EE243 HW #3

(1) Process rate $= \lambda$

a. Dropping spikes probability $= p$

Keeping spikes probability $= 1-p$

Show M is poisson $((1-p)\lambda s)$

$Pr(M=m) \rightarrow$ We want to find this

$$Pr(M=m) = \sum_{n=0}^{\infty} \underbrace{Pr(M=m \mid N=n)}_{\text{bernoulli}} \underbrace{Pr(N=n)}_{\text{poisson}}$$

$$\binom{n}{m}(1-p)^m (p)^{n-m} \qquad \frac{e^{-\lambda s}(\lambda s)^n}{n!}$$

$$= \sum_{n=0}^{\infty} \frac{\cancel{n!}}{m!(n-m)!}(1-p)^m (p)^{n-m} \frac{e^{-\lambda s}(\lambda s)^n}{\cancel{n!}}$$

$$= \sum_{n=0}^{\infty} \frac{(1-p)^m (p)^{n-m}}{m!(n-m)!} e^{-\lambda s}(\lambda s)^n$$

When $m > n$, $Pr(M=m \mid N=n)$ is $\emptyset$ so we can take those values out.

$$\sum_{n=m}^{\infty} \frac{(1-p)^m (p)^{n-m}}{m!(n-m)!} e^{-\lambda s}(\lambda s)^n$$

2
$=2$
$=3$
$2^2 \cdot 3^2 = 36$
$(2 \cdot 3)^2 = 36$

$$\frac{(1-p)^m}{m!} e^{-\lambda s} \sum_{n=m}^{\infty} \frac{(\lambda s)^n (p)^{n-m}}{(n-m)!} = \frac{(1-p)^m e^{-\lambda s}(\lambda s)^m}{m!} \sum_{n=0}^{\infty} \frac{(\lambda s)^{n-m}(p)^{n-m}}{(n-m)!}$$

$$= \frac{(1-p)^m e^{-\lambda s}(\lambda s)^m}{m!} \sum_{n=0}^{\infty} \frac{(\lambda s \cdot p)^{n-m}}{(n-m)!} = \frac{(\lambda s(1-p)^m)\, e^{-\lambda s}}{m!} \cdot e^{(\lambda s)p}$$

$$= \frac{(\lambda s(1-p))^m}{m!} e^{\lambda s} e^{-\lambda s p} = \frac{(\lambda s(1-p))^m}{m!} e^{\lambda s(1-p)} = \text{poisson } (\lambda s(1-p))$$

b. The rate of the process is $(1-p)\lambda$

c. The distribution is a poisson distribution with parameters $p \to$ probability of dropping, $\lambda \to$ rate and $T \to$ the time interval.

hw3p2-Copy1

May 4, 2018

0.1 Homework 3, Problem 2 on homogeneous Poisson processes

ECE C143A/C243A, Spring Quarter 2018, Prof. J.C. Kao, TAs T. Monsoor, X. Jiang and X. Yang.

0.2 Background

The goal of this notebook is to model a neuron as a homogeneous Poisson processes and evaluate its properties. We will consider a simulated neuron that has a cosine tuning curve described in equation (1.15) in *TN* (*TN* refers to *Theoretical Neuroscience* by Dayan and Abbott.)

$$\lambda(s) = r_0 + (r_{\max} - r_0)\cos(s - s_{\max})$$

where λ is the firing rate (in spikes per second), s is the reaching angle of the arm, $s_{\max}$ is the reaching angle associated with the maximum response $r_{\max}$, and r_0 is an offset that shifts the tuning curve up from the zero axis. This will be refered as tuning equation in the following questions.

Let $r_0 = 35$, $r_{\max} = 60$, and $s_{\max} = \pi/2$.

Note: If you are not as familiar with Python, be aware that if 1 is of type `int`, then 1 / a where a is any `int` greater than 1 will return 0, rather than a real number between 0 and 1. This is because Python will return an `int` if both inputs are `ints`. If instead you write 1.0 / a, you will get out the desired output, since 1.0 is of type `float`.

In [1]:
```
"""
ECE C143/C243 Homework-3 Problem-2

"""
import numpy as np
import matplotlib.pyplot as plt
import nsp as nsp # these are helper functions that we provide.
import scipy.special

# Load matplotlib images inline
%matplotlib inline

# Reloading any code written in external .py files.
%load_ext autoreload
%autoreload 2
```

0.2.1 (a) (6 points) Spike trains

For each of the following reaching condition ($s = k \cdot \pi/4$, where $k = 0, 1, \ldots, 7$), generate 100 spike trains according to a homogeneous Poisson process. Each spike train should have a duration of 1 second. You can think of each of each spike train sequence as a trial. Therefore, we generate 100 trials of the neuron spiking according to a homogeneous Poisson Process for 8 reach directions.

Your code for this section should populate a 2D `numpy` array, `spike_times` which has dimensions `num_cons` $\times$ `num_trials` (i.e., it is 8×100). Each element of this 2D numpy array is a numpy array containing the spike times for the neuron on a given condition and trial. Note that this array may have a different length for each trial.

e.g., `spike_times.shape` should return `(8, 100)` and `spike_times[0,0]` should return the spike times on the first trial for a reach to the target at 0 degrees. In one instantiation, our code returns that `spike_times[0,0]` is:

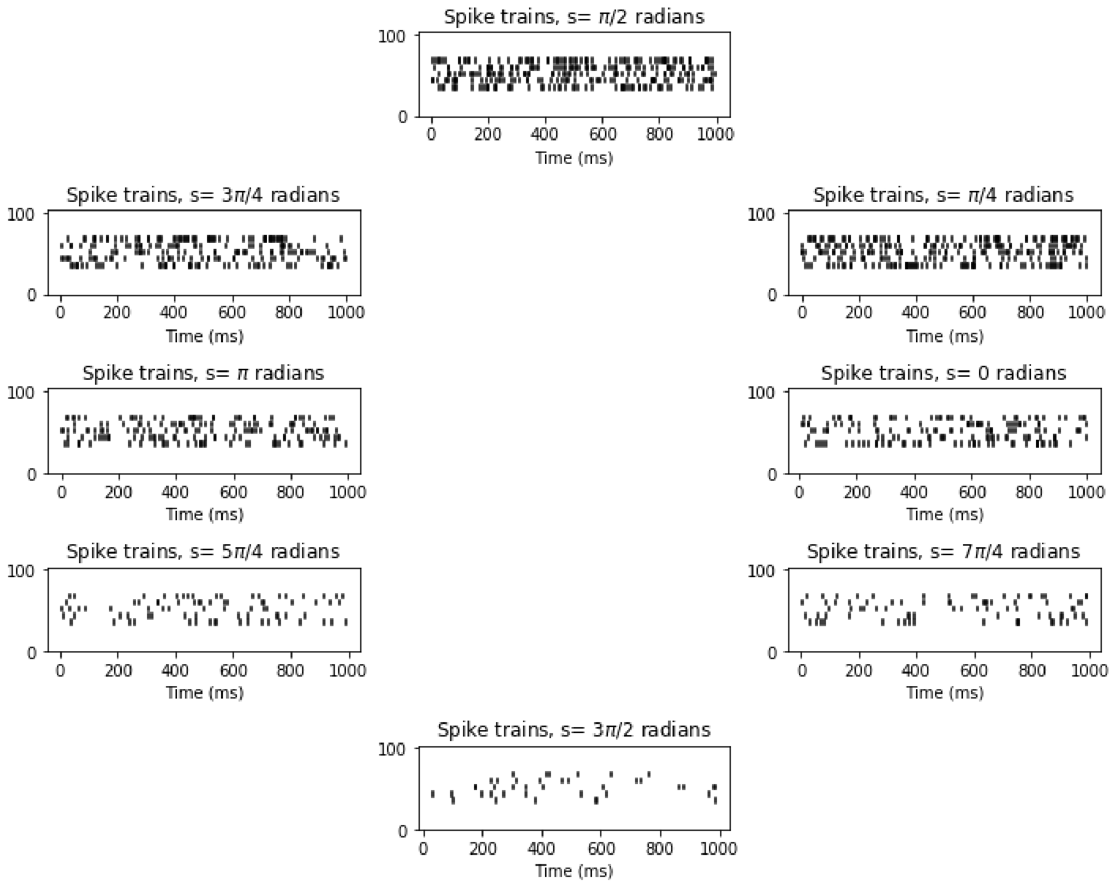
```
array([   0.        ,    5.94436383,   10.85691999,   26.07821145,
50.02836141,   67.417219  ,   74.2948356 ,  119.19210112,        139.41789878,
176.59511596,  244.40788916,  267.3643421 ,        288.42590046,  324.3770265 ,
340.26911602,  407.75730065,        460.76250631,  471.23773964,  489.41659607,
514.60180131,        548.71822693,  565.6036432 ,  586.20557118,  601.11595447,
710.37485206,  751.60837895,  879.93536952,  931.26983289,        944.1130483 ,
949.38455374,  963.22509374,  964.67365483,        966.3865719 ,  974.3657882 ,
987.25729081])
```

Of course, this varies based off of random seed.

In [2]:
```python
## 2a
bin_width = 20                                  # (ms)
s = np.arange(8)*np.pi/4                        # (radians)
num_cons = np.size(s)                            # num_cons = 8 in this case, number of direc
r_0 = 35 # (spikes/s)
r_max = 60 # (spikes/s)
s_max = np.pi/2 # (radians)
T = 1000 #trial length (ms)
num_trials = 100 # number of spike trains to generate

tuning = r_0 + (r_max-r_0)*np.cos(s-s_max) # tuning curve
spike_times = np.empty((num_cons, num_trials), dtype=list)
for con in range(num_cons):

    for rep in range(num_trials):
        #====================================================#
        # YOUR CODE HERE:
        #   Generate homogeneous Poisson process spike trains.
        #   You should populate the np.ndarray 'spike_times' according
        #   to the above description.
        #====================================================#
        spike = nsp.GeneratePoissonSpikeTrain(T, tuning[con])
        spike_times[con, rep] = np.array(spike)
        spike_times[con,rep] = spike_times[con,rep][1:]
        pass
```

```
        #=========================================================#
        # END YOUR CODE
        #=========================================================#
```

```
In [3]: s_labels = ['0', '$\pi$/4', '$\pi$/2', '3$\pi$/4', '$\pi$', '5$\pi$/4', '3$\pi$/2', '7$\
        num_plot_rows = 5
        num_plot_cols = 3
        subplot_indx = [9, 6, 2, 4, 7, 10, 14, 12]
        num_rasters_to_plot = 5 # per condition

        # Generate and plot homogeneous Poisson process spike trains
        plt.figure(figsize=(10,8))
        for con in range(num_cons):

            # Plot spike rasters
            plt.subplot(num_plot_rows, num_plot_cols, subplot_indx[con])
            nsp.PlotSpikeRaster(spike_times[con, 0:num_rasters_to_plot])

            plt.title('Spike trains, s= '+s_labels[con]+' radians')
            plt.tight_layout()
```



