Data Analysis in Geophysics (CERI 7104/8104)
Homework 2 – Due 10/26/18

This assignment gives you practice programming in MATLAB. The first problem focuses on how to "vectorize" your code by solving a problem using array operations. The second problem uses seismic backprojection to locate sources of seismic waves.

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. **Seismic Backprojection** Write a MATLAB script to locate the source of a seismic signal using backprojection, as described below. You will apply your code to the included data. "hw2.mat" contains two matrices: the matrix `receivers` contains the (x, y) coordinates (in kilometers) of a set of seismic stations recording signals (the `receivers` matrix is 20×2), and the matrix `data` contains the data recorded at each station (the `data` matrix is 20×5000). Each station has recorded 250 seconds of data at 20 Hz. You will need to write code that takes the recorded signals, applies an appropriate time shift to each signal for a candidate source location, and then adds up the shifted signals and calculates the sum of the squares. Source locations that are more likely will have a larger summed signal (if done correctly, you should identify three sources in the included problem).

You should conduct this analysis using several functions, for which I have pro-

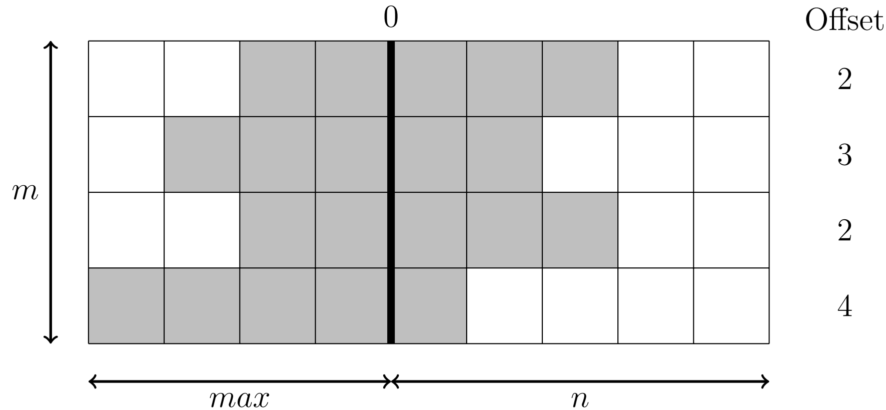
vided template files that you need to fill in. Doing it in this way will ensure that your code can integrate with the other pieces of code that you are writing in the other homework assignments. Additionally, it is important that you not assume any specific input sizes for the number of source points, receiver points, or data samples, as this will ensure that your code works correctly for other problem sizes.

- The function `travel_time` takes a source point and a matrix of receiver points and calculates the travel time between all possible pairs of them. The source point is a vector of length 2 holding the (x, y) coordinates of the source point in km, and the receiver points are contained in a matrix with dimensions $m \times 2$, where m is the number of receivers, holding the (x, y) coordinates in km. You should assume that the source is at a depth of 30 km, the receivers are at the surface, and the p-wave travels at 6 km/s. The function should return the travel time (in seconds) between the source and all receiver points. (This should sound familiar to you; your Python code will be used to determine these travel times for your final project.)
- The function `time_shift` takes a vector of travel times (in seconds) of length m and a sampling frequency (in Hertz) and calculates how many samples each travel time corresponds to, returning a vector holding the shift numbers (note that each shift must be an integer). This information will be used to determine how to shift each signal when you perform the backprojection.
- The function `compute_amplitude` takes a matrix holding a number of time series (size is $m \times n$, where there are m stations and each station has recorded n samples) and a vector holding m integer offsets. This function does the actual backprojection by shifting each of the m signals by the appropriate amount, then adding all m signals together, squaring the summed signal, and then summing again over all n time samples. The function should return the summed value of the shifted time series.

  Hint: First create a matrix of zeros that has dimensions $m \times (n + max)$, where max is the largest offset in the vector of offsets. Then go through all m stations, and put the shifted signal into the correct position in the matrix, with one signal in each row. Sum up each column, square the array, then sum the row.

  To determine how to shift each signal, you can think of the final n entries in each row as the place where the signal would be if there was no time shift. Each signal will then be offset backwards in time by the shift for that particular station. Note that true "backprojection" would involve reversing the signal in time and then shifting the reversed signal *forward* in time based on the travel time to model the signals traveling to the source location; however since we are summing up all of the time samples, it is

equivalent to keep the signal as forward in time and then shift the signal *backwards* in time. This is illustrated below for a case where there are 4 stations and each station has 5 samples – matrix entries shaded gray are nonzero, and the heavy vertical black line indicates the zero offset position.



You will be able to do this calculation without any loops and using only array math, though you will need to do a single loop over all m stations first to populate the matrix holding the shifted signals.

- The function `backprojection` takes a matrix of source points with dimensions $s \times 2$, where s is the number of source points to be considered holding the (x, y) coordinates of all source points in kilometers, a matrix of receiver points of dimension $m \times 2$, where m is the number of receivers, a data matrix holding a number of time series (size is $m \times n$, where there are m stations and each station has recorded n samples), and the sampling frequency (a single number in Hertz). This function will then call the necessary functions described above to compute the backprojection. It returns a matrix with dimensions $s \times 3$ where the first two columns are the x and y coordinates of the source point and the third column holds the calculated backprojected amplitude for each source point.
- You should try all possible source locations on a 100 km by 100 km grid, with a resolution of 1 km in each direction. Plot your results using `pcolor` to display your summed backprojected signal (`pink` and `hot` are good colormaps to use for this particular problem, though I would reverse the colors by using `colormap(flipud(pink))` to set the colormap) and `scatter` to show the location of all stations. `pcolor` requires that your data be in a matrix, so you will need to reshape the output of `backprojection` to a 100×100 matrix for plotting.
- You also need to write a driver script that loads the data, creates the matrix of source points, calls the `backprojection` function with the appropriate arguments, and plots the results.

Submission Information: Please turn in the following, either 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. I have provided the necessary function files for the second problem, but it is up to you to create the necessary files beyond that in order to make your code well organized and understandable.

- 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 image of possible seismic source locations as a pseudocolor plot with the seismic station locations overlaid on the plot as described in the problem statement. Be sure that you label all axes and use a font that is large enough to be easily legible.

- A write-up containing your answers to all questions posed for problem 1 (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.