Lab assignment #8: Partial Differential equations, Pt. I

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)

Due Friday, November 6th 2020, 5 pm

First, try to sign up for room 1. If it is full, sign up for room 2 (and ask your partner to join you there if they are in room 1).

Room 1 Alex Cabaj (Marker, alex.cabaj@mail.utoronto.ca),
URL: `https://gather.town/2d25ninrKWziTcd4/PHY407AC`
PWD: `phy407-2020-TA-alex`

Room 2 Pascal Hogan-Lamarre (Marker, pascal.hogan.lamarre@mail.utoronto.ca),
URL: `https://gather.town/k9a7e9j0MjrTEoqw/PHY407-PHL`
PWD: `phy407-2020-phl`

General Advice

- **Work with a partner!**
- This lab's topics revolve around computing solutions to PDEs using basic methods.
- Read this document and do its suggested readings to help with the pre-lectures and labs.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Carefully check what you are supposed to hand in for grading in the section "Lab Instructions".
- 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 from each other. That way, if anything goes wrong, you can test each piece independently.
- One way to practice modularity is to define external functions for repetitive tasks. An external function is a piece of code that looks like this:

```
def MyFunc(argument):
    """A header that  explains the function
    INPUT:
    argument [float] is the angle in rad
    OUTPUT:
    res [float] is twice the argument"""
    res = 2.*argument
    return res
```

Place them in a separate file called e.g. `MyFunctions.py`, and call and use them in your answer files with:

```
import MyFunctions as fl2  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl2.MyFunc(ZehValyou)
```

Computational background

Boundary value problem methods (for Q1): The first class of PDEs that Newman discusses in Chapter 9 are solutions of elliptic partial differential equations of which the Poisson equation (9.10) is a classic example. He discusses 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 would like you to 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 Q1a, we will be using the latter approach.

Stream plots show streamlines of a vector field $\vec{F}$. That is, at each location, $\vec{F}(\vec{x})$ is tangential to the local streamline. For example, for an electric field $\vec{E}$, streamlines are electric field lines. Matplotlib's `streamplot` function (https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.streamplot.html) can do such a plot, and colour-code the lines with the scalar of your choice. See example below for electric field lines due to the presence of two point charges $\pm 4\pi\epsilon_0$, separated by a distance $d = 2$. It is adapted from the example at the bottom of the web page above.

```
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()
```

Time/space PDE problems For Q2-Q3, 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, but online resources exist, simply starting with https://en.wikipedia.org/wiki/Lax%E2%80%93Wendroff_method.

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 solvoing 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).

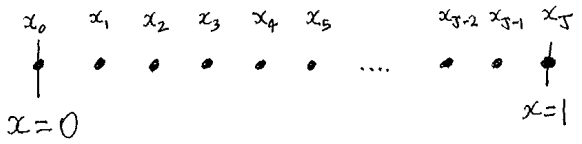

Figure 1: Notations for spatial grid.

Substituting these approximations into eqn. (3) and rearranging, it is easy to show that

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

However, as you will show below, this very simple discretization is unconditionally unstable for typical wave equations. You will implement an additional scheme, the Lax-Wendroff method, which is second-order accurate in time and has improved stability. This method will be introduced in the Q3.

Animations in matplotlib Animations are possible with 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
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 replots 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 does represent a bit more work.

Physics background

Electrostatics. In empty space, the electrostatic potential $V(x,y)$ satisfies Laplace's equation $\nabla^2 V = 0$. In Q1, we will use relaxation methods as in Newman's Chapter 9 to solve for V given some boundary conditions.

Shallow Water Equations (for Q2-3). The shallow water equations 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

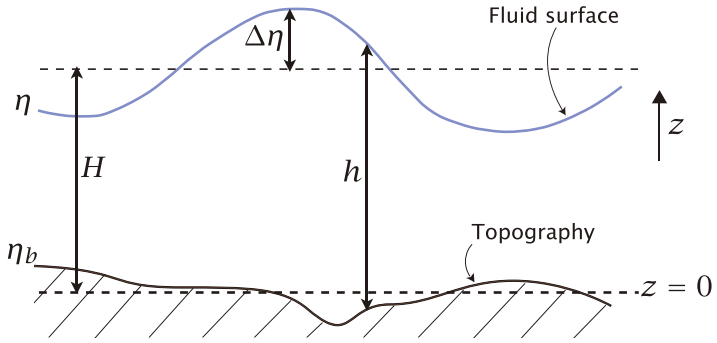

Fig. 3.1

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 simulating tsunamis in the ocean. In this lab, we will attempt to do the latter, albeit in a highly simplified setting.