0.2.2 Plotting the spike rasters.

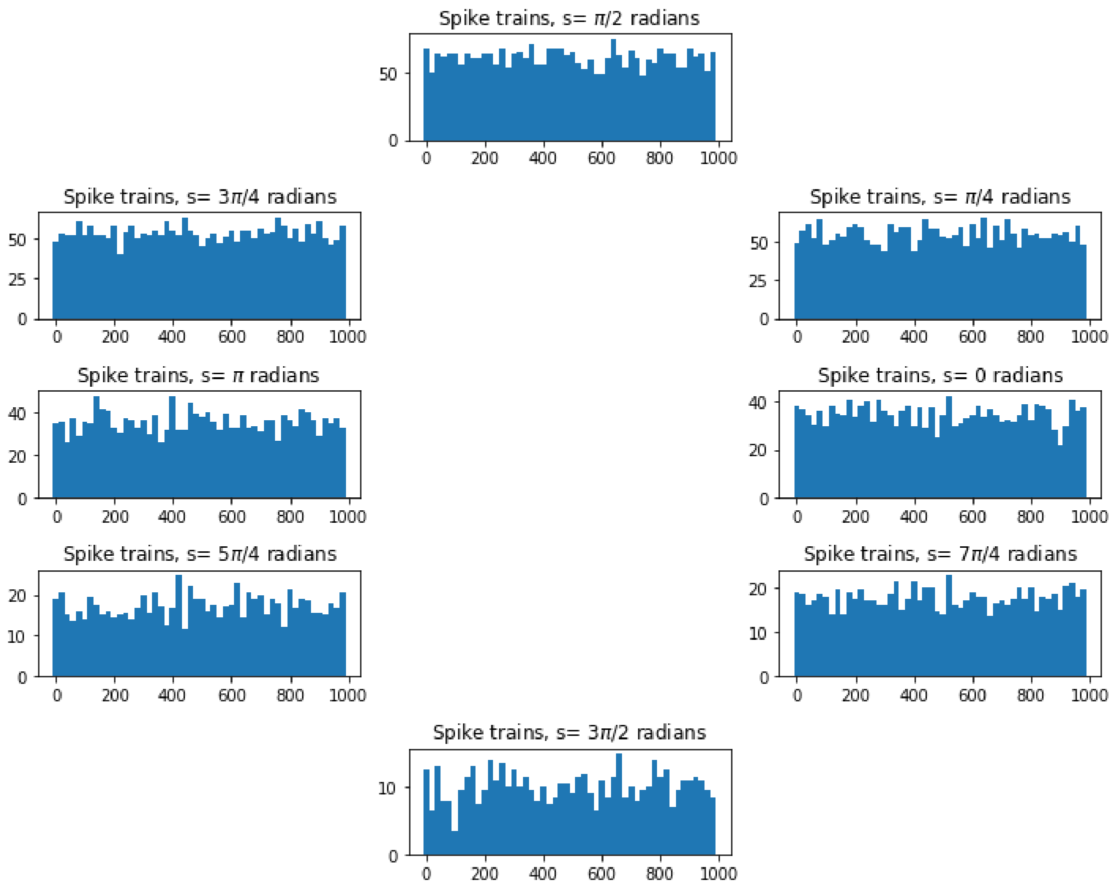
The following code plot 5 spike trains for each reaching angle in the same format as shown in Figure 1.6(A) in *TN*. You should take a look at this code to understand what it's doing. You may also want to look at the `PlotSpikeRaster` function from `nsp`.

The plots should make intuitive sense given the tuning parameters.

0.2.3 (b) (5 points) Plot spike histograms

For each reaching angle, find the spike histogram by taking spike counts in non-overlapping 20 ms bins, then averaging across the 100 trials. Plot the 8 resulting spike histograms around a circle, as in part (a). This time, as we'll allow you to represent the data as you like, you will have to also plot each histogram on your own. The spike histograms should have firing rate (in spikes / second) as the vertical axis and time (in msec, not time bin index) as the horizontal axis.

```
In [4]: ## 2b

        plt.figure(figsize=(10,8))

        for con in range(num_cons):
            plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
            #=====================================================#
            # YOUR CODE HERE:
            #   Generate and plot spike histogram for this condition
            #=====================================================#
            full_digitized = []
            bins = range(0,1001,20)
            for rep in range(num_trials):
                data = np.ndarray.tolist(spike_times[con, rep])
                full_digitized.extend(np.ndarray.tolist(np.digitize(data, bins)))
                counts = [full_digitized.count(x) for x in set(full_digitized)]
                counts = [x / 2. for x in counts]
            pass
            plt.bar(bins[0:50],counts, width=20)
            #=====================================================#
            # END YOUR CODE
            #=====================================================#
            plt.title('Spike trains, s= '+s_labels[con]+' radians')
            plt.tight_layout()
```



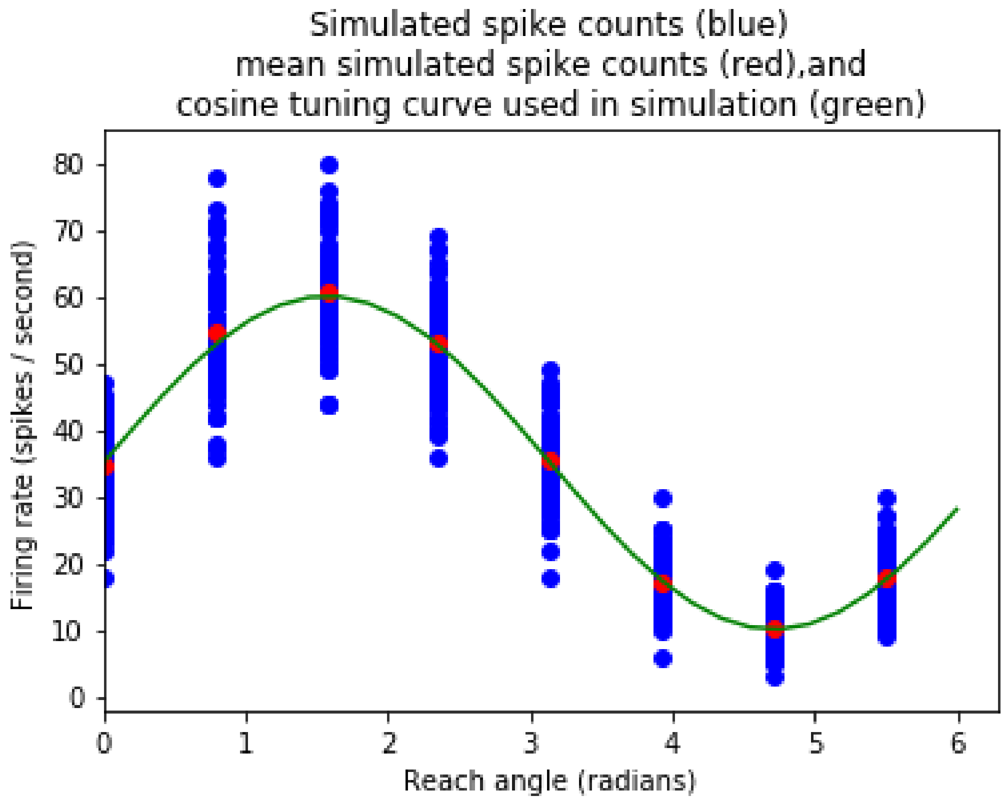
0.2.4 (c) (4 points)Tuning curve

For each trial, count the number of spikes across the entire trial. Plots these points on the axes like shown in Figure 1.6(B) in *TN*, where the x-axis is reach angle and the y-axis is firing rate. There should be 800 points in the plot (but some points may be on top of each other due to the discrete nature of spike counts). For each reaching angle, find the mean firing rate across the 100 trials, and plot the mean firing rate using a red point on the same plot. Now, plot the tuning curve of this neuron in green on the same plot.

```
In [5]: ## 2c
        #======================================================#
        # YOUR CODE HERE:
        #   Plot the single trial spike counts and the tuning curve
        #   on top of each other.
        #======================================================#
        spike_counts = []
        for con in range(num_cons):
            for rep in range(num_trials):
                spike_counts.append([con, len(spike_times[con, rep])])
```

```
for i in range(len(spike_counts)):
    plt.scatter(spike_counts[i][0]*np.pi/4, spike_counts[i][1], color='blue')
by_100 = np.array_split(spike_counts, 8)
means = []
spike_counts = []
for i in range (len(by_100)):
    values = []
    for j in range(100):
        values.append(np.mean(by_100[i][j][1]))
        spike_counts.append((by_100[i][j][1]))
    means.append(np.mean(values))
    plt.scatter(i*np.pi/4, means[i], color='red')
time = np.linspace(0,6,num=30)
plt.plot(time, r_0 + (r_max-r_0)*np.cos(time-s_max), color='green')
pass

#=====================================================#
# END YOUR CODE
#=====================================================#
plt.xlabel('Reach angle (radians)')
plt.ylabel('Firing rate (spikes / second)')
plt.title('Simulated spike counts (blue)\n'+
          'mean simulated spike counts (red),and\n'+
          'cosine tuning curve used in simulation (green)')
plt.xlim(0, 2*np.pi)
```

Out[5]: (0, 6.283185307179586)



Question: Do the mean firing rates lie near the tuning curve?

