CERI 7104/8104 Homework 2 Solutions

Here are my solutions to Homework 2. I have included three functions that solve each of the individual problems, as well as the driver code "hw2.m" that sets up the problems, calls the three functions that produce the solutions, and then creates the plot that I show here.

Many of you may have seen different times than you will see if you run my solutions. That is okay: there are some differences in the exact implementations, the other tasks a computer is carrying out when you do the computations, differences in memory access times depending on exactly how you set up the problem, etc. If you are doing a long and serious calculation in MATLAB, you would want to experiment with different ways of vectorizing (or not vectorizing) in setting up your calculation. You would also want to do a more statistically robust way to determining average execution times, which was not necessary here.

1. (a) Most of you had no trouble with the loop versions of the code. The solution propagates from left to right with a dimensionless speed of 1. You can see this by noting that the initial Gaussian was centered at $x = 0.25$ at $t = 0$, and at $t = 0.5$ the pulse had moved to $x = 0.75$, giving a speed of $v = 0.5/0.5 = 1$. The amplitude decreases slightly as the pulse propagates. Initially, the peak value is $u = 1$, but at the end of the calculation the peak value has decreased to about $u = 0.75$. As I alluded to in the homework problem statement, this is an artifact of the finite difference method, and the amplitude is not supposed to change. More accurate methods than this simple 1st order approximation give much better amplitude values.

   Some of you had trouble figuring out the wave speed. Some people used the absolute value of the signal amplitude, rather than the spatial location of the peak, some did not correctly account for the amount of time that elapsed during the calculation.

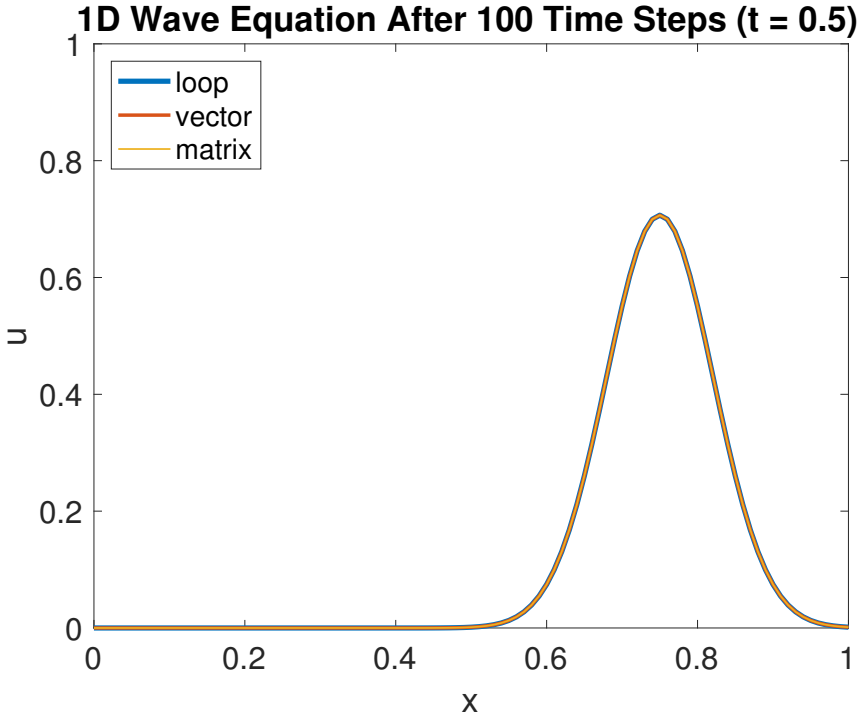
   One other "problem" with many people's code is that they defined the initial conditions inside of both `for` loops. While this is not incorrect, it is unnecessary and thus less efficient than defining the initial conditions once before starting the loops. I did not take off points for this, but noted when you did some unneeded calculations.

   (b) There was not much trouble with the vectorized version of the code. This version can be slower or faster than the loop version, depending on exactly how you implement it, so your code may not produce exactly the same results as mine. It seems that the `circshift` function is slower than doing the shift using `u([1 1:end-1])`, so some people saw speedups, while others did not.

   (c) The trick to this part is figuring out how to set up the matrix, and then

using a matrix power to advance the solution to the final time step in one line. Matrix multiplication involves many more operations than the looping versions of the code, and since most of the operations are trivial multiplication by zero or addition of zero. Many of you did not specify this when comparing the methods. Even though this does not make a big difference here (and some of you even found that matrix multiplication was the fastest method for you), the extra operations can be costly for big problems.

Here is a plot of my final solution using all three methods (they plot one on top of another and are essentially the same):



In general, most codes were correct. Documentation on the functions was not always adequate – at the very least, your function documentation should explain what a function does and what its inputs and outputs are. For examples, see the solutions and my code for the MATLAB 4 lab for examples. While I don't expect everyone's code to be as polished as mine, it is important that your documentation explain these basic aspects of the function. The entire idea of using functions is to promote code re-use, and making the interface clear means that you do not have to wade through the entire code to know how to use it. Some of you did not check any of your inputs with assertions, which is something you should *always* be in the habit of doing. In this case, you should check that the time step, grid spacing, and number of time steps are positive, otherwise your code will not work correctly.

Additionally, not all of you wrote your code in the most flexible manner to let you re-use it. In particular, some of you did not write code that let you use an arbitrary number of grid points or time steps, or arbitrary initial/boundary conditions. If I were to give you another set of problems to do that had a different number of grid points, time steps, initial conditions, or boundary conditions, you would need to write your code again from scratch, which defeats much of the purpose of writing a function to begin with. In general, you should try to reduce each function call to do one specific task, which in this case is to either solve the wave equation according to the given method, with the number of grid points, time steps, grid spacing, time spacing, tolerance, and boundary conditions specified elsewhere.

Note that in my solutions, the function input parameters provide all of these details to the function, and each of my functions just solves the sets of equations, independent of the details of the problem. If I wanted to run my code again with a different set of inputs, all I would need to do is change the places in the driver script that defined the problem. I would not need to change the functions themselves, as they will keep on working just fine and can be re-used. This is much more flexible, and will save time down the road. This is something to keep in mind when writing programs for your research.

2. For the backprojection problem, most of your codes were well organized and solved the problem correctly. I graded your codes on correctness (8 points) by using a series of test functions (for the `compute_amplitude`, `travel_time`, and `time_shift` functions), though I found in a few cases you did not pass due to a minor difference between your implementation and mine. In those cases, if your code appeared to be correct to my eye, I still gave you full credit. I also graded on coding style (7 points), which was good for the most part other than a few functions that were not sufficiently documented. As with above, see my solutions for the wave equation functions for examples.

   All of you missed some important cases for assertions at the beginning of each function. All three functions needed to have at least one assertion by my count (some of you added some additional ones, which is fine), and my test functions looked for these cases and I took off points if you did not check for that condition somewhere in your implementation. In particular, your travel time function needed to check that the source and receiver coordinates had a length of 2, your time shift function needed to check that the travel times and frequency were positive (note that your backprojection code will crash if you try to shift the signal by a negative amount), and your compute amplitude function should check that the number of signals and offsets matched. Be sure you catch these cases when you resubmit your code for the final project.

   As with the Python program, I will give you one additional chance where I will

run your code on my test functions and let you know the results (in particular if it passed the assertions, as most people received full credit on the main tests), and I am happy to look at your code again if you lost points on documentation.