



Exploration and Modeling of Neuron Tuning Curve

A dissertation presented
by

Baihan Lin

to
NBIO 303: Computational Neuroscience,
Department of Physiology and Biophysics

in partial fulfillment of the requirements
for the credits of
Computational Neuroscience
in the subject of
Neurobiology

University of Washington
Seattle, Washington, USA

January 2014

Mentors:

Prof. Adrienne Fairhall
Dr. Alison Weber

Author:

Baihan Lin

Abstract:

This thesis is to explore the utilization of MATLAB into the modeling and visualization of neuron tuning curve. The problem being solved here is simulating the experiment using 300 ms spike train in response to the input. The data is arranged in variable “spiketrain” and consists of matrixes of trials, each of which has millisecond timebins which are either zero if there was no spike in that bin, or one if there was.

The objective of the thesis is to plot raster plots of the spike train, calculate the sum, mean, and standard deviation within and among the trials, and compare them under different stimulus via distribution plot and d-prime. In further exploration, here models the whole process using different methods in different angles towards the mapping of tuning curve and discuss the possible manipulation to the tuning curve for an improvement of discriminability between data.

Keyword:

Matlab, Sum, Mean, Standard Deviation, D' (d-prime), Gaussian Distribution, Histogram

Contents:

Abstract	2
Keywords	2
1. Question 1: (Generate Noisy Data)	4
a) Make raster plots	
b) Calculate Mean and Standard Deviation	
2. Question 2: (Discriminate the Response)	7
a) Make Distribution Histogram	
b) Calculate D' (d-prime) as an indicator of discriminability	
3. Question 3: (Map out the tuning curve)	9
a) Method 1: the dumb ----- to explore the tendency	
b) Method 2: the functional ---to generalize the model	
c) Method 3: the cool -----to try a new direction	
4. Question 4: (Increase the discriminability)	18
a) Method 1: Change tuningcurve	
b) Method 2: Change maxrate	
c) Method 3: Change tau	
5. Graph Summary	24
6. Important codes	28
a) BaihanLinNBIO303HW1	
b) generateNoisyDataGeneral	
c) figureTuningCurveMethod2	
d) figureTuningCurveMethod3	
e) For all other codes, they are all attached in the zip file waiting to be reviewed.	
Bibliography	36
Acknowledgement	37

Question 1:

(To Generate Noisy Data)

Two goals:

1. Make raster plots
2. Calculate Mean and Standard Deviation

Here to compare, I changed the variables of generateNoisyData from input to pre-set. So the example here is input as 50 or 70, trial number both 1000.

generateNoisyData5090 (7090 is basically the same, thus not shown here)

```
% This code will generate a spiking response to an input (which is 50 here)
based on a tuning
% curve and assuming Poisson firing. The response adapts with a fixed
% timecourse. It runs 90 trials.
```

```
x1 = 50; % stimulus input
```

```
ntrials = 90; % trials
```

```
maxrate = 300; % 30 Hz max firing rate
```

```
rate = maxrate*tuningCurve(x1);
```

```
tau = 100; % adaptation time constant in msec
```

```
nmsec = 300; % number of milliseconds to record for
```

```
times= 1:nmsec; % time units
```

```
spiketrain5090 = zeros(ntrials,nmsec); % set up output data
```

```
ratecurve = rate*exp(-times/tau)*.001; % adapting rate function
```

```
for j = 1:ntrials;
```

```
    for i = 1:nmsec;
```

```
        if(rand(1)<ratecurve(i)),
```

```
            spiketrain5090(j,i) = 1;
```

```
        end;
```

```
    end;
```

```
end;
```

Mean Codes:

```
%*****
% Question 1:

% make spiketrain workspace
generateNoisyData5090; % spiketrain5030 workspace: 50input, 90trials
generateNoisyData7090; % spiketrain7030 workspace: 70input, 90trials

% make raster plot with two subplots.
figure;
subplot(2,1,1)
raster5090 = imagesc(spiketrain5090);
xlabel('spikes')
ylabel('trials')
title('stimulus input = 50');
subplot(2,1,2)
raster7090 = imagesc(spiketrain7090);
xlabel('spikes')
ylabel('trials');
title('stimulus input = 70');

%% Do the statistics
% make matrix to store the sum of spikes in each trail in a column
Sum5090 = sum(spiketrain5090, 2);
Sum7090 = sum(spiketrain7090, 2);

% calculate the means of spikes in different trails
Mean5090 = mean(Sum5090);
Mean7090 = mean(Sum7090);
disp(['Mean of input 50 and 70 are ', num2str(Mean5090), ' and ',
num2str(Mean7090)]);

% calculate the standard deviation of spikes in different trails
Std5090 = std(Sum5090);
Std7090 = std(Sum7090);
disp(['Standard Deviation of input 50 and 70 are ', num2str(Std5090), ' and ',
', num2str(Std7090)]);
```

So finally, the statistics we calculated will be printed out on screen. The graph is on next page →

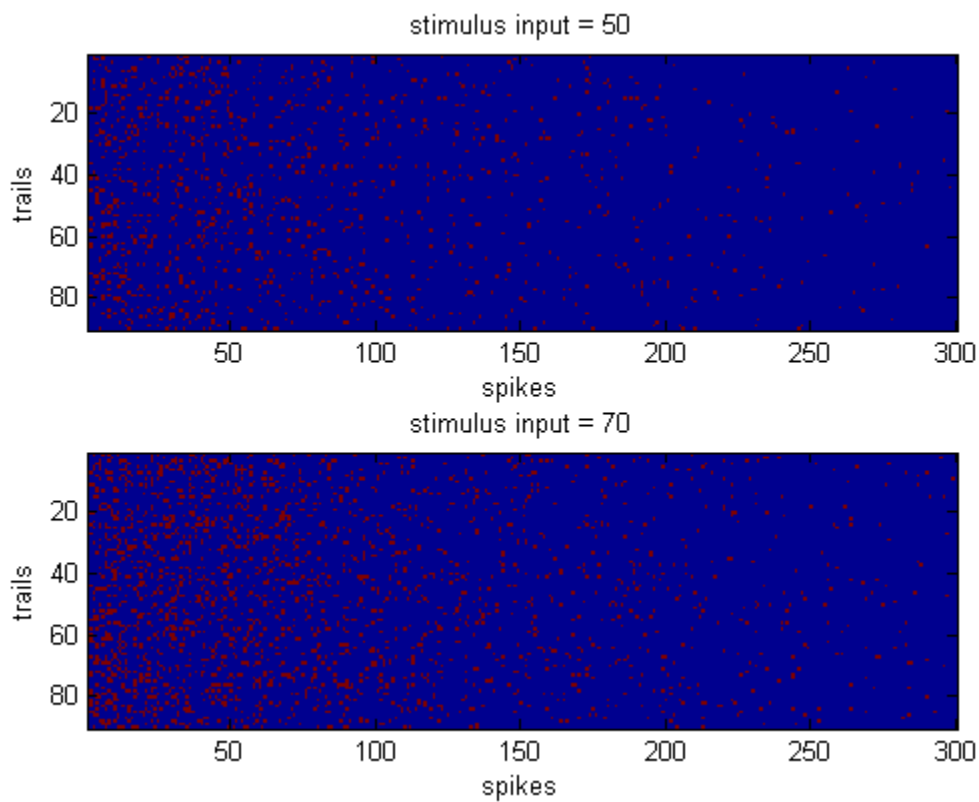
Graph (raster plots)

Figure 1. The comparison between the spike raster plots under stimulus input 50 and 70, trials number 1000.

