Simple/Complex Cell STA

This experiment consists of 2 parts. The first part generates spike trains using different methods and compare them. The second part is the implementation of an Linear-Nonlinear-Poisson cascade model. I implemented the Receptive Field of both a simple cell and a complex cell then simulates their responses using grating images and white noise images. I also used enough white noise images to get the estimated STA.

PART I: Spike Trains with Bernoulli and Exponential ISI

1. Deciding Parameters

In order to simulate a spike train of 100 seconds with 10 spikes per second, I choose the probability p = 0.01 for each time bin of 1 ms. This means that the mean of ISI's is 0.1 second.

2. Bernoulli Process

```
function bnlspks = bnl(time)
% Return Bernoulli binary sequence in 1ms time bins of given time
    p = 0.01;            % Probability to spike
    bnlspks = zeros(1, time * 1000);
    for t = 1 : time * 1000
        bnlspks(1, t) = binornd(1, p);      % Binomial randomize result for each bin
    end
end
```

3. Exponential ISI

```
function isi = expISI(time)
% Return exponential random ISI's to fill in given time
% Time has unit in seconds
    p = 10;
    miu = 1 / p;        % Mean of ISI
    temp = zeros(1, round(time / miu * 2));    % Make it enough long to hold the ISI’s
    tpassed = 0;
    count = 0;      % Count how many ISI's data is needed to add up to given time

    while tpassed <= time
        count = count + 1;
        temp(1, count) = exprnd(miu);
        tpassed = tpassed + temp(1, count);
    end

    isi = zeros(1, count);        % Now we know the size to create the return vector
    for i = 1 : count
        isi(1, i) = temp(1, i);
    end
end
```

4. Spike Count

The following function *spkcnt()* generate the 100s long spike train of length 1s each, and outputs the 1-second spike count sequence. The parameter states which method we are going to use. 1 for Bernoulli, 2 for exponential ISI, and 3 for Poisson.

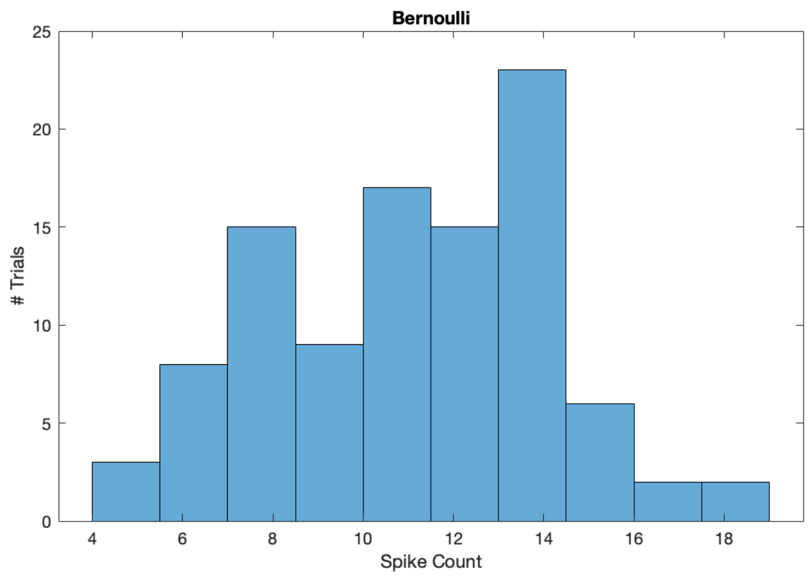
```
function spkc = spkcnt(num)
% Return the spike count of 100 trials each is of length 1 second
% Parameter num gives the method number

    spkc = zeros(1, 100);

    if num == 1             % Use Bernoulli generated sequence
        bnlspks = bnl(100);
        for i = 1 : 100
            for j = 1 : 1000
                spkc(1, i) = spkc(1, i) + bnlspks(1, j + 1000 * (i - 1));
            end
        end

    elseif num == 2         % Use exponential ISI
        isi = expISI(110);  % Generate enough ISI for 110 seconds just in case
        countt = 1;         % Count trail
        time = 0;         % Count time passed
        for i = 1 : length(isi)
            if time > 1         % Reaches next trial
                if countt < 100
                    countt = countt + 1;
                    spkc(1, countt) = spkc(1, countt) + 1;
                    time = time - 1;
                    continue
                else
                    break
                end
            end
            spkc(1, countt) = spkc(1, countt) + 1;     % Each ISI signals one spike
            time = time + isi(1, i);        % Increment time passed
        end

    elseif num == 3                 % Poisson
        spkc = poissrnd(10, [1, 100]);
    end
end
```

The followings are the histogram analysis for the three methods.

```
bnl = spkcnt(1);
histogram(bnl, 10);
exp = spkcnt(2);
histogram(exp, 10);
poi = spkcnt(3);
histogram(poi, 10);
```

