Homework 3, Problem 4 on real neural data.

ECE C143A/C243A, Spring Quarter 2022, Prof. J.C. Kao, TAs T. Monsoor, W. Yu.

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 *n*th 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 $\times$ 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.

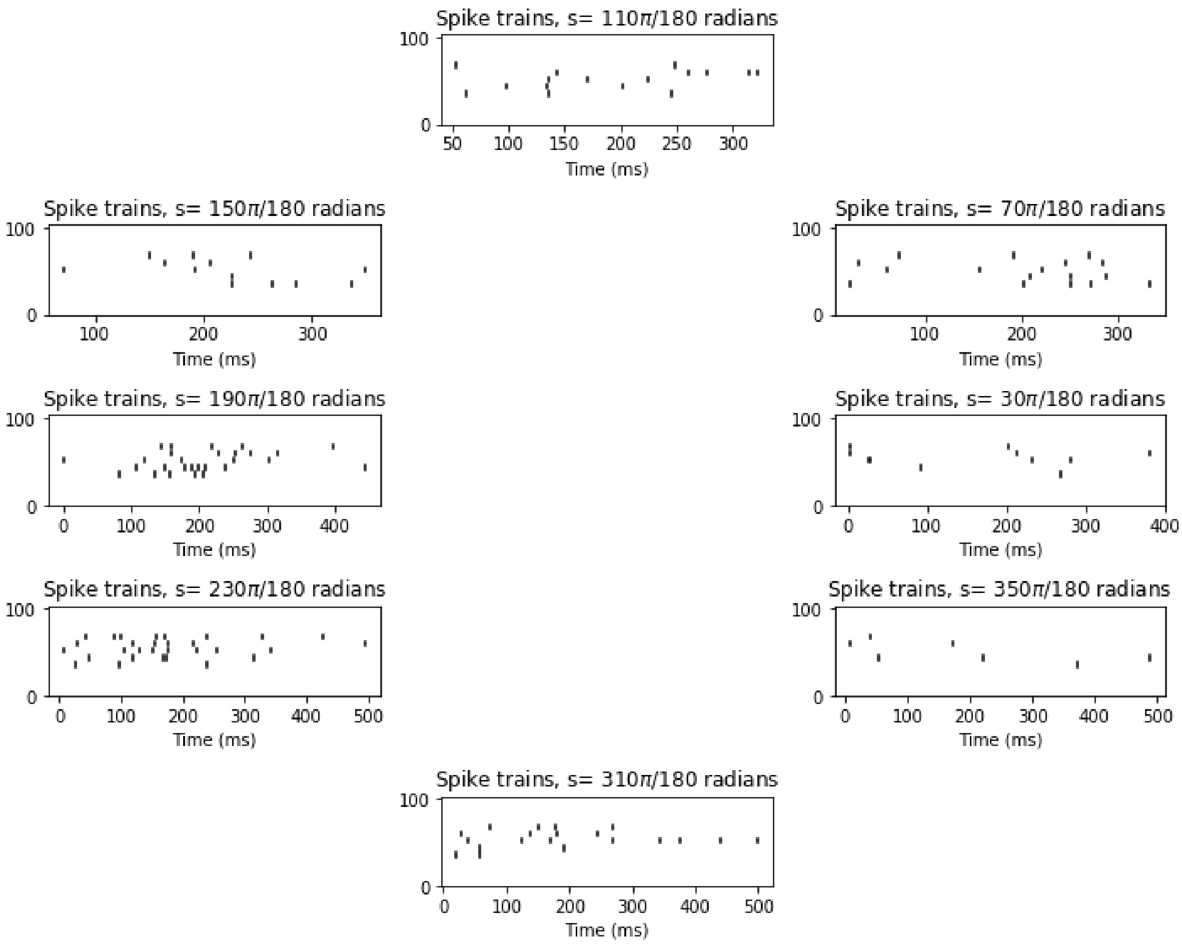
