination file name, containing also the export path
- **counter = None, type int** specify the index of the data functions if more than one solution (e.g., multiple Tx or times) should be stored in the same *h5* file.
- **new = True, type bool** open new *h5* file, otherwise, continue writing into *f*
- **close = True, type bool** close *h5* file or leave it open for writing further solutions
- **f = None, type str** filename to continue writing in an opened, existing file; this makes only sense if *counter* is not *None* or *0*
- **ri = 'data', type str** specify a tag to name the data function in the *h5* file container; e.g., *'real'* and *'imag'* can be used to store both parts of the complex field in one file, reducing the I/O overhead
- **f, type str** file name of export file if not closed

custEM.misc.profiler module

@author: Rochlitz.R

custEM.misc.profiler.**export_config_file**(*PP*)
Write model parameters - mainly MP and FE instance dictionaries - to file in the specified export directory (*out_dir*) using JSON.

- **PP, type class** PostProcessing instance

custEM.misc.profiler.**export_resource_file**(*PP*)
Write consumed computational resources and times to file in the specified export directory (*out_dir*) using JSON.

- **PP, type class** PostProcessing instance

custEM.misc.profiler.**get_logger**(*log_level*, *profiler*, *out_path*)
Initialize logger for all custEM prints during simulations. The information in the command promt will be printed corresponding to the specified *debug_level* level in the MOD class. Furthermore, a *log* file is stored in the export directory of the results if *profiler=True*, using always the *debug* level for the log files.

- **log_level, type str or int** specify log_level, see MOD class description
- **profiler, type bool** flag to set if *log*-field should be created or not
- **out_path, type bool** path where *log*-file should be created

custEM.misc.profiler.**max_mem**(*total=True*, *to_store=False*, *logger=None*, *fmt='GiB'*)
Print the maximum memory requirements, either for each MPI process or in total.

- **total=True, type bool** Flag if memory consumption should be printed in total or per process
- **to_store = False, type bool** set **True** if value should be returned

custEM.misc.profiler.**release_memory**()
Release unnecessary blocked memory, work in progress.

custEM.misc.pyhed_calculations module

@author: Rochlitz.R

class custEM.misc.pyhed_calculations.**PHC**(*config_file*)
Bases: object

PyhedCalculations (PHC) class for reference solution calculated with **pyhed** sub-module of **COMET** library.

- **calc_reference()** calucalte semi-analytic reference solution for given 1D CSEM setup-

Load a config file (JSON format) from a custEM modeling to obtain the Tx information and calculate results for E and/or H on specified coordinates.

```
>>> from custEM.misc import pyhed_calculations as ph
>>> calculator = ph.PHC("CONFIG_FILE")
>>> calculator.calc_reference(np.array([0., 100., 0.]), 'EH', max_procs=4)
```

calc_reference(*coords*, *EH*, *freq=None*, *max_procs=1*)
Calculate semi-analytic reference solution for 1D CSEm setups.

- **coords, type array of shape (:, 3)** array of target coordinates
- **EH, type str** specify either 'E' for electric or 'H' for magnetic fields
- **max_procs = 1, type int** allowed number of processes for **pyhed** internal multiprocessing

custEM.misc.synthetic_definitions module

@author: Rochlitz.R

custEM.misc.synthetic_definitions.**almost_flat_topo**(*x*, *y*, *h=0.1*)
Almost flat topo specified as function for mesh generation tests.

custEM.misc.synthetic_definitions.**anti_cone_bathy**(*x*, *y*, *h=1000.0*, *z=0.0*)
Example topography with a cone hole for testing bathy-tools.

custEM.misc.synthetic_definitions.**borehole_tx**()
Description of Tx grounded in two different boreholes.

custEM.misc.synthetic_definitions.**cone_bathy**(*x*, *y*, *h=1000.0*, *z=0.0*)
Example topography with a cone hill for testing bathy-tools.

custEM.misc.synthetic_definitions.**double_cone_bathy**(*x*, *y*, *h=1000.0*, *sigma=1.0*, *zl=0.0*)
Example topography with two cones for testing bathy-tools.

custEM.misc.synthetic_definitions.**example_2_loop_tx**()
Crooked loop Tx description for frequency-domain example 2

custEM.misc.synthetic_definitions.**example_3_line_tx**(*x=None*)
Deprecated, not used anymore.

custEM.misc.synthetic_definitions.**example_4_topo_cos_sin**(*x*, *y*, *h=200.0*)
Example topography for frequency-domain example 4, x- and y-directed cosine/sine topography.

custEM.misc.synthetic_definitions.**flat_topo**(*x*, *y*, *h=0.0*)
Flat topo specified as function for implementation tests.

custEM.misc.synthetic_definitions.**gaussian_topo**(*x*, *y*, *h=3000000000.0*, *sigma=1000.0*, *z=0.0*, *x_off=0.0*, *y_off=0.0*)
Example topography with hill descriped as gaussian function.

custEM.misc.synthetic_definitions.**gaussian_topo_hole**(*x*, *y*, *h=3000000000.0*, *sigma=1000.0*, *x_off=0.0*, *y_off=0.0*)
Example topography with hole descriped as gaussian function.

custEM.misc.synthetic_definitions.**pyramid_topo**(*x*, *y*, *h=1000.0*, *slope=1.0*, *z_off=0.0*, *x_off=0.0*, *y_off=0.0*)
Pyramid-shaped topography.

custEM.misc.synthetic_definitions.**roof_topo**(*x*, *y*, *h=1000.0*, *slope=1.0*, *z_off=0.0*, *x_off=0.0*, *ylim=[-1000.0, 1000.0]*)
Pyramid-shaped topography with cutted top (roof).

custEM.misc.synthetic_definitions.**sample_topo_func**(*x*, *y*, *h=100.0*)
Example sine topography for test scripts.

custEM.misc.synthetic_definitions.**subtopo_func1**(*x*, *y*, *h=200.0*)
Example sine-shaped subsurface topography in x-direction.

custEM.misc.synthetic_definitions.**subtopo_func2**(*x*, *y*, *h=0.1*)
Slope of subsurface layer in y-direction.

custEM.misc.synthetic_definitions.**surface_anomly_outcrop_1**()
Example outcrop polygon for surface anomaly.

custEM.misc.synthetic_definitions.**surface_anomly_outcrop_2**()
Example outcrop polygon for surface anomaly.

custEM.misc.synthetic_definitions.**surface_anomly_outcrop_3**()
Example outcrop polygon for surface anomaly.

custEM.misc.synthetic_definitions.**topo_func1**(*x*, *y*, *h=100.0*)
Example cosine-topography in y-direction.

custEM.misc.synthetic_definitions.**topo_func2**(*x*, *y*, *h=200.0*)
Slope in x-direction.

custEM.misc.synthetic_definitions.**topo_func3**(*x*, *y*, *h=200.0*)
Example sine-topography in y-direction.

custEM.misc.synthetic_definitions.**tx_slope**(*x*, *y*, *h=10.0*, *z_off=0.0*, *x_off=50.0*, *flank=10.0*, *ylim=[-60.0, 60.0]*)
Topography description building a slight slope just beneath a loop_r Tx.

custEM.misc.synthetic_definitions.**valley**(*x*, *y*, *height=1000.0*, *z=0.0*)
Example topography with a valley for testing bathy-tools.

Module contents

misc

Submodules:

- **misc** for general utility functions used in all submodules
- **profiler** for analysing model and performance statistics
- **pyhed_calculations** for calling **pyhed**
- **synthetic_definitions** for defining supporting python functions
- **anomaly_expressions** for defining anomalies with FEniCS, deprecated

custEM.post package

Submodules

custEM.post.interp_tools_fd module

@author: Rochlitz.R

class custEM.post.interp_tools_fd.**FrequencyDomainInterpolator**(*PP*, *dg_interpolation*, *self_mode*, *fs_type=None*)

Bases: *custEM.post.interpolation_base.InterpolationBase*

Interpolator class for interpolating 3D modeling results on arbitrary meshes, which can be generated using this class as well.