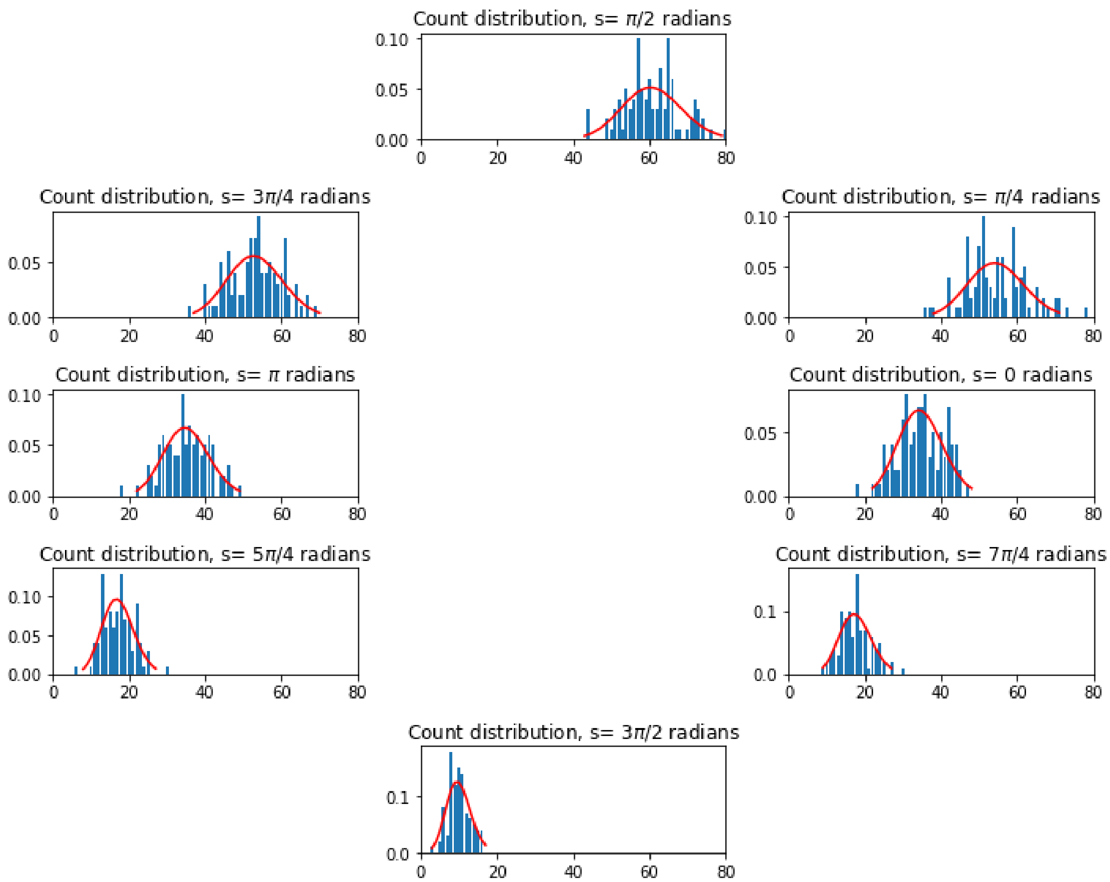
Your answer: Yes, the mean firing rates lie near the tuning curve

0.2.5 (d) (6 points) Count distribution

For each reaching angle, plot the *normalized* distribution (i.e., normalized so that the area under the distribution equals one) of spike counts (using the same counts from part (c)). Plot the 8 distributions around a circle, as in part (a). Fit a Poisson distribution to each empirical distribution and plot it on top of the corresponding empirical distribution.

```
In [6]: ##2d
        from scipy.stats import poisson
        plt.figure(figsize=(10,8))
        max_count = np.max(spike_counts)
        spike_count_bin_centers =  np.arange(0,max_count,1)
        for con in range(num_cons):
            plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
            #=====================================================#
            # YOUR CODE HERE:
            #   Calculate the empirical mean for the Poisson spike
```

```
#   counts, and then generate a curve reflecting the probability
#   mass function of the Poisson distribution as a function
#   of spike counts.
#=====================================================#
unique, counts = np.unique(spike_counts[con*100:(con+1)*100-1], return_counts = True
plt.bar(unique, counts/100.)
pass

#=====================================================#
# END YOUR CODE
#=====================================================#

#=====================================================#
# YOUR CODE HERE:
#   Plot the empirical count distribution, and on top of it
#   plot your fit Poisson distribution.
#=====================================================#
mu = np.mean(spike_counts[con*100:(con+1)*100-1])
x = np.arange(poisson.ppf(0.01, mu),poisson.ppf(0.99, mu))
plt.plot(x, poisson.pmf(x, mu), 'red')
pass


#=====================================================#
# END YOUR CODE
#=====================================================#
plt.xlim([0, max_count])
plt.title('Count distribution, s= '+ s_labels[con]+' radians')
plt.tight_layout()
```



Question: Are the empirical distributions well-fit by Poisson distributions?

Your answer: Yes, as can be seen in the graphs above, the Poisson distribution fits the empirical distribution well

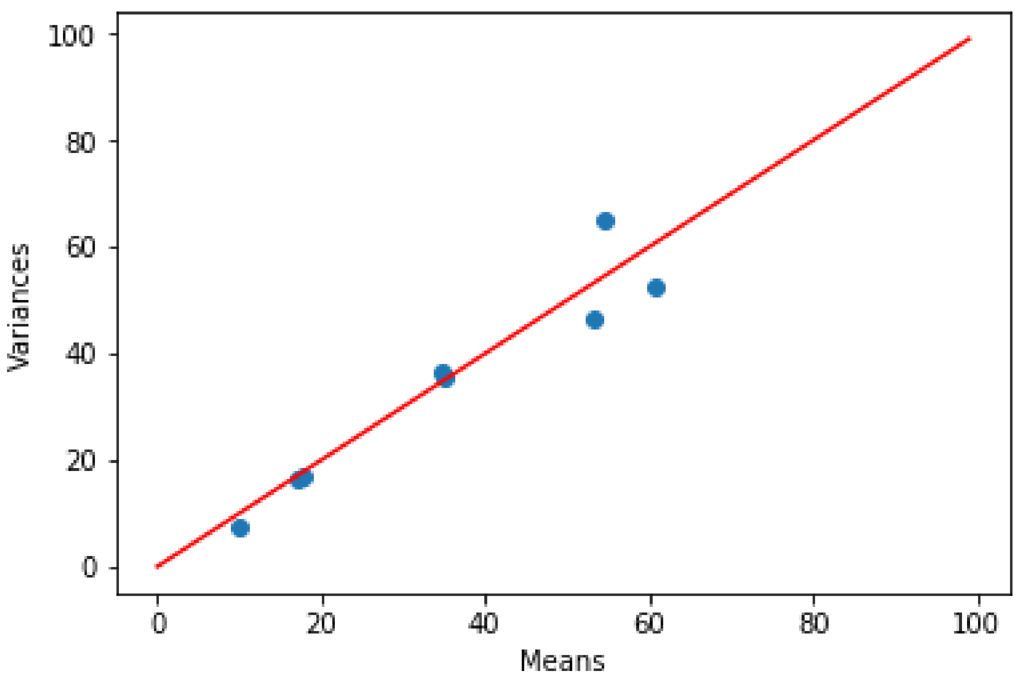
0.2.6 (e)(4 points) Fano factor

For each reaching angle, find the mean and variance of the spike counts across the 100 trials (using the same spike counts from part (c)). Plot the obtained mean and variance on the axes shown in Figure 1.14(A) in *TN*. There should be 8 points in this plot -- one per reaching angle.

```
In [7]: ## 2e
        #======================================================#
        # YOUR CODE HERE:
        #   Calculate and plot the mean and variance for each of
        #   the 8 reaching conditions.  Mean should be on the
        #   x-axis and variance on the y-axis.
        #======================================================#
        means = []
```

```
variances = []
for con in range(num_cons):
    means.append(np.mean(spike_counts[con*100:(con+1)*100-1]))
    variances.append(np.var(spike_counts[con*100:(con+1)*100-1]))

plt.scatter(means, variances)
plt.xlabel('Means')
plt.ylabel('Variances')
pass

plt.plot(range(0,100), range(0,100), color='red')
#==========================================================#
# END YOUR CODE
#==========================================================#
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x151a4fc0f0>]
```



Question: Do these points lie near the 45 deg diagonal, as would be expected of a Poisson distribution?

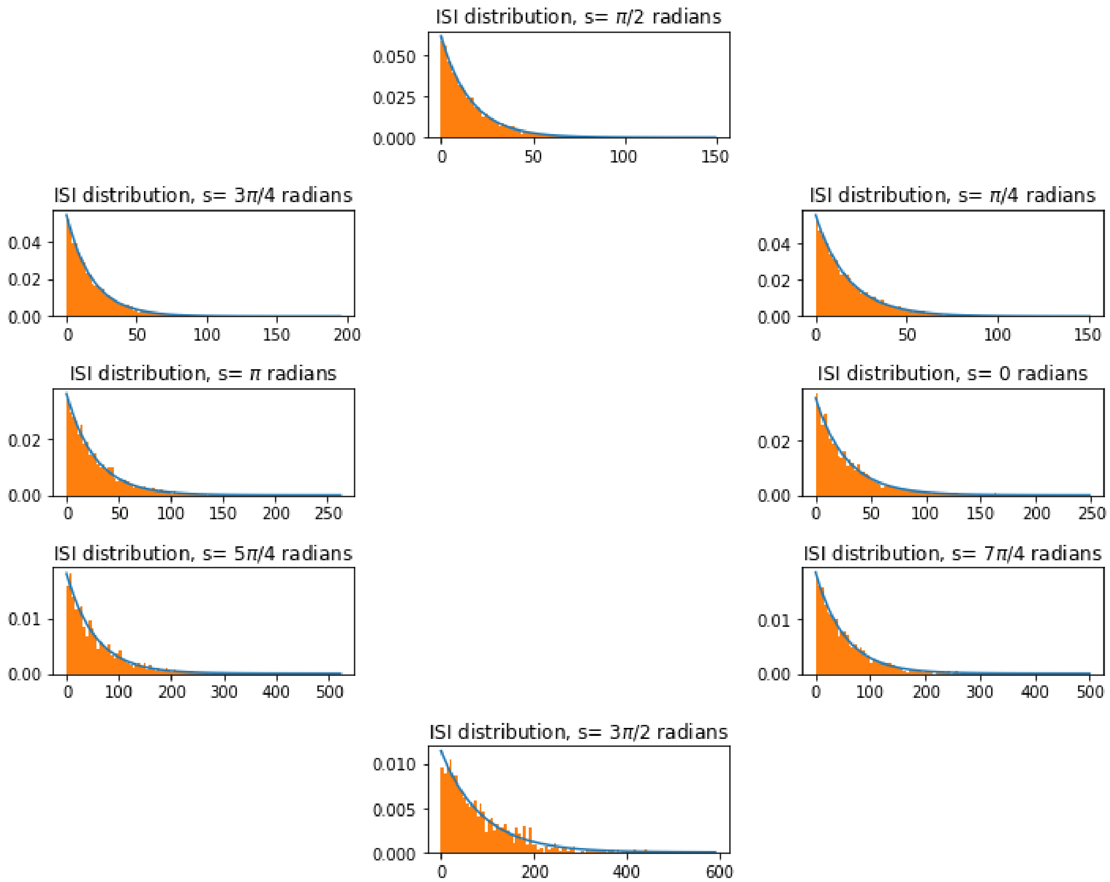
Your answer: Yes, the points lie near the 45 degree diagonal as would be expected

0.2.7 (f) (5 points) Interspike interval (ISI) distribution

For each reaching angle, plot the normalized distribution of ISIs. Plot the 8 distributions around a circle, as in part (a). Fit an exponential distribution to each empirical distribution and plot it on top of the corresponding empirical distribution.