Display on screen would be:

```
Mean of input 50 and 70 are 14.3222 and 25.0222
Standard Deviation of input 50 and 70 are 3.777 and 4.7522
```

Question 2:

(Discriminate the Response)

Two Goals:

1. Make Distribution Histogram
2. Calculate D' (d-prime) as an indicator of discriminability

Mean Codes:

Notes: here all the main codes depend on their previous cells, and here they are just extracted to be explained in different chapters.

```
%%
% *****
% Question 2:

% try to create line of best fit models: Gaussian distribution
x=0:50;
f5090 = 90*exp(-(x-Mean5090).^2/(2*Std5090.^2))/(Std5090*sqrt(2*pi));
f7090 = 90*exp(-(x-Mean7090).^2/(2*Std7090.^2))/(Std7090*sqrt(2*pi));

% make frequency distribution histogram
figure;
scale = 0:50;
Distribution5090 = histc(Sum5090, scale);
bar(scale,Distribution5090)
xlabel('spikes')
ylabel('frequency')
hold on
Distribution7090 = histc(Sum7090, scale);
bar(scale, Distribution7090, 'r')
xlabel('spikes')
ylabel('frequency')
xlim([0 50])
ylim([0 20])
title('Distribution Histogram of Spikes')

% for Gaussian distribution
hold on
```

```

plot(x, f5090)
hold on
plot(x, f7090, 'r')

% explor how discriminable

dprime = abs((Mean5090 - Mean7090)/sqrt((Std5090^2+Std7090^2)/2))
disp(['dprime is ', num2str(dprime)])

```

Graph (distribution histograms)

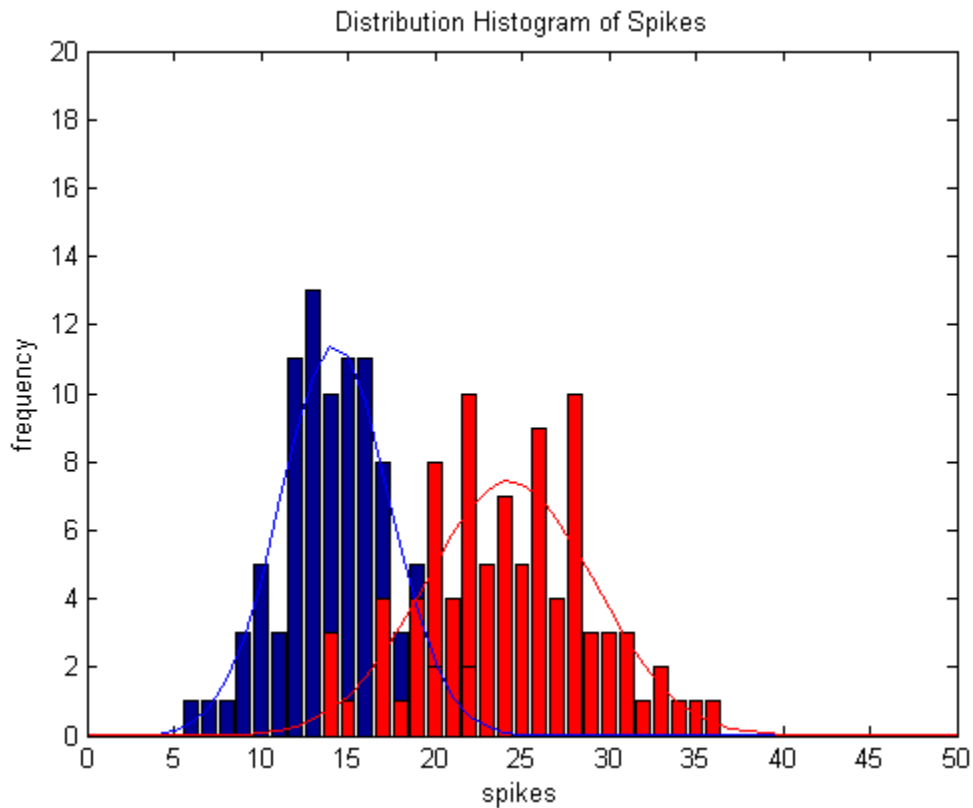


Figure 2. The comparison between the distribution histograms and their simulated Gaussian distribution under stimulus input 50 and 70, trials number 1000.

Display on screen would be:

```

dprime =

    2.4928

dprime is 2.4928

```


Question 3:

(Map out the tuning curve)

Three Attempts

(Exploration → Problem-Solving → Extending):

1. Method 1: the dumb ----- to explore the tendency
2. Method 2: the functional ---to generalize the model
3. Method 3: the cool -----to try a new direction

Mean Codes:

```
%%
% *****
% Question 3
% Map out the tuning curve, with error bars

choice = input('So here we are at question 3, the fun part.\n To map out
the tuning curve, I tried 3 methods.\n Which of my 3 methods do you like
to use?\n 1: the dumb method, to explore the trend\n 2: the functional method,
to generalize the model\n 3: the cool method, to try a new direction\n please
enter 1, 2, or 3. Thx!\n')

switch choice
    case 1
        disp('please wait for a little while...Thank you for the patience.')
        figureTuningCurveMethod1;
    case 2
        disp('This is fast. So you may only wait for a few seconds...Thank
you for the patience.')
        figureTuningCurveMethod2;
    case 3
        disp('Every innovation comes at a price.\n Please wait for a little
while...Thank you for the patience.')
        figureTuningCurveMethod3;
    otherwise
        disp('Please make the right choice, alright?')
end
```

Display on screen would be:

```

So here we are at question 3, the fun part.
To map out the tuning curve, I tried 3 methods.
Which of my 3 methods do you like to use?
  1: the dumb method, to explore the trend
  2: the functional method, to generalize the model
  3: the cool method, to try a new direction
please enter 1, 2, or 3. Thx!
fx

```

If press 1:

```

choice =

    1

please wait for a little while...Thank you for the patience.
fx

```

Then this code runs:

figureTuningCurveMethod1

```

% This is the most orthodoxical, simplistic and dumb method among the three
I thought of.

```

```

% So here I basically created 20 different scripts to replicate what I did
% in question 1 and 2, then plotted them on the graph here.

```

```

generateNoisyData101000; % spiketrain workspace: 10input, 1000trials
generateNoisyData201000; % spiketrain workspace: 20input, 1000trials
generateNoisyData301000; % spiketrain workspace: 30input, 1000trials
generateNoisyData401000; % spiketrain workspace: 40input, 1000trials
generateNoisyData501000; % spiketrain workspace: 50input, 1000trials
generateNoisyData601000; % spiketrain workspace: 60input, 1000trials
generateNoisyData701000; % spiketrain workspace: 70input, 1000trials
generateNoisyData801000; % spiketrain workspace: 80input, 1000trials
generateNoisyData901000; % spiketrain workspace: 90input, 1000trials
generateNoisyData1001000; % spiketrain workspace: 100input, 1000trials
generateNoisyData1101000; % spiketrain workspace: 110input, 1000trials
generateNoisyData1201000; % spiketrain workspace: 120input, 1000trials
generateNoisyData1301000; % spiketrain workspace: 130input, 1000trials

