Portfolio Assignment 3: Using python to model plasma waves

David Dickinson d.dickinson@york.ac.uk

Semester 1

1 Guidance

You should submit all of your code, figures and any working or written answers (this can be a scan of hand written work and does not need to be typed up, but should be legible).

It should be possible for the marker to run your code by typing `python <your_file>` (where `<your_file>` is the submitted code file name) and this should produce any relevant plots.

Some sample example results are shown at the end in order to allow you to check your implementation. These are intentionally not presented perfectly and you should make sure that any figures you produce are appropriately labelled and clear.

The code you submit should try to follow best practice including, but not limited to, clear structure, sensible variable names, avoiding duplication, clear structure and appropriate code comments.

Should you make a mistake in, or be unable to write, code that is used in a later part of the assignment the error will be carried forward as appropriate so that you are not penalised multiple times for the same mistake.

Should your results not agree with the sample results then you may use the sample results to inform your response to any interpretation or discussion questions if you wish. You should make it clear which set of results you are referring to.

2 Overview

In this assignment you are going to use Python and the SciPy library to simulate waves in a one dimensional, periodic plasma. Here we will be treating the plasma species as fluids. The relevant fluid equations are

$$\frac{\partial n_s}{\partial t} + \frac{\partial \left(U_s n_s\right)}{\partial x} = 0$$

$$m_s n_s \frac{\partial U_s}{\partial t} = q_s n_s E_x - \frac{\partial P_s}{\partial x}$$

where n_s is the density, U_s is the fluid flow, q_s is the charge, m_s is the mass, E_x is the electric field in the x direction, $P_s = n_s T_s$ is the pressure and the s subscript refers to the species that we are considering. Here we have assumed that there is no *net* flow and that there is no magnetic field in order to simplify the full fluid equations slightly. To close the system we need a way to calculate the electric field, E_x, and the temperature, T_s.

For the temperature we will assume that this is a specified constant, i.e. that it doesn't vary in space or time. Under this treatment we find

$$\frac{\partial P_s}{\partial x} = T_s \frac{\partial n_s}{\partial x}$$

For E_x we will use Poisson's equation

$$\frac{\partial E_x}{\partial x} = \frac{\rho}{\epsilon_0}$$

where ρ is the total charge density and ϵ_0 is the permittivity of free space.

Taken together, these equations form a closed system describing the behaviour of the density and flow of multiple plasma species interacting through a self-consistent electric field.

Here we will assume a Hydrogen plasma consisting of protons and electrons so that we have $\rho = q_p \left(n_p - n_e\right)$ where q_p is the proton charge, n_p is the proton density and n_e is the electron density. We will also assume that the protons are static as $m_p \gg m_e$, so that n_p is constant in time and $U_p = 0$ for all times.

We will choose to normalise length scales to the reference Debye length, $\hat{x} = x/\lambda_D$, with

$$\lambda_D = \sqrt{\frac{\epsilon_0 T_r}{q_r n_r}}$$

and temporal scales to the reference plasma frequency, $\hat{t} = t\omega_p$, with

$$\omega_p = \sqrt{\frac{n_r q_r^2}{m_r \epsilon_0}}$$

where the r subscript stands for reference (and we will later choose either proton or electron properties for each of these). The electric field is normalised as $\hat{E}_x = E_x \lambda_D / T_r$, the electron flow is normalised as $\hat{U}_e = U_e / (\lambda_D \omega_p)$, densities are normalised to the reference density as $\hat{n}_s = n_s / n_r$, charge to the reference charge as $\hat{q}_s = q_s / q_r$, mass to the reference mass as $\hat{m}_s = m_s / m_r$ and temperature to the reference temperature $\hat{T}_s = T_s / T_r$. With these normalisations we end up with the following normalised system

$$\frac{\partial \hat{n}_e}{\partial \hat{t}} + \hat{U}_e \frac{\partial \hat{n}_e}{\partial \hat{x}} + \hat{n}_e \frac{\partial \hat{U}_e}{\partial \hat{x}} = 0$$

$$\frac{\partial \hat{U}_e}{\partial \hat{t}} = \frac{\hat{q}_e}{\hat{m}_e} \hat{E}_x - \frac{\hat{T}_e}{\hat{m}_e \hat{n}_e} \frac{\partial \hat{n}_e}{\partial \hat{x}}$$

$$\frac{\partial \hat{E}_x}{\partial \hat{x}} = \hat{q}_p \left(\hat{n}_p - \hat{n}_e \right) = r\hat{h}o$$

