BTRY 4840 / 6840 / CS 4775
Computational Genetics and Genomics

Problem Set 2

Problem 1: Convolutions of probability distributions [40 pts]

You are eager to explore more of statistics using Andrew and Dilip's protein-sequence generator with its specialized probability density function. You are especially drawn to thinking about the behavior of sums of random variables. You consider the outcome of generating three amino acids with the generator, represented by the random variables X_1, X_2, and X_3. You feel like you noticed an unusual trend in what you had assumed were arbitrarily assigned numeric values, so you want to know the distribution of the sum of the three values, $Y = X_1 + X_2 + X_3$. You can see by working out individual cases that this distribution has some interesting properties. The distribution of a sum of random variables is known as a "convolution" of the distributions of those variables. For example, the observed values might be $x_1 = 2, x_2 = 5, x_3 = 3$, and in this case $y = 10$.

(a) [6 pts] Compute $P(Y = 6)$ by simple enumeration of cases. Assume a pdf of the form given in problem set 1. That is:

$$f_X(x) = \begin{cases} \pi_1 = 1/2 & x = 1 \\ \pi_2 = 1/4 & x = 2 \\ \pi_3 = 1/8 & x = 3 \\ \pi_4 = 1/16 & x = 4 \\ \pi_5 = 1/32 & x = 5 \\ \pi_6 = 1/32 & x = 6 \\ 0 & x \notin \{1, \dots, 6\} \end{cases}$$

Let $Y_n = \sum_{i=1}^{n} X_i$ be a random variable representing the sum of n values given by the model, and let h_{Y_n} be its pdf. You know you can work out h_{Y_n} for small n by brute force, but you can see that the problem quickly becomes unwieldy as n grows larger. However, there is an important recurrence relation linking h_{Y_n} and $h_{Y_{n-1}}$.

(b) [6 pts] Using the recurrence relation, express $h_{Y_n}(y)$ in terms of $h_{Y_{n-1}}$ and f_X, for $y \in \{n, n+1, \dots, 6n-1, 6n\}$ (it is 0 for all other values of y). You do not need to plug in actual values for f_X; just express your result in terms of the π_i's. You will need to sum over possible values of the last value, x_n. Be sure to explicitly define the base case of the recurrence ($n = 1$). (Hint: consider which values of y_{n-1} and x enable you to obtain a total sum equal to y.)

(c) [10 pts] Devise a simple dynamic programming algorithm based on this recurrence relation. The input to the algorithm will be n and $\{\pi_1, \dots, \pi_6\}$, and the output will be a vector giving $h_{Y_n}(y)$ for the range of possible values of y. Structure it as an iterative, rather than a recursive, algorithm. Be sure to include all necessary initializations.

What is the algorithm's worst-case running time, expressed as a function of n using big-O notation? Implement your algorithm in Python or R. Use it to compute $h_{Y_{50}}(y)$ for $y \in \{50, 51, \ldots, 300\}$. Find the 10 highest and 10 lowest probability outcomes and print the values in two columns, with values of y followed by corresponding (probability) values of $h_{Y_{50}}(y)$.

You are quite happy with your dynamic programming algorithm. You report to Andrew and Dilip that you can compute the exact distribution of Y_n on your laptop almost instantaneously for modest-sized n. They are proud of your progress, but think you might be able to do these computations analytically for large n as well. They show you two ways to approximate h_{Y_n}.

Andrew first observes that, if one assumes the random variables $X_1, ..., X_n$ are geometrically distributed (i.e., they each have the pdf g_X defined in problem set 1), then the distribution of Y can be obtained analytically. Recall that, under this geometric model, X_i is the number of Bernoulli trials required to obtain a success. Thus, after n such series of Bernoulli trials, there must have been exactly n successes and $y_n - n$ failures, where $y_n = \sum_{i=1}^{n} x_i$. The pdf g_{Y_n} (specified below) therefore gives the probability of n successes and $y_n - n$ failures in y_n trials, such that the last trial is a success. This is our pdf of interest, under the assumption that the X_i's are geometrically distributed.