In [ ]:
```
"""
ECE C143/C243 Homework-3 Problem-4

"""

# 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
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

(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

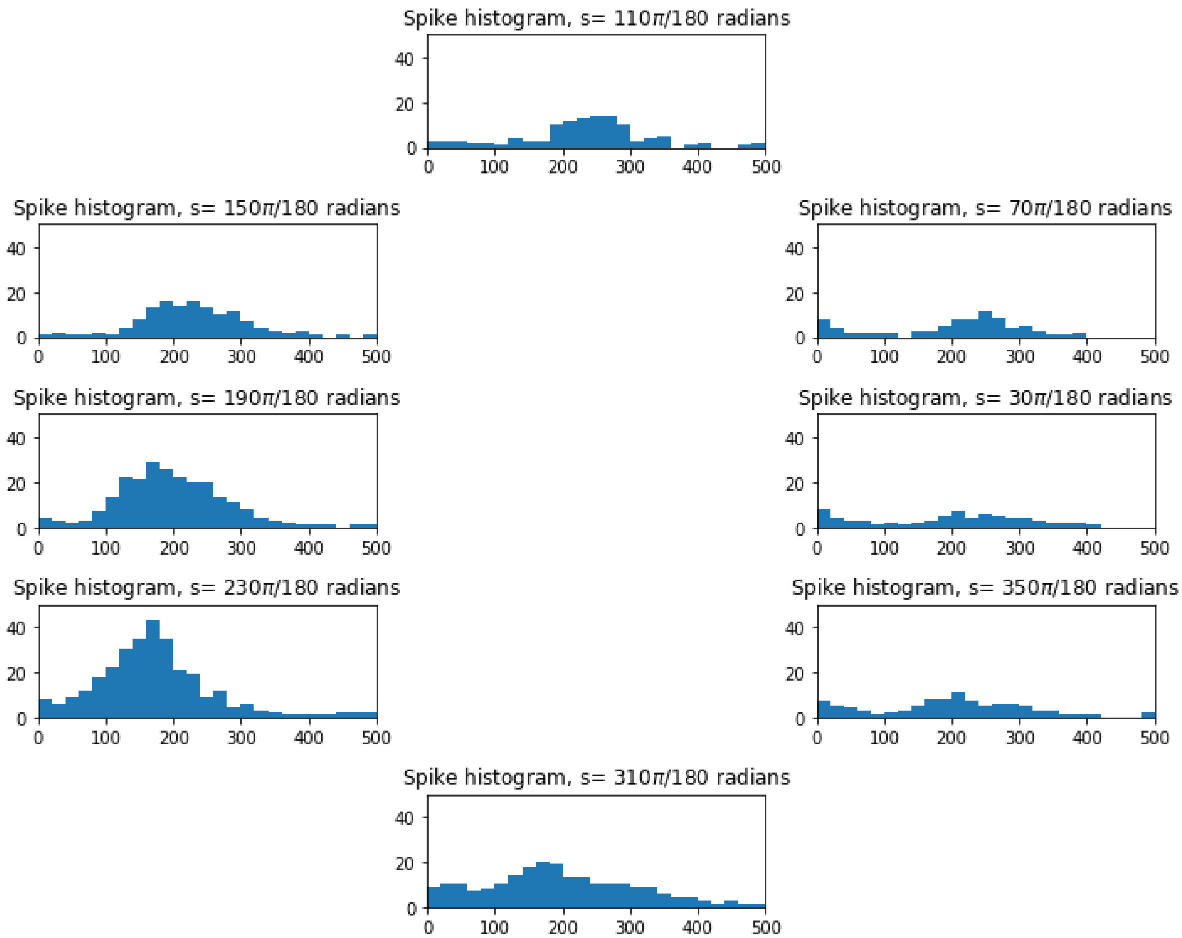
In [ ]:
```
## 4a
# print(data['trial'][0,0]["spikes"][0])


