Development of a particle in cell method

How to install and use the programs

Ana Sofia Sousa, Diogo Simões

Advanced Plasma Physics

Master's in Physics Engineering
Instituto Superior Técnico, ULisboa

1 Dependencies

The languages used are `C++20` and `Python3`. Any version of Python from `3.8` should do the trick. The programs are designed to work on Linux. The repository may be cloned from Github as follows:

```
mkdir PIC
cd PIC
git clone https://github.com/fpa-ad/main.git
cd main
cd extern
git clone https://github.com/pybind/pybind11.git
cd ..
mkdir bin
mkdir lib
```

In order for the GUI to work and the results to be visualized, the following packages need to be installed:

```
sudo pip install pyqt5
sudo pip install pyqtgraph
sudo pip install matplotlib
sudo pip install imageio
```

For Ubuntu 22.04 or similar, the following package may be necessary for the windows:

```
sudo apt-get install libxcb-xinerama0
```

2 Structure

At this point, you should have a folder/file structure as follows, minus some unimportant back-end files (images and the like):

```
main
├── extern
│   └── pybind11
├── main
│   ├── FFT.py
│   ├── plots.py
│   └── main.cpp
├── python
│   ├── gui.py
│   ├── plots.py
│   ├── wrapper.cpp
│   └── main.py
├── src
│   ├── Field.h & Field.C
│   ├── interface.h & interface.C
│   ├── myutils.h & myutils.C
│   ├── particle.h & particle.C
│   ├── plasma.h & plasma.C
│   └── sim.h & sim.C
├── lib
├── bin
└── Makefile
```

2.1 Languages

The code developed, as seen by the contents of the `src` folder, is mostly `C++` and respective header files. However, we have also developed a `Python` "wrapper"(`python/wrapper.cpp`), which allows for much quicker usage in this language. The interaction with the main source programs is done through `interface.C`, so as to keep them separate.

There are two examples of how to run simulations, in both languages: `main/main.cpp` and `python/main.py`. The former can be compiled using the Makefile, then executed:

```
1 make main.exe
2 bin/main.exe
```

The command to make the executable file also produces the library necessary. The `Python` file needs only to be executed, after the necessary library is compiled:

```
1 make libfc
2 python3 python/main.py
```

2.2 Source

There are 4 key source `C++` files, which contain the Physics: `Field.C`, `particle.C`, `plasma.C` and `sim.C`.

`particle.C` defines the particles themselves, which are encoded as instances of a class. These objects store the expected attributes, as well as methods to advance their positions and velocities, for a given time step and electromagnetic fields (given at the particles' positions).

`Field.C` defines the *Field* class, wherein the electric and/or magnetic are calculated on the specified grid points and then interpolated to the necessary coordinates (the particles' positions). It uses finite differences methods, with a variety of internal calculations, to solve the Poisson equation.

Next, `plasma.C` encompasses both particles and fields, by defining a class which holds all of the necessary parameters and attributes:

- how many types of particles, how many particles of which type, their charge to mass ratio (not really necessary, but useful) and the *particle* objects themselves;
- how many fields (1 for electric or 2 for electric and magnetic), the background field levels and the *Field* objects themselves;
- the size of the box and spatial step, in both directions.

There are methods to return some of these internal attributes, as needed, as well as the electromagnetic fields' values in a given position. There is also a *move* function, which takes a time step and sets out to calculate the next iteration of the plasma as a whole, advancing the particles and re-calculating the fields.

The simulation itself is controlled an instance of *sim*, defined in `sim.C`. When a new simulation object is created, it has a timestamp (an optional name may be given as well), the usual box/grid parameters and a *plasma* object, of course. Methods are defined here to run the simulation according to the time step, to a maximum time, printing out some "snapshots"along the way in the `output` folder (automatically created).

`myutils.C` has, as the name suggests, some very nifty definitions, which are useful for the remaining files.

The interaction with `Python` is completely confined to `interface.C`, where functions are defined, latter to be "wrapped". These feature input and/or output in `Python`-like objects, but processing in pure `C++`, through the 4 major files already mentioned.

2.3 GUI

For ease of use, a Graphics User Interface was also developed. This program lets the user fully parameterize the simulation, adding different particle types, specifying box parameters and background fields, runs the simulation in a separate thread and then presents the results in a gif form (both the evolution of the particles and a histogram with the evolution of the velocities). The results, text and image snapshots as well as final gifs, are also stored in the `output` folder, created automatically.

The GUI must be executed from the outer `main` folder, after the library is compiled:

```
1 make libfc
2 python3 python/gui.py
```

2.4 Analysis

The program `main/plots.py` takes all of the simulations in the `output` folder and processes them, thus creating all of the plots and gifs, storing them along with the text files.

`python/FFT.py`, as the name indicates, does a Fast Fourier Transform, in order to obtain simulated dispersion relations.