Introduction to Particle-in-Cell Simulations

Plasma Physics Course

18th and 25th October 2012 | Paul Gibbon

Outline

- Lecture 4: Particle-in-Cell Simulation Basics
- Lecture 5: Hands-on Tutorial

Introduction to Particle-in-Cell Simulations

Part I: Getting to know an electrostatic Particle-in-Cell code

18th and 25th October 2012 | Paul Gibbon

Lecture 2: Getting to know an electrostatic Particle-in-Cell code

PIC simulation

The particle-in-cell code currently represents one of the most important plasma simulation tools. It is particularly suited to the study of *kinetic* or *non-Maxwellian* effects. The simplest variation of this technique is a '1D1V'-configuration: 1 space coordinate plus 1 velocity, the numerical behavior of which was first considered by John M. Dawson, one of the pioneers of kinetic plasma simulation, some forty years ago. Advances in computing power have enabled the PIC method to be extended to fully electromagnetic, 3D3V simulation. For tutorial purposes, however, we stick to the simplest possible reduced geometry for the sample PIC codes offered here, which offer both an apprenticeship in 'professional' plasma simulation, as well as plenty of insight into the behavior of laser-produced plasmas.

Difference equations

The heart of an electrostatic code is based on a cyclic iteration of the following difference equations:

Particle pusher:

$$\begin{aligned} v_i^{n+\frac{1}{2}} &= v_i^{n-\frac{1}{2}} + \frac{q_i}{m_i} E_i^n \Delta t, \\ x_i^{n+1} &= x_i^n + v_i^{n+\frac{1}{2}} \Delta t. \end{aligned} \tag{1}$$

Density gather:

$$\begin{aligned} \rho_j^{n+1} &= \sum_i q_i S(x_i - x_j), \\ S &= 1 - \frac{|\, x_i - x_j \,|}{\Delta x}. \end{aligned} \tag{2}$$

Field integration:

$$E_{j+\frac{1}{2}}^{n+1} = E_{j-\frac{1}{2}}^{n+1} + \rho_j^{n+1} \Delta x. \tag{3}$$

Code structure

The routines for the above three steps — PUSH, DENSITY and FIELD, respectively — can be found in the source directory in the corresponding files. The control routine at the top of the code, MAIN – see file `espic.c/f90`) – calls each of these routines in turn, as well as some initialization routines (INIT, LOADX, LOADV) and diagnostic routines (DIAGNOSTICS, PLOTS, HISTORIES). All main variables and storage arrays are described in the header-file `es.h`.

Normalization:

$$t \to \omega_p t; \qquad x \to \omega_p x/c; \qquad v \to v/c$$
$$E \to eE/m\omega_p c; \qquad \phi \to e\phi/mc^2; \qquad n_{e,i} \to n_{e,i}/n_0$$

Prerequisites

- Download site: `https://trac.version.fz-juelich.de/bops/wiki/`
- Tarball: `espic.tar.gz` or `espic.zip`
- Source code (eg C-version): `C/ *.c, es.h, makefile`
- Compiler: GNU's gcc or gfortran
- Graphics:
  - various GLE scripts
  - Gnuplot-script for multiplot-graphics: `multiplot.gnu`
  - your favourite plot program!

Installation

Linux and MAC users:

Unpack the tar file (name may differ) with:

```
tar xvfz espic.tar.gz
```

and ‘cd’ to the installation directory.

Windows users

1. First obtain and install MinGW from `http://sourceforge.net/projects/mingw/files/`. During the Setup check the boxes for compiler(s) you need, plus the MinGW Developer ToolKit This creates a basic Unix environment emulator under Windows.
2. Download and unpack the espic.zip file with an archiving tool. Put this somewhere in your ’home’ directory.
3. Open a MinGW terminal/shell and ‘cd’ to the ESPIC directory.

Installation

The directory structure resulting from unpacking the archive file (zip or tar) should look something like this:

`C/`	...	C source code
`f90/`	...	fortran source code
`graphics/`	...	graphics scripts
`doc/`	...	tutorial worksheets

To compile:

Go to the C/ source directory, and edit the `Makefile`: adjust and tune the flags to match your machine type (FC=gcc). Then do:

```
make
```

Running the code

The purpose of the present exercises is to gain first-hand experience in using an electrostatic particle-in-cell code. Beginners are advised to start with the 'bare-bones' version, and build in their own diagnostics as needed. A more extensive 'tutor' version is also provided to give hints in case you get stuck, and for those wishing to move on to the more advanced projects.

The main code variables and input parameters are described in the file `es.h`. Their initial values can be changed by editing `init.c` and recompiling with `make`.