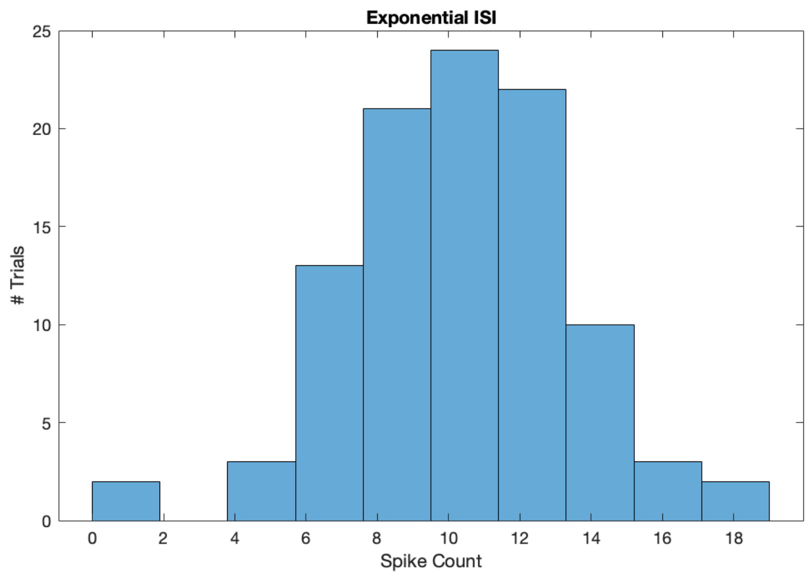
<u>Bernoulli</u>

FF = mean / variance = 1.1180



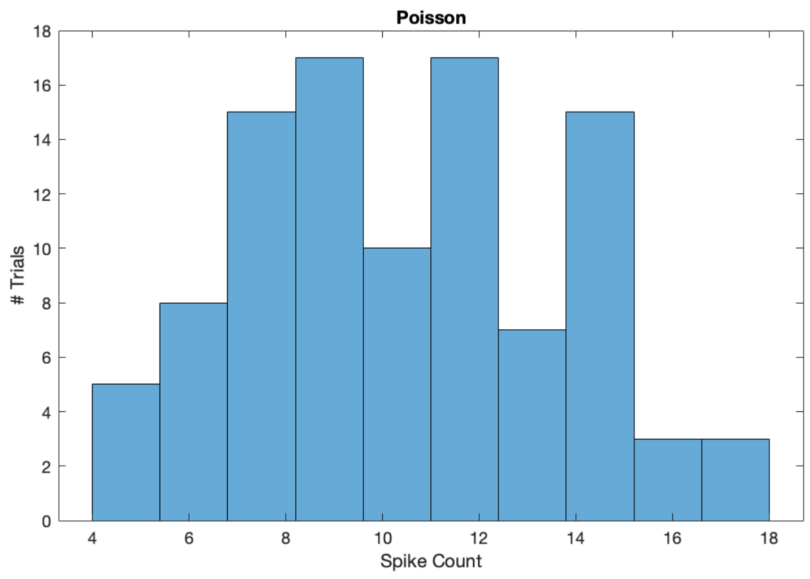
Exponential ISI

FF = mean / variance = 0.9771

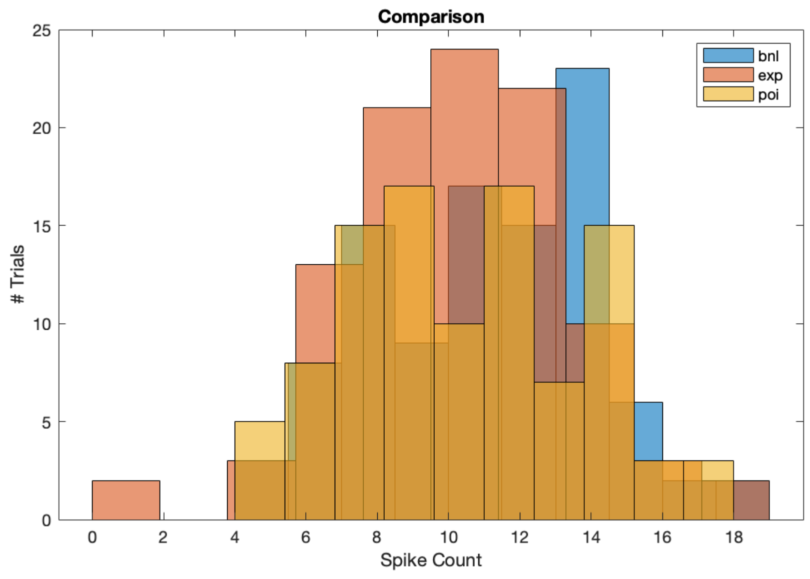


Poisson Distribution

FF = mean / variance = 0.9998



Comparison



Thus the 3 methods all lead to Fano Factor approximately 1, as expected from a Poisson distribution. From the superimposed histogram we can see that they are overlapping each other at mean value about 10 spike/second.

PART II: Simple/Complex Cell Simulation and STA Analysis

In this part, I implemented a RF for a simple cell and a complex cell. The image I used all have dimensions 50 x 50 of size ±5 degrees along each boundary, with 0.2 deg x 0.2 deg for each pixel. I used the following Gabor function:

$$RF(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}\right)\cos\left(kx - \phi\right)$$

where $\sigma_x = 1$ deg, $\sigma_y = 2$ deg, k = 1 / 0.56 deg.

