1a)

$$Pr(M(S)=m \mid N(S)=n) = \frac{P(N=n \mid M=m)\cdot P(M=m)}{P(N=n)}$$

$$\frac{\cancel{P(N=n)}\, P(M=m)}{\cancel{N(N=n)}} \qquad P_r(N=n) = \frac{(\lambda s)^n e^{-\lambda s}}{n!}$$

$$= P(M=m) = (1-P)$$

Law of Total Probability

$$P(M=m) = \sum_{=0}^{\infty} P_r(M=m, N=n)$$

$$\sum_{n=m}^{\infty} P(M=m \mid N=n) P(N=n)$$

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

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

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

$$x = ((1-P)\lambda s) \quad \text{then} = \frac{x^m e^{-x}}{m!} \approx \boxed{\text{Poisson}((1-P)\lambda s)}$$

1b) $\boxed{\text{Rate} = (1-P)\lambda}$ $\sim \text{Poisson}((1-P)\lambda s)$

1c) $\sim \text{Poisson}(P\lambda s)$ $= \boxed{\frac{(P\lambda \tau)^m e^{-(P\lambda \tau)}}{m!}}$

Dropped

Homework 3, Problem 2 on homogeneous Poisson processes

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

In [4]:
```
import pip
pip.main(['install', 'scipy'])
```

```
WARNING: pip is being invoked by an old script wrapper. This will fail in a future ve
rsion of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the underlyin
g issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip dire
ctly.
Requirement already satisfied: scipy in c:\users\schir\appdata\local\programs\pyt
Requirement already satisfied: numpy<1.25.0,>=1.17.3 in c:\users\schir\appdata\lo
WARNING: You are using pip version 22.0.3; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\schir\AppData\Local\Programs\Pyth
```

Out[4]:
```
0
```

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 `int`s. If instead you write 1.0 / a, you will get out the desired output, since 1.0 is of type `float`.

In [15]:
```
"""
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
```

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

(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. Also note that time at 0 is not a spike.

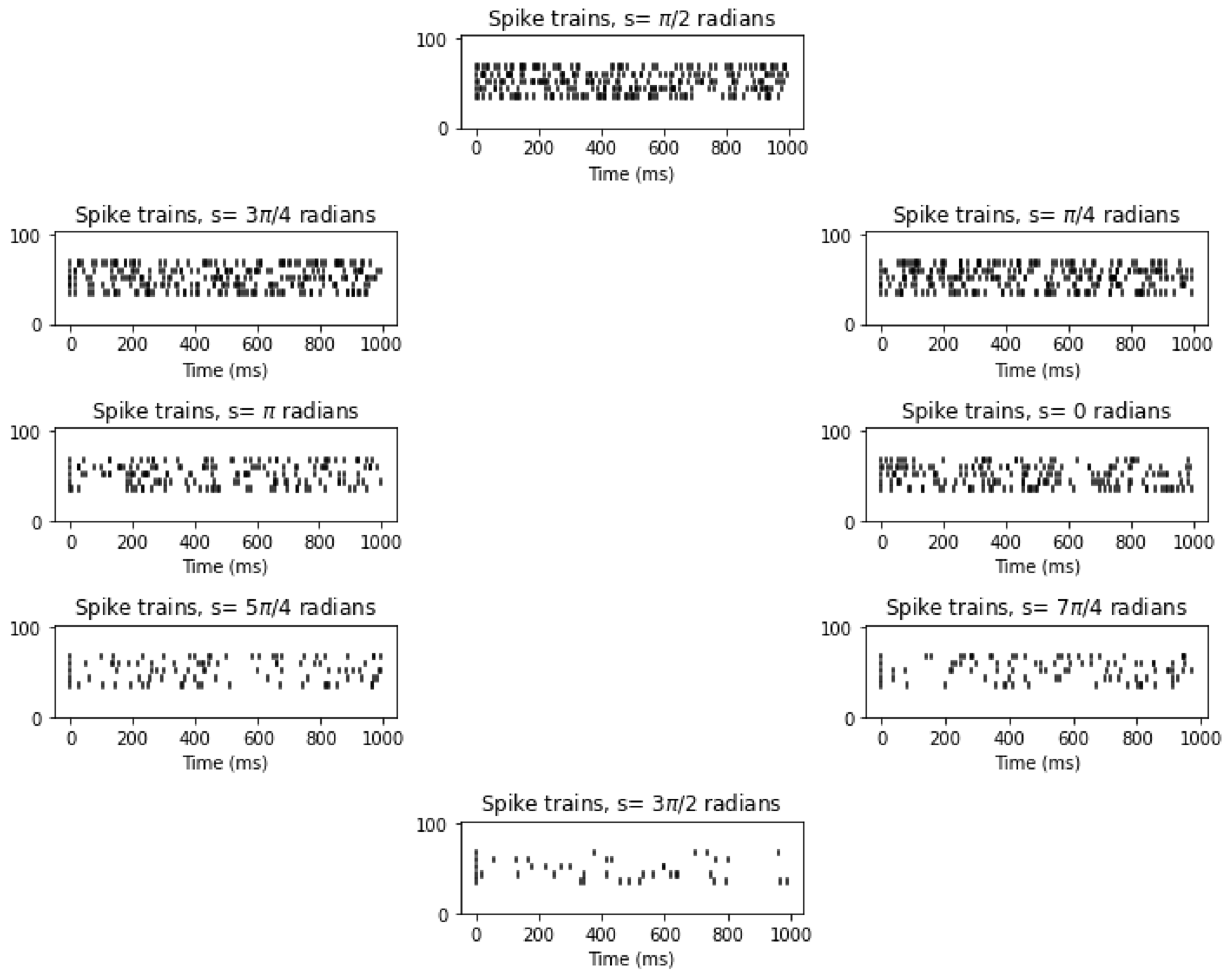
In [17]:

```
## 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 dir
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_times[con, rep] = nsp.GeneratePoissonSpikeTrain(1000,35 + (25 * np.cos((
        #======================================================#
        # END YOUR CODE
        #======================================================#
```

In [18]:

```
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()
```



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.

