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

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)
TA & Marker: Alex Cabaj (alex.cabaj@mail.utoronto.ca)
Partial marking by Pascal Hogan-Lamarre (pascal.hogan.lamarre@mail.utoronto.ca)

Due Friday, 5 November 2021, 11:59 pm

It appears that there is no need for a second room. Should it be incorrect and the room overflows, N.G. will open up the room he had for the previous labs (grab the link on the old question documents, e.g., Lab #3).

Room (also for office hour) Alex Cabaj and Nicolas Grisouard,
`https://gather.town/app/vmCKPZTSEKgGYlZI/PHY407AC1`
PWD: `tr0p0sph3r3`

Physics Background

Temperature distributions in heat conducting materials The distribution of temperature $T(x, y)$ in the interior of heat conducting materials, in steady state and in the absence of internal heat sources, satisfies $\nabla^2 T = 0$. To determine the distribution of heat in a body, the solution of Laplace's equation requires boundary conditions, either on the temperature on the boundary or the temperature gradient on the boundary. Mathematically, it is known that there is a unique solution that satisfies Laplace's equation and the boundary conditions. As mentioned in the computational background, you will use a relaxation type method to solve this elliptic equation numerically.

Waves on a string In one dimension the wave equation is given by (9.26):

$$\frac{\partial^2 \phi}{\partial t^2} = v^2 \frac{\partial^2 \phi}{\partial x^2}, \tag{1}$$

where ϕ is the wave field of interest and v is the phase speed of waves. In Exercise 9.5, ϕ corresponds to the vertical displacement of a string under tension, which is initially at rest and then struck with a hammer. In this case, $v = \sqrt{T/\mu}$, with T the tension in the string (in N) and μ the mass per unit length of the string (in kg m^{-1}) (though you will not use this fact).

Burger's equation Burgers' equation is a classical nonlinear wave equation which in one spatial dimension takes the form:

$$\frac{\partial u}{\partial t} + \epsilon u \frac{\partial u}{\partial x} = 0 \tag{2}$$

or, in conservative form,

$$\frac{\partial u}{\partial t} + \epsilon \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0. \tag{3}$$

Burger's equation is like a wave propagation equation in which the phase speed of the wave v is replaced by the actual wave velocity u. This nonlinearity allows the wave to steepen and so is one way to model water waves approaching a beach. As a model system, Burger's equation is the starting point for modelling solitons with the KdV (Korteweg-de Vries) equation (which includes an additional dispersion term).

Computational Background

Boundary Value Problems The first class of PDEs that Newman discusses in Chapter 9 are solutions of elliptic partial differential equations, of which the Laplace and Poisson equations are classic examples. He discusses the Jacobi 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 overrelaxation which is implemented according to (9.17). To obtain the solution of a field $T(x, y)$ that satisfies Laplace's equation

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0, \tag{4}$$

subject to boundary conditions (see Physics Background), this method is written

$$T(x, y) \leftarrow \frac{1+\omega}{4}\left[T(x+a, y) + T(x-a, y) + T(x, y+a) + T(x, y-a)\right] - \omega T(x, y), \tag{5}$$

where the left arrow indicates replacement of the left hand side with the right, and ω is a relaxation parameter. In these methods, you can pick a fixed number of iterations or can choose a level of convergence. In Q1 we will be comparing methods and so will use both approaches.