1. Simple Cell Vertical RF

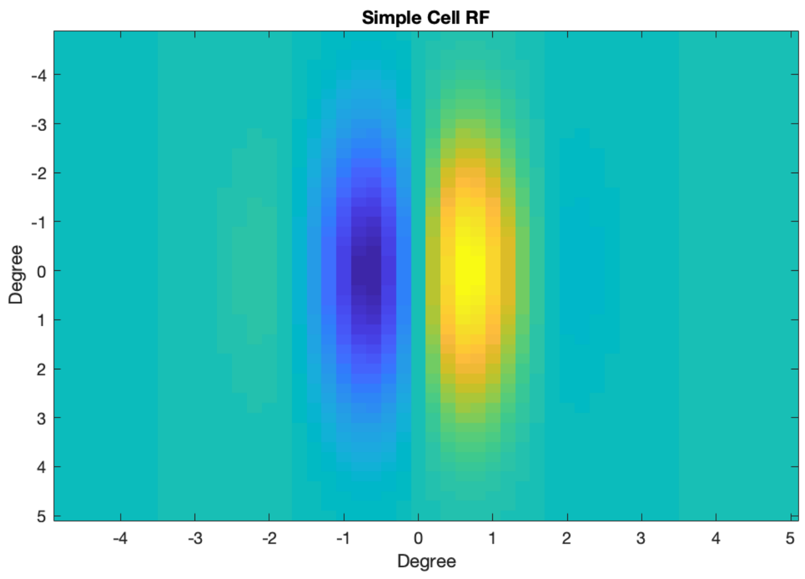
I used phase $\phi = \pi / 2$ and a half-wave squaring in this model.

```
function rfval = getRFval(x, y, phase)
% Get RF value at specific point in the RF
    sigX = 1;
    sigY = 2;
    k = 1 / 0.56;
    part1 = 1 / (2 * pi * sigX * sigY);
    part2 = exp(0 - x^2 / (2 * sigX^2) - y^2 / (2 * sigY^2));
    part3 = cos(k * x - phase);
    rfval = part1 * part2 * part3;
end

function rf = getSimpRF(phase)
% Get RF of degrees from -5 to 5 with pixels of 0.2 * 0.2 deg^2
    rf = zeros(50);
    for i = 1 : 50
        for j = 1 : 50
            x = -5 + i * 0.2;
            y = -5 + j * 0.2;
            rf(j, i) = getRFval(x, y, phase);
        end
    end
end
```

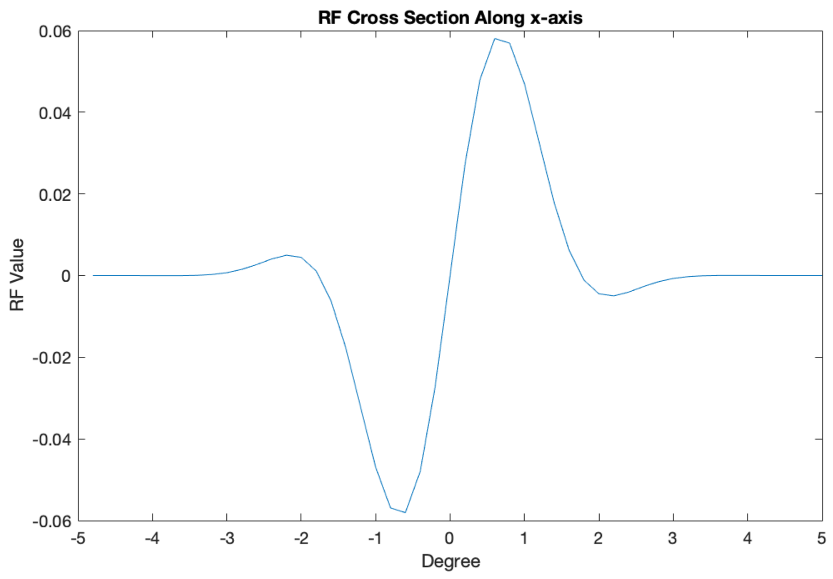
Use the following script to plot.

```
rf = getSimpRF(pi/2);
imagesc([-4.8:0.2:5], [-4.8:0.2:5], rf);
plot([-4.8:0.2:5], rf(24,:));
```

<u>2D Image</u>



<u>Cross Section along X-axis</u>



2. Mean Spike Rate in Response to Grating Image

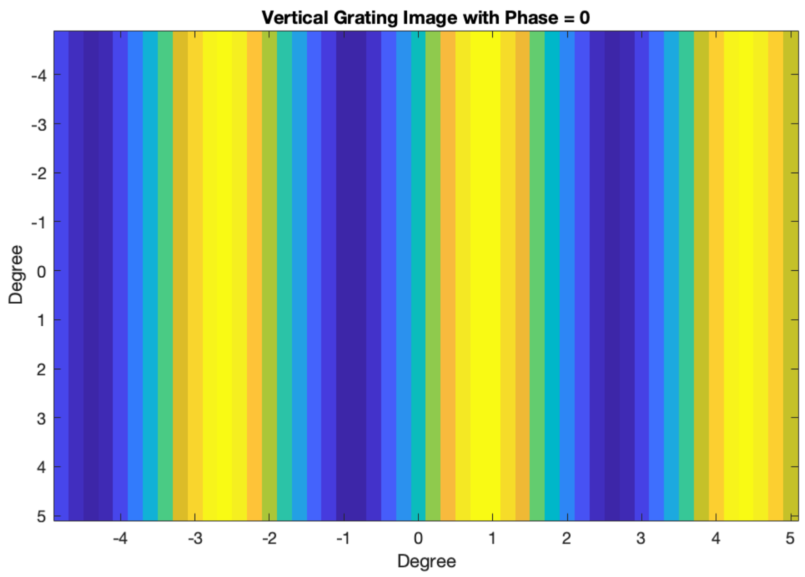
For the vertical grating image, I used the following equation as a function of phase α:

$$I(x, y) = \sin\,(k(xcos\theta + ysin\theta) - \alpha)$$

where θ is the orientation angle.

```
function imval = getImval(x, y, phase, theta)
% Get image value at given point with given phase and orientation
% Theta = 0 if image is vertical
    k = 1 / 0.56;
    imval = sin(k * (x * cos(theta) + y * sin(theta)) - phase);
end

function image = getImage(phase, theta)
% Get image of degrees from -5 to 5 with pixels of 0.2 * 0.2 deg^2
% Of given phase and orientation
    image = zeros(50);
    for i = 1 : 50
        for j = 1 : 50
            x = -5 + i * 0.2;
            y = -5 + j * 0.2;
            image(j, i) = getImval(x, y, phase, theta);
        end
    end
end
```

The following is a vertical grating image with $\alpha = 0$.



The followings are the half-wave squaring and full wave-squaring function I used.

```
function halfsqR = halfsq(r)
% Return the half squared RFt * I
    if r <= 0
        halfsqR = 0;
    else
        halfsqR = r^2;
    end
end
function fullsqR = fullsq(r)
% Return the full square of r
    fullsqR = r^2;
end
```

Note that we want:

$$r = r_0 f(RF^t \circ I)$$

where f is either half-wave squaring or full-wave squaring, and r_0 is the scale factor. So the next function *meanR()* calculates the unscaled spike rate responded by the RF on the given image, followed by a *scale* function to scale max spike rate to 50Hz.

```
function r = meanR(rf, image, sqnum)
% Calculate the unscaled mean spike rate from given RF and image

    colRF = reshape(rf.', [], 1);       % Turn RF into column vector
    colI = reshape(image.', [], 1);     % Turn I into column vector
    tcolRF = colRF.';                   % colRF transpose

    if sqnum == 1
        r = halfsq(tcolRF * colI);
    else
        r = fullsq(tcolRF * colI);
    end
end
```

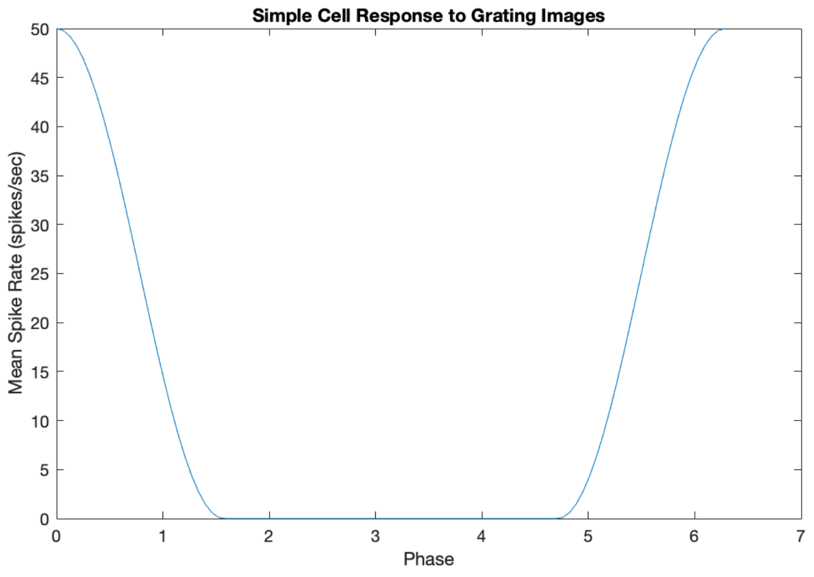
```
function scaledR = scale(r)
% Scale mean spike rate sequence r to 50 max
    max = r(1, 1);

    for rate = r
        if rate > max
            max = rate;
        end
    end

    r0 = 50 / max;
    scaledR = r0 * r;
end
```

Finally, I have a function *getGratR()* to generate sequence of scaled spike rate in response to grating images from the given phase sequence.

```
function r = getGratR(rf, alpha, sqnum)
% Return sequence of mean spike rate with given RF
% In response to grating images with given sequence of phases
% sqnum states the squaring function we use
    theta = 0;
    r = zeros(1, length(alpha));
    for i = 1 : length(alpha)
        image = getImage(alpha(1, i), theta);   % Generate grating image
        r(1, i) = meanR(rf, image, sqnum);
    end
    r = scale(r);
end
```

Now we can use this function to plot with α from 0 to 2π.

```
alpha = [0 : 2 * pi / 99 : 2 * pi];
r = getGratR(rf, alpha, 1);
plot(alpha, r);
```

