PX390, Autumn 2019, Assignment 4: Spontaneous symmetry breaking

November 12, 2019

1 Introduction

This is the first assignment where you will build a somewhat sophisticated piece of numerical software from scratch.

At the dawn of the universe, the huge energy densities meant that all the forces were unified together, and matter was in a statistically uniform state. Later on, a condensation process occurred, where certain initially symmetric features of the universe because highly asymmetric. For example, the observable universe is filled almost entirely with normal matter, rather than antimatter, even though the basic physical laws are symmetric with respect to the switch. The dominant explanation for the broken symmetry is a process called spontaneous symmetry breaking. The idea is that even if the basic laws are symmetric, small perturbations that move the system away from a symmetric state tend to increase with time. The obvious example is a ball placed on top of a hill. Even though the hill is symmetrical, we expect that in practise the ball will end up rolling down a particular side of the hill at some point.

As a model problem to show how the early universe might separate into regions with different local properties, we consider an equation that models the evolution of a 2-D vector field on a 1-D spatial domain as a function of time.

Note that you don't need to understand anything about quantum mechanics or cosmology (certainly I don't!): just solve the equations below and consider the description of the problem as light entertainment. The boundary conditions in particular are chosen to pick out certain features of the dynamics rather than be physically relevant.

2 Equations to solve

What you will be simulating is the *state vector* $\mathbf{Z}(x,t)$, with $\mathbf{Z}$ a 2-D cartesian vector ($\mathbf{Z} = (U,V)$ for real U and V) as a function of distance along the line x, in the periodic domain $[0,L]$, and time t, in the domain $[0,t_f]$. The equation to be solved is

$$\frac{\partial \mathbf{Z}}{\partial t} = \frac{\partial^2 \mathbf{Z}}{\partial x^2} + \nabla(Z^2 - Z^4) \qquad (1)$$

with $\nabla = (\partial/\partial U, \partial/\partial V)$. We have periodic boundaries so $\mathbf{Z}(x+L,t) = \mathbf{Z}(x,t)$

The initial condition is given by

$$\mathbf{Z}(x,0) = [K + A\cos(2\pi x/L)]\hat{\mathbf{U}} + B\sin(2\pi x/L)\hat{\mathbf{V}} \tag{2}$$

where we have $\hat{\mathbf{U}} = (1,0)$ and $\hat{\mathbf{V}} = (0,1)$.

The algorithm used must be stable (I will describe what is required later in this document) and be at least second order in space and first order in time. Note that I do not require you to use an implicit method. Stability may require choosing a sufficiently small timestep.

The code must compile and generate no warnings when compiled with

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

3 Stability

The equations are nonlinear: in class we have only considered linear stability of numerical schemes. For convergence of nonlinear equation solvers it is often sufficient to ensure that the short wavelength modes (on the grid scale) are locally stable when linearised around a smooth large-scale solution.

I expect that given a nearly homogeneous state $\mathbf{Z}(x,t) = \mathbf{R} + \epsilon\delta\mathbf{Z}(x,t)$, that the choice of numerical scheme and timestep will be such that the linearised equations for $\delta\mathbf{Z}$ will be numerically stable (in the limit where ϵ is small). You should find this linearised equation for $\mathbf{Z}$ and apply Von-Neumann stability analysis. Note that the continuous equation has an instability around certain values of $\mathbf{Z}$.

4 Specification

The x grid has N grid points; the grid will be taken to run from $x_0 = 0$ to $x_{N-1} = L - \delta x$, where δx is the grid spacing.

5 Input

The input parameters will be read from a file called 'input.txt'.

1. L, right x boundary of domain.
2. N, number of grid points
3. t_f, length of time to run for.
4. t_D, timestep for diagnostic output.
5. K, initial condition parameter.
6. A, initial condition parameter.

7. B, initial condition parameter.

I will in general leave you to determine whether these should be read as integers or floating point numbers based on their meaning (note that the example input file does not necessarily have decimal points on all the values that should be floating point).

Note that I am not requiring you to decide whether the input parameters given are sensible. You may print warnings to standard output if you wish. It is sufficient to use scanf to read in the input parameters and simply test that the reads were successful: I am not specifically forbidding the use of other ways of reading input but I found roughly 50% of students attempting a more complicated input scheme got it wrong.

There is an example input file and output file on the assignment page.

6 Output

The code outputs simulation data into a file called 'output.txt' at a fixed interval in time, t_D: it should output the initial values at t=0 and those at $t = t_D, 2t_D, 3t_D$ etc.

The time, t, x coordinate and the values of $U(x,t)$ and $V(x,t)$ (the components of $\mathbf{Z}$) are written in a single output line for each gridpoint, at each output time. Please don't add extra comments/blank lines to the output! See the example output file if you are unclear of the correct format (this output just shows the general idea rather than the correct output grid values for the example input).