3D Data Processing in Structural Biology 76562 (Spring 2021)
Assignment 3

(due: May 11)

Last update- 3/05

Part I - optimization

1) Implement (in Python) Markov Chain Monte-Carlo (MCMC) optimization on a discrete grid. You may use some of the code provided in the next page. Follow these specifications:
   - **Configuration space -**
     A square grid (matrix) of *n x n* cells (=*configurations* or *states*). The tuple (i,j) indicates the configuration at row *i* and column *j*.
     An example for a 3x3 grid -

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2
   - **Command line -**
     "`python mcmc_discrete.py <n> <m> <k_BT>`"
     n - number of rows/columns of grid [default=5]
     m - number of iterations of MCMC optimization [default=1000]
     k_BT - the denominator for Metropolis criterion (see slides) [default=1.0]
   - **Energy -**
     The energy of state s=(i,j) is $E(s)=1.0*i+0.5*j$
     Comment (advanced, only for those interested):
     The energy units are specified in physical units called k_BT, *where* k_B is the Boltzmann constant, and *T* is the temperature. Note that the energy divided by k_BT is unitless, so the exponent of E/k_BT or $\Delta E/kBT$ in the Boltzmann distribution and Metropolis-Hastings criterion, respectively, is valid. For additional reading about the physical meaning of k_BT https://en.wikipedia.org/wiki/KT_(energy)
   - **Starting configuration -**
     Begin at a random configuration picked from the discrete uniform distribution using e.g. np.random.randint()
   - **Proposal distribution -**
     At each time step, propose moving randomly (using the discrete uniform distribution) to any neighboring configurations that is directly up, down, left, or right of the current configuration. Accept or reject as explained in class.
     Tip: you can rely on the provided function get_neighbours(c, n)
   - **Output -**
     A comma-separated list with the total number of visits to each configuration. Verify they sum to the number of iterations *m*. For example, in a 3x3 grid with *m*=500 -
     `5,80,9`

```
            3,40,1
            12,9,341
def get_cmdline_parser():
    parser = argparse.ArgumentParser(
        description='Run MCMC on a discrete n x n configuration space.')
    parser.add_argument('n',  type=int, default=5, nargs='?',
                        help='number of rows/columns of grid')
    parser.add_argument('m',  type=int, default=1000, nargs='?',
                        help='number of iterations of MCMC optimization')
    parser.add_argument('kT', type=float, default=1.0, nargs='?',
                        help='kT - the denominator for the metropolis criterion'
                         ' (Boltzmann constant times temperature)')
    return parser

def is_valid(c, n):
    ''' Return True if c is a valid 2-D coordinate on an n x n grid
    with 0-based indices '''
    if len(c)!=2:
        return False
    return (c[0]>=0 and c[1]>=0 and c[0]<n and c[1]<n)

def get_p_accept_metropolis(dE, kT, p_forward, p_backward):
    '''
    return the probability to accept the metropolis criteria
    for the specified conditions

    dE - change in energy from current to proposed configuration
    kT - the factor of Boltzmann constant (kB) and the temperature (T)
    p_forward - probability to propose a move from current to proposed configuration
    p_backward - probability to propose a move from proposed to current configuration
    '''
    p = np.exp(-dE/kT) * p_backward / p_forward
    return min(p, 1.0)

def E(c):
    assert(len(c)==2)
    return 1.0*c[0] + 0.5*c[1]

def get_neighbours(c, n):
    ''' get up/down/left/right neighbours on an n x n grid with 0-based indices'''
    assert(is_valid(c, n))
    ret_value= []
    if c[0]>0:
        ret_value.append((c[0]-1, c[1]))
    if c[0]<n-1:
        ret_value.append((c[0]+1, c[1]))
    if c[1]>0:
        ret_value.append((c[0], c[1]-1))
    if c[1]<n-1:
        ret_value.append((c[0], c[1]+1))
```

```
return ret_value
```