There are four ways of interpolation:

1. Interpolation is conducted in serial and might take some time, because exported solutions need to be imported again in serial for the serial interpolation. Anyway, multi-threading is automatically used to run several interpolation tasks simultaneously.
2. Parallel interpolation based on a PETSc transfer matrix. Fields are always interpolated on a discontinous Lagrange VectorFunctionSpace as auxiliary step, but do not require parallel export of the solutions on the computation mesh and serial import afterwards. That can save lots of time and disc space if only results on interpolation positions are of interest.
3. Function data are evaluated at single single locations (not implemented in this class yet).
4. Automated parallel interpolation using maunally created interpolation-matrices, see *InterpolationBase*

- **interpolate()** this function is a wrapper to call the *interpolate_in_serial.py* file, which calls the FEniCS interpolation routines.
- **interpolate_parallel()** parallel interpolation between different function spaces and meshes.
- **export_interpolation_data()** export interpolated data

export_interpolation_data(*field_real*, *field_imag*, *V_serial*, *fi*, *ti*, *export_name*, *export_pvd*, *aux_h5=False*, *real_serial=None*, *imag_serial=None*)

Export interpolated results as complex numpy arrays and *pvd* files to be viewed in Paraview. The flag *aux_h5* is used to enable a temorary h5 export for the parallelized interpolation, dumping the interpolated, parallelized FEniCS functions to file. This file is read again in serial to export a conform numpy array. Note, this step requires insignificant effort as functions on the interpolation meshes are usually way smaller than functions defined on the original mesh.

interpolate(*interp_meshes*, *EH=['E_t', 'H_t']*, *interp_p=None*, *fs_type=None*, *dg_interpolation=None*, *export_name=None*, *export_pvd=True*, *interp_dir=None*, *freqs=None*)

Customized interpolation function.

- **quantitiy, type str** Either **E_t, H_t, E_s** or **H_s**.
- **interp_mesh, type str** name of the mesh to interpolate on.

- **mute = False, type bool** set True, if info-interpolation messages should not be printed.
- **interp_p = 2, type int** polyonimal order of interpolation mesh *FunctionSpaces*
- **fs_type = None, type str** Specify if source functions are defined on Nedelec ("ned", default) or Lagrange ("cg") function spaces
- **dg_interpolation = None, type bool** set True or False to overwrite *self.dg_interpolation* for manually enabling discontinuous or continuous interpolation
- **export_name = None, type str** specify a custom export name of the interpolated data if desired.
- **interp_dir, type str** specify, if a custom interpolation directory is desired, otherwise, the defualt interpolation directory is located in the corresponding results directory

interpolate_parallel(*interp_mesh*, *quantity*, *mute=True*, *interp_p=None*, *fs_type=None*, *dg_interpolation=None*, *export_name=None*, *export_pvd=True*, *interp_dir=None*, *fi=0*)

Interpolation in parallel between different function spaces and different meshes. Note that a discontinous Lagrange VectorFunctionSpace might be required as auxiliary step to interpolate between Nedelec and Lagrange spaces.

custEM.post.interp_tools_td module

@author: Rochlitz.R

class custEM.post.interp_tools_td.**TimeDomainInterpolator**(*PP*, *dg_interpolation*, *self_mode*)

Bases: *custEM.post.interpolation_base.InterpolationBase*

Interpolator class for interpolating 3D modeling results on arbitrary meshes, which can be generated using this class as well.

DOCS TO BE FILLED

eval_at_rx(*rx_positions*, *rx_name*, *EH='EH'*, *interp_dir=None*, *mute=True*)

will be added if required, so far no reasonable usage

export_interpolation_data(*electric*, *magnetic*, *V_serial*, *t_id*, *export_name*, *export_pvd*, *aux_h5=False*, *E_serial=None*, *H_serial=None*, *EH='EH'*)

Export interpolated results as complex numpy arrays and *pvd* files to be viewed in Paraview. The flag *aux_h5* is used to enable a temorary h5 export for the parallelized interpolation, dumping the interpolated, parallelized FEniCS functions to file. This file is read again in serial to export a conform numpy array. Note, this step requires insignificant effort as functions on the interpolation meshes are usually way smaller than functions defined on the original mesh.

interpolate(*interp_meshes, EH=['E', 'H'], interp_p=None, dg_interp=False, export_pvd=True, interp_dir=None*)

Customized interpolation function.

- **interp_mesh, type str** name of the mesh to interpolate on
- **mute = False, type bool** set True, if info-interpolation messages should not be printed
- **interp_p = 2, type int** polyonimal order of basis functions on interpolation mesh
- **interp_dir, type str** specify, if a custom interpolation directory is desired; otherwise, the defualt interpolation directory is located in the corresponding results directory
- **dg_interpolation = None, type bool** set True or False to overwrite *self.dg_interpolation* for manually enabling discontinuous or continuous interpolation

interpolate_parallel(*interp_mesh*, *quantity='EH'*, *mute=True*, *interp_p=None*, *dg_interp=False*, *interp_dir=None*)

Customized interpolation function.

- **interp_mesh, type str** name of the mesh to interpolate on.
- **mute = False, type bool** set True, if info-interpolation messages should not be printed.
- **interp_p = 2, type int** polyonimal order of interpolation mesh *FunctionSpaces*
- **interp_dir, type str** specify, if a custom interpolation directory is desired, otherwise, the defualt interpolation directory is located in the corresponding results directory
- **dg_interpolation = None, type bool** set True or False to overwrite *self.dg_interpolation* for manually enabling discontinuous or continuous interpolation

custEM.post.interpolation_base module

@author: Rochlitz.R

class custEM.post.interpolation_base.**InterpolationBase**(*PP*, *dg_interpolation*, *self_mode*)

Bases: object

Interpolator class for interpolating 3D modeling results on arbitrary meshes, which can be generated using this class as well.

FEniCS does not support interpolation between different meshes in parallel. If not using *auto_interpolate* during *solve_main_problem()*, interpolation is conducted in serial and might take some time. Multi-threading is used to run several interpolation tasks simultaneously.

- **update_interpolation_parameters()** update interpolation parameters such as line or slice information for corresponding mesh generation, if necessary, and interpolation.
- **create_line_mesh()** create straight line mesh(es) in serial; horizontal lines following topography are supported, which is controlled by the *on_topo* or *on_water_surface* flags
- **create_slice_mesh()** create straight slice mesh(es) in serial, horizontal slices following topography are supported, which is controlled by the *on_topo* or *on_water_surface* flags
- **create_path_mesh()** create path mesh based on list or array of given coordinates in serial; paths following topography are supported, which is controlled by the *on_topo* or *on_water_surface* flags
- **create_path_meshes()** create multiple path meshes at once
- **find_quantity()** utility function to access field data via strings
- **check_interpolation_meshes()** check if all interpolation meshes exist before interpolation
- **create_interpolation_matrices()** create interpolation matrices for *auto_interpolate* and jacobian computations
- **auto_interpolate_parallel()** interpolate in parallel on the fly using interpolation matrices

```
>>> M = MOD('Test_1', 'halfspace_mesh', 'E_t', p=1,
>>>         owerwrite=False, load_existing=False)
>>>
>>> M.IB.create_line_mesh('x', -5e3, 5e3, 400, z=50.)
>>> M.IB.interpolate(
>>>     'E_t', 'x0_-5000.0_x1_5000.0_y_0.0_z_0.0_n_400_topo_line_x')
```

auto_interpolate_parallel(*EH*, *fi=0*)

check_interpolation_meshes(*interp_meshes*, *m_dir*, *logger*)
Utility function to specify interpolation mesh directories and check if every interpolation mesh in the list is valid and exists.

convert_imp_rhoa_tipper(*E*, *H*, *fi*, *pi*, *impedances*, *rhoa_phase*, *tipper*, *npy_files*, *mat_files*, *derivatives*, *ned_coord_system*)
Convert E- and H-field to MT quantities.

