Objectives:

* Solve Poisson
* Find the

**Theory**

1. **Principle**

We are modelling the interaction of a plasma with an electromagnetic field by considering charged particles moving in an electrostatic field due to their forces and applied field.

Moving particles

The field is calculated using Maxwell’s equations by knowing the positions of all the particles.

The forces on the particles are found using the electric field (Newton-Lorentz forces: **put the formula**).

We calculate the field from the initial charge, then move the particles (small distances) and recalculate the field due to the particles at their new positions and velocities. **Put the formulas**

We then repeat this procedure for many steps.

We print at the end snapshots at particular times? T = 0 ; t = 60\*dt ; t = 120\*dt

Initial conditions

The cycle starts at t = 0 with some appropriate initial conditions (?) on the particle positions and velocities.

Then the velocities at t=-deltat is extrapolated and the field is calculated.

The grid

We do not use Coulomb’s laws because we are not interested in close encounters because there are very few since we have many particles in a characteristic length (which is the Debye length). We use a spatial grid on which the fields will be calculated. The cells have to be smaller than the Debye length because they need to small enough to resolve a Debye length in order to measure the charge density and thus the electric field.

We can note that the grid provides a smoothing effect by not resolving spatial fluctuations that are smaller than the grid size.

The densities (weigthing)

We establish the densities on the grid and then interpolate the fields from the particles to the particles.

The particles are described by (qi, mi, vi, xi).   
The field is known at the grid points and stored in an array so that it can be recalled easily.

We could approach this with different weighting methods and see the effect.

The first method would be to approximate the particle as a rectangle.

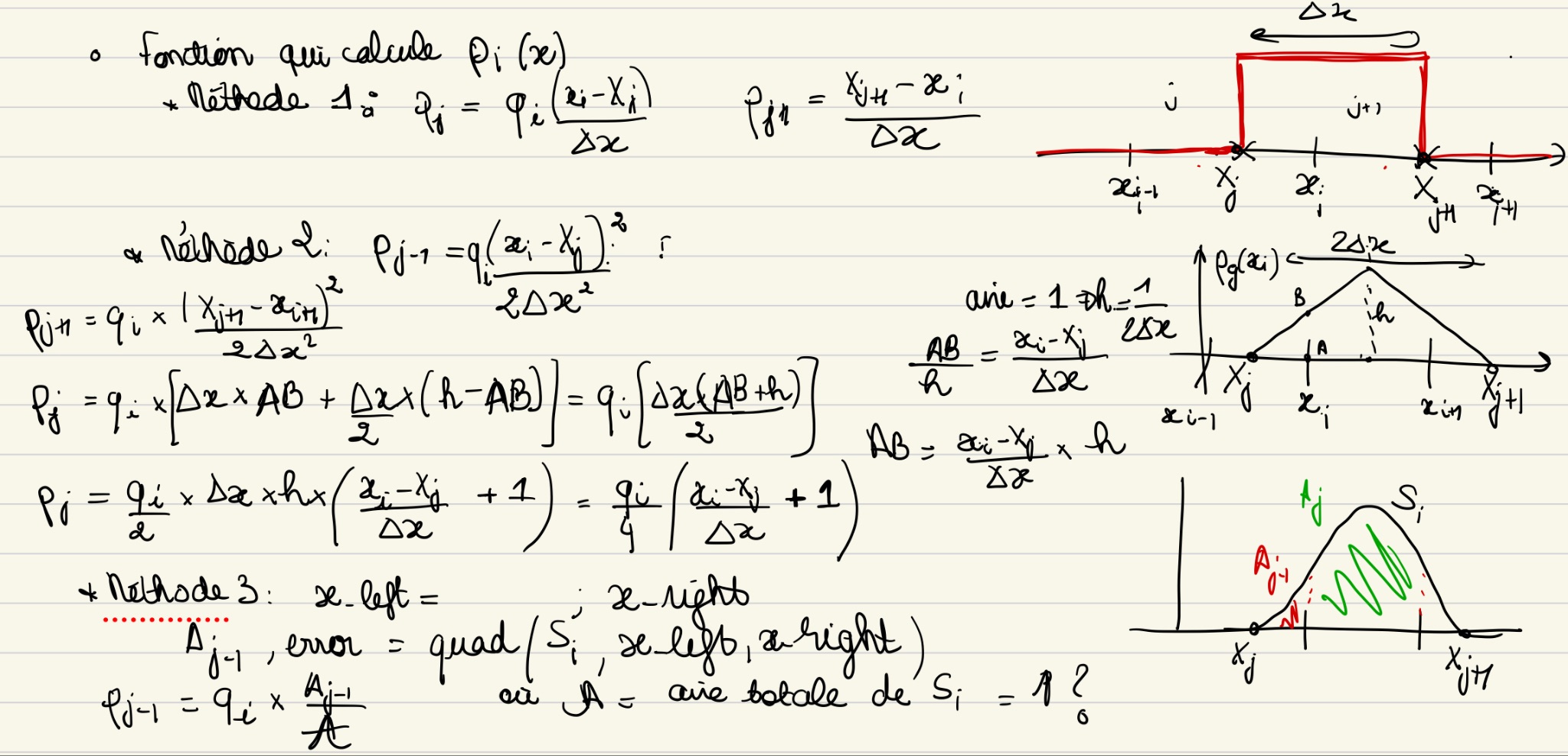
Then rhoi = …. Put drawing.

