Sample Final Exam Solutions

Problem 1

As stated in the lecture notes[1], in order for the volume around around the trajectory to be conserved, the Jacobian of the phase-space transformation must equal to 1. In the same lecture, we showed that the Jacobian for the transformation $(X_i, P_i) \rightarrow (X_{i+1}, P_{i+1})$ induced by Euler's algorithm is:

$$J = \left| 1 + \frac{\Delta t^2}{M} \frac{d^2 U}{dX_i^2} \right| \tag{1}$$

Recall that a general Newton equation has the form:

$$M \frac{d^2 X}{dt^2} = -\frac{dU}{dX} \tag{2}$$

Thus, for the Newton equation in this problem, $-dU/dX = C$, and hence $d^2U/dX^2 = 0$ for all X. Substituting this result into equation (1), we conclude that the Jacobian $J = 1$. Hence, the Euler algorithm conserves phase space volume for this equation.

Problem 2

Our distribution is given by the probability density function $p(x) = e^{-x}$ on the interval $[0, \infty]$. In order to sample from it, we first compute the distribution function $P(X) = \int_0^X p(x)dx$. (*Exercise:* Verify that $p(x)$ is indeed a valid probability density function on the interval $[a, b] = [0, \infty]$. That is, show that $\int_a^b p(x)dx = 1$.)

$$P(X) = \int_0^X e^{-x} dx = -e^{-x}\Big|_0^X = 1 - e^{-X} \quad \Rightarrow \quad P^{-1}(X) = -\ln(1 - X)$$

Since the values for P(X) are distributed uniformly on the interval $[0, 1]$, we can sample from our distribution as follows:

1. Generate Y uniformly at random on the interval $[0, 1]$.
2. Find an X s.t. $Y = P(X)$; that is, let $X = P^{-1}(Y)$.

Thus, to sample form the distribution $\exp[-x]$ we pick y uniformly at random from $[0, 1]$, and let $x = -\ln(1 - y)$.

[1] Handout 16, *Initial Value Solvers*

Problem 3

The random sampling procedure here is completely identical to the previous problem. As an exercise, demonstrate that to sample from this distribution, we need to pick y uniformly at random from $[0, 1]$, and set

$$x = \left(\frac{3y}{2c}\right)^{2/3}$$

Also, find the value of c so that $p(x)$ is a valid probability density function on $[0, 2]$.

Problem 4

a) This problem is very similar to the one discussed in lecture[2]. We wish to minimize the overall distance between the already-aligned structures $\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \ldots, \mathbf{r}^{(M)}$, and the new structure $\mathbf{r}$. The overall distance is simply the sum of individual distances from $\mathbf{r}$ to $\mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(M)}$. Hence, the new function to be optimized is:

$$\overline{D^2} = \sum_{n=1}^{N}\left(\sum_{m=1}^{M}\left(\mathbf{r}_n^{(m)} - \mathbf{U}\mathbf{r}_n\right)^T\left(\mathbf{r}_n^{(m)} - \mathbf{U}\mathbf{r}_n\right)\right)$$

The constraint equation remains the same: we require that $\mathbf{U^T U} = \mathbf{I}$. Using Lagrange multipliers, we can re-write this constraint as $\mathbf{\Lambda}(\mathbf{U^T U} - \mathbf{I}) = \mathbf{0}$, where $\mathbf{\Lambda}$ is the Lagrange multiplier matrix. We can rewrite this constraint in scalar form as

$$\sum_{i=1}^{3}\sum_{j=1}^{3}\Lambda_{ij}\left(\sum_{k=1}^{3}u_{ki}u_{kj} - \delta_{ij}\right) = 0,$$

where $\delta_{ij} = \left\{\begin{array}{lll} 1 & \text{if} & i = j \\ 0 & \text{if} & i \neq j \end{array}\right\}$. Thus, the new constrained optimization formula is:

$$\overline{D^2} = \sum_{n=1}^{N}\left(\sum_{m=1}^{M}\left(\mathbf{r}_n^{(m)} - \mathbf{U}\mathbf{r}_n\right)^T\left(\mathbf{r}_n^{(m)} - \mathbf{U}\mathbf{r}_n\right)\right) - \sum_{i=1}^{3}\sum_{j=1}^{3}\Lambda_{ij}\left(\sum_{k=1}^{3}u_{ki}u_{kj} - \delta_{ij}\right) \tag{3}$$

b) To develop an algorithm to solve this new optimization problem, first note that it is not much different from the one studied in lecture. The only difference is that the new formula now incorporates an additional summation over M different structures. Recall from lecture the formula for $\partial\overline{D^2}/\partial u_{ij}$ for the *original* function:

$$\frac{\partial\overline{D^2}}{\partial u_{ij}} = \sum_{k=1}^{3}u_{ik}\left(\sum_{n=1}^{N}\left(r_{ni}^{(1)}r_{nj}^{(2)}\right) + \Lambda_{kj}\right) - \sum_{n=1}^{N}r_{ni}^{(2)}r_{nj}^{(1)} = 0$$

