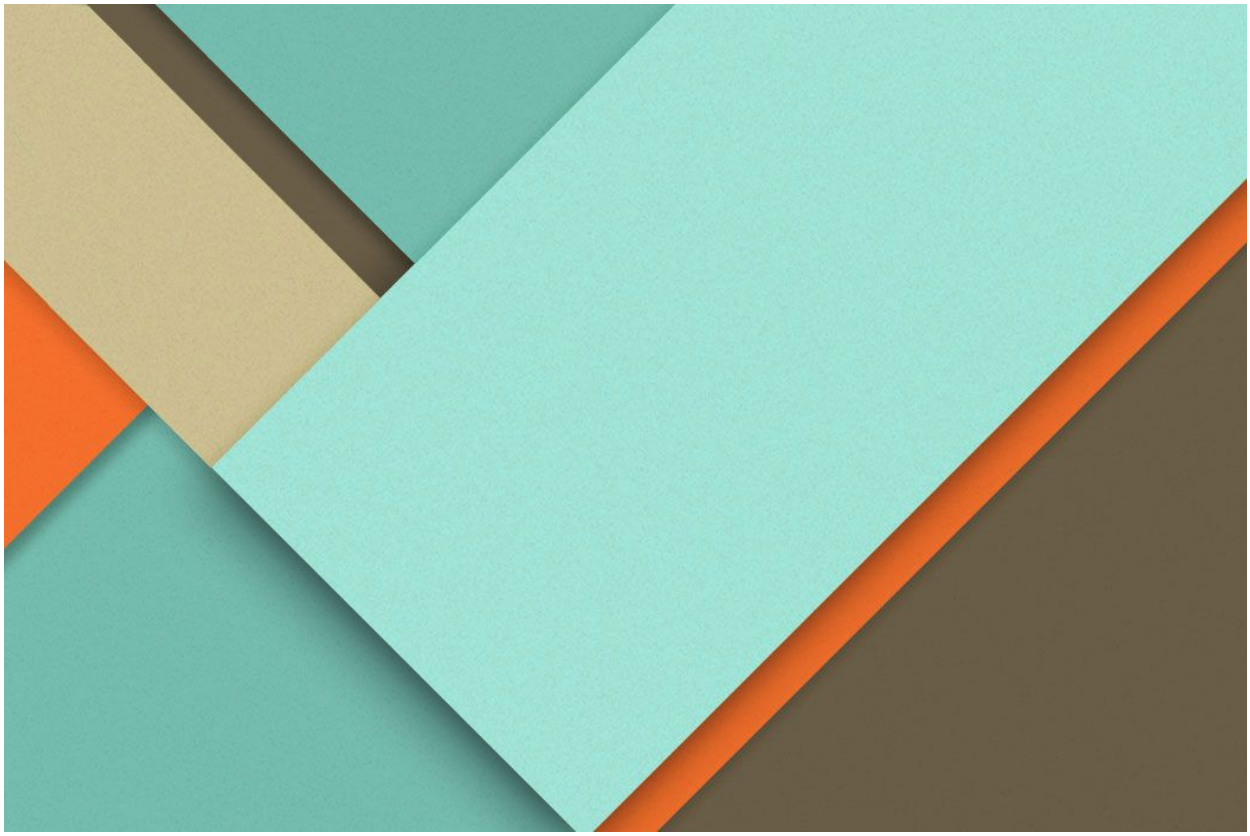


EMOOPIC

an **E**lectro-**M**agnetic **O**bject-**O**riented **P**article-**I**n-**C**ell simulation framework

USER MANUAL

The EMOOPIC Group
Alex Glasser, Jeff Lestz, Noah Mandell, Ian Ochs, Yuan Shi, Denis St. Onge

APC524 Final Project, Princeton University
01.16.2017

Introduction

Welcome to EMOOPIC, an Electro-Magnetic Object-Oriented Particle-In-Cell simulation framework! EMOOPIC is a Particle-In-Cell (PIC) code that is capable of capturing the relativistic dynamics of charged particles in self-consistent electromagnetic fields (EM) using the object-oriented (OO) language C++. EMOOPIC can be used in the study of many astronomical systems like stars and galaxies, as well as many laboratory devices like fusion reactors and semiconductor processors.

