PX390, Autumn 2022, Assignment 3 (edited 7th Nov)

November 7, 2022

The purpose of this exercise is to further expand your knowledge of C and your ability to understand and test numerical code.

You are given a code that should solve a diffusion equation (instead of previous years' elaborate backstory, lets just say the person that wrote it quit and left in a huff), but is broken in various ways, and needs to be fixed. Your job is to read the specification, and to produce a correct code that does what the specification says it should do

You should submit a single C source file for this assignment. You should submit your code via the link in the assignment section of the moodle page.

The code must compile and generate no warnings when compiled with

```
gcc -Wall -Werror -std=c99 -lm
```

There are a finite number of bugs, but I'm not going to tell you how many: make sure that your code actually works by testing it rather than just looking for obvious errors.

Tips:

1. The compiler is your friend. Start with the first warning/error messages and work through them until there are none left.

2. Some bugs are hard to catch just by inspecting the code. Adding printf statements is usually the easiest way to check that the code is doing what you think it is: check the maths at the first timestep, for example.

3. You can often catch bugs in numerical code by looking at the output graphically: does it do something wierd at the boundaries? A variety of tools (Matlab/matplotlib/Origin) exist which can take numerical output and plot it for you. This is one of the things you should learn while doing this assignment as it will be essential later on.

4. Learn how to use debuggers (gdb). Memory checking tools like valgrind can help catch issues to do with reading/writing into an incorrect memory location.

5. The boundary/initial conditions are a bit complicated: if you want to test that your code is correctly solving the equation, you can temporarily choose simpler ones with closed form analytic solutions.

6. Bugs include both incorrect lines of code as well as missing functionality. There are comments which are misleading: I'm not treating these as bugs, but you might like to fix them as you go. I note that the 'read_input' function, and the corresponding function prototype are actually correct; please don't touch them.

Specification:

The code must use a simple difference scheme to stably solve the differential equations

$$\frac{\partial Y}{\partial t} = C\frac{\partial^2 Y}{\partial x^2} + Z(x) \tag{1}$$

on a 1D x domain $x \in [-L/2, L/2]$, as an initial value problem. We define

$$Z(x) = S\exp(-x^2) \tag{2}$$

The domain length, grid size, length of time over which to solve the equation and the coefficients C and S are read in from a file: the function that reads this data (and the function prototype) is the only part of the code that is bug free (i.e. don't change this bit), but the way it is called may not be right.

Boundary conditions for $Y(x,t)$ in the x domain are $Y(L/2,t) = Y(-L/2,t) = 0$. The initial condition is $Y(x,0) = 0$. There are N_x grid points, with the first grid point at $x = -L/2$, and the final point at $x = L/2$.

Input:

A file 'input.txt' is used for input: note that there is an example on the Moodle page. Each line contains one parameter: L on the first line, then the number of grid points N_x, the diffusion coefficient C, the simulation time t_F, the timestep d_t, the number of steps per diagnostic output n_D, and the source amplitude S on the last line. You may assume all these inputs are positive.

If the timestep is too large, which we take to be true if $Cd_t/d_x^2 > 1/8$, the code should immediately exit with a warning. It should also exit immediately with a warning if it was unable to read the parameters.

Note, because of an error in the specification, the value S was not used in the first version of this specification document.

Output:

The code outputs simulation data at a fixed interval in time, at the end of every n_D timesteps: it should output the initial values (at t=0), and at $d_t n_D, 2d_t n_D$ etc. This may or may not include the the final computational timestep.

The time t, x coordinate, and the corresponding value of Y, are written in a single output line for each gridpoint after each timestep. Please don't add extra comments/blank lines to the output!