[2]Handout 5: *More about Rotations*

Note that in our new function, the $r_{ni}r_{nj}$ term would appear an additional M times for each index n. Likewise, for every $m = 1 \dots M$, we would would get a new term $r_{ni}^{(m)} r_{nj}$, for all indices n. All other new terms that appear in equation (3) in part **a)** do not have u_{ij} as a factor: therefore, they will go to zero when we differentiate with respect to u_{ij}. Hence, the new formula for $\partial \overline{D^2} / \partial u_{ij}$ is:

$$\frac{\partial \overline{D^2}}{\partial u_{ij}} = \sum_{k=1}^{3} u_{ik} \left(\sum_{n=1}^{N} \sum_{m=1}^{M} r_{ni} r_{nj} + \Lambda_{kj} \right) - \sum_{n=1}^{N} \sum_{m=1}^{M} r_{ni}^{(m)} r_{nj} = 0$$

Finally, we construct the two matrices mentioned in the lecture notes: the symmetric matrix $\mathbf{S}$ and the non-symmetric matrix $\mathbf{R}$

$$\begin{aligned} \mathbf{S} : S_{ij} &= \sum_{n=1}^{N} \sum_{m=1}^{M} r_{ni} r_{nj} = M \sum_{n=1}^{N} r_{ni} r_{nj} \\ \mathbf{R} : R_{ij} &= \sum_{n=1}^{N} \sum_{m=1}^{M} r_{ni}^{(m)} r_{nj} \end{aligned}$$

From this point on, the solution proceeds identically to the lecture notes. We can show that $\mathbf{U}\mathbf{a}_k = \mathbf{b}_k$, for the right and left eigenvectors of $\mathbf{a}_k$ and $\mathbf{b}_k$ of $\mathbf{R^T R}$. We then compute these eigenvectors, and the eigenvalues λ_k, and construct $\mathbf{U}$ by setting $\mathbf{U} = \sum_{k=1}^{3} \mathbf{b}_k \mathbf{a}_k^T$. We check that $\det(\mathbf{U}) > 0$, to ensure that $\mathbf{U}$ is a valid rotation matrix, and if not, we negate the smallest non-zero λ_k. Finally, we use the minimum-distance formula:

$$\overline{D^2} = \sum_{n=1}^{N} \left(\sum_{m=1}^{M} \left(\mathbf{r}_n^{(m)} \right)^2 - M \mathbf{r}_n^2 \right) - 2 \sum_{k=1}^{3} \sqrt{\lambda_k}$$

Problem 5

There are multiple valid ways to design the Monte-Carlo procedure for sequence alignment. However, the key elements of a successful Monte-Carlo procedure are always the same: each step should alter one of the sequences in some way, new alignment score must be computed, acceptance criteria for the alignment must be specified, and the step must be accepted or rejected based on those criteria. The result of the Monte-Carlo procedure should be a valid sequence alignment that, we hope, is close to optimum with high probability.

The procedure we will give below will work as follows. Suppose you have two sequences, A and B, of length n and m, respectively. Without loss of generality, let $n \geq m$. We will start off our Monte-Carlo simulation from the simplest possible valid alignment of A against B. Namely, let A' and B' be the aligned "versions" of A and B. Since, $|A| = n > m = |B|$, we can set $A' = A$ and $B' = B$, and then pad B' with $(n - m)$ gaps, so that $|A'| = |B'|$.

$$\begin{array}{lccccccc} A': & a_1 & a_2 & \dots & a_m & a_{m+1} & \dots & a_n \\ B': & b_1 & b_2 & \dots & b_m & - & - & - \end{array}$$

We are now ready to specify the actual Monte-Carlo procedure. Each Monte-Carlo step consists of the following parts:

1. Select either A' or B', at random with equal probability.
2. Suppose, without loss of generality, that A' was selected in part 1. Select an index i at random from A'.
3. If $A'(i)$ contains a gap, remove this gap from $A'(i)$. If both $A'(i)$ and $B'(i)$ contain amino acids, insert a gap into A' in position i. Finally, if $A'(i)$ contains an amino acid, but $B'(i)$ contains a gap, reject this Monte-Carlo step outright (this way, we will never align a gap against a gap).
4. We now need to adjust B' so that that $|A'| = |B'|$. If we added a gap to A' in part 3, select a position j in B' at random and insert a gap there. (If $A'(j)$ contains a gap, we re-select j at random, so as not to align two gaps against each other). If we removed a gap from A' as in part 3, select a gap from B' at random, and remove it.
5. Compute the score of the new alignment. Let ΔS be the change in score as the result of the changes made during parts 1-4. If $\Delta S > 0$, accept this Monte-Carlo step (since our goal is to maximize the alignment score). Otherwise, accept the change with probability $\exp(\frac{\Delta S}{T})$, where T is the temperature.
6. Lower the temperature linearly at the end of each step.

Note that since each step makes sure that A' and B' represent a valid alignment – because $|A'| = |B'|$ and two gaps are never aligned against one another – we are guaranteed a valid alignment by the end of the simulation. An optimal alignment cannot be guaranteed, of course, but since we give high preference to mutations that raise the alignment score, we hope that, given enough time, the Monte-Carlo procedure will produce an answer that is at least close to the optimal. As in project 2, when implementing this procedure, we would need to experiment with parameters like the number of steps and starting temperature in order to improve our chances of convergence.

As an aside, let us consider how useful this method is for the sequence alignment problem. At each step of our Monte Carlo chain, we either insert or delete an amino acid from either sequence, and so must recompute the score of the entire alignment past the insertion or deletion point. Thus, each step takes $O(n)$ time. Let K be the number of steps it takes for our Monte-Carlo to converge. For different protein sequences, K will vary depending on the sequence length, the similarity of the two sequences, e.t.c, but in general we expect to take at least several thousand steps to converge. Hence, in most cases, $K \gg n$ (since most proteins are under 300 amino acids long), and the total running time of the Monte-Carlo is $O(Kn)$, which is much greater than $O(n^2)$ required by the dynamic programming alignment algorithm. Plus, dynamic programming guarantees us an optimal alignment, whereas Monte-Carlo does not. Thus, Monte-Carlo methods are not very useful for performing pair-wise global alignments of two protein sequences. Nonetheless, Monte-Carlo methods could be of use in other facets of the sequence alignment problem: particularly, in simultaneously aligning multiple sequences, which is far more difficult and costly to implement than pair-wise sequence alignment.

Problem 6

Consider the function $f(\mathbf{x}) = -\frac{\mathbf{x^T A x}}{\mathbf{x^T x}}$, where $\mathbf{A}$ is an $n \times n$ positive definite matrix, and $\mathbf{x}$ is an $n \times 1$ vector. We will show that by minimizing $f(\mathbf{x})$, we can compute the largest eigenvalue of $\mathbf{A}$. Specifically, we will demonstrate that $\min_{\mathbf{x}} f(\mathbf{x}) = -\lambda_{\max}$, where $\lambda_{\max}$ is the largest eigenvalue.

For $i = 1..n$, let λ_i be the eigenvalues of $\mathbf{A}$, and let $\mathbf{a}_i$ be the corresponding eigenvectors. Assume that the eigenvalues are sorted in decreasing order, so that $\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. By a well-known theorem in linear algebra, the eigenvectors of $\mathbf{A}$ span $\mathbb{R}^n$. This means that any vector $\mathbf{x}$ can be represented as a linear combinations of $\mathbf{A}$'s eigenvectors: that is,

$$\mathbf{x} = \sum_{i=1}^{n} w_i \mathbf{a}_i \tag{4}$$

where the w_i's are scalar coefficients.