- **E, H, type numpy array ((2, n_pos, 3))** Electromagnetic fields for two source polarizations on all receiver positions on the current rx_path
- **fi, type int** frequency counter
- **pi, type int** path counter
- **impedances, rhoa_phase, tipper, type bool** flags to enable or disable corresponding output
- **npy_files, mat_files, type bool** flags controling output for Python and/or Matlab

convert_mt_data(*impedances=True*, *rhoa_phase=True*, *tipper=True*, *npy_files=True*, *mat_files=False*, *derivatives=False*, *ned_coord_system=False*)

create_interpolation_matrices(*xs*, *return_mixed=False*)

create_line_mesh(*axis*, *start*, *stop*, *n_segs*, ***interp_kwargs*)
Create line interpolation meshes. Only lines parallel to the coordinate axes are supported. Topography for horizontal lines is also supported. For arbitrary lines, use the *create_path_mesh* method.

- **axis, type str** a string which specifies along which axis a line mesh should be generated, valid values are *'x'*, *'y'* and *'z'*
- **start, stop, type float** start and stop cooridinates of the line
- **n_segs, type int** number of equally distributed segments along a line
- ****interp_kwargs,** see **update_interpolation_parameters** docs

create_path_mesh(*path*, *suffix=None*, *rank=None*, ***interp_kwargs*)
Create path interpolation mesh to interpolate on arbitraily distributed points.

- **path, type array of shape (n, 3)** array containing the 3D points to build the path mesh
- **suffix = '', type str** exchange the export name ending "path" with something else, e.g., it is useful to name a path mesh like a line mesh for plotting along a coordinate axis later on
- **rank = 0, type int** use other than root process for multi-threaded serial path mesh generation, e.g., if called from the *create_path_meshes* method
- ****interp_kwargs,** see **update_interpolation_parameters** docs.

create_path_meshes(*paths*, *name=None*, *names=None*, *suffix=None*, ***interp_kwargs*)
Create path interpolation mesh to interpolate on arbitraily distributed points.

- **paths, type list** list containing coordinate arrays to build the path meshes
- **name = None, type str** specify a single name and use increasing integers ids to name the different path meshes; must be *None* if *names != None*
- **names = None, type str** specify a specific name for each path in the list; must be *None* if *name != None*
- **suffix = '', type str** exchange the export name ending "path" with something else, e.g., it is useful to name a path mesh like a line mesh for plotting along a coordinate axis later on

- ****interp_kwargs,** see **update_interpolation_parameters** docs

create_slice_mesh (*axis*, *dim*, *n_segs*, ***interp_kwargs*)

Create slice interpolation meshes. So far, only slices parallel to the coordinate axes are supported. Topography for z-normal slices is supported as well. Alternatively, use the flexible *create_path_mesh* method.

- **axis, type str** a string which specifies along which normal axis a slice mesh should be generated, valid values are *'x'*, *'y'* and '*z*'
- **dim, type float or list of two float** one float specifies constant dimensions in both slice-parallel directions; a list of two floats specifies different dimensions along the two y/z, x/z or x/y axes for axis=*'x', *'y'* or *'z'*, respectively.
- **n_segs, type int or list of two int** one integer specifies an equal number of segments in both slice-parallel directions; a list of two integers specifies a different number of segments along the two y/z, x/z or x/y axes for axis=*'x', *'y'* or *'z'*, respectively.
- ****interp_kwargs,** see **update_interpolation_parameters** docs

export_npy (*complex_data*, *fi*, *pi*, *EH*, *rank=0*)

find_quantity (*string*, *fs_type*)

Utility function to access *dolfin* solution functions via a string.

- **string, type str** Either **'E_t, 'E_s', 'H_t'** or **'H_s'**.

merge_tx_results (*interp_meshes, EH=['E_t', 'H_t']*)

update_interpolation_parameters (***interp_kwargs*)

update and check given keyword arguments for the Interpolator class.

- All interpolation parameters can be specified individually when

calling the line - or slice-mesh generation functions. These values are just specified as class attributes to have reasonable default parameters available.

- **x, y, z = 0., type float** either offset for line interpolation meshes in not-the-axis direction or offset of slice interpolation meshes in slice-normal direction.
- **a_tol = 0.01, type float** numerical tolerance for cutting the line- or slice interpolation meshes. Does not need to be changed in general.
- **update_print_flag = False, type bool** define if updated interpolation keyword arguments should be printed or not. By defualt, the latter are not printed when initialzing the class for the first time but afterwards, this flag is set **True**.
- **self.on_topo = True, type bool** if the mesh has topography in z-direction, horizontal line or slice interpolation meshes at the surface are automatically adjusted to match the crooked surface or follow the surface with a parallel offset.
- **max_procs = "MPI_SIZE", type df.MPI.size(df.mpi_comm_world())** number of parallel processes used during the *mpirun* call
- **force_create = False,** if *True*, overwrite existing interpolation meshes with the same name
- **on_topo = True,** automatically apply the topography used for the mesh-generation to interpolation meshes following the surface directly or with a constant offset
- **on_water_surface = False,** if True, ignore bathymetry values (z < 0) when creating interpolation meshes which include a sea domain and use *z=0* for corresponding positions instead
- **line_name = None, type str** specify a custom name for line interpolation meshes

- **slice_name = None, type str** specify a custom name for slice interpolation meshes
- **path_name = None, type str** specify a custom name for path interpolation meshes

custEM.post.interpolation_utils module

@author: Rochlitz.R

custEM.post.interpolation_utils.**build_line_mesh**(*IB*, *axis*, *start*, *stop*, *n_segs*, *line_name*, *on_topo*)

Generate a 1D line-mesh for interpolation.

see *create_line_mesh* docs

custEM.post.interpolation_utils.**build_path_mesh**(*IB*, *points*, *path_name*, *on_topo*, *suffix*)

Generate an interpolation mesh for interpolation with an arbitrary list of coordinates.

see *create_path_mesh* docs

custEM.post.interpolation_utils.**build_slice_mesh**(*IB*, *axis*, *dim*, *n_segs*, *slice_name*, *on_topo*)

Generate a 2D slice-mesh for interpolation.

see *create_slice_mesh* docs

custEM.post.plot_tools_fd module

@author: Rochlitz.R

class custEM.post.plot_tools_fd.**PlotFD**(*mod*, *mesh*, *approach*, ***plot_kwargs*)

Bases: *custEM.post.plot_utils.PlotBase*

Plot class for visualization of custEM results.

- **import_line_data()** load data interpolated on lines.
- **import_slice_data()** load data interpolated on slices.
- **load_default_line_data()** import all data interpolated on default coordinate-axes.
- **load_default_slice_data()** import all data interpolated on default slices orthogonal to the coordinate axes.
- **plot_line_data()** plot either total or secondary **E** or **H** fields component-wise with real and imaginary parts on an observation line.
- **plot_line_errors()** plot mismatches between two datasets interpolated on the same line or between data and reference solutions component-wise or magnitude with real and imaginary parts on an observation line.
- **plot_slice_data()** plot either total or secondary **E** or **H** fields component-wise with real and imaginary parts on an observation slice.
- **plot_slice_errors()** plot mismatches between two datasets interpolated on the same slice or between data and reference solutions component-wise or magnitude with real and imaginary parts on an observation slice.
- **add_to_slice_plot()** plot 2D data in an existing (sub)figure for custom illustrations.
- **add_errors_to_slice_plot()** plot mismatches of components or field magnitudes of 2D data in an existing (sub)figure for custom illustrations.

- **add_3D_slice_plot()** adds visulaization of 2D data (e.g., with topography) to an existing (sub)figure as 3D plot.
- **init_main_parameters()** utility function for the import-functions.
- **init_component_integers()** utility function for evaulating the correct coordinate directions.
- **calc_pyhed_reference()** deprecated, might not work anymore, might be re-implemented soon.
- **eval_properties()** evaluate plot-properties.
- **reduce_slice_data()** if slice data very interpolated on a grid that is too fine, one could apply a stride (default *1*) to reduce the amount of data, otherwise, 2D visualization can become very slow or even crash.
- **print_import_error()** print warning, if specified data could not be imported
- **get_coords()** evaluate correct coordinates for 1D or 2D visualization
- **get_coord_names()** utility function for **get_coords()**
- **init_cmap()** initalize colormap (adjust colormap type and range)
- **add_cbar()** add a proper colorbar to a figure
- **rel_errors(), abs_errors(), ratio()** calculate relative, absolute errors or ratio between two datasets using the absolute ('abs') values of the latter
- **mean_errors(), median_errors()** calculate **mean* or *median* of an error distribution using the absolute ('abs') values of the latter
- **save_plot()** save a plot in the save-directory
- **further class-external utility functions are defined at the bottom!** see description at the bottom

abs_angles(*data1*, *data2*, *name=None*, *store=True*)
Calculate angles between vectors

abs_errors(*data1*, *data2*, *name=None*, *store=True*)
Calculate absolute errors / mismatch between two datasets.

add_3D_slice_plot(*fig*, *full_name*, *slicE*, *sp_pos=111*, *fs=12*, *q_slicE=None*, *q_name=None*, *ri='real'*, *cbar=True*, *EH='E'*, *mesh=None*, *n_colors=101*, *q_length=0.2*, *label=None*, *c_lim=None*, *cmap='magma'*, *equal_axis=True*, *err_type='rel'*)
Plot 2D EM-data with real 3D geometry (topography) at specified positions in a given figure. In development (there are still some hard coded keyword arguments which need to be made variable!)