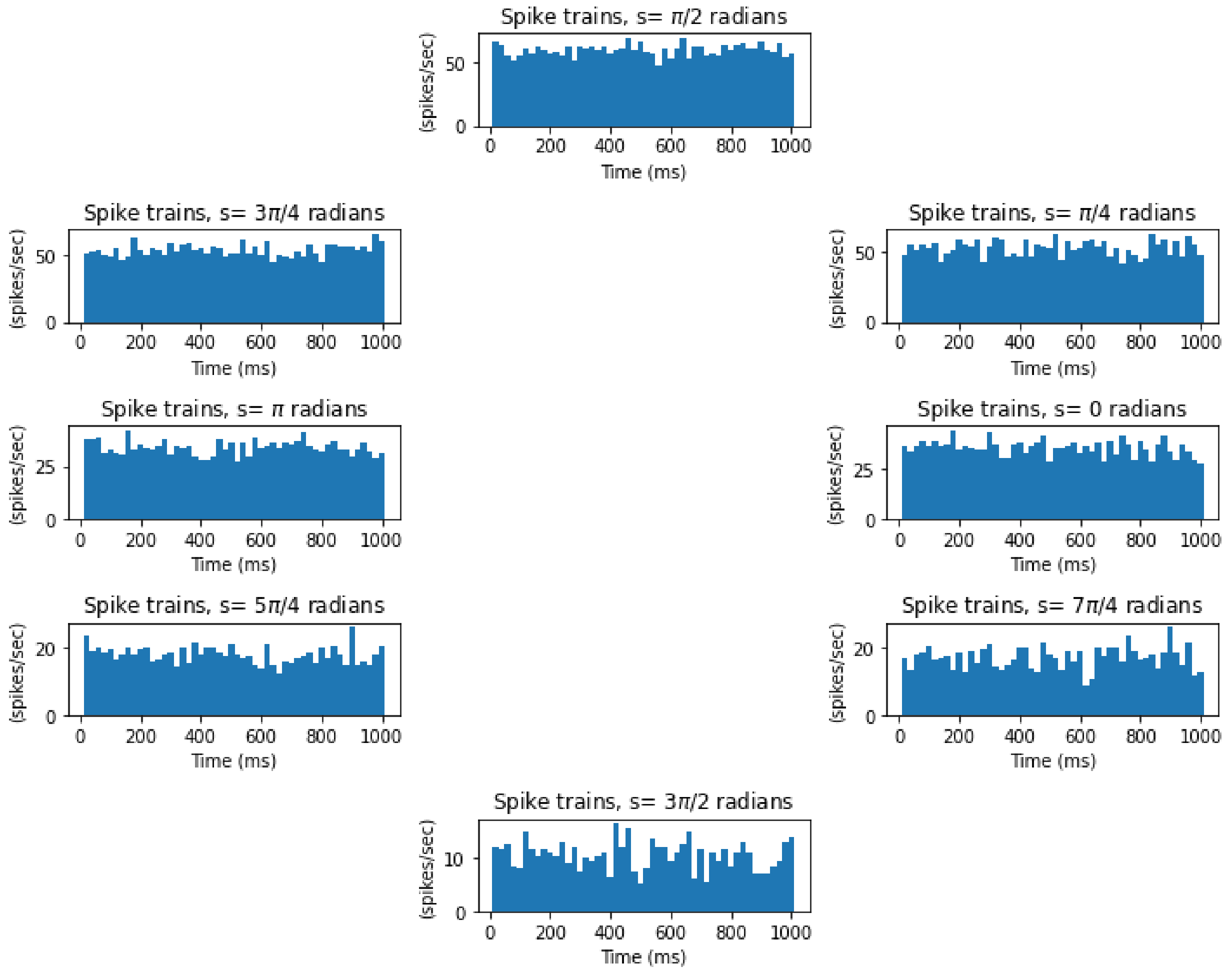
(b) (5 points) Plot spike histograms

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.

Suggestion: you can use plt.bar to plot the histogram, it is important to set the `width` for this function, e.g. width = 12.

In [19]:
```
## 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
    #=====================================================#
    #Bins for firing rates
    avg_fire_rate = np.array([])
    #bins to label time
    time_bin_arr = np.array([])

    prev_bin_time = 0 #ms time bin
    bin_time = 20 #ms time bin
    for time_bin in range(50): #50 total time bins to get to 1000ms

      spike_in_bin = 0
      for sample_spike_train in range(100):
        for spike_time in spike_times[con, sample_spike_train]:
          if (prev_bin_time <spike_time):
            if (spike_time <= bin_time):
              spike_in_bin +=1
            else:
              break #exit checking spikes if above bint time
      #update bin fire rate
      avg_fire_rate = np.append(avg_fire_rate, [spike_in_bin *50/(100)])#100 samples c
      #update bin time in ms
      time_bin_arr = np.append(time_bin_arr, bin_time)

      bin_time += 20
      prev_bin_time += 20
    #print(avg_fire_rate)
    #print(time_bin_arr)
    plt.bar(time_bin_arr, avg_fire_rate, width = 20)
    plt.ylabel("(spikes/sec)")
    plt.xlabel("Time (ms)")

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



(c) (4 points)Tuning curve

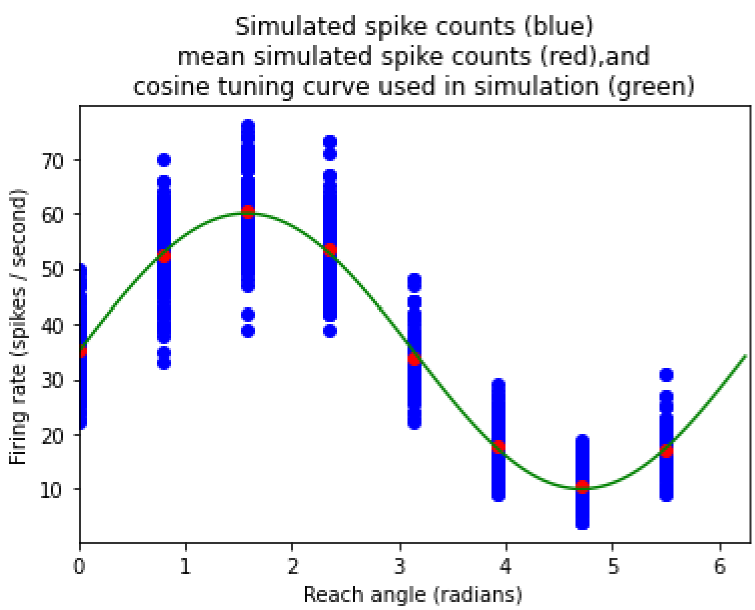
For each trial, count the number of spikes across the entire trial. Plots these points on the axes like shown in Figure 1.6(B) in *TN*, where the x-axis is reach angle and the y-axis is firing rate. There should be 800 points in the plot (but some points may be on top of each other due to the discrete 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 [20]:
```
## 2c
spike_counts = np.zeros((num_cons, num_trials)) # each element in spike_counts is the
#=====================================================#
# YOUR CODE HERE:
#   Plot the single trial spike counts and the tuning curve
#   on top of each other.
#=====================================================#
x_angle_arr = []
spike_rate_arr = []

avg_angle_spike = []

s_idx = 0
for angle in s: #for each angle
  cond_avg = 0
```

```
  for trial in range(100): #for each trial for that angle (100 trials)
     spikes_in_trial = len(spike_times[s_idx, trial]) -1
     spike_rate_arr.append(spikes_in_trial)
     x_angle_arr.append(angle)
     cond_avg += spikes_in_trial
     if (trial == 99):
       avg_angle_spike.append(cond_avg/100)
  s_idx += 1
#plotting
X = np.arange(0, np.pi*2, 0.05)
y = 35 + (60-35) * np.cos(X - (np.pi/2))
plt.plot(X,y, color = 'g', label = 'tuning curve')
plt.scatter(x_angle_arr, spike_rate_arr, color = 'b', label = 'simulated trial average
plt.scatter(s, avg_angle_spike, color = 'r', label = "mean spike rate")

#=====================================================#
# 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[20]:

```
(0.0, 6.283185307179586)
```



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

italicized text#### Your answer: Yes! Also the variance for higher firing rates appears to higher than lower firing rates which is expected.

