efield_from_tdJ.py User Guide

Overview

The `efield_from_tdJ.py` Python tool is designed to compute the electric field from a general time-dependent current density (J) input. The tool applies the Jefimenko electric field equation, dependent solely on time-dependent J, from Shao (2016). This equation (equation 17), with the Jefimenko magnetic field equation, can be considered a 3-dimensional generalization of the Uman, McLain, and Krider equations and is particularly useful in studies of lightning physics.

`efield_from_tdJ.py`, when supplied with 1) a configuration file (.ini) specifying a discrete space-time (called the simulation volume), 2) a CSV input file providing the time-dependent J over the simulation volume, and 3) a probe position (not necesarily contained within the simulation volume), will compute the electric field at the probe position and broken down into static, induction, and radiation terms.

Dependencies

The `efield_from_tdJ.py` tool a Python3 script and was developed using Python 3.6. The only package that must be installed on top of the built-in modules is `numpy`. The required install via pip can be performed using the requirements.txt file in the top directory.

Program Inputs

For computing the electric field three inputs are required: 1) a configuration file, 2) an input current density, and 3) a probe position. Each input has its own required formats and are described in detail in the following sub-sections.

Simulation Volume

The efield_from_tdJ.py tool initializes the simulation volume as a SimulationVolume object (defined in `src/simulation_geometry.py`) from a .ini configuation file. The configuration file is provided using the `--geometry-config` option. See the sample and cylindrical configuration files in the `config` directory.

The simulation volume is the discretized volume/grid over which the time-dependent current density is specified. The simulation volume includes time binning (time steps) and spatial binning (cells). For electric field calculations from contributing discrete cells, the cell size itself can drastically impact the result if the probe position (point at which the electric field is calculated) is contained within the simulation volume grid. This is due to the singularly present in the '$1/r^n$' terms of the generalized Uman, McLain, and Krider E-field equation.

As presently written, the tool only supports cartesian and cylindrical coordinate systems.

A valid configuration file needs the following five sections: "geometry," "x_0," "x_1," "x_2," and "t". Each section is described below.

The "geometry" Section

The "geometry" section defines the coordinate frame in the broadest sense. The `coordinate_system` element can have values "cartesian" or "cylindrical" and selects the coordinate system for the simulated volume.

The other three expected elements are `x_0`, `x_1`, `x_2` and they give the names the coordinates of the three dimensional space. For cylindrical coordinates, the expected dimension names are "r", "theta", and "z", which represent the radial, azimuthal, and z coordinates, respectively. For cartesian corrdinates, the expected dimension names are “x”, “y”, and “z.”

The names `x_#` are meant to abstract away the exact coordinate frame in the code.

The “x_#” Sections

Each `x_#` section provides details of the discretization of the coordinate mapped to it in the “geometry” section. These sections provide parameters for initializing a simulated axis (`SimulationAxis` object), three of which define the simulated volume spatial grid.

The elements of the axis sections are:

- `spacing` : The bin spacing along the axis. Accepted values are “linear”, “log”, “log2”, and “log10”. The value “log” is used for the natural log.
- `num_bins` : the number of bins to create along the dimension. The number of bin edges is `num_bins + 1`.
- `units` : the physical units of used when setting the axis domain. The valid values depent on the dimension type (or coordinate). For most spatial axes, the accepted value is “m” for meters. For angular dimensions, the accepted values are “rad” and “deg,” for radians and degrees, respectively. For a time axis (see “The ‘t’ Section”), the expected value is “s”. The “s” and “m” units can be prefixed by a single character giving the unit scale. The prefixes are: “n” for “nano-,” “u” for “micro-,” “m” for “milli-,” “c” for “centi-,” and “k” for “kilo-.”
- `min` : minimum extent of the axis (first bin lower edge) in units of `units`.
- `max` : maximum extent of the axis (last bin upper edge) in units of `units`.
- `bin_width` : can be specify the width of each bin in units of `units`. Only compatible with linear bin spacing. When used, the `SimulationAxis` will create `num_bins` number of linearly spaces bins, each with width `bin_width` and starting with the first lower edge at `min`.

Generally, the minimum elements needed to specify an axis are:

1. `num_bins`, `units`, and `bin_width`.
2. `max`, `num_bins`, `units`, and `spacing`.

For both options, `min` isn’t listed because its defaulted to 0, but the user can change the `min` if needed.

