FYS-2009 Introduction to Plasma Physics
Mandatory Assignment 2

October 17, 2022

Due date: 28.10.2022

In this second mandatory assignment, you will solve Gauss' law given a collection of particles and connect this electrostatic field solver to the particle mover, giving a full, 1D electrostatic particle-in-cell code.

The assignment consists of three exercises, with a total of 10 parts. Each part is weighted equally. You need 50% to pass.

You may use any programming language, but Python or Matlab is preferred. Deliver your code along with the solution to the exercises.

You should provide clearly labelled plots for all exercises, with axes or labels indicating units as well as variables. One way to achieve this is to deliver the mandatory as a Jupyter notebook producing the plots. In Jupyter, you can provide markdown cells in between the code, including LaTeX-style equations. It may simplify your notebook if you provide a '.py'-file with the methods which you call from the notebook.

Exercise 1: Solving the Poisson equation

1. In this exercise, you will devise a numerical method to find the electric field E for given electron and ion charge densities ρ_e and ρ_i by solving the 1D Poisson equation

$$\varepsilon_0 \frac{\partial E}{\partial x} = \rho_i - \rho_e$$

on a periodic grid, as discussed in the numerical exercise 2 in the problem set 2.

You will solve this problem using a *finite discrete Fourier transform*, which is analogous to the Fourier transform used to analyse wave motion, only for functions known on finite and discrete set of points. The theory and detailed implementation of this method is covered in the course FYS-2006.

The central idea is that for the periodic fields ρ_j and E_j with $\rho_0 = \rho_J$ and $E_0 = E_J$ we may write the fields as a sum of plane waves, each with wavelength equal to an integer sub-multiple of the length L:

$$\rho_{s,j} = \frac{1}{J} \sum_{m=0}^{J-1} \hat{\rho}_{s,m} \exp(\mathrm{i}2\pi mj/J),$$

$$E_j = \frac{1}{J} \sum_{m=0}^{J-1} \hat{E}_m \exp(\mathrm{i}2\pi mj/J),$$

where the *Fourier coefficients* are

$$\hat{\rho}_{s,m} = \sum_{j=0}^{J-1} \rho_{s,j} \exp(-\mathrm{i}2\pi mj/J),$$

$$\hat{E}_m = \sum_{j=0}^{J-1} E_j \exp(-\mathrm{i}2\pi mj/J),$$

and the index s stands for either e or i.

In particular, note that the $m = 0$ Fourier coefficient $\hat{\rho}_{s,0} = \sum_{j=0}^{J-1} \rho_j = q_s n_0 = q_s N_s/L$, where n_0 is the particle density and $N_e = N_i = N$ is the total number of particles of each species. Also, since there is no large-scale electric field, $\hat{E}_0 = 0$.

(a) To get the other Fourier coefficients ($m \neq 0$), we relate the continuous variable x to the discrete indices j using the identities $x = j\Delta x$ and $L = J\Delta x$, and substituting $\rho_{s,j}$ and E_j for $\rho_s(x)$ and $E(x)$:

$$\rho_s(x) = \frac{1}{J} \sum_{m=0}^{J-1} \hat{\rho}_{s,m} \exp(\mathrm{i}2\pi mx/L),$$

$$E(x) = \frac{1}{J} \sum_{m=0}^{J-1} \hat{E}_m \exp(\mathrm{i}2\pi mx/L).$$

This allows to perform the derivative $\dfrac{\partial E}{\partial x}$ by considering each term with a Fourier coefficient in the sum over m. Inserting these expressions into Poisson's equation and collecting terms with the same wavenumber $2\pi m/L$, show that the Fourier coefficients are given by

$$\hat{E}_m = -\mathrm{i} \frac{L}{2\pi\varepsilon_0 m} \left(\hat{\rho}_{i,m} - \hat{\rho}_{e,m}\right).$$

(b) Solve the Poisson equation using the discrete Fourier transform for arbitrary ρ_e and ρ_i. You may use the following functions from the np.fft module:

```
# get the Fourier coefficients a_hat given the array a
a_hat = numpy.fft.rfft(a)

# get the array b given the Fourier coefficients b_hat
b = numpy.fft.irfft(b_hat)

# get the frequencies m/(delta_x J)
# in the same order as the Fourier coefficients a_hat
freq = numpy.fft.rfftfreq(J, delta_x)

# to write the complex number z=a + i b, use
z = a+1.j*b
```

The name 'rfft' indicates that this is a so-called fast Fourier transform for real-valued signals.

(c) Test your solution by setting $E(x) = \sin(2\pi x/L) + \sin(6\pi x/L)$, finding $\rho_i - \rho_e$ and solving Poisson's equation numerically. Start by setting $J = 2^n$, $n = 8$. Your numerical solution should be very close to the original $E(x)$. How small can you make n without distorting the result?

(d) Test your solution by setting $E(x) = (x - L/2)\exp(-\sigma(x - L/2)^2)$ finding $\rho_i - \rho_e$ and solving Poisson's equation numerically. Ensure your σ is so large that $E(0) = E(L)$ is very close to zero. Your numerical solution should be very close to the original $E(x)$. What happens if you set σ too small or too large?