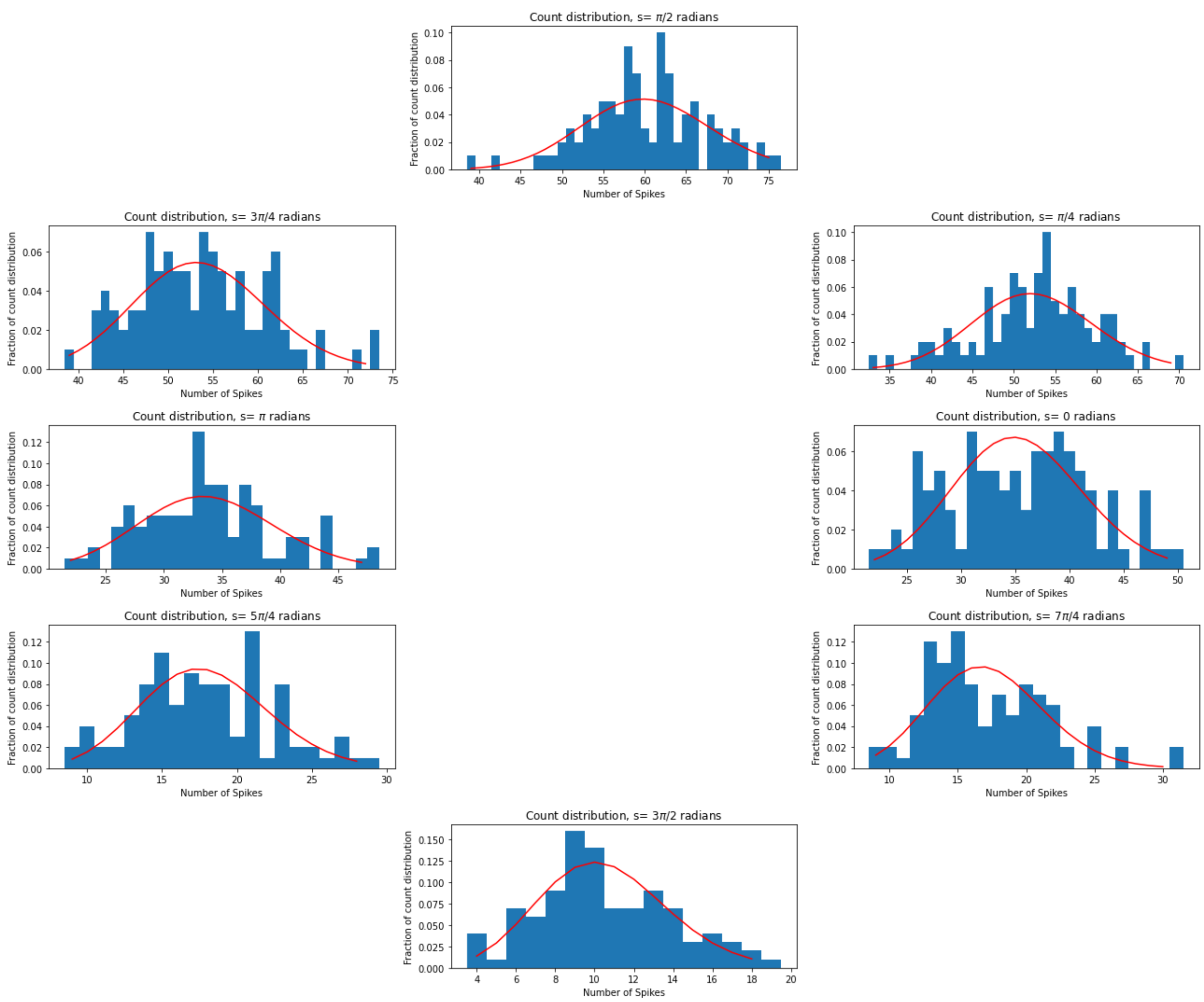
(d) (6 points) Count distribution

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.