Installation

The EMOOPIC source code can be obtained via Github. After installing Git if you don't already have it, issue the following command in a suitable folder:

```
>> git clone https://github.com/DenSto/APC_524_Project.git
```

This will install a folder called APC_524_Project . Contained in this folder are all the source code files (in the src directory), a README file, source code for the unit testing suite (in the test directory), as well as several example and test case input files (in the data directory). External libraries required for running and unit testing are also provided: the Google test source code (in the googletest directory), a tarball to install the HDF5 library, and a tarball to install the libconfig library. Visualization scripts (which require MATLAB) can be found in the visualization directory. Code documentation generated with Doxygen is provided in both LaTeX (in the latex directory) and HTML (in the html directory) formats. Finally, several relevant journal articles relating to the algorithms used in EMOOPIC can be found in the papers directory.

External requirements: Compilers for C and C++ (the GNU compiler suite is recommended), and an MPI library (Open MPI is recommended).

Building the code

To build the code, enter the src directory and type:

```
>> make
```

This will first build and install the HDF5 and the libconfig libraries, if they are not already installed. It will then compile the source code into serial (`EMOOPIC`) and a parallel (`EMOOPIC_par`) executables. Note that compiling for the first time will take several minutes.

Running unit tests

To build the unit tests (to make sure that everything is working properly), navigate to the `test` directory and type:

```
>> make run_tests
```

This will build the Google test unit testing library, if it is not already built, and then build a suite of unit tests, which can be run to test for proper function. The `run_tests` target then automatically runs the full suite of unit tests. Note that unit tests are currently available only for the *serial* version of the code.

Running the code

Serial:

To run the code serially, simply navigate to the src directory and type:

```
>> ./EMOOPIC <path to input file>
```

Parallel:

To run the code in parallel via MPI, type:

```
>> mpirun -np <# of procs> ./EMOOPIC_par <path to input file>
```

This will run the code according to the options in the input file.

Example input files for both serial and parallel execution are included in `data/unitest`.

Input Files & Parameters

An `EMOOPIC` input file follows a standard format that can be read by the `libconfig` library. It consists of five groups: `domain`, `runtime`, `initialization`, `boundary`, and `diagnostics`. We will briefly describe each group; for more details on the specific

parameters that are specified in each group, please refer to the fully commented example input file we have provided, which can be found in `data/unitest/input.txt`.

1. `domain`

   This group specifies parameters related to the physical origin (`xyz0`), extent (`Lxyz`) and decomposition (`nCell`) of the simulation domain. It also specifies the MPI processor decomposition (`nProc`; for the parallel version only, see below).

2. `runtime`

   This group specifies basic runtime parameters pertaining to the length (`nTimesteps`) and physical start time (`startTime`) of the simulation, as well as a parameter controlling the verbosity of debugging statements printed during runtime (`debug`).

3. `initialization`

   This group controls the initialization of both the particles and the fields, and thus consists of two subgroups: `particles` and `fields`.

   The `particles` subgroup specifies the total number of particles (`nParticles_tot`), the total physical number density of all simulation particles (`dens_phys`), the number of different particles species (`nspecies`), as well as physical parameters for each species, such as mass ratio, charge ratio, initial temperature, and whether the species consists of test particles that do not deposit charge (`isTestParticle`). Note that the `dens_frac` parameter sets the fractional density of each species, and thus should be an array of length `nspecies` whose elements sum to unity. Finally, the `relativity` parameter specifies whether to use the relativistic particle pusher.

