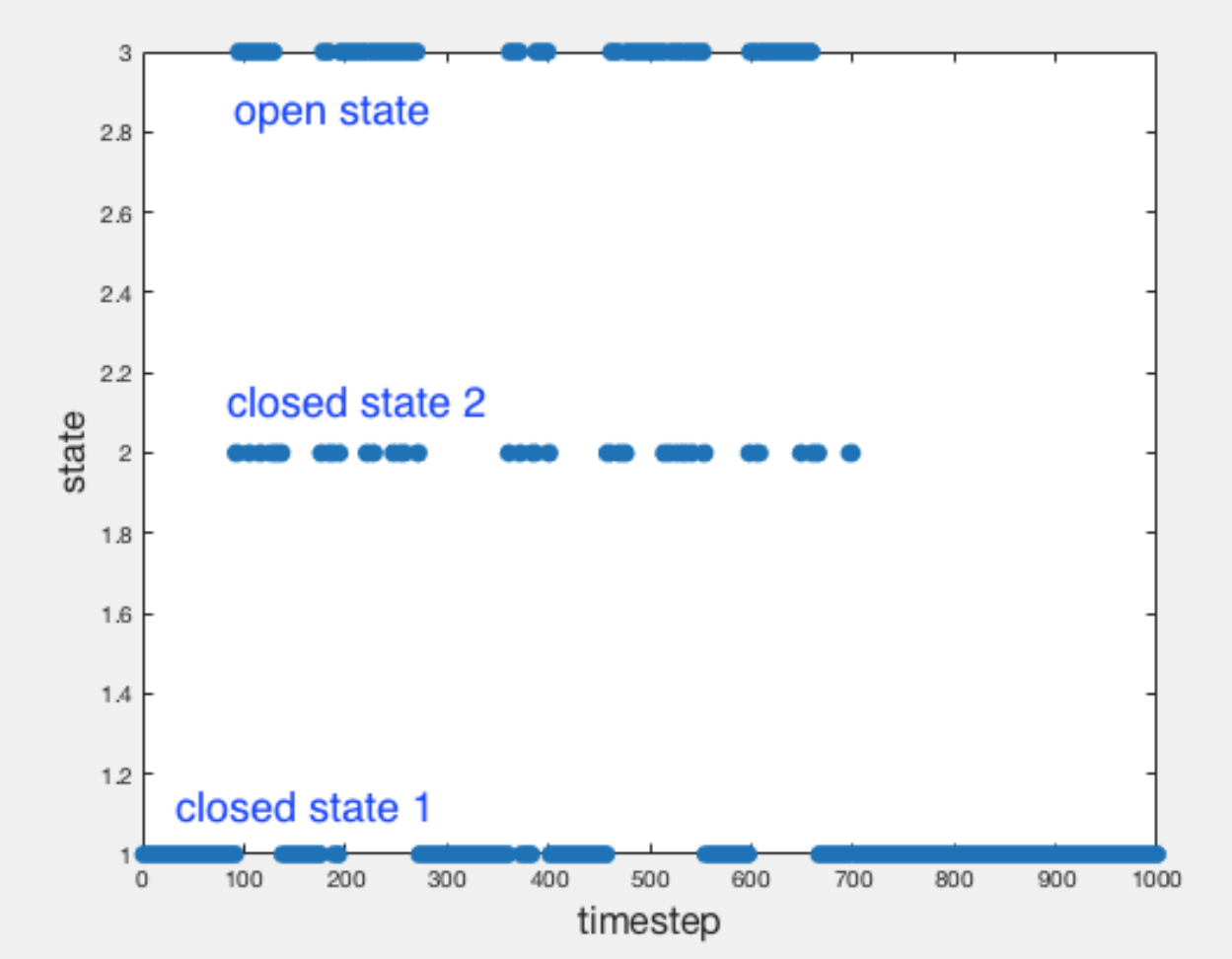
Problem 3: Simulating Markov Chains and Dwell Times