For simplicity, we will restrict ourselves to the 1D version of the shallow-water equations, which are (see also fig. 2 for notations)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} \quad \text{and} \quad \frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}(uh) = 0, \tag{6}$$

where u is the fluid velocity in the x directions, $h=\eta-\eta_b$ 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. Note that all the variables are functions of x 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.

In particular, 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. **[25% of the lab] Electrostatics and Laplace's equation** (adapted from Newman, ex. 9.3, p. 417)

   (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.

   For simplicity let us model the system in two dimensions. Using the *Gauss-Seidel* method without overrelaxation, write a program to calculate the electrostatic potential in the

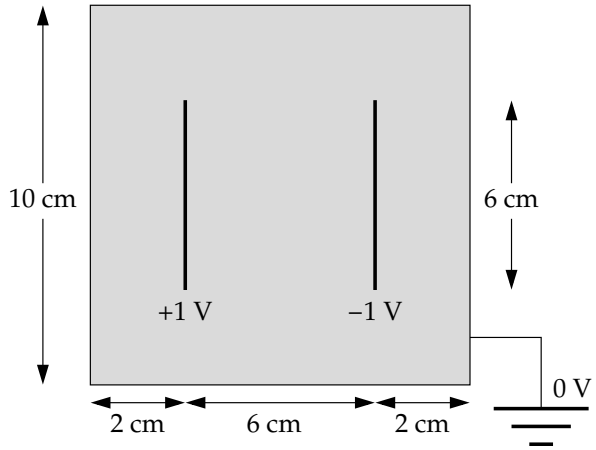

Figure 3: Capacitor for Q1 (from Newman, p. 417).

box on a grid of 100 × 100 points, where the walls of the box are at voltage zero and the two plates (which are of negligible thickness) are at voltages ±1 V as shown. Have your program calculate the value of the potential at each grid point to a precision of 10^{-6} volts and then make a contour plot of the potential (filled or not), and a stream plot of the electric field lines, colour-coded by the value of the electric potential. You may overlay them on the same figure (careful about colour schemes and readability!) or plot two separate figures.

Note: This exercise is similar to Newman's Ex. 9.1, which you don't have to do. In effect, the capacitor plates are part of the boundary condition in this case: they behave the same way as the walls of the box, with potentials that are fixed at a certain value and cannot change.

HAND IN PLOTS AND WHICHEVER COMMENTS ARE RELEVANT.

(b) Now, implement the same method with overrelaxation. Try with $\omega = 0.1$ and $\omega = 0.5$. What do you notice?

SUBMIT YOUR WRITTEN ANSWERS, AND CODE.

2. **[50% of the lab] Simulating the shallow water system, Part I**

(a) **Problem set up:** 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

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

Now 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).

HAND IN YOUR DERIVATION.

(b) **FTCS Implementation:** 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!):

- Take your domain as $x = [0, L]$, with $L = 1$ m, and use a grid spacing as shown in fig., with $J = 50$, so $\Delta x = 0.02$ m.
- Take $g = 9.81\ \mathrm{m\,s^{-2}}$
- Assume $\eta_b = 0$, $H = 0.01$ m (flat bottom topography).
- As boundary conditions, you should impose $u(0,t) = u(L,t) = 0$, 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:

$$\left.\frac{\partial F}{\partial x}\right|_0^n \approx \frac{1}{\Delta x}\left(F_1^n - F_0^n\right), \qquad \left.\frac{\partial F}{\partial x}\right|_J^n \approx \frac{1}{\Delta x}\left(F_J^n - F_{J-1}^n\right). \tag{8}$$

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

$$u(x,0) = 0, \qquad \eta(x,0) = H + Ae^{-(x-\mu)^2/\sigma^2} - \langle Ae^{-(x-\mu)^2/\sigma^2}\rangle, \tag{9}$$

with $A = 0.002$ 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

See the background material for an example of how to create an animation out of a series of lineplots, which will be helpful for visualizing your results. As part of your write up, you should include plots of $\eta(x,t)$ at $t = 0$ s, $t = 1$ s and $t = 4$ s.

Note 1: you may (actually, should) find that your solution eventually diverges. Do not be alarmed, this doesn't necessarily mean you have an error in your code!