2) Using your MCMC implementation, run over a grid of 5x5 after *m*=10, *m*=100, and *m*=500,000 iterations, using $k_BT=1.0$. For each value of m, show a nicely formatted heat map (google "heat maps python" if needed) visualizing the relative frequency of each configuration (the total number of visits divided by *m*). For example, if configuration (1,2) was visited 21 times after 10000 iterations, its relative frequency is 0.0021.
   Comments:
   - Include a legend.
   - Each cell in the heat map should also be labeled with its relative frequencies, with precision of two digits (e.g. 0.32).
   - No need to provide code for the heatmap
   - You may optionally use the following code snippet:
     ```
     import seaborn as sbn
     import matplotlib.pyplot as plt
     plt.figure()
     m= np.sum(stats) # stats is an nxn matrix
     sbn.heatmap(C/m, annot=True, fmt=".2f", linewidths=.5)
     ```
3) What has changed in the heat maps as the number of iterations of MCMC has increased? Why? (1-2 sentences)
4) Let `C` be the output of the MCMC algorithm. For the 500,000 iterations run, show the heatmap **in log-space**, using:
   `np.max(np.log1p(C)) - np.log1p(C)`
   Unlike q. 2, do not divide by *m - this would lead to a mistake when using np.log1p*
5) What is the approximate difference between consecutive rows and columns in the previous question? Why? (1-2 sentences)
6) **Bonus (+3)**
   Why is the numpy.log1p() function typically used in such cases in practice? You may google "pseudocounts". (1-2 sentences)
   Trivia: you may read here about surprising adverse effect of using pseudocounts on statistical estimates, including in the context of COVID-19, here - https://www.biorxiv.org/content/10.1101/2020.05.19.100214v1.full
7) Run the MCMC algorithm as before, but using a k_BT command-line parameter value of 0.1, 1.0, 2.0 or 50.0 and *m*=500,000 iterations.
   - Show the new heat maps in log-space using
     `(np.max(np.log1p(C)) - np.log1p(C))`
   - Explain briefly what has changed and why, in comparison with the default k_BT value of 1.0 (2-3 sentences)
8) Let `C` be the output of the MCMC algorithm, and `kT` the value of k_BT. For the same three runs as in the previous question, please show the heatmap of:
   `kT * (np.max(np.log1p(C)) - np.log1p(C))`
9) Why did we multiply by k_BT? (1-2 sentences).
   Tip: what distribution are we approximating using MCMC with Metropolis-Hastings

10) Is there anything unexpected in the differences among consecutive rows/columns at a very low value of $k_B T$? What happened? (2-3 sentences)
Tip: When we sample randomly a finite number of times, do we always get the same mean?

11) A force vector can be computed from the energy function, by computing the gradient of the energy function, and then computing the opposite vector. For example, if the energy at configuration *(x,y)* is $E(x,y)=x^2+y^2$, then the force vector at that configuration *is (-2x, -2y)*. Assume the same energy function as in question 3, but instead of using a discrete 2D-grid configuration space, assume a continuous configuration space.
Q: What is the equation for the force vector as a function of the configuration in your MCMC implementation?

Part II - Rosetta modeling

In this part we will use the Rosetta modeling tool in order to predict the structure of a nanobody with a PDB id 6DLB.
Rosetta is a software suite that includes algorithms for computational modeling and analysis of protein structures. It includes for example de novo protein design, enzyme design, ligand docking, and structure prediction.

We will use a specific application of Rosetta - RosettaAntibody. This application is designed for modeling antibodies and has the option to model nanobodies.

The modeling in RosettaAntibody consists of two main stages:

1. Modeling the Fr, CDR1 and CDR2 regions using homology modeling tools.
2. De-novo modeling of CDR3 (H3- heavy 3) using loop modeling tools.

Resources for further reading::

- Welcome to RosettaCommons | RosettaCommons
- List of Rosetta command line options.
- RosettaAntibody3: Protocol Workflow.
- RosettaAntibody3: Modeling CDR H3 and Optimize VL-VH simultaneously
- Next-generation kinematic loop modeling and torsion-restricted sampling

Note -You will have to run stages 1 and 2 from the CSE computers (you can do it remotely).