   The `fields` subgroup specifies parameters related to the field initialization (`init`) and evolution (`electrostatic`). For the initialization, one can choose to either solve the Poisson equation (`init="poisson"`) to produce fields consistent with the initial particle distribution, or initialize constant fields (`init="constant"`). Uniform constant background fields can also be initialized (`B0` and `E0`).

4. `boundary`

   This group controls the boundary conditions for both the particles and the fields, and consists of two subgroups: `particles` and `fields`. For both subgroups, the `conditions` parameter is a six-element array of strings specifying the boundary condition on the left and right of each of the three x, y, and z spatial dimensions. Note that the conditions must be self-consistent or else they will generate warnings (i.e. if one chooses periodic on the left in one of the dimensions, then one must also choose periodic on the right in that dimension).

For the `particles` subgroup, the options for the `conditions` parameter are `"periodic"` or `"reflecting"`.

For the `fields` subgroup, the options for the `conditions` parameter are `"periodic"` or `"external"` (*experts only*).

Experts only: if the `"external"` option is chosen, the `external` subgroup in the `fields` subgroup controls capabilities to externally inject electromagnetic waves into the simulation. See the comments in `data/unitest/input.txt` for more details.

5. `diagnostics`

   This group controls parameters related to which diagnostics to write and the frequency with which they are written (`nstep_fields` and `nstep_particles`). For particles, one can specify how many particles per species to track on each core (`output_pCount`); for fields, one can specify which fields to write (`which_fields`).

Physical units

Note that where relevant, the physical units of a parameter are specified in the comments in the input file.

Input files for parallel version

To make an input file that can be used with the parallel version, `EMOOPIC_par`, simply change the `nProc` parameter, which should be a 3-element array giving the desired number of processors in each dimension. Note that the product of the elements of the `nProc` array must equal the number of processes specified in the `mpirun` execution call above.

Batch submission on Adroit cluster

`EMOOPIC` has been tested on the Adroit cluster at Princeton University. (Note to build on Adroit, use `module load openmpi` to load the Open MPI library before issuing the `make` command.) An example SLURM batch submission file has been provided in `src/run_adroit`, and can be submitted from the `src` directory by typing

```
>> sbatch run_adroit
```

For more details about the batch submission file, please refer to the SLURM documentation.

Output

Running the code generates three types of output files, written to the directory where the input file is located.

1. `output.h5` -- a binary HDF5 file which writes out the fields (specified by the `which_fields` parameter in the input file) as a function of space and time. To view the structure of the file, use

```
>> h5dump -A output.h5
```

To view the full contents of the file, use the command

```
>> h5dump output.h5
```

However, these files can be very large, so use the previous command with caution. Instead, use an HDF5 reader or our provided MATLAB visualization scripts to process the contents of the file.

2. `track_<mpi_rank>_<part_rank>.dat` -- an ASCII file writing the position and velocity of a tracked particle during its trajectory. `<mpi_rank>` is the processor number, and `<part_rank>` is the rank of the particle being tracked on that processor.

3. `history.dat` -- an ASCII file writing the global energy and momentum of the system (broken down into x,y,z momentum, and kinetic, electric, and magnetic energy components).

Visualization

The output files can all be plotted with MATLAB functions located in the `visualization` directory. Each MATLAB function assumes that the data files are in the current working directory. The MATLAB functions also share some common optional input parameters:

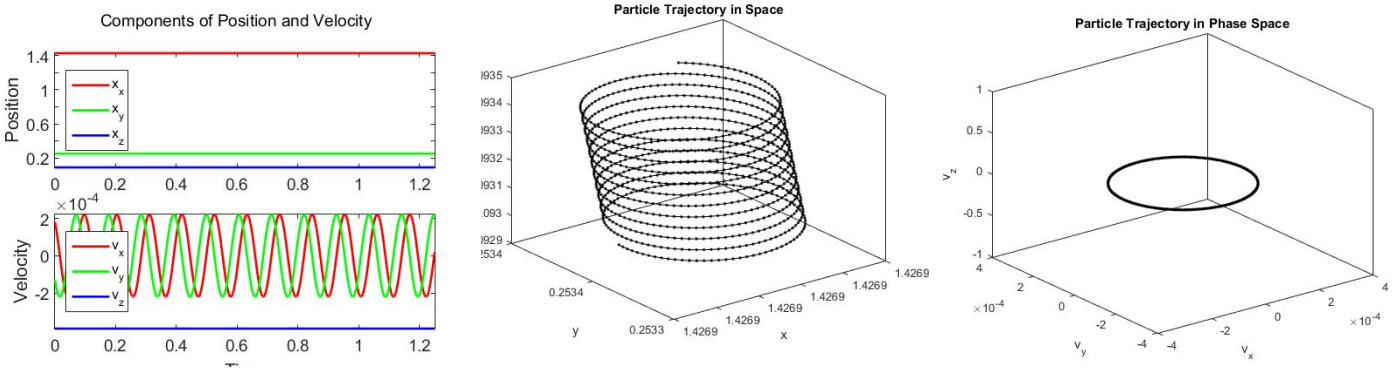
`do_plot` -- when set to 1, plots data, when set to 0, does not plot (0 might be used to return arrays to manipulate without the overhead of generating plots)
`do_save` -- when set to 1, saves figures to files, when set to 0, does not.

Particles

Three MATLAB functions exist for plotting single particle trajectories in `track.dat` files.

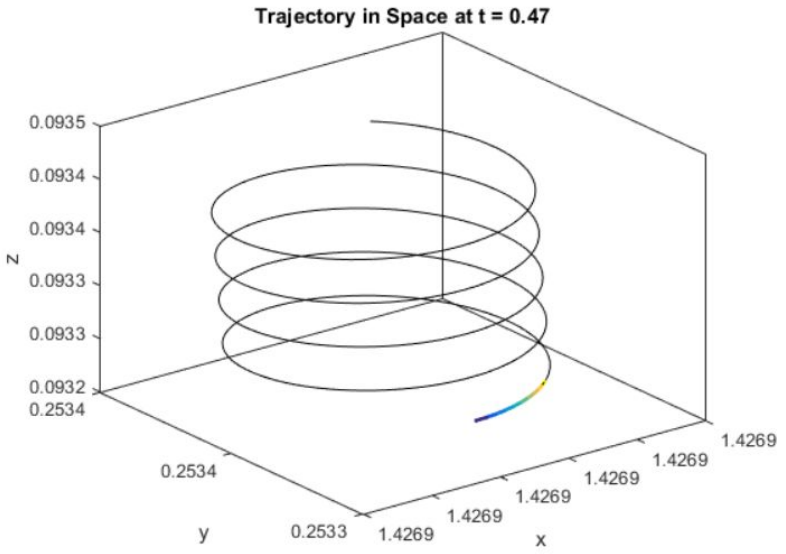
1. `[t,x,v] = plot_particle(mpi_rank,part_rank,do_plot,do_save)`
   Description: Reads a `track.dat` particle trajectory data file. Generates 3 plots:
   1. 1D plots of x,y,z components of position and velocity vs time
   2. 3D plot of position in space
   3. 3D plot of velocity in phase space

   Input:
   `mpi_rank` -- first integer in `track_<mpi_rank>_<part_rank>.dat` filename to read. Represents the rank of the processor the tracked particle was on.
   `part_rank` -- second integer in `track_<mpi_rank>_<part_rank>.dat` filename to read. Represents the rank of the tracked particle on its processor.
   Optional Input:
   `do_plot` -- as documented in section 5. Default value `do_plot` = 0
   `do_save` -- as documented in section 5. Default value `do_save` = 1
   Output:
   `t` -- vector of times in the trajectory
   `x` -- array of x,y,z components of position
   `v` -- array of x,y,z components of velocity