Animations in matplotlib Animations are possible with `matplotlib`. Here is a sample code that will animate the curve `sin(x-t)`.

```python
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 it 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.

Methods for solving Burger's equation This is to remind you of the notation used in Lab08: the space and time directions are discretized into N_x and N_t steps of sizes Δx and Δt, respectively:

$$x_i = i\Delta x, \quad i = 0, ..., N_x - 1 \tag{6}$$
$$t_j = j\Delta t, \quad j = 0, ..., N_t - 1 \tag{7}$$

Then, we write $u_i^j = u(x_i, t_j)$.

As we saw during the lecture and in § 9.3.2, the FTCS method is not appropriate to solve the wave equation, which the Burgers equation (3) is an extension of. One simple method that is appropriate, if not particularly efficient, is the leapfrog method. We saw it when solving ODEs, and its extension when solving an equation of the type of (3) is

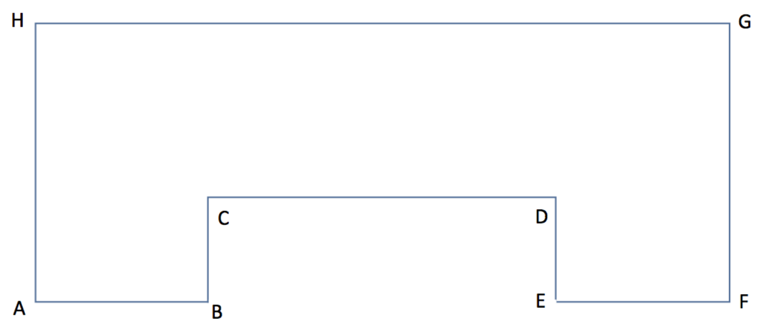
$$\frac{u_i^{j+1} - u_i^{j-1}}{2\Delta t} = -\frac{\epsilon}{2}\frac{\left(u_{i+1}^j\right)^2 - \left(u_{i-1}^j\right)^2}{2\Delta x}, \tag{8}$$

or

$$u_i^{j+1} = u_i^{j-1} - \frac{\beta}{2}\left[\left(u_{i+1}^j\right)^2 - \left(u_{i-1}^j\right)^2\right], \quad \text{with} \quad \beta = \epsilon\frac{\Delta t}{\Delta x}. \tag{9}$$

That is, the discretization is centred in time and in space. It implies that you always need "starter" values at the edges of your space/time domain, for which eqn. (9) does not apply. In space, this is taken care of by the boundary values, while in time, you need to initiate the leapfrog stepping with one forward Euler step.

Questions



Seg-ment	Length (cm)	Temperature (ºC)
AB	5	Increases linearly from 0 at A to 5 at B
BC	3	Increases linearly from 5 at B to 7 at C
CD	10	7
DE	3	Decreases linearly from 7 at D to 5 at E
EF	5	Decreases linearly from 5 at E to 0 at F
FG	8	Increases linearly from 0 at F to 10 at G
GH	20	10
HA	8	Decreases linearly from 10 at H to 0 at A

Figure 1: Domain shape for Q1. The origin of the axes is at A.

1. **Temperature distribution in a heat conductor (35% of the lab mark)**

   Consider the illustration in fig. 1 of a two dimensional rectangular metal plate with an indentation whose dimensions and boundary temperatures are given in the table. The origin of the domain is at A. In other words, A is at $(x, y) = (0, 0)$, F is at $(x, y) = (20\text{ cm}, 0)$ and H is at $(x, y) = (0, 8\text{ cm})$.

   (a) Write a program to calculate the distribution of temperature under steady state conditions and in the absence of heat sources, using Gauss-Seidel relaxation with replacement and overrelaxation. Construct a grid of spacing 0.1 cm. You can use the script from the textbook called `laplace.py` for guidance; note that it implements the Jacobi method. The program should produce a contour plot (your choice of contour lines, filled contour lines, or other type of plot) of the temperature distribution as a function of x and y. Have the script animate the process of iteration, with one frame at each iteration (see the Computational Background information on animations). As the script runs you should see the solution converge to a solution that stops changing much with subsequent iterations. After you have done the rest of the question, hand in this code.

   NOTHING TO SUBMIT (YET).

   (b) To evaluate the impact of overrelaxation, run the program two times for 100 iterations: first with $\omega = 0.0$ and second with $\omega = 0.9$. Produce a plot of the solution for each case. What do you notice about the difference between the two cases?

   HAND IN THE PLOTS AND A WRITTEN ANSWER.

   (c) Now have your program calculate the value of the temperature at each grid point to a precision of 10^{-6} °C, with $\omega = 0.9$. As you watch the animation, does the non-converged solution you obtain look symmetric about the vertical bisector $x = 10$? Why or why not? After the solution converges, what is temperature at the point $x = 2.5$ cm, $y = 1$ cm?

   HAND IN THE CODE AND PLOT OF THE CONVERGED SOLUTION.

2. **FTCS solution of the wave equation (35% of the lab)**

   Do Exercise 9.5 (a) and (b) in Newman.

   HAND IN PLOTS THAT SHOW EVIDENCE OF THE SOLUTION GOING UNSTABLE.

   Notes:

   - *The text suggests using the `visual` module `vPython`, but as far as I can tell, it is discontinued.*
   - *For the animation, I also recommend to not display every frame, but skip every 20 or 100 frames, depending on how smooth you want it. To do so, add a counter to your code, and an if statement of the sort*

```
if frame_counter % 3 == 0:  # will display every 3 frames
    # display one frame
```

3. **Solving Burger's equation (30% of the lab)**

Write a code that solves Burgers' equation using the method as described in the Computational Background for this problem. Recall that you will have to apply one forward time step to the initial condition before being able to apply eqn. (9). You will need to derive the appropriate equation for this step. Also, at the boundary points, i.e. u_0^j and u_{N-1}^j, instead of applying eqn. (9), you should just set the values to be equal to the chosen boundary conditions.

Your code should have parameters for ϵ, Δx, Δt and the length of the spatial domain L_x and for how long the simulation will be run, T_f. Given Δx and L_x, you can find N_x, and similarly for the time dimension. Run your code with $\epsilon = 1$, $\Delta x = 0.02$, $\Delta t = 0.005$, $L_x = 2\pi$, $T_f = 2$. Set the initial condition to be $u(x, t = 0) = \sin(x)$, and the boundary conditions to be $u(0, t) = 0$ and $u(L_x, t) = 0$.

Create and save plots of your solution at time $t = 0$, 0.5, 1, 1.5. What is happening to the disturbance as time progresses? Do you notice any artifacts that seem unrealistic in the solution?

HAND IN CODE, PLOTS, AND WRITTEN ANSWERS TO QUESTIONS.