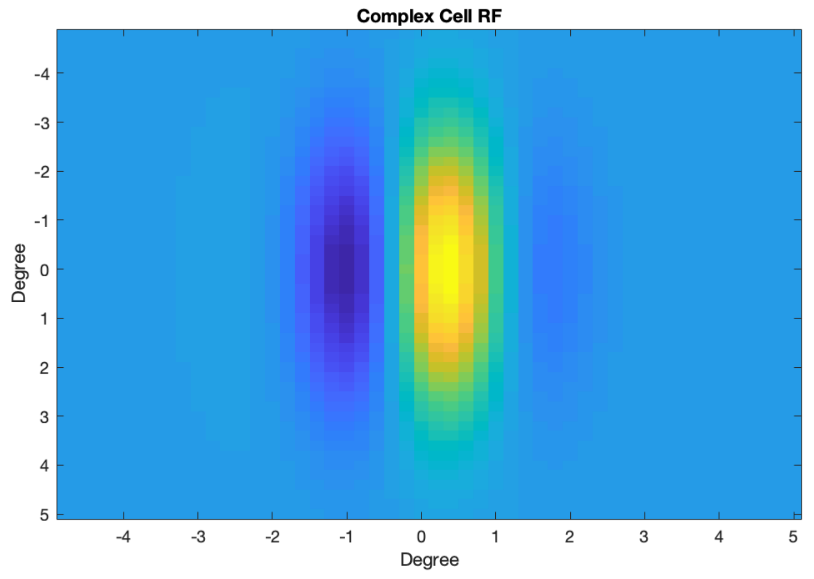


3. Complex Cell RF

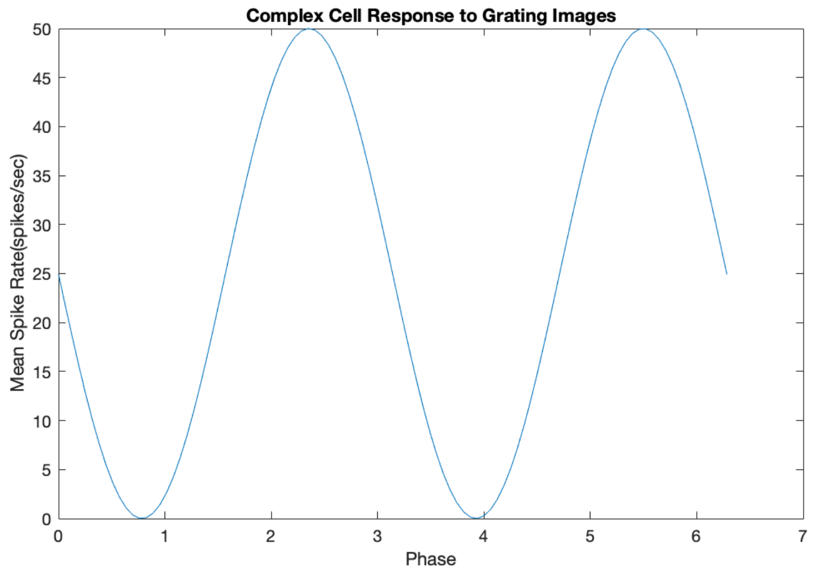
I used simple cells $\phi = \pi / 2$ and $\phi = 0$ to form the complex cell, and changed the half-wave squaring to a full-wave squaring. The following is the function to get a complex cell.

```
function compRF = getCompRF(phase1, phase2)
% Return the complex cell RF with the 2 given phases
    compRF = getSimpRF(phase1) + getSimpRF(phase2);
end
```

Here's the 2D image of a complex cell with $\phi = \pi / 2$ and $\phi = 0$.

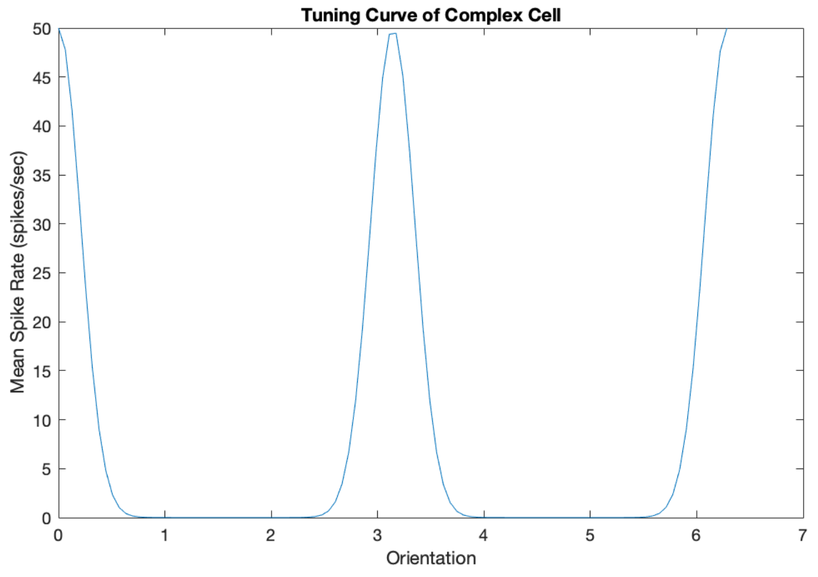


Use the same function as in section 2, and we get the plot of mean spike rate in response to the grating images as a function of phase α from 0 to 2π, on the complex cell RF we now have. We also use a full-wave squaring now.



Now, in order to plot the Tuning Curve, I used the following function and script. I fixed phase α = 0 and varied orientation θ from 0 to 2π.

```
function tcurve = getTuning(rf, theta, sqnum)
% Return the Tuning curve on the given RF
% Of grating images of given theta(orientation) sequence
% sqnum states the squaring function used
    alpha = 0;
    tcurve = zeros(1, length(theta));
    for i = 1 : length(theta)
        image = getImage(alpha, theta(1, i));   % Generate grating image
        tcurve(1, i) = meanR(rf, image, sqnum);
    end
    tcurve = scale(tcurve);
end

theta = [0 : 2 * pi / 99 : 2 * pi];
tcurve = getTuning(rf, theta, 2);
plot(theta, tcurve);
```



We can see that the spike rate is at peak when θ is a multiple of π.