Finally we will pick our reference values as $n_r = n_p$, $q_r = q_p$, $T_r = T_p$ and $m_r = m_e$ so that we have $\hat{n}_p = 1$, $\hat{q}_p = 1$, $\hat{q}_e = -1$, $\hat{T}_e = T_e / T_i$ and $\hat{m}_e = 1$.

3 Solving for the electric field

Task [2 marks]: By introducing a Fourier basis, $\alpha_j = \exp\left[i\hat{k}_j \hat{x}_j \right]$, to the normalised Poisson equation show that

$$\bar{E}_j = \frac{\bar{\rho}_j}{ik_j}$$

This is the updated Poisson

or

$$\bar{E}_j = \frac{1 - \bar{n}_{e,j}}{ik_j}$$

where k_j is the j^{th} wavenumber and variables with bars are the Fourier coefficients of the relevant field, e.g. $\hat{E}_x = \sum_j \bar{E}_j \alpha_j$. You should show all your working and clearly state any assumptions / properties that are used.

Task [3 marks]: Write a python function that uses a Fourier spectral approach to solve Poisson's equation to return the normalised electric field, $\hat{E}_x(\hat{x})$, when given the normalised electron density, $\hat{n}_e(\hat{x})$, and $\hat{x}$. Your function should use `fft`, `ifft` and `fftfreq` from `scipy.fft` and should return real numbers (rather than complex). You may assume that the average electric field is zero. What does this

Hint: You will need to take care of how you handle the zero-wavenumber part when in Fourier space. Here you may force the electric field coefficent for the zeroth wavenumber to zero, i.e. $\bar{E}_{j=0} = 0$. You might want to re-visit Lecture 7, Practical 3 and the labs.

Task [2 mark]: Suppose we have $\hat{n}_e = 1 + \sin 2\pi\hat{x}$ where $\hat{x} =$ `numpy.linspace(0, 1, 101, endpoint = False)`. Plot this $\hat{n}_e$, the normalised electric field that your function returns when passed this $\hat{n}_e$. Overplot $1 - d\hat{E}_x/d\hat{x}$ with a dashed line. You may use the code below to calculate $d\hat{E}_x/d\hat{x}$ given $\hat{E}_x$ and $\hat{x}$.

```
from scipy.fft import fftfreq, ifft, fft

def fourier_derivative(field, xx):
    """This function calculates d field / dx using a Fourier approach"""
    wavenumbers = fftfreq(len(xx), xx[1] - xx[0])
    return ifft(complex(0, 1) * wavenumbers * fft(field)).real
```

4 Finding the time derivatives

Task [6 marks]: Write a python function, suitable for use with solve_ivp, that calculates and returns $\frac{\partial \hat{n}_e}{\partial \hat{t}}$ and $\frac{\partial \hat{U}_e}{\partial \hat{t}}$ given the current time, state (i.e. $\hat{n}_e$ and $\hat{U}_e$), $\hat{x}$ grid and normalised electron temperature, $\hat{T}_e$. Your function should take these values as arguments but may make use of $\hat{q}_e = -1$ and $\hat{m}_e = 1$.

Hint: Here the state vector will consist of the value of $\hat{n}_e$ at all $\hat{x}$ grid points, followed by the value of $\hat{U}_e$ at all $\hat{x}$ grid points and you should therefore return a vector containing $\frac{\partial \hat{n}_e}{\partial \hat{t}}$ at all grid points followed by $\frac{\partial \hat{U}_e}{\partial \hat{t}}$ at all grid points. You may also find the earlier code snippet helpful.