Note 2: 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. 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.*

HAND IN YOUR CODE, PLOTS, EXPLANATORY NOTES.

(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 Newman's 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 6a,b about $(u,\eta) = (0,0)$ are, with constant topography η_b,

$$\frac{\partial u}{\partial t} = -g\frac{\partial \eta}{\partial x} \quad \text{and} \quad \frac{\partial \eta}{\partial t} = -H\frac{\partial u}{\partial x}. \tag{10}$$

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,

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

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

HAND IN YOUR DERIVATION.

3. **[25% of the lab] Simulating the shallow water system, Part II**

We can do much better than the FTCS scheme. As one example (of many possible computational schemes), in this question you will implement the Two-Step Lax-Wendroff scheme. There is a slightly higher computational cost for this scheme, but it leads to improved stability and accuracy.

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, i.e.,

$$u^{n+1/2}_{j+1/2} = \frac{1}{2}\left(u^n_{j+1} + u^n_j\right) - \frac{\Delta t}{2\Delta x}\left(F^n_{j+1} - F^n_j\right). \tag{12}$$

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

$$u^{n+1}_j = u^n_j - \frac{\Delta t}{\Delta x}\left(F^{n+1/2}_{j+1/2} - F^{n+1/2}_{j-1/2}\right). \tag{13}$$

(a) Implement this scheme, and run it using the same set up described above. Again, save figures at $t = 0, 1, 4$ s.

HAND IN YOUR PLOT(S) AND BRIEF COMMENTS.

(b) As a final step, you should allow for variable bottom topography. This should be a relatively straightforward change to your code. Instead of having η_b be a constant, it will be an array defined on your grid. Thus, when you calculate the fluxes F^n_j 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 η_b 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.

- Domain: $x = [0,1]$, $J = 150$, so $\Delta x = 1/150$.
- Average altitude of free surface $H = 0.01$ m,

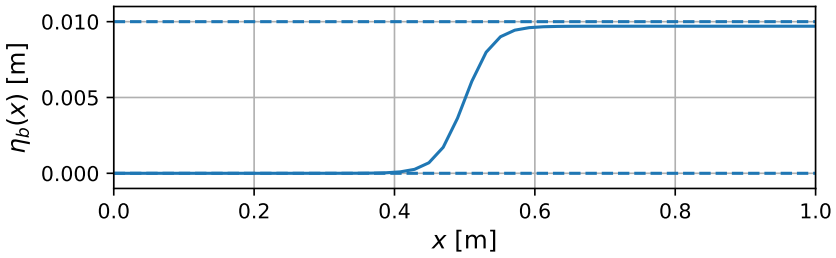

Figure 4: Bottom topography for tsunami simulation.

- Use a bottom topography that goes from $\eta_b = 0$ on the side of the abyssal ocean, has a sharp change in depth in the middle of the domain, simulating the presence of a continental shelf break, to end up at $\eta_b = H - 4\times10^{-4}$ m (see Figure 4), namely,

$$\eta_b(x) = \frac{\eta_{bs}}{2}\{1+\tanh[(x-x_0)\alpha]\}, \tag{14}$$

  with $\eta_{bs} = H - 4\times10^{-4}$ m, $\alpha = 1/(8\pi)$ m^{-1} and $x_0 = 0.5$ m.

- Boundary conditions: $u(0,t) = u(L,t) = 0$, i.e., rigid walls.

- Timestep: $\Delta t = 0.001$; number of iterations: 5,000 at least.

- Initial conditions (wide gaussian bump at far left boundary, i.e., in deep ocean), i.e.,

$$u(x,0) = 0, \qquad \eta(x,0) = H + Ae^{-x^2/\sigma^2} - \langle Ae^{-x^2/\sigma^2}\rangle, \tag{15}$$

  with $A = 2\times10^{-4}$ m, $\sigma = 0.1$ m, and where $\langle\rangle$ is the same as in eqn. (9).

You should see a shallow, wide bump starting at the left of the domain move to the right, towards the continental shelf. 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 $t = 0$, $t = 1$, $t = 2$, and $t = 4$ s.

HAND IN YOUR CODE, PLOTS, EXPLANATORY NOTES. EVEN THOUGH THE CODE IS A MODIFICATION OF THE PREVIOUS CODE, PLEASE DO NOT TRY TO COMBINE THEM INTO ONE, AND HAND IN SEPARATE SCRIPTS FOR Q2 AND Q3, FOR MARKING PURPOSES.