In deriving g_{Y_n}, one must account for the fact that the successes and failures could have occurred in any order, except that the last trial must have been a success. The constraint on the last trial can be accounted for by setting this one aside, and counting the number of sequences of $n-1$ successes and $y_n - n$ failures, or, equivalently, the number of binary sequences of length $(n-1) + (y_n - n) = y_n - 1$ that have exactly $n-1$ '1's. This number is the same as the number of ways of choosing $n-1$ positions for '1's in $y_n - 1$ slots, which is given by $\binom{y_n-1}{n-1}$. Therefore, the pdf g_{Y_n} is given by

$$g_{Y_n}(y) = \begin{cases} \binom{y-1}{n-1}(1-\phi)^{y-n}\phi^n & y \in \mathbb{N} \\ 0 & y \notin \mathbb{N} \end{cases}$$

(Note that the final success *does* have to be considered in the exponent of ϕ; it's just excluded in accounting for the number of possible sequences.) This distribution is known as the *negative binomial* distribution. It has mean $\mu = \frac{n(1-\phi)}{\phi}$ and variance $\sigma^2 = \frac{n(1-\phi)}{\phi^2}$. The negative binomial distribution is closely related to the binomial distribution, and is sometimes used as a more flexible alternative to the geometric distribution (the geometric distribution is the same as the negative binomial distribution with $n = 1$) or the Poisson distribution, because it has two parameters rather than one, and can take a variety of shapes (much the way the gamma distribution can for continuous data).

(Beware that the parameterization of the negative binomial pdf used here is slightly different from the one ordinarily used. The pdf is often parameterized in terms of the number of 1s and the number of 0s, instead of the number of 1s [n] and the number of trials [y]. If you make use of an off-the-shelf implementation of the pdf [as in R], you may need to adjust for this fact.)

Second, Dilip appeals to the central limit theorem. Y_n is a sum of n iid random variables and therefore, for sufficiently large n, will be approximately normally distributed, with mean $\mu_n = nE(X_i)$ and variance $\sigma_n^2 = n\text{Var}(X_i)$, where $E(X_i)$ and $\text{Var}(X_i)$ are the mean and variance, respectively, of the individual X_i's, and can be computed from f_X from problem set 1 (quoted above).

(d) [10 pts] Together in one graph, plot $h_{Y_{50}}$, as computed by dynamic programming, its negative-binomial approximation, and its normal approximation. In all cases, base your plots not on the MLE's of the parameters, but on the known parameters of the generator's true pdf (see above). For example, compute $E(X_i)$ and Var(X_i) from this pdf and assign ϕ such that $\frac{1-\phi}{\phi} = E[X_i]$. This will ensure that differences in your plots reflect real differences in the distributions, and not sampling error in your estimates. Discuss the differences in the variances of the three distributions. You will need to compute explicitly the variance of the distribution obtained by dynamic programming by considering each possible Y_n.

(e) [8 pts] Compute a one-sided p-value for an observed value of $y_n = 300$ where $n = 50$, i.e., $\sum_{w>300} h_{Y_{50}}(w)$. Notice that in this case the convolution has a trivial form and you can write down an exact expression for the p-value. Check that the output of your dynamic programming algorithm agrees with this p-value, modulo some possible numerical error from repeated multiplications of very small numbers. Now compare the exact p-value to approximate p-values based on the normal approximation and the negative binomial approximation. Which approximation is more accurate? What might account for any observed inaccuracies? (*Hints: you can compute the normal and negative binomial p-values in R using the pnorm and pnbinom functions, respectively. Use 'lower.tail=F' for best accuracy. The function pnbinom(j,k,phi,lower.tail=F) returns the probability of seeing more than j "failures" when k "successes" have been reached, with Pr(success)=phi for a single Bernoulli trial. In interpreting the differences between the approximations, think in particular about how the assumptions behind them affect the tails of the distributions.*)

Problem 2: Sequence alignment [60 pts]

Your roommate, who is not a scientist but is interested in genetics, sets up a primitive DNA sequencing lab in your backyard, takes samples from two earthworms, and attempts to sequence a gene from each one. Unfortunately, your roommate is not good at experimental biology and only manages to obtain sequences of length $n = 5$ and $m = 7$, respectively. They are:

$$\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_n) = \text{GACTT}$$
$$\mathbf{y} = (y_1, \ldots, y_j, \ldots, y_m) = \text{GGCAATC}$$

Your roommate wants to align sequences $\mathbf{x}$ and $\mathbf{y}$ in order to estimate the evolutionary divergence of the two worms and looks to you for help. You decide to use the Needleman-Wunsch global alignment algorithm described on pp. 19–21 in Durbin et al. For this simple problem, you use the following score function for matches and mismatches:

$$s(x_i, y_j) = \begin{cases} +1 & x_i = y_j \\ -1 & x_i \text{ and } y_j \text{ differ by a } \textit{transition} \\ -3 & x_i \text{ and } y_j \text{ differ by a } \textit{transversion} \end{cases}$$

Recall that a transition is a mutation of a purine to a purine (A $\leftrightarrow$ G) or a pyrimidine to a pyrimidine (C $\leftrightarrow$ T). All other mutations are transversions.

You plan to start with a *linear* gap penalty with a cost per gap of $d = 2$ (see Equation 2.4 in Durbin et al.).