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 ,230.0/1
80 ,310.0/180 ,350.0/180]) # radians
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))
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.
        #=====================================================#
        spike_times[con, rep] = np.where(data['trial'][rep,con]["spikes"][0])[
0]
        #=====================================================#
        # 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()
```



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

    edges = np.arange(0,max_t+bin_width,bin_width)
    heights = np.zeros_like(edges)
    for (i,height) in enumerate(heights):
        if edges[i] < edges[-1]:
            spike_count =0
            for trial in range(num_trials):
                spike_count += np.sum((spike_times[con,trial]<=edges[i+1])*(sp
ike_times[con, trial]>=edges[i]))/num_trials
            heights[i] = (spike_count/20)*1000
    plt.bar(edges, heights, width=20, align='edge')

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



(c) (4 points) Tuning curve

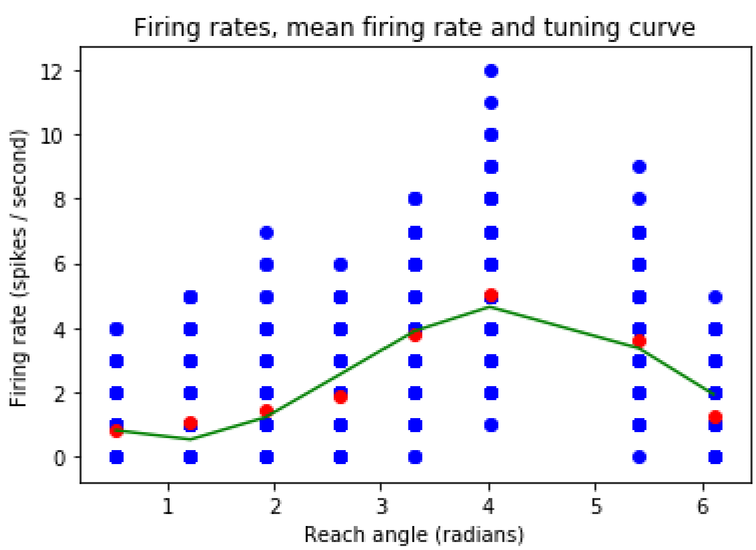
For each trial, count the number of spikes across the entire trial. Plots these points on the axes shown in Figure 1.6(B) in *TN*. There should be $182 \cdot 8$ points in the plot (but some points may be on top of each other due to the discrete nature of spike counts). For each reaching angle, find the 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 \eqn{tuning} 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 [ ]:
```
#=========================================================#
# 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 = np.zeros((num_cons, num_trials))
for con in range(num_cons):
    for rep in range(num_trials):
        spike_counts[con,rep] = int(len(spike_times[con,rep][1:]))
    plt.scatter(np.array([s[con]]*num_trials),spike_counts[con], c='blue')
mean_rates = np.mean(spike_counts,axis=1)
plt.scatter(s,mean_rates, c='red')