- **fig, type matplotlib figure object** figure to plot in
- **full_name, type str** full name: a valid key for the dictionary **self.slice_data**, (characterization via **mod** or **key** args. in progress).
- **sliceE, type str** slice name: a valid key for the dictionary **self.slice_coords**
- Most keyword arguments are identical to the ones documented in the **plot_line_data()**, **plot_line_errors()**, **plot_slice_data()** and **add_to_slice_plot()** functions. Therefore, only new keyword arguments are listed below:
- **q_name = None, type str** valid key for the dictionary **self.slice_data**, used to select a dataset for plotting vector arrows with a lower grid/data density than the model specified via **full_name** for the contour plot.

- **q_sliceE, type str** valid key for the dictionary **self.slice_coord**, used to select a dataset for plotting vector arrows with a lower grid/data density than the model specified via **slicE** for the contour plot.
- **q_length = 0.2, type float** specify a custom length for all vector arrows.

add_cbar(*fig, c_lim, cmap='magma', cmap2=False, pos=[0.9, 0.12, 0.03, 0.75], horizontal=False, pos2=[0.05, 0.12, 0.03, 0.75], diff=0.05, n_colors=101, fs=12, symlog=False, label=None, label_pos=[0.0, 1.02], additional=0, where='left'*)

Add colorbar to a 2D / 3D data plot at a custom position with custom style options.

add_errors_to_slice_plot(*ax, pos, err_key=None, key=None, key2=None, comp='mag', mask=None, ri='real', EH='E', mesh=None, n_colors=101, label=None, fs=12, e_lim=[0.1, 100.0], tcolor='#002C72', cmap=None, equal_axis=True, err_type='rel'*)

Add a selected dataset, i.e., a specific component, real or imag part, to a subplot specified by a given figure axes object and position. In development (quiver option needs to get variable and no hard coded keyword arguments!)

ax, type matplotlib figure axes object axes object of a matplotlib figure which might be separated into numerous subplots.

pos, type list of two entried specify the subplot position of the **ax** object to plot in.

err_key, type str specify error data to plot - valid key for one of the dictionaries: **self.rel_mag_errors**, **self.rel_comp_errors**, **self.abs_mag_errors**, **self.abs_comp_errors**, **self.ratio_mag_errors**, or **self.ratio_comp_errors**.

- All keyword arguments are identical to the ones documented in the **plot_line_data()**, **plot_line_errors()**, **plot_slice_data()** and **add_to_slice_plot()** functions. Therefore, only new keyword arguments are listed below:

add_phase_bar(*fig, pos=[0.9, 0.12, 0.03, 0.75], dummy=False, n_colors=3, fs=12, label='$\\phi$ (°)', p_lim=1.0, horizontal=False, where='left', diff=0.05, label_pos=[0.0, 1.02], delta=False, absdelta=False*)

Add colorbar to a 2D / 3D data plot at a custom position with custom style options.

add_to_line_plot(*ax, pos, errors=False, mod=None, mod2=None, comp='x', log_scale=None, tcolor='#002C72', ap=False, key=None, key2=None, sf=False, sf2=False, grid=True, ri='real', EH='H', mesh=None, label=None, km=1000.0, llabel=None, fs=12, ylim=[0, 100.0], xlim=[-5.0, 5.0], err_type='rel', **kwargs*)

Add either line data or errors to a specific subplot.

- All keyword arguments are identical to the ones documented in the previous methods.

add_to_slice_plot(*ax*, *pos*, *comp='mag'*, *ri='real'*, *n_colors=101*, *mod=None*, *mesh=None*, *key=None*, *EH='E'*, *label=None*, *fs=12*, *c_lim=None*, *sf=False*, *tcolor='#002C72'*, *horz_only=False*, *cmap=None*, *equal_axis=True*, *quiver=True*, *title=None*, *save=True*, *s_name=None*)

Add a selected dataset, i.e., a specific component, real or imag part, to a subplot specified by a given figure axes object and position. In development (quiver option needs to get variable and no hard coded keyword arguments!)

ax, type matplotlib figure axes object axes object of a matplotlib figure which might be separated into numerous subplots.

pos, type list of two entried specify the subplot position of the **ax** object to plot in.

- Most keyword arguments are identical to the ones documented in the **plot_line_data()**, **plot_line_errors()** and **plot_slice_data()** functions. Therefore, only new keyword arguments are listed below:
- **comp = 'mag', type str** specify if either the **'x'**, **'y'**, **'z'** component or the magnitude (**'mag'**) should be visualized.
- **ri = 'real', type str** plot either **'real'** or **'imag'** parts of the data.
- **quiver = True, type bool** plot vector, this option is not optimized yet and work in progress.

amp_phase_comp(*data*, *name=None*, *store=True*, *shift=0.0*)
Calculate component-wise amplitudes and phases.

amp_phase_mag(*data*, *name=None*, *store=True*)
Calculate amplitude and phase of vector magnitudes.

import_line_data(*linE*, *mod=None*, *mesh=None*, *approach=None*, *EH='EH'*, *sf=None*, *path=None*, *key=None*, *stride=1*, *tx=None*, *freq=None*)
import line data from regular line (line-name must end with either "_x", "_y" or "_z" for x-, y-, z-directed lines, respectively).

- **linE, type str** name of the line-mesh on which the required data were interpolated
- **mod = None, type str** specify if a different model than the overall defined **self.mod** (input when initiallizing Plot instance) should be imported
- **mesh = None, type str** specify if a different mesh than the overall defined **self.mesh** (Plot instance) should be imported
- **approach = None, type str** specify if a different approach than the overall defined **self.approach** (Plot instance) should be imported
- **EH='EH', type str** Alternatively **E** or **H**, of solely electric or magentic fields should be imported
- **sf=False, type bool** Set **True**, if secondary fields should be imported instead of total fields
- **path = None, type str** if non-default interpolation directorys were chosen, specify the path in which the interpolation results are located
- **key = None, type str** specify a **key** which can be used all over the **Plot** class functions to identify the imported data with this name; Otherwise, the name will generated by default *(mod + '_E_t_on_' + linE)*
- **stride = 1, type int** downsample observation points; return only every *stride*-th point
- **tx = None, type int** if multiple Tx have been computed in the same model, specify which one to import
- **freq = None, type int** if multiple frequencies have been computed in the same model, specify which one to import

import_point_data(*points*, *mod=None*, *mesh=None*, *approach=None*, *EH='EH'*, *sf=None*, *path=None*, *key=None*, *tx=None*, *freq=None*)
import line data from regular line (line-name must end with either "_x", "_y" or "_z" for x-, y-, z-directed lines, respectively).