To launch code from the source directory, just do:
`./espic`

This should generate some output onto the terminal and plain-text format data files with a suffix `.data`. These can be inspected with an editor or plotting program.

Project I: Thermal equilibrium and numerical heating

1. In its present state the code uses *periodic* boundary conditions (`bc-particle =1`): Particles which cross a simulation boundary (left or right) are fed back in at the opposite side with their velocity unchanged (see routine BOUNDARIES). For self-consistency, all field quantities must satisfy: $f(x \pm L) = f(x)$. Using this periodic system, try out a simulation with the following parameters (edit file `init.c`):

```
grid-length=10   vte=0.1          rho0=1.0
dt=0.2           nx=100           ne=5000
ntrun=500        ihist=5          igraph=250
```

   If you change any parameter, you will need to recompile with `make` or possibly `make clean; make`.

2. Run the code and examine the output data with a graphics program (e.g.: gnuplot, xmgr, gle). The datafiles should have names like `densityN.data, fieldN.data`, etc., where N is the snapshot number.

3. The file `hist.data` contains the time-history of the total kinetic energy of the plasma electrons `Ukin`. In order to check the energy conservation of the system, we also need the *potential* energy, given by: $1/2 \int_0^L E(x)^2 dx$.

   Build this additional diagnostic into routine HISTORIES, so that the output file `hist.data` contains 3 energies: (`Ukin, Upot, Utot=Ukin+Upot`).

4. Using the energy conservation diagnostic, experiment with different timesteps `dt`. Determine the maximum value for which energy conservation is maintained within, say 10%.

5. Two further essential diagnostics are the particle *phase space* (v, x) and velocity distribution $f(v_x)$, outputs for which are *missing* from the code in its initial skeletal form. The first of these can be naturally incorporated in routine PLOTS; the distribution function is best computed in a new, separate routine.

6. Now repeat the simulation above with fewer grid points, e.g. `nx=10`, and compare the total energies, phase space and velocity distributions of the two simulations. What do you notice? In the second case, the plasma Debye-length, $\lambda_D = v_{te}/\omega_p$, is not resolved by the grid; an error which leads to the so-called 'aliasing' instability (mixing of plasma modes k, $k + n2\pi/L$ – see Birdsall & Langdon, pp 175–181). Try experimenting further with the grid resolution: how small must $\Delta x = L/N_x$ be to avoid numerical heating?

Project II: Nonlinear plasma waves

1. A longitudinal plasma wave manifests itself as a disturbance in the electron density: $n_e = n_0 + n_1(x, t)$. We can excite such a wave numerically by displacing the initial positions of the particles. If the density perturbation at $t = 0$ is to have the form $n_1 = A \sin kx$, one can show (Birdsall & Langdon, Section 5-2) that the particles need to be displaced by an amount:

$$\delta x \equiv x(t) - x_0 = \frac{A}{n_0 k} \cos kx.$$

2. Make the necessary modifications to the code (for example in routine `loadx`) in order to initialize such a plasma wave structure.
Suggested parameters are as follows (routine `init.c`):

```
grid-length=2*pi A=0.1, k=1       vte=0.05
rho0=1.0         dt=0.2           nx=100
ne=2000          ntrun=150        igraph=pi/dt
```

   Note that for large amplitudes A, some particles may be displaced across the simulation boundary, so an additional call to BOUNDARIES after LOADX may be necessary to avoid 'losing' particles at the edges.

3. How can the statistics or signal:noise ratio in the electron density n_e be improved?

4. (optional) Compare the wave amplitudes and relative phases n_1, E_1, ϕ_1 with those expected from linear theory.

5. How well does the code conserve energy? Examine ΔU_{tot} as a function of Δt. (For the energy diagnostics, see Project I, step 3.)

6. Now try increasing the wave amplitude, e.g.: $A > 0.2$. What happens to the waveforms of density and field? At later times (a few plasma periods, depending on the plasma parameters), one should be able observe some significant wave-particle interaction (trapping and/or wave breaking).

7. Investigate this process further by varying `A, vte, rho0` etc. Hint: it is particularly helpful here to look at the particle phase space (v, x) and velocity distribution $f(v_x)$ (see Project I, step 5 to build in the diagnostic).

8. For very large amplitudes, the electron velocities will eventually become relativistic ($v \sim c$). Since there are no magnetic fields in the model, this can easily be accomodated in PUSH by substituting the proper velocity, $u = \gamma v$ for v. Care should be taken, however, to incorporate this relativistic upgrade in the code diagnostics, such as the kinetic energy $U_{kin} = mc^2(\gamma - 1)$, phase space (p_x, x), and distribution functions $f(p)$.