Exercise 2: Particle weighting to the grid

2. In this exercise, you will generate a charge density on a grid ρ_j from a collection of charged particles.

Assume we have N charged particles with charge q at locations x_n, where $n = 1, \ldots, N$. We seek to estimate the charge density ρ_j on a periodic grid with J grid points $j\Delta x$, where $j = 0, \ldots, J-1$. Initialise to zero an array of length J to represent the discrete charge density field ρ_j, i.e. such that for all $j = 0, \ldots, J-1$ set $\rho_j = 0$.

Assuming that the position x_n of the particle n is such that $j \leqslant x_n/\Delta x < j+1$, update the charge density at the two nearest grid points using the following linear interpolation

$$\rho_j \to \rho_j + \frac{q}{\Delta x}\frac{(j+1)\Delta x - x_n}{\Delta x}$$
$$\rho_{j+1} \to \rho_{j+1} + \frac{q}{\Delta x}\frac{x_n - j\Delta x}{\Delta x}$$

Keep in mind that the grid is periodic when assigning the charge of the particle to the charge density at the grid points: if $j = J-1$ then $j+1 = 0$. This special case must be accounted for.

We use this particular weighting to ensure that particles do not generate any forces on themselves purely due to the weighting of the particles and the interpolation of the electric field. This weighting is in agreement with the electric field interpolation in numerical exercise 2, problem set 2.

(a) Implement the charge deposition onto the density grid algorithm.

(b) Assign particles uniformly between $x = 0$ and $x = L$ using the numpy uniform generator:

```
N = 10 # number of particles
L = 10 # length of domain in units of Debye length

seed = 1234 # seed for the random number generator
rng = np.random.default_rng(seed=seed)
particle_positions = rng.uniform(0, L, size=N)
```

For comparison, create an array of same size with the theoretical charge density value `rho_theoretical` given by the constant $\rho_{\text{theo}} = qN/L$.

You should check that the normalisation is correct: if ρ_j are the elements of the charge density array, then we approximate

$$qN = \int_0^L \rho(x)\mathrm{d}x \approx \sum_{j=0}^{J-1} \rho_j \Delta x,$$

such that `numpy.sum(rho)` should return the same as `q*N/delta_x`. The sum of the theoretical uniform charge distribution should give the same result.

Plot the computed charge density distribution against the theoretical one. How many particles do you need (for fixed L and Δx) in order to have agreement within 10%? To check this you can for example check that the command

```
print(max(numpy.abs(rho-rho_theoretical)/rho_theoretical))
```

returns a number smaller than 0.1.

Exercise 3: Connecting the particle mover and the electric field solver

Note: This exercise is large and it may be hard to get everything working right. If you have managed both above exercises, you are above 50%. Make sure you are keeping up with the rest of the course as well.

Two example scripts, methods.py and solver.py have been provided on Canvas. They contain function definitions and comments, but no code. You may use these as a starting point to fill in the blanks of your own project or as a framework to work from.

3. You should by now have implemented a method to solve particle motion given an electric field on a grid as well as a method to get the electric field on the grid from the particles. In this exercise, you will connect your particle mover with the electric field solver and test it for consistency.

Use N electrons moving freely on a one-dimensional periodic domain of length L. The domain is discretized into a grid with J grid points separated by Δx, so $x = j\Delta x$, where $j = 0, \ldots, J-1$. Assume you know the ion charge density $\rho_i(x)$. Let the simulation run for a total time T with time step Δt in the particle mover.

A good starting point for the discretization is the choice $L = 10$, $\Delta x = 1/10$, $T = 10$ and $\Delta t = 1/10$ where L and Δx are in units of Debye length and T and Δt are in units of the reciprocal of the electron plasma frequency.

In order to get the electric force on each particle due to all other particles, follow these steps. Here, the electric field and the charge densities are found on the grid.

1. Calculate the electron charge density as in Numerical Exercise 5
2. Add the ion charge density to get the total charge density.
3. Calculate the electric field as in Numerical exercise 4.
4. Find the electric force on each particle using the interpolation scheme in Numerical exercise 2.

(a) Connect your code:

1. Initialise the particles position and velocity at time $t = 0$.
2. *If you implemented the Boris or leap frog algorithm* calculate the electric force on the particles and find the velocities at time $t = -1/2$.
3. For each time step, find the electric force on each particle and run the particle mover.
4. Save the particles new position and velocity.

(b) Initialise a single particle with zero velocity in the simulation domain. Check that the velocity does not change appreciably over time.

(c) Initialise a collection of N stationary and equidistant particles in the domain (remember that the domain is periodic, so take care that the distance between the first and last particles is the same as the rest). Run the simulation and check that the particles remain stationary. Try several different particle numbers, both odd and even numbers.

(d) Initialise a collection of N equidistant particles in the domain. Let the velocity distribution of the particles be Maxwellian. Run the simulation, and check that the particles velocity distribution remains the same. You might need a large number of particles to get good agreement.