- **points, type str** name of the path-mesh on which the required data were interpolated
- **mod = None, type str** specify if a different model than the overall defined **self.mod** (input when initiallizing Plot instance) should be imported
- **mesh = None, type str** specify if a different mesh than the overall defined **self.mesh** (Plot instance) should be imported

- **approach = None, type str** specify if a different approach than the overall defined **self.approach** (Plot instance) should be imported
- **EH='EH', type str** Alternatively **E** or **H**, of solely electric or magentic fields should be imported
- **sf=False, type bool** Set **True**, if secondary fields should be imported instead of total fields
- **path = None, type str** if non-default interpolation directorys were chosen, specify the path in which the interpolation results are located
- **key = None, type str** specify a **key** which can be used all over the **Plot** class functions to identify the imported data with this name; Otherwise, the name will generated by default *(mod + '_E_t_on_' + linE)*
- **tx = None, type int** if multiple Tx have been computed in the same model, specify which one to import
- **freq = None, type int** if multiple frequencies have been computed in the same model, specify which one to import

import_slice_data(*slicE*, *mod=None*, *mesh=None*, *approach=None*, *EH='EH'*, *sf=None*, *path=None*, *key=None*, *coord_key=None*, *stride=1*, *tx=None*, *freq=None*)

import slice data from regular slices (slice-name must end with either "_x", "_y" or "_z" for x-, y-, z-directed slice-normals, respectively).

- **slicE, type str** name of the slice-mesh on which required data were interpolated
- **mod = None, type str** specify if a different model than the overall defined **self.mod** (input when initiallizing Plot instance) should be imported
- **mesh = None, type str** specify if a different mesh than the overall defined **self.mesh** (Plot instance) should be imported
- **approach = None, type str** specify if a different approach than the overall defined **self.approach** (Plot instance) should be imported
- **EH='EH', type str** Alternatively **E** or **H**, of solely electric or magentic fields should be imported
- **sf=False, type bool** Set **True**, if secondary fields should be imported instead of total fields
- **path = None, type str** if non-default interpolation directorys were chosen, specify the path in which the interpolation results are located
- **key = None, type str** specify a **key** which can be used all over the **Plot** class functions to identify the imported data with this name; Otherwise, the name will generated by default: *(mod + '_E_t_on_' + linE)*
- **coord_key = None, type str** specify a **coord_key**, if interpolated data are imported several times with a different **stride**, e.g. for 'contour' and 'vector- arorow '(plt.contourf and plt.quiver) visualization at the same time. Later on, the underlying data coordinates can be referenced for plotting by these **coord_key**s
- **stride = 1, type int** specify, if only every *stride*th (e.g., 10th) data point from a dense slice-interpolation mesh should be imported (stride equal in x- and y- direction). Too large 2D datasets make visualization VERY slow and even lead to crashes
- **tx = None, type int** if multiple Tx have been computed in the same model, specify which one to import
- **freq = None, type int** if multiple frequencies have been computed in the same model, specify which one to import

load_default_line_data(*interp_meshes='small'*, ***kwargs*)

laod all or specific line data from a default set of line-interpolation meshes.

- **interp_meshes = 'small', type str** name of the default set of line-interpolation meshes
- further keyword arguments listed in the **import_line_data()** docs

load_default_slice_data(*interp_meshes='small', **kwargs*)
laod all or specific line data from a default set of slice-interpolation meshes.

- **interp_meshes = 'small', type str** name of the default set of slice-interpolation meshes
- further keyword arguments listed in the **import_slice_data()** docs

mean_errors(*key, comp='mag', err_type='rel'*)
Calculate Mean of abs(relative errors between two datasets).

median_errors(*key, comp='mag', err_type='rel'*)
Calculate Median of abs(relative errors between two datasets).

plot_line_data(*mesh=None, EH='EH', mod=None, s_name=None, km=1000.0, ylim=None, xlim=[-5.0, 5.0], grid=True, sf=False, label=None, legend=True, title=None, sharey=False, ap=False, key=None, new=True, fs=12, save=True, **kwargs*)
Plot EM-data (if topo: projected) on 1D straight lines, separated by real and imaginary parts as well as the three vector components. The y-axis is always scaled logarithmically.

- **mesh = None, type str** specify if data are not based on the same coordinates as the ones of the first dataset imported.
- **EH = 'EH', type str** specify if either both (default) or only **E** or **H** fields should be plotted.
- **mod = None, type str** model name to access data from another model that **self.mod** which was imported in the same **Plot** instance for comparison.
- **s_name = None, type str** specify a custom name for saving plot(s) in **self.s_dir**. default is **title + '_' + mesh + '.pdf'**
- **km = 1e3, type float** Automatically convert coordinates from 'm' to km', set to **1.** if 'm' should be kept or something else.
- **ylim = None, type list with two arguments 'y_min' and 'y_max'** specify custom amplitude limits if required.
- **xlim = [-5., 5.], type list with two arguments 'x_min' and 'x_max'** specify custom coordinate limits if required.
- **grid = True, type bool** switch on grid lines or not.
- **sf = False, type bool** set **True** If secondary fields and no total fields of the given model should be visualized.
- **label = None, type str** specify custom line label if required.
- **legend = True, type bool,** switch on legend or not.
- **title = None, type str** specify figure title if required. If **s_name** is **None**, title- and mesh-name will be used to autmatically generate a save-name.
- **key = None, type str** access a certain model via the **key** if required. This overwrites the **mod** argument.
- **new = True, type bool** set to False, if the current data should be added to the last line plot, hence, no new figure will be created.
- **fs = 12, type int** font size for legend and ticklabels.
- **save = True, type bool** flag which controls if the figure is saved.
- ****kwargs, type keyword arguments** additional keyword arguments for plots, e.g., 'lw' or 'color'.

plot_line_errors(*mod=None, mod2=None, mesh=None, EH='EH', err_type='rel', key=None, key2=None, km=1000.0, ylim=[0.01, 100.0], new=True, sf=False, sf2=None, log_scale=True, pyhed_ref=False, magnitude=False, label=None, xlim=[-5.0, 5.0], legend=True, fs=12, title=None, save=True, s_name=None, **kwargs*)

Plot mismatch between two EM datasets, separated by real and imaginary parts as well as the three vector components.

- Most keyword arguments are identical to the ones documented in the **plot_line_data()** function. Therefore, only differing arguments are listed here.
- **mod2 = None, type str** specify the 2nd model for mismatch calculation.
- **key2 = None, type str** access the second model via the **key** if required. This overwrites the **mod2** argument.
- **err_type = 'rel', type str** specify which type of errors between both datasets should be used: Either **'rel'** or **'abs'** or **'ratio'**.
- **log_scale = False, type bool** set **True** for logarithmic error representation.

plot_magnitude_angle_misfits(*mod=None, mod2=None, mesh=None, EH='EH', cmap_width=0.9, err_type='rel', key=None, key2=None, n_colors=101, e_lim=[0.1, 100.0], p_lim=10.0, sf=False, fs=12, a_lim=[1e-12, 1.0], label='err [%]', cmap=None, shift=0.0, equal_axis=True, xlim=None, ylim=None, title=None, save=True, s_name=None*)

Description!

plot_point_data(*mesh=None*, *EH='EH'*, *mod=None*, *s_name=None*, *ylim=None*, *grid=True*, *ap=False*, *sf=False*, *new=True*, *label=None*, *legend=True*, *title=None*, *sharey=False*, *key=None*, *stations=None*, *fs=12*, *save=True*, ***kwargs*)

work in priogress, see also *plot_line_data* docs

plot_point_errors(*mod=None, mod2=None, mesh=None, EH='EH', err_type='rel', key=None, key2=None, ylim=[-100.0, 100.0], new=True, sf=False, sf2=False, log_scale=False, pyhed_ref=False, stations=None, label=None, xlim=[-5.0, 5.0], legend=True, fs=12, title=None, save=True, s_name=None, **kwargs*)

Plot mismatch between two EM datasets, separated by real and imaginary parts as well as the three vector components.

- Most keyword arguments are identical to the ones documented in the **plot_line_data()** function. Therefore, only differing arguments are listed here.
- **mod2 = None, type str** specify the 2nd model for mismatch calculation.
- **key2 = None, type str** access the second model via the **key** if required. This overwrites the **mod2** argument.
- **err_type = 'rel', type str** specify which type of errors between both datasets should be used: Either **'rel'** or **'abs'** or **'ratio'**.
- **log_scale = False, type bool** set **True** for logarithmic error representation.

plot_slice_data(*mesh=None*, *mod=None*, *sf=False*, *km=1.0*, *ap=False*, *n_colors=21*, *EH='EH'*, *label=None*, *fs=12*, *c_lim=None*, *cmap=None*, *equal_axis=True*, *xlim=None*, *ylim=None*, *key=None*, *title=None*, *save=True*, *s_name=None*, *shift=0.0*, *var=None*, *dummy=True*, *kk=0*)

Plot 2D EM-data (if topo: projected) on straight slices, separated by real and imaginary parts as well as the three vector components. The y-axis is always scaled logarithmically. Amplitudes are represented by the color map.

