# Introduction to plasma dynamics - GRB project
# I. Getting started

November 1, 2018

## 1 Theory: Newtonian, electrostatic two-stream instability

We want to model a simple one-dimensional system where two electron beams of equal density are counter-streaming along the $x-$direction. You can look at the two beams as if they were two distinct particle populations, labelled $r$ (beam travelling to the right) and $l$ (beam travelling to the left). For each population $s$, taking the moments of a Maxwellian distribution function gives the Euler equations for the macroscopic quantities

$$\frac{\partial n_s}{\partial t} + \frac{\partial (n_s v_s)}{\partial x} = 0, \tag{1}$$

$$m_s \frac{\partial v_s}{\partial t} + m_s v_s \frac{\partial (v_s)}{\partial x} = q_s E, \tag{2}$$

where $n_s(x,t)$ and $v_s(x,t)$ are the number density and velocity of each species, and $m_s$, $q_s$ are the particle mass and charge (extra task: can you derive the equations above?). The electric field $E$ along the $x-$direction, assuming no magnetic fields, respects Poisson's equation

$$\frac{\partial E}{\partial x} = \sum_s \frac{q_s n_s}{\epsilon_0}, \tag{3}$$

where $\epsilon_0$ is the dielectric constant of vacuum space. For simplicity, you can assume units in which $\epsilon_0 = 1$.

Your first task is to derive the dispersion relation and growth rate of the two-stream instability, which characterises the system above when small perturbations along the $x-$direction are introduced in an equilibrium. Follow carefully the steps below:

- Assume an initial state characterised by the equilibrium (constant in time and space) quantities $n_r = n_l = n_0$, $v_r = -v_l = v_0$, and $E_0 = 0$.
- Perturb the equilibrium of each species with small variations to the equilibrium quantities, i.e. introduce the perturbations

$$n_s \to n_{s,0} + n_{s,1}(x,t),$$
$$v_s \to v_{s,0} + v_{s,1}(x,t),$$
$$E \to E_0 + E_1(x,t).$$

- Introduce these quantities in the Euler and Poisson's equations above to derive the *linearised evolution equations*, that describe the variation in time and space of the perturbations you introduced. In doing so, neglect higher-order terms (e.g. terms such as $n_1 v_1$, which are much smaller than terms such as $n_1 v_0$).
- Assume now a wave-like behaviour for the perturbed quantities, e.g. $n_1(x,t) \propto \exp(ikx - \omega t)$, where $k$ is the wave-number and $\omega$ is the frequency. Substitute this in the linearised equations above. Your final aim is to eliminate all perturbation terms and obtain an equation that only contains $\omega$, $k$, and constant terms such as $n_0$. This is called *dispersion relation*. In the final form you give to this equation, try to introduce the *plasma frequency* $\omega_{p,s} = \sqrt{q_s^2 n_s/(\epsilon_0 m_s)}$ in order to gather some constants and give it a more compact shape.

- You now obtained an equation for the unknown $\omega(k)$, which characterises the perturbations evolving in your system. We are interested in solving this equation, and in particular we want to find all its *imaginary* solutions: these point to instabilities (real solutions are simple waves, which are not of interest here). More specifically, your task now is i) to identify what condition on $k$ allows for imaginary solutions to exist, and ii) assuming this is satisfied, solve the dispersion relation analytically for all imaginary $\omega(k)$.

- Finally, we want to identify the maximum imaginary $\omega$. From the results of your calculations, extrapolate what $k$ corresponds to the maximum possible value of $\omega$. This is what we call the *maximum growth rate* of the two-stream instability.

This is your first task for the theory part. All these calculations are standard steps in the theoretical study of kinetic plasmas. As such, it is easy to find them online and copy-paste; however, I encourage you not to do so, and try to obtain your own result. Remember that whenever you're stuck on something, we can look at it together. This first task is especially important for the next ones, as it gives you an idea of the standard procedure needed to tackle these problems with pen-and-paper calculations.

# 2 Numerics: Your first electrostatic PiC code

The study of plasmas from the kinetic point of view can be successfully carried out with simulation codes. Oppositely to pen-and-paper calculations, these allow for the inspection of nonlinear processes that result from instabilities such as the two-stream. Particle acceleration in GRBs is one of those processes.