Assuming the state of our ion channel at the next timestep only depends on its current state, we can model this problem using a Markov Chain. At each timestep, we randomly generate a number uniformly between 0 and 1 using rand. Based on the previous state, s_k, we decide which new state to transition to based on this random value. If it is less than a_{1k}, transition to state 1, else if it is between a_{1k} and $a_{1k} + a_{2k}$ transition to state 2, and so on. Repeating this process, we can iteratively generate the state of our channel at every timestep.

```
% Problem 3: Simulating Markov Chains and Dwell times
% Part 2)
A = [0.98 0.1 0;
    0.02 0.7  0.05;
    0 0.2 0.95];


numsteps = 100000; %number of timesteps simulated

%list of states on this realization. xlist(k)=1 means in state 1 at
%timestep k, etc
states=zeros(1,numsteps);

%initial state
states(1)=1;

for k=1:numsteps-1

    %uniformly distributed random number - will use for transitions from
    %timestep k to current timestep k+1
    rd=rand ;

    % take the column corresponding to our current state
    probs = A(:, states(k));
    intervals = cumsum(probs);

    new_state = 1;
    while rd > intervals(new_state)
       new_state = new_state + 1;
    end

    states(k+1) = new_state;
end
```

```
%---- Plots state over time
figure
set(gca,'FontSize',18)
plot(1:numsteps,states,'.','MarkerSize',20)
xlabel('timestep','FontSize',16)
ylabel('state','FontSize',16)
```

Once we have these states, we can plot them as a function of time to get a look at how the system evolves. Using 1000 timesteps, we can see the system tends to have fairly long dwell times in state 1 (closed 1) or state 3 (open), but relatively short dwell times in state 2 (closed 2). Looking at the diagonal of the transition matrix A, this makes sense as entry A_{22}, the entry corresponding to dwelling in state 2 for a step, is significantly lower than entries A_{11} and A_{33}.



To calculate the dwell times in the closed state, it is easier to reduce state one and state two to a single closed state.

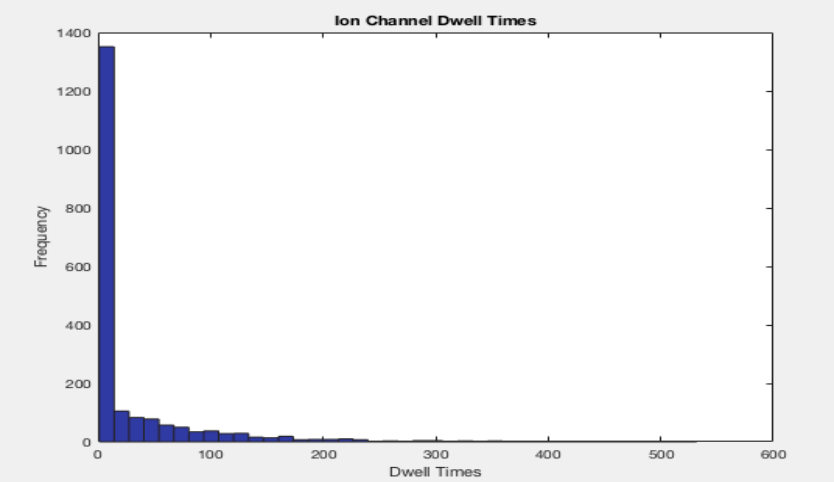
```
% Part 3)
% 1 corresponds to open, 0 corresponds to closed
reduced_states = (states == 3);
```

After doing so, we can calculate the dwell times within the closed states. I start by first finding all the indices where my state changes from open to closed or vice versa. After doing so, I know between each adjacent pair of these state change indices that I remain in a single state the entire time. The length of these dwells then is just the difference of these indices. Since I only have two states, open and closed, every time I switch states, I know I am transitioning between a period of dwell in a closed state and a period of dwell in an open state. Thus, every other dwell time is associated to a closed state. If I start closed, my odd indices contain the closed dwell times, and if I start open the even indices contain them instead. Lastly, I generate a histogram to get a visual of the distribution of these dwell times.

```
% Part 4): Calculate Dwell times
% dwell time = number of steps where you stay in same state
start_state = reduced_states(1);

% what indices do I swap states in?
swap_indices = [1, find(diff(reduced_states)), numsteps];

% number of indices between the times I swap states
dwell_times = diff(swap_indices);

% Since there are only two overall states, open/closed, every state change
% is either open -> closed or vice versa.
if start_state == 0
    closed_dwell_times = dwell_times(1:2:end);
else
    closed_dwell_times = dwell_times(2:2:end);
end
```