```

```
generateNoisyData1401000; % spiketrain workspace: 140input, 1000trials
generateNoisyData1501000; % spiketrain workspace: 150input, 1000trials
generateNoisyData1601000; % spiketrain workspace: 160input, 1000trials
generateNoisyData1701000; % spiketrain workspace: 170input, 1000trials
generateNoisyData1801000; % spiketrain workspace: 180input, 1000trials
generateNoisyData1901000; % spiketrain workspace: 190input, 1000trials
generateNoisyData2001000; % spiketrain workspace: 200input, 1000trials
```

```
% make matrix to store the sum of spikes in each trail in a column
```

```
Sum101000 = sum(spiketrain101000, 2);
Sum201000 = sum(spiketrain201000, 2);
Sum301000 = sum(spiketrain301000, 2);
Sum401000 = sum(spiketrain401000, 2);
Sum501000 = sum(spiketrain501000, 2);
Sum601000 = sum(spiketrain601000, 2);
Sum701000 = sum(spiketrain701000, 2);
Sum801000 = sum(spiketrain801000, 2);
Sum901000 = sum(spiketrain901000, 2);
Sum1001000 = sum(spiketrain1001000, 2);
Sum1101000 = sum(spiketrain1101000, 2);
Sum1201000 = sum(spiketrain1201000, 2);
Sum1301000 = sum(spiketrain1301000, 2);
Sum1401000 = sum(spiketrain1401000, 2);
Sum1501000 = sum(spiketrain1501000, 2);
Sum1601000 = sum(spiketrain1601000, 2);
Sum1701000 = sum(spiketrain1701000, 2);
Sum1801000 = sum(spiketrain1801000, 2);
Sum1901000 = sum(spiketrain1901000, 2);
Sum2001000 = sum(spiketrain2001000, 2);
```

```
% calculate the means of spikes in different trails
```

```
Meanall(10) = mean(Sum101000);
Meanall(20) = mean(Sum201000);
Meanall(30) = mean(Sum301000);
Meanall(40) = mean(Sum401000);
Meanall(50) = mean(Sum501000);
Meanall(60) = mean(Sum601000);
Meanall(70) = mean(Sum701000);
Meanall(80) = mean(Sum801000);
Meanall(90) = mean(Sum901000);
Meanall(100) = mean(Sum1001000);
```

```
Meanall(110) = mean(Sum1101000);
Meanall(120) = mean(Sum1201000);
Meanall(130) = mean(Sum1301000);
Meanall(140) = mean(Sum1401000);
Meanall(150) = mean(Sum1501000);
Meanall(160) = mean(Sum1601000);
Meanall(170) = mean(Sum1701000);
Meanall(180) = mean(Sum1801000);
Meanall(190) = mean(Sum1901000);
Meanall(200) = mean(Sum2001000);

% calculate the standard deviation of spikes in different trails
Stdall(10) = std(Sum101000);
Stdall(20) = std(Sum201000);
Stdall(30) = std(Sum301000);
Stdall(40) = std(Sum401000);
Stdall(50) = std(Sum501000);
Stdall(60) = std(Sum601000);
Stdall(70) = std(Sum701000);
Stdall(80) = std(Sum801000);
Stdall(90) = std(Sum901000);
Stdall(100) = std(Sum1001000);
Stdall(110) = std(Sum1101000);
Stdall(120) = std(Sum1201000);
Stdall(130) = std(Sum1301000);
Stdall(140) = std(Sum1401000);
Stdall(150) = std(Sum1501000);
Stdall(160) = std(Sum1601000);
Stdall(170) = std(Sum1701000);
Stdall(180) = std(Sum1801000);
Stdall(190) = std(Sum1901000);
Stdall(200) = std(Sum2001000);

% make graph
stmls = 10:10:200;
figure;
plot(stmls, Meanall(stmls))
title('Tuning Curve')
xlabel('Stimulus')
ylabel('Response')
errorbar(stmls, Meanall(stmls), Stdall(stmls))
```

Graph by this (tuning curve with error bar)

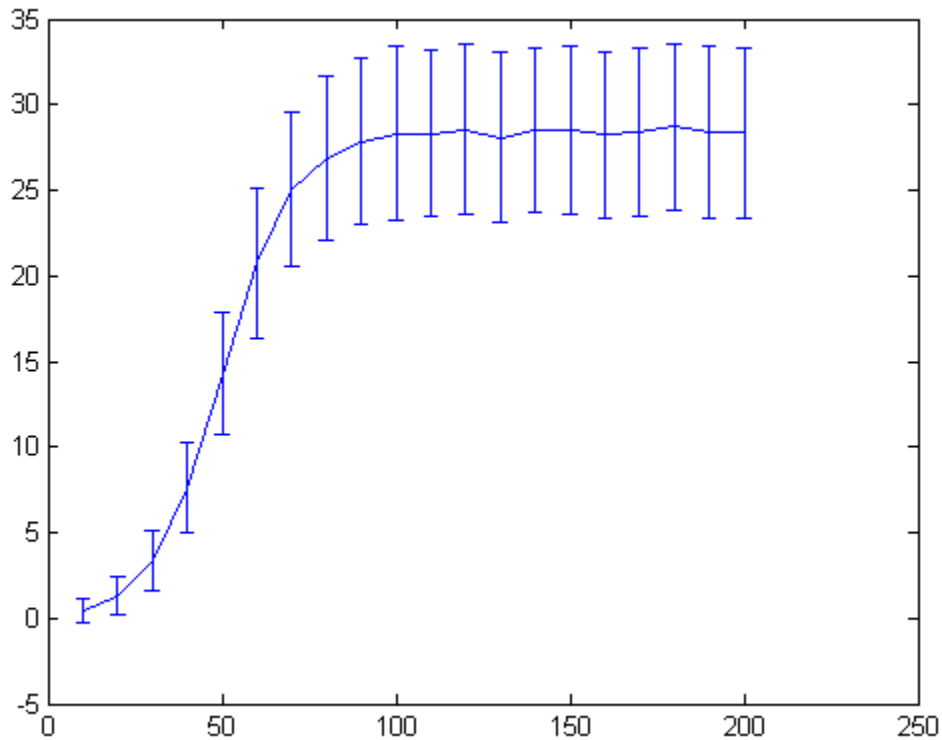


Figure 3. the tuning curve mapped based on method 1: the dumb, to explore the tendency.

If press 2:

```
choice =
    2

This is fast. So you may only wait for a few seconds...Thank you for the patience.
```

Then this code runs:

figureTuningCurveMethod2

```
%%
% So we use a function that calculate the mean and standard deviation based
on the given stimulus input
```

```
for k = 1:5:400
    [meanall(k),stdall(k)] = generateNoisyDataGeneral(k,1000);
end

stmls = 1:5:400;

figure;
plot(stmls,meanall(stmls))
title('Tuning Curve')
xlabel('Stimulus')
ylabel('Response')
errorbar(stmls, meanall(stmls), stdall(stmls));
```

This code calls:

generateNoisyDataGeneral

```
% This code will generate a spiking response to an input based on a tuning
% curve and assuming Poisson firing. The response adapts with a fixed
% timecourse.
%
% The interesting part of this function is that it is based on the x, which
% is stimulus input, and y, which is trials, that you input in the (). So
% it is a general form of functions like generateNoisyData5090 and
% generateNoisyData7090.
%

%function [Meantemp] = generateNoisyDataGeneral(x, y)
function [Meantemp, Stdtemp] = generateNoisyDataGeneral(x, y)
%function [Meantemp] = generateNoisyDataGeneral(x)

sinput = x; % stimulus input
ntrials = y; % number of trials

maxrate = 300; % 30 Hz max firing rate
rate = maxrate*tuningCurve(sinput);
tau = 100; % adaptation time constant in msec
nmsec = 300; % number of milliseconds to record for
times= 1:nmsec; % time units
spiketraintemp = zeros(ntrials,nmsec); % set up output data
ratecurve = .001*rate*exp(-times/tau); % adapting rate function
```

```

for j = 1:ntrials;
    for i = 1:nmsec;
        if(rand(1)<ratecurve(i)),
            spiketraintemp(j,i) = 1;
        end;
    end;
end;

Sumtemp = sum(spiketraintemp, 2);
Meantemp = mean(Sumtemp);
Stdtemp = std(Sumtemp);
%stdall(x) = std(Sumtemp)
end

```

Graph by this (tuning curve with error bar)

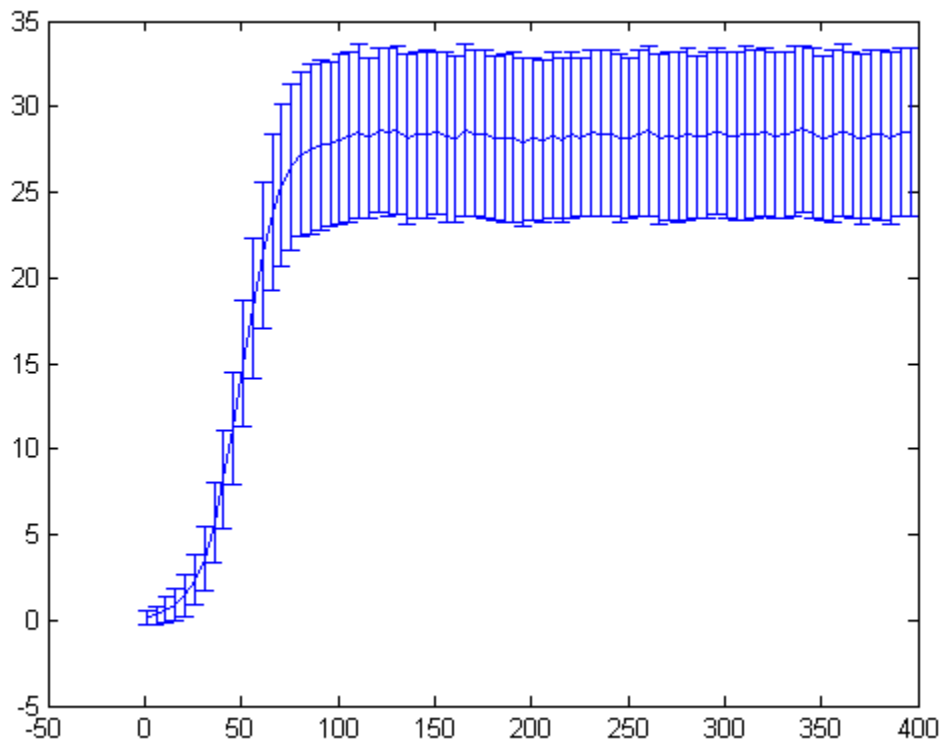


Figure 4. the tuning curve mapped based on method 2: the functional, to generalize the model of the tuning curve.