Most keyword arguments are identical to the ones documented in the **plot_line_data()** and **plot_line_errors()** functions. Therefore, only differing arguments are listed here.

- **n_colors = 101, type int** number of different colors for the colormap
- **cmap = None, type str** specify a custom matplotlib colormap via string-name. Currently supported: **'magma'**, **'viridis'** and **'RdBu_r'** by default: plt.cm.magma is used for *E* fields,

  plt.cm.virids is used for *H* fields, plt.cm.RdBu_r is used for error plots
- **c_lim = None, type list with two arguments 'c_min' and 'c_max'** specify custom amplitude limits if required
- **equal_axis = True, type bool** set to **False** for non-equal x- and y-axis aspect ratio

plot_slice_errors(*mod=None, mod2=None, mesh=None, EH='EH', err_type='rel', key=None, key2=None, ap=False, n_colors=101, c_lim=[0.1, 100.0], p_lim=1.0, sf=False, fs=12, pyhed_ref=False, mask=None, label='err [%]', cmap=None, equal_axis=True, xlim=None, ylim=None, magnitude=False, shift=0.0, title=None, save=True, s_name=None*)

Plot mismatches between two 2D EM-datasets, separated by real and imaginary parts as well as the three vector components. The y-axis is always scaled logarithmically. Amplitudes are represented by the color map.

- All keyword arguments are identical to the ones documented in the **plot_line_data()**, **plot_line_errors()** and **plot_slice_data()** functions. It is referred to the latter.

ratio(*data1*, *data2*, *name=None*, *store=True*)

Calculate ratio between two datasets, e.g., primary field magnitudes / secondary field magnitudes.

rel_errors(*data1*, *data2*, *name=None*, *store=True*)

Calculate relative errors / mismatch between two datasets.

save_plot(*EH*, *title*, *save*, *s_name*, *fs=10*, *mesh=None*, *png=False*)

custEM.post.plot_tools_td module

@author: Rochlitz.R

class custEM.post.plot_tools_td.**PlotTD**(*mod*, *mesh*, *approach*, ***plot_kwargs*)

Bases: *custEM.post.plot_utils.PlotBase*

Plot class for visualization of custEM results.

!!! IN DEVELOPMENT !!!

import_line_data(*linE*, *mod=None*, *mesh=None*, *approach=None*, *EH='EHdH'*, *shut_off=True*, *shut_on=False*, *sf=None*, *path=None*, *key=None*, *stride=1*, *tx=None*, *times=None*)

custEM.post.plot_utils module

@author: Rochlitz.R

class custEM.post.plot_utils.**PlotBase**

Bases: object

arrange_line_data(*data*, *comp*, *step*)

calc_pyhed_reference(*key*, *EH='E'*, *line=None*, *mod=None*, *config_file=None*, *mesh=None*)

Deprecated! Might be edited for future purposes.

eval_properties(*mod*, *key*, *label*, *EH*, *sf*, *mesh*, *line=True*, *points=False*)
Evalute names, labels, field qunatities etc. for all kinds of plots.

general_import(*path*, *quantity*, *mesh*, *key*, *comp=None*, *stride=1*, *ckey=None*)
General import functions for interpolated results.

get_coords(*name*, *for_pyhed=False*)
Initialize coordinates (km) for visualization, depending on the direction of the input line- or slice-mesh.

init_cmap(*cmap=None*, *var='E'*)
Initialize colormaps for 2D / 3D data visualization.

init_component_integers(*string*, *line=True*)
Initializes integer mapping to handel x-, y-, and z-directed line and slice coordinates.

init_main_parameters(*mod*, *mesh*, *approach*, *path*)
Initializes the most important class attributes.

load_model_parameters(*mod=None*, *mesh=None*, *approach=None*)
Deprecated! Might be edited for future purposes.

print_import_error(*field_type*, *name*, *path*)
Print an import error if a specified dataset for visulaization could not be found.

rearange_point_data(*data*)

reduce_slice_data(*data*, *comp1*, *comp2*, *step*)
Apply a stride for importing slice datasets to significantly reduce the amount of data to be plotted later on, if the interpolation mesh was set too dense.

custEM.post.plot_utils.**adjust_axes**(*ax*, *equal_axis*, *x_lim=None*, *y_lim=None*, *size=3*, *width=2*)
Adjust axis properties (equal resolution for x- and y-coordinates).

custEM.post.plot_utils.**adjust_log_ticks**(*ax*, *minlog*, *maxlog*, *symlog=False*, *additional=0*)
Improve style of ticks and labels of a logarithmic colorbar.

custEM.post.plot_utils.**adjust_subfigure_box_axes**(*ax*)
As the title and the comment says...

custEM.post.plot_utils.**adjust_ticks_and_labels**(*ax*, *remove_top_right=False*)

custEM.post.plot_utils.**eval_colors**(*data=None*, *n_colors=101*, *c_lim=None*, *quant=99*, *symlog=False*)
Utility function for properly evaluating colorbar ranges using 99 % quantiles.

custEM.post.plot_utils.**make_plain_subfigure_box**(*height=3*, *width=2*, *fs=12*, *log_scale=True*, *sharex=True*, *sharey=True*, *coord_axis=True*)
Create a plain subfigure box for custom illustrations, also used by the default plot functions.

custEM.post.plot_utils.**make_subfigure_box**(*height=3, width=2, fs=12, var=['E', 'E', 'E'], sf=False, log_scale=True, cc='x', ylim=None, err_plot=None, sharex=True, sharey=False, xlim=None, grid=True, sliceplot=False, ap=False, add_km=True, clr=None*)
Initialize default fig/ax objects with six or eight subfigures for all kinds of line- and slice-data plots.

Module contents

post

Submodules:

- **interpolation_base** for interpolation purposes
- **plot_tools_fd** for visualization purposes of frequency-domain data
- **plot_tools_td** for visualization purposes of time-domain data

---

8.2.1.2 Module contents

@author: Rochlitz.R

custEM.**run_tests**()
Starting test functions for conda builds.

8.2.2 Index

genindex

8.3 Tips & Tricks

We introduced this section to share some useful tips based on our and users' modeling experience and aim to successively update the subsequent list.

We ask users to give us a hint if they are missing some information that would have been useful to avoid modeling bugs or saving time for designing scripts.

8.3.1 General advise

- Feel free to contact us in case of any issue, questions, or wish of a functionality which does not exist so far.
- Many useful features such as automated Rx/Tx handling and interpolation, utilitiy functions and others are not represented by the provided examples. Again, feel free to ask if setting up problems feels too tricky.
- If not interested in approaches or a self-validation with different custEM approaches, we strongly recommend to use the total electric-field formulation in frequency-domain. Of course, for flat-surface models, the secondary electric or magentic fields formulations are also very useful. Time-domain approaches and Natural-source modeling can only use the generally applicable total electric-field formulation for now.
- All approaches (except A-V-nodal, only VTI) support generally anisotropic conductivities and magentic permeabilities. Modeling with induced polarization (IP) effects and electric permittivities is only supported in frequency-domain with the total electric field formulation.
- Airspace conductivities are usually set to 1e-10 or 1e-12 Ohmm. For most CSEM setups, we found that 3-4 orders of magnitude contrast between airspace and subsurface conductivities are totally sufficient to obtain accurate solutions. Higher contrasts result in worse conditioning of the system matrix and slightly longer computation times but do not increase the solution accuracy significantly (the effect is << 1 % of relative errors).

- The first time assembly, in particular for p2 function-spaces, can take a while due to initial pre-compiling tasks of dolfin
- Change the recently implemented debug_levels to your needs: 10 is verbose 20 is standard output, might be furhter reduced in future versions 30 is almost quiet as there are only a few warnings 40 is not used internally 50 only print critical errors Uncought exceptions are always raised with traceback information.

