Assignment instructions: Completing this assignment requires submitting, for each problem, a written part or a `Matlab` program, and in some cases, both of these. Any written part should be handed in during class; Matlab programs should be emailed to `esam448@u.northwestern.edu`. Please comment your code to explain your implementation. Including *all* of the programs for a given assignment in a single email will be appreciated. **The collaboration policy from Assignment 1 is still in effect**.

1. **Breaking a substitution code.** Persi Diaconis related the story that one day a psychologist from the state prison system showed up at Stanford with a collection of coded messages. (See the additional online information accompanying this assignment.) It was guessed that a simple substitution code was being used, i.e., each letter in the alphabet had been replaced by a symbol. An approach was made to decode the messages using the statistics of written English. Basically, if at a point in a message one is at one letter of the alphabet, the probability of the letter that follows is *not* uniform. A large standard text was downloaded and the probability of first-order transitions was recorded, i.e., $P(y|x)$, where x is the current letter and y is the next. This is a matrix of transition probabilities. Given an encoded message and a function f that maps a permuted ordering of letters to the standard order, one can associate a likelihood with that particular decoding choice via

$$L(f) = \prod_i P(s_{i+1}|s_i)\,,$$

where the s_i are the decoded letters for that choice of f. Permuted orderings that produce a high likelihood (using the normal transition probabilities of English) are good candidates for the decrypted message. On the course website the text of Tolstoy's *War and Peace* can be found. This text has had most punctuation removed and all upper case letters mapped to lower case. Specifically, the only characters used are the ones in the Matlab string

```
string=' ,-.0123456789;?abcdefghijklmnopqrstuvwxyz'
```
,

i.e., the lower case letters, the digits, a semicolon, a period, a hyphen, a comma, a question mark and a space (the space is the *first* character in the string). Use the text of *War and Peace* to approximate the transition probabilities, $P(y|x)$. Make sure to include the space.

In addition, the website also has an encoded message. Use Metropolis sampling to decode the message. In particular:

(a) Start with a preliminary guess for the decoding permutation f. (Hint: there are good reasons for a random initial guess.)

(b) Compute the likelihood of the decoded message, $L(f)$.

(c) Change to a new decoding permutation f_* by randomly flipping two characters in the decoding function f.

(d) Compute the new decoding likelihood $L(f_*)$.

(e) Accept f_* with Metropolis-Hastings probability $\min(L(f_*)/L(f), 1)$.

(f) Repeat the loop (c)-(e) until no more changes occur or a maximum of $L(f)$ is found.

Write and submit a Matlab function

```
function [decoded,numits] = lastname_firstname_hw3_prob1(encoded,reference)

[...any statements you need...]

end
```

That takes as input the reference text from *War and Peace* and the mystery encoded text as strings, and writes as output the decoded message and how many iterations (including rejections) it took your algorithm to produce it. Note that this means that the first thing your code should do is compute the transition probabilities $P(y|x)$ from the reference text.

It is possible that sometimes the likelihood may have a local maximum and the iteration can get stuck. If your function doesn't decode the message properly on the first attempt, try again. This is why using a random initial guess may be helpful — it may produce a more likely result on a subsequent iteration. (Of course, this intended to be merely a demonstration of the utility of Markov Chain Monte Carlo sampling and not a serious code-breaking method.)

Hints: Some of the transition probabilities in English are zero, and this will make the overall likelihood $L(f)$ zero if such a transition occurs in the incompletely decoded message. To avoid this problem, replace all zero transition probabilities with a very small but finite value, e.g., 10^{-14}. In addition, to avoid the possibility of numerical underflow, work with the *logarithm* of the likelihood $L(f)$ instead.

2. The goal of this problem is to simulate the transcription (or, expression) of RNA. Synthesis of RNA is usually catalyzed by an enzyme — RNA polymerase — that copies a specific section of DNA (the coding sequence). Transcription usually begins with the binding of the enzyme to a promoter sequence upstream of the coding sequence. The binding of RNA polymerase to the promotor is random, hence the expression of RNA is random. In addition, repressors can also bind to DNA, halting expression. Thus, RNA expression can be turned on or off, as shown in Figure 1.

   We can make a simple Markov model of this process with the state diagram shown in Figure 2. There are two states for the promotor region, 'on' or 'off'. When in the 'off' state it can switch to

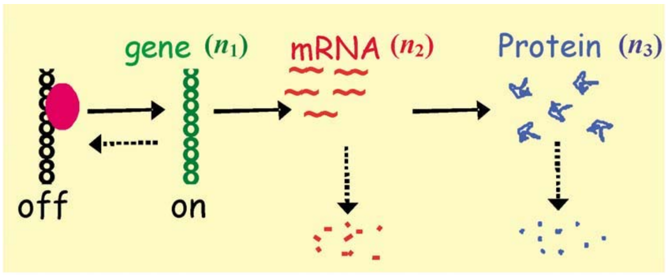

Figure 1: Simple schematic of RNA transcription, with stochastic repression. The colored ellipse is meant to represent a repressor.

