Data Analysis in Geophysics (CERI 7104/8104)
Homework 2 – Due 10/6/17

This assignment gives you practice "vectorizing" MATLAB codes, something that was discussed in Lab 10, as well as general MATLAB topics including indexing and plotting. We will vectorize finite difference numerical solutions for two types of partial differential equations to compare the time of execution for looped and vectorized versions.

1. **One-way wave equation in 1D.** Consider the one-way wave equation:

$$\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x}. \tag{1}$$

Solutions to this equation are waves traveling from left to right. To solve this equation, we need to specify initial conditions $u(x, t = 0)$ and the boundary conditions at the left side of the domain (say at $x = 0$) $u(0, t)$. We will find a solution over the spatial domain $x = [0, 1]$.

We can construct a finite difference approximation for this equation. We will discretize the spatial domain using a grid spacing of Δx and a time step of Δt, and refer to the solution at spatial point m and time point n as u_m^n. A finite difference approximation for the wave equation is:

$$\frac{u_m^{n+1} - u_m^n}{\Delta t} = -\frac{u_m^n - u_{m-1}^n}{\Delta x}. \tag{2}$$

Using this equation, if we have a solution over the spatial domain at time step n (i.e. we know u_m^n for all values of m), we can advance the solution in time by solving the above for u_m^{n+1}:

$$u_m^{n+1} = u_m^n - \frac{\Delta t}{\Delta x}\left(u_m^n - u_{m-1}^n\right). \tag{3}$$

The boundary condition at $x = 0$ is imposed by setting $u_1^{n+1} = u(0, (n+1)\Delta t)$.

(a) Use MATLAB to solve the one-way wave equation in 1D over the spatial domain [0,1] given the initial conditions $u(x, t = 0) = \exp(-(x - 0.25)^2/0.005)$ and boundary condition $u(0, t) = 0$. Do not worry about the fact that these two conditions are slightly different for $u(0, 0)$, as the difference is small relative to the errors in approximating the derivatives – you can pick either one for setting $u(0, 0)$. Use a grid spacing of $\Delta x = 0.01$ and a time step of $\Delta t = 0.005$. Have your code take 100 time steps. Do this using two nested `for` loops, one over the spatial grid and one over the time steps. Time the execution of your program using `tic` and `toc` as discussed in lab. Plot $u(x)$.

How did the signal propagate? At what speed did the wave propagate (recall that the total time is the number of time steps multiplied by Δt)? What happens to the amplitude of the signal as it propagates? (This is an artifact of the numerical method, not a property of the solution.)

(b) We will find two ways that we can vectorize this procedure, and we will compare the results of both of them. First, vectorize by doing array math; you will need to use a vector that has had its indices shifted by one in order to accomplish this. There are two ways to do this: the fastest is to evaluate a matrix with another matrix, or you can use the `circshift` function, which in my experience is slower. Either way, you can eliminate the `for` loop over the spatial index. You will still have a loop over the time steps. Run this version of the code, and verify that it gives you the same answer as the loop version. Time this version using `tic` and `toc`, and compare with the loop version. Do you notice a substantial difference?

(c) Another way to vectorize is to recognize that you can write Eq. 3 as a matrix multiplication operation (remember that a matrix times a vector gives you back another vector). Figure out what matrix will advance the solution by one time step. You will find some of the techniques for vectorizing definitions of arrays useful for this (the diagonal matrix function `diag` will be useful).

If we want to advance our solution by more than one time step, we can do this by repeatedly multiplying by the above matrix. Since MATLAB can easily take the power of a matrix, we can eliminate the second `for` loop and solve this problem in a single line. Use MATLAB to solve the wave equation in this way, and verify that you get the same solution as before. Time your code, and compare with the other versions.

What version is the slowest? What version is the fastest? Explain in a few sentences why you think the different versions execute in the amount of time that you observe. *Hint:* are the different versions doing the same number of calculations? Does it matter for the execution time? (The exact answer depends somewhat on the details of your implementation, so you should not worry if the execution time does not change significantly).

2. **Heat Equation in 2D.** Now we will look at the time-independent Heat Equation in 2D (also known as Laplace's Equation):

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

This equation describes the steady-state temperature in a material. For this equation, we need to specify values of T on all four sides of the spatial domain, which we will take to be the region $[0, 1] \times [0, 1]$ (i.e. the lower left corner is (0,0) and the upper right corner is (1,1), Fig. 1).