(a) [10 pts] As a warm-up, and to build intuition about the problem, align the two sequences by hand. Following Figure 2.5 in Durbin et al., draw a table representing the matrix $F(i, j)$, with sequence $\mathbf{x}$ at the top and sequence $\mathbf{y}$ on the left of the table. Fill in $F(i, 0)$ and $F(0, j)$, for $0 \leq i \leq m$, $0 \leq j \leq n$ according to the base cases for the recurrence described in the text. Then fill the matrix out, from left to right and top to bottom, using the recurrence relation given in Equation 2.8. Maintain back pointers as you fill out the matrix, and once you have finished filling out the table, read off the best alignment using the traceback method described in the text. Turn in both the best alignment and the entire matrix (pencil and paper are fine!).

Now that you have worked through algorithms for sequence alignment, you feel excited and ready to do some programming. You decide to write a program to align two sequences, given an arbitrary score function and gap penalty. Unimpressed by your roommate's sequencing ability, you decide to test your program on sequences downloaded from the UCSC Genome Browser (http://genome.ucsc.edu). You find the following two sequences from an intron in the beta globin locus (available for download from the course website):

```
>human
GGGTGGGAAAATAGACCAATAGGCAGAGAGAGTCAGTGCCTATCAGAAACCCAAGAGTCTTCTCT
GTCTCCACATGCCCAGTTTCTATTGGTCTCCTTAAACCTGTCTTGTAACCTTGATA
>mouse
AAAGGGAAACATAGACAGGGGACACTCAAAGTTAGTGCCTGCTGGAAAGCAGACCTCTGTCTCCA
AGCACCCAACTTCTACTTGTGAGCTGCCTTGTAACCTGGATA
```

For your score function and gap penalty, you decide to use the default values for the BLASTZ program (Schwartz et al., *Genome Research* 13:103-107, 2003).

The score function is given by the matrix:

	A	C	G	T
A	91	−114	−31	−123
C	−114	100	−125	−31
G	−31	−125	100	−114
T	−123	−31	−114	91

where the value at row a and column b is $s_{a,b}$, the score for an alignment of an a and a b. (Note that the matrix is symmetric.)

BLASTZ uses an *affine* gap penalty (see Durbin et al., Equation 2.5), of the form:

$$\gamma(g) = -d - (g - 1)e \tag{1}$$

where $\gamma(g)$ is the penalty for a gap of length g, d is the gap-open penalty, and e is the gap-extension penalty. BLASTZ uses values of $d = 430$ and $e = 30$. However, you decide to take one step at a time and start by assuming a linear gap penalty (as above) with $d = 100$.

(These values are all much larger than the ones from our example above, but note that the optimal alignment depends only on their relative values, not their absolute values. This is because alignment is essentially the problem of deciding on *tradeoffs* between various types of mismatches and alignment gaps. In practice, large integers are used instead of smaller floating-point numbers because computers are faster at integer arithmetic than at floating-point arithmetic.)

(b) [20 pts] Write a program (in Python or R) to align two DNA sequences globally, using the Needleman-Wunsch algorithm with linear gap penalties. Use this program to find the optimal alignment of the beta globin sequences. Be sure to write code to perform trace back and print out the actual alignment. Ideally your program would read two sequences from a file in FASTA format (as shown above), would read a score matrix from a separate file, and would allow the gap penalty d to be specified as a command-line argument. However, if you're new to programming or pressed for time, you can simply define these parameters as constants at the beginning of the program. Turn in the source code for your program (with comments!), the optimal alignment, and the score of the optimal alignment. Please print the alignment so that it can easily be read, i.e., in a constant-width font and without line-wraps. Do not print the alignment vertically but rather using the same format as examples from lecture (one sequence on one line and the other sequence beneath it with bases and gap characters in each column corresponding to aligned bases/gaps). If necessary, you can separately show alignment columns $1 - k, (k+1) - 2k, (2k+1) - 3k, \ldots$, with k such that each block fills about a page-width.

(c) [20 pts] Modify your program to use affine gap penalties, as discussed on pp. 29–30 of Durbin et al., and run it on these sequences. Are the alignments different? If so, which one is better and why? As before, turn in the source code for your program, the optimal alignment, and its score.

(d) [10 pts] Experiment with 4 or 5 different values for the gap penalties, in the affine case. Try making gaps very cheap or very expensive, and try changing the proportions between the gap-open and gap-extension penalties. Show that you can obtain more or less "gappy" alignments by tinkering with these parameters. How do you think one would decide on the "right" score matrix and gap penalties, assuming that you have access to a dataset with alignments that are "correct"?