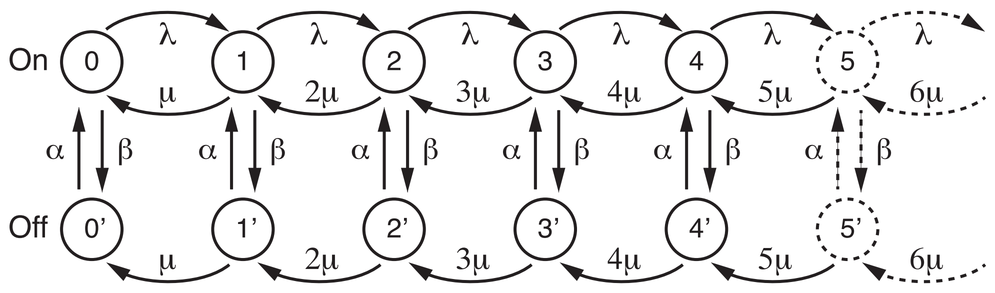

Figure 2: Simple Markov state diagram for transcription with stochastic repression.

'on' with rate α, and when 'on' it switches to the 'off' state with rate β. When the promotor region is 'on', RNA molecules are created stochastically with the rate λ. Finally, at all times RNA molecules degrade (are destroyed) at a rate μk, where k is the number of RNA molecules present.

Note that if transcription were always 'on', this would be the M/M/∞ queue discussed in the course notes. From the discussion of that problem in the notes, we know that the steady-state distribution of RNA in this case would then be Poisson with parameter λ/μ. **The goal of this problem is to determine how the stochastic off-and-on switching of the promotor region affects the process.**

Write and submit a Matlab function

```
function [probs] = lastname_firstname_hw3_prob2(...)

[...any statements you need...]

end
```

where the arguments of the function are (they are written separately because it does not fit on one line in this document)

```
... = alpha,beta,lambda,mu,num_ensemble,t_final
```

and where the output, `prob`, is a vector giving an estimate of the steady-state probabilities for different numbers of RNA molecules, i.e., `prob(1)` $= p_0$, `prob(2)` $= p_1$, ..., `prob(k+1)` $= p_k$, etc. It should hopefully be clear that `alpha` $= \alpha$, `beta` $= \beta$, `lambda` $= \lambda$ and `mu` $= \mu$. Because the goal is to find the steady-state probability distribution, one can reduce the total time duration one needs to simulate (`t_final`) by running a number of simulations simultaneously in an ensemble (so here `num_ensemble` is the number of simulations performed simultaneously).

Start each member of the ensemble with zero RNA molecules and the promotor in the 'off' state. It will therefore take some amount of time for the number of RNA molecules to grow to near the steady-state value. Explain in your write-up how to determine an estimate for the time constant T governing the approach to the steady-state using the deterministic version of this process. Please also give an estimate for the steady-state value. In your code make sure to exclude from your estimation of the steady-state probabilities any simulation results with $t < 5T$. These transient simulation results will skew your steady-state probability distribution if they are included.

Important: The steady-state values for p_k should be estimated from the average time the number of RNA molecules is equal to k in each simulation within the ensemble. The Gillespie method, of course, does not use equal-length time steps. Thus, longer time steps should count more in the average, i.e., the contribution of each Gillespie step to the overall average should be *weighted* by that particular time step. (An average across ensembles should also be performed, of course.)

In your write up please also include some plots for the specific case $\alpha = 0.1$, $\beta = 0.5$, $\lambda = 10.0$ and $\mu = 0.05$. Specifically, include a plot of the number of RNA molecules as a function of time for a *single* simulation (`num_ensemble=1`), and a plot of the steady-state probability distribution for a larger ensemble run long enough to generate a reasonable approximation to the steady state.

Finally, recall that at the beginning of the discussion of this problem it was mentioned that if transcription is always on ($\alpha \gg \beta$) the steady-state probability distribution should be Poisson. Compare the steady-state probability distribution with a Poisson distribution. This can be done in Matlab using

```
binvals=[0:length(probs)-1]';
dist_params  = fitdist(binvals, 'Poisson', 'frequency',floor(1e6*probs+0.5));
poisson_vals = pdf('Poisson',binvals,dist_params.Params(1));
```

if `probs` is a column vector. Models (not discussed here) suggest that a negative Binomial distribution should be a better fit to the simulated distribution (and to real RNA expression values), so also compare the simulated steady-state distribution to the negative Binomial distribution. (Change `'Poisson'` to `'NegativeBinomial'` and note the negative Binomial distribution has *two* parameters.) Is the negative Binomial distribution a better fit? Is the result consistent with your expectations if you decrease β to 0.005 and reduce λ to 1.75 while keeping the other parameters the same?