In [8]:
```
## 2f
plt.figure(figsize=(10,8))

for con in range(num_cons) :
    plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])


    #======================================================#
    # YOUR CODE HERE:
    #   Calculate the interspike interval (ISI) distribution
    #   by finding the empirical mean of the ISI's, which
    #   is the inverse of the rate of the distribution.
    #======================================================#
    times = []
    for ren in range(num_trials):
        times.append(spike_times[con][ren])
    relevant_ISI = []
    for k in range(num_trials):
        relevant_ISI.append([x - times[k][j - 1] for j, x in enumerate(times[k])][1:])
    #print(relevant_ISI)
    flat_list = [item for sublist in relevant_ISI for item in sublist]
    flat_list_mean = np.mean(flat_list)
    flat_list_lambda = 1/flat_list_mean
    sampled_flat_list = np.linspace(0, max(flat_list), 100)
    curve = flat_list_lambda*np.exp(-flat_list_lambda*sampled_flat_list)
    plt.plot(sampled_flat_list, curve)
    pass

    #======================================================#
    # END YOUR CODE
    #======================================================#

    #======================================================#
    # YOUR CODE HERE:
    #   Plot Interspike interval (ISI) distribution
    #======================================================#
    bins = 100
    plt.hist(flat_list, bins, density=True)
    pass

    #======================================================#
    # END YOUR CODE
    #======================================================#
```

```
plt.title('ISI distribution, s= '+ s_labels[con]+' radians')
plt.tight_layout()
```



Question: Are the empirical distributions well-fit by exponential distributions?

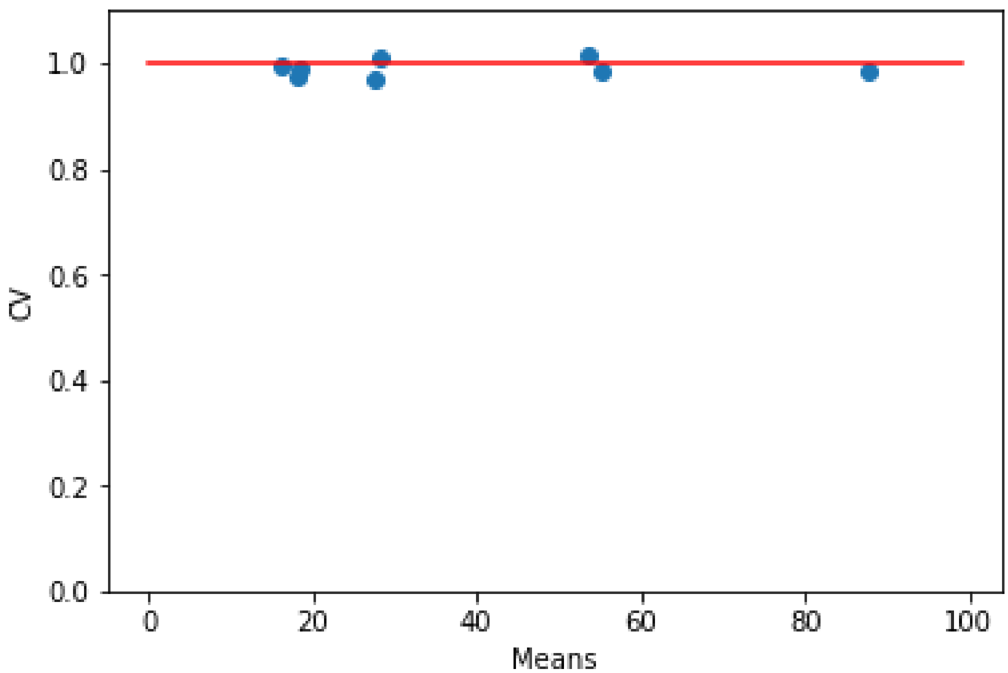
Your answer: Yes they do, as can be seen in the graphs above. Bins used = 100

0.2.8 (g) (5 points) Coefficient of variation (C_V)

For each reaching angle, find the average ISI and C_V of the ISIs. Plot the resulting values on the axes shown in Figure 1.16 in *TN*. There should be 8 points in this plot.

```
In [9]: #2g
        #==================================================#
        # YOUR CODE HERE:
        # Calculate and plot coeffcient of variation
        #==================================================#
        means = []
        cv = []
```

```
for con in range(num_cons):
    relevant_values = spike_times[con,:]
    relevant_ISI = []
    for i in range(100):
        relevant_ISI.append([x - relevant_values[i][j - 1] for j, x in enumerate(relevan
    flat_list = [item for sublist in relevant_ISI for item in sublist]
    means.append(np.mean(flat_list))
    cv.append(np.std(flat_list)/np.mean(flat_list))
plt.scatter(means, cv)
plt.ylabel('CV')
plt.xlabel('Means')
plt.ylim(0,1.1)
plt.plot(range(0,100), np.ones(100), color='red')
pass

#======================================================#
# END YOUR CODE
#======================================================#
```



Question: Do the C_V values lie near unity, as would be expected of a Poisson process?

Your answer: yes, as can be seen from the graph above, they all lie near 1

hw3p3

May 4, 2018

0.1 Homework 3, Problem 3 on inhomogeneous Poisson processes

ECE C143A/C243A, Spring Quarter 2018, Prof. J.C. Kao, TAs T. Monsoor, X. Jiang and X. Yang.

In this problem, we will use the same simulated neuron as in Problem 2, but now the reaching angle s will be time-dependent with the following form:

$$s(t) = t^2 \cdot \pi,$$

where t ranges between 0 and 1 second. This will be refered as *s(t)* equation in the questions.

In [1]:
```
"""
ECE C143/C243 Homework-3 Problem-3

"""
import numpy as np
import matplotlib.pyplot as plt
import nsp as nsp # these are helper functions that we provide.
import scipy.special

# Load matplotlib images inline
%matplotlib inline

# Reloading any code written in external .py files.
%load_ext autoreload
%autoreload 2
```

0.1.1 (a) (6 points) Spike trains

Generate 100 spike trains, each 1 second in duration, according to an inhomogeneous Poisson process with a firing rate profile defined by tuning equation,

$$\lambda(s) = r_0 + (r_{\max} - r_0)\cos(s - s_{\max})$$

and the $s(t)$ equation,

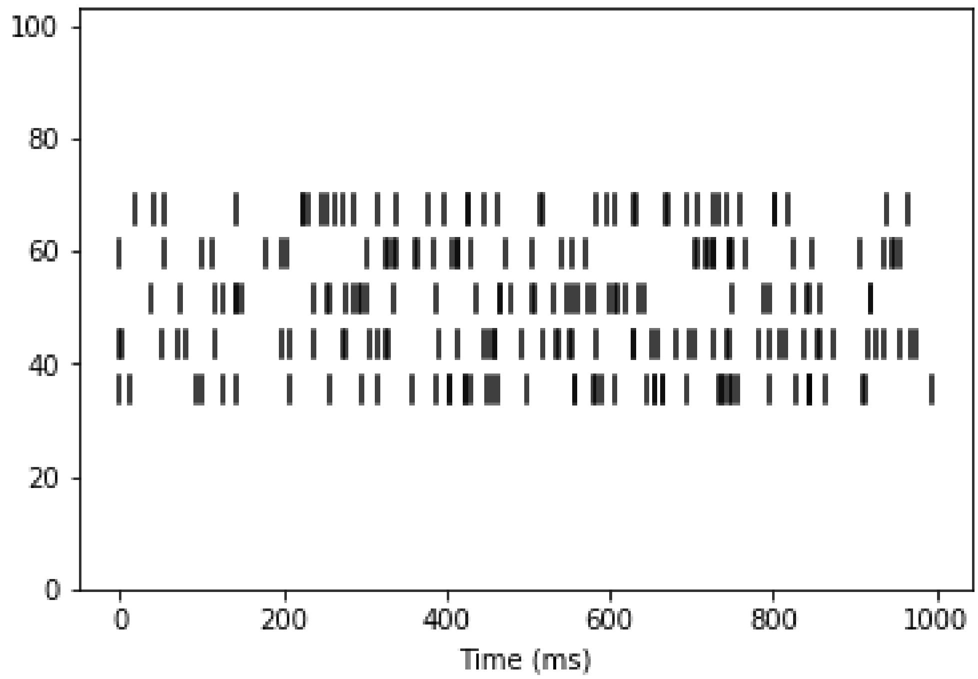
$$s(t) = t^2 \cdot \pi$$

In [2]:
```
r_0 = 35 # (spikes/s)
r_max = 60 # (spikes/s)
s_max = np.pi/2 # (radians)
T = 1000 # trial length (ms)
```

In [3]:
```
np.random.exponential(1.0/r_max * 1000)
```

Out[3]:
```
4.740174461262221
```

In [4]:
```
## 3a
num_trials = 100 # number of total spike trains
num_rasters_to_plot = 5 # number of spike trains to plot
#=====================================================#
# YOUR CODE HERE:
#   Generate the spike times for 100 trials of an inhomogeneous
#   Poisson process.  Plot 5 example spike rasters.
#=====================================================#
def GeneratePoissonSpikeTrain( T, rate ):
#GENERATEPOISSONSPIKETRAIN Summary of this function goes here
#   T in ms
#   r in spikes/s
#   returns spike_train, a collection of spike times

    spike_train = np.array(0)
    time = 0
    spike_train_final = []
    while time <= T:
        time_next_spike = np.random.exponential(1/rate * 1000)
        time = time + time_next_spike
        spike_train = np.append(spike_train,time)
    #discard last spike if happens after T
    if (spike_train[np.size(spike_train)-1] > T) :
        spike_train = spike_train[:-1]
    for i in range(len(spike_train)):
        random_number = np.random.uniform()
        lambda_value = r_0 + (r_max - r_0)*np.cos(((spike_train[i]/1000)**2) * np.pi - s
        if random_number < lambda_value / r_max:
            spike_train_final.append(spike_train[i])


    return np.asarray(spike_train_final)


one_hundred_spike_trains = []
for i in range(100):
    one_hundred_spike_trains.append(GeneratePoissonSpikeTrain(1000,r_max))

nsp.PlotSpikeRaster(one_hundred_spike_trains[0:num_rasters_to_plot])
#=====================================================#
# END YOUR CODE
#=====================================================#
```