For option 1, the `SimulationAxis` object will create an axis of `num_bins` linearly spaced bins, each of width `bin_width` (the width in units of `units`) and starting at `min`. So, if `bin_width` is 2 m and there are 20 bins, the axis will span from 0 to 40 meters.

For option 2, the SimulationAxis will get the domain from `min` (default 0) to `max` and chop it into `num_bins` pieces spaced according to ‘spacing“.

The “t” Section

The final section, named “t,” defines the time axis. All rules and elements described in ‘The “x_#” Sections’ applies to this section with the exception that `units` must be “s” (seconds) either bare or with a prefix (as described in `units` above).

Current Density CSV File

The current density is supplied to `efield_from_tdJ.py` using the `--ifile` command-line option. The expected input is a simple 7-column comma-separated value (CSV) file. Each line of the input CSV, representing a current density entry from a given cell and time step, is assumed to be formatted as follows:

```
{INDEX_T},{INDEX_X_0},{INDEX_X_0},{INDEX_X_0},{J_0},{J_1},{J_2}
```

where:

- `{INDEX_T}` - (type: integer) The index of the time step for the current density entry
- `{INDEX_X_0}` - (type: integer) The index of the cell for the current density along the first axis.
- `{INDEX_X_1}` - (type: integer) The index of the cell for the current density along the second axis.
- `{INDEX_X_2}` - (type: integer) The index of the cell for the current density along the third axis.
- `{J_0}` - (type: floating point) The value of the first component of the current density.
- `{J_1}` - (type: floating point) The value of the first component of the current density.
- `{J_2}` - (type: floating point) The value of the first component of the current density.

In the above, the first axis is the one defined in the "x_0" configuration file section, the second is from the "x_1" section, and the third is from the "x_2" section. It is assumed that the value of the index is valid for the associated axis (index is greater than 0 and less than `num_bins - 1`). In this way, the exact coordinate represented by the axis (e.g, "x" or "theta") is abstracted away.

As with the notation for the indices, the `{J_0}` component is along axis "x_0," `{J_1}` along "x_1," and `{J_2}` along "x_2".

Probe Position

The last major input is the probe position. This is the location at which the electric field is being measured. In `efield_from_tdJ`, distances ('r') are computed from the probe to each cell center. Due to the '$1/r^n$' terms in the Jefimenko equation, the user should be mindful of the exact position when placing the probe within the simulated volume, particularly when cell sizes are small.

The probe position is supplied from the command line in one of two ways: by cell index or by coordinates. In either case, the `--probe_coord_syst/-s` option is essential in declaring the corrdinate system of the probe position. By default, the coordinate system is "cartesian."

Probe Position By Index

The probe position is specified by index from the command-line by setting the `--is-index` flag and then using the `--indices/-I` option. For example, putting the probe at the cell formed by the intersection of all three axes in cylindrical coordinates would look like:

```
python3 efield_from_tdJ.py -s cylindrical -I 0 0 0 --is-index ...
```

The three arguments of `-I` are the indices in the simulated volume of axes "x_0," "x_1," and "x_2," respectively. So, it is important to confirm that the order of the coordinates agree with the order of the axes set in the "geometry" section of the configuration file.

When placing a probe by index, the exact position used is the corner of the cell defined by the intersection of the 3 lower magnitude edges of each axis bin containing the cell. In the code, this is named the "LLL" or "lll" corner.

Probe Position By Coordinates

The probe position is specified by the command-line by using the `--coordinates/-c` option. If one wanted to put the probe at position '$(x, y, z) = (1m, 2m, 3m)$', the command-line would look as follows:

```
python3 efield_from_tdJ.py --coordinates 1 2 3 ...
```

Notice that `-s` is not set because it defaults to "cartesian."

When setting the position by coordinates, it is important to know that `efield_from_tdJ.py` will assume a particular coordinate order and units depending on the coordinate system. For cartesian, it will assume the arguments are in 'x, y, z' order and all values are in units of meters. For cylindrical, it will assume the arguments are in 'r, θ, z' order and that the units are meters, degrees, and meters, respectively.

The `efield_from_tdJ.py` script will prevent the user from putting a probe on a cell center, thus avoiding divide by zero errors, by shifting the position to the lower-lower-lower ("LLL") corner of the cell whose center the position falls on. A message will be sent to the console when this happens.