4. Reverse Correlation/STA

In the last section I will perform the Poisson spiking estimation based on white noise images and calculate the reverse correlation (Spike Triggered Average) of both the simple and complex cells. Before I do the simulation, I want to show the functions I used to perform the task.

First I have a function to generate any number of white noise images.

```
function images = genImages(num)
% Generate white noise 50 * 50 images of of given number
    images = zeros(num, 50, 50);
    for i = 1 : num
        images(i, :, :) = rand(50, 50);
    end
end
```

I have the next function to calculate a sequence of mean spike rates based on the sequence of input images.

```
function r = meanRseq(rf, images, sqnum)
% Return the sequence of mean spike rate of given RF
% in response to the given image sequence
% sqnum states the method of squaring
    r = zeros(1, length(images));
    for i = 1 : size(images, 1)
        image = zeros(50, 50);
        for row = 1 : 50
            for col = 1 : 50
                image(row, col) = images(i, row, col);
            end
        end
        r(1, i) = meanR(rf, image, sqnum);
    end
    r = scale(r);
end
```

Next, I get the Poisson spiking sequence based a sequence of mean spike rates. I scaled the model such that the average spike count per image is about 0.2 spikes.

```
function spikes = getSpks(r)
% Return the poisson spike sequence with given spike rate sequence
    spikes = zeros(1, length(r));
    for i = 1 : length(r)
        spikes(1, i) = poissrnd(r(1, i) / 100); % Scales to about 0.2 spikes per image
    end
end
```

To calculate the STA, I choose the images that lead to spike count >= 1 and get the average. The following function does just that.

```
function sta = getSTA(spikes, images)
% Get the estimated STA from given spike sequence
% Corresponding to given white noise image sequence
    sta = zeros(50, 50);
    count = 0;
    for i = 1 : length(spikes)
        if spikes(1, i) >= 1        % Choose only images which leads to spikes >= 1
            image = zeros(50, 50);
            for row = 1 : 50
                for col = 1 : 50
                    image(row, col) = images(i, row, col);
                end
            end
            count = count + 1;
            sta = sta + image;
        end
    end
    sta = sta / count;
end
```

Since the image sequence can be very big, I use another function to get the STA's on 1000 images each time and make the estimation better and better as more trials are performed. It also records the correlation coefficient with the actually RF at the end of each 1000-image trial.

```
function [sta, cor] = avgSTA(rf, tnum, sqnum)
% Generate best approximation STA on the given RF
% Perform given number of trials
% Each trial uses 1000 white noise images
% Also generates sequence of correlation coefficients up to each trial

    cor = zeros(1, tnum);
    sta = zeros(50, 50);
    count = 0;
    for i = 1 : tnum
        images = genImages(1000);     % Generate white noise images
        r = meanRseq(rf, images, sqnum);       % Calculate mean spike rate for each
        spikes = getSpks(r);            % Get spike sequence
        sta = sta + getSTA(spikes, images);    % Get STA for each trial and sum them up
        count = count + 1;
        temp = sta / count;
        cor(1, i) = corr(rf(:), temp(:));    % Correlation coefficient up to now
    end
    sta = sta / count;          % Calculate the average
end
```

In order to simulate, simply use the following script to get the STA. I used 3.000.000 white noise images for both the simple cell and the complex cell.

```
[simpSTA, simpCor] = avgSTA(simpRF, 3000, 1);
imagesc([-4.8:0.2:5], [-4.8:0.2:5], simpSTA);
[compSTA, compCor] = avgSTA(compRF, 3000, 2);
imagesc([-4.8:0.2:5], [-4.8:0.2:5], compSTA);
```

