

Assignment 2 – Atomic-Level Molecular Modeling

CS 279, BIOE 279, BMDS 245, BIOPHYS 279, and CME 279

Due: **Thursday, October 29, 2026 at 1:00 PM**

The goal of this assignment is to understand the biological and computational aspects of molecular energy functions and of predicting the three-dimensional structure of a folded protein.

1 Preliminaries

1. In this handout, ordinary paragraphs are background context, gray boxes are commands for you to type, and anything shown in **purple** is a task for you to do.
2. There is more software installation required for assignment 2 than assignment 1 so, again, we **highly recommend** students use the in-person LTS machines. All the necessary software is already installed on the LTS machines. Only fully remote students should use remote LTS machines.
3. If you still choose to work on your own computer, make sure you can run PyMOL 3.1 and Python 3.9. As mentioned in the [assignment 2 software guide](#), you will need to have conda (either Miniconda or Anaconda) installed as well.
4. **Please save your code from the “Exercises” listed below.** We’ll ask you to append the code you wrote for the Exercises at the end of your written submission.
5. In Part 4, you will be running Colab notebooks. Historically, students have run into issues on Safari, so we recommend alternative browsers like Chrome.

2 Force Fields and Free Energy Calculation

2.1 Conformational States

Proteins and other biomolecules are constantly changing shape as their atoms move around. Let’s define an individual conformation c of a biomolecular system as a particular arrangement of all the atoms that is specified by precise coordinates for each atom. The amount of time the system spends in any one conformation is extremely small. When we talk about protein structure prediction, we’re looking to find a set of similar conformations in which the protein will spend most of its time. We refer to a set C of similar conformations as a **conformational state**.

We know that the probability that a biomolecular system is in some conformation c is Boltzmann distributed according to $U(c)$, where $U(c)$ is the potential energy associated with conformation c . We can thus express the total probability that the biomolecular system adopts some conformation in conformational state C by summing the probabilities of adopting each conformation in C . The probability of each conformation follows the Boltzmann distribution, so the probability that the protein is in state C is the sum over the conformations in that state:

$$\begin{aligned} \Pr(c \in C) &= \frac{1}{Z} \sum_{c \in C} e^{-U(c)/(k_B \cdot T)} \\ &\propto \sum_{c \in C} e^{-U(c)/(k_B \cdot T)} \end{aligned} \quad (1)$$

k_B : Boltzmann constant

T : temperature

Z is a normalizing constant such that the summed probability across all possible conformations is 1. If we're interested in relative probabilities of conformational states, we don't need to calculate Z .

Intuitively, a conformational state is well-populated if its conformations have low energy and if there are many of them. Temperature changes the balance between these two factors.

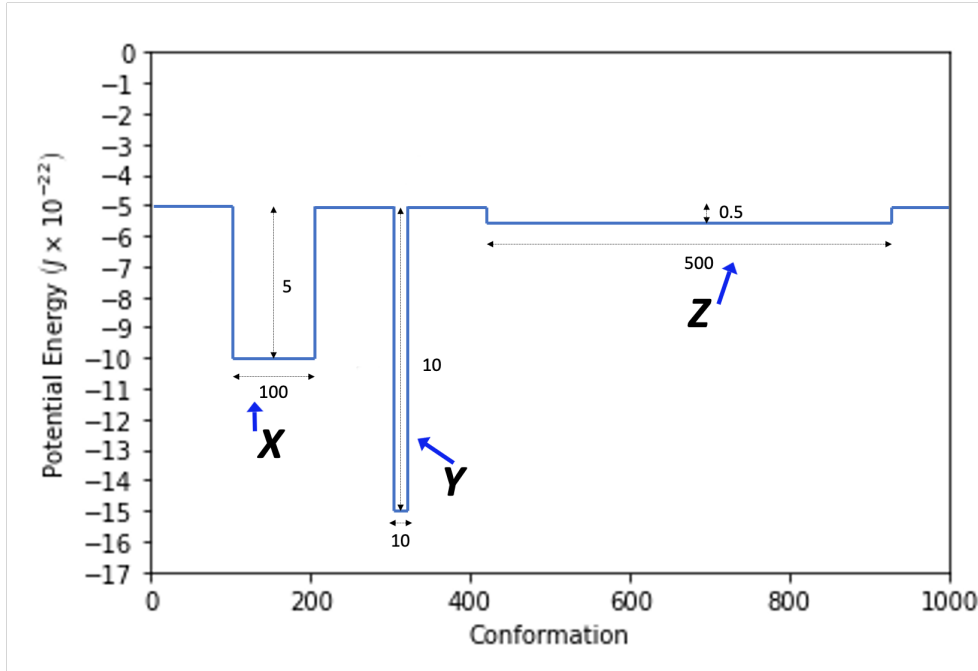


Figure 1: Potential Energy vs. Conformation, labeled.

Question 1: Consider Figure 1. Let's say this plot represents the potential energy of a given protein (and the water surrounding it) as it adopts different conformations.

- (a) Without doing any calculations, which conformational state (X, Y, or Z) would you expect to be preferred at extremely low temperatures, and which conformational state (X, Y, or Z) would you expect to be preferred at extremely high temperatures? Why? Hint: consider the number of conformations in each conformational state, and the potential energies of the conformations in each conformational state.
- (b) Now, calculate the approximate relative percentage present of each conformational state at each of the temperatures: 1 K, 100 K, and 310 K by using Equation (1). Display your results in a table as shown below. Note that 310 K (equivalent to approximately 37 °C or 98.6 °F) is normal human body temperature.

	Percentage Present of each Conformational State (%)		
Temperature (K)	X	Y	Z
1			
100			
310			

Hints: $k_B = 1.38 \times 10^{-23}$ J/K. You may ignore the time the protein spends outside of conformational states X, Y, and Z. Pay close attention to the sign of the potential energy.

This question should illustrate that the structure of a protein (or any molecule for that matter) varies with temperature.

3 Deriving a knowledge-based energy function

In this problem, we will build a “knowledge-based” energy function. Instead of approximating the potential energy of atomic interactions and then integrating these to compute free energies, this type of energy function directly estimates the free energy of a candidate structural model by incorporating knowledge of experimentally determined protein structures. Candidate structural models are deemed lower energy (more likely) if they share properties with structures of other proteins. The energy function we will implement is inspired by the paper [Statistical potential for assessment and prediction of protein structures](#) by M.-Y. Shen and A. Sali, which has accrued approximately 3000 citations.

Our knowledge-based energy function will be based on distances between pairs of residues. We will record the distances between residues in a training set of protein structures from the PDB and give favorable scores to candidate structural models that include inter-residue distances that are observed more frequently than would be expected by random chance. Each pair of residue types (e.g. serine and tryptophan) will have its distances recorded separately. In this way, the energy function will be able to distinguish which types of residues it is favorable to position closely.

Specifically, the overall energy for a candidate structural model will be a sum over energy values for

each pair of residues.

$$G = \sum_{i=1}^N \sum_{j=i+1}^N g(d_{ij}, A_i, A_j) \quad (2)$$

where N is the number of residues in the protein; d_{ij} is the distance between residues i and j ; A_i and A_j are the amino acid types of residues i and j , respectively; and g is a per-residue-pair energy function. We'll go into more detail about the per-residue-pair energy function later.

We provide starter code in `statistical_potential.py` (see the Appendix below for more detail about it). As you complete the exercises below, you should only edit the code in places between the “#####” lines. We've provided unit tests so that you can check the correctness of your code as you implement the components. To run the unit tests, open a terminal window (not the PyMOL terminal) and navigate to your Assignment 2 directory. You can then run the unit tests using the following command.

```
> python test_statistical_potential.py -v
```

To run only some of the tests, add `-k` followed by part of a test name. For example, this runs only the `test_expected_distribution` test:

```
> python test_statistical_potential.py -v -k expected_distribution
```

The test names are listed in `test_statistical_potential.py`. Until you finish all the exercises, it is normal for the tests of functions you have not yet implemented to fail.

Note: You may have to replace `python` with `python3` in order to point to the correct python version. If you are on the LTS machines, you may need to type `python3.9`.

3.1 Distribution of inter-residue distances in structures from the PDB

First, we will compute distributions (histograms) of the distances between pairs of residues in a large set of structures from the PDB. We will compute a separate distribution for distances between each pair of residue types (i.e., one for distances between aspartate and alanine residues, one for distances between aspartate and arginine residues, etc.).

In the code, we will represent these distributions as a list of counts for various distance bins. The i^{th} entry in the list will store the count for distances between i and $i + 1$ angstroms (Å). For example, the first entry will correspond to the number of occurrences between 0 and 1 Å, the second entry will correspond to the number of occurrences between 1 and 2 Å, and so on. These lists will each have 15 entries, and we will ignore residue pairs with distances over 15 Å.

Exercise 1: In “`statistical_potential.py`” implement the function “`observed_distribution`”. Your code should pass the tests prefixed with “`test_observed_distribution`” in the file “`test_statistical_potential.py`” after implementing this (the first 3 unit tests). Plot the observed distribution for aspartate and arginine by running the below command and include the plots in your submission.

```
> python statistical_potential.py ASP ARG
```

Note: While distributions are usually shown as bar charts (histograms), this assignment displays distributions as line graphs to clearly show the difference between the observed and expected distributions. The code will print “Expected distribution not implemented” and “Energy not implemented”. This is normal: you will implement these functions in the next exercises, and until then the expected distribution and energy are plotted as zero.

3.2 Accounting for the expected distribution of inter-residue distances

Given that aspartate and arginine residues typically have opposite charges and presumably interact favorably with one another, it might seem a bit surprising to see that there are many more occurrences of these residues far apart than close together. But is this really unexpected?

We will now consider how many occurrences one would expect in each distance bin if all the amino acid residues were positioned randomly. If the residues were uniformly distributed in space (i.e., equally likely to be anywhere), the expected number of occurrences of residue distances between distance 0 and distance d_{ij} would be proportional to the volume of a sphere centered at a given point with radius d_{ij} . Therefore, calculating the expected number of occurrences between distances d_{ij} and $d_{ij} + 1$ is equivalent to asking how much volume is located at distances between d_{ij} and $d_{ij} + 1$ from a given point. **Note:** The expected number of occurrences between distance 0 and distance d_{ij} is proportional to the volume of a sphere with radius d_{ij} because the distances are measured in 3 dimensions (x , y , and z).

Two important notes: (1) The expected number of occurrences of distances is influenced by the overall number of each residue type in the protein. You do not need to include this in your solution. Elsewhere in the code, we will normalize the entire expected distribution so that the expected and observed counts are the same at 15 Å. (2) In the Shen and Sali paper, they consider a very complex expected count function that accounts for the fact that at distances approaching the size of the protein the expected counts decrease. Do not attempt to model this effect for the purposes of this assignment.

Exercise 2: Implement the function “`expected_distribution`”. Your code should pass all the test cases from Exercise 1 as well as the “`test_expected_distribution`” test case after implementing this (so it passes the first four of seven test cases). Hint: the volume of a sphere with radius r is $\frac{4}{3}\pi r^3$

Run the code again for aspartate and arginine. You should now see that there are more occurrences of distances in the 4 to 10 Å range than we would expect by chance.

Question 2: In the plot for aspartate and arginine, you should see that there are fewer observed than expected occurrences for very short distances (0–3 Å). We will soon see that this is the case for every pair of residue types. What physical effect causes this?
Hint: How wide is an atom?

3.3 Deriving residue-pair energies

We will now combine the observed and expected distributions we computed above into the per-residue-pair energy function, g . Specifically, the energies will be given by the negative log ratio of the two distributions. (In Python, `math.log` computes the natural logarithm).

$$g(d_{ij}, A_i, A_j) = -\ln \frac{N_{obs}(d_{ij}, A_i, A_j)}{N_{exp}(d_{ij}, A_i, A_j)} \quad (3)$$

Here $N_{obs}(d_{ij}, A_i, A_j)$ is the number of occurrences of residue types A_i and A_j at a distance of d_{ij} in the set of solved protein structures and $N_{exp}(d_{ij}, A_i, A_j)$ gives the corresponding expected count.

This way of combining the two distributions favors distances that are observed more often than expected and results in terms that have the mathematical properties of energies. Following the convention for energies, distances that are more favorable should get lower scores. Distance bins with greater observed than expected counts should also get negative scores.

Exercise 3: Implement the function “`distributions_to_energy`” in the code. Your code should now pass the “`test_distributions_to_energy`” test, in addition to all the tests from the previous exercises (so it passes the first 5 test cases).

We will now run the code for several pairs of residue types and examine the learned per-residue-pair energy function. Remember, to run the code on, for example, aspartate (ASP) and glutamate (GLU), you can use the command:

```
> python statistical_potential.py ASP GLU
```

Question 3:

- (a) Run your code for aspartate (ASP) and arginine (ARG), and then aspartate (ASP) and glutamate (GLU). What physical interaction explains the difference in energies?
- (b) Run your code for leucine (LEU) and isoleucine (ILE). What “effect” causes these residues to interact favorably?
- (c) Run your code for cysteine (CYS) and cysteine (CYS). What special property of cysteine explains the favorability of positioning cysteine residues very close to one another?

Hint: Slide 38 in "[Biomolecular structure \(including protein structure\)](#)" contains examples of amino acid properties.

Exercise 4: Compute per-residue-pair energy functions for all pairs of residue types by running

```
> python statistical_potential.py compute
```

This will plot all of the energy functions and write them to a file called “`energies.txt`” in the “`starter_code`” directory. You don’t have to submit the “`energies.txt`” file for this exercise, but “`energies.txt`” is required for the upcoming questions. Test cases are not relevant for this question.

3.4 Protein structure prediction

Now that we have code to compute per-residue-pair energies, we can sum them all to get scores for entire proteins. We will use these scores to rank candidate structural models for a set of 20 proteins. For each protein, we've provided 300 candidate models, each of which is labeled with an RMSD to an experimentally determined structure (the "native structure"). Ideally, our scoring function will assign the most favorable (lowest) score to models similar to the native structure.

Exercise 5: *Implement the "scorer" function. This function computes the energy for a single structural model. Your code should now pass all seven test cases.*

After your code passes the test cases, **use the below command to run the scoring for all of the test proteins.**

```
> python statistical_potential.py score
```

For each protein, the code will print 1) the RMSD of the structural model selected by the energy function, 2) the lowest RMSD model across all candidate models, and 3) the median RMSD of the candidate models. You should see that the energy function performs much better than the median RMSD (which is what would be expected if we picked a random structure), but that the model by no means has perfect performance. For each protein, the code will also produce a scatter plot of RMSD vs. energy for each candidate structural model.

Question 4:

- (a) *For what fraction of the proteins does the energy function predict a close to native candidate structural model, where "close to native" is defined as having an RMSD of less than 4.0 Å? Use the RMSD of the model selected by the energy function (the first of the three values printed for each protein), not the lowest or median RMSD.*
- (b) *Include the scatter plots of RMSD vs. energy in your write-up.*
- (c) *Such scatter plots are commonly included in publications describing knowledge-based energy functions. What relationship between the RMSDs and energies is needed such that one can confidently select a near-native candidate structural model based on the energy function? A simple answer is sufficient.*
- (d) *Here we are using the energy function to score candidate structural models generated with other software. Energy functions are also often used to guide generation of candidate models ("sampling"). As you saw in lecture, sampling algorithms generally permit transitions that increase the predicted energy. Based on inspection of the scatter plots for the proteins with PDB codes 1bbh and 1cid, why is this essential to arrive at a near-native structural model? **Hint:** Does the energy increase monotonically with the RMSD in these plots?*

Let's visualize a case where our energy function makes a correct prediction. **Open PyMOL and navigate to the assignment directory using the "cd" command. Load the "native" (experimentally determined) structure, a "correct" (close to native) structural model, and an incorrect structural model.**

```
PyMOL> load test-pdbs-moulder/1mdc/native.pdb
PyMOL> load test-pdbs-moulder/1mdc/model2.pdb, incorrect
PyMOL> load test-pdbs-moulder/1mdc/model4.pdb, correct
```

Then, align both of the candidate structural models to the native structure (ex: **align correct, native** in PyMOL). You should see that the “correct” model has a much lower RMSD than the “incorrect” model.

We’ve provided a PyMOL script “visualize.py” to show residue pairs that are assigned particularly favorable per-residue-pair energies by our energy function. (Note that this script depends on “energies.txt” to get the energies, so you must have computed the energies before running this script.) Running the below commands will add lines between residue pairs with highly favorable (less than -0.75) per-residue-pair energy in the structural models.

```
PyMOL> run visualize.py
PyMOL> show_energies correct
PyMOL> show_energies incorrect
```

Question 5: *A key region of the protein that is correctly predicted in the correct structural model, but incorrectly predicted in the wrong structural model, involves the residues in the vicinity of residue 18. In the correct structural model, interactions between several pairs of residues in this region are deemed to be highly favorable by the energy function. What “effect” causes these interactions to be highly favorable? What is surrounding residue 18 in the incorrect model that is likely energetically unfavorable?*

Hint: *A quick way to locate residue 18 is to use the command “select resid 18”. Aside from other amino acid residues, are there any other molecules that might play a role?*

4 Protein structure prediction with AlphaFold

Now we will do structure prediction using AlphaFold 2 and 3. Like the scoring function that we just created, AlphaFold is a knowledge-based method, in that it was trained to reproduce experimentally determined structures of proteins, as opposed to directly capturing the underlying physical laws. However, there are several fundamental differences between the knowledge-based potential we just created and AlphaFold that enabled AlphaFold’s unprecedented success.

First, AlphaFold does not just score protein conformations, but actually directly outputs 3D structures. Second, our knowledge-based potential started with a pre-specified functional form, which introduced a variety of constraints on the final scoring function, whereas AlphaFold uses a deep learning approach that can learn far more expressive functions.

Question 6: *Name one assumption that we made in constructing our knowledge-based energy function that limits its ability to represent complex scoring functions. **Note:** There are multiple possible answers, and we’re not looking for one specific answer.*

Finally, AlphaFold is able to leverage two types of widely available experimental data that can aid in structure prediction: sequences of related proteins and experimentally determined structures of related proteins. To gain insight into how AlphaFold leverages these data types, we will start by predicting the structure of flavin reductase using AlphaFold 2 with different combinations of input

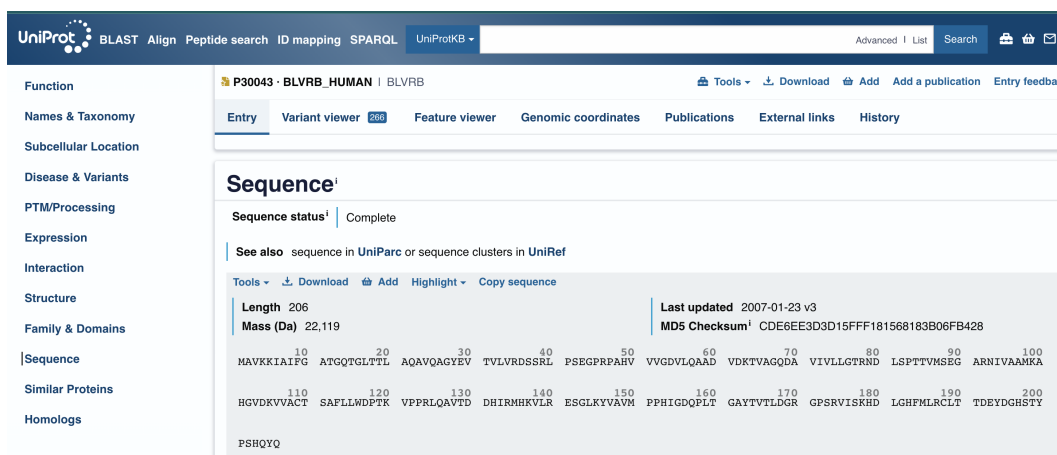
data.

AlphaFold 2: using related sequences and structures

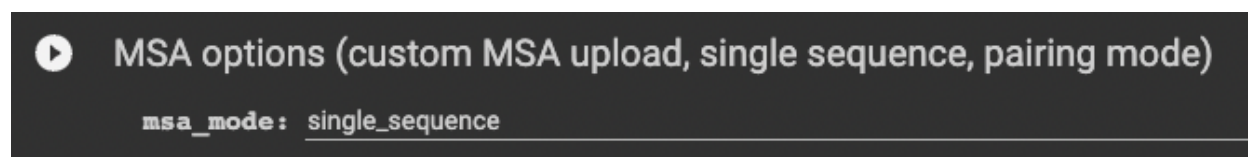
We will use a Google Colab notebook version of AlphaFold called [ColabFold](#). The notebook contains helpful usage information. As mentioned in the Preliminaries section, we recommend avoiding Safari and using a different browser (e.g. Chrome).

As a baseline, let's use AlphaFold 2 to predict the structure of flavin reductase using only its sequence. First, we will need to look up the amino acid sequence. Amino acid sequences for most characterized proteins are available from UniProt, a useful resource that organizes a large amount of information about each protein (<https://www.uniprot.org/>).

[Here](#) is the entry for flavin reductase. Get the sequence by navigating to the “Sequence” section and clicking “Copy sequence”.



Now, let's set up the ColabFold run. Paste the flavin reductase sequence into the “query_sequence” text box. Then, to disable the use of related sequences (multiple sequence alignments), set “msa_mode” to “single_sequence”. Give your run an appropriate `jobname` e.g. `flavin_sseq`.



To run the prediction in ColabFold, navigate to "Runtime" in the Colab toolbar and select "Run all". The results should automatically be downloaded as a .zip file. Unzip the file and visualize the top-ranked model using PyMOL. ColabFold predicts 5 potential models, which correspond to 5 .pdb files. We will focus on only the top-ranked model, which includes “rank_001” in the file name. Compare the predicted structure to the experimentally determined structure of flavin reductase with PDB ID: 1HDO. You can download this structure using the `fetch` command in PyMOL. Align the predicted and solved structures.

```
PyMOL> fetch 1hdo
PyMOL> load <path to your rank_001 .pdb file>, flavin_sseq
PyMOL> align flavin_sseq and polymer, 1hdo and polymer, cycles=0
```

Question 7: *What is the RMSD between the predicted and experimentally determined structures? Do you notice any differences between the predicted structure and the experimental structure?*

Note: *We are not looking for one specific answer. We expect a general comparison of the two structures.*

In addition to the sequence of the query protein, AlphaFold can make use of information about the sequences of related proteins. Before being input to the model, these sequences are aligned together to form a “multiple sequence alignment” (MSA).

Let’s make a prediction for the same protein, but this time using MSA information. **Ensure that the flavin reductase sequence is pasted into the “query_sequence” textbox. Set the “msa_mode” to mmseqs2_uniref_env (“MMSeqs2: (UniRef+Environmental”). Set the jobname to flavin_msa and run the prediction again. You can re-run all of the cells after updating the options by clicking “Run all”. Again, download the results and compare the “rank_001” model to the experimentally determined structure.**

```
PyMOL> load <path to your rank_001 .pdb file>, flavin_msa
PyMOL> align flavin_msa and polymer, 1hdo and polymer, cycles=0
```

Question 8: *What is the RMSD between the predicted and experimentally determined structures? Do you notice any differences between the predicted structure and the experimental structure?*

Note: *We are not looking for one specific answer. We expect a general comparison of the two structures.*

If you like, you can view the sequences used to make the prediction by opening the .a3m file in a text editor.

AlphaFold is also able to make use of experimentally determined structures of related proteins. AlphaFold can search the PDB for templates automatically, but here we will give it one ourselves so that we know exactly what it used. Our template is PDB entry 4R1S, a protein from petunias (the plant) that is similar in sequence to flavin reductase and carries out a similar function.

First, download the template in PyMOL:

```
PyMOL> fetch 4R1S
PyMOL> pwd
```

fetch loads the structure and also saves it as a file called 4r1s.cif. pwd prints the folder where that file was saved; you will need it in a moment.

Then, in ColabFold:

1. **query_sequence** – Paste the **flavin reductase** sequence in.
2. **msa_mode** – Set to **mmseqs2_uniref_env**, as in the previous prediction.
3. **template_mode** – Set to **custom**.
4. **jobname** – Set to **flavin_template**.

5. Click **Runtime** → **Restart session and run all**.

6. An upload button will appear under the first cell. Click it and select **4r1s.cif** from the folder that **pwd** printed. The prediction will continue once the file is uploaded.

Again, download the results and compare the “rank_001” model to the experimentally-solved structure (1HDO not 4R1S).

```
PyMOL> load <path to your rank_001 .pdb file>, flavin_template
PyMOL> align flavin_template and polymer, 1hdo and polymer, cycles=0
```

Question 9: *What is the RMSD between the predicted and experimentally determined structures? Do you notice any differences between the predicted structure and the experimental structure?*

Note: *We are not looking for one specific answer. We expect a general comparison of the two structures.*

Question 10: *Compare the template structure (4r1s) to the experimentally determined structure of flavin reductase. Based on what you see, why are the structures of homologous proteins useful for protein structure prediction? Phrase your answer in terms of the “sequence determines structure, structure determines function” paradigm.*

We note that, while either the multiple sequence information or the templates are sufficient here to give a correct prediction, there are some proteins for which only one or the other source of information is available (and in some especially challenging cases, both types of information are required to make an accurate prediction).

AlphaFold 3: proteins bound to small molecules

AlphaFold 3 can additionally model molecules other than proteins — DNA, RNA, ions, and small molecules such as drugs — together with the protein in a single prediction.

We will predict structures of proteins together with drugs, and compare the predictions against experimentally determined structures.

We will use a [Google Colab notebook](#) version of AlphaFold 3 (modified from ColabFold) that we have prepared for this course. **Open the link, then click File → Save a copy in Drive so that you have your own copy to run. Then click Runtime → Change runtime type and select a T4 GPU.** To run a prediction, fill in the "Input sequences" cell and then **Run all**. The first run installs dependencies like AlphaFold 3 and its weights, which takes several minutes; later runs skip this step.

Prediction 1: CA2 with its drug

Our first protein is human carbonic anhydrase II (CA2), an enzyme that interconverts carbon dioxide and bicarbonate. It is the target of the drug acetazolamide, which is prescribed for glaucoma and altitude sickness, and which works by binding to the enzyme’s active site. An experimentally derived structure of CA2 bound to acetazolamide is available in the PDB as **3HS4**.

Acetazolamide inhibits carbonic anhydrase II by binding to its active site, where the enzyme also holds a zinc ion that is essential for catalysis. We will give AlphaFold 3 the protein, the drug, and the zinc ion.

Find CA2's protein sequence by heading to [UniProt](#). Go to the "sequence" tab and click "copy sequence", as you did for flavin reductase.

In the "Input sequences" cell of the Colab notebook:

- **protein** – paste the copied protein sequence.
- **jobname** – set to `ca2_azm_zn`.
- **msa_mode** – set to `mmseqs2_server`.
- **seeds** – leave at 1.
- **dna, rna, and ligand_ccd** – leave empty.
- **ligand_smiles** – paste the SMILES string for acetazolamide and zinc below.

```
CC(=O)Nc1nnc(s1)S(=O)(=O)N:[Zn+2]
```

Both molecules are given as SMILES strings, which write out a molecule's atoms and bonds as a single line of text; the colon separates the two molecules. These inputs tell AlphaFold what these molecules are but not their 3D shape or where they go.

Important: give each prediction a different **jobname**. If you reuse a jobname, the new run will delete the previous run's results and you will not be able to compare them.

Then click **Runtime** → **Run all**. Then find the top-ranked model from the downloaded results. The model is a `.cif` file.

To compare your prediction with the experimental (crystal) structure, **open PyMOL and run:**

```
PyMOL> fetch 3HS4
PyMOL> remove 3HS4 and resn GOL
PyMOL> load <path to your model>.cif, pred_cplx
PyMOL> align pred_cplx and polymer, 3HS4 and polymer, cycles=0
```

Aligning on **polymer** fits only the protein, so that the agreement of the drug and zinc with the experimentally determined structure is a real test rather than something you imposed. In the prediction, acetazolamide and zinc are named **LIG_B** and **LIG_C** because we gave them as SMILES strings. In 3HS4, they are named **AZM** and **ZN**.

Question 11: *Was AlphaFold 3 successful at predicting the protein-drug complex? Include an image of your prediction vs experimentally determined structure 3HS4.*

Prediction 2: a harder molecule to predict

Although we have just seen AlphaFold correctly predict the structure of CA2 and acetazolamide, this doesn't mean that AlphaFold is always correct. Acetazolamide is a well-known drug, and the

PDB contains many structures of carbonic anhydrase II bound to it, so AlphaFold has likely seen this interaction many times during training. Let's look at a protein-drug complex whose structure was determined experimentally after AlphaFold 3 was trained.

Lemborexant is a drug approved for treating insomnia. It blocks the orexin 2 receptor (OX2R), a G protein-coupled receptor (GPCR) involved in sleep regulation. The experimental structure of lemborexant bound to OX2R (PDB 7XRR) was released in November 2022, after the cutoff date for AlphaFold 3's training data (September 2021). AlphaFold 3 was trained on structures of related complexes, but not on this one.

Copy the OX2R sequence from [UniProt](#) as you did for CA2. Then in the AlphaFold 3 Colab notebook,

- **protein** – paste the OX2R sequence.
- **jobname** – set to `ox2r_lemborexant`.
- **msa_mode** – set to `mmseqs2_server`.
- **seeds** – leave at 1.
- **dna, rna and ligand_ccd** – leave empty.
- **ligand_smiles** – paste the SMILES string for lemborexant below.

```
Cc1ncc(OC[C@]2(C[C@H]2C(=O)Nc3ccc(F)cn3)c4cccc(F)c4)c(C)n1
```

Run the notebook, download the results, and load the top-ranked model into PyMOL:

```
PyMOL> fetch 7xrr
PyMOL> load <path to your model>.cif, pred_ox2r
PyMOL> align pred_ox2r and polymer, 7xrr and polymer, cycles=0
```

As before, aligning on `polymer` fits only the protein, so where the drug ends up is a real test of the prediction. In the experimental structure, lemborexant is named `NRK`; in the prediction, it is named `LIG_B`, so we select it by its chain (`chain B`). **Zoom in on the drug and color the two copies differently:**

```
PyMOL> show sticks, (7xrr and resn NRK) or (pred_ox2r and chain B)
PyMOL> color green, 7xrr and resn NRK and elem C
PyMOL> color magenta, pred_ox2r and chain B and elem C
PyMOL> zoom 7xrr and resn NRK, 5
```

Question 12: *Include an image of the two copies of the drug. Compare how the predicted and experimentally determined copies sit in the binding pocket. Did AlphaFold correctly predict how the drug binds? Explain what you see.*

Lemborexant has two **chiral centers**: carbon atoms bonded to four different atoms (or groups of atoms). These four groups can be arranged around the carbon in two ways that are mirror images of each other, like a left and a right hand. Molecules that have the same atoms and bonds but differ in this 3D arrangement are called **stereoisomers**.

Figure 2 shows a molecule with one chiral center (black) and its mirror image. Both have the same four groups attached to the chiral carbon, but no rotation makes one identical to the other. Stereoisomers that are mirror images of each other like this are called **enantiomers**. Because lemborexant has two chiral centers, it has four possible stereoisomers: flipping the arrangement at one or both centers gives a different molecule, and only one of the four is the drug.

This distinction matters because proteins bind molecules in three-dimensional pockets, much as a right-hand glove fits only a right hand. One stereoisomer may fit the binding pocket well, while another may not fit at all.

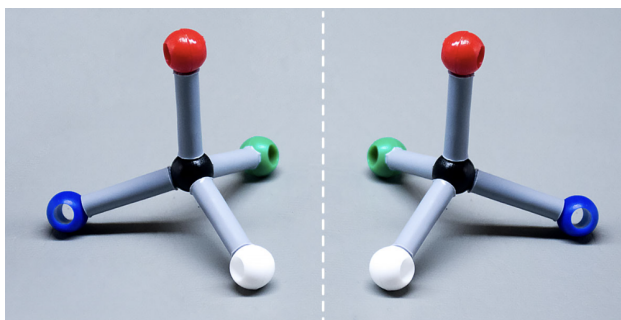


Figure 2: Molecules with chiral carbons that are non-superimposable mirror images (enantiomers)

The SMILES string we gave AlphaFold specified the arrangement of neighbors at each chiral center. The arrangement at each chiral center is correct in the experimentally determined structure. **Zoom in on the small three-membered ring in each copy of the drug and compare the groups attached to its two carbons:**

```
PyMOL> zoom 7xrr and resn NRK
PyMOL> zoom pred_ox2r and chain B
```

Question 13: *Is the predicted drug the same molecule, with the same arrangement at its chiral centers, as the drug in the experimentally determined structure?*

5 Feedback

Again, we'd love some feedback on the assignment! You will receive full credit for this portion for providing any response. Please fill out the assignment [feedback form](#) and provide the code word below. We encourage constructive criticism.

Question 14: *What is the feedback form code word?*

6 Challenge Question

This question is intentionally more challenging than the rest of the homework. It is **not required**, and you can receive full credit for the assignment without attempting it. We will never deduct

points for a challenge question submission and will award extra credit depending on the quality of your solution.

6.1 Overview

To complete this question, you will need PyMOL and PyRosetta 4. We **strongly recommend** using the LTS Macs, which already have both installed. If you choose to work on your own machine, follow the software instructions in the [Assignment 2 Software Guide](#).

In this challenge, you will use PyRosetta to fold a small protein (PDB 1PRB) starting from a fully extended chain, using the Rosetta energy function and the Monte Carlo algorithm. You will compare two ways of proposing moves: changing dihedral angles, and inserting fragments from known protein structures. The starting chain is `pdb/1prb_seq.pdb`, and the experimental structure is `pdb/1prb.pdb`.

6.2 Part A: Monte Carlo

The Metropolis Monte Carlo algorithm works as follows:

1. Start from an initial conformation c .
2. Repeat:
 - Propose a new conformation c' by making a small random move to c .
 - If $U(c') \leq U(c)$, accept the move: $c \leftarrow c'$.
 - Otherwise, accept the move with probability $e^{-\Delta U/(k_B T)}$, where $\Delta U = U(c') - U(c)$. If it is not accepted, keep c .

Moves that lower the energy are always accepted, and moves that raise it are accepted with a probability that shrinks the more the energy rises. This rule is called the Metropolis criterion. In the code, $k_B T = 1$.

Implement `acceptMove()` and `predict()` in `predictor.py`. `predict()` should also keep track of the lowest-energy conformation it finds and return it. The comments in the code list the helper functions you can use.

6.3 Part B: Two ways to propose a move

To watch the runs, first connect PyMOL to PyRosetta as described in the software guide (`run PyMOLRosettaServer.py` in PyMOL). Run all `predict.py` commands below in the Terminal, not in PyMOL.

Method 1: dihedral moves. `sampleMove()` in the `DihedralPredictor` class is written for you. It picks a random residue and changes its ϕ angle, its ψ angle, or both at once (a “shear” move: $\phi + \Delta$ and $\psi - \Delta$). The change Δ is drawn from a normal distribution with mean 0 and standard deviation of 25 degrees. Run 100,000 iterations and watch the protein move in PyMOL:

```
> python3.9 predict.py pdbs/1prb_seq.pdb dihedral100k.pdb -dihedral -pymol -100000
```

Method 2: fragment moves. Instead of changing one angle at random, each move copies the backbone shape of a short stretch of residues (a “fragment”) from a known protein structure onto the matching part of the chain. `sampleMove()` in the `FragmentPredictor` class is written for you, and fragment libraries of 9-residue and 3-residue pieces are in `frags/frag9` and `frags/frag3`. **Run 100,000 iterations with 9-residue fragments:**

```
> python3.9 predict.py pdbs/1prb_seq.pdb frag9.pdb -frag9 -pymol -100000
```

Each run saves the lowest-energy structure it found in the out folder (e.g. `out/frag9.pdb`), along with snapshots taken every 10% of the way through the run.

Tips

- Leave out `-pymol` to run much faster. You will still get the final structure, but you won’t see it move.
- Feel free to experiment: change the number of iterations (e.g. `-5000`), use 3-residue fragments (`-frag3` instead of `-frag9`), or add `-fa` to score with all atoms instead of a simplified model.
- To get energy values for your write-up, you can print the current and lowest energies inside `predict()`, for example each time it saves a snapshot.
- To compare a prediction with the experimental structure, run these in PyMOL from the `challenge_problem` folder:

```
PyMOL> load pdbs/1prb.pdb, native  
PyMOL> load out/frag9.pdb  
PyMOL> align frag9, native
```

Question 15: *Compare how well the two methods (Method 1: dihedral moves; Method 2: fragment moves) predict the structure of 1PRB. Support your answer with energy values and screenshots. A good answer has a few sentences on each method and one or two paragraphs comparing them (2–3 paragraphs in total). You do not need to submit any code. Points you may want to consider:*

- *What are the limitations of knowledge-based structure prediction methods?*
- *If our goal is to find low-energy conformations, why would we ever accept a move that raises the energy?*
- *Did either method converge to a final structure? If so, after how many iterations?*

7 Submission Instructions

Similar to Assignment 1, please submit this homework on Gradescope (<https://www.gradescope.com/>). Type up your answers in the text editor of your choice (LaTeX, Word, etc.), making sure to

clearly note the question number associated with your answer. When submitting, make sure to tag your answers with the correct questions on Gradescope.

Copy the code for your implementation of `statistical_potential.py` from all the exercises to the end of your writeup. You only need to include the functions implemented in the exercises: `observed_distribution()`, `expected_distribution()`, `distributions_to_energy()`, and `scorer()`. Then convert the entire document to a pdf. Go to Gradescope, navigate to CS279, “Assignments”, then select **Assignment 2**.

If you attempted the challenge question, include your answer and screenshots in the same write-up and tag those pages as the challenge question on Gradescope. You do not need to submit code for the challenge question.

If you have any issues submitting, please let the course staff know via Ed.

Appendix: How the starter code fits together

“statistical_potential.py” does one of three things, depending on how you run it. The flow charts below show which functions each command calls. Functions are shown in boxes, and the variables passed between them in italics.

- `python statistical_potential.py resi-type1 resi-type2` computes and plots the observed and expected distance distributions and the energies for one pair of residue types (Exercises 1–3).
- `python statistical_potential.py compute` does the same for every pair of residue types and saves the energies to “energies.txt” (Exercise 4). No coding is needed for this step.
- `python statistical_potential.py score` reads “energies.txt” and uses your “scorer” function to score every candidate structural model (Exercise 5).

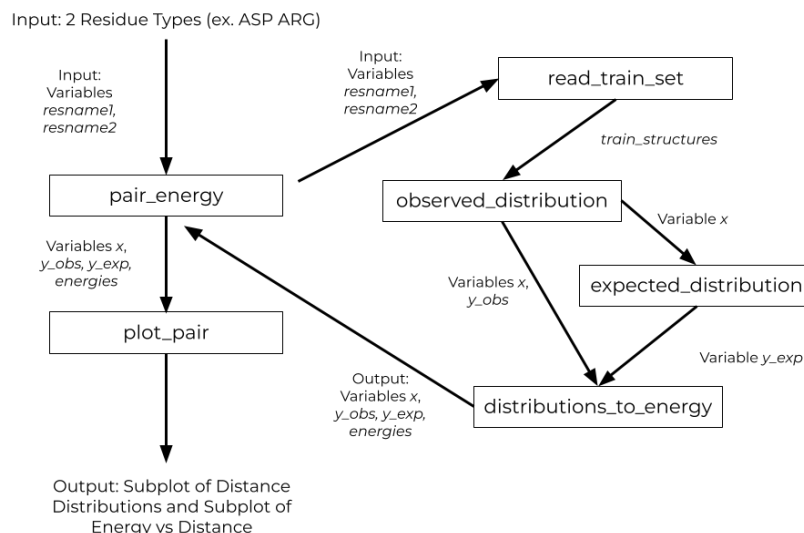


Figure 3: `python statistical_potential.py resi-type1 resi-type2`. You will code `observed_distribution` in Exercise 1, `expected_distribution` in Exercise 2, and `distributions_to_energy` in Exercise 3.

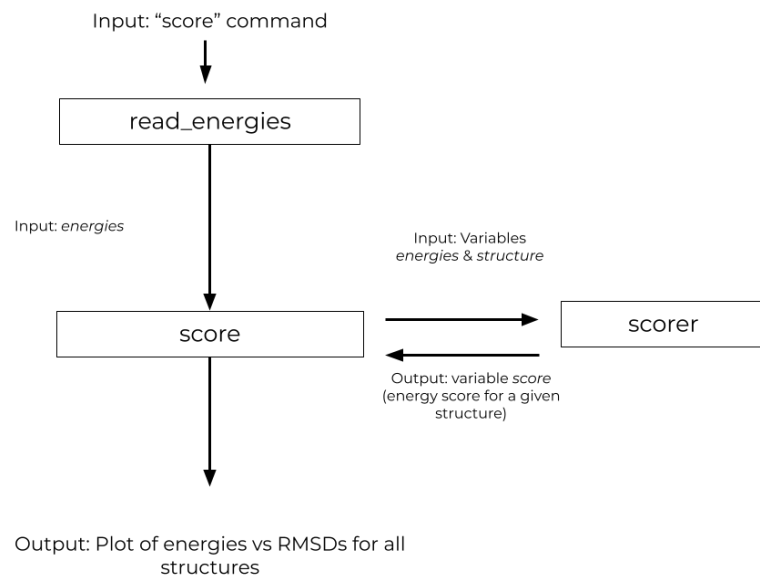


Figure 4: `python statistical_potential.py score`. You will code **scorer** in Exercise 5.