If press 3:

```
choice =  
  
3  
  
Every innovation comes at a price.\n Please wait for a little while...Thank you for the patience.
```



Then this code runs:

figureTuningCurveMethod3

```
% The cool part is that I created a 3D matrix, so one dimension is the  
% spikes, one is trials, one is stimulus. The I reduce the dimension using  
% methods like mean(), sum(), and std(), finally I squeeze the matrix into  
% two functions of mean and std which can be plotted.
```

```
alldata = zeros(ntrials, nmsec, 400);
```

```
ntrials = 1000;
```

```
nmsec = 300; % number of milliseconds to record for
```

```
times= 1:nmsec; % time units
```

```
tau = 100; % adaptation time constant in msec
```

```
for k = 1:80 % Because I want to plot out all stimulus 1:5:400, thus third  
dimension is 80.
```

```
x1 = 5*k;
```

```
maxrate = 300; % 30 Hz max firing rate
```

```
rate = maxrate*tuningCurve(x1);
```

```
ratecurve = rate*exp(-times/tau)*.001; % adapting rate function
```

```
for j = 1:ntrials;
```

```
for i = 1:nmsec;
```

```
if(rand(1)<ratecurve(i)),
```

```
alldata(j,i,k) = 1;
```

```
end
```

```
end
```

```
end;
```

```
end
```



```

allmean = mean(squeeze(sum(alldata,2)),1);
allstd = std(squeeze(sum(alldata,2)),1);

stmls = 1:80;

figure;
plot(5*stmls,allmean(stmls))
title('Tuning Curve')
xlabel('Stimulus')
ylabel('Response')
errorbar(5*stmls, allmean(stmls), allstd(stmls))

```

Graph by this (tuning curve with error bar)

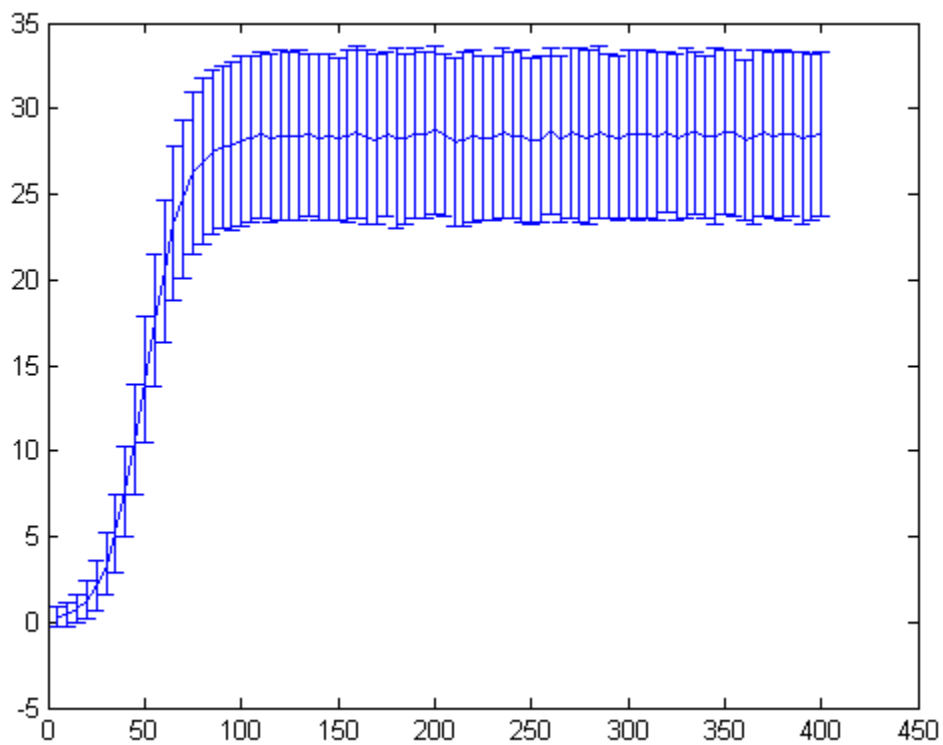



Figure 5. the tuning curve mapped based on method 3: the cool, use 3D matrix as a new direction to explore tuning curve.

Question 4:



(Increase the discriminability)

Three Directions:

1. Method 1: Change tuningcurve
2. Method 2: Change maxrate
3. Method 3: Change tau

Mean Codes:

```
%%
% *****
% Question 4

% I think of 3 methods.

% Method 1: Change tuningcurve function
% change tuningcurve function: f = 1./(1 + exp(-(x-50)/10))
% to
% f = n*1./(1% + exp(-(m*x-50)/10)), where n and m both >1.
% so that the shape of the graph is thinner and taller, and more
% discriminable.

% Method 2: Change maxrate
% increase maxrate to increase rate and make bigger data (taller), thus more
discriminable.

% Method 3: Change tau
% increase tau to increase ratecurve and make bigger data (taller), thus
more discriminable.

selection = input('So here is question 4.How to make tuning curve more
discriminable at 20?\n I got 3 methods to make tuning curve more discriminable.
Which to try?\n 1: change tuningcurve function\n 2: change maxrate\n 3:
change tau\nplease input 1, 2 or 3 and follow instruction.\n');
[x15, s15] = generateNoisyDataGeneral(15,1000);
[x25, s25] = generateNoisyDataGeneral(25,1000);
result = dprimef (x15, x25, s15, s25);
```

```
switch selection
    case 1
        disp('You made a great choice!\n Now first look at original graph and
d-prime between stimulus 15 and 25.\n')
        figureTuningCurveMethod2;
        disp(['dprime is ', num2str(result)]);
        selection1 = input('Then, we will now change turningcurve function
from f = 1./(1 + exp(-(x-50)/10))\n to f = n./(1% + exp(-(m*x-50)/10)), where
n>1, m>1\n Here as an example we set n=5, m=4.\n Are you ready to see the
graph and dprime now? Press 4 to see.\n');
        switch selection1
            case 4
                AfterTCchange;
                [x115, s115] = generateWeirdData1(15,1000);
                [x125, s125] = generateWeirdData1(25,1000);
                result1 = dprimef (x115, x125, s115, s125);
                disp(['dprime is ', num2str(result1), '--> Much bigger, right?
--> More discriminable.']);
            otherwise
                disp('You coward, you should have pressed 4!');
        end
    case 2
        disp('You made a great choice!\n Now first look at original graph and
d-prime between stimulus 15 and 25.\n')
        figureTuningCurveMethod2;
        disp(['dprime is ', num2str(result)]);
        selection2 = input('Then, we will now increase maxrate from 300 to
n\n Here as an example we set n=600.\n Are you ready to see the graph and
dprime now? Press 5 to see.\n');
        switch selection2
            case 5
                AfterMRchange;
                [x215, s215] = generateWeirdData2(15,1000);
                [x225, s225] = generateWeirdData2(25,1000);
                result2 = dprimef (x215, x225, s215, s225);
                disp(['dprime is ', num2str(result2), '--> Much bigger, right?
--> More discriminable.']);
            otherwise
                disp('You coward, you should have pressed 5!');
        end
    case 3
```



```

disp('You made a great choice!\n Now first look at original graph and
d-prime between stimulus 15 and 25.\n')
figureTuningCurveMethod2;
disp(['dprime is ', num2str(result)]);
selection3 = input('Then, we will now increase tau from 100 to n.\n
Here as an example we set n=300\n Are you ready to see the graph and dprime
now? Press 6 to see.\n');
switch selection3
    case 6
        AfterTauchange;
        [x315, s315] = generateWeirdData3(15,1000);
        [x325, s325] = generateWeirdData3(25,1000);
        result3 = dprimef (x315, x325, s315, s325);
        disp(['dprime is ', num2str(result3), '--> Much bigger, right?
--> More discriminable.']);
    otherwise
        disp('You coward, you should have pressed 6!');
end
otherwise
    disp('Please make the right choice, alright?')
end
end

```

The screen displays:

```

So here is question 4.How to make tuning curve more discriminable at 20?
I got 3 methods to make tuning curve more discriminable. Which to try?
1: change tuningcurve function
2: change maxrate
3: change tau
please input 1, 2 or 3 and follow instruction.
fx

```

If press 1

```

You made a great choice!\n Now first look at original graph and d-prime between stimulus 15 and 25.

