Phys 256 — Homework 5

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.

Problem 1: Fitting Exponential Decay (15 points)

Consider noisy data (x_i, y_i) described by an exponential model $f(x) = ae^{-\lambda x}$, with adjustable parameters $\vec{p} = (a, \lambda)$. The data is available in the dataset `decay_data.csv` on the course website. Our goal is to fit this data with the best model parameters by minimizing the sum of squared residuals,

$$\boxed{\chi^2(a, \lambda) = \sum_{i=1}^{N} \left[y_i - ae^{-\lambda x_i}\right]^2 .}$$

(a) **Implement model and objective:** Implement the exponential model `exp_model(x, p)` and the sum of squared residuals `chi_squared(f, data, p)`, where `p = np.array([a, lam])` and `data` is a $N \times 2$ Numpy array containing the datapoints.

(b) **Visualize:** Plot a contour map of $\chi^2(a, \lambda)$ over $0 \leq a \leq 5$, $0 \leq \lambda \leq 2$. Include a colorbar and estimate the minimum 'by eye.'

(c) **Gradient descent:** Implement `gradient_descent(f, p0, eta, nsteps)` using analytical or numerical gradients. Return the full trajectory $\vec{p}_n$. Starting from $\vec{p}_0 = (2, 0.3)$, use gradient descent to minimize χ^2 and overlay the trajectory on your contour plot. Report the final best-fit parameters $\vec{p}_*$ and χ^2, and plot the resulting best-fit curve on top of the noisy data. Explore several learning rates η, illustrating slow convergence,

rapid convergence, and instability. Plot $\chi^2(\vec{p}_n)$ versus iteration for three choices of η, and briefly explain the behavior.

(d) **Compare to SciPy:** Finally, compare your results to the pre-built `scipy.optimize.curve_fit` function. Compare its optimal parameters and χ^2 with gradient descent, and plot the data alongside the fitted curve.

Submission: Submit your implementations of `exp_model, chi_squared, gradient_descent` in a python file `p1.py` for auto-grading.

Problem 2: AI-Assisted Protein Folding (15 points)

Consider a chain of N beads connected by rigid links of length a, with bending angles $\vec{\theta} = (\theta_1, \ldots, \theta_{N-2})$. The total energy is

$$E(\vec{\theta}) = \kappa \sum_{i=1}^{N-2}(1 - \cos\theta_i) - \epsilon \sum_{\substack{0 \le i < j < N \\ j-i \ge 2}} \exp\left[-\frac{(r_{ij} - r_0)^2}{2\sigma^2}\right],$$

where r_{ij} is the distance between beads i and j. As in lecture, beads may overlap.
Use an AI coding assistant to help complete the following tasks:

(a) **Generate code:** Write a detailed prompt asking the AI to implement `bead_positions(angles, a)` and `energy(angles, a, kappa, epsilon, r0, sigma)` for arbitrary $N \ge 3$, where `angles` is a Numpy array containing the angles $\vec{\theta}$. In your prompt, specify the geometry, angle conventions, indexing, and expected input/output formats. Submit your prompt and the generated code.

(b) **Validate:** Write at least four independent tests, including straight-chain geometry, reflection symmetry $E(\vec{\theta}) = E(-\vec{\theta})$, and agreement with a hand-calculated energy. Test your code for $N = 3, 4, 5$, and specify which parameters $a, \kappa, r_0, \epsilon, \sigma$ you chose.

(c) **Exploring the 1D energy landscape:**

For $N = 3$ beads, the energy depends on one bending angle θ. Use golden-section search to minimize $E(\theta)$ for the default parameters $a = \kappa = r_0 = 1, \epsilon = 3, \sigma = 0.2$. Explore the results for at least 5 different initial brackets $[a, b]$.

(d) **Exploring the multi-dimensional energy landscape:** For $N = 5$ beads, the energy depends on three bending angles $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$. Use gradient descent to minimize $E(\vec{\theta})$ using the default parameters $a = \kappa = r_0 = 1, \epsilon = 3, \sigma = 0.2$. Start from at least 20 randomly chosen initial configurations $\theta_i \in [-\pi, \pi]$ and describe how the choice of initial condition affects your results. Also explore different learning rates η and discuss how this choice affects your results.

Submission: Include your AI prompts, Python code, and test results. Describe any corrections made to the AI-generated code. Submit your implementations of `bead_positions, energy` in a python file `p2.py` for auto-grading.