0.1.2 (b) (5 points) Spike histogram

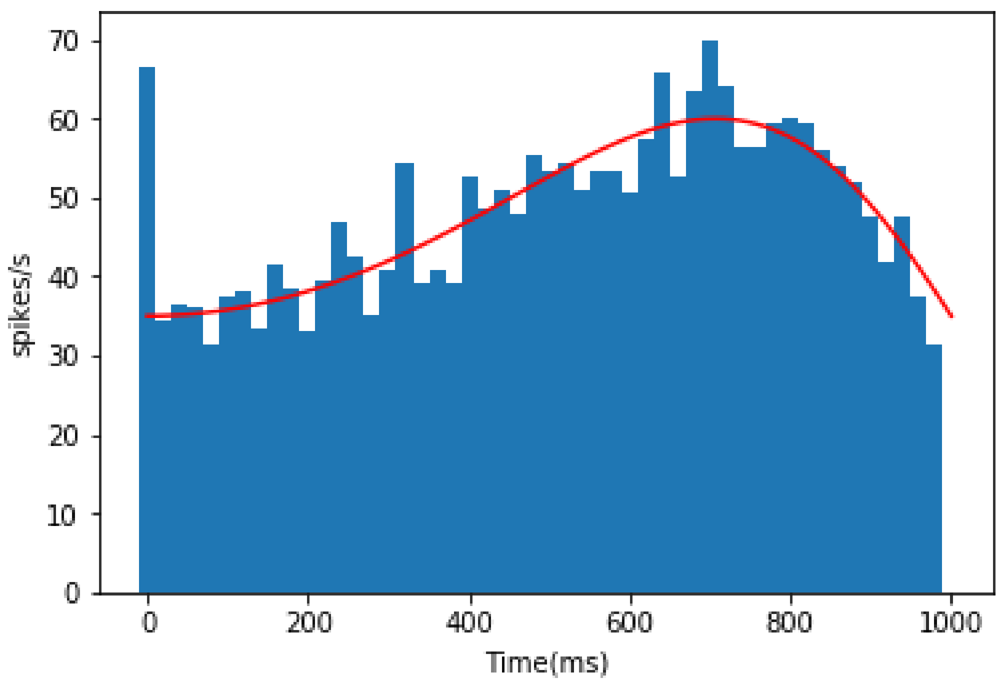
Plot the spike histogram by taking spike counts in non-overlapping 20 ms bins, then averaging across the 100 trials. The spike histogram should have firing rate (in spikes / second) as the vertical axis and time (in msec, not time bin index) as the horizontal axis. Plot the expected firing rate profile defined by equations tuning equation and *s(t)* equation on the same plot.

Question: Does the spike histogram agree with the expected firing rate profile?

```
In [5]: # 3b
        bin_width = 20 # (ms)
        #=====================================================#
        # YOUR CODE HERE:
        #   Plot the spike histogram
        #=====================================================#
        flat_list = [item for sublist in one_hundred_spike_trains for item in sublist]
        full_digitized = []
        bins = range(0,1001,20)
        data = flat_list
        full_digitized.extend(np.ndarray.tolist(np.digitize(data, bins)))
        counts = [full_digitized.count(x) for x in set(full_digitized)]
        counts = [x / 2. for x in counts]
        plt.bar(bins[0:50],counts, width=20)
```

```
time = np.linspace(0,1000,num=100)
plt.plot(time, r_0 + (r_max - r_0)*np.cos(((time/1000)**2) * np.pi - s_max), color='red'

#=====================================================#
# END YOUR CODE
#=====================================================#

plt.ylabel('spikes/s')
plt.xlabel('Time(ms)')
```

Out[5]: Text(0.5,0,'Time(ms)')



Question: Does the spike histogram agree with the expected firing rate profile?

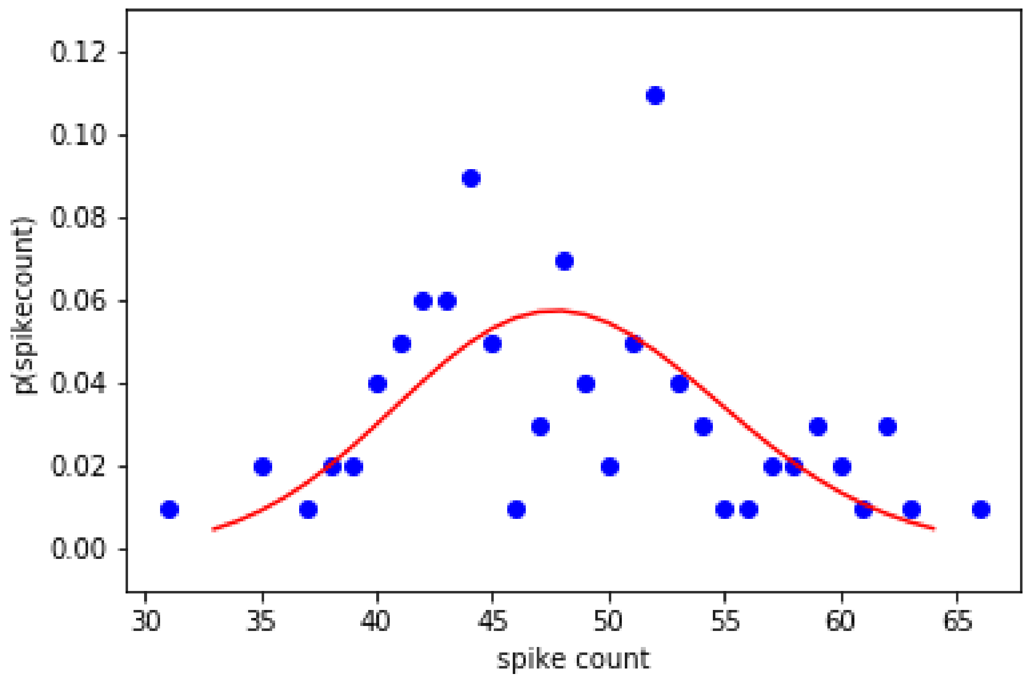
Your Answer: Yes, as can be seen above, the spike histogram agrees with the expected firing profile.

0.1.3 (c) (6 points) Count distribution

For each trial, count the number of spikes across the entire trial. Plot the normalized distribution of spike counts. Fit a Poisson distribution to this empirical distribution and plot it on top of the empirical distribution.

In [7]:
```
#=====================================================#
# YOUR CODE HERE:
```

```python
#   Plot the normalized distribution of spike counts
#=====================================================#
from scipy.stats import poisson
one_hundred_counts = []
for i in range(100):
    one_hundred_counts.append(len(one_hundred_spike_trains[i]))
unique, counts = np.unique(one_hundred_counts, return_counts = True)
plt.scatter(unique, counts/100, color='blue')
mu = np.mean(one_hundred_counts)
x = np.arange(poisson.ppf(0.01, mu), poisson.ppf(0.99, mu))
plt.plot(x, poisson.pmf(x, mu), color='red')
#=====================================================#
# END YOUR CODE
#=====================================================#
plt.xlabel('spike count')
plt.ylabel('p(spikecount)')
plt.show()
```



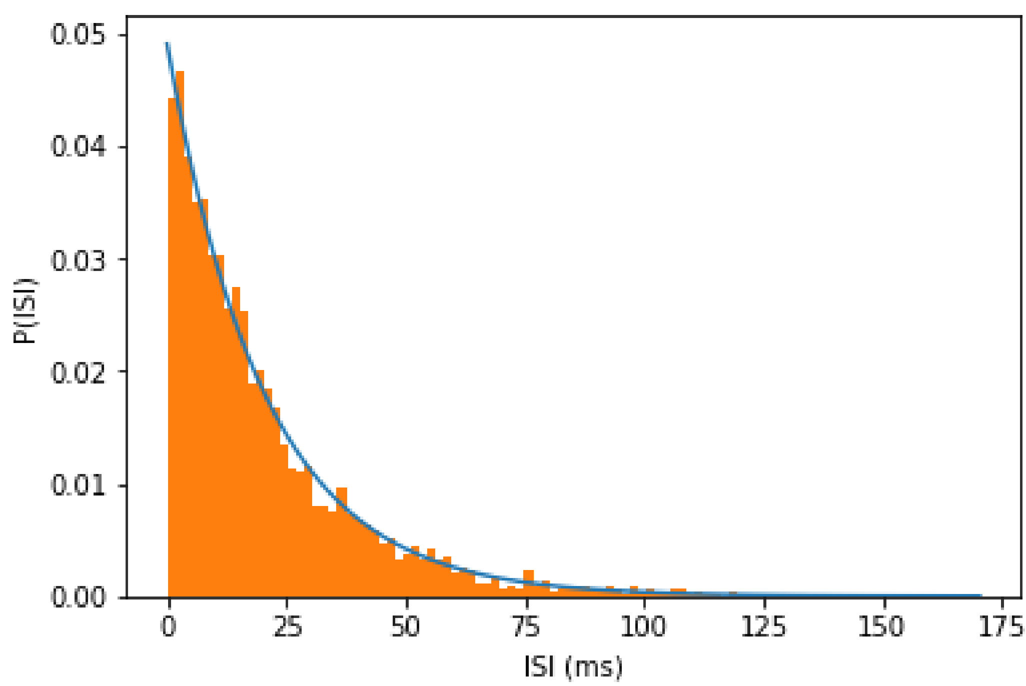
Question: Should we expect the spike counts to be Poisson-distributed?

Your Answer: In general, we should still have a poisson distribution which is clearly shown in the graph above. Since we are taking all points from a poisson distribution, even in the inhomogenous case, we should still have a poisson distribution.

0.1.4 (d) (5 points) ISI distribution

Plot the normalized distribution of ISIs. Fit an exponential distribution to the empirical distribution and plot it on top of the empirical distribution.

In [8]:
```
#=====================================================#
# YOUR CODE HERE:
#   Plot the normalized distribution of ISIs
#=====================================================#
relevant_values = one_hundred_spike_trains
relevant_ISI = []
for i in range(100):
    relevant_ISI.append([x - relevant_values[i][j - 1] for j, x in enumerate(relevant_va
flat_list = [item for sublist in relevant_ISI for item in sublist]
flat_list_mean = np.mean(flat_list)
flat_list_lambda = 1/flat_list_mean
sampled_flat_list = np.linspace(0, max(flat_list), 100)
curve = flat_list_lambda*np.exp(-flat_list_lambda*sampled_flat_list)
plt.plot(sampled_flat_list, curve)
bins = 100
plt.hist(flat_list, bins, density=True)

#=====================================================#
# END YOUR CODE
#=====================================================#
plt.xlabel('ISI (ms)')
plt.ylabel('P(ISI)')
plt.show()
```



Question: Should we expect the ISIs to be exponentially-distributed? (Note, it is possible for the empirical distribution to strongly resemble an exponential distribution even if the data aren't exponentially distributed.)

Your Answer: In the inhomogenous case, we should not expect the ISIs to be exponentially distributed.

hw3p4

May 4, 2018

0.1 Homework 3, Problem 4 on real neural data.

ECE C143A/C243A, Spring Quarter 2018, Prof. J.C. Kao, TAs T. Monsoor, X. Jiang and X. Yang.

We will analyze real neural data recorded using a 100-electrode array in premotor cortex of a macaque monkey(The neural data have been generously provided by the laboratory of Prof. Krishna Shenoy at Stanford University. The data are to be used exclusively for educational purposes in this course.). The dataset can be found on CCLE as `ps3_data.mat`.

