Computational Lab 9 : Solving PDEs, part II

Due November 15th, 2019 (@ 17:00)

- Read this document and do its suggested readings to help with the pre-labs and labs.
- This lab's topics revolve around computing solutions to PDEs using advanced methods, particularly spectral decomposition and the Crank–Nicolson method.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Whether or not you are asked to hand in pseudocode, you **need** to strategize and pseudocode **before** you start coding. Writing code should be your last step, not your first step.
- Test your code as you go, **not** when it is finished. The easiest way to test code is with `print('')` statements. Print out values that you set or calculate to make sure they are what you think they are.
- Practice modularity. It is the concept of breaking up your code into pieces that as independent as possible form each other. That way, if anything goes wrong, you can test each piece independently.

Physics Background

Shallow Water Equations (for Q1) : The shallow water equations are a simplification of the full Navier-Stokes equations, which describe the motion of fluid flow (see Chapters 8 and 9 of *Introduction to Ocean Waves* by Salmon, available at http://www-pord.ucsd.edu/~rsalmon/, for a good, basic introduction). 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 4km). In particular, it is common to use the shallow water equations for modelling the tides, and for simulating tsunamis in the ocean. In this lab, we will attempt to do the latter, albeit in a highly simplified setting.
The shallow water equations are:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -g\frac{\partial \eta}{\partial x} \tag{1a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -g\frac{\partial \eta}{\partial y} \tag{1b}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}(u(\eta + H)) + \frac{\partial}{\partial y}(v(\eta + H)) = 0 \tag{1c}$$

where u and v are the fluid velocities in the x and y directions, η is the displacement of the surface, H is the depth of the bottom topography, which is typically a known function, and g is the acceleration due to gravity. See figure 2. Note that all the variables are functions of x, y and t, but not z (i.e. every column of fluid moves together). This is a consequence of the assumptions that lead to the shallow water equations.
For simplicity, we will restrict ourselves to the 1D version of the equations:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} \tag{2a}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}(u(\eta + H)) = 0. \tag{2b}$$

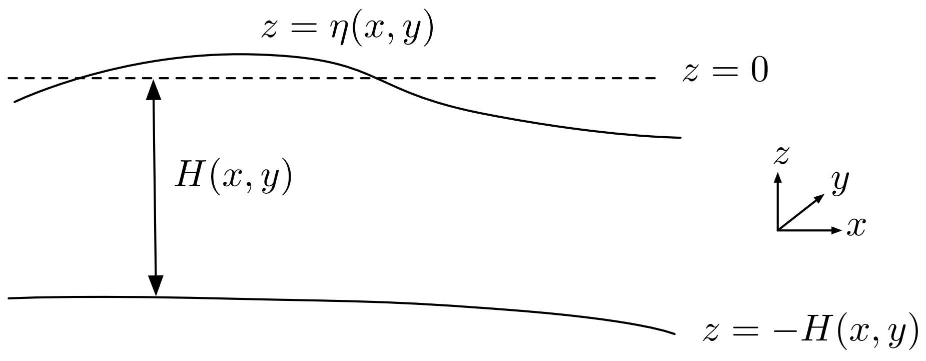

Figure 1: Variable definitions for shallow water system. Figure shows a cross-section on the $x-z$ plane. The dashed line represents the equilibrium surface (where we set $z=0$), the top solid line represents the free surface of the fluid, and the bottom solid line represents the fixed bottom topography. Adapted from *Introduction to Ocean Waves* by Salmon.

Waves on a string (for Q2): The one-dimensional wave equation is, using Newman's notation,

$$\frac{\partial^2\phi}{\partial t^2} = v^2\frac{\partial^2\phi}{\partial x^2}. \tag{3}$$

To fully specify the problem we require the boundary conditions that the string is fixed on each end:

$$\phi(x=0,t) = \phi(x=L,t) = 0, \tag{4}$$

and initial conditions on the displacement and velocity:

$$\phi_0(x) = \phi(x,t=0), \psi_0(x) = \psi(x,t=0) = \frac{\partial\phi(x,t=0)}{\partial t}. \tag{5}$$