5 Application

Task [4 marks]: Using solve_ivp and the functions you have written so far solve the system of equations with $\hat{x} =$ `numpy.linspace(0, 1, 401, endpoint = False)` from $\hat{t} = 0$ to $\hat{t} = 2\pi$, recording the solution at 251 uniformly spaced times over this range. The initial $\hat{n}_e$ and $\hat{U}_e$ are determined by the below code snippet and you should use `kx = 4` and `amp = 0.01`. You should use $\hat{T}_e = 0$.

```
def initial_ne(xx, kx, amp):
    from numpy import cos, pi
    nx = len(xx)
    ne0 = 1 + amp * cos(2 * pi * xx * kx)
    ne0[:nx//4] = 1.0
    ne0[3*nx//4:] = 1.0
    return ne0

def initial_Ue(xx):
    return xx * 0.0
```

Task [3 marks]: Using the `pcolormesh` method of `matplotlib.pyplot` produce a correctly labelled colour map of the resulting $\hat{n}_e\left(\hat{x}, \hat{t}\right)$ with $\hat{x}$ on the y-axis and $\hat{t}$ on the x-axis. Comment on the behaviour you observe.

6 Extension

Task [2 marks]: Solve the system under the same conditions as in the previous section but with $\hat{T}_e = 1.0$ and produce the corresponding colour map of $\hat{n}_e\left(\hat{x}, \hat{t}\right)$.

Task [2 marks]: How does the behaviour compare to the case with $\hat{T}_e = 0$? Discuss the physics behind the main difference observed.

Task [1 mark]: How does changing the solve_ivp integration method to `Radau` change the solution?

Hint: You may wish to focus on the behaviour of the solution at late times, e.g. try plotting $\hat{n}_e\left(\hat{x}, \hat{t} \sim 5\right)$.

7 Example plots

To help you check your implementation the following figure shows an example result. It should be noted that whilst the figures shows "correct" data the presentation can be improved for full marks.

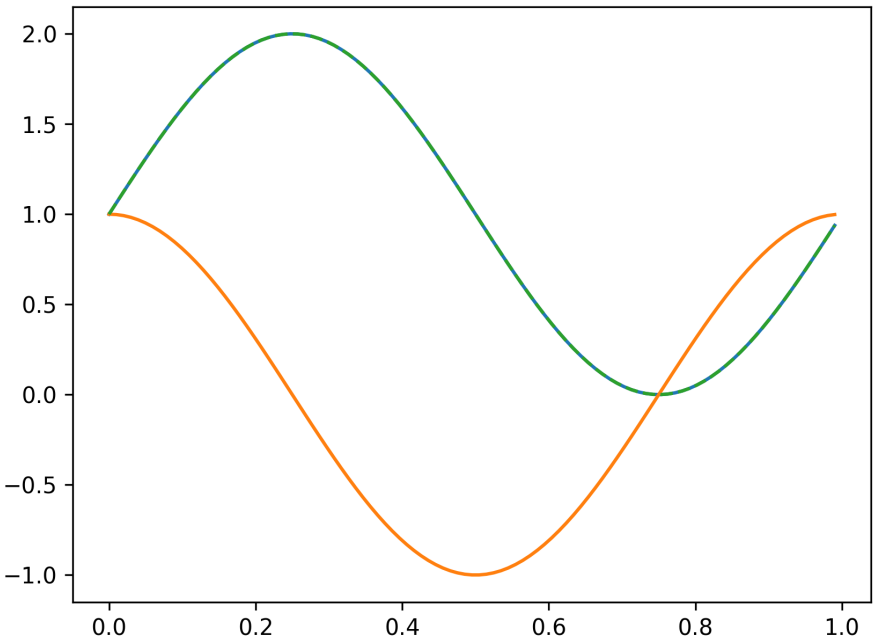

Figure 1: Plot of initial $\hat{n}_e$ and the corresponding $\hat{E}_x$ and $1 - d\hat{E}_x/d\hat{x}$.