The following describes the data format. The .mat file has a single variable named *trial*, which is a structure of dimensions (182 trials) $\times$ (8 reaching angles). The structure contains spike trains recorded from a single neuron while the monkey reached 182 times along each of 8 different reaching angles (where the trials of different reaching angles were interleaved). The spike train for the nth trial of the k th reaching angle is contained in *trial(n,k).spikes*, where $n = 1, \ldots, 182$ and * k $= 1, \ldots, 8$. *The indices* k * $= 1, \ldots, 8$ correspond to reaching angles $\frac{30}{180}\pi$, $\frac{70}{180}\pi$, $\frac{110}{180}\pi$, $\frac{150}{180}\pi$, $\frac{190}{180}\pi$, $\frac{230}{180}\pi$, $\frac{310}{180}\pi$, $\frac{350}{180}\pi$, respectively. The reaching angles are not evenly spaced around the circle due to experimental constraints that are beyond the scope of this homework.

A spike train is represented as a sequence of zeros and ones, where time is discretized in 1 ms steps. A zero indicates that the neuron did not spike in the 1 ms bin, whereas a one indicates that the neuron spiked once in the 1 ms bin. Due to the refractory period, it is not possible for a neuron to spike more than once within a 1 ms bin. Each spike train is 500 ms long and is, thus, represented by a 1×500 vector.

We load this data for you using the `sio` library. Be sure that `ps3_data.mat` is in the same directory as this notebook / on the system path. If you prefer to have it on a different path, specify it in the `sio.loadmat` command.

In [1]:
```
"""
ECE C143/C243 Homework-3 Problem-4

"""

# Importing the necessary packages

import numpy as np
import matplotlib.pyplot as plt
import nsp as nsp
import scipy.special
import scipy.io as sio
```

```
# Importing the Matlab data
data = sio.loadmat('ps3_data.mat') # load the .mat file.
num_trials = data['trial'].shape[0]
num_cons = data['trial'].shape[1]

# Load matplotlib images inline
%matplotlib inline

# Reloading any code written in external .py files.
%load_ext autoreload
%autoreload 2
```

```
In [2]: interested_data = data['trial'][181,1][1]
        print(interested_data[0])
        print(np.argwhere(interested_data[0] == 1))
```

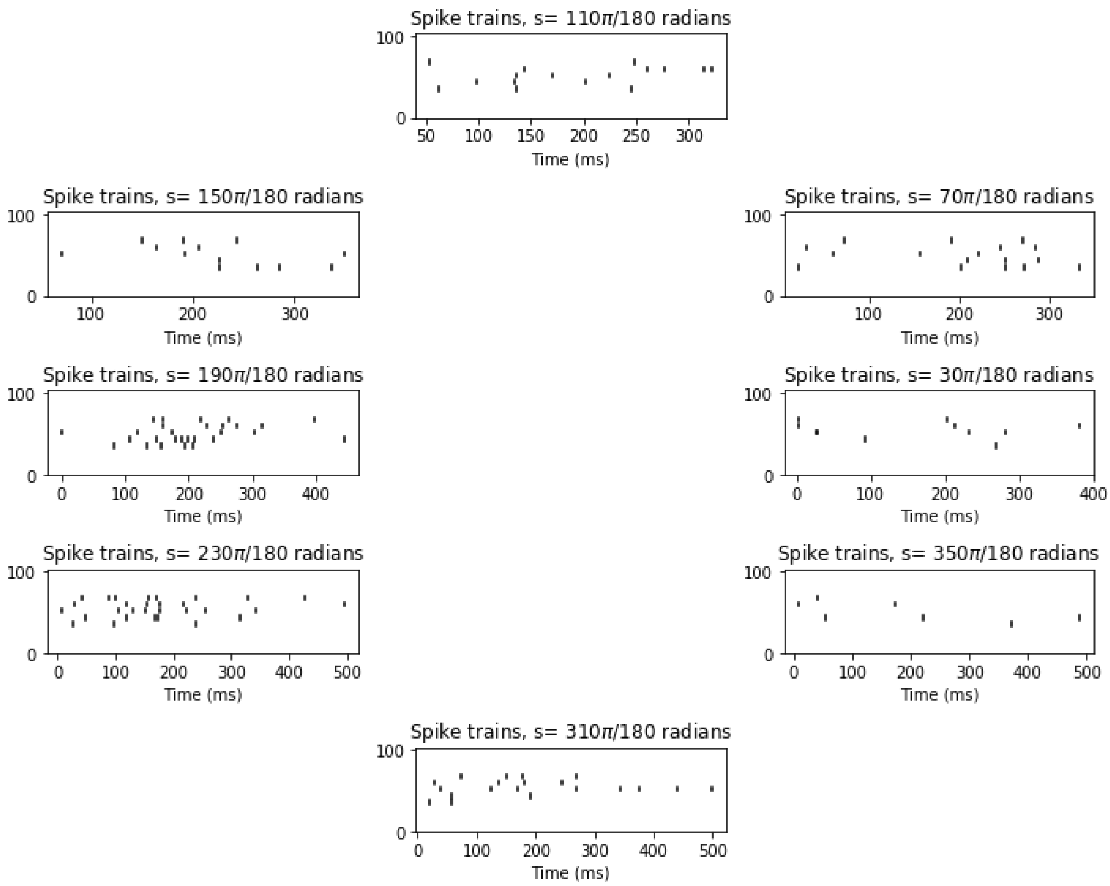
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[[108]
 [172]
 [235]]
```

0.1.1 (a) (6 points) Spike trains

Generate the `spike_times` matrix for the real data. This should have the same `spike_times` format described in problem 2. The following code, when complete, will plot 5 spike trains for each reaching angle in the same format as shown in Figure 1.6(A) in *TN*. To simplify the plotting

```
In [3]: ## 4a

        T = 500; #trial length (ms)

        num_rasters_to_plot = 5; # per reaching angle

        s = np.pi*np.array([30.0/180,70.0/180,110.0/180 ,150.0/180 ,190.0/180 ,230.0/180 ,310.0/
        s_labels = ['30$\pi$/180', '70$\pi$/180', '110$\pi$/180', '150$\pi$/180', '190$\pi$/180'
```

```
            '230$\pi$/180', '310$\pi$/180', '350$\pi$/180']

# These variables help to arrange plots around a circle
num_plot_rows = 5
num_plot_cols = 3
subplot_indx = [9, 6, 2, 4, 7, 10, 14, 12]

# Initialize the spike_times array
spike_times = np.empty((num_cons, num_trials), dtype=list)

plt.figure(figsize=(10,8))
spike_counts = np.empty([num_cons, num_trials], dtype=list)
for con in range(num_cons):
    for rep in range(num_trials):
        #=====================================================#
        # YOUR CODE HERE:
        #   Calculate the spike trains for each reaching angle.
        #   You should calculate the spike_times array that you
        #   computed in problem 2.  This way, the following code
        #   will plot the histograms for you.
        #=====================================================#
        interested_data = data['trial'][rep,con][1][0]
        spike_times[con, rep] = np.argwhere(interested_data == 1)
        spike_counts[con,rep] = np.size(spike_times[con,rep])
        pass

        #=====================================================#
        # END YOUR CODE
        #=====================================================#

    plt.subplot(num_plot_rows, num_plot_cols, subplot_indx[con])
    nsp.PlotSpikeRaster(spike_times[con, 0:num_rasters_to_plot])
    plt.title('Spike trains, s= '+s_labels[con]+' radians')
    plt.tight_layout()
```




```
In [4]: print(spike_counts[4])
```

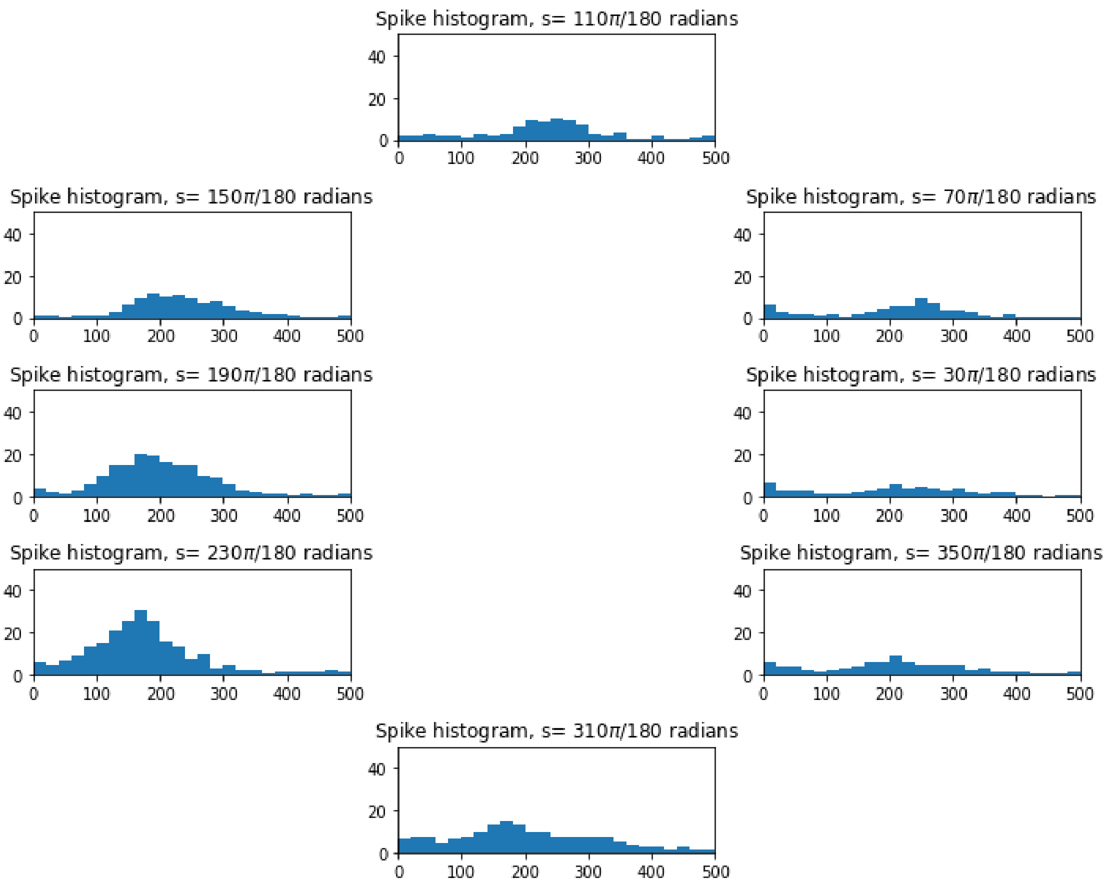
```
[5 8 5 5 5 7 6 6 7 7 3 6 9 9 5 5 8 7 4 5 4 4 8 6 7 6 8 6 8 8 3 5 2 4 6 7 5
 5 5 7 7 5 3 7 7 5 9 8 8 6 3 3 5 6 3 5 8 3 3 3 4 5 5 2 4 6 8 4 4 4 3 5 3 4
 8 4 4 2 4 6 6 4 7 7 2 6 5 1 4 3 4 5 4 4 7 4 6 5 5 1 3 4 2 4 7 4 3 4 3 7 2
 6 2 5 6 6 4 3 4 4 5 5 5 7 3 4 7 6 6 2 5 4 4 3 5 3 4 5 7 5 5 3 4 2 5 7 0 4
 4 5 5 4 3 3 4 4 4 2 4 3 3 4 5 3 6 5 3 6 6 6 3 3 5 4 3 4 4 6 2 5 8 4]
```

0.1.2 (b) (5 points) Spike histogram

For each reaching angle, find the spike histogram by taking spike counts in non-overlapping 20~ms bins, then averaging across the 182 trials. The spike histograms should have firing rate (in spikes / second) as the vertical axis and time (in msec, not time bin index) as the horizontal axis. Plot the histogram for 500ms worth of data. Plot the 8 resulting spike histograms around a circle, as in part (a).

