School of Physics & Astronomy

THE UNIVERSITY of EDINBURGH

Modelling and Visualisation in Physics

PHYS10035 (SCQF Level 10)

Friday 12th May, 2023 13:00 - 16:00
(May Diet)

Please read full instructions before commencing writing.

Examination Paper Information

Answer all questions

Special Instructions

- **Open Notes examination:** students are permitted to use course notes and own prepared material, but **not** published textbooks, during examination.
- Only authorised Electronic Calculators may be used during this examination.
- A sheet of physical constants is supplied for use in this examination.
- Attach supplied anonymous bar codes to *each* script book used.

Special Items

- School supplied Constant Sheets
- School supplied barcodes

Chairman of Examiners: Prof. James Dunlop
External Examiner: Prof. Ian Ford

Anonymity of the candidate will be maintained during the marking of this examination.

In this exercise we will consider a cellular automaton model for the rock-paper-scissors game. You will first implement a parallel deterministic update rule, and then a random sequential one. In the model, each cell in a two-dimensional $N \times N$ square lattice can be in one of three states: 'rock' (R), 'paper' (P) or 'scissors' (S). Each cell has eight neighbours: four along the principal axes and four along the diagonal (i.e., as in the Game of Life). We consider periodic boundary conditions to determine the neighbours of each cell.

a. In the parallel deterministic version, all cells in the $N \times N$ lattice are updated at the same time, according to the following three rules:

   - A cell in the R state with more than two (i.e., 3 or more out of 8) neighbours in the P state changes its state to P.
   - A cell in the P state with more than two S neighbours becomes S.
   - A cell in the S state with more than two R neighbours becomes R.

   Write a Python code to follow the evolution of this rock-paper-scissors parallel update cellular automaton. Your algorithm should allow you to change the size of the lattice, N. The initial condition you should consider is one in which the lattice is divided into three 'pie wedges', with a different state in each of the wedges. **[15]**

b. Describe the behaviour you see when you run your code with $N = 100$ (i.e., a 100×100 lattice). Record the number of R states at a fixed point in the lattice over time, and plot it on a graph, commenting on the results. **[5]**

c. We now consider a variant of the algorithm, which is random and sequential, rather than parallel and deterministic. As an initial state, you should now consider one where each cell is randomly set to either R, S, or P. The update rules in this case become:

   - A cell in the R state with at least one neighbour (1 out of 8) in the P state changes its state to P with probability p_1.
   - A cell in the P state with at least one S neighbour becomes S with probability p_2.
   - A cell in the S state with at least one R neighbour becomes R with probability p_3.

   Write another Python code to follow the evolution of this rock-paper-scissors random sequential cellular automaton. While this new code may be a small variation of the previous one, it is still best to keep two separate codes. **[10]**

d. By using your random sequential code and fixing $N = 50$, $p_1 = p_2 = 0.5$, find the average fraction of the minority phase in steady state as a function of p_3 and its variance, for $0 \leq p_3 \leq 0.1$. You should choose a suitable resolution in p_3 for the calculation, plot the data in a graph and comment on the results. You do not need to add error bars. Note the minority phase is the one corresponding to the smallest fraction in the lattice between R, S and P (this could change in principle for different values of p_3). **[10]**

e. Compute the average fraction of the minority phase for $p_1 = 0.5$ and varying p_2, p_3, in the ranges $0 < p_2 < 0.3$ and $0 < p_3 < 0.3$. Plot the resulting data as a heatmap, commenting on the results. **[10]**