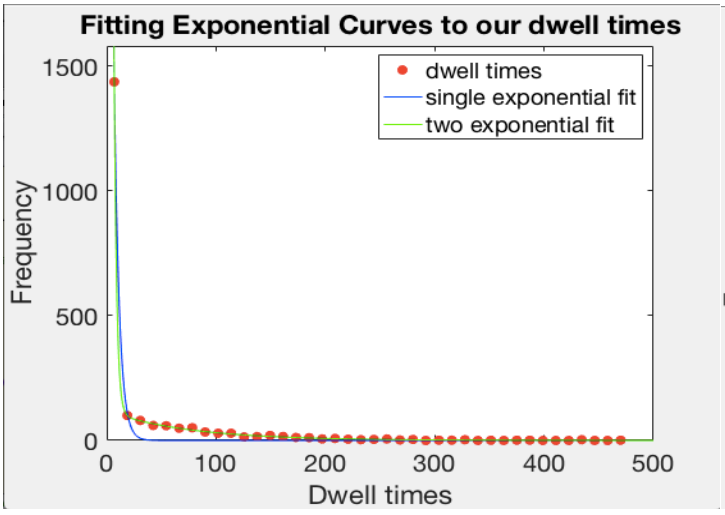
To gain a better understanding of how these dwell times are distributed, I generated a histogram. To get sufficient resolution, I experimented with several different system sizes.

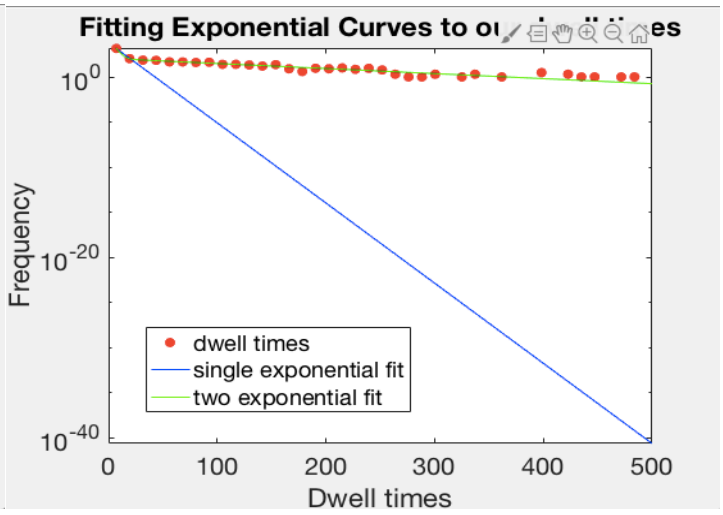
```
figure(2)
hist(closed_dwell_times, 40)
xlabel("Dwell Times");
ylabel("Frequency");
title('Ion Channel Dwell Times');
```



Increasing the timesteps (to say 100,000), we can get a better look at the distributions of these dwell times. As expected, it is steeply exponential. To determine if this data can be appropriately modeled with just a single exponential, we will attempt to fit one to it. For comparison, we will also fit a sum of two exponentials to it. We do so using the fit function.

```
figure(3)
[N, X] = hist(closed_dwell_times,40);
expfit = fit(X.', N.', 'exp1'); % fit to first degree exponential
expfit2 = fit(X.', N.', 'exp2'); % fit to second degree exponential

plot(X, N, 'r.', 'markersize', 20), hold on;
plot(expfit, 'b-'), hold on;
plot(expfit2, 'g-')
ylim([0,1.1*max(N)])
legend({'dwell times', 'single exponential fit', 'two exponential fit'})
set(gca, 'fontsize', 20);
title("Fitting Exponential Curves to our dwell times");
```





Above we see that the single exponential fit is a poor representation of the data. The left image is the normal plot, and the right is a corresponding log plot in y. The single exponential decreases much too rapidly to appropriately model the data, and the log plot on the right emphasizes how poorly it performs. On the other hand, the two exponential fit appears quite accurate which aligns with our findings in class. As this system has two closed states, we would expect its dwell times within those states to be modeled by the sum of two exponentials.