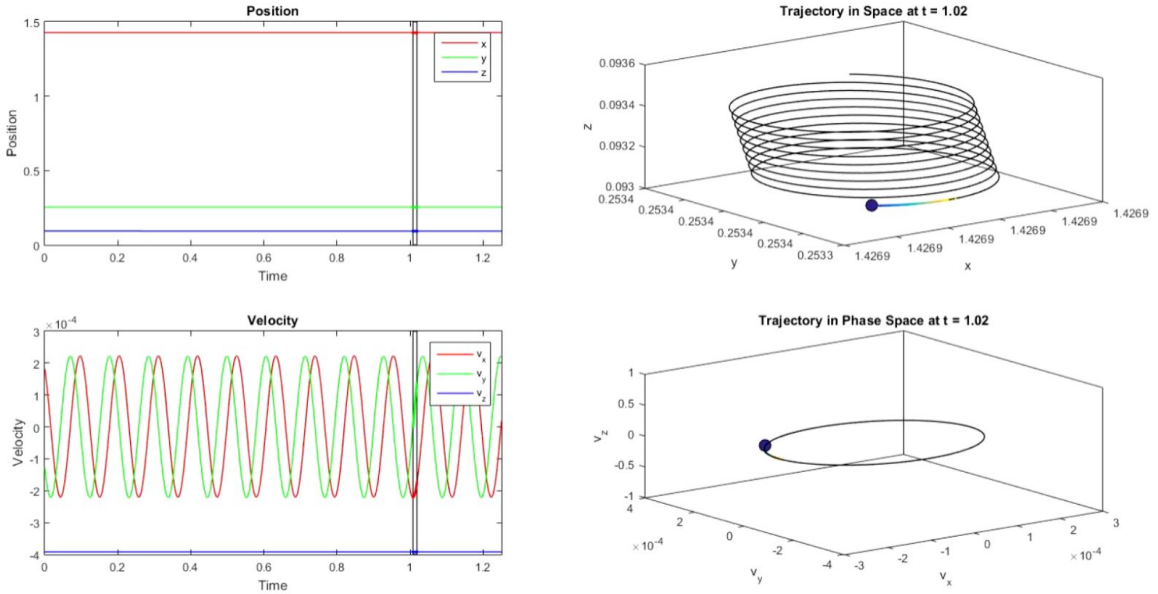


2. `movie_particle(t,vec,movtype,do_save)`
   Description: Plots an animated movie of a particle trajectory in space or phase space.
   Input:
   `t` -- vector of times in the trajectory, such as created by `plot_particle`
   `vec` -- array of particle positions or velocities, such as created by `plot_particle`
   `movtype` -- 1 to indicate position plotting, 2 to indicate velocity plotting
   Optional Input:
   `do_save` -- as documented in section 5. Default value `do_save = 1`