```

Then the previous scene in question 3 appears, plus the following:

```

dprime is 1.0555
Then, we will now change turningcurve function from  $f = 1./(1 + \exp(-(x-50)/10))$ 
to  $f = n./(1\% + \exp(-(m*x-50)/10))$ , where  $n>1$ ,  $m>1$ 
Here as an example we set  $n=5$ ,  $m=4$ .
Are you ready to see the graph and dprime now? Press 4 to see.

```

Press 4

dprime is 4.3772--> Much bigger, right? --> More discriminable.

And the **Graph**:

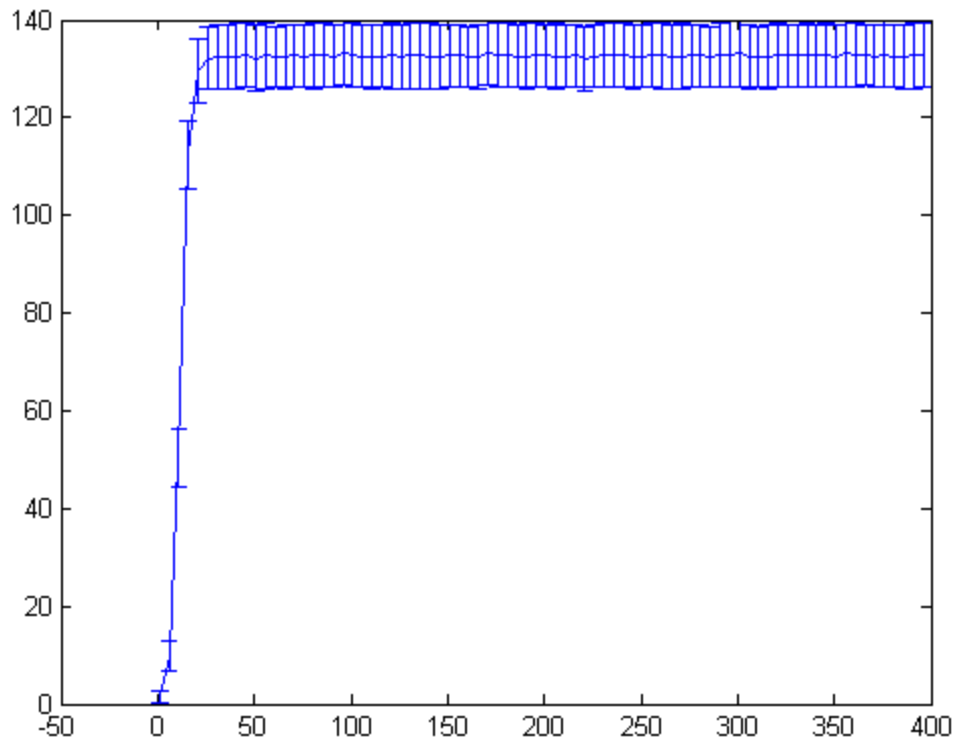


Figure 6. the manipulated tuning curve mapped under changed tuningCurve function.

If press 2

You made a great choice!\n Now first look at original graph and d-prime between stimulus 15 and 25.

Then the previous scene in question 3 appears, plus the following:

dprime is 1.144
Then, we will now increase maxrate from 300 to n
Here as an example we set n=600.
Are you ready to see the graph and dprime now? Press 5 to see.

Press 5:

dprime is 1.5979--> Much bigger, right? --> More discriminable.

And the **Graph**:

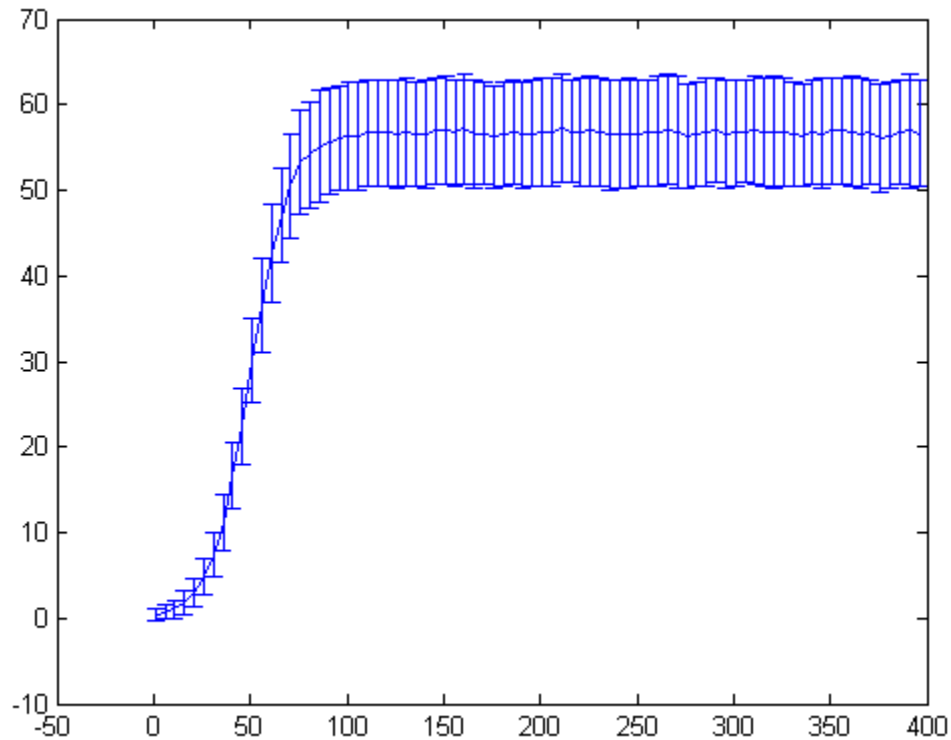


Figure 7. the manipulated tuning curve mapped under changed maxrate.

If press 3

You made a great choice!\n Now first look at original graph and d-prime between stimulus 15 and 25.

Then the previous scene in question 3 appears, plus the following:

dprime is 1.0681
Then, we will now increase tau from 100 to n.
Here as an example we set n=300
Are you ready to see the graph and dprime now? Press 6 to see.

Press 6:

dprime is 1.552--> Much bigger, right? --> More discriminable.

And the **Graph**:

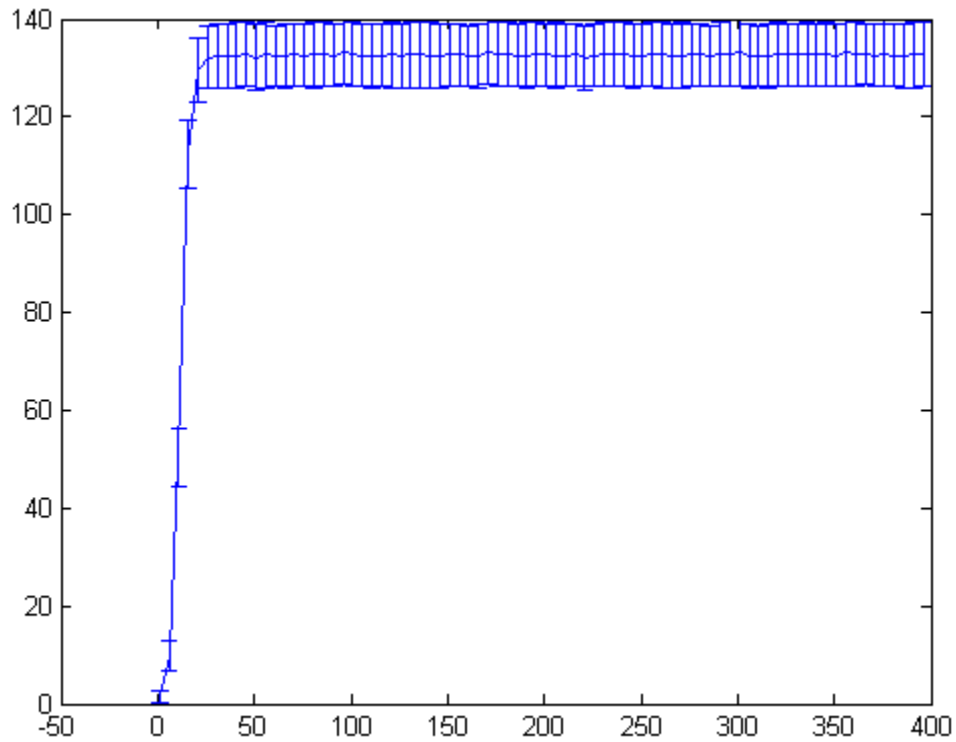


Figure 6. the manipulated tuning curve mapped under changed tuningCurve function.