Problem 4: Simulating Markov Chains and Neural Spiking

Now, to calculate the probability of a spike at any given timepoint, we need to calculate some of the parameters of our system, namely p_{in} and p_{out}. As we have already shown these Markov Chains will have a stable distribution, we will go about finding it. First, we simply define all our conditions.

```
%% Problem 4: Simulating Markov Chains and Neural Spiking
clear all; close all; clc;
IN = [0.98 0.1 0;
    0.02 0.7  0.05;
    0 0.2 0.95];

OUT = [0.9 0.1 0;
       0.1 0.6 0.1;
        0 0.3 0.9];

N_in = 100;
N_out = 50;
T_range = 1:100;
```

We know that a stable distribution, by definition, will reach a point where matrix multiplication no longer changes it. Thus, the stable distribution is an eigenvector of our transition matrix with associated eigenvalue of one. We expect the other eigenvalues to be less than one so that as we take increasing powers of our transition matrix, their effects approach zero. Below, we can see this is indeed the case. Since we are interested in the probability the channels are open, we will take the last entries of π_{in} and π_{out} to be p_{in} and p_{out}.

```
[V_in, D_in] = eig(IN);

% we see third eigenvalue = 1, so our stable distribution is third column
% of V, normalized so it sums to one.
pi_in = V_in(:,3)/sum(V_in(:,3))

[V_out, D_out] = eig(OUT);

% We see third eigenvalue = 1, so our stable distribution is the third
% column of V, normalized so it sums to one.
pi_out = V_out(:,3) / sum(V_out(:,3))

p_in = pi_in(3);
p_out = pi_out(3);
```

```
V_in =

   -0.2488   -0.7391    0.7715
    0.7979    0.0692    0.1543
   -0.5490    0.6700    0.6172


D_in =

    0.6594         0         0
         0    0.9706         0
         0         0    1.0000
```

```
V_out =

    0.1961    0.7071    0.3015
   -0.7845   -0.0000    0.3015
    0.5883   -0.7071    0.9045


D_out =

    0.5000         0         0
         0    0.9000         0
         0         0    1.0000
```

```
pi_out =

    0.2000
    0.2000
    0.6000
```

```
pi_in =

    0.5000
    0.1000
    0.4000
```

Now, to calculate the probability of a spike at each threshold value (T) of interest, we simply apply the formula we derived earlier.