Output:
none



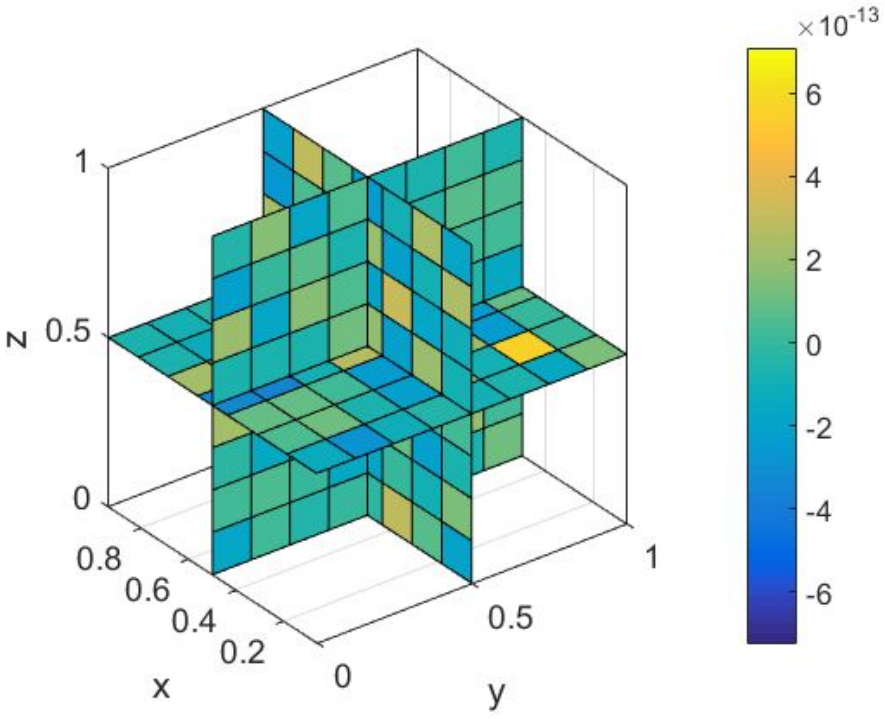
3. `[t,x,v] = movie_full_particle(mpi_rank,part_rank,do_save)`
   Description: Reads a track.dat file to simultaneously plot full particle data in animated movie.
   Makes 4 plots in same figure of position components, velocity components, position trajectory in space, velocity trajectory in phase space.
   Input:
   Same as for `plot_particle`, without `do_plot` option
   Output:
   Same as for `plot_particle`



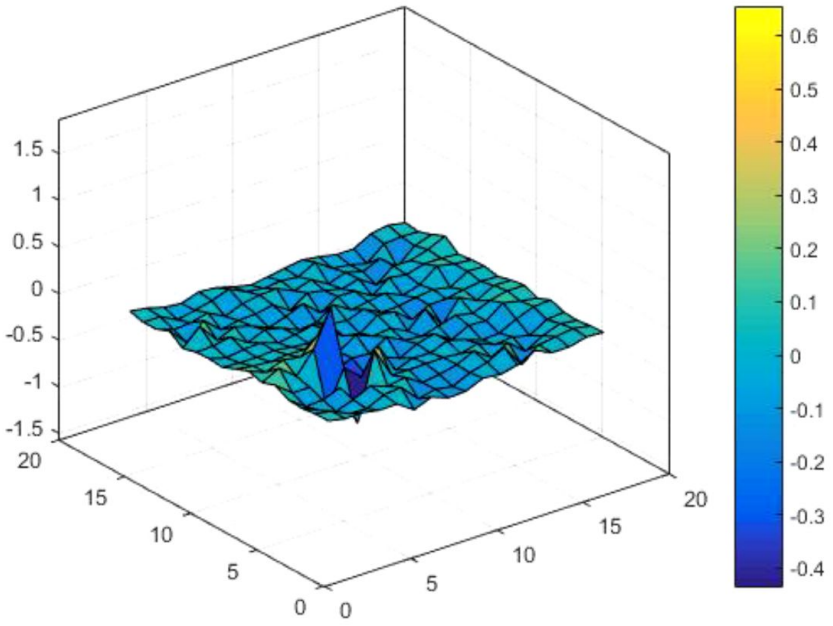
Fields

Two MATLAB functions exist for plotting 3D or 2D field data in time.

1. `[t,x,y,z,field] = plot_field(fstr,itimes,do_plot,do_save)`
   Description: Reads output.h5 hdf5 file to make 3D plot of a single field in time.
   Input:
   `fstr` -- a string determining which field to visualize. Options are: 'Ex', 'Ey', 'Ez', 'Bx', 'By', 'Bz', 'Jx', 'Jy', 'Jz', 'rho'. Note that the `output.h5` file only has the fields that were specified for writing by the `which_fields` parameter in the input file.
   Optional Input:
   itimes -- a vector of indices of time slices to plot. If it is a positive scalar, that individual time slice is plotted. If times = -1, all available time slices are plotted. Default value itimes = -1
   `do_plot` -- as documented in section 5. Default value `do_plot = 1`
   `do_save` -- as documented in section 5. Default value `do_save = 1`
   Output:
   `t` -- vector of all times the field data is written for
   `x,y,z` -- vectors defining the physical grid that the field lives on
   `field` -- 4D array of field data in (x,y,z,t) coordinates