Computational Background

Time/spatial PDE problems: We will introduce and implement some techniques for solving time-dependent partial differential equations. For simplicity, we will only consider examples with one space dimension, thus variables will depend on x and t. Concepts include finite differences, von Neumann stability analysis, the forward-time centred-space (FTCS) numerical scheme, and the Lax-Wendroff numerical scheme.
Note that the Lax-Wendroff numerical scheme introduced in this lab is not discussed in the Newman textbook. You can refer to *Numerical Recipes* by Press et al. for more details (in particular, Section 19.1 from the 2nd edition, or Section 20.1 from the 3rd edition). The 2nd edition of this textbook is available in its entirety online for free at http://apps.nrbook.com/fortran/index.html.
A large class of time-dependent PDEs can be written in flux-conservative form. In one space dimension, this is given as

$$\frac{\partial\vec{u}}{\partial t} = -\frac{\partial\vec{F}(\vec{u})}{\partial x}, \tag{6}$$

where,in general, $\vec{u}$ and $\vec{F}$ are vectors consisting of a set of multiple fields. $\vec{u}$ is the variable we are solving for, 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

$$\left.\frac{\partial u}{\partial t}\right|_j^n \approx \frac{1}{\Delta t}\left(u_j^{n+1} - u_j^n\right), \qquad \left.\frac{\partial F}{\partial x}\right|_j^n \approx \frac{1}{2\Delta x}\left(F_{j+1}^n - F_{j-1}^n\right), \tag{7}$$

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 Equation 6 and rearranging, it is easy to show

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{2\Delta x}\left(F_{j+1}^n - F_{j-1}^n\right). \tag{8}$$

However, this very simple discretization is unconditionally unstable for typical wave equations. An alternative scheme used in the lab, the Lax-Wendroff method, is second-order accurate in time and has improved stability.

The Two–Step Lax–Wendroff scheme: The Lax–Wendroff scheme provides improved stability and accuracy over the FTCS method, at a slightly higher computational cost. See *Numerical Recipes* for a more complete discussion. The first step is to calculate the values at the "half-steps" $t_{n+1/2}$ and $x_{j+1/2}$ using the Lax method:

$$u_{j+1/2}^{n+1/2} = \frac{1}{2}\left(u_{j+1}^n + u_j^n\right) - \frac{\Delta t}{2\Delta x}\left(F_{j+1}^n - F_j^n\right). \tag{9}$$

Once you have the values of u and η at the half steps, you can calculate the fluxes $F_{j\pm1/2}^{n+1/2}$ at the half steps. Then, calculate the values at the next full timestep (and at the "normal" grid points) using a simple FTCS scheme:

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(F_{j+1/2}^{n+1/2} - F_{j-1/2}^{n+1/2}\right). \tag{10}$$

Spectral method (for Q2): Section 9.3.4 of the Newman text discusses solving the wave-on-string problem using spectral methods. The solution method is to decompose the waveform into a Fourier series of a finite number of sine waves, each of which satisfies the boundary conditions. An example of the discretized Fourier series is shown in Section 9.3.4 and the coefficients in the series can be obtained using the fast discrete sine transform (DST) that is discussed briefly in Section 7.4.3 and is made available in the package `dcst.py`, which implements the fast cosine and sine transforms. (We will use slightly different notation from the book in Q2 below.)
The spectral method writes the solution in terms of Fourier sine series and uses the computer to calculate the series for a finite number of terms. We need to derive expressions for derivatives in spectral space, as discussed in class, and then find the solution doing an inverse sine transform.
If you understand the principles of this question there is actually very little coding involved. You will see, in fact, that the code you have runs almost instantantly, in contrast to typical FTCS problems where you have to integrate a long time to get a snapshot of the system at the desired time. The spectral method is great as long as the conditions are right to apply it!

Crank-Nicolson method (for Q2): Remember that the FTCS method suffered from stability problems. The Crank-Nicolson (C–N) scheme involves a combined implicit/explicit scheme that is numerically stable. Its used quite often for time-evolution of PDEs. Section 9.3.3 of the Newman text discusses the method in detail as applied to the wave equation. For the wave equation, C–N is stable (i.e. stable for all time step sizes) because the magnitudes of the eigenvalues always equal 1. This means that initial fourier modes neither grow nor decay in the evolution, precisely the behaviour one expects for the solution to the wave equation. For the time-dependent Schrodinger equation, the C-N equation is given at the top of page 440. It results in a set of simultaneous equations, one for each grid point. This means that we can solve the system using our methods for solving linear systems (from Chapter 6). One particular aspect is taking advantage of the banded structure of matrices to solve for a linear set of equations. The module `banded.py`, provided by Newman in his online materials, has the functionality to do this.

Questions

1. **[50%] Simulating the shallow water system:**

   (a) Show that the 1D shallow water Equations (2) can be rewritten in the flux-conservative form of Equation 6, with $\vec{u} = (u, \eta)$ and

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

Now use the Lax–Wendroff scheme to discretize the 1D shallow water equations. You should have equations to integrate in time the velocity field u and the height field η assuming the bottom boundary $H(x_j)$ is given.