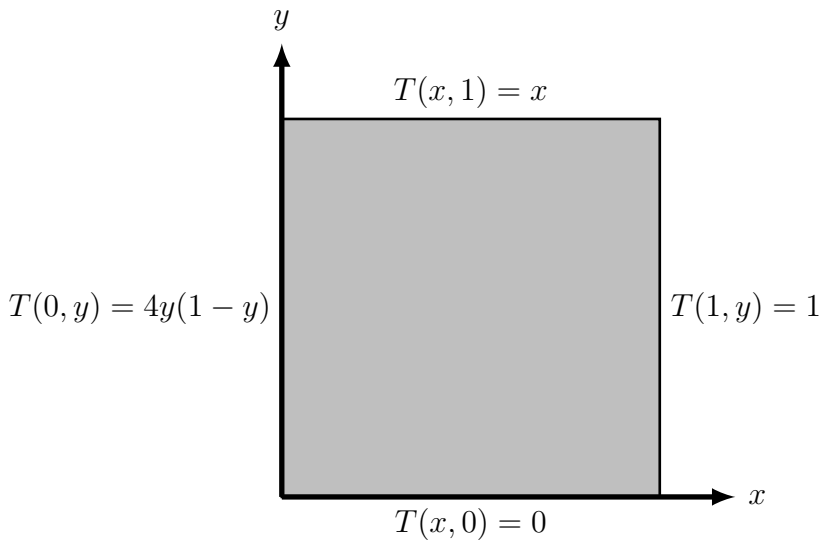

Figure 1: Spatial domain and boundary conditions to be used for solving the heat equation.

We will construct a finite difference approximation to this equation. We discretize the spatial domain using a grid spacing of Δx (this applies to both spatial directions), and define the temperature at point $(m\Delta x, n\Delta x)$ to be $T_{m,n}$. Then an approximation to the governing equation is:

$$\frac{T_{m+1,n} - 2T_{m,n} + T_{m-1,n}}{\Delta x^2} + \frac{T_{m,n+1} - 2T_{m,n} + T_{m,n-1}}{\Delta x^2} = 0 \tag{5}$$

This is a system of equations that we can solve iteratively. First, initialize the solution to any value (zero everywhere is fine), and set the boundary values to the corresponding boundary conditions. Then, update the solution at point $T_{m,n}$ to be:

$$T_{m,n} = \frac{1}{4}\left(T_{m+1,n} + T_{m-1,n} + T_{m,n+1} + T_{m,n-1}\right), \tag{6}$$

while always keeping the value of T at the boundaries at the value specified by the boundary conditions. This is called the "method of relaxation" as the solution "relaxes" towards its steady-state value as you iterate. When the solution stops changing as you iterate, then the solution has converged.

(a) Solve the heat equation using the method of relaxation using MATLAB. The boundary conditions are $T(0, y) = 4y(1-y)$, $T(1, y) = 1$, $T(x, 0) = 0$, and $T(x, 1) = x$ (the set-up is illustrated in Fig. 1), and you should use a grid spacing $\Delta x = 0.01$. First, do this using `for` loops over the two spatial dimensions, and a `while` loop to iterate over the solutions until your

solution has converged. You can consider your solution to have converged when no point changes by more than 10^{-6} when you iterate. Time the execution of your code, and make a plot of the final temperature profile.

(b) Now vectorize your code by doing shifting the array values and then doing array math. After doing this, the only loop left in your code should be the `while` loop that iterates over the solutions. This should be similar to how you did this for the wave equation. Make a plot of the final temperature distribution (it should be the same as above if your code is working correctly), and compare the time of execution of your code to the loop version. Which version is fastest? How does the time difference compare to the differenes you found for the wave equation?

Because the heat equation is an iterative method and thus you do not know the number of times to execute the loop, we cannot eliminate the while loop in our solution.

Submission Information: Please turn in the following, preferably in your Public folder, or as an email attachment sent to me:

- Your code. I recommend writing functions to implement your solution for each sub-problem, plus a driver script that calls the appropriate functions and plots the results. Note that if you want to make your code re-usable in the future on different problems, you should pick appropriate arguments to pass to the functions – for example, for each wave propagation problem, you will need to pass a vector holding initial conditions, plus the time step, spatial grid spacing, and number of time steps. Make sure all of your functions are documented, and your code is commented, properly indented, and readable so I understand what you are doing.

- Two plots, each saved as an EPS, PDF, or PNG file (you are also welcome to save your plots as `.fig` files for your own use, but you should submit a version in a standard graphics format for grading). One plot should show all of your solutions to the wave equation at the final time step (either on the same axes, or as three different subplots), and one plot should show your steady-state solutions to the heat equation as two separate subplots. You should label all relevant axes, and provide a title or legend describing what is plotted. Please be sure that your graphs are clear and legible.

- A write-up containing your answers to all questions posed in this handout (if there is a question mark, you need to write a few sentences addressing that question). Plain text files or word processor files are fine.