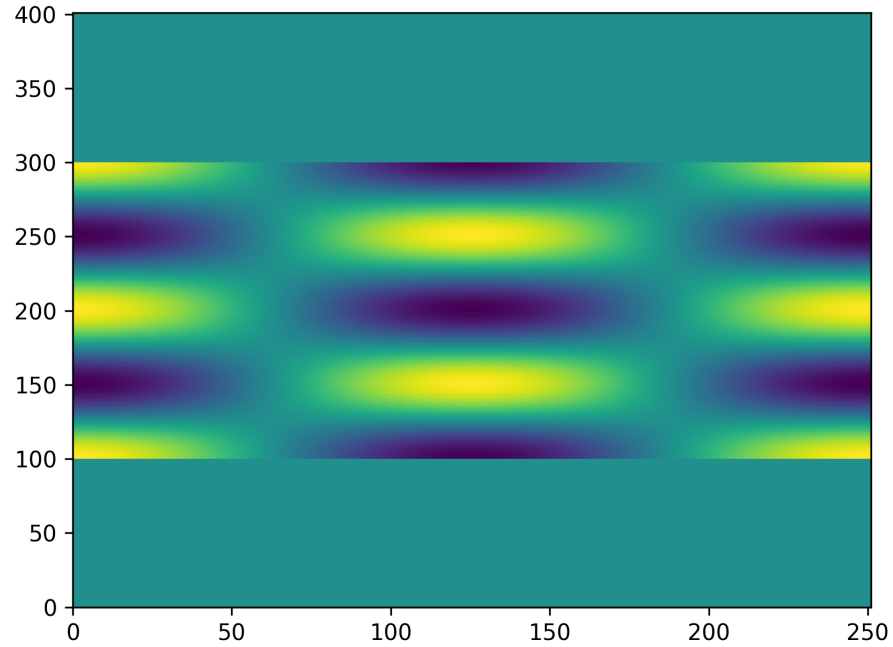

Figure 2: Colour map of solution with $\hat{T}_e = 0.0$.

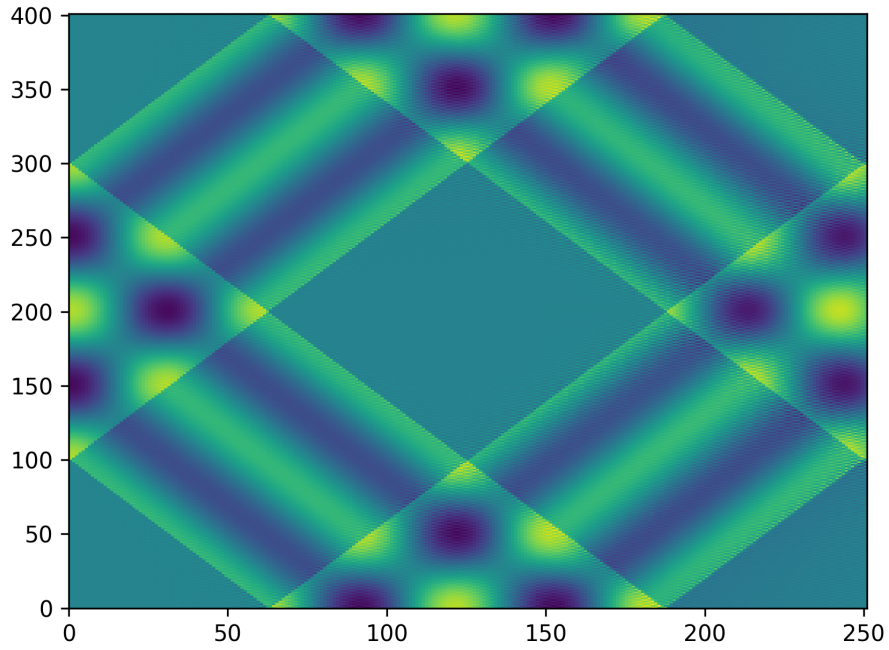

Figure 3: Colour map of solution with $\hat{T}_e = 1.0$.