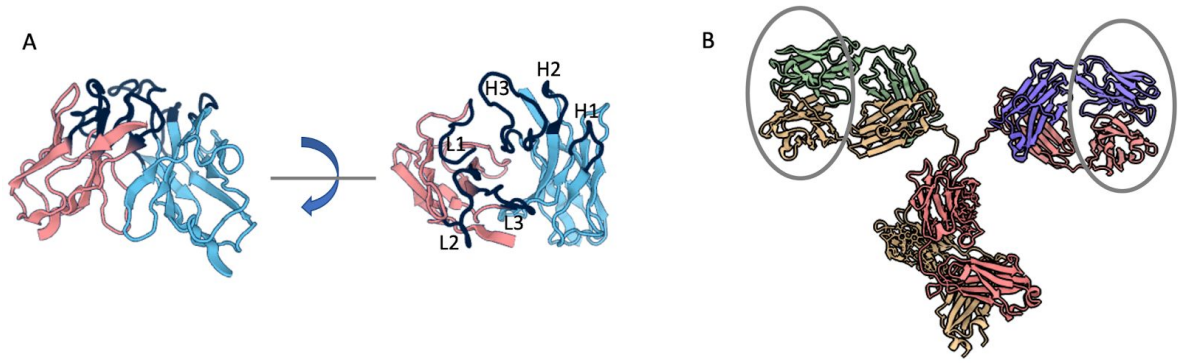
3D Data Processing in Structural Biology 76562 (Spring 2020)
Assignment 3

(due: part 1 - June 15th; parts 2 and 3 - June 21st)

Part I - docking

1) Take part in CASP-CAPRI effort - Antibody-antigen target (T1036). The sequences and target information is here: https://predictioncenter.org/casp14/target.cgi?id=24&view=all In this target we will model the complex between an antibody and Varicella-zoster virus subunit. The viral protein is a trimer (3 copies of the same protein form a complex), while the antibody consists of two chains (heavy (blue) and light (pink) - Fig 1A). We are modeling only the variable domains of the antibody (tips of the Y shape, circled in Fig 1B). Each domain contains 3 loops (shown in dark blue) called complementarity determining regions (CDRs) that interact with the antigen. These 6 loops (labeled H1-3 and L1-3) differ between the antigens and determine their specificity.



The task consists of three parts: (i) (optional) homology modeling of the Varicella-zoster virus subunit and antibody; (ii) generation of models by docking; and (iii) scoring and selection of top10 models for submission.

- Homology modeling (optional) - you can use provided models or build your own using MODELLER (https://salilab.org/modeller/tutorial/) or any tool of your choice
- Docking - you can use any of these tools. Some of them have support for antibody docking, in this case they focus the antibody interaction surface to CDRs:
  - i) ClusPro - https://cluspro.bu.edu
  - ii) ZDock - http://zdock.umassmed.edu/
  - iii) pyDock - https://life.bsc.es/pid/pydockweb
  - iv) PatchDock - http://bioinfo3d.cs.tau.ac.il/PatchDock/
  - v) Any docking tool of your choice
- Scoring - Rescore the best structures with scoring functions available from CCharppi https://life.bsc.es/pid/ccharppi

Select ten models and submit them along with the protocol (=description of steps) you used to obtain them.

Part II - optimization

2) 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
```

```python
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
```

4) 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)
```

5) What has changed in the heat maps as the number of iterations of MCMC has increased? Why? (1-2 sentences)
6) Let `C` be the output of the MCMC algorithm. For the 500,000 iterations run, show the heatmap of:
   `np.max((np.log1p(C)))-np.log1p(C))`
7) What is the approximate difference between consecutive rows and columns in the previous question? Why? (1-2 sentences)
8) **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
9) 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.
   - Explain briefly what has changed and why, in comparison with the default k_BT value of 1.0 (2-3 sentences)
10) 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))`
11) Why did we multiply by k_BT? (1-2 sentences)
12) Is there anything unexpected in the differences among consecutive rows/columns at a very low value of k_BT? What happened? (2-3 sentences)
    Tip: Consider sampling errors and rare events in your answer

13) Energy is the integral of force over distance. Therefore, a force vector can be computed from the gradient of the energy function. For technical reasons, we define the force

vector to be the opposite of the gradient 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 that instead of using a discrete grid, we had a continuous 2D configuration space - what is the equation for the force vector as a function of the configuration in your MCMC implementation?

14) In the slides, you may find the equation for iteratively updating the system's configuration during Brownian dynamics (BD):

$X_{n+1} = X_n + \frac{\Delta t}{k_B T} D f(X_n) + \sqrt{6D\Delta t} \cdot R$	
X_n	particle configuration at time step *n*
$f(X_n)$	sum of all forces acting on particle
R	random diffusion vector of magnitude ~ $Norm(0,1)$
D	diffusion coefficient
Δt	time step

Write down **pseudocode** for a BD algorithm in continuous 2D configuration space, using the force vector you computed in the previous question, and starting from a random configuration in the rectangle from (-0.5, -0.5) to (4.5, 4.5). You must adapt the algorithm to never leave the rectangle from (-0.5, -0.5) to (4.5, 4.5). No worries if you do not understand the precise meaning of all the parameters.

15) In the previous question, it is given that the diffusion coefficient *D* must be proportional to k_BT (it is fine not to know what a diffusion coefficient is). As the value of k_BT tends (שואף) to 0.0, can you think of a relation between the BD algorithm and gradient-based methods? (1-2 sentence)

Tip: read question 13 again

16) **Bonus (+20 points).**

Implement the pseudocode for the BD algorithm from the previous question. For parameters, make sure you keep *D* equal to k_BT; and set Δt to 0.1. Run for a 1,000,000 iterations and then:

- Submit reasonably documented code and how to run it
- Show a heat map for $k_BT=1.0$ and for $k_BT=0.1$ as in the MCMC questions, binning the results in a grid of 5 x 5, with cell size of 1 x 1 units.
- Show the normalized log plots as before.
- Are the results similar to what you got for MCMC (1-2 sentences)?

Part III - course overview

17) Did you go over all course slides and attended/watched all lessons?
    Yes / No
18) What is a configuration space? Provide two examples discussed in the course and describe them briefly (1-2 sentences each).
19) Write down the input and the output of the following methods of structural bioinformatics, as described in the course (1-2 short sentences each):
    - *ab initio* folding
    - Comparative modeling
    - Protein-protein docking
    - Flexible peptide docking - refinement
    - Flexible peptide docking - *ab initio*
    - Flexible peptide docking - blind
    - Molecular dynamics
    - Motion planning
20) List three protein-protein docking algorithms. For each algorithm, explain the main underlying concept (in no more than 2-3 sentences).
21) What are gradient-based methods for optimization intended for? What is their main limitation compared with some other optimization methods studied in the course? (1-2 sentences)
    Note that some recent gradient-based methods may be less sensitive to this limitation, this is out of the scope of this course
22) We learned briefly about Molecular, Langevin, and Brownian dynamics. Look at the pseudocode for the molecular dynamics algorithm in the slides, as well as the iterative update equation for Langevin and Brownian dynamics. What does the random (stochastic) component in the latter stand for? (1-2 sentences)?
23) Molecular dynamics and motion planning are alternative techniques for exploring the configuration space. Only one of them has “memory”, in the sense that it creates a map of the configuration space.
    Which one? What data structure is used to maintain the map?