HW2: Protein Folding and MCMC

CS 2840 Spring 2025

Released: Thursday, March 6, 2025

Due: Thursday, April 3, 2025, **11:59pm**

Overview

All homework assignments in this course will be submitted on gradescope, and must be completed in python. For this assignment, your submission should include a PDF file with your answers to 1, 2, and 3, as well as `metropolis.py`.

1 Debiasing MCMC

1.1 Background

In this HW, you will be implementing the Metropolis-Hastings Monte Carlo method to explore the underlying distribution of protein confirmations. Proteins tend towards lower energy states, defined by functions over the atomic positions within a protein. Their equilibrium distributions often take the form:

$$\pi(x) = \frac{1}{Z} e^{-E(x)}$$

Here, raising E - the energy function - into the exponent forces lower energy states to have exponentially higher probability. Because π is a probability distribution, $\sum_i \pi(x_i) = 1$ where x_i represents a possible configuration of atomic positions. Finally, note that Z represents a normalizing constant or **partition function** that ensures that this summation sums to 1. In particular, $Z = \sum_i e^{-E(x_i)}$. Well, then shouldn't approximating the distribution be pretty easy? Not quite. In a system with n atoms, our protein has around $3n - 6$ degrees of freedom in 3D space - and insanely large domain! It would be computationally intractable to compute the true Z and find local minima/wells in our energy function analytically. This is where MCMC comes in!

1.2 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) refers to a broad class of algorithms that use a Markov chain to generate samples from a target probability distribution. Recall the formulation of Markov chains below (not a rigorous definition by any means). Note that the next step in the chain only relies on the previous step.

$$\mathbb{P}(X_{k+1} = x_{k+1} | X_k = x_k, X_{k-1} = x_{k-1}, \ldots, X_0 = x_0) = \mathbb{P}(X_{k+1} = x_{k+1} | X_k = x_k)$$

If we let a thoughtfully-constructed Markov chain go for a while, in a perfect world, we would have a sample $\langle x_0, \ldots, x_k \rangle$ for big k where $\tilde{\pi}(x)$, defined as $\tilde{\pi}(x) = \frac{1}{k} \sum_{i=0}^{k} \mathbb{1}_{x_i = x}$, is a good estimator for *the stationary distribution of the chain*, $\pi(x)$, for all x. We saw in class that only irreducible, aperiodic Markov chains have a unique stationary distribution – you may assume both of these properties throughout the problem.

When we consider Markov chains, we leverage a transition function q that dictates the probability of transitioning between any two states. If we consider a high dimensional distribution for atomic positions, like π, for example, defining an explicit function q does not really make sense (and, may not necessarily have stationary

distribution π). In MCMC, we take an initial state $X_0 = x_0$ by sampling from the initial distribution λ and define something called a proposal distribution q (*different from the transition function* q). The goal will be to use the proposal distribution to construct a Markov chain with stationary distribution π, such that a single long run of the chain will give us a good estimate of π. Throughout this problem, we will go through an example of how and when our Markov chain has the correct stationary distribution.

1.3 Metropolis-Hastings

One particular variant of MCMC, Metropolis-Hastings, defines a process where we sample from our underlying stationary distribution π in an iterative process. Starting from X_k, we use q to "propose" a next step $\tilde{X}_k$. Different from a Markov chain, however, we do not accept the proposal with probability one (although, the resulting machine is still Markov). Instead, we do the following:

$$\mathbb{P}(X_{k+1} = j | X_k = i) = Q_{ij}\mathsf{min}(1, \frac{\pi(j)}{\pi(i)}) = Q_{ij}\mathsf{min}(1, \frac{\frac{1}{Z}e^{-E(j)}}{\frac{1}{Z}e^{-E(i)}}) = Q_{ij}\mathsf{min}(1, \frac{e^{-E(j)}}{e^{-E(i)}})$$

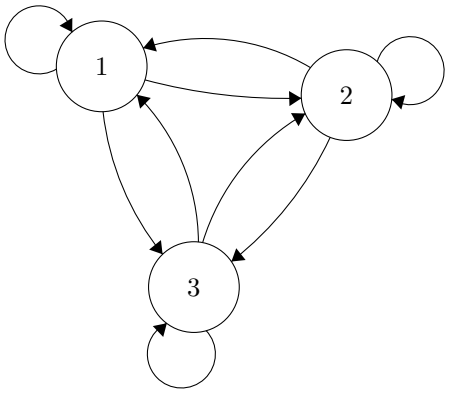
Here, Q_{ij} represents the proposal distribution, the likelihood of proposing state j when at state i ($Q_{ij} = \mathbb{P}(\tilde{X}_k = j | X_k = i)$). Using Q, generating the transition probability matrix (using the above equation) is simple. One super nice feature of this is we no longer need to compute the partition function! The transition probability is solely based on the energy ratio. This is the power of Metropolis-Hastings. ***(Note: This is not the correct formulation of Metropolis-Hastings - we will patch it up in Problem 2).***

