PHY407-Lab08

Due Nov 8 2022

Computational Background

Boundary value problem methods The first class of PDEs discussed in Chapter 9 are solutions of elliptic partial differential equations of which the Poisson equation (9.10) is a classic example. The textbook covers the Jacob method, which is a relaxational method for solving Laplace's or Poisson's equation, and then speedups to this method using overrelaxation and Gauss-Seidel replacement. For this lab we will focus on Gauss-Seidel, with or without overrelaxation. To obtain the solution of a field $\phi(x, y)$ that satisfies Laplace's equation

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0, \tag{1}$$

subject to boundary conditions (see Physics Background), overrelaxation is written [see (9.17)]

$$\phi(x, y) \leftarrow \frac{1+\omega}{4}\left[\phi(x+a, y) + \phi(x-a, y) + \phi(x, y+a) + \phi(x, y-a)\right] - \omega\phi(x, y), \tag{2}$$

where the left arrow indicates replacement of the left hand side with the right, and ω is a relaxation parameter. For these methods, you can either pick a fixed number of iterations or can choose a level of convergence for the solution. In this lab, we will be using the latter approach.

Flux-conservative PDE problems A large class of time-dependent PDEs can be written in flux-conservative form. In one space dimension, this is given as

$$\frac{\partial\vec{u}}{\partial t} = -\frac{\partial\vec{F}(\vec{u})}{\partial x}, \tag{3}$$

where, in general, $\vec{u}$ and $\vec{F}$ are vectors consisting of a set of multiple fields. The variable we are solving for is $\vec{u}$, while $\vec{F}$ is a given function that can depend on $\vec{u}$ and on spatial derivatives of $\vec{u}$.

The simplest scheme to discretize this system is the forward-time centred-space (FTCS) scheme, in which one uses a forward difference for the time derivative, and a centred difference for the spatial derivative. If, for example, the problem is one dimensional and $\vec{u}$ and $\vec{F}$ have just one component field, then

$$\left.\frac{\partial u}{\partial t}\right|_j^n \approx \frac{1}{\Delta t}\left(u_j^{n+1} - u_j^n\right), \qquad \left.\frac{\partial F}{\partial x}\right|_j^n \approx \frac{1}{2\Delta x}\left(F_{j+1}^n - F_{j-1}^n\right), \tag{4}$$

where superscripts like n refer to the time step index and subscripts like j refer to the spatial index (see figure 1). Substituting these approximations into eqn. (3) and rearranging, it is easy to show that

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{2\Delta x}\left(F_{j+1}^n - F_{j-1}^n\right). \tag{5}$$

Note, as you will see in this lab, this very simple discretization is unconditionally unstable for typical wave equations.

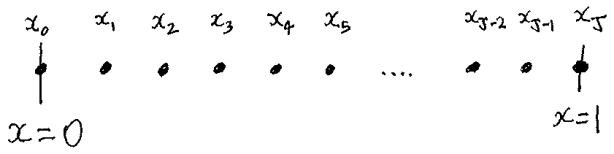

Figure 1: Notations for spatial grid.

Optional: Stream Plots This type of plot is used to illustrate a vector field $\vec{F}$. At each location, $\vec{F}(\vec{x})$ (e.g. electric field) is tangential to the local streamline (e.g. electric field line). Matplotlib's `streamplot` function (reference page `https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.streamplot.html`) can colour-code the lines with the scalar of your choice. Adapted from this page is the example below, for electric field lines due to the presence of two point charges $\pm 4\pi\epsilon_0$, separated by a distance $d = 2$.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 100)  # does not include zero
y = np.linspace(-1, 1, 50)
X, Y = np.meshgrid(x, y)
R1 = ((X-1)**2 + Y**2)**.5  # 1st charge located at x=+1, y=0
R2 = ((X+1)**2 + Y**2)**.5  # 2nd charge located at x=-1, y=0

V = 1./R1 - 1./R2  # two equal-and-opposite charges
Ey, Ex = np.gradient(-V, y, x)  # careful about order

fig = plt.figure(figsize=(6, 3))
strm = plt.streamplot(X, Y, Ex, Ey, color=V, linewidth=2, cmap='autumn')
cbar = fig.colorbar(strm.lines)
cbar.set_label('Potential $V$')
plt.title('Electric field lines')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.axis('equal')
plt.tight_layout()
plt.show()
```

Optional: Animations in Matplotlib Here is a sample code that will animate the curve $\sin(x - t)$.