Now, let us return to the function $f(\mathbf{x})$. Substituting equation (4) for $\mathbf{x}$, we get:

$$\begin{aligned}
f(\mathbf{x}) &= -\frac{\mathbf{x^T A x}}{\mathbf{x^T x}} \\
&= -\frac{(\sum_{i=1}^n w_i \mathbf{a}_i) \cdot (\mathbf{A} \sum_{i=1}^n w_i \mathbf{a}_i)}{(\sum_{i=1}^n w_i \mathbf{a}_i) \cdot (\sum_{i=1}^n w_i \mathbf{a}_i)} \\
&= -\frac{(\sum_{i=1}^n w_i \mathbf{a}_i) \cdot (\sum_{i=1}^n w_i \lambda_i \mathbf{a}_i)}{(\sum_{i=1}^n w_i \mathbf{a}_i) \cdot (\sum_{i=1}^n w_i \mathbf{a}_i)} \qquad \text{(Since } \mathbf{A}\mathbf{a}_i = \lambda_i \mathbf{a}_i\text{)} \\
&= -\frac{\sum_{i=1}^n \sum_{j=1}^n w_i w_j \lambda_j (\mathbf{a}_i \cdot \mathbf{a}_j)}{\sum_{i=1}^n \sum_{j=1}^n w_i w_j (\mathbf{a}_i \cdot \mathbf{a}_j)}
\end{aligned}$$

Note that minimizing $f(\mathbf{x})$ is the same as maximizing $-f(\mathbf{x})$. Since $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, we can upper-bound $-f(\mathbf{x})$ as follows:

$$-f(\mathbf{x}) = \frac{\sum\sum w_i w_j \lambda_j (\mathbf{a}_i \cdot \mathbf{a}_j)}{\sum\sum w_i w_j (\mathbf{a}_i \cdot \mathbf{a}_j)} \leq \frac{\lambda_1 \left(\sum\sum w_i w_j (\mathbf{a}_i \cdot \mathbf{a}_j)\right)}{\sum\sum w_i w_j (\mathbf{a}_i \cdot \mathbf{a}_j)} = \lambda_1$$

Hence, we have shown that for all $\mathbf{x}$, $-f(\mathbf{x}) \leq \lambda_1$, and hence $f(\mathbf{x}) \geq -\lambda_1 = -\lambda_{\max}$. To demonstrate that the lower bound for $f(\mathbf{x})$ is always attainable, let $\mathbf{x} = \mathbf{a}_1$ (that is, set w_1=1, and w_i=0 for all $i = 2 \ldots n$). Then,

$$f(\mathbf{x}) = -\frac{\mathbf{a_1^T A a_1}}{\mathbf{a_1^T a_1}} = -\frac{\lambda_1 \mathbf{a_1^T a_1}}{\mathbf{a_1^T a_1}} = -\lambda_1 = -\lambda_{\max}.$$

Thus, $\min_{\mathbf{x}} f(\mathbf{x}) = -\lambda_{\max}$. We can now compute the largest eigenvalue by minimizing $f(\mathbf{x})$ by using either the steepest descent or conjugate gradient approach.

Problem 7

As noted in lecture[3], at equilibrium probability, for a coordinate X and some change quantity Δ:

$$P(X+\Delta|X)P_{eq}(X) = P(X|X+\Delta)P_{eq}(X+\Delta),$$

or alternatively

$$\frac{P_{eq}(X+\Delta)}{P_{eq}(X)} = \frac{P(X+\Delta|X)}{P(X|X+\Delta)} \tag{5}$$

By the specified selection criteria, our Markov chain accepts each step with probability:

$$P(X_{i+1}|X_i) = \min\left[W(X_{i+1}|X), W(X_{i+1}|X)\cdot\exp\left(-\frac{E(X_{i+1})-E(X_i)}{T}\right)\right],$$

where $W(X_{i+1}|X) = \exp\left(-(X_i - X_{i+1})^2\right)$. Let $\Delta E = E(X_{i+1}) - E(X_i)$. Clearly, if $\Delta E > 0$, then $\exp(-\Delta E/T) < 1$, and so $P(X_{i+1}|X_i) = W(X_{i+1}|X)\cdot\exp(-\Delta E/T)$. On the other hand, $\Delta E \leq 0$ implies $P(X_{i+1}|X_i) = W(X_{i+1}|X)$. Thus, at equilibrium probability:

$$\frac{P(X+\Delta|X)}{P(X|X+\Delta)} = \frac{\exp\left(-\Delta^2\right)\cdot\exp\left(-\frac{E(X+\Delta)-E(X)}{T}\right)}{\exp\left(-\Delta^2\right)} = \exp\left(-\frac{E(X+\Delta)-E(X)}{T}\right)$$

The equilibrium distribution is therefore exponential with parameter T.[4]

[3] Handout 12: *Monte-Carlo Sampling*

[4] In order to solve **Problem 8**, you have to be familiar with local sequence alignment, which was not covered in class. Such type of a problem will not appear on the final (although global sequence alignment is still fair game).