The most common approach to the kinetic simulation of plasmas is the employ Particle-in-Cell (PiC) methods. You first goal is to implement the simplest possible PiC algorithm in a computer code. You are free to use your preferred coding language. Python is a good choice, as it allows for easily analysing the results. Here I will try to guide you through the construction of an explicit, electrostatic PiC code. Once again, if you get stuck, talk to me before trying to google for codes that someone else made.

From the theoretical point of view, we want a code that solves the Vlasov-Maxwell system of equations, where we assume a one-dimensional system and one degree of freedom for the particle motion. Let us further assume that no magnetic fields are present. The evolution of the electric field is then given by

$$\frac{\partial E}{\partial t} = -J, \tag{4}$$

where $J$ is the current produced by the particle motion. The particles move according to the nonrelativistic Newton's equations

$$\frac{dx_p}{dt} = v_p, \tag{5}$$

$$\frac{dv_p}{dt} = \frac{q_p}{m_p} E, \tag{6}$$

which drive the change in position $x_p$ and velocity $v_p$ of each particle $p$. In the equations above we have assumed SI units and a normalisation for which $c = \mu_0 = \epsilon_0 = 1$, which is customary for PiC codes.

A PiC method solves the equations above by employing a finite-difference discretization in time and space. It makes use of a computational grid, i.e. a physical domain of length $L$ is divided in $N_g$ cells of length $\Delta x = L/N_g$. Each of the two cell edges is a "grid node", and since there are $N_g$ cells, there are $N_g + 1$ nodes in the domain. On each node we define the discrete electric field $E_g$.

In a PiC method, the particles are represented as a number $N_p$ of "macro-particles". Each of these can be thought of as a collection of physical particles behaving in the same way. In practice, in the code each macro-particle is treated as of it were one single particle obeying Newton's equations of motion.

All in all, we want to obtain a code that solves the discrete system of equations

$$\frac{E_g^{n+1} - E_g^n}{\Delta t} = -J_g^n, \tag{7}$$

$$\frac{x_p^{n+1} - v_p^n}{\Delta t} = v_p, \tag{8}$$

$$\frac{v_p^{n+1} - v_p^n}{\Delta t} = \frac{q_p}{m_p} E_p^n, \tag{9}$$

for each grid node $g$ and each particle $p$, in a number of time steps $\Delta t$.

The exchange of information between particles and grid is carried out via interpolation. In particular, with the information from the particles we must form the current, $J_g$, which drives the evolution of $E_g$ on the grid. Then, by interpolating $E_g$ at the particle location, we can evolve the particle position and velocity. We can formally write this step as

$$J_g = \frac{1}{\Delta x} \sum_p q_p v_p W_{pg}(x_p - x_g), \tag{10}$$

$$E_p = \sum_g E_g W_{pg}(x_p - x_g), \tag{11}$$

where $W_{pg}$ is an appropriate interpolation function. We will see later on what this means in terms of lines of code. The overall algorithm can be represented by a four-step cycle (Figure 1).

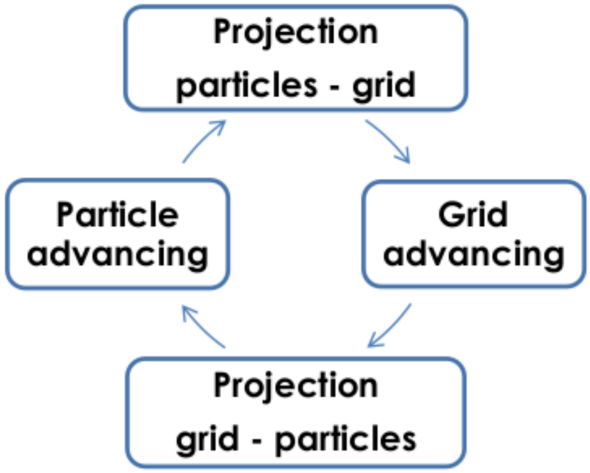


Figure 1: Typical four-step computational cycle of a PiC method. Finite-difference approximations of the field equations are solved on a computational grid, and the field quantities projected at the computational particle locations. The particle are evolved, and the associated information gathered at the grid points. The cycle is then restarted at the new time step.