A second method would be to consider the particles triangular-shaped.

Then rhoi = …. Put drawing.

The third method is to consider the particle with a Gaussian density.

Then rhoi = …. Put drawing.



Obtaining E

We now have the densities rhoi. In order to obtain the electric field in each field, we use phi.

The differential equations to be solved are: (1), (2)

Which are combined to obtain Poisson’s equation (3).

We do the FFT of rho(x) which gives rho(k). We then obtain phi(k).

Indeed, (7).

We do the IFFT to obtain phi(x).

Using (3) we have (4) and (5).

Page 18 ????

Integration: the leap-frog method

Storing xi, vi for several previous time steps would be too costly. We choose a method that needs a minimum of information, the leap-frog method.

Write equations (1), (2), (3), (4) p.13.

X and v are not known at the same time.

F = qE.

vi(t + dt/2) = vi(t-dt/2) + Fi(t)*dt/mi ; xi(t+dt) = xi(t) + vi(t+dt/2)*dt.  
Why need vi(-dt/2) to calculate xi(t).

In order to do that, we need to have enough particles colliding, hence Nd >> 1, and enough particles colliding in the debye cells, hence L >> lambda debye. This is what characterises a collisionless plasma.

Furthermore, the physical behaviour of a plasma is one where electrons and ions move in their Coulomb fields with sufficient kinetic energy to inhibit recombination, so another characterization of a plasma is : Ke / PE = Nd >> 1. (3). Page 5. Why simulate?

We simulate with far fewer particles than is actuall plasams so we will use higher collision rates, ex. Nu << wp.

We only simulate over limited time and space and so can tolerate small errors. We even use mass ratios mion/melectron like 100 instead of 1836. We find that finite time and space gridding may itself produce waves, even instabilities that are non physical.

The ElectroStatic 1-dimensional program, or program ES1.

Code Structure

The code is structured into these modules:

Production, analysis and design.

1. Production

Use- runs the simulation.

Classes

1. Particle-

Represents individual particles in the simulation

Attributes- position, velocity, and charge. Shape? Explore on that

1. Field-

Represents the electric field in 1D

Attributes- grid points , field strength.

1. PicSimulation-

Core class that sets up and manages the simulation parameters (e.g., number of particles, time step) and advances the simulation in time.

Maybe has the functions in it. Also could store the data over time.

Attributes- particle array, the field grid, current and saved data.

Functions

1. initialize\_particles()

Initialises particle positions and velocities according to the initial conditions. An array of particles?

1. compute\_density()

Calculates the charge density at each grid point, using the different weighting methods?

1. solve\_poisson()

Solves Poisson’s equation to obtain the electric field from the charge density, using the computed density.

1. update\_particle\_positions()

Updates particle positions based on their velocities, leapfrog method and stuff.

1. update\_particle\_velocities()

Updates particle velocities based on the electric field.

1. advance\_time\_step()

Advances the simulation by one time step by calling the functions above in sequence.

Main Loop

A `run\_simulation()` function or a method in `PICSimulation` that iterates through time steps, saving relevant data (like particle positions and electric fields) for analysis and visualisation.

2. Analysis

<https://chatgpt.com/share/67337861-bef4-800b-8e27-f17d97a355fc>

Classes

1. DataProcessor

Class to handle the raw simulation data from the PicSimulation and compute quantities like kinetic energy, potential energy, and momentum.

Functions

1. Any calculation for data we’d like to extrapolate: energy, momentum, FFT.
2. export\_data()

Saves processed data in a structured format (e.g., CSV or JSON) for further use.

3. Visualization

Classes

1. Visualizer

A class that reads processed data and provides plotting capabilities.

Functions

Any visualisation we’d like to create. For example:

1. plot\_density\_profile()

Plots the density profile over time or at a specific time step.

1. plot\_field\_profile()

Plots the electric field distribution at a specific time step.

1. plot\_energy\_vs\_time()

Plots kinetic and potential energy as a function of time to check energy conservation.

1. animate\_particles()

Generates an animation showing particle positions over time.

---

Main script

- Import the `simulation`, `analysis`, and `visualization` modules.

- Create an instance of `PICSimulation` specifying the initial and parameters, aswell as the required data. call `run\_simulation()` to generate data.

- Pass the output data to `DataProcessor` for analysis.

- Use `Visualizer` to visualize and animate the results.