Program Outputs

The `efield_from_tdJ.py` program will write three output ASCII text files, one for each electric field term: static, induction, and radiation. These files are place in the directory specified by the `--output/-o` option and are named by the below convention:

`efield_{FIELD_TERM}_{UNIQUE_ID}_{DATE}_{TIME}.txt,`

where: - {FIELD_TERM} : identifies the term of the full electric field. One of: "static," "induction," or "radiation." - {UNIQUE_ID} : a unique string identifier supplied by the user. Default : "probe0". - {DATE} : the date the file was written. Format : "YYYYMMDD" (e.g., 20250603) - {TIME} : the time the file was written. Format : "hhmmss" (e.g., 121212)

The output file ASCII file has 4 columns providing the time, 'E_x' component, 'E_y' component, and 'E_z' component.

For the time column, each timestamp is the center of a time bin of width `bin_width` converted to units of seconds. The number of time entries (and associated electric field entries) is equal to `num_bins` for the time axis multiplied by the scalar `N_TO_BINS`, where `N_TO_BINS` is the argument supplied to the `--ntime_bins/-n` command-line option. By default, '$N\_TO\_BINS = 2$'.

For each electric field component, the value is provided as a decimal value and the units are 'V/m'.

Column labels and units are provided in the header line of the output.

Command Line Interface

For information on how to use `efield_from_tdJ.py` run:

```
$ python3 efield_from_tdJ.py --help
$ ./efield_from_tdJ.py --help # alternatively run as executable
```

The help command listed the following options:

- `-h, --help` : show this help message and exit
- `-s, --probe_coord_syst` : Coordinate system of the provided probe position. Options: "cartesian" or "cylindrical". Default: "cartesian"
- `-c, --coordinates` : Coordinates of the probe position in meters from the origin. The coordinate system determines how these positional arguments are interpetted. If COORD_SYST="cartesian", X_0 = X, X_1 = Y, and X_2 = Z. If COORD_SYST="cylindrial", X_0 = radius (meters), X_1 = theta (degrees), and X_2 = z (meters).
- `--is_index` : If is_index is set, the –indices argument is expected for the probe position. Otherwise, the –coordinates argument is expected.
- `--no-time-offset` : If this flag is set, the origin of the time axis always corresponds with the origin of the input time axis. By default, the time axis is shifted so the signal starts at the 10th time point.
- `-I, --indices` : Indices of the probe position in spatial array. When indices are used, the probe is placed in the corner of the gridded volume element with index (I_0, I_1, I_2). The exact corner is the intersection of the three lower edges of the orthogonal 3D grid. The index order corresponds with the order of dimensions provided in the geometry config.
- `-o, --output` : Required full output directory path
- `-i, --input` : Required full input filepath
- `-u, --unique_id` : String to place in the unique ID token of the output file name. This allows the used to tweak file names to associate the result with some string that identifies it.

- `-n, --ntime_bins` : Number of time bins in the output time axis. The default is 2 times the number of bins as the input time axis.
- `-g, --geometry-config` : Configuration file (INI) providing the simulated volume and geometry
- `--verbose` : If flag is set, higher verbosity is enabled and debugging information is printed.

All important command-line options are discussed in detail in the appropriate sections above.

Directory Structure

The project has the following directory structure:

```
.
|-- config
|-- src
\-- utils
```

The `config` directory contains the two pre-shipped configuration files. Both files define the same simulated volume but `sample_config.ini` contains comments elaborating on different elements.

The `src` directory contains the source code for the `efield_from_tdJ.py` tool, including the `efield_from_tdJ.py` script itself and other files imported for the computation (`efield_math.py`), storing globals (`calculation_globals.py`), reading the input (`read_process_inputs.py`), writing the output (`write_outputs.py`), defining the SimulationVolume class and related classes (`simulation_geometry.py`), and etting physical constants (`physical_constants.py`).

The `utils` directory has two utility scripts. The `bin_to_csv.py` converts a binary file to a `efield_from_tdJ.py` compatible input. This script is unlikely to be of use to a user. The other utility, `plot_ascii_output.py`, is much more broadly useable and creates a PDF plot of a given ASCII output from `efield_from_tdJ.py`.

References

1) Shao, X.-M. (2016) "Generalization of the lightning electromagnetic equations of Uman, McLain, and Krider based on Jefimenko equations," J. Geophys. Res. Atmos., 121, 3363–3371, doi:10.1002/2015JD024717.