2. Cellular automata

3.1. The game of life

The evolution of the Game of Life on a lattice is fully deterministic and is set by the following set of rules:

- Any live cell with less than 2 live neighbours dies.
- Any live cell with 2 or 3 live neighbours lives on to the next step.
- Any live cell with more than 3 live neighbours dies.
- Any dead cell with exactly 3 live neighbours becomes alive.

Note that in Game of Life, unlike in the Ising model (and in the SIRS model which we consider next), it is conventional to consider as neighbours all cells which have at least a point in contact with a given cell; i.e. there are 8 neighbours for each cell (north, south, east, west, and the four neighbours along the lattice diagonals).

1. Write a Python code to simulate the game of life on a 50×50 2d square lattice with periodic boundary conditions. Your program should allow the user to choose between a random initial condition and one in a set of selected initial conditions, which should include a state developing into an oscillator and a state developing into a glider (or a perpetually moving structure). You should consult the lecture notes when building such selected initial conditions. Your program should also show a visualistion of the state of the system as it is running.

2. For the glider, code up a function which tracks the position of its centre of mass as a function of time. The centre of mass of a set of lattice points $\{\mathbf{r}_i\}_{i=1,\ldots,n}$ is given simply by $\mathbf{r}_{\rm cm} = \frac{\sum_{i=1}^{n} \mathbf{r}_i}{n}$. However, you should account for periodic boundary conditions, for example by discarding data points for which the glider is crossing some of the boundaries. Use the centre of mass data to estimate the speed of the glider. [If you prefer/feel like, you can replace for this exercise the glider with any other moving structure which you may have found.]

3.2. The SIRS model for epidemics spreading

The SIRS model comprises agents (i.e., people or other organisms), i, arranged on a two-dimensional lattice. The agents may contract some infection (e.g., the annual flu), and can be in one of three states:

- S: susceptible to the infection;
- I: infected;

- R: recovered from the infection (and hence immune to it).

The SIRS model is called like this because agents go through these states cyclically: S changes to I when a susceptible agent comes into contact with an infected one; recovery changes the agent from I to R; eventually, in the absence of contact with infected agents, the immune response wears off and R changes back to S.

More precisely, the approach which we shall use in this Checkpoint to simulate this model is through a random sequential updating scheme. Here, as in the Ising model, only one site is updated in each timestep, and the site to update is chosen randomly. Again as in the Ising model, if there are N sites in the system, N such updates is called a sweep. The set of rules which define the (stochastic) SIRS model is as follows,

- $S \to I$ with probability p_1 if at least one neighbour of i is I; otherwise site i is unchanged.
- $I \to R$ with probability p_2.
- $R \to S$ with probability p_3.

1. Write a Python code to simulate the SIRS model on a 50×50 2d square lattice with periodic boundary conditions, (so that $N = 50^2 = 2500$). Your program should allow the user to choose the system size, probabilities p_1, p_2 and p_3, and use the random sequential updating scheme. As usual, it should also be able to show a visualisation of the state of the lattice as it is running.

2. Find some parameter sets which leads to (i) an absorbing state with all sites susceptible to the infection; (ii) a dynamic equilibrium between S, I and R; (iii) a case with cyclic wave of infections through the lattice. Visualise the dynamics in each case.

3. Now set $p_2 = 0.5$. The average number of infected sites, $\langle I \rangle$, is a quantity that provides a quantitative measure of the different possible behaviours (just as the magnetisation in the Ising model). Using this measure, obtain the phase diagram of the system in the $p_1 - p_3$ plane (i.e., a diagram showing the regions where qualitatively different behaviour emerges). To do this, you should plot, in the same $p_1 - p_3$ plane, the value of the average fraction of infected sites $\langle I \rangle / N$, e.g. as a contour or colour plot. To do the averaging, for each value of (p_1, p_3), you can use a long simulation after equilibration: you can use 100 sweeps as equilibration time, and a runtime of 1000 sweeps (measurements every 10) will be sufficient for this problem. If you have time, you can rerun the code (the random numbers will be different), so this will give an idea of the error.

4. Where are the waves in your phase diagram? To pinpoint these, a good way is to show a countour plot of the variance of the number of infected

sites $\frac{\langle I^2\rangle - \langle I\rangle^2}{N}$ as a function of p_1 and p_3 (still for $p_2 = 0.5$). You should also compute this variance along a cut at fixed $p_3 = 1/2$, between $p_1 = 0.2$ and $p_1 = 0.5$ – the reduced parameter space allows longer runs (10000 sweeps or more) which you can use to get a more accurate plot.

5. Modify your program so that some fraction of agents are permanently immune to the infection (i.e., in the R state). For $p_1 = p_2 = p_3 = 0.5$, what fraction of immunity – $f_{\rm Im}$ – is required to prevent the spreading of the infection? A good way to address this question is via a plot of the average infected fraction versus $f_{\rm Im}$. It is good to add error bars (you can do it quite simply with the code you've already got!).