2. `slice_field(field,slicedir,islice,itimes,do_save)`
   Description: Makes 2D plot of field data in time.
   Input:
   `field` -- 4D array of field data in (x,y,z,t) coordinates, such as created by plot_field
   Optional Input:
   `slicedir` -- integer specifying which direction to slice in: 1,2,3 for x,y,z. Default value slicedir = 3
   `islice` -- index of plane to show in sliced direction. Default value `islice = 1`
   `itimes` -- vector of times to plot, same as plot_field. Default value `itimes = -1`
   `do_save` -- as documented in section 5. Default value `do_save = 1`
   Output:
   none



Energy

One MATLAB function exists for plotting the global energy balance of the simulation.

1. `[t,px,py,pz,K,B2,E2,En] = plot_history(do_plot, do_save)`
   Reads a `history.dat` energy history data file. Generates a plot of the simulation's total momentum, kinetic energy, magnetic energy, electric energy, and total energy as a function of time to track energy (non)conservation.

Optional Input:

`do_plot` -- as documented in section 5. Default value `do_plot = 0`

`do_save` -- as documented in section 5. Default value `do_save = 1`

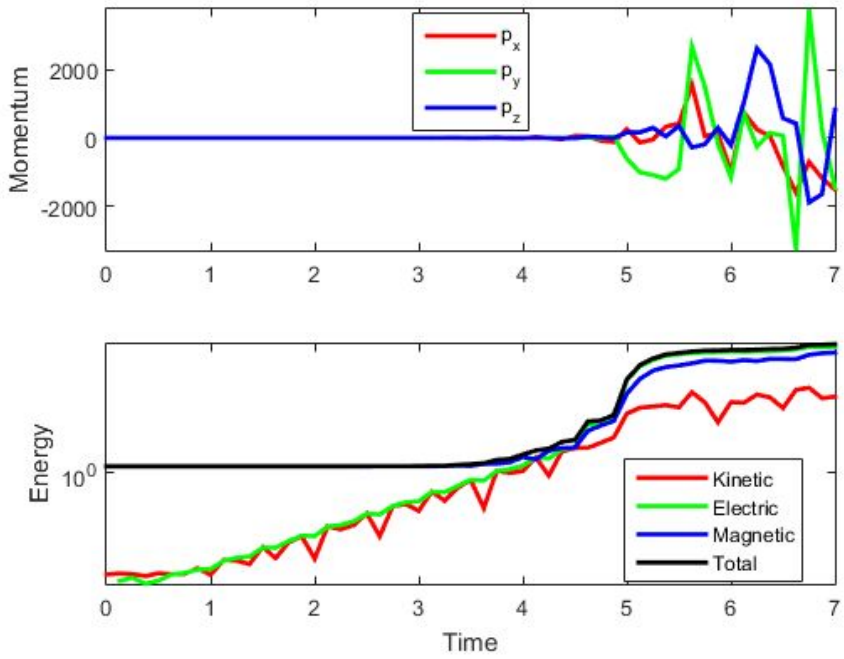
Output:

`t` -- vector of times energy is written out for

`px,py,pz` -- vectors of x,y,z components of momentum summed over all particles in the simulation at each point in time

`K,B2,E2` -- vectors of total particle kinetic energy, magnetic field energy, electric field energy at each point in time

`En` -- vector of total energy (kinetic + magnetic + electric)



Advanced Users/Becoming a Contributor

For those interested in modifying or examining the source code, a good place to start is the Doxygen-generated documentation.

This is available in either hyperlinked PDF format, in `latex/refman.pdf`, or as HTML, at `html/index.html`.