Please plot the empirical distribution as well as the fit

In [21]:
```
##2d

plt.figure(figsize=(18,15))
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.
    #=====================================================#
    angle = s[con]
    mean_fire_rate = avg_angle_spike[con]
    poisson_dis= 1 * mean_fire_rate   ## ~poisson(lamda *t) where t is one second
    probs_rate = [] #y axis
    rate_count = [] #x axis

    def Poisson_func(N, poisson_dis):
      """Returns Y probability values for a given Poison dist given Number and lamda_t
      return ((poisson_dis ** N) * (np.exp(-poisson_dis)) / (np.math.factorial(N)))

    #=====================================================#
    # YOUR CODE HERE:
    #   Plot the empirical count distribution, and on top of it
    #   plot your fit Poisson distribution.
    #=====================================================#
    spike_freq_trial = {}
    for trial in range(100): #for each trial for that angle (100 trials)
      spikes_in_trial = len(spike_times[con, trial]) -1
      if spikes_in_trial in spike_freq_trial:
        spike_freq_trial[spikes_in_trial] =  spike_freq_trial[spikes_in_trial] + 1
      else:
        spike_freq_trial[spikes_in_trial] = 1
    for key, value in spike_freq_trial.items():
      probs_rate.append(value/100)
      rate_count.append(key)
    plt.bar(rate_count, probs_rate , width = 1, label = 'Empirical Count Distribution'

    #graph perfect possion
    count_max = max(rate_count)
    count_min = min(rate_count)
    X = np.arange(count_min, count_max, 1)
    Y = []
    for x in X:
      Y.append(Poisson_func(x, poisson_dis))
    plt.plot(X, Y, color = 'r', label = 'Fit Poisson Distribution')
    #plt.legend(loc="upper left")
    plt.ylabel('Fraction of count distribution')
```

```
    plt.xlabel('Number of Spikes')


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



Question:

Are the empirical distributions well-fit by Poisson distributions?

Your answer:

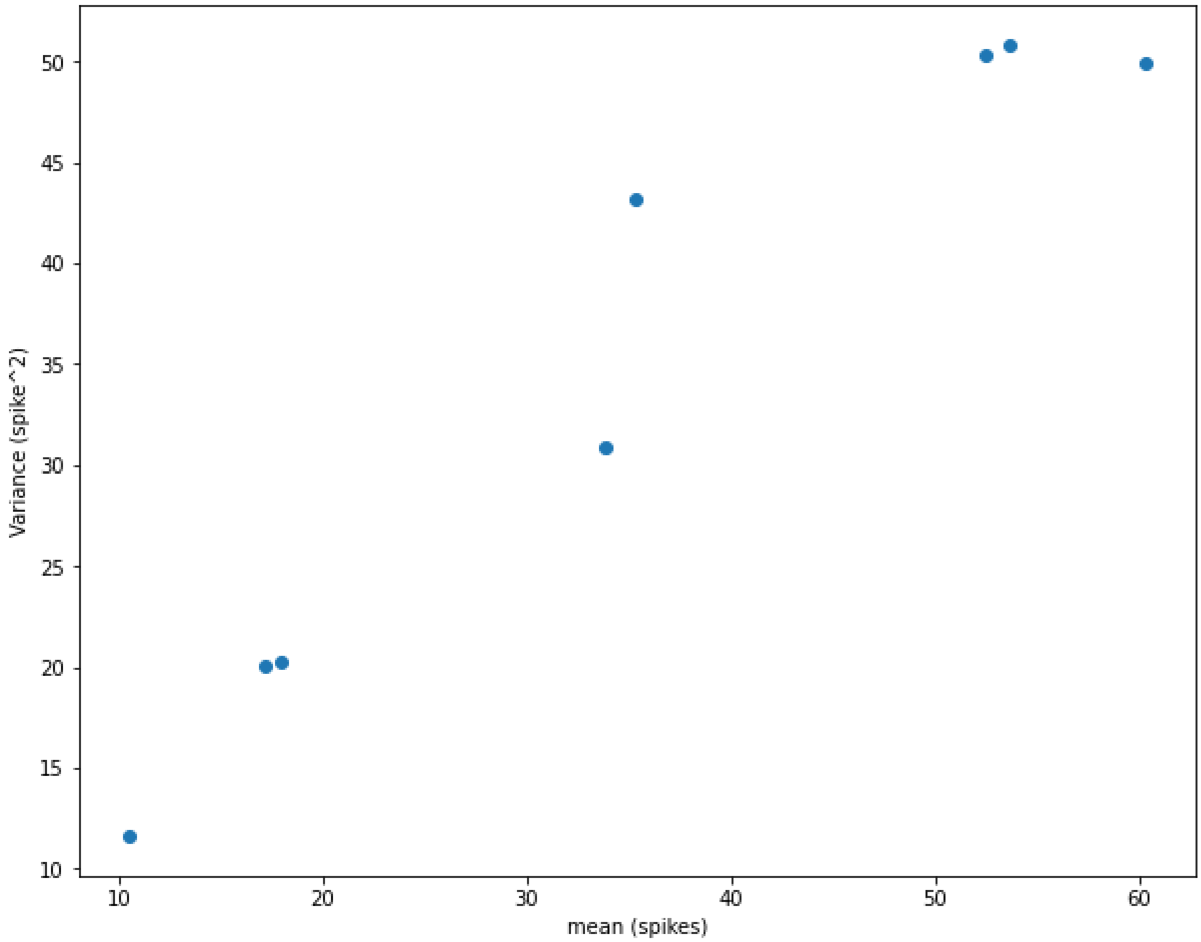
Pretty well yes though obviously not perfect since it is discrete.

(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 [22]:
```
## 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.
#==================================================#
var_arr = []
index_mean = 0
plt.figure(figsize=(10,8))
s_idx = 0
for angle in s: #for each angle
  angle_mean = avg_angle_spike[index_mean]
  error_sum = 0
  for trial in range(100): #for each trial for that angle (100 trials)
    spikes_in_trial = len(spike_times[s_idx, trial]) -1
    error_sum += (spikes_in_trial-angle_mean)**2
  var_arr.append(error_sum/100)
  index_mean += 1
  s_idx += 1

plt.scatter(avg_angle_spike, var_arr)
plt.xlabel("mean (spikes)")
plt.ylabel("Variance (spike^2)")
#==================================================#
# END YOUR CODE
#==================================================#
```

Out[22]:
```
Text(0, 0.5, 'Variance (spike^2)')
```



Question:

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

Your answer: Yes they do!!!

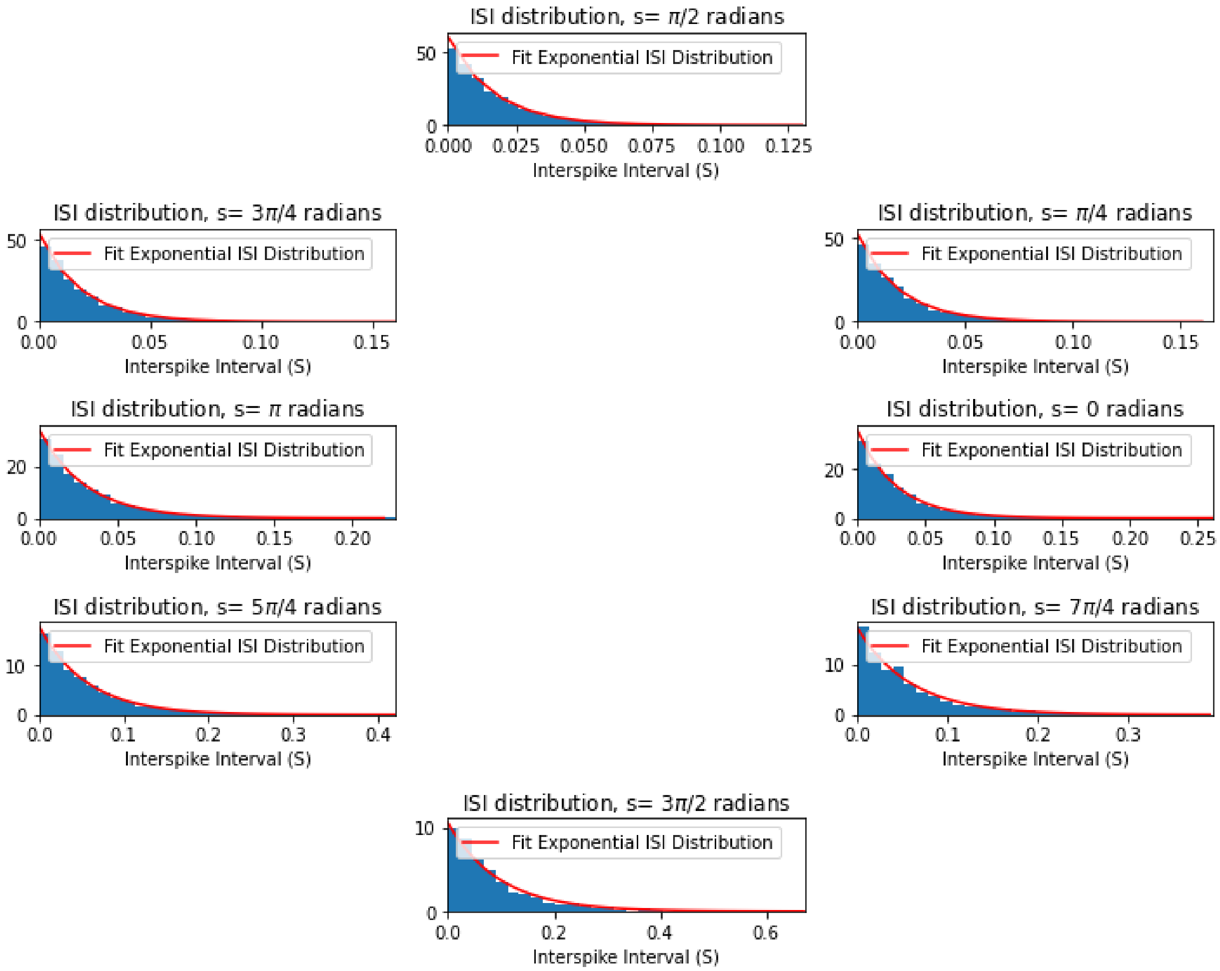
(f) (5 points) Interspike interval (ISI) distribution

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.

Please plot the empirical distribution as well as the fit

In [23]:
```
## 2f
plt.figure(figsize=(10,8))
ISI_by_Angle =[]
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.
#====================================================#
def ISI_value(lamda, t):
  return (lamda * np.exp(-1 *lamda *t))
mean_fire_rate = avg_angle_spike[con]
lamda = mean_fire_rate
#time_dist_arr = []
ISI_arr = []
prev_time = 0
for trial in range(100): #for each trial for that angle (100 trials)
  for spike_time in spike_times[con, trial]:
    #if reached the end break
    if (spike_time == spike_times[con, trial][-1]):
      break
    if (spike_time != 0):
      isi = (spike_time - prev_time)/1000
      ISI_arr.append(isi)
    prev_time = spike_time
plt.hist(ISI_arr, bins = 30, density = True)
ISI_by_Angle.append(ISI_arr)
#====================================================#
# END YOUR CODE
#====================================================#