1.4 Problem 1a

Part One

We can treat atomic arrangements of proteins as states of our Markov process, each with their own energy and probability defined by the stationary distribution π. In this question, let us simplify our state space to $S = \{1, 2, 3\}$. For the purpose of understanding Metropolis-Hastings, let us say we also know the true $\pi = \big[0.3, 0.4, 0.3\big]$. With the following proposal matrix Q, **provide the transition matrix P for our state space** using the above formulation (it should apply to the below graph). Make sure to show your work. Note that if the chain does not accept a proposed move, it stays in the same state.

$$Q = \begin{bmatrix} 0 & 0.2 & 0.8 \\ 0.1 & 0 & 0.9 \\ 0.5 & 0.5 & 0 \end{bmatrix}, \quad P = ???$$



Part Two

Now that we have a transition probability matrix P, let us analyze it. As we learned in class, we can use P to determine what stationary distribution $\tilde{\pi}$ our Markov process will converge to over many steps by solving $\tilde{\pi} = \tilde{\pi}P$. Using P, **derive the $\tilde{\pi}$ that our Metropolis-Hastings algorithm will converge to**. (hint: to do this, you might find it helpful to use a system of equations solver).

Do you notice anything suspicious in our resulting $\tilde{\pi}$? Do you think our proposal function is unbiased? Explicitly explain why this process resulted in a $\tilde{\pi}$ different from our true π, making sure to discuss the probabilities in Q and the true stationary distribution π.

1.5 Correcting our Metropolis-Hastings Formulation

It turns out we had a hidden bias in our proposal function because it was *asymmetric*. In other words $Q_{ij} \neq Q_{ji}$. It turns out that the actual formulation of Metropolis-Hastings accounts for this as follows:

$$\mathbb{P}(X_{k+1} = j | X_k = i) = Q_{ij} \mathsf{min}(1, \frac{\pi(j)Q_{ji}}{\pi(i)Q_{ij}})$$

Intuitively, we can understand this as accounting for the bias in proposing j when at state i.

1.6 Problem 1b

Using this corrected process, solve for P again, and then derive the resulting $\tilde{\pi}$. What do you notice? Make sure to show your work.

2 Stationary Problems

In this problem, we'll see what happens when we question our two major assumptions in 1.: that our Markov chain is irreducible and aperiodic.

a) A Markov chain is irreducible when all states can be reached by some pathway with nonzero probability from all other states. In the case that this is not true, we call the each set of states that are all reachable from each other a "communicating class". A communicating class is closed if there is no escape from it at all; a communicating class is open if it is possible to transition from that class to another communicating class (but not back, otherwise we would be able to combine both classes into a single communicating class). Prove that all Markov chains have at least one closed communicating class by contradiction. It might be helpful to think of the set of communicating classes as its own directed graph.

b) Consider a Markov chain with two closed communicating classes and a transition matrix:

$$\begin{bmatrix} 0.6 & 0.4 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 0.8 & 0.2 \\ 0 & 0 & 0.1 & 0.9 \end{bmatrix}$$

Write an expression that describes all stationary distributions for this Markov chain. Your answer will need an extra parameter λ that ranges from 0 to 1.

c) Suppose our initial distribution is $\lambda = \big[0.2, 0.2, 0.5, 0.1\big]$. What is the expected distribution we observe over a long run of the chain? Do we actually observe this distribution?

d) A Markov chain is aperiodic when there is no "rigid cycle" in the chain, i.e. the greatest common divisor of the timesteps $t = 1, 2, \ldots$ where we can possibly see each state is 1. Consider the irreducible Markov chain with transition matrix:

$$\begin{bmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{bmatrix}$$

What is the stationary distribution π of this chain?