In conclusion, we can see that all three ways successfully increase the discriminability of the response around 20. This is because we change the intrinsic parameter of tuning curve which makes the graph more steep and less spread.

Graph Summary:

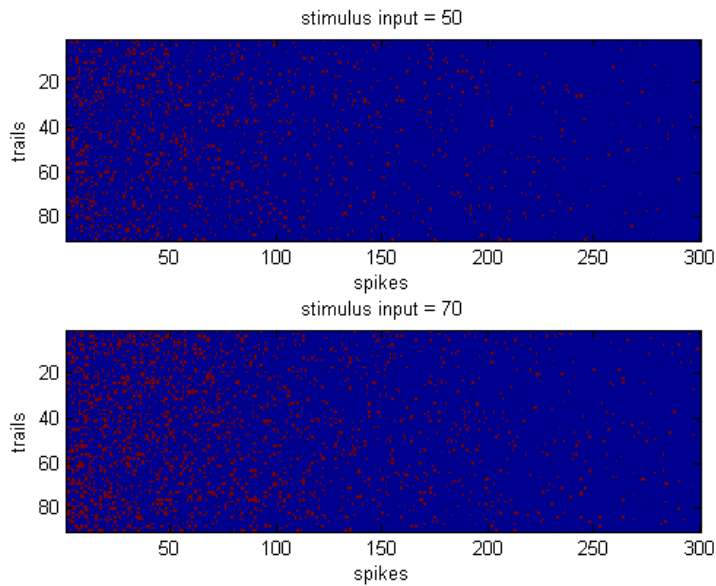


Figure 1. The comparison between the spike raster plots under stimulus input 50 and 70, trials number 1000.

Generated in question 1.

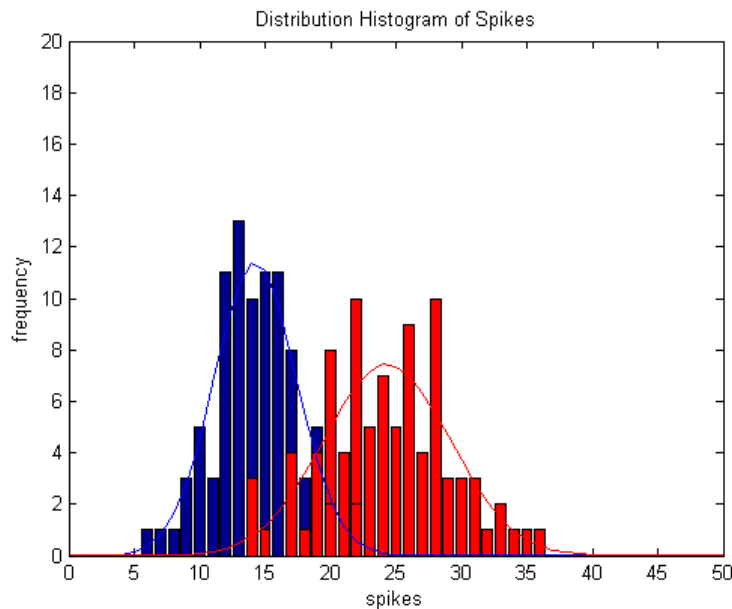


Figure 2. The comparison between the distribution histograms and their simulated Gaussian distribution under stimulus input 50 and 70, trials number 1000.

Generated in question 2.

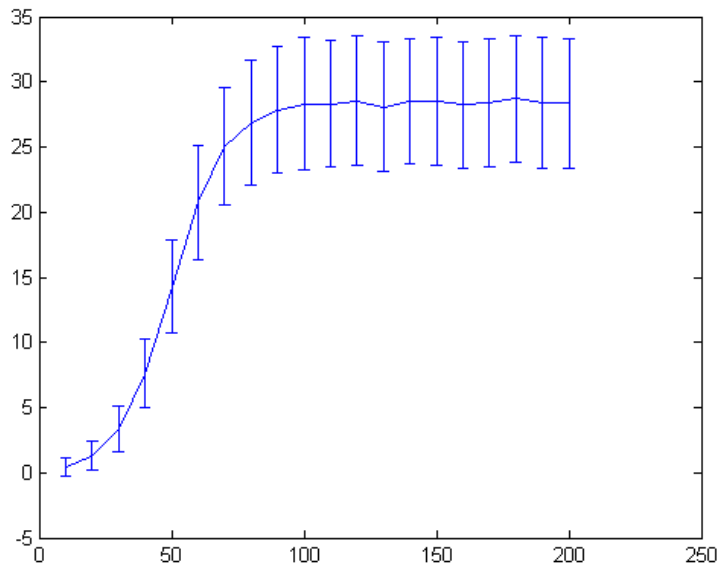


Figure 3. the tuning curve mapped based on method 1: the dumb, to explore the tendency.

Generated in question 3, figureTuningCurveMethod1.

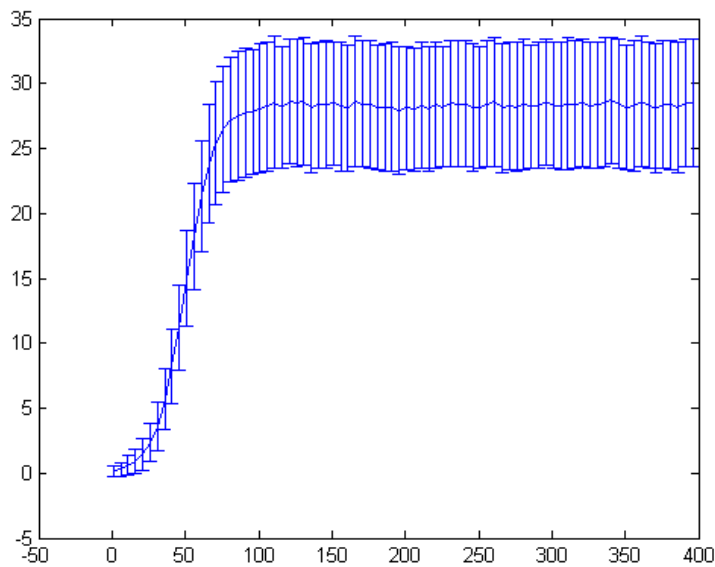


Figure 4. the tuning curve mapped based on method 2: the functional, to generalize the model of the tuning curve.

Generated in question 3, figureTuningCurveMethod2.

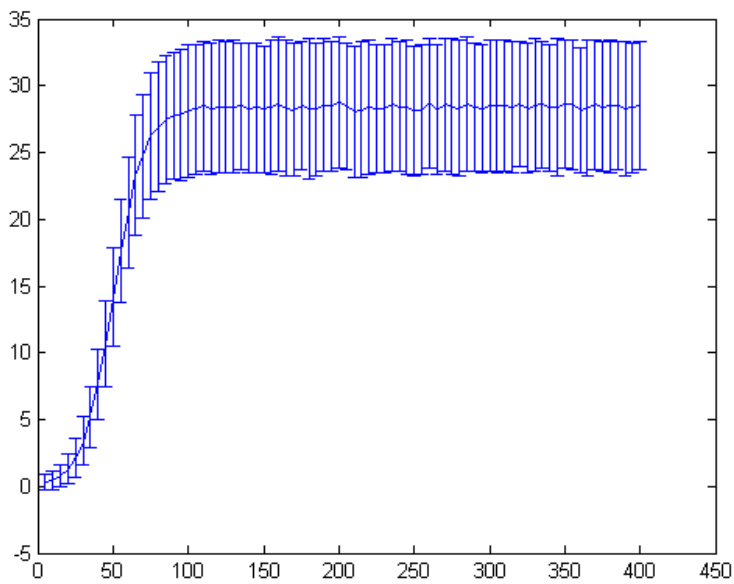


Figure 5. the tuning curve mapped based on method 3: the cool, use 3D matrix as a new direction to explore tuning curve.