#====================================================#
# YOUR CODE HERE:
#   Plot Interspike interval (ISI) distribution
#====================================================#
X = np.arange(0, max(ISI_arr), .01) #loingest interval of .25 sec
Y = []
for t in X:
  Y.append(ISI_value(lamda, t))
plt.plot(X, Y, color = 'r', label = 'Fit Exponential ISI Distribution')
plt.legend(loc="upper left")
#plt.ylabel('Number of occurnences ISI"s')
plt.xlabel('Interspike Interval (S)')
#====================================================#
# END YOUR CODE
#====================================================#
plt.title('ISI distribution, s= '+ s_labels[con]+' radians')
plt.tight_layout()
plt.xlim(0, max(ISI_arr))
```



Question:

Are the empirical distributions well-fit by exponential distributions?

Your answer:

Yes!!

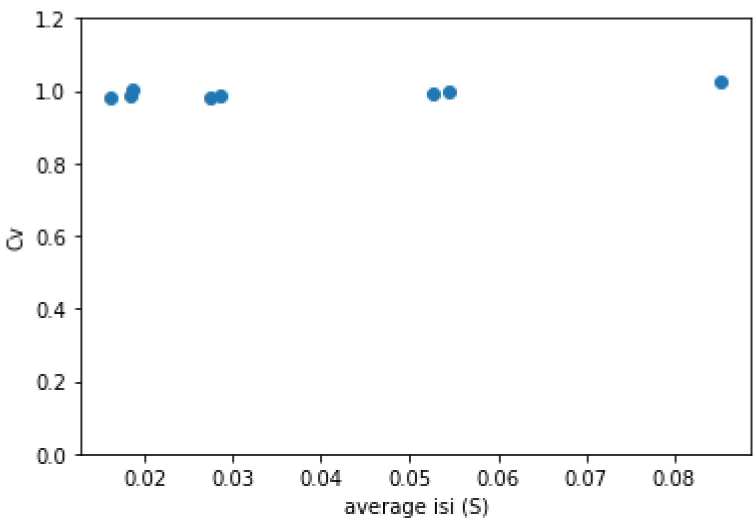
(g) (5 points) Coefficient of variation (C_V)

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 [25]:
```
#2g
#=====================================================#
# YOUR CODE HERE:
# Calculate and plot coeffcient of variation
#=====================================================#
X = np.arange(0, .25, .01)
CV = []
mean_isi = []
for angle in ISI_by_Angle:
  mean = np.mean(angle)
  std = np.std(angle)
  cv = std/mean
  CV.append(cv)
  mean_isi.append(mean)
```

```
plt.ylim(0,1.2)
plt.scatter(mean_isi, CV )
plt.xlabel('average isi (S)')
plt.ylabel('Cv')



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

Out[25]: Text(0, 0.5, 'Cv')



Question:

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

Your answer:

Yes it does

Homework 3, Problem 3 on inhomogeneous Poisson processes

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

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 [5]:
```
import pip
pip.main(['install', 'scipy'])
```

```
WARNING: pip is being invoked by an old script wrapper. This will fail in a future ve
rsion of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the underlyin
g issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip dire
ctly.
Requirement already satisfied: scipy in c:\users\schir\appdata\local\programs\pyt
Requirement already satisfied: numpy<1.25.0,>=1.17.3 in c:\users\schir\appdata\lo
WARNING: You are using pip version 22.0.3; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\schir\AppData\Local\Programs\Pyth
```

Out[5]: 0

In [6]:
```
"""
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
```

(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,

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

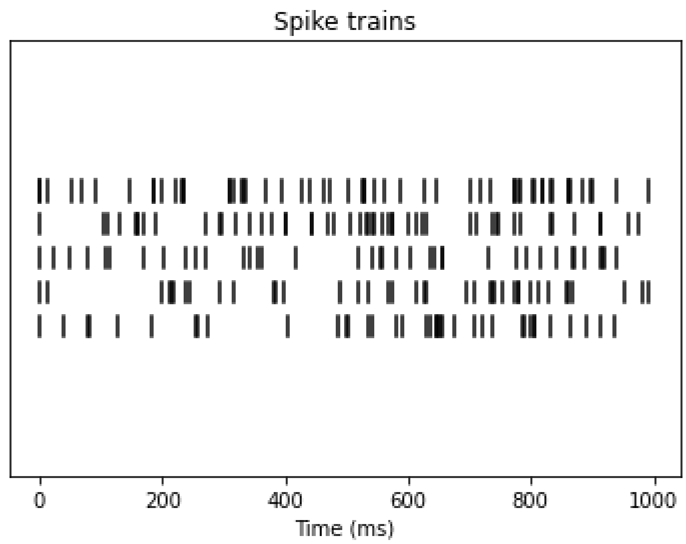
In [7]:
```
r_0 = 35 # (spikes/s)
r_max = 60 # (spikes/s)
s_max = np.pi/2 # (radians)
T = 1000 # trial length (ms)
```