8.3.2 Meshing advise

- With TetGen 1.6, a new tetrahedralization algorithm was introduced. If TetGen crashes unexpectetly or freezes, try with the old algorithm via adding the "-D" option to the TetGen call
- In custEM, we always use a right-handed coordinate system: x - pointing positive towards geographical East direction y - pointing positive towards geographical North direction z - pointing positive towards geographical Upwards direction (height values positive above surface and negative below)
- Useful values for TetGen quality: 1.2 for p1 and 1.4 - 2.0 (1.6 often good compromise between accuracy and mesh size) for p2
- It is strongly recommended to use the *refine_rx* method for refining all receiver positions with surrounding triangles of a certain radius, generally useful values: r between 0.1 and 10 m for p1 and r between 1. and 100 m for p2
- Include Tx either on the surface with the *build_surface* method (in case of topography, any z-values will be ignored and shifted to the surface height from the elevation model) or anythere in the mesh by using the *add_tx* method before calling TetGen
- Same holds for Rx. Important: Since the optimum refinement is achieved by using the surrounding triangle-discretization as explained above, the refined shapes must be included as path in the mesh. Obviously, the mesh does not contain the original Rx coordinates as nodes. It is possible to store the original coordinate positions of single or multiple Rx paths to the mesh parameter file (for automated parallel interpolation during the simulation) by using the *add_rx* method.
- If you are working with digital elevation models (DEM) in a gridded *.asc* format, check if the sortation of the gridded data fits to the definition of x/y/z axes in custEM (right-handed coordinate system).
- It is recommended to avoid very short or large segments in a transmitter path description.
- Increase the TetGen tolerance in case of crashes due to the combination of very large domains and very short segments
- Appending a boundary mesh for halfspace models is not required and equivalent to increasing the original domain dimensions about the same factors.

8.3.3 Configuration advise

- Even though described in other instructions separately, we want to point out that for implementation reasons, currently the *mpirun* syntax needs to be used for serial computations as well. For instance, (instead of *python run_example_1.py*) use:

  –> mpirun -n 1 python run_example_1.py

- Mesh generation or visualization scripts always have to be called in serial with the usual command line syntax or using a python editor, for instance:

–> python mesh_example_1.py –> python plot_example_1.py

- In very few cases, the FE simulation crashes during the magnetic field conversion because of the errors:

  –> ERROR: hdgraphFold2: out of memory

  or:

  –> PETSc reason DIVERGED_PC_FAILED, residual norm ||r|| = 0.000000e+00

  They are caused by MUMPS and the best chance to avoid them is to set the MOD class keyword argument *serial_ordering* to *True* or slightly change the number of *mpirun* processes for the simulation of a specific setup.

- The common handling so far is to provide a *mesh*, *run*, and *visualization* script, but it is also possible to add all tasks to a single script, performing the meshing and plotting tasks in serial while embedded in the *mpirun* environment. It can be also useful to start any combination of custEM-scripts with help of a master bash-file. In case of related questions, feel free to ask the developers for further advice.

8.3.4 Syntax change advise

- General anisotropic conductivities and magnetic permeabilites (except A-V nodal approach) are now set as list of 1 value or scalar (isotropic), 3 values (VTI-anisotropic, main diagonal of conductivity tensor), or 6 values (general anisotropic, upper triangle matrix of conductivitiy tensor) for any domain marker. Example: [1e-2, [1e-1, 1e-1, 1e-2]] specifies one isotropic value for domain marker 1 and a vti value for domain marker 2, domain marker 0 always belongs to the airspace using vacuum properties.
- The *overwrite* flag was replaced by two different flags - *oberwrite_results* (alias for old *overwrite* flag) and *overwrite_mesh*. The usage is described in the MOD class.
- To support more general setups for secondary-field computations, the new syntax requires to specify *sigma_ground* (containing all sigma values except the airspace, corresponding to domain markers 1-N) and a *sigma_0* background resitivity vector of equal length. Hence, now any domain can be handled as anomly in contrast to the old syntax, which supported only anomlies in layers (the arguments *sigma_anom* and *anomaly_layer_markers*) are not allowed anymore.
- Spawning additional processes for interpolation was disabled in favor of an optimized serial interpolation routine. It can take now a list of interpolation meshes and qunatities at once which is simpler and faster

8.4 Development

The custEM toolbox is under steady academic development. Aside from a few exceptions, only one author can spend effort on implementing new features, updating the documentation, tutorials and examples, or resolving issues. It is possible that a few examples or methods might not work properly or that syntax errors occur in different custEM versions, even though we tried to fix as many issues as possible in custEM version 1.0.0.

We kindly ask for apology in such cases and encourage users to report any bugs, which is the most effective way to increase the robustness of custEM. Furthermore, please contact us in case of missing features and we will try to add the corresponding functionalities.

8.4.1 Release notes

01.03.2022 - custEM version 1.2.0

- use superior TetGen version 1.6
- syntax adaption of all existing examples and tutorials
- implemented efficient algebra for Jacobi calculations
- additional practical meshing functionalities to handle real survey data
- many updated on custEM wrapper for pyGIMLi to run inversions
- further updates on MT postprocessing

30.07.2021 - custEM version 1.1.0

- immense progress on inv_tools
- efficient explicit Jacobi calculations
- efficient threading over multiple frequencies
- wrapper to use custEM as forward operator for pyGIMLi
- updates on MT postprocessing - needs still testing!

26.04.2021 - custEM version 1.0.0

- final updates for custEM version 1.0
- resolved a few remaining bugs
- overhaul of API doc strings
- update of all finished examples
- update of tutorials

22.04.2021 - custEM version 0.99.95

- pre-release candidate for custEM version 1.0
- resolved a couple of remaining bugs
- resolved issue to convert new meshes with the same name instead of accidently using the already converted onces if not deleted manually
- simplified and updated interpolation mesh generation to work without seperatly spawned processes for more simplicity and robustness
- seperate transmitter currents for multiple Tx in time-domain and DC approaches

03.03.2021 - custEM version 0.99.94

- manual interpolation matrices implemented
- automated receiver-position handling
- dg-paramter functions implemented for all approaches as standard
- handling of general anisotropic sigma/mu/eps values now user-friendly
- natural-source modeling approach debugged and post-processing simplified
- added MT Dublin test model and IP examples
- option of serial-ordering for avoiding/reducing? random MUMPS crashes

10.02.2021 - custEM version 0.99.93

- major development step towards flexibility, user-friendliness and inversion
- logger from standard Python logging module added and working very well, need to replace remaining non-logged prints to write proper log files
- multiple-transmitter support for all time-domain approaches
- electric and magnetic field approaches now support handling multiple frequencies internally in combination with multiple transmitters
- optimized performance exploting reused-factorization as best as possible
- potential approaches can now be updated quite easy to support this feature as well, if required by users
- overhaul of many outdated code parts and related name conventions

28.01.2021 - custEM version 0.99.92

- includes custEM version 0.99.91
- natural-source modeling approach based on E-field formulation tested
- validation against Dublin test model successful, now proper MT class required
- better handling of arbitrary source currents, source and observation time for time-stepping approach added
- fix of cell-identification bug in RHSAssembly class
- started internal support of multiple frequencies instead of looping over several MOD instances handling only a single frequency
- further doc-string updates

9.12.2020 - custEM version 0.99.90

- removed some minor bugs which still led to crashes in various approaches
- doc strings overhaul and minor optimizations of *mod, fem, misc* sub-modules
- test scripts overhaul and test script for time-domain approaches added
- proper documentaion of time-domain modeling examples
- validation of implementation to consider induced polarization parameters

19.11.2020 - custEM version 0.99.16

- summarized updates since v. 0.98, see also gitlab descriptions of commits
- main work in this time was spent on applications and time-domain approaches
- advanced mesh tools with node and edge markers
- pyhed fixed, identical sign conversion
- pyhed incorporated as third-party conda package now, not in wrap directory
- time stepping approach optimized
- rational krylov approach added
- inverse fourier transformation based approach added
- documentation updated
- few syntax changes

12.06.2019 - custEM version 0.98

- conda installer added, installation simplified significantly
- custEM tests during conda package build enabled
- support got FEniCS version older than 2018 disabled
- documentation updated

20.05.2019 - custEM version 0.97

- alternative tranformation factors for PF E-field tested with help of empymod
- compatibility with FEniCS 2019.1
- potential approaches now correctly implemented with identical structure (based on E=iw*A + iw*grad*V) (based on H=iw*F + iw*grad*U)
- corresponding post_processing adapated
- F-U (Tau-Omega) approach added
- bugfixes in meshgen and bathy tools

13.03.2019 - custEM version 0.96

- lot's of bugs fixed (overall compatibility with FEniCS 2018.1)
- multiple RHS supported for total field approaches and framwork for adding this feature for secondary field approaches already implemented
- test, example and tutorial files updated to be again compatible with a this new version of custEM
- All provided scripts were tested also with previous versions based on FEniCS 2017.2, but it is heavily recommended to use this newest version with FEnICS 2018 to benefit from the increased performance!