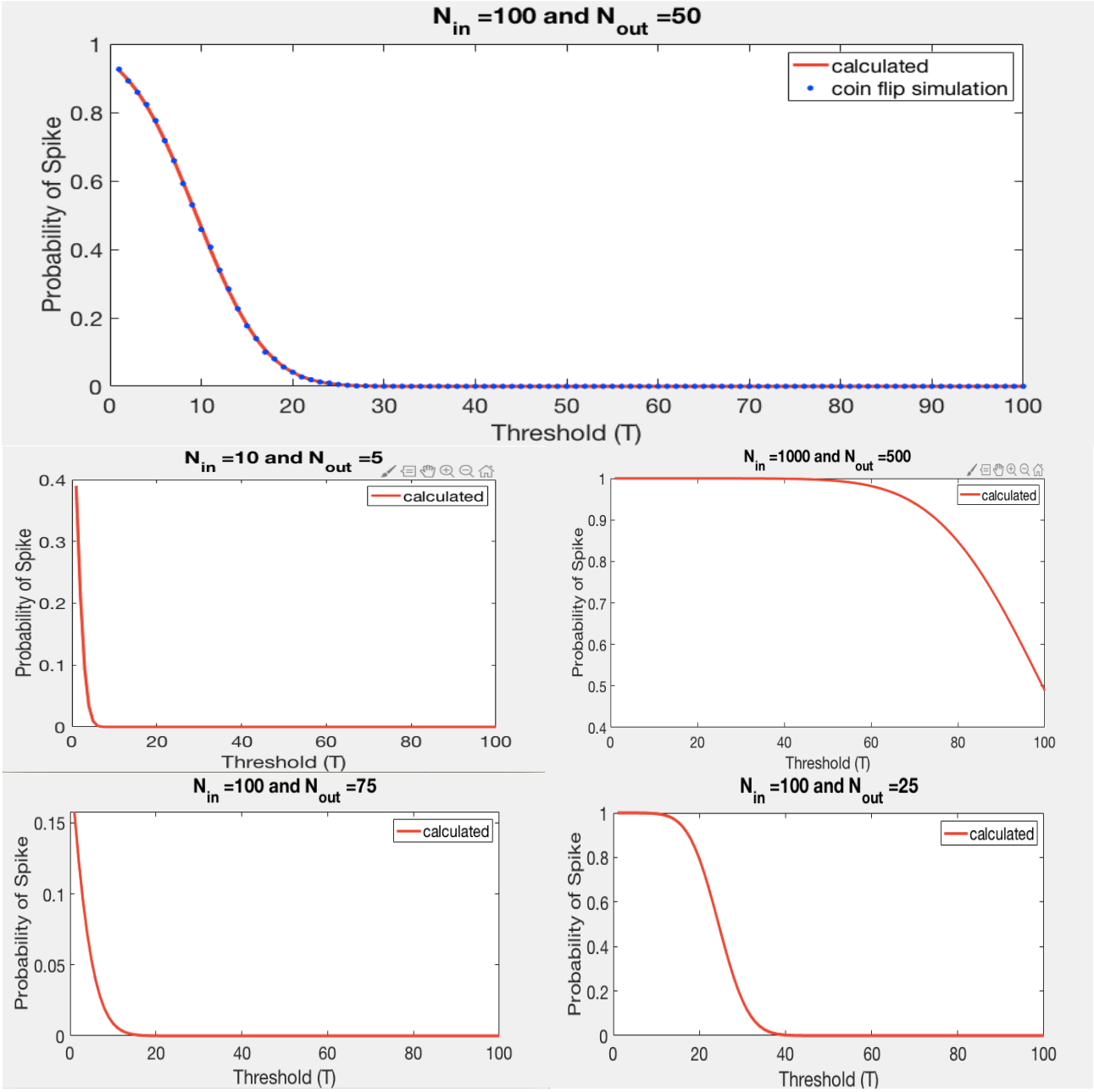
```
% Calculating using math
probs = zeros(1, length(T_range));

for index = 1:length(T_range)
    T = T_range(index);
    probs(index) = sum(binopdf(T+1+(0:N_in-T-1), N_in, p_in) ...
        .*binocdf(0:N_in-T-1, N_out, p_out));
end
```

Alternatively, we could have taken these stable distribution probabilities and simply done a series of coin flips to determine if a channel was open or not. Doing this for each channel, and if our trial number is sufficiently high, we can get a fairly accurate estimate of the true spike probability.

```
% Calculating Using Coin tosses

trials = 10000;
probs2 = zeros(1,length(T_range));
for index = 1:length(T_range)
    T = T_range(index);

    n_ins = sum(rand(N_in,trials) <= p_in);
    n_outs = sum(rand(N_out,trials) <= p_out);

    %n_ins = binornd(ones(1,trials) * N_in, ones(1,trials)*p_in);
    %n_outs = binornd(ones(1,trials) * N_out, ones(1,trials)*p_out);

    probs2(index) = sum(n_ins - n_outs > T) / trials;
end
```

Lastly, we plot our results to get a good visual.

```
figure(4)
plot(T_range, probs, 'r.-'), hold on;
plot(T_range, probs2, 'b.-'), hold off;
legend({'calculated', 'simulation'})
xlabel('Threshold (T)');
ylabel('Probability of Spike');
```



The first plot shows the two calculation methods used earlier produce fairly similar results, giving us confidence in our results. We have excluded results where N_{out} is larger than N_{in} as they produce uninteresting results. All the plots seem to have either this S-like structure, or a basic exponential decay. The plots with a lower overall channel count and runs where N_{out} is closer to N_{in} seem to exhibit this decay, while those with a larger count general exhibit this S-like structure. This might be because when N_{in} is twice the size of N_{out}, n_{in} has a greater expected value than n_{out} and if the channel count is large the variation about these expected values is relatively small, and it is not until T approaches the difference in these expectations that the chance of a spike starts to drop significantly. When the expected value of n_{in} is below that of n_{out}, our spike rate is consistently low.