In [8]:
```
np.random.exponential(1.0/r_max * 1000)
```

Out[8]:
```
45.591543952190094
```

In [9]:
```
## 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 GenerateInhomogenousSpikeTrain(T):
  spike_train = np.array(0)
  time = 0
  #T is in ms
  while time <=T:
    reach_angle_s = ((time/1000) **2) * np.pi
    rate = 35 + 25 * np.cos((reach_angle_s - np.pi/2) )
    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]
  return spike_train


spike_times = np.empty((1, num_trials), dtype=list)
for rep in range(num_trials):
  spike_times[0, rep] = GenerateInhomogenousSpikeTrain(1000)


nsp.PlotSpikeRaster(spike_times[0, 0:num_rasters_to_plot])
plt.title('Spike trains')
plt.yticks([])




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

Out[9]:
```
([], [])
```



(b) (5 points) Spike histogram

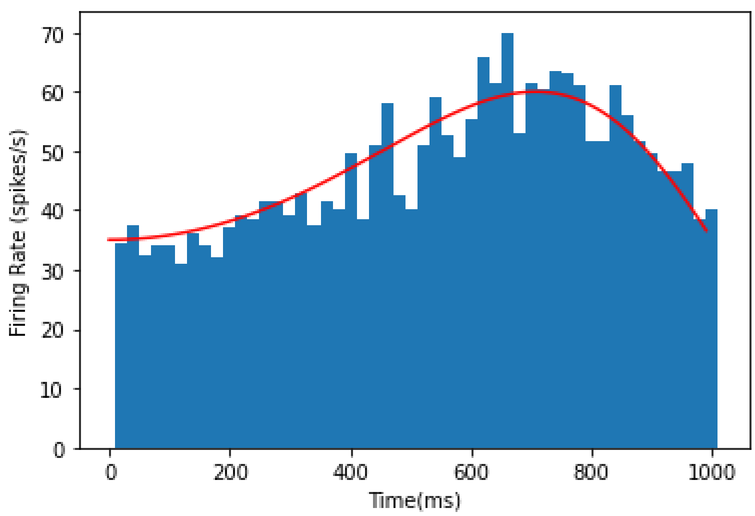
Plot the spike histogram by taking spike counts in non-overlapping 20 ms bins, then averaging across the 100 trials. The spike histogram should have firing rate (in spikes / second) as the vertical axis and time (in msec, not time bin index) as the horizontal axis. Plot the expected firing rate profile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

In [19]:

```
# 3b
bin_width = 20 # (ms)
#====================================================#
# YOUR CODE HERE:
#   Plot the spike histogram
#====================================================#
#Bins for firing rates
avg_fire_rate = np.array([])
#bins to label time
time_bin_arr = np.array([])




prev_bin_time = 0 #ms time bin
bin_time = 20 #ms time bin
for time_bin in range(50): #50 total time bins to get to 1000ms
    spike_in_bin = 0
    for sample_spike_train in range(100):
      for spike_time in spike_times[0, sample_spike_train]:
        if (prev_bin_time <spike_time):
          if (spike_time <= bin_time):
            spike_in_bin +=1
          else:
            break #exit checking spikes if above bin time
    #update bin fire rate
    avg_fire_rate = np.append(avg_fire_rate, [spike_in_bin *50/(100)])#100 samples and
    #update bin time in ms
    time_bin_arr = np.append(time_bin_arr, bin_time)
    bin_time += 20
    prev_bin_time += 20
```

```
x = np.arange(0,1, .01)
Y = []
for X in x:
    ST = (X**2)*np.pi
    Y.append(35 + 25*np.cos(ST-np.pi/2))

plt.plot(x*1000, Y, c = 'r')
plt.bar(time_bin_arr, avg_fire_rate, width = 20)
plt.ylabel('Firing Rate (spikes/s)')
plt.xlabel('Time(ms)')
#=====================================================#
# END YOUR CODE
#=====================================================#
```

Out[19]: Text(0.5, 0, 'Time(ms)')



Question:

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

Your Answer:

Yes it Does!

(c) (6 points) Count distribution

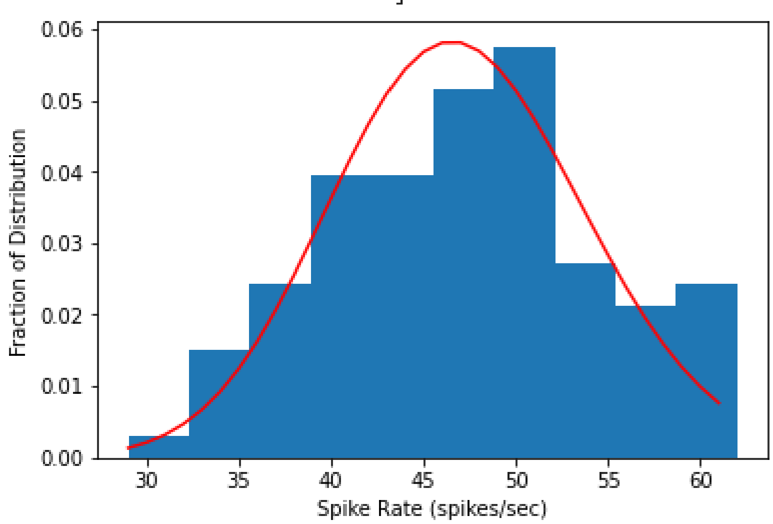
For each trial, count the number of spikes across the entire trial. Plot the normalized distribution of spike counts. Fit a Poisson distribution to this empirical distribution and plot it on top of the empirical distribution.

In [11]:

```
#=====================================================#
# YOUR CODE HERE:
#   Plot the normalized distribution of spike counts
#=====================================================#
#number of spikes seen in each sample
Num_spikes = []
Spike_times = []
Sample_rates = []
```

```
for sample in range(100):
  sample_arr = []
  for spike_time in spike_times[0, sample]:
    sample_arr.append(spike_time)
  Spike_times.append(sample_arr)
  Num_spikes.append(len(sample_arr)-1)
for st in Num_spikes:
  Sample_rates.append(st)
avg_rate = np.mean(Sample_rates)

plt.hist(Num_spikes, density = True)


def Poisson_func(N, poisson_dis):
      """Returns Y probability values for a given Poison dist given Number and lamda_t
      return ((poisson_dis ** N) * (np.exp(-poisson_dis)) / (np.math.factorial(N)))
count_max = round(max(Sample_rates))
count_min = round(min(Sample_rates))
print(count_max)
X = np.arange(count_min, count_max, 1)
print("hi")
print(X)
poisson_dis = 1 * avg_rate
Y = []
for x in X:
  Y.append(Poisson_func(x, poisson_dis))
plt.plot(X, Y, color = 'r', label = 'Fit Poisson Distribution')