SUBMIT YOUR WRITTEN SOLUTIONS

(b) Implement the Lax–Wendroff scheme in Python, and run it using the following setup

- Take your domain as $x = [0, 1]$ (in metres), and use a grid spacing $\Delta x = 0.02\,\text{m}$.
- Take $g = 9.81\text{ms}^{-2}$
- Assume $H(x) = 0.01\,\text{m}$ (flat bottom topography).
- As boundary conditions, you should impose $u(0, t) = u(1, t) = 0$, i.e. rigid walls. Note that 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:

$$\left.\frac{\partial F}{\partial x}\right|_0^n \approx \frac{1}{\Delta x}\left(F_1^n - F_0^n\right), \qquad \left.\frac{\partial F}{\partial x}\right|_J^n \approx \frac{1}{\Delta x}\left(F_J^n - F_{J-1}^n\right). \tag{11}$$

- Use a timestep of $\Delta t = 0.01\,\text{s}$.
- Use the following initial conditions:

$$u(x, 0) = 0, \qquad \eta(x, 0) = Ae^{-(x-\mu)^2/\sigma^2} \tag{12}$$

  with $A = 0.002\,\text{m}$, $\mu = 0.5\,\text{m}$ and $\sigma = 0.05\text{m}$. This represents a small gaussian peak at the centre of the domain.

SUBMIT YOUR CODE AND FIGURES SAVED AT T=0S,T=1S,T=4S

(c) As a final step, you should allow for variable bottom topography. This should be a relatively straightforward change to your code. Instead of having H be a constant, it will be an array defined on your grid. Thus, when you calculate the fluxes F_j^n you need to make sure you are using the value of the topography at the grid point j (in fact, for the Lax-Wendroff scheme you will need H at the half-steps $j + 1/2$, so you should just estimate it at these points as the mean between the two nearest integer points).

You should now be able to simulate a 1D tsunami! Try the following setup. (Note, for this question you can still use $g = 9.81\text{ms}^{-2}$, although the values given for the depth and width of the system do not really represent something like the ocean. However, the example still lets you see the effects of changes in bottom topography.)

- Domain: $x = [0, 1]$. $J = 150$, so $\Delta x = 1/150$.
- Use a bottom topography that has a change in depth, simulating the presence of a continental shelf (see Figure 2):

$$H(x) = 0.001 + 0.1\exp(-7x) \tag{13}$$

- Boundary conditions: $u(0, t) = u(1, t) = 0$, i.e. rigid walls.
- Timestep: $\Delta t = 0.001$.
- Initial conditions (wide gaussian bump at far left boundary, i.e. in deep ocean):

$$u(x, 0) = 0, \qquad \eta(x, 0) = Ae^{-(x-\mu)^2/\sigma^2} \tag{14}$$

  with $A = 0.0005\,\text{m}$, $\mu = 0$ and $\sigma = 0.1\,\text{m}$.

You should see a shallow, wide bump starting at the left of the domain move to the right, towards the shallower water. What happens to the waveform when it reaches the shallower waters (in terms of its shape, height, speed, etc.)?

Make plots of $\eta(x, t)$ (you might find it helpful to also plot the bottom topography on the same figure) at $t = 0$, $t = 1$, $t = 2$, $t = 3$ and $t = 4$.

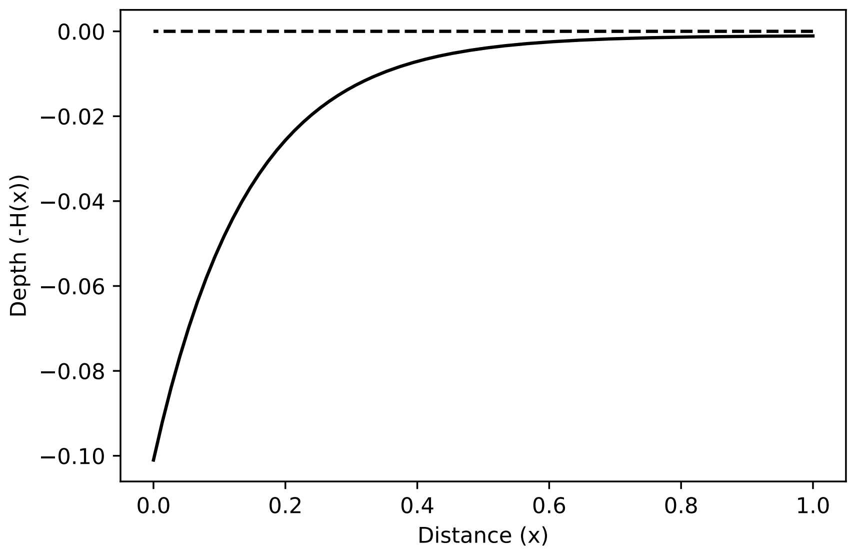

Figure 2: Bottom topography for tsunami simulation.