```
from numpy import arange, pi, sin
from pylab import clf, plot, xlim, ylim, show, pause, draw
t = arange(0, 4*pi, pi/100)  # t coordinate
x = arange(0, 4*pi, pi/100)  # x coordinate
for tval in t:
    clf()  # clear the plot
    plot(x, sin(x-tval))  # plot the current sin curve
    xlim([0, 4*pi])  # set the x boundaries constant
    ylim([-1, 1])  # and the y boundaries
    draw()
    pause(0.01)    #pause to allow a smooth animation
```

The script simply clears each frame and then re-plots it, with a little pause to make it run smoothly. This is not an ideal method but gets the job done. A "cleaner" method would be to follow the guidelines of `https://matplotlib.org/api/animation_api.html`, but it requires a bit more work.

Physics background

Electrostatics. In empty space, the electrostatic potential $V(x, y)$ satisfies Laplace's equation $\nabla^2 V = 0$. The electric field is $\vec{E} = -\nabla V$ and the electrostatic force felt by a particle with charge q is given by $q\mathbf{E}$. A suitable equation of motion for the particle is $\ddot{\mathbf{x}} = q\mathbf{E}/m$, where m is the mass of the charged particle. We can use this equation of motion a bit more broadly, if the electric fields are slowly varying in time and the particle isn't moving too fast.

In this lab, you will be calculating an electric field with strong spatial variations and allowing it to oscillate in time. This kind of configuration can be used to confine charged particles, based on the idea that charged particles in an oscillating electric field tend to be attracted to regions where the electric field strength is weak.

Shallow Water Equations These are a simplification of the full Navier-Stokes equations, which describe the motion of fluid flow (see § 3.1 of *Atmospheric and Oceanic Fluid Dynamics* by Geoffrey K. Vallis, available on the UofT library website at `http://go.utlib.ca/cat/11621853`, for a basic introduction; we will not consider the Earth's rotation and you can therefore ignore f, the Coriolis parameter). They are appropriate for describing the motion in shallow layers of fluid under the influence of gravity. By "shallow" we mean that the depth of the fluid is small compared to the relevant horizontal length scales of the fluid. Thus, in some cases, the ocean can be considered a shallow fluid, as long as we are interested in phenomena whose horizontal length scales are much larger than the depth of the ocean (on average, around 4 km). In particular, it is common to use the shallow water equations for modelling the tides, and for simulating tsunamis in the ocean.

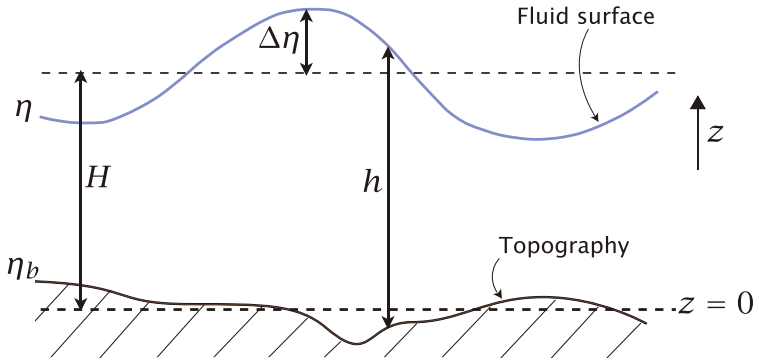

Fig. 3.1

Figure 2: Variable definitions for the shallow water system. Figure shows a cross-section on the (x, z) plane. The upper dashed line represents the equilibrium surface, the lower dashed line the $z = 0$ origin, H is the average (in time and space) water column height, the top solid line represents the free surface of the fluid at $z = \eta(x, t)$, and the bottom solid line represents the fixed bottom topography at $z = \eta_b(x)$. From Vallis (2017, fig. 3.1).

For simplicity, we will restrict ourselves to the 1D version of the shallow-water equations:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} \tag{6}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial (uh)}{\partial x} = 0$$

where u is the fluid velocity in the x direction, $h = \eta - \eta_b$ is the water column height, η the altitude of the free surface, η_b is the altitude of the bottom topography (which is typically a known function), and g is the

acceleration due to gravity. (See also fig. 2 for notations.) Note that all the variables are functions of x and t, but not z (i.e. every column of fluid moves together), as a consequence of the assumptions that lead to the shallow water equations.