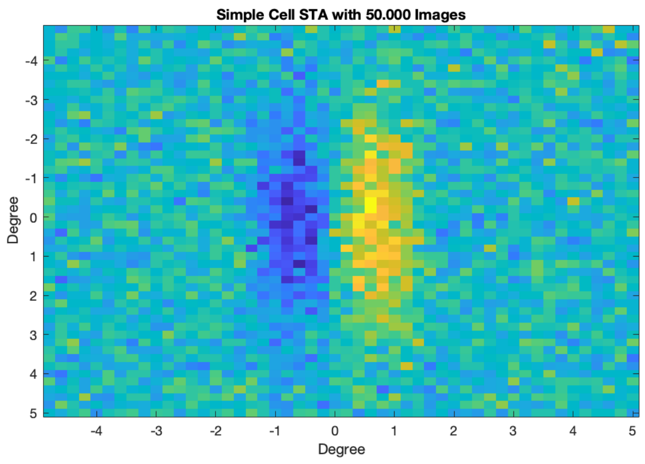
Simple Cell

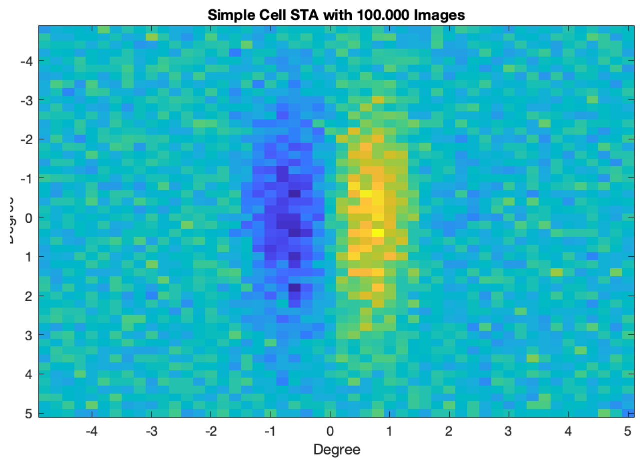
50.000 images: corr = 0.7066

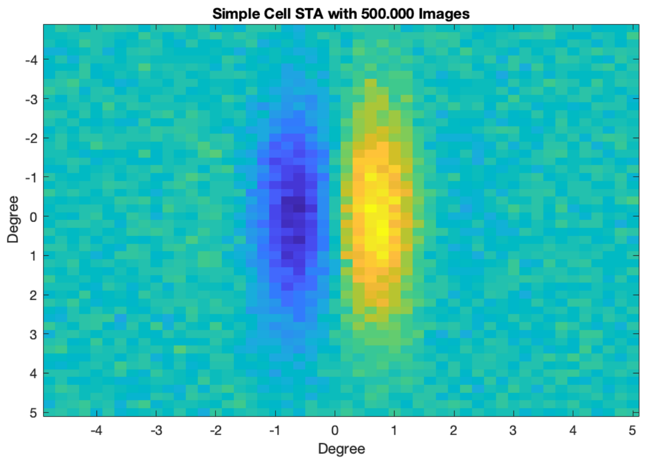
100.000 images: corr = 0.8170

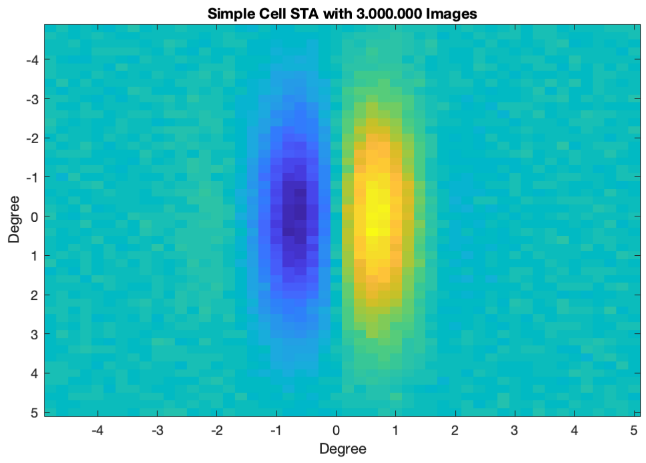
500.000 images: corr = 0.9574

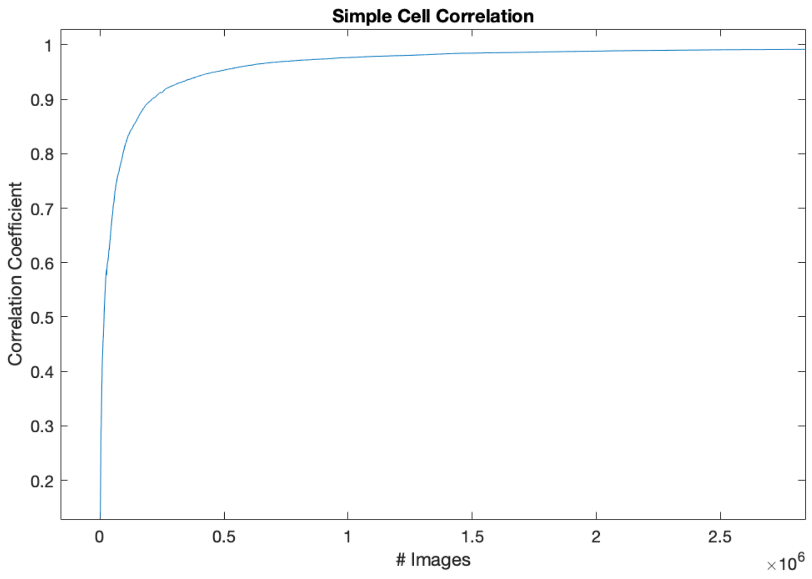
3.000.000 images: corr = 0.9921











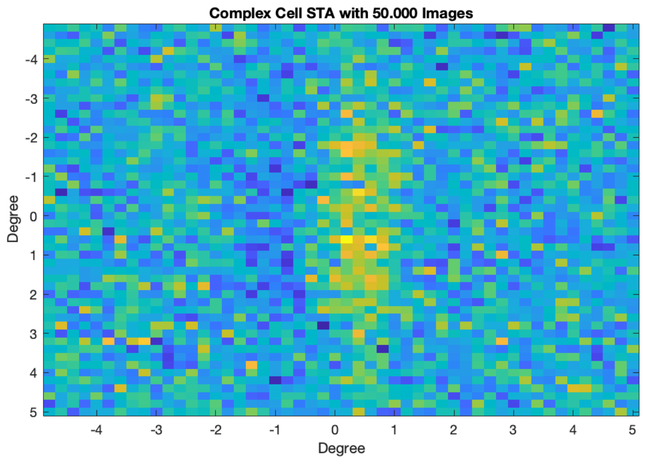
Complex Cell

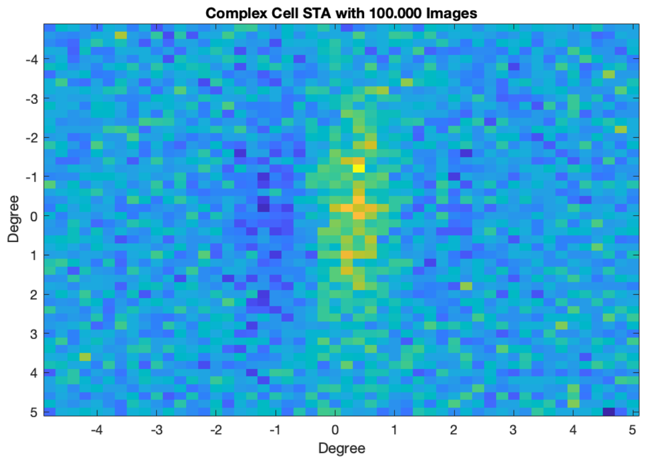
50.000 images: corr = 0.4044

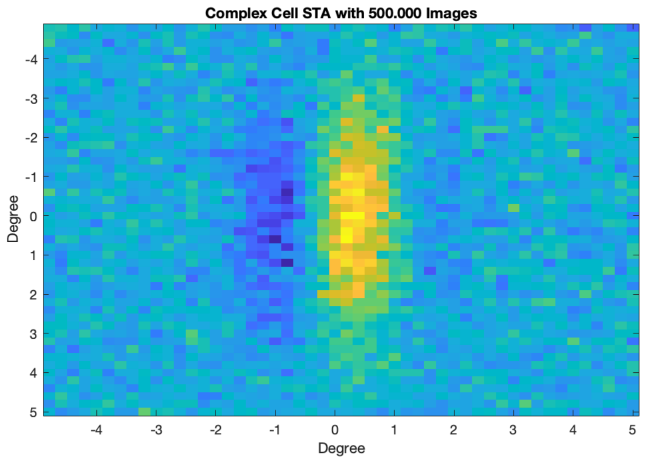
100.000 images: corr = 0.5250