HAND IN YOUR CODE, PLOTS, EXPLANATORY NOTES.

2. **[50%] Solving the wave equation:** (Based on exercise 9.5 from the textbook) Consider a piano string of length L, initially at rest. At time $t = 0$ the string is struck by the piano hammer a distance d from the end of from the string:

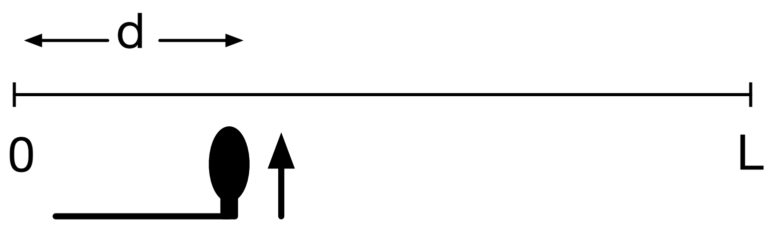

Figure 3: A Piano String and hammer.

The string vibrates as a result of being struck, except at the ends, $x = 0$, and $x = L$, where it is held fixed.

(a) Write a program that uses the FTCS method to solve the complete set of simultaneous first–order equations, (Eq. 9.28 in the textbook), for the case $v = 100\text{ms}^{-1}$, with the initial condition that $\phi(x) = 0$ everywhere but the velocity $\psi(x)$ is nonzero, with profile

$$\psi(x) = C\frac{x(L-x)}{L^2}\exp\left[-\frac{(x-d)^2}{2\sigma^2}\right], \tag{15}$$

where $L = 1m$, $d = 10cm$, $C = 1\text{ms}^{-1}$, and $\sigma = 0.3m$. You will also need to choose a value for the time–step h. A reasonable choice is $h = 10^{-6}s$.

Note: You might be wondering why you're using the FTCS method to solve the wave equation, it is unstable after all. The goal here is to give you the framework using a simple algorithm that you can reuse for the Crank-Nicolson algorithm and spectral method later.

SUBMIT THE CODE

(b) Plot a sequence of images to show the motion of the piano string. A simple method is to plot the string as a continuous line and plot a few lines on the graph. Another approach would be to add a marker to show where you are calculating the solution. Run the simulation and make plots at times $t = [2, 4, 6, 12, 100]$ ms.

The textbook suggests making an animation of the motion using the `visual` package. The `visual` package is not compatible with recent versions of Python, and animations are particularly hard to grade.

SUBMIT PLOTS AT THE PRESCRIBED TIMES. IF YOUR SOLUTION GOES UNSTABLE, INCLUDE A PLOT SHOWING THE INSTABILITY (THESE MIGHT BE THE SAME PLOT).

(c) Repeat part a using the Crank–Nicolson method to solve the equations. You should be able to increase the timestep relative to the FTCS method.

SUBMIT CODE AND PLOTS WITH THE SAME TIMES AS PART B.

(d) Suppose you can expand the initial conditions of the piano string in terms of a Fourier sine series:

$$\phi_0(x) = \sum_{k=1}^{\infty} \tilde{\phi}_{0,k} \sin(k\pi x/L), \ \psi_0(x) = \sum_{k=1}^{\infty} \tilde{\psi}_{0,k} \sin(k\pi x/L).$$

Note that all the terms in this series vanish at $x = 0$ and $x = L$.

Show by substitution that the general solution to (3) -(5) in terms of the Fourier series is

$$\phi(x,t) = \sum_{k=1}^{\infty} \sin(k\pi x/L) \left[\tilde{\phi}_{0,k} \cos(\omega_k t) + \frac{\tilde{\psi}_{0,k}}{\omega_k} \sin(\omega_k t) \right] \tag{16}$$

where you will need to derive the expression for the frequency ω_k, which depends on k.

SUBMIT A CONCISE DERIVATION, EITHER HAND-WRITTEN OR IN TEX.

(e) Using a series solution like this but truncating at a finite N, use the `dst` and `idst` functions in `dcst.py` to calculate and plot the solution $\phi(x,t)$ for the same problem as in part a, at $t = 2, 4, 6, 12, 100$ ms. You can use a very large number of Fourier coefficients in this calculation - since you only have to do one Fourier transform.

SUBMIT CODE AND PLOTS FOR EACH TIME REQUESTED, t.

(f) Compare your spectral solution to the FTCS solution and Crank–Nicolson.

SUBMIT SEVERAL LINES OF TEXT EXPLAINING THE DIFFERENT APPROACHES, AND THE RELATIVE ADVANTAGES AND DISADVANTAGES OF EACH.

Question:	1	2	Total
Points:	50	50	100