#=====================================================#
# END YOUR CODE
#=====================================================#
plt.xlabel('Spike Rate (spikes/sec)')
plt.ylabel('Fraction of Distribution')
plt.show()
```

```
62
hi
[29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
 53 54 55 56 57 58 59 60 61]
```



Question:

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

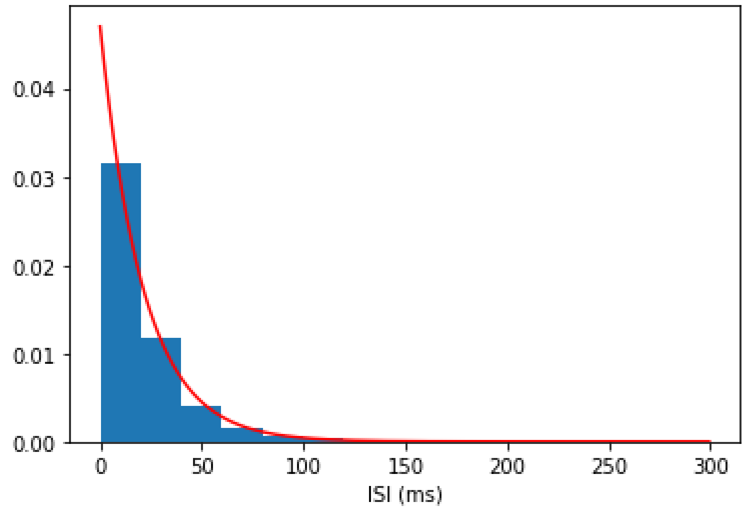
Your Answer:

It will not be modeled like a homogenous Poisson proceess since the rate is changing (inhomogenous process).

(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 [12]:
```
#====================================================#
# YOUR CODE HERE:
#   Plot the normalized distribution of ISIs
#====================================================#

#spike ISIs for each sample
Spike_ISIs = []
ISI = []

for sample in Spike_times:
  prev_time = 0
  isi_arr = []
  for st in sample:
    cur_time = st
    if (st != 0):
      if (st == sample[-1]):
        Spike_ISIs.append(isi_arr)
        break
      isi = cur_time - prev_time
      isi_arr.append(isi)
    prev_time = cur_time
for sample in Spike_ISIs:
  for isi in sample:
    ISI.append(isi)
plt.hist(ISI, bins = 15, density = True)


max_isi = max(ISI)
x = np.arange(0, max_isi, .05)
y = avg_rate/1000 * np.exp(-1 * avg_rate/1000 * x)
plt.plot(x,y, c='r')


#====================================================#
# END YOUR CODE
#====================================================#
plt.xlabel('ISI (ms)')
plt.ylabel('')
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: No we should not though it does sort of look like it

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 $n = 1, \dots, 182$ and $k = 1, \dots, 8$. The indices $k = 1, \dots, 8$ 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 [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
from scipy.optimize import curve_fit

# 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
```

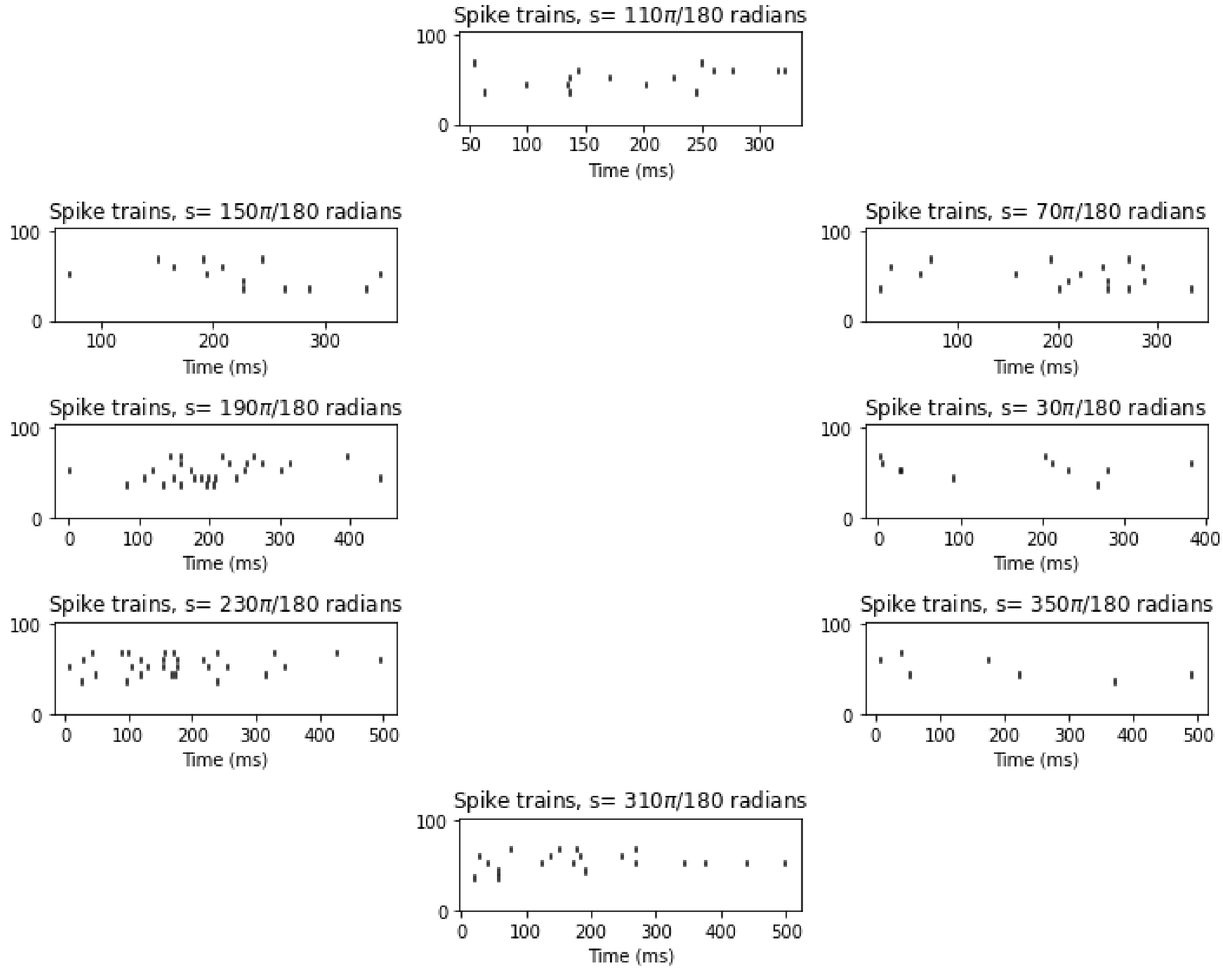
(a) (6 points) Spike trains

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 [2]:
```
## 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.
s_labels = ['30$\pi$/180', '70$\pi$/180', '110$\pi$/180', '150$\pi$/180', '190$\pi$/18
            '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)

print(array for array in data['trial'])