1. Homology modeling stage (5-10 minutes running time):

Generation of a model of a nanobody from sequence in RosettaAntibody is done using homology modeling techniques. segments from known structures with similar sequences are used. The input sequence is split into several components: heavy-chain framework (FRH), CDR1, CDR2, CDR3. For each component, RosettaAntibody searches a database of known structures for the closest match by sequence (Using Blast+) and then assembles those structural segments into a model.

a) Before we run Rosetta we need to set some environment variables, run the following commands in your terminal before running Rosetta:

```
setenv ROSETTA /cs/labs/dina/tomer.cohen13/Rosetta
setenv ROSETTA3_DB $ROSETTA/main/database
setenv ROSETTA_BIN $ROSETTA/main/source/bin
setenv PATH $PATH':'$ROSETTA_BIN
setenv PATH $PATH':'/cs/labs/dina/tomer.cohen13/Blast/bin
```

b) Download the Ex3 folder from moodle, we will run all of the commands in the exercise from this folder.

c) Now, we can run RosettaAntibody from the terminal using the following command:

antibody.linuxgccrelease -n_multi_templates 1 -vhh_only -fasta 6dlb.fa -exclude_homologs true | tee grafting.log

Flags explained:

-*n_multi_templates* : number of models to generate (using different templates each time). When modeling nanobodies 1 model is enough. When modeling antibodies it is recommended to generate 10 different models.
-*vhh_only*: this flag tells rosetta we are modeling a nanobody (modeling only a heavy chain without a light chain).
-*fasta*: the fasta file (a format used for protein/DNA sequences) to use for the modeling.
-*exclude_homologs*: will not use templates with more than 80% sequence identity (because we are modeling an already solved structure). When modeling an unknown structure, remove this flag.

Understanding the output:

The expected output of this stage is:

- *grafting.log:* a file with all the outputs of Rosetta during the modeling.
- *grafting*: a folder containing: the model we created (model-0.relaxed.pdb), fasta file for each component and the alignment information of each component.

Q1: inspect the grafting.log file, what is the length and the position of each CDR? (also called H1, H2, H3).

Note- the positions here are in Rosetta numbering (from 1 and without gaps/repetitions). So they are not the same indexes as in the PDB file).

Q2: inspect alignment files (.align extension), how many hits were found for each component? (FrH, H1, H2, H3).

Note - when modeling nanobodies, Rosetta uses a 'dummy' light chain. So you can ignore any output files regarding it (frl, l1-3, orientation).

2. H3 loop modeling (1 hour of running time for each loop):

Now, we will model the CDR3 loop using de-novo modeling. The H3 loop is completely remodeled in the context of the nanobody framework using the next-generation KIC loop modeling protocol (you can read about it in the given links).

a) First, make a directory called H3_modeling in the Ex3 directory.
b) Run the following command in your terminal:

antibody_H3.linuxgccrelease @abH3.flags -s grafting/model-0.relaxed.pdb -nstruct 1 -out:file:scorefile H3_modeling_scores.fasc -out:path:pdb H3_modeling > h3_modeling-0.log

You can run this step (b) directly on the CSE computers (not remotely) or you can send it to the 'hm' cluster. To send it to the cluster:

1. Write 'cd <your Ex3 directory path>' to the file "script_example.sh".
2. Write all the environment variables defined above to the file "script_example.sh".
3. Write the 'antibody_H3.linuxgccrelease' command below to the file "script_example.sh".
4. Type 'ssh hm' in your terminal, it will ask you for your CSE password.
5. Send it from the terminal using 'sbatch <your script name>' command.

You can monitor your script progress using the 'squeue -u <CSE username>', for any further information about CSE slurm clusters see: Slurm - CsWiki

Important!!! - use the 'hm' cluster for running this step **only!**

Flags explained:

-*s* : the starting model (we model only the CDR3 now).
-*nstruct*: number of models to generate, each will have an independent KIC run.
-*out:file:scorefile*: the name for the energy score file.
-*out:path:pdb*: the name of the folder that will contain the finished models.
- *abH3.flags:* additional flags.

Note - the modeling of each loop is completely independent from the modeling of the rest, so one can run multiple antibody_H3 programs in parallel (just run it again and use the same output directory) . Because of limited time and resources, we decided to model in the exercise only 1 loop.

Understanding the output:

The expected output of this stage is:

- *H3_modeling-0.log* : file with all the outputs of Rosetta during the modeling.
- *H3_modeling* : folder containing the 3 new models you created.
- *H3_modeling_scores.fasc:* file containing the energy function value for each model (total_score column) and all the different parameters of the energy function.

Q3: what is the total score for your model? Add the score file in your submission.
Q4: align the model you created to the reference structure (ref.pdb) in pymol/chimeraX, color each model in a different color.
Evaluate your model, is it similar to the reference model? Which components are modeled accurately ? Which are modeles less accurately? (out of Fr, H1, H2, H3).
What is the RMSD of the model?
Add the image to your submission

Tip: To find H1,H2,H3 you can use the following link in order to renumber the models according to the Rosetta numbering. Then, you can color them using the positions you found in the previous questions.

3. Clustering the loops:

In the folder *H3_modeling_100* you can find 100 other models we generated using RosettaAntibody in the same way you did in the previous stage. We also added for you a csv file named *H3_modeling_100_scores.csv*, containing the scores and RMSDs of all the models.

Note- no need to add the code of this section to your submission.

a) **Q5:** make a scatter plot of the RMSD vs Rosetta total score (x=RMSD, y=score). Evaluate Rossetta energy function using your plot.

   Tip: What do we expect from a good energy function?

b) **Q6**: For n=[1, 5, 10]:
   Which model ('description' column) has the minimal RMSD out of the n top models according to the total score (top n models - n models with the lowest energy score)? What is the model's RMSD?

   Tip: if you are using python, you might find the function read_csv() (from pandas library) useful.

c) Calculate the RMSD between each pair of models in the folder *H3_modeling_100.* Save the data in a 100x100 matrix where:

matrix[i][j] = RMSD(model i+1, model j+1)
(the models numbering start from 1)

Tip: In order to calculate the RMSD between each pair, you can use the rmsd program provided for you. You can use it directly from the terminal using the following command:

rmsd -t <model1.pdb> <model2.pdb>

If you are using python, you can use the fast_rmsd() function provided for you (it uses the rmsd program with subprocess).

(the rmsd program works only on linux)

Tip: use the pickle library in order to save ('dump') your numpy matrix. This way, you will only need to calculate it one time. In later runs, you can use the load function in order to retrieve it.

Note- no need to submit the matrix.

d) Plot the matrix (heatmap plot) from the previous stage after clustering its columns/rows using K-means clustering with k=[1,5,10].
It doesn't matter if you use columns/rows for clustering- just let us know in the exercise what you chose.
Q7 : Add the 3 images to your submission.

Tip: if you are using python, you can use KMeans from sklearn.cluster.
Then you can call the function fit_predict() with the matrix you calculated in the previous question. This will give you a list of labels.

e) **Q8:** For n=[1, 5, 10]:
After clustering the models using K-means with k=n, Which model ('description' column) has the minimal RMSD from all the top 1 models of each cluster?
(for each cluster, find the model with the minimal score. Then, find the model out of the n models you got with the minimal RMSD).
What is the model's RMSD?

f) **Q9**: align the model with the minimal RMSD for n=10 from the previous question to ref.pdb in pymol/chimeraX (model in red, ref in yellow) . Align 2 other models, each from a different cluster out of the 9 remaining clusters.
Add the image to your submission.

g) **Q10**: which method seemed to work better for n=[1, 5, 10]?

Submission

Yout submission should include a directory named Ex3.tar with the following files:

1. **part1.pdf -** answers for Part 1 in pdf format, including all the required images.
2. **part2.pdf -** answers for Part 2 in pdf format, including all the required images.
3. **mcmc_discrete.py -** python code for Part 1.
4. **H3_modeling_scores.fasc -** your score results for Rosetta H3 modeling.
5. **H3_modeling -** the directory containing the 1 model you created.