# perform regression
A = np.stack((np.ones_like(s).T,np.sin(s).T, np.cos(s).T))
k = np.linalg.pinv(A).T@mean_rates
c0 = k[0]
theta0 = np.arctan2(k[1],k[2])
c1 = k[1]/np.sin(theta0)
tuning = c0 + c1*np.cos(s-theta0)
plt.plot(s, tuning, c='green')

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

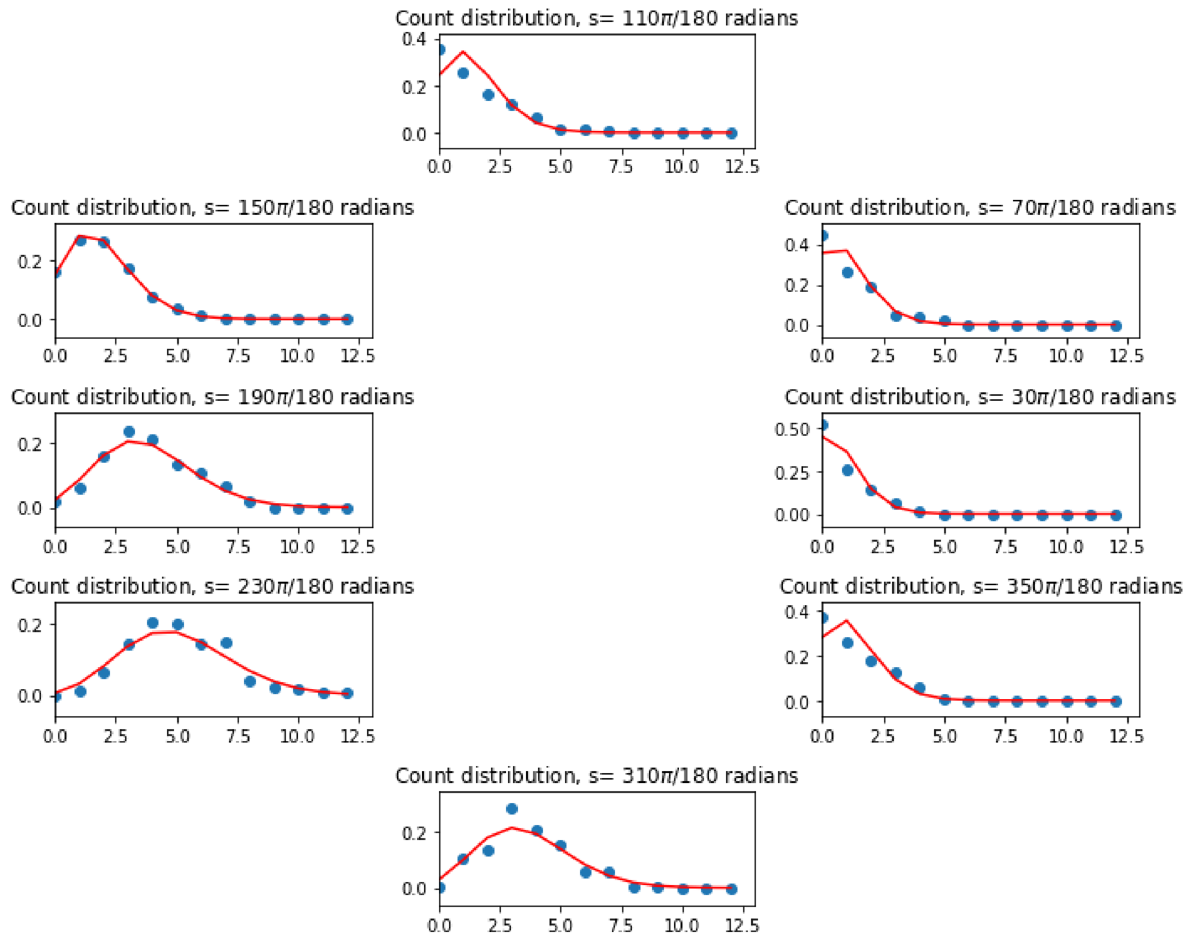
Out[ ]: `Text(0.5,1,'Firing rates, mean firing rate and tuning curve')`



(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 [ ]:
```
plt.figure(figsize=(10,8))
max_count = 13.0
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.
    #=====================================================#
    bins = np.zeros_like(spike_count_bin_centers)
    for i,item in enumerate(bins):
        if spike_count_bin_centers[i]<spike_count_bin_centers[-1]:
            bins[i] = np.sum((spike_counts[con]>=spike_count_bin_centers[i])*(
spike_counts[con]<spike_count_bin_centers[i+1]))/float(num_trials)
        else:
            bins[i] = np.sum((spike_counts[con]>=spike_count_bin_centers[i])*(
spike_counts[con]<max_count+1))/num_trials
    est_rate = mean_rates[con]
    p_dist = np.power(np.array([est_rate]*len(spike_count_bin_centers)),spike_
count_bin_centers)*np.exp(-est_rate)/scipy.special.factorial(spike_count_bin_c
enters)
    #=====================================================#
    # END YOUR CODE
    #=====================================================#


    #=====================================================#
    # YOUR CODE HERE:
    #   Plot the empirical distribution of spike counts and the
    #   Poission distribution you just calculated
    #=====================================================#
    plt.scatter(spike_count_bin_centers,bins)
    plt.plot(spike_count_bin_centers, p_dist, c="red")
    #=====================================================#
    # 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?

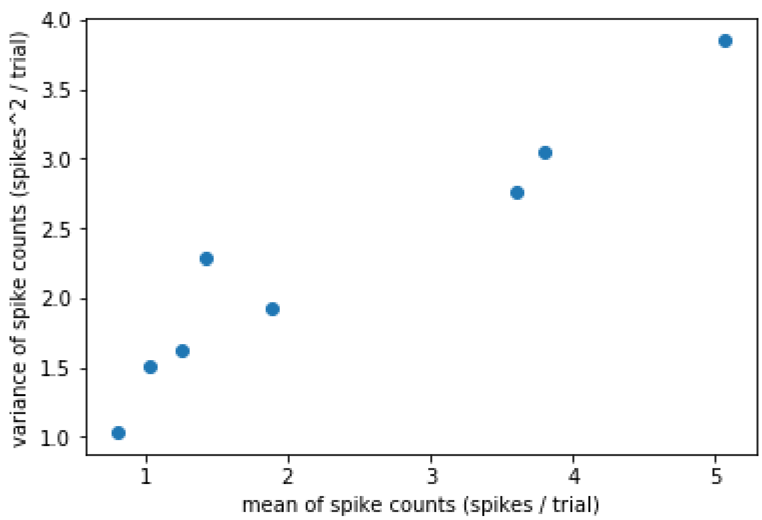
Your answer:

Empirical distributions may differ from an idealized poisson distribution because the poisson distribution is just an approximation. It does not account for the refractory period nor the many other phenomena in the brain that occur. The process is only approximately poisson, and so some deviation from the idealized distribution is expected.

(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 [ ]:
```
## 4e

#========================================================#
# YOUR CODE HERE:
# Plot the mean and variance of spike counts on the axes
#========================================================#
var_rates =np.var(spike_counts,axis=1)
plt.scatter(mean_rates, var_rates)
plt.xlabel("average rate($\\frac{spikes}{s}$)")
plt.ylabel("Var Rates($\\frac{spikes}{s}$)")

#========================================================#
# 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?

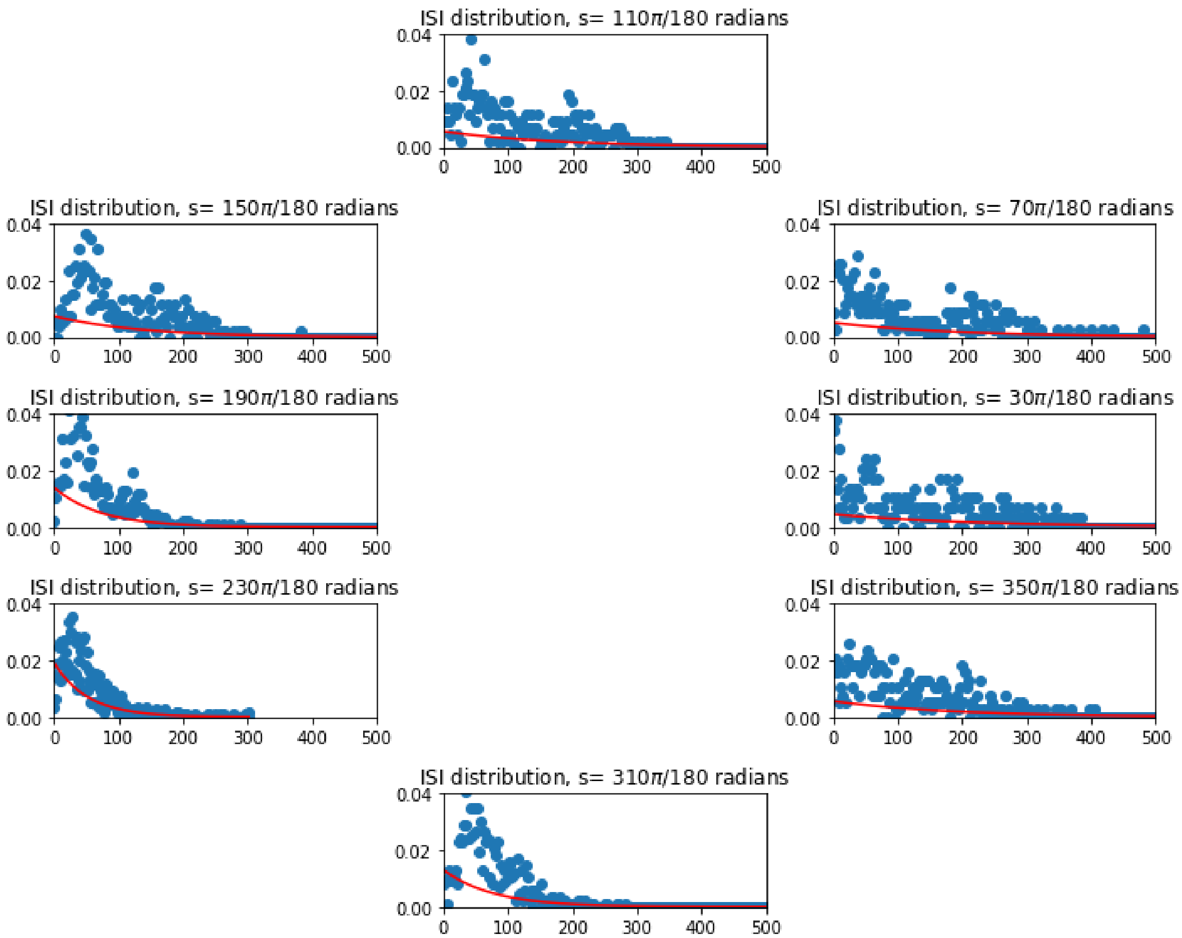
Your answer:

