

Phys 256 — Homework 4

Student Name

Instructions

- You may use AI assistants to help write, test, and debug code, but you must understand your implementation and use your own scientific judgment to assess its validity. **Do not use AI to write the report.**
- Submit a PDF report containing in-line code snippets, plots, numerical results, explanations, and descriptions of relevant test cases. Submit your Python code separately.
- In your report, we are looking for you to **exercise and demonstrate your independent scientific judgment**. Thoughtful reports with careful, critical discussion of the results will receive a higher grade than those that simply run the AI-generated code and paste in the results.
- Submit each coded problem in a separate file named `p1.py`, `p2.py`, etc. You may import functions from previous problems. Include all necessary files in one ZIP archive.
- Use NumPy and Matplotlib where appropriate. Fix random seeds or specify them so that stochastic results can be reproduced. Label all plot axes.
- Unless stated otherwise, use **open (non-periodic) boundaries** for two-dimensional lattice simulations. Updates must be simultaneous: compute the entire new state from the old state before modifying the array.

Problem 1: Gauss iterated map (6 points)

Consider the Gauss iterated map

$$x_{n+1} = F(x_n) = \exp(-\alpha x_n^2) + \beta.$$

where α and β are tunable parameters.

Write a function `gauss_map(x0, alpha, beta, N)` that returns the orbit $x_0, x_1, \dots, x_N$ as a NumPy array.

Fix $\alpha = 4.9$, choose $x_0 = 0.1$, and construct a bifurcation diagram for $-1 \leq \beta \leq 1$ for at least 100 equally spaced values of β . To do so, compute $N = 600$ iterative timesteps, discard the first 500 iterations, and plot the remaining 100 values of x_n . Identify examples of parameter values exhibiting (i) convergence to a fixed point, (ii) periodic oscillations, and (iii) irregular, apparently chaotic dynamics. Show a short time trace for each example and describe the differences.

In your report, explain why discarding the first 500 iterations is useful. Include one simple test of your implementation.

Problem 2: Conway's Game of Life (6 points)

[Conway's Game of Life](#) is a cellular automaton on a two-dimensional grid. Each site is either alive (1) or dead (0). In one time step, all sites are updated according to the number k of live cells among their eight neighbors:

- A live cell survives if $k = 2$ or $k = 3$; otherwise it dies.
- A dead cell becomes alive if $k = 3$; otherwise it stays dead.

Implement a function `life_step(A)` that returns the next generation of a two-dimensional NumPy array A . The input array must not be modified. Implement the neighbor count using **array slicing and/or Boolean operations**, rather than nested loops over the individual cells. Sites outside the array are always dead.

Test your code using

1. an initial 2×2 sub-block of live cells (which remains unchanged)
2. a horizontal row of three live cells, or *blinker* (which oscillates with period two)
3. a least one more example of *still lifes* or *oscillators* taken from the Wikipedia page [Conway's Game of Life](#)
4. plot or display at least four consecutive generations of a *glider* on a sufficiently large grid.

Explain how your tests verify the update rules.

Problem 3: Optimizing Game of Life initial conditions (6 points)

Can we choose the initial configuration of a cellular automaton to produce a desired outcome? Consider a 25×25 Game of Life grid, initially empty except for **exactly five live cells** in its central 3×3 square.

Let $A(T)$ be the configuration after $T = 10$ generations, and define the objective function

$$J(A_0) = - \sum_{i,j} A_{ij}(T).$$

We wish to *minimize* J , or equivalently maximize the number of live cells after 10 generations. There are only $\binom{9}{5} = 126$ possible starting configurations. Search them all and find an initial arrangement minimizing J . Show the optimal initial pattern and the resulting pattern after 10 generations. Report the number of live cells at $T = 10$. If more than one configuration ties for the optimum, showing any one is sufficient.

Compare your optimal pattern against at least one other five-cell starting pattern. Explain how this illustrates *optimization over initial conditions* rather than optimization over a continuous parameter.

Problem 4: Forest fires and site percolation (6 points)

Consider an $L \times L$ forest that we briefly discussed in class; in this problem you will develop this simulation yourself. Each site independently contains a tree with probability p , and is otherwise empty. Ignite *all occupied sites on the left boundary*. In each time step, burning trees ignite every unburned tree immediately above, below, left, or right of them; burning trees then become burned. Burned trees cannot ignite again. The spreading is **deterministic** once the initial forest is chosen.

Implement a function `fire_spans(A)` that takes a binary forest occupancy array `A` ($1 = \text{tree}$, $0 = \text{empty}$) and returns whether the fire eventually reaches the right boundary. Use two-dimensional arrays and open boundary conditions. Verify your implementation with $L = 50$ for a completely empty forest and a completely occupied forest.

Estimate the probability $P_{\text{span}}(p)$ that fire reaches the right edge, using at least 100 independently generated forests for each p . Plot your estimate for $0.40 \leq p \leq 0.80$, with density increments no larger than 0.02. Estimate the value $p_{1/2}$ where $P_{\text{span}}(p_{1/2}) \approx 1/2$.

Compare your result with the infinite-lattice site-percolation threshold $p_c \approx 0.5927$. Explain why our deterministic fire model is exactly equivalent to a site-connectivity problem, and why $p_{1/2}$ for a finite lattice need not equal p_c exactly.

Problem 5: Optimizing forest management (6 points)

Suppose the original forest has tree density $p_0 = 0.8$. We can remove a fraction q of its trees, selected independently at random, to reduce the chance of fire spreading across it. The remaining density is $p_0(1 - q)$.

Consider the objective function

$$J(q) = P_{\text{span}}(p_0(1 - q)) + cq, \quad 0 \leq q \leq 0.6,$$

where the first term represents fire risk and the second term is the cost of tree removal. Use $c = 0.5$.

Using your simulation from Problem 4, evaluate $J(q)$ for at least 31 equally spaced values of q , averaging over at least 300 independent forests for each point. Plot P_{span} , cq , and $J(q)$ against q , and find the value of q that minimizes your numerical estimate of J .

Compare your optimum with

$$q_c = 1 - \frac{p_c}{p_0}.$$

Why should these values be related, and why need they not be identical? Repeat the optimization with $c = 2$ and explain how the optimum changes. Why is it important to average over many independent trials of the simulation?

Problem 6: Strategic firebreak (Bonus 2 points)

Instead of removing trees at random, suppose we can choose which trees to remove after inspecting the forest. On a 50×50 realization with initial density $p_0 = 0.8$, devise an algorithm that removes the **minimal** set of occupied sites so that no occupied path remains from the left boundary to the right boundary.

Show a picture of the forest before and after removal, and report how many trees you removed. Compare your result with the number that would be removed on average by random thinning at the value q_c from Problem 5. Explain why targeted removal can be more efficient.