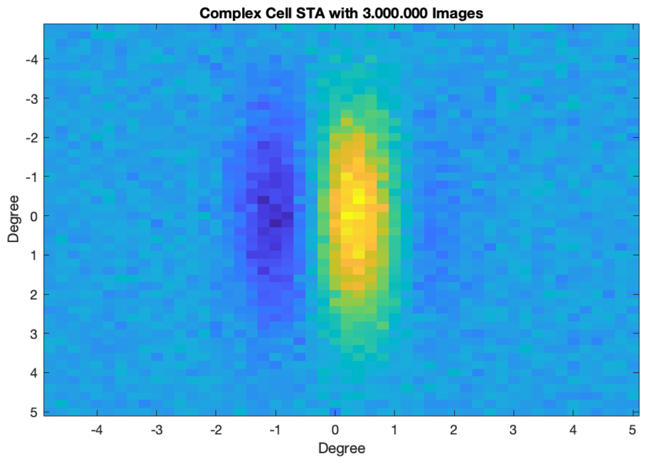
500.00 images: corr = 0.8044

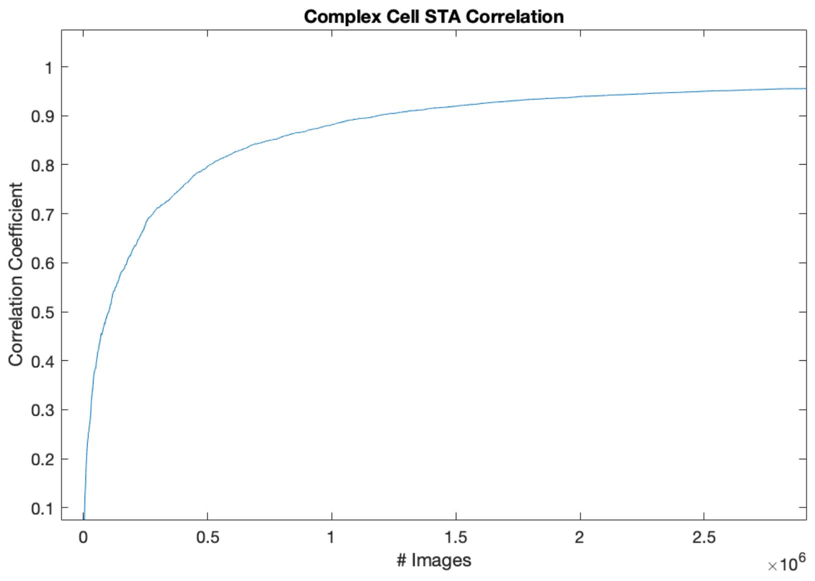
3.000.000 images: corr = 0.9596











From the simulations, we can see that the correlation of the STA and RF get asymptote to 1 as more and more white noise images are given as input. It can also be found that the correlation coefficient in the complex cell is rising slower compared to that of the simple cell. This might be due to the squaring difference. Maybe the full-wave squaring are making the STA harder to estimate the original RF.