Also note, these equations support wave solutions. These waves are non-dispersive, their phase and group speeds over flat bottom being equal to $\sqrt{gH}$, where H is the water column height at rest.

Questions

1. Electrostatics and Laplace's equation [30%]

(a) Consider the simple model of an electronic capacitor sketched in fig. 3, consisting of two flat metal plates enclosed in a square metal box.

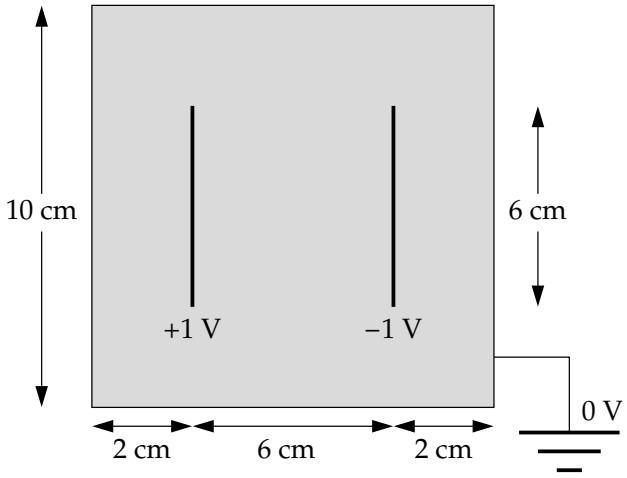

Figure 3: Capacitor for Q1 (from Newman, p. 417).

For simplicity let us model the system in 2D. Using the Gauss-Seidel method without over-relaxation, write a program to calculate the electrostatic potential at each grid point in the box, with the following conditions:

- Grid of 100×100 points
- Walls of the box at 0 voltage
- The two plates (of negligible thickness) at voltages ± 1 V
- Precision (at each grid point) 10^{-6} V

Use the results to make a contour plot of the potential (filled or not, it's your choice).

Submit the code and plot.

(b) Repeat, with over-relaxation. Try with $\omega = 0.1$ and $\omega = 0.5$. What do you notice?

Submit the code, plot, and written answer.

(c) **Optional** Make a stream plot of the electric field lines, colour-coded by the value of the electric potential.

Simulating the shallow water system [70%]

(a) Show that the 1D shallow water equations (6) can be rewritten in the flux-conservative form of eqn. 3, with $\vec{u} = (u, \eta)$ and

$$\vec{F}(u, \eta) = \left[\frac{1}{2}u^2 + g\eta, (\eta - \eta_b)u \right]. \tag{7}$$

Then use the FTCS scheme (eqn. 5) to discretize the 1D shallow water equations. You should have two equations of the form $u_j^{n+1} = \ldots$ and $\eta_j^{n+1} = \ldots$ where the right-hand sides depend on u and η at the timestep n. Assume $\eta_b(x)$ is a known function (i.e., $\eta_b(x_j) = \eta_{b,j}$ is given).

Submit written answer.

(b) Implement the 1D shallow water system with the FTCS scheme in Python. Use the following set up (perhaps simulating waves in a very shallow one dimensional flat bathtub!):

- Domain $x = [0, L]$, with $L = 1$ m
- Grid spacing as shown in fig.1, with $J = 50$, so $\Delta x = 0.02\,\text{m}$.
- Take $g = 9.81\ \text{m}\,\text{s}^{-2}$
- Flat bottom topography described by $\eta_b = 0$, $H = 0.01$ m
- Boundary conditions $u(0, t) = u(L, t) = 0$ (i.e. rigid walls)
- Timestep $\Delta t = 0.01\,\text{s}$.
- Initial conditions:

$$u(x, 0) = 0, \qquad \eta(x, 0) = H + Ae^{-(x-\mu)^2/\sigma^2} - \left\langle Ae^{-(x-\mu)^2/\sigma^2} \right\rangle, \tag{8}$$

with $A = 0.002\,\text{m}$, $\mu = 0.5$ m, $\sigma = 0.05$ m, and where $\langle\rangle$ is the x-average operator, ensuring that H retains its definition of the free surface altitude at rest. This represents a small Gaussian peak at the centre of the domain.

Make plots of η vs x at $t = 0$ s, $t = 1$ s and $t = 4$ s.

Hand in the code and plots.

Hints:

- You will have to treat the grid points at the boundaries separately, because you cannot use a centred difference approximation here, since you do not have values for u or η at $j = -1$ or $j = J+1$. To get around this issue, use forward and backward differences at each of the boundary points to approximate the spatial derivative of the fluxes:

$$\left.\frac{\partial F}{\partial x}\right|_0^n \approx \frac{1}{\Delta x}\left(F_1^n - F_0^n\right), \qquad \left.\frac{\partial F}{\partial x}\right|_J^n \approx \frac{1}{\Delta x}\left(F_J^n - F_{J-1}^n\right). \tag{9}$$

- You should find that your solution eventually diverges. Do not be alarmed, this doesn't necessarily mean you have an error in your code!
- In the finite difference scheme, you need to use *only* the previous time step values to update the variable. The easiest way to do this is by defining second arrays for u and η (e.g., called `u_new` and `eta_new`), and then your update lines in the loop can look like:

```
u_new[j] = u[j] + ...  # (all in terms of u and eta)
eta_new[j] = eta[j] + ...  # (all in terms of u and eta)
eta = np.copy(eta_new)
u = np.copy(u_new)
```

It is important to use the `np.copy` command in the last 2 lines, otherwise python just updates `eta` at the same time as `eta_new` in the next iteration. (Technically, this is because Python treats the equality sign as assignment to a pointer when the object on the right-hand side is a list or Numpy array. If you don't know what a pointer is, don't worry about understanding the technicality here.)

(c) Do a von Neumann stability analysis of the FTCS scheme for the 1D shallow water equations (as described in § 9.3.2, pp. 425-430 of the textbook). Note that this type of stability analysis can only be applied to linear equations, so first you need to show that the linearized eqns (6) about $(u, \eta) = (0, H)$ are, with constant topography η_b,

$$\frac{\partial u}{\partial t} = -g\frac{\partial \eta}{\partial x} \qquad (10)$$
$$\frac{\partial \eta}{\partial t} = -H\frac{\partial u}{\partial x}$$

Then you can closely follow the development in the textbook for the stability analysis of the wave equation (pp. 429–430) to show that the magnitude of the eigenvalues λ is, for both eigenvalues,

$$|\lambda| = \sqrt{1 + \left(\frac{\Delta t}{\Delta x}\right)^2 gH \sin^2(k\Delta x)}. \qquad (11)$$

Do you expect the FTCS scheme to be stable? Why or why not?

Submit your derivation.

(d) **Optional** Create an animation of η vs x, out of lineplots generated at several t values, using your code from Part (a).