```
In [5]: ## 4b
        bin_width = 20 # (ms)
        bin_centers = np.arange(bin_width/2,T,bin_width) # (ms)
```

```python
plt.figure(figsize=(10,8))
max_t = 500 # (ms)
max_rate = 50 # (in spikes/s)

for con in range(num_cons):
    plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
    #=====================================================#
    # YOUR CODE HERE:
    #   Plot the spike histogram
    #=====================================================#
    flat_list = [item for sublist in spike_times[con] for item in sublist]
    full_digitized = []
    bins = range(0,501,20)
    data = flat_list
    full_digitized.extend(np.ndarray.tolist(np.digitize(data, bins)))
    flat_full_digitized = [item for sublist in full_digitized for item in sublist]
    counts = [flat_full_digitized.count(x) for x in range(1,26)]
    counts = [x*1/5 for x in counts]
    plt.bar(bin_centers, counts, width=20)

    pass

    #=====================================================#
    # END YOUR CODE
    #=====================================================#
    plt.axis([0, max_t, 0, max_rate])
    plt.title('Spike histogram, s= '+s_labels[con]+' radians')
    plt.tight_layout()
```



0.1.3 (c) (4 points) Tuning curve

For each trial, count the number of spikes across the entire trial. Plots these points on the axes shown in Figure 1.6(B) in *TN*. There should be $182 \cdot 8$ points in the plot (but some points may be on top of each other due to the discrete nature of spike counts). For each reaching angle, find the mean firing rate across the 182 trials, and plot the mean firing rate using a red point on the same plot. Then, fit the cosine tuning curve eqntuning to the 8 red points by minimizing the sum of squared errors

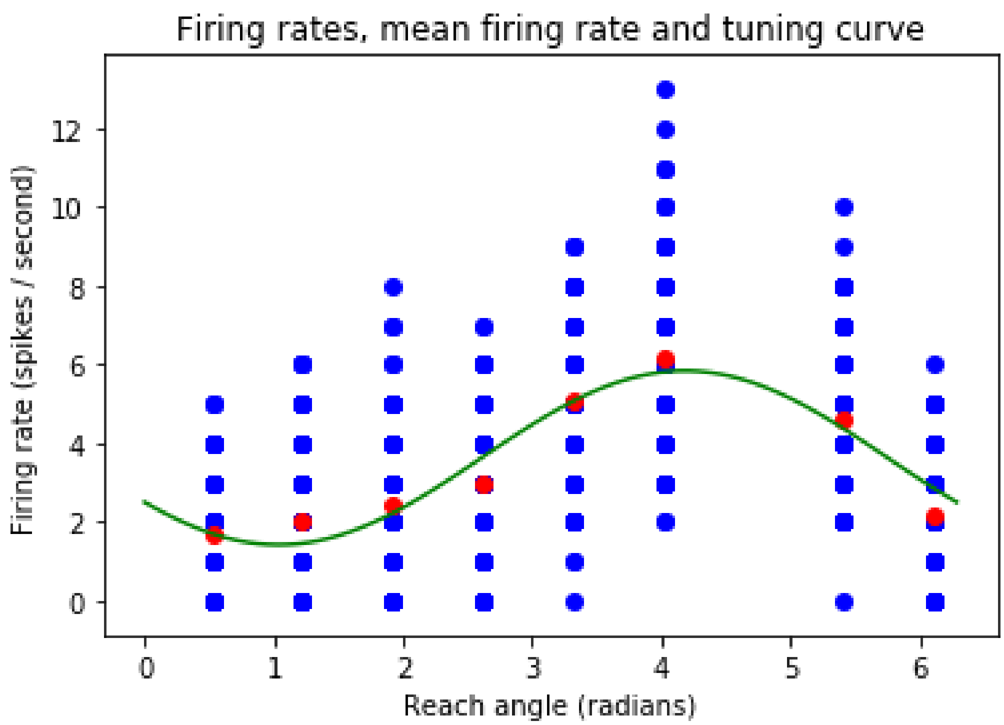
$$\sum_{i=1}^{8} \left(\lambda(s_i) - r_0 - (r_{\max} - r_0) \cos(s_i - s_{\max}) \right)^2$$

with respect to the parameters r_0, $r_{\max}$, and $s_{\max}$. (Hint: this can be done using linear regression; refer to Homework # 2.) Plot the resulting tuning curve of this neuron in green on the same plot.

```
In [6]: #=====================================================#
        # YOUR CODE HERE:
        # Tuning curve. Please use the following colors for plot:
```

```
# Firing rates(blue);Mean firing rate(red); Cosine tuning curve(green)
#======================================================#
spike_counts = []
for con in range(num_cons):
    for rep in range(num_trials):
        spike_counts.append([con, len(spike_times[con, rep])])
for i in range(len(spike_counts)):
    #plt.scatter(spike_counts[i][0]*np.pi/4, spike_counts[i][1], color='blue')
    plt.scatter(s[spike_counts[i][0]], spike_counts[i][1], color='blue')
by_100 = np.array_split(spike_counts, 8)
means = []
spike_counts = []
for i in range (len(by_100)):
    values = []
    for j in range(128):
        values.append(np.mean(by_100[i][j][1]))
        spike_counts.append((by_100[i][j][1]))
    means.append(np.mean(values))
    plt.scatter(s[i], means[i], color='red')


a = np.array([[1, np.sin((30*np.pi)/180), np.cos((30*np.pi)/180)],
              [1, np.sin((70*np.pi)/180), np.cos((70*np.pi)/180)],
              [1, np.sin((110*np.pi)/180),np.cos((110*np.pi)/180)],
              [1, np.sin((150*np.pi)/180), np.cos((150*np.pi)/180)],
              [1, np.sin((190*np.pi)/180), np.cos((190*np.pi)/180)],
              [1, np.sin((230*np.pi)/180), np.cos((230*np.pi)/180)],
              [1, np.sin((310*np.pi)/180), np.cos((310*np.pi)/180)],
              [1, np.sin((350*np.pi)/180), np.cos((350*np.pi)/180)]
             ])
b = np.array([1.703125, 2.015625, 2.421875, 2.9453125, 5.0546875, 6.1640625, 4.6328125,
k0,k1,k2 = np.linalg.lstsq(a,b)[0]
c0 = k0
theta0 = np.arctan(k1/k2)
c1 = k1/ np.sin(theta0)
theta0 = theta0*180/(np.pi)
theta = np.linspace(0, 2*np.pi, num=80)
plt.plot(theta *np.pi/ np.pi,c0 + c1 *np.cos(theta - theta0 * np.pi/180),'g',2)
pass

#======================================================#
# END YOUR CODE
#======================================================#
plt.xlabel('Reach angle (radians)')
plt.ylabel('Firing rate (spikes / second)')
plt.title('Firing rates, mean firing rate and tuning curve')
```

Out[6]: Text(0.5,1,'Firing rates, mean firing rate and tuning curve')

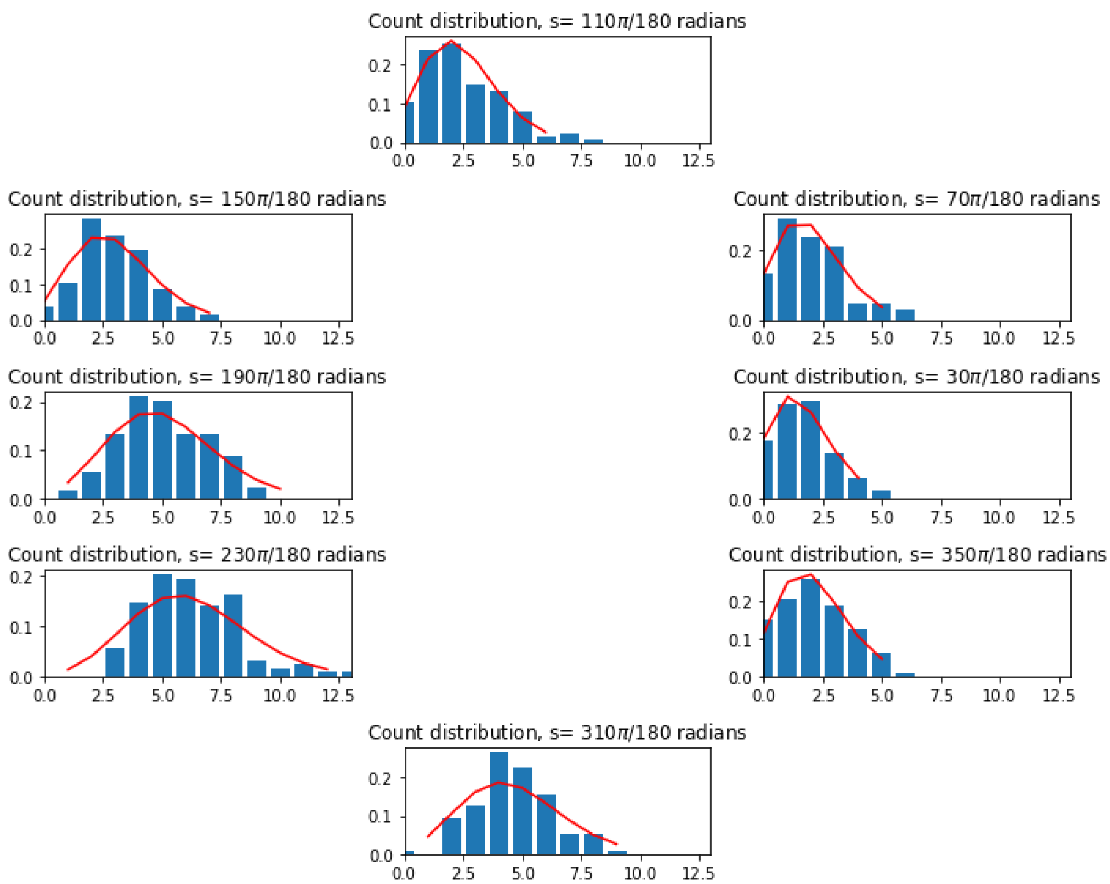


0.2 (d) (6 points) Count distribution

For each reaching angle, plot the normalized distribution of spike counts (using the same counts from part (c)). Plot the 8 distributions around a circle, as in part (a). Fit a Poisson distribution to each empirical distribution and plot it on top of the corresponding empirical distribution.

In [7]:
```
plt.figure(figsize=(10,8))
max_count = 13
spike_count_bin_centers =  np.arange(0,max_count,1)

for con in range(num_cons):
    plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])