The points are near a 45 degree angle but slightly diverge at a slighter angle. This is indicative of the fact that real data is well approximated by a poisson process but it is not truly a poisson process.

(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 [ ]:
```
## 4f
plt.figure(figsize=(10,8))
num_ISI_bins = 200
avg_ISI = np.zeros(num_cons)
var_ISI = np.zeros(num_cons)
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.
    #======================================================#
    ISI = np.zeros_like(spike_times[con])
    mean_ISI = np.zeros_like(spike_times[con])
    max_ISI = 0.0
    all_ISI = np.empty([0])
    for rep in range(num_trials):
        ISI[rep] = np.diff(np.append([0],spike_times[con,rep]))
        mean_ISI[rep] = np.mean(ISI[rep]) if len(ISI[rep]) else 500
        max_ISI = np.maximum(np.max(ISI[rep]),max_ISI) if len(ISI[rep]) else 5
00
        all_ISI = np.append(all_ISI, ISI[rep])
    avg_ISI[con] = np.mean(mean_ISI)
    var_ISI[con] = np.var(all_ISI)
    lambd = 1/avg_ISI[con]
    ISI_bin_centers =  np.linspace(0,max_ISI,num_ISI_bins)

    bins = np.zeros_like(ISI_bin_centers)
    for i,item in enumerate(bins):
        data_points =0;
        for rep in range(num_trials):
            if ISI_bin_centers[i]<ISI_bin_centers[-1]:
                bins[i] += np.sum((ISI[rep]>=ISI_bin_centers[i])*(ISI[rep]<ISI
_bin_centers[i+1]))
                data_points += len(ISI[rep])
            else:
                bins[i] += np.sum((ISI[rep] >= ISI_bin_centers[i])*(ISI[rep]<m
ax_ISI+1))
                data_points += len(ISI[rep])
        bins[i]/= data_points
    p_dist = lambd*np.exp(-lambd*ISI_bin_centers)
    plt.scatter(ISI_bin_centers, bins)
    plt.plot(ISI_bin_centers, p_dist, c ="red")

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



Question:

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

Your answer:

The empirical distributions differ from the exponential distributions due to the absolute refractory period of the neurons. The neurons are incapable of firing immdeiately after a previous fire, so the empirical distributions apprear closer to gamma distributions rather than the idealized exponential distribtiuion.