Generated in question 3, figureTuningCurveMethod3.

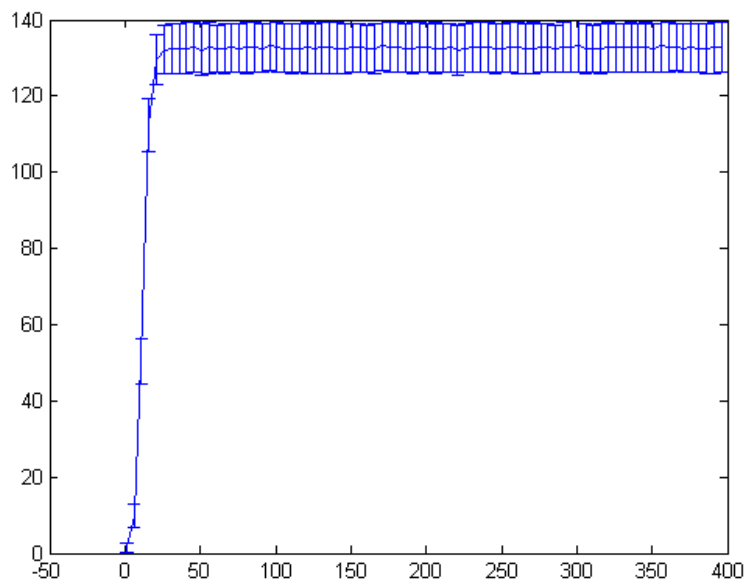


Figure 6. the manipulated tuning curve mapped under changed tuningCurve function.

Generated in question 4, Method 1.

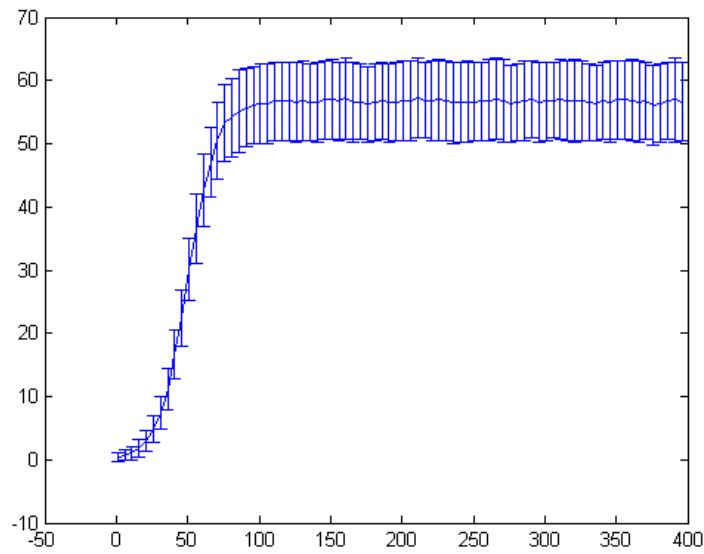


Figure 7. the manipulated tuning curve mapped under changed maxrate.

Generated in question 4, Method 2.

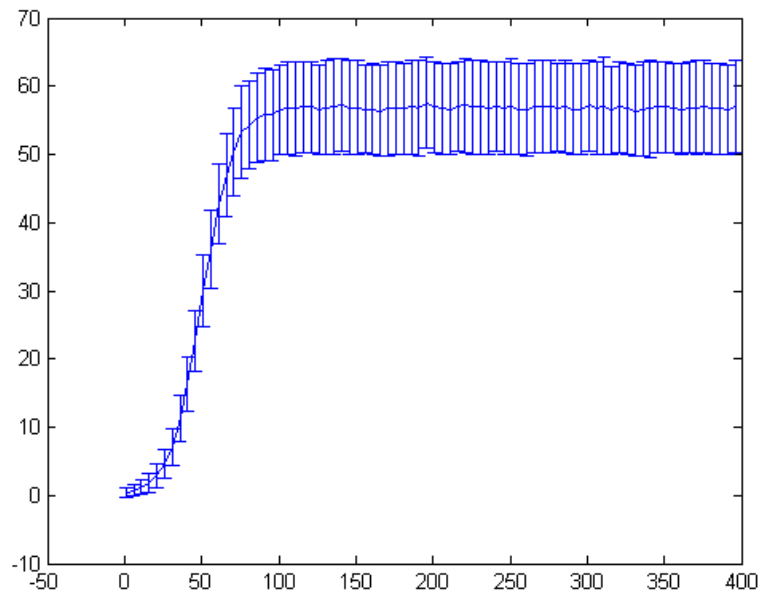


Figure 8. the manipulated tuning curve mapped under changed τ .

Generated in question 4, Method 3.

Important Codes:

a) BaihanLinNBIO303HW1

```
%*****
% Question 1:

% make spiketrain workspace
generateNoisyData5090; % spiketrain5030 workspace: 50input, 90trials
generateNoisyData7090; % spiketrain7030 workspace: 70input, 90trials

% make raster plot with two subplots.
figure;
subplot(2,1,1)
raster5090 = imagesc(spiketrain5090);
xlabel('spikes')
ylabel('trials')
title('stimulus input = 50');
subplot(2,1,2)
raster7090 = imagesc(spiketrain7090);
xlabel('spikes')
ylabel('trials');
title('stimulus input = 70');

%% Do the statistics
% make matrix to store the sum of spikes in each trail in a column
Sum5090 = sum(spiketrain5090, 2);
Sum7090 = sum(spiketrain7090, 2);

% calculate the means of spikes in different trails
Mean5090 = mean(Sum5090);
Mean7090 = mean(Sum7090);
disp(['Mean of input 50 and 70 are ', num2str(Mean5090), ' and ',
num2str(Mean7090)]);

% calculate the standard deviation of spikes in different trails
Std5090 = std(Sum5090);
Std7090 = std(Sum7090);
disp(['Standard Deviation of input 50 and 70 are ', num2str(Std5090), ' and ',
num2str(Std7090)]);
```

```
%%
% *****
% Question 2:

% try to create line of best fit models: Gaussian distribution
x=0:50;
f5090 = 90*exp(-(x-Mean5090).^2/(2*Std5090.^2))/(Std5090*sqrt(2*pi));
f7090 = 90*exp(-(x-Mean7090).^2/(2*Std7090.^2))/(Std7090*sqrt(2*pi));

% make frequency distribution histogram
figure;
scale = 0:50;
Distribution5090 = histc(Sum5090, scale);
bar(scale,Distribution5090)
xlabel('spikes')
ylabel('frequency')
hold on
Distribution7090 = histc(Sum7090, scale);
bar(scale, Distribution7090, 'r')
xlabel('spikes')
ylabel('frequency')
xlim([0 50])
ylim([0 20])
title('Distribution Histogram of Spikes')

% for Gaussian distribution
hold on
plot(x, f5090)
hold on
plot(x, f7090, 'r')

%%
% explor how discriminable

dprime = abs((Mean5090 - Mean7090)/sqrt((Std5090^2+Std7090^2)/2))
disp(['dprime is ', num2str(dprime)])

%%
```

```
% *****
% Question 3
% Map out the tuning curve, with error bars

choice = input('So here we are at question 3, the fun part.\n To map out
the tuning curve, I tried 3 methods.\n Which of my 3 methods do you like
to use?\n 1: the dumb method, to explore the trend\n 2: the functional method,
to generalize the model\n 3: the cool method, to try a new direction\n please
enter 1, 2, or 3. Thx!\n')

switch choice
    case 1
        disp('please wait for a little while...Thank you for the patience.')
        figureTuningCurveMethod1;
    case 2
        disp('This is fast. So you may only wait for a few seconds...Thank
you for the patience.')
        figureTuningCurveMethod2;
    case 3
        disp('Every innovation comes at a price.\n Please wait for a little
while...Thank you for the patience.')
        figureTuningCurveMethod3;
    otherwise
        disp('Please make the right choice, alright?')
end

%%
% *****
% Question 4

% I think of 3 methods.

% Method 1: Change tuningcurve function
% change tuningcurve function:  $f = 1./(1 + \exp(-(x-50)/10))$ 
% to
%  $f = n*1./(1 + \exp(-(m*x-50)/10))$ , where n and m both >1.
% so that the shape of the graph is thinner and taller, and more
% discriminable.

% Method 2: Change maxrate
```

```
% increase maxrate to increase rate and make bigger data (taller), thus more discriminable.
```

```
% Method 3: Change tau
```

```
% increase tau to increase ratecurve and make bigger data (taller), thus more discriminable.
```

```
selection = input('So here is question 4.How to make tuning curve more discriminable at 20?\n I got 3 methods to make tuning curve more discriminable. Which to try?\n 1: change tuningcurve function\n 2: change maxrate\n 3: change tau\n please input 1, 2 or 3 and follow instruction.\n');
[x15, s15] = generateNoisyDataGeneral(15,1000);
[x25, s25] = generateNoisyDataGeneral(25,1000);
result = dprimef (x15, x25, s15, s25);
```

```
switch selection
```

```
    case 1
```

```
        disp('You made a great choice!\n Now first look at original graph and d-prime between stimulus 15 and 25.\n')
```

```
        figureTuningCurveMethod2;
```

```
        disp(['dprime is ', num2str(result)]);
```

```
        selection1 = input('Then, we will now change turningcurve function from  $f = 1./(1 + \exp(-(x-50)/10))$  to  $f = n./(1 + \exp(-(m*x-50)/10))$ , where  $n>1, m>1$ \n Here as an example we set  $n=5, m=4$ .\n Are you ready to see the graph and dprime now? Press 4 to see.\n');
```

```
        switch selection1
```

```
            case 4
```

```
                AfterTCchange;
```

```
                [x115, s115] = generateWeirdData1(15,1000);
```

```
                [x125, s125] = generateWeirdData1(25,1000);
```

```
                result1 = dprimef (x115, x125, s115, s125);
```

```
                disp(['dprime is ', num2str(result1), '--> Much bigger, right? --> More discriminable.']);
```

```
            otherwise
```

```
                disp('You coward, you should have pressed 4!');
```

```
        end
```

```
    case 2
```

```
        disp('You made a great choice!\n Now first look at original graph and d-prime between stimulus 15 and 25.\n')
```

```
        figureTuningCurveMethod2;
```

```
        disp(['dprime is ', num2str(result)]);
```

```
selection2 = input('Then, we will now increase maxrate from 300 to
n\n Here as an example we set n=600.\n Are you ready to see the graph and
dprime now? Press 5 to see.\n');
switch selection2
    case 5
        AfterMRchange;
        [x215, s215] = generateWeirdData2(15,1000);
        [x225, s225] = generateWeirdData2(25,1000);
        result2 = dprimef (x215, x225, s215, s225);
        disp(['dprime is ', num2str(result2), '--> Much bigger, right?
--> More discriminable.']);
    otherwise
        disp('You coward, you should have pressed 5!');
    end
case 3
    disp('You made a great choice!\n Now first look at original graph and
d-prime between stimulus 15 and 25.\n')
    figureTuningCurveMethod2;
    disp(['dprime is ', num2str(result)]);
    selection3 = input('Then, we will now increase tau from 100 to n.\n
Here as an example we set n=300\n Are you ready to see the graph and dprime
now? Press 6 to see.\n');
    switch selection3
        case 6
            AfterTauchange;
            [x315, s315] = generateWeirdData3(15,1000);
            [x325, s325] = generateWeirdData3(25,1000);
            result3 = dprimef (x315, x325, s315, s325);
            disp(['dprime is ', num2str(result3), '--> Much bigger, right?
--> More discriminable.']);
        otherwise
            disp('You coward, you should have pressed 6!');
        end
    otherwise
        disp('Please make the right choice, alright?')
    end
end
```


b) generateNoisyDataGeneral

```
% This code will generate a spiking response to an input based on a tuning
% curve and assuming Poisson firing. The response adapts with a fixed
% timecourse.
%
% The interesting part of this function is that it is based on the x, which
% is stimulus input, and y, which is trails, that you input in the (). So
% it is a general form of functions like generateNoisyData5090 and
% generateNoisyData7090.
%

%function [Meantemp] = generateNoisyDataGeneral(x, y)
function [Meantemp, Stdtemp] = generateNoisyDataGeneral(x, y)
%function [Meantemp] = generateNoisyDataGeneral(x)

sininput = x; % stimulus input
ntrials = y; % number of trails
% ntrials = 1000; % for debugging

% sininput = 10 % for debugging
% ntrils = 20 % for debugging

maxrate = 300; % 30 Hz max firing rate

rate = maxrate*tuningCurve(sininput);
tau = 100;      % adaptation time constant in msec
nmsec = 300;    % number of milliseconds to record for
times= 1:nmsec; % time units

spiketraittemp = zeros(ntrials,nmsec);      % set up output data

ratecurve = .001*rate*exp(-times/tau); % adapting rate function

for j = 1:ntrials;

    for i = 1:nmsec;

        if(rand(1)<ratecurve(i)),
            spiketraittemp(j,i) = 1;
        end;
```

```
end;  
  
end;  
  
Sumtemp = sum(spiketraintemp, 2);  
Meantemp = mean(Sumtemp);  
Stdtemp = std(Sumtemp);  
%stdall(x) = std(Sumtemp)  
end
```

c) figureTuningCurveMethod2

```
% So we use a function that calculate the mean and standard deviation based  
% on the given stimulus input  
  
for k = 1:5:400  
    [meanall(k),stdall(k)] = generateNoisyDataGeneral(k,1000);  
end  
  
stmls = 1:5:400;  
  
figure;  
plot(stmls,meanall(stmls))  
title('Tuning Curve')  
xlabel('Stimulus')  
ylabel('Response')  
errorbar(stmls, meanall(stmls), stdall(stmls));
```

d) figureTuningCurveMethod3

```
% The cool part is that I created a 3D matrix, so one dimension is the  
% spikes, one is trails, one is stimulus. The I reduce the dimension using  
% methods like mean(), sum(), and std(), finally I squeeze the matrix into  
% two functions of mean and std which can be plotted.
```

```
alldata = zeros(ntrials, nmsec, 400);

ntrials = 1000;
nmsec = 300; % number of milliseconds to record for
times= 1:nmsec; % time units
tau = 100; % adaptation time constant in msec
%nbins = 400;

for k = 1:80 % Because I want to plot out all stimulus 1:5:400, thus third
dimension is 80.
    x1 = 5*k;

    maxrate = 300; % 30 Hz max firing rate

    rate = maxrate*tuningCurve(x1);
    ratecurve = rate*exp(-times/tau)*.001; % adapting rate function

    for j = 1:ntrials;
        for i = 1:nmsec;
            if(rand(1)<ratecurve(i)),
                alldata(j,i,k) = 1;
            end
        end
    end;
end

allmean = mean(squeeze(sum(alldata,2)),1);
allstd = std(squeeze(sum(alldata,2)),1);

stmls = 1:80;

figure;
plot(5*stmls,allmean(stmls))
title('Tuning Curve')
xlabel('Stimulus')
ylabel('Response')
errorbar(5*stmls, allmean(stmls), allstd(stmls))
```

For all other codes, they are all attached in the zip file waiting to be reviewed.

Bibliography:

Driscoll, T. A. (2009). *Learning MATLAB*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Pärt-Enander, E. (1996). *The MATLAB handbook*. Harlow, England: Reading, Mass.

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2004). *Digital Image processing using MATLAB*. Upper Saddle River, N.J: Pearson Prentice Hall.

Feng, J. (2004). *Computational neuroscience: Comprehensive approach*. Boca Raton: Chapman & Hall/CRC.

Acknowledgement:

Thank Prof. Adrienne Fairhall for giving us insightful lectures about the dynamic field of computational neuroscience and setting challenging questions for us to explore!

Thank Dr. Alison Weber for clarifying the concepts vividly and giving me instrumental suggestions and guidance throughout the problem solving selflessly with her busy time!

Thank my friends in NBIO 303 who support me as a freshman without MATLAB experience with encouragement and tolerance!

Thank University of Washington for giving us the platform to scientifically explore problems and subjects!

I will continue the voyage of exploring the infinite realm of computational neuroscience in my academic career fearlessly.

Baihan Lin
January 2014