    #=====================================================#
    # YOUR CODE HERE:
    #   Find the empirical mean of the poission distribution
    #   and calculate the Poisson distribution.
    #=====================================================#
    from scipy.stats import poisson
    unique, counts = np.unique(spike_counts[con*128:(con+1)*128-1], return_counts = True
    plt.bar(unique, counts/128.)
    pass

    #=====================================================#
```

```
# END YOUR CODE
#======================================================#


#======================================================#
# YOUR CODE HERE:
#   Plot the empirical distribution of spike counts and the
#   Poission distribution you just calculated
#======================================================#
mu = np.mean(spike_counts[con*128:(con+1)*128-1])
x = np.arange(poisson.ppf(0.01, mu),poisson.ppf(0.99, mu))
plt.plot(x, poisson.pmf(x, mu), 'red')
pass

#======================================================#
# END YOUR CODE
#======================================================#
plt.xlim([0, max_count])
plt.title('Count distribution, s= '+ s_labels[con]+' radians')
plt.tight_layout()
```



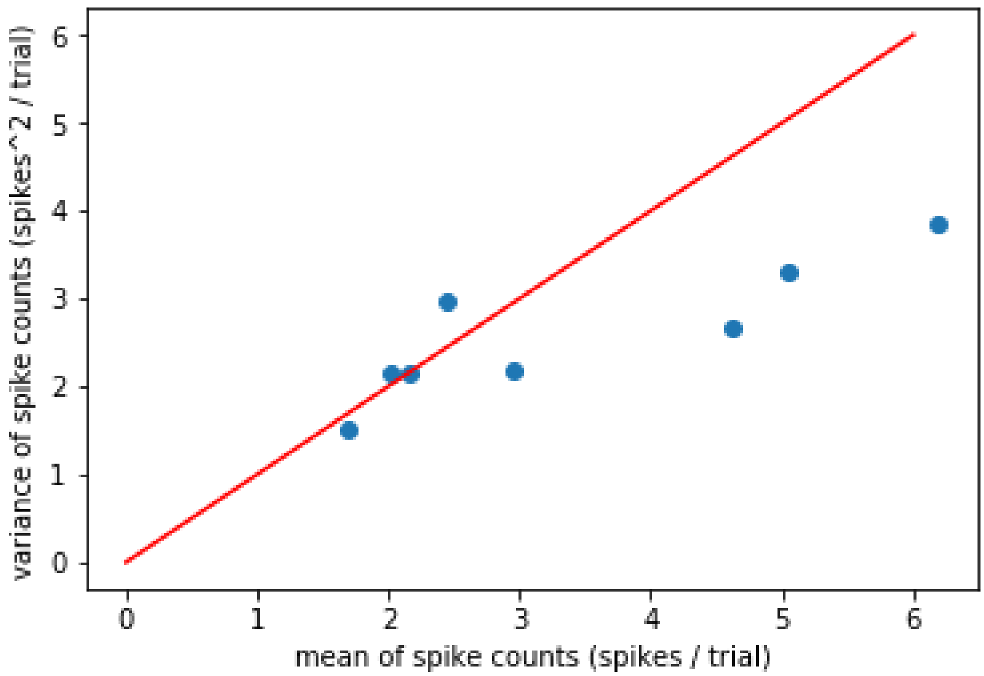
Question: Why might the empirical distributions differ from the idealized Poisson distributions?

Your answer: The empirical may differ from the idealized Poisson because we are using real data which is not simulated and there is always noise which is involved in collecting real data.

0.2.1 (e) (4 points) Fano factor

For each reaching angle, find the mean and variance of the spike counts across the 182 trials (using the same spike counts from part (c)). Plot the obtained mean and variance on the axes shown in Figure 1.14(A) in *TN*. There should be 8 points in this plot -- one per reaching angle.

In [12]:
```
## 4e

#======================================================#
# YOUR CODE HERE:
# Plot the mean and variance of spike counts on the axes
#======================================================#
means = []
variances = []
for con in range(num_cons):
    means.append(np.mean(spike_counts[con*128:(con+1)*128-1]))
    variances.append(np.var(spike_counts[con*128:(con+1)*128-1]))

plt.scatter(means, variances)
plt.xlabel('Means')
plt.ylabel('Variances')
pass
plt.plot(range(0,7), range(0,7), color='red')
pass

#======================================================#
# END YOUR CODE
#======================================================#
plt.xlabel('mean of spike counts (spikes / trial)')
plt.ylabel('variance of spike counts (spikes^2 / trial)')
plt.show()
```



Question: Do these points lie near the 45 deg diagonal, as would be expected of a Poisson distribution?

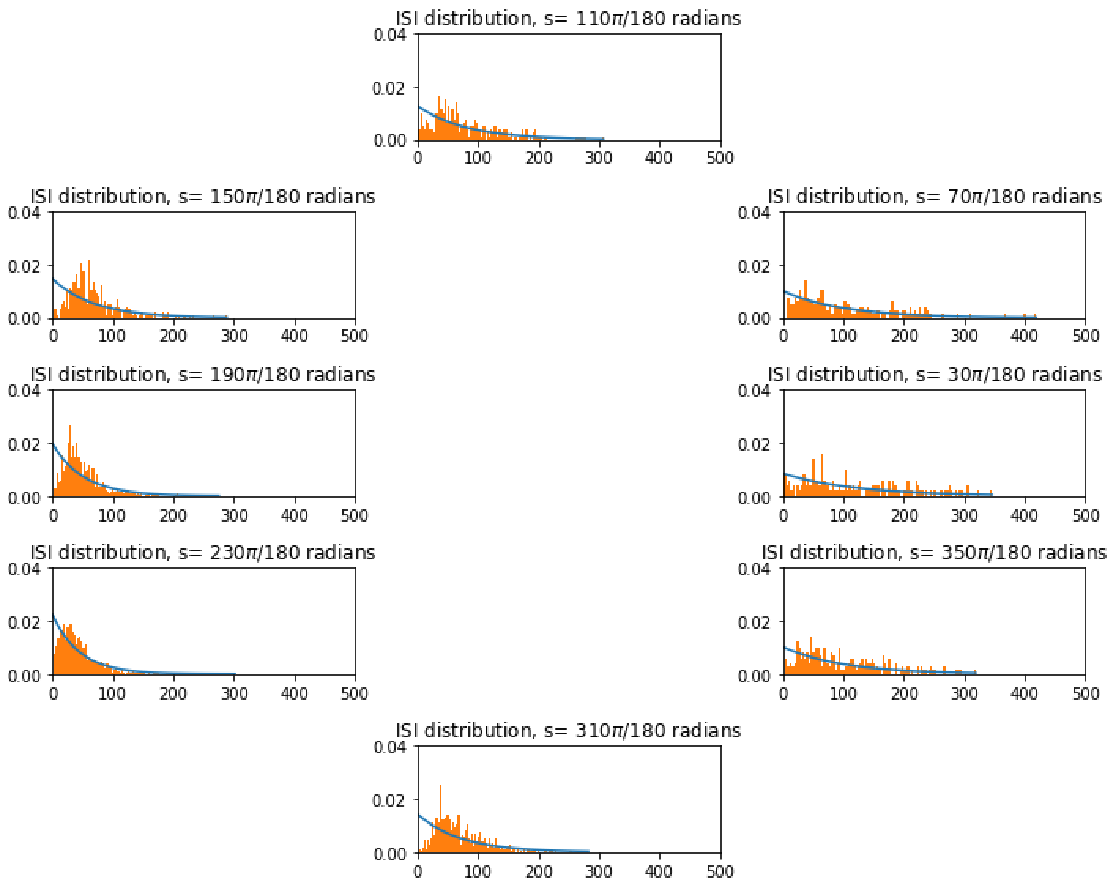
Your answer:Some of the points lie near the 45 deg degree but there are also others that are a bit off the diagonal. A best fit line would definitely be something close to a 45 degree line through the points however.

0.2.2 (f) (5 points) Interspike interval (ISI) distribution

For each reaching angle, plot the normalized distribution of ISIs. Plot the 8 distributions around a circle, as in part (a). Fit an exponential distribution to each empirical distribution and plot it on top of the corresponding empirical distribution.

```
In [8]: ## 4f
        plt.figure(figsize=(10,8))
        num_ISI_bins = 200
        for con in range(num_cons) :
            plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
            #=====================================================#
            # YOUR CODE HERE:
            #   Plot the interspike interval (ISI) distribution and
            #   an exponential distribution with rate given by the inverse
            #   of the mean ISI.
            #=====================================================#
```

```python
relevant_values= []
for ren in range(num_trials):
    relevant_values.append(spike_times[con][ren])
test = []
for i in range(num_trials):
    test.append(np.ndarray.tolist(relevant_values[i]))
flat_list = []
for j in range(num_trials):
    flat_list.append([item for sublist in test[j] for item in sublist])
relevant_ISI = []
for k in range(num_trials):
    relevant_ISI.append([x - flat_list[k][j - 1] for j, x in enumerate(flat_list[k])
flat_list_ISI = [item for sublist in relevant_ISI for item in sublist]
flat_list_mean = np.mean(flat_list_ISI)
flat_list_lambda = 1/flat_list_mean
sampled_flat_list = np.linspace(0, max(flat_list_ISI), 100)
curve = flat_list_lambda*np.exp(-flat_list_lambda*sampled_flat_list)
plt.plot(sampled_flat_list, curve)
bins = 100
plt.hist(flat_list_ISI, bins, density=True)
pass

#=====================================================#
# END YOUR CODE
#=====================================================#
plt.title('ISI distribution, s= '+ s_labels[con]+' radians')
plt.axis([0, max_t, 0, 0.04])
plt.tight_layout()
```



0.2.3 Question:

Why might the empirical distributions differ from the idealized exponential distributions?

Your answer: The empirical distributions differ from the idealized exponential distributions because of the refractory periods which are present and also, it is an inhomogenous poisson process.