09.01.2019 - custEM version 0.95

- version 0.94 included
- custEM now compatible with FEniCS 2018.1
- Significant performance boost due to parallel HDF5 support for FEniCS conda package and new MUMPS version, no matrix size restrictions anymore!
- Performance boost due to debugged symmetric MUMPS solver (previous custEM versions were bugged and called the general LU solver)
- Full Maxwell equations with displacement currents for E-field approach available, in progress for H-field approach
- H-field approach debugged
- Time-domain time-stepping approach added (not fully supported yet)
- Bathymetry/Topography meshing tools improved
- Irregular topography data supported

28.08.2018 - custEM version 0.93

- bug for primary E-fields of line sources fixed!
- alternative and superior way of incorporating transmitters in meshes added
- examples 2 reworked
- first steps for meshing costal areas with topography, bathymetry initialized

28.08.2018 - custEM version 0.92

- ReadtheDocs bug fixed: API for modules added!
- restructured repository (custEM modules now subdirectory of repository)
- examples added to table of contents

23.08.2018 - custEM version 0.91

- fixed sorting-bug workaround for primary-field calculations!
- removed minor bugs in meshgen submodule
- added topography meshing tutorial to the tutorials directory
- enabled edge preservation by using not only *.poly* but also *.edge* files for TetGen which is especially an advantage for implementing all kind of CSEM transmitters. As a consequence, TetGen does not need to be called with the '-M' flag anymore for edge preservation, which saves up to 30% of dofs!

Summary of main features before 09.08.2018

- documentation added:
  - documentation of main page, installation notes, etc. added
  - 3 test scripts provided (*tests* directory)
  - 4 examples provided for reproducibility (*examples* directory)
  - 3 tutorials for meshing, FE computation and visualization available
  - automatically updated online documentation on readthedocs.io
- six *approaches* with arbitrary anisotropic conductivities supported for finite element modeling of CSEM problems in frequency domain:
  - Total and Secondary E-field: *E_t* and *E_s*
  - Secondary H-field: *H_s*
  - Total and Secondary A-Phi on mixed elements: *Am_t* and *Am_s*
  - Total and Secondary A-Phi on nodal elements: *An_s* (only xyz-directed anisotropy supported until now, extension possible)
  - automated post-processing
- meshgen tools robustly implemented for land-based or airborne CSEM setups:
  - full-space, half-space or layered-earth tetrahedral meshes
  - topography can be applied to any horizontal surface or interface
  - transmitters or observation lines on surface shifted automatically to follow topography if required
  - domain markers applied automatically and imported for computation from **MOD** instance.
  - arbitrary anomalies within layers or reaching the surface, several regularly shaped bodies pre-defined
  - enclosing tetrahedral mesh to increase domain size and minimize boundary artifacts
- interpolation on lines or slices:
  - lines or slices parallel to coordinate axes
  - crooked "horizontal" lines or slices following topography
  - multi-threading applied to speed up interpolation
- visualization of line or slice data:

- automated line or slice plots of all electric or magnetic field components
- misfit calculations and visualization
- flexible generation of plots with patches combining various data

8.5 Tutorials

The four tutorial files contain the most important commands and parameter options currently available in custEM. Users can comment or uncomment all arguments and keyword arguments to explore the usage of custEM.

8.5.1 NOTICE

- As custEM requies MPI for all realistic simulations, the jupyter notebooks are currently not maintained because embedding MPI runs in jupyter notebooks does not work robustly. Please refer to the python files instead.
- Only the meshgen tutorials work as jupyter notebook. For running simulations, please use the provided Python script equivalent to the jupyter notebook. Run simulations such as the **run_tutorial.py** python script by calling it from the command line with the *mpirun* syntax:

  –> mpirun -n 4 python run_tutorial.py

  Here, 4 parallel MPI processes are used.

Without modification, the provided "run" example requires about 7 GB of RAM and can be conducted in serial or parallel (please remember that even in serial, the "mpirun -n 1 python run_tutorial.py" syntax is required.

The *html* docs are also accessible from the main documentation of **custEM** The *run_tutorial* is independed from the **meshgen_tutorial**, but the **plot_tutorial** depends on the **run_tutorial**. Therefore, for using the **plot_tutorial**, it is necessary to run the FE computations before.

8.5.2 Mesh generation

8.5.3 FEM computation

8.5.4 Visualization

8.6 Examples

The following examples (**examples** directory in the repository) are useful to learn about the usage of different features of custEM. Furthermore, they can be used to reproduce the presented results or to benchmark individual solutions.

Please note that the handling / code syntax / solutions can differ between various custEM versions and lead to incompatibility issues, if the example files version does not match the custEM version.

Anyway, the custEM version 1.0 contains the time-domain and frequency-domain modeling benchmark examples. Additional (unpublished) examples will be added soon to present the handling of advanced modeling problems. In custEM version 1.0, all examples were tested for their functionality using the most recent code syntax.

8.6.1 Frequency Domain (Rochlitz et. al., 2019)

The results of the four frequencdy-domain modeling examples, presented by Rochlitz et. al. (2019), can be reproduced by executing the provided python files in the corresponding directories, denoted *fdom_eN*, *N*=1-4. Further information is provided in the README files of each subdirectory.

8.6.2 Time Domain (Rochlitz et. al., 2021)

The results of the three time-domain modeling examples, presented by Rochlitz et. al. (2021), can be reproduced by executing the provided python files in the corresponding directories. denoted *tdom_eN*, *N*=1-3, Further information is provided in the README files of each subdirectory.

8.6.3 Further examples

8.6.3.1 1D validation of frequency-domain IP modeling

- added as fd_e5

8.6.3.2 MT validation of 3D Dublin test model

- added as mt_e1

Copyright 2016-2018 by R. Rochlitz

8.7 License

The custEM toolbox is licensed under the GNU GENERAL PUBLIC LICENSE (GPL). The terms of the GPL are listed below.

8.7.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007. The original text is provided here:

<http://www.gnu.org/licenses/gpl-3.0.html>

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

8.7.2 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS

8.7.3 0 Definitions

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

8.7.4 1 Source Code

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

8.7.5 2 Basic Permissions

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

8.7.6 3 Protecting Users' Legal Rights From Anti-Circumvention Law

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

8.7.7 4 Conveying Verbatim Copies

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

8.7.8 5 Conveying Modified Source Versions

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

8.7.9 6 Conveying Non-Source Forms

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require

no special password or key for unpacking, reading or copying.

8.7.10 7 Additional Terms

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8.7.11 8 Termination

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

8.7.12 9 Acceptance Not Required for Having Copies

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

8.7.13 10 Automatic Licensing of Downstream Recipients

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

8.7.14 11 Patents

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients.

"Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

8.7.15 12 No Surrender of Others' Freedom

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

8.7.16 13 Use with the GNU Affero General Public License

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

8.7.17 14 Revised Versions of this License

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

8.7.18 15 Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

8.7.19 16 Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

8.7.20 17 Interpretation of Sections 15 and 16

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

8.8 Authors

Major development, maintanance and corresponding author for **custEM**:

Raphael Rochlitz - raphael.rochlitz@leibniz-liag.de

Support for 1D layered-earth EM-field solutions by **COMET**:

Nico Skibbe - nico.skibbe@leibniz-liag.de

Additional support and contact for issues regarding **pyGIMLi**:

Thomas Günther - thomas.guenther@leibniz-liag.de

Python Module Index

C

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X