Scientific Programming Assessment

Overview

Your task is to write some Python code to perform a simulation of a system, run the simulation to investigate its properties, and analyse and present the results.

The system you will be simulating is a simplified model of the propagation of a forest fire. The model is not intended to be physically accurate, but instead is used as a way of exploring how model parameters can affect the dynamics of a system.

The *forest fire model* is a grid based system, which follows a simple set of rules. These kinds of systems are known as cellular automata[1], another famous example of which is *Conway's Game of Life*[2].

Part 1 – The forest fire model

The first part of the project is to implement the forest fire model using Python code.

The first thing to consider is the definition of the grid. Each cell in the grid will be in one of a set of three states:

1. Empty
2. Tree
3. On fire

An example of a grid state is shown in Figure 1.

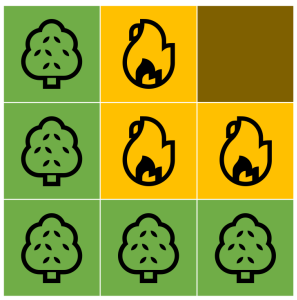

Figure 1: An example of a 3×3 grid. There are five cells with trees in, three cells that are on fire, and one empty cell.

The second thing to consider is how the grid changes over time. The model runs one "time step" at a time and each time step the current state of the grid is used to decide what the state of the grid should be in the next time step. This is run repeatedly, so the state of the grid in time step t is based on the state in the previous time step, $t-1$.

Within each time step, the algorithm works on a cell-by-cell basis and so each cell is updated in turn every time step. The value that should go into the cell in column y, row x at time t ($g^t_{x,y}$) will be based on the values in the current cell and the neighbouring cells from the previous time step ($g^{t-1}_{neigh.}$). Generally the "neighbouring cells" are defined as being those immediately above, below and to each side as in Figure 2.

The rules of the system define exactly how the state at time t should be set and are as follows:

1. A burning cell in the previous time step turns into an empty cell in this time step (burn-out)

[1] `https://en.wikipedia.org/wiki/Cellular_automaton`
[2] `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`

2. A tree will start burning in this time step if at least one neighbour (see Figure 2) in the previous time step is burning (fire spread)
3. A tree turns into fire with probability f, even if no neighbour is burning (lightning strike)
4. An empty space fills with a tree with probability p (tree growth)

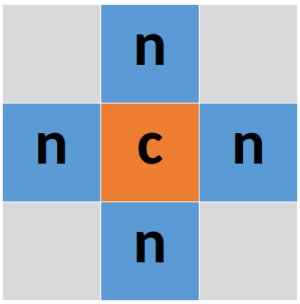

Figure 2: An example 3×3 grid showing the neighbourhood of a cell, c, as the cell itself and the 4 cells touching it, n. This is known as the *von Neumann neighbourhood*.

For each cell, the four rules are run in order until one matches. If no rule matches, the cell remains unchanged from the previous time step. Once all cells have been set for time step t, the simulation moves on to the next time step.

It is usual to start the system with each cell randomly set to be either a tree or empty. Fires will start randomly due to rule 3, which will then spread.

Some things you will need to think about and make a decision on:

- What should you set the lightning-strike rate, f, and the tree-growth rate, p, to?
- How should the initial grid be filled in?
- How big should the grid be?
- What should happen at the edges of the grid?

Part 2 – Steady-state investigation

If the probability of tree growth, p, is zero then as the fire spreads, it will burn all the trees in the grid and you will end up with a grid of empty cells. Alternatively, if the probability of lighting-strike, f, is zero then eventually all the cells will contain trees.

Between these two extreme states there exists a *steady state* where the number of trees and the number of fires both remain approximately constant. A steady-state requires that neither the number of tree or the number of fires hits 0% or 100% during the run of the simulation. It also requires that the fires that are present in any one iteration are not solely from new fires. They must be from existing fires spreading. This means, that if the lightning strike rate is, e.g. 2% that there must be more than 2% of the cells on fire to be considered a steady state. In order to be confident that the steady-state has been found, your simulation should run for at least 100 iterations, and likely needn't run for more than 1000. An example of what this might look like is shown in Figure 3.

Investigate what relationship there is between the tree-growth rate, p, and the lightning-strike rate, f, for there to be a steady state, as well as whether it depends on the size of the grid or any other factors. Include in your investigation which rules or factors are causing and supporting the steady state as well as how much it depends on the initial conditions or any other decisions you made in part 1.

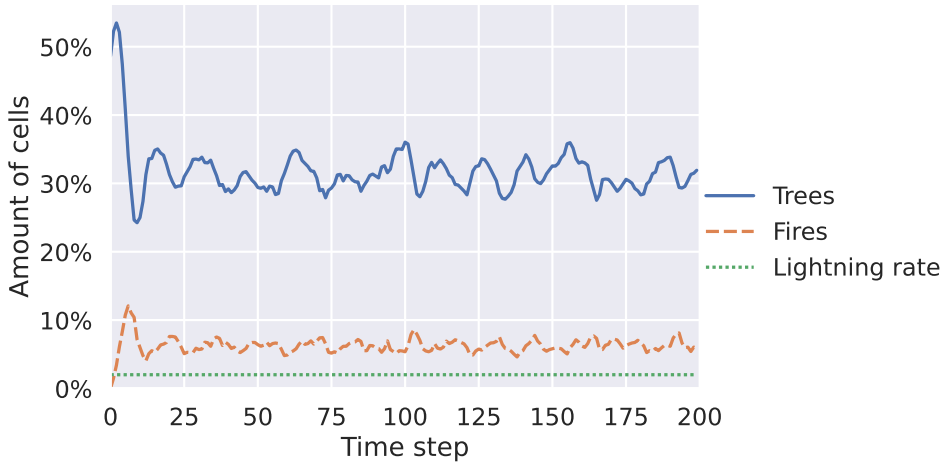

Figure 3: An example of the the number of trees and fires being in a steady state.

Part 3 – Additional model feature

Once you have the base model working, and have investigated the steady-state condition, you should either add a new rule into the model or adapt one of the existing rules. Examples of adaptations you might make are: adding in a random chance of rain which puts out fires, adding in a random chance of a tree dying without catching fire (old-age), or adding in the effect of wind. You are encouraged to make up your own rule as long as you explain what it is.

With the new rule in place, investigate how it affects the dynamics of the system and how it affects the steady-state.

Again, it is not important that the rules are truly physically accurate. You should hypothesise the effect that a new rule will have, implement that rule and then measure the result of it.

The output

The output of the project should be a single `.zip` or compressed `.tar` file containing:

- The Python code written to run the simulation, along with any tests (both as `.py` files). The code should be runnable and be able to reproduce the results you present.
- The code to analyse and visualise the output of the simulation (`.py` or `.ipynb`), along with any output or data files needed to run this code.
- A plain-text "readme" file describing how to run all the code. This should be as long as it needs to be to explain to someone unfamiliar with your code to get it running, but might be anywhere between 5 and 50 lines long depending on what you include. It should not include analysis outputs or anything which should be in the report.
- A short PDF scientific report with a description of your model (including design/model decisions you made), your approach to investigating the steady state and the results thereof, a description of your "additional model feature", the results of how it affects the results of the simulation and a conclusion.

  The report should not include any code snippets or descriptions of exactly what Python features were used to solve each problem, just a summary of your approach. It should be readable as a stand-alone document.

  It should be a maximum of four pages, including graphs, with a single-spaced, 11pt font.