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_arr=  data['trial'][rep,con][1]
        spike_arr = spike_arr[0]
        time = 0
        spike_raster = []
        for spike in spike_arr:
          time += 1
          if (spike == 1):
            spike_raster.append(time)
        spike_times[con, rep] = np.array(spike_raster)
        #====================================================#
        # 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()
```

```
<generator object <genexpr> at 0x7ff33ec08c50>
```

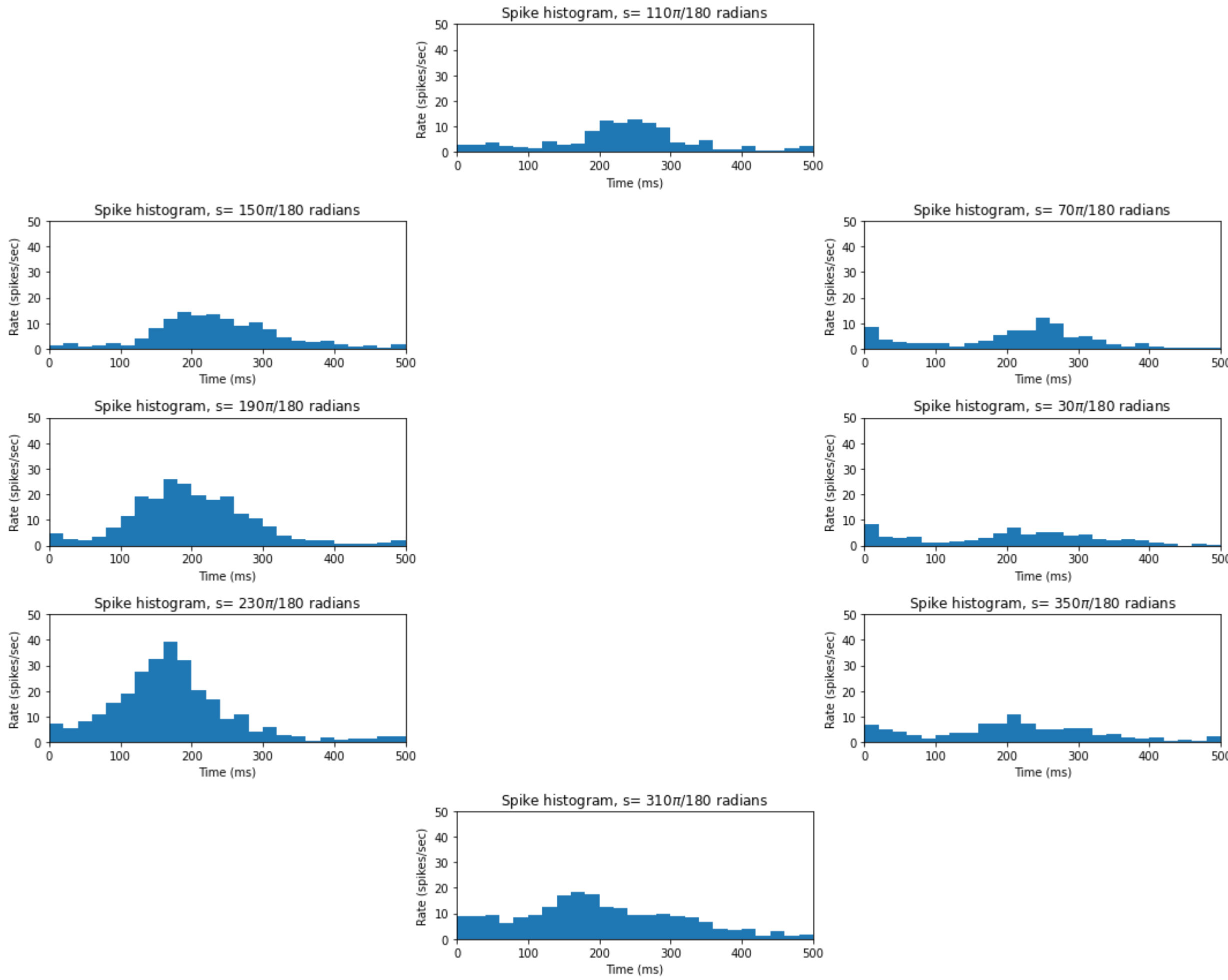


(b) (5 points) Spike histogram

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 [3]:
```
## 4b
bin_width = 20 # (ms)
bin_centers = np.arange(bin_width/2,T,bin_width) # (ms)
plt.figure(figsize=(15,12))
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
    #=====================================================#
```

```
    con_spikes = []
    for rep in range(num_trials):
      for spikes in spike_times[con, rep]:
        con_spikes.append(spikes)
    con_spikes.sort()
    prev_time = 0
    curr_time = 20
    rate_bins = []
    while prev_time < 500:
      num_spikes = 0
      for time in con_spikes:
        if time > curr_time:
          break
        if (prev_time < time) and (time < curr_time):
          num_spikes +=1
      avg_rate = num_spikes/(186 * .02)
      rate_bins.append(avg_rate)
      prev_time = curr_time
      curr_time += 20
    bin_arr = []
    start = 10
    for nums in range(25):
      bin_arr.append(start + nums*20)
    plt.bar(bin_arr, rate_bins, width = 20)
    plt.xlabel('Time (ms)')
    plt.ylabel('Rate (spikes/sec)')




    #plt.hist(con_spikes, bins =range(0, max_t, bin_width)


    #=====================================================#
    # 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

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

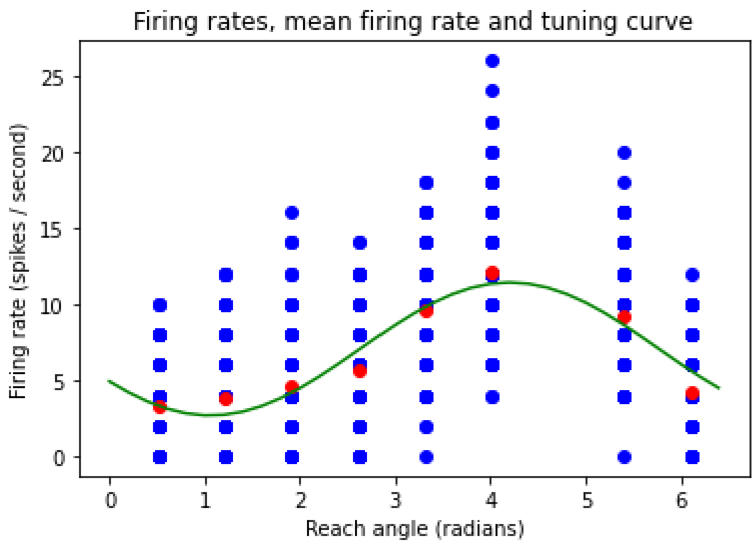
with respect to the parