PX390 2023-24, Assignment 3

1 Intro

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 which should solve two coupled partial differential equations. The code is, however, broken and needs to be fixed. Your task is to read the specification, understand the program, and 'debug' it. At the end of this task the program should do what these specifications say it is meant to do. Some of these problems with the code are basic C programming errors, other are more subtle mismatches between the required and actual code behaviour.

You should submit a single C source file with a list of corrected bugs, at the top of the C source file (in comments), where I have added a placeholder (this description will not be marked but helps both of us keep track of what you have changed). You may 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 -o assign3 assign3.c -lm
```

There are a finite number of bugs, but we will not tell you how many: make sure that your code actually works by testing it rather than just looking for obvious errors. Some errors may be repeated in multiple statements, or may effect multiple lines of code.

1.1 How is it marked

Marks are given for the absence of each bug (in an otherwise correctly working code) and for the code compiling correctly. You may lose marks in more than one category for sufficiently severe problems. A correctly working code will contain a stable and consistent finite difference method for solving the equations below.

1.2 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 strange 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. It is expected that you will allocate arrays on the heap and that the allocated memory is released (freed) after use.

2 Specification

The code must use a simple, convergent, finite difference scheme to solve the coupled differential equation for real-valued functions u and v with

$$\frac{\partial u}{\partial t} - K\frac{\partial^2 u}{\partial x^2} - v(u+1) = 0 \tag{1}$$

$$\frac{\partial v}{\partial t} - K\frac{\partial^2 v}{\partial x^2} + u(v+1) = 0 \tag{2}$$

The equation is to be solved on a 1D x domain with $x \in [0, \text{length}]$, as an initial value problem. The domain length, number of grid points, length of time over which to solve and the coefficients are defined at the start of the main.

The initial condition is $u(x,0) = 1.0 + \sin(2\pi x/[\text{length}])$, $v(x,0) = 0.0$. The first grid point is at $x = 0$, and the final point at $x = [\text{length}] = (nx-1)\times\delta x$. The function and its derivatives are periodic, so periodic boundary conditions should be enforced, that is, $u(x=0) = u(x = [\text{length}])$ and $v(x=0) = v(x = [\text{length}])$. The provided code may take a slightly different approach to periodic boundaries than your used to so examine closely. Please note that at no point should the values of u or v blow up to infinity.

2.1 Output

The code outputs simulation data at a fixed interval in time equal to the chosen time step dt: it should output the initial values (at t=0), and at $dt, 2dt, 3dt \ldots$ (but not necessarily the final value).

The time t, x coordinate, and u and v are written in that order in a single output line for each gridpoint. **Do not add extra comments/blank lines to the output**, which should contain only numbers! Remember that I will need to read the numbers, so if it is output in an unreadable format then you'll lose marks.