e) Note that one consequence of formulating stationary distributions as $\pi P = \pi$ is that $\lim_{k\to\infty} \pi P^k = \pi$ (one of the reasons we call it stationary!). Any transition matrix can be written in terms of its eigenvalues and eigenvectors by:

$$P = \sum_{i=1}^{n} \lambda_i \phi_i \psi_i^T$$

Where ϕ_i are the right eigenvectors of P, i.e. $P\phi_i = \lambda_i \phi_i$, and ψ_i are the left eigenvectors of P, i.e. $\psi_i P = \lambda_i \psi_i$ (do not worry about the eigenvector notation, just there for completeness). It turns out that, in general, the eigenvalues of a transition matrix have magnitude less than or equal to 1. Compute the eigenvalues of our current transition matrix (feel free to use an online eigenvalue calculator). Does $\lim_{k\to\infty} P^k$ exist? What does this say about the convergence of this Markov chain to a particular distribution (π)? What would you guess one of the eigenvalues of periodic Markov chains is, in general?

f) Discuss how near-reducibility (only low probability paths connecting what would otherwise be separate classes) and near-periodicity (high likelihood for the Markov chain to cycle) of a Metropolis-Hastings Markov Chain on a protein might manifest. How would each affect our sampling of the equilibrium distribution? What would we be forced to do to get a good sample of the equilibrium distribution in either case?

3 MCMC for Proteins!

In this problem, we'll spend some time understanding MCMC in the context of protein folding. Of course, some of the notions we use here are unrealistic (for the purpose of simplifying your programming task), which we'll mention as we go.

One small aside: a PDB file is one of the few standardized formats for encoding protein structure information. The content we care about for this assignment is the list of atoms and their coordinates, which look like:

```
1  ATOM      7  CA  TYR A   2      12.658   5.808  15.732  1.00  0.07           C
```

Where the three floats with three decimals are the x, y, and z positions of the atom in space. We've implemented the parsing of these files for you, but its a common file format for storing 3D protein structures (*so, if you want to explore this field on e.g. a final project, read* `1xy2.pdb` *for a short example*).

3.1 Implementation

Your first task will be to download `metropolis.py` and the other support files on the course website. The main file is a nice stencil for setting up your very own Metropolis-Hastings Monte Carlo (MHMC) simulator for protein folding. We've provided you a few utilities and filled in a couple of necessary functions:

1. `__init__`: This function takes in a few parameters we'll think of as global to each simulation instance - the PDB file to apply MHMC to, the temperature, the variance of each random move, and some other hyperparameters. The only real content besides setting instance variables is reading in the PDB file to get initial coordinates and helpful data for printing out PDB files with updated coordinates.
2. `_read_pdb`: Reads and PDB file and returns relevant data. You should not have to use this function, given we only anticipate it to be necessary in the already-implemented constructor.
3. `_calculate_distance_matrix`: Calculate the matrix D from a set of coordinates, where D_{ij} is the euclidean distance between atom i and atom j.
4. `_save_trajectory`: Output a trajectory (or a single structure) into a PDB file, given an output location and a list of coordinate-energy pairs. This should be useful for examining specific structures, like the minimum energy structure over a full MHMC trajectory.
5. `analyze_results`: Function that takes in an MHMC trajectory (e.g., from `run_simulation`) and outputs a bunch of analysis that might be helpful to you.

After understanding this file, your next task is to set up a python environment with the requirements in the support file `requirements.txt`. If you're not sure how to do this, we actually endorse the use of generative AI to set up e.g. an anaconda environment (although, remember that generative AI is not allowed for anything you actually submit to us).

From here, you should fill in the TODOs in `metropolis.py`, including:

1. `propose_move`: As in the header of this function, you should produce a new move by choosing a random atom in the `coordinates` array, sampling a 3×1 vector from the normal distribution with mean `bias` and standard deviation `move_size`, and adding this vector to the `coordinates` array at your random index.

2. `_calculate_lj_energy`: The Lennard-Jones potential is a common way to measure non-bonded interactions between atoms (for example, in the lecture slides from 2/25). We expect you to implement it by the following formula:

$$\sum_{i>j} 4\epsilon[(\frac{\sigma}{d_{ij}})^{12} - (\frac{\sigma}{d_{ij}})^{6}]$$

   ONLY where i and j are not considered "bonded" according to the bond mask. This function has some really interesting properties that we won't get into (e.g., if you examine its Taylor series). For our purposes, just think of it as modeling van der Waals forces.

3. `_calculate_bonded_energy`: Here, we just apply what's called a harmonic potential to each pair of atoms we consider "bonded". In particular, we compute the following sum over all bonded atoms i, j:

$$\sum_{i,j} k(d_{ij} - r_0)^2$$

   Where k is a spring constant and r_0 is the equilibrium bond length. We assume that all bonds have the same length in our version, as you'll see in the parameters of the function (maybe not the greatest assumption!).

4. `calculate_energy`: In this function, we compute the set of bonded atoms according to some cutoff distance (defined in the function header), compute the bonded and non-bonded energies according to the helper functions we just implemented, and return the correct result given `self.energy_mode` $\in$ {both, bonded, nonbonded}.

5. `metropolis_step`: Here, we perform a step according by proposing a new state from $Q =$ `propose_move`, computing the acceptance probability in a biased or unbiased way (depending on the value of `proposal_corr`), and randomly deciding whether to accept or not. In this case, $\pi(x) \propto e^{-\frac{E(x)}{kT}}$.

6. `run_simulation`: In this function, using the initial coordinates from the PDB file (already stored for you), we run MHMC using metropolis_step. Remember to only return a list of accepted states as the trajectory!

One thing you may notice here is that it is not so trivial to apply the Metropolis-Hastings correction to our proposal function (i.e. the normal distribution with specified mean μ and variance σ^2). Use the following property of the normal distribution to implement the correction $\frac{q(x|x')}{q(x'|x)}$:

$$q(x'|x) \propto e^{-\frac{\|x'-x-\mu\|^2}{2\sigma^2}}$$

Once you have completed this section, you should be able to pass the autograder, which has detailed unit tests for the first five of the functions listed above. Note that we cannot test that you have done everything perfectly; i.e., the way you implement the proposal correction. *We will check for this manually.* You should also be able to get a full trajectory by calling `run_simulation` on an object set up like:

```
    sim = MetropolisSimulation(
    input_pdb=input_pdb,
    temperature=500.0,
    max_steps=2000,
    output_prefix="outs/",
    save_frequency=save_freq,
    energy_mode="both"
    )
```

3.2 Testing on a Real Protein

Oxytocin is a neuropeptide that plays an important role in regulating social behavior, emotional processing, and stress responses in humans. Disregulation of oxytocin signalling can lead to a range of mental health conditions. As a result, we might naturally be curious about its folding dynamics in this course!

In this section, answer the following questions using the support file `pdb/1xy2.pdb` in the context of the MHMC simulator we have set up.

a) **Temperature 1**. Begin by running MHMC on oxytocin using all the default parameters in the metropolis stencil with temperatures 300K, 1000K, and 10000K for 15000 steps, saving PDB files every 500-1000 (if you want to look at them at this link). Run the simulation without any bias in the random move generator. What are the acceptance rates for each case? Plot energy against number of accepted steps using the output trajectory and include the plots in your submission.

b) **Temperature 2**. Explain, using the mathematical form of your acceptance probability calculation, why you observe these empirical results. While you are doing so, take the limit of the acceptance probability in the uncorrected Metropolis-Hastings setting as $T \to \infty$.

c) **Bias 1**. Try running MHMC on oxytocin using all default parameters at 300K with biases of $0.1, 0.5$, and 1.0 Åin an uncorrected setting. How does the acceptance probability change? Why? Take a look at the atom positions when we bias with 1.0 in the final structure and compare them with the atom positions when we bias with 0 (from part a). What do you notice? Do you think that, in the long run, we will sample a representative set of 3D coordinates from the true distribution of oxytocin's structure?

d) **Bias 2**. Try running MHMC on oxytocin using the same parameters and biases, but in the corrected setting. What happened to the acceptance probability? Why? Did the issue with the coordinates in the final structure resolve at all?

e) **Energy 1**. Run the standard variation of our simulation (e.g. temperature 300K, unbiased, with standard parameters) under all three possibilities for `energy_mode`. Like in a), plot energy against the number of accepted steps. Which energy function is least stringent, based on these plots? How does the average magnitude of state energies (e.g. across bonded vs. non-bonded) impact acceptance probability, from a theoretical and empirical perspective? Relate the results here to our temperature results.

f) **Energy 2**. Upload your final structure in each simulation to this link and evaluate whether you think any of these energy functions are reasonable. Comment on how the simplifications we made in our bonding energy calculations impact the full and bonded energy simulation results (which simplification was most horrendous?). Under the assumption that the accuracy of an energy function scales with its runtime, do you think it is feasible to run MHMC with much more accurate energy functions, given the runtime you have experienced thus far?