With the information above, your first task is to construct a skeleton PiC code that at least models correctly the particle-grid interaction. The simulation of actual physics will come after you become familiar with each part of your code, and you understand the relationship between a physical system and its representation in terms of code lines. Follow carefully the steps below:

- Create a new script or program file. Define essential parameters for the grid: number of grid cells $N_g = 32$, domain length $L = 1$, cell size $\Delta x = L/N_g$. Then, define the node position array $x_g = 0 : \Delta x : L$, of size $N_g + 1$ (remember: $N_g$ cells implies $N_g + 1$ nodes). Finally, create appropriate arrays of grid quantities, $E_g$ and $J_g$, with size equal to that of the node position array, and initialise them as 0 everywhere.

- Now create the particle quantities. First, choose a number of particles per cell, e.g. $nppc = 10$. Then, create the particle position array, which has size $N_p = N_g nppc$. Since we want $nppc$ particles in each grid cell, it is easier to first define the initial "particle spatial step" $\Delta x_p = \Delta x/nppc$, and then initialise the particle position array as $x_p = \Delta x_p/2 : \Delta x_p : L - \Delta x_p/2$. Finally, create a particle velocity array (again of size $N_p$). Use your preferred routine to initialise the particle velocities by drawing numbers from a Maxwellian distribution. The Maxwellian spread, in this case, is the thermal velocity $v_{th}$, which we can choose as e.g. 0.01. In Matlab this would be equivalent to writing $v_p = v_{th} randn(N_p, 1)$. Finally, choose a charge-to-mass ratio $q_p/m_p = 1$, constant for all particles.

- We are almost ready to write our time integration cycle, but we need a couple of additional parameters. First, we have to choose a time step $\Delta t$. We typically want to satisfy the condition $v_{th}\Delta x/\Delta t < 1$. For this initial case, we can be extra careful and choose $\Delta t = \Delta x/2$. Finally, we

need to choose the particle charge $q_p$, since by only choosing the charge-to-mass ratio $q_p/m_p$ we did not define this quantity. This is usually done by choosing a *background charge density* $\rho_0 = 1$, and then assigning each particle a charge $q_p = \rho_0 \Delta x/nppc$. This will be a constant number for all particles, hence it needs not be defined as an array.

- We can now proceed with building the four-step PiC cycle. Initialise a `for` cycle over a number $N_t = 100$ of time steps. Your corresponding time array will be $t = 0 : \Delta t : N_t \Delta t$. At each time step, you are required to execute the following steps:
  - Calculate the electric field at the particle location, $E_p^n$ (see below);
  - Evolve the particle position and velocity, $x_p^{n+1} = x_p^n + \Delta t v_p$, $v_p^{n+1} = v_p^n + (q_p/m_p)E_p^n$;
  - Calculate the new current $J_g^{n+1}$ (see below);
  - Evolve the electric field, $E_g^{n+1} = E_g^n - \Delta t J_g^{n+1}$;
- The first and third steps above require interpolation, a feature we have yet to discuss. We are going to employ *first order b-splines* as our interpolation functions $W_{pg}(x_p - x_g)$. We will further assume a support of $\Delta x$ for these functions. In practice, this means that the interpolation functions are defined as

$$W_{pg}(x_p - x_g) = \begin{cases} 1 - |x_p - x_g|/\Delta x & \text{if} \quad |x_p - x_g| < \Delta x \\ 0 & \text{otherwise} \end{cases}. \tag{12}$$

  If you analyse this expression carefully, its practical meaning is that, considering a certain particle at position $x_p$, it will receive information from all the grid nodes at position $x_g$ which are *not farther than a distance* $\Delta x$. The same is true when fixing a node at position $x_g$ and considering to gather information from all particles at positions $x_p$ surrounding it. Let us analyse how this translates in terms of code lines: equation (11) tells us that the electric field acting on a particle located in a specific cell at position $x_p$ is determined by the electric field of the *two* surrounding nodes, because only these two are close enough for the value of the b-spline above not to be 0. Hence, if a particle is located between nodes $i$ and $i + 1$,

$$\begin{aligned} E_p &= E_i \left(1 - \frac{|x_p - x_i|}{\Delta x}\right) + E_{i+1} \left(1 - \frac{|x_p - x_{i+1}|}{\Delta x}\right) \\ &= E_i \left(1 - \frac{x_p - x_i}{\Delta x}\right) + E_{i+1} \frac{x_p - x_i}{\Delta x}. \end{aligned} \tag{13}$$

  As a consequence, step 1 of the PiC cycle above can be easily done in combination with step 2:
  - Create a `for` loop over all particles;
  - For each particle, identify the two surrounding nodes: these will have indices $i =$`floor`$(x_p/\Delta x)$ (if the first node has index 0) and $i + 1$;
  - Calculate $E_p^n$ with the formula above and immediately compute $v_p^{n+1}$ and $x^{n+1}$.
- The only missing step is the calculation of the current $J_g$ (step 3 in the PiC cycle). While a particle receives information only from two surrounding nodes, each node will receive information from an unknown (large) number of particles, i.e., all those particles which are not farther than a distance $\Delta x$. The fact that you do not know in advance how many particles will contribute to a certain node is an apparent obstacle: it seems to require a first loop over all nodes, then a second nested loop over all particles, searching for those particles that are close to a certain node. You can immediately guess that this is a serious waste of computational effort. We are instead going to adopt a smarter procedure that only requires one loop over all particles. This is done in the following steps:
  - Initialise $J_g = 0$ everywhere;
  - Do a `for` loop over all particles. For each particle, locate its surrounding nodes, which have indices $i =$`floor`$(x_p/\Delta x)$ and $i+1$. Since this particle will only contribute to these two nodes, you can immediately store its contribution, i.e.

$$J_i = J_i + \frac{q_p v_p}{\Delta x} \left(1 - \frac{x_p - x_i}{\Delta x}\right), \tag{14}$$

$$J_{i+1} = J_{i+1} + \frac{q_p v_p}{\Delta x} \frac{x_p - x_i}{\Delta x}. \tag{15}$$

At the end of the cycle over the particles, all nodes will have received the correct contribution.

When $J_g$ is computed, it can be used to update $E_g$ and restart the time iteration.

- There is one final point that requires your attention: boundary conditions must be imposed on fields and particles. For this tasks we are always going to assume periodic boundaries. For the grid quantities ($E$ and $J$) this means that the last node on the right of the domain actually corresponds to the first node on the left: they are *the same node*. This does not have a direct consequence on $E$, since you are just updating the electric field point by point via the current $J$. It does, however, have an effect on the computation of the current. At the end of the `for` loop over the particles, when you compute the current, you will have to add the following lines:

$$J_0 = J_0 + J_{N_g+1}, \tag{16}$$

$$J_{N_g+1} = J_0. \tag{17}$$

These are needed because the leftmost node with index 0 is going to receive information only from particles to its right; but since this node actually corresponds to the rightmost node with index $N_g + 1$, it should also receive contributions from the particles to the left of the latter. The vice versa also applies. With the two lines above, you can fix this problem.

For the particles, boundary conditions are conceptually easier: if the domain is periodic, it just means that all particles exiting from the right will re-enter from the left, and vice versa. Hence, at the end of the particle update step you should add the lines:

$$\text{if} \quad x_p > L \rightarrow x_p = x_p - L, \tag{18}$$

$$\text{if} \quad x_p < 0 \rightarrow x_p = x_p + L. \tag{19}$$

This will automatically fix the boundary conditions for the particles.

If you carefully followed all steps above, you should now have a working electrostatic, explicit, non-relativistic PiC code. Your first task ends with the verification that the code is actually working: this is easily checked by running the code for the chosen $N_t = 100$ time steps. If nothing crashes and the calculation ends successfully, you will know that the implementation is correct *if nothing happens*: the distribution function remains a Maxwellian at all times. You can easily check this with a scatter plot of the phase space $(x_p, v_p)$: at the last time step it should look practically the same as it did at the first time step. This is because what you are modelling here is a plasma in thermal equilibrium. Without external perturbations, it will maintain its equilibrium forever.

This task ends once you have a working code. You are welcome to interact with me whenever you are stuck or if something is unclear. Once again, what I ask you here is to put effort in understanding each step and trying to produce your very own implementation of a PiC code. It is quite easy to retrieve such codes on the internet (or copy from prof. Gibbon's examples), but I ask you not to do so. What is more important here is not that you end result is the perfect code, but that you truly understood what the concept behind it is.

An advice: when you code anything, *anything*, keep it clean and commented. Comments are extremely important for you (and others) to remember what you did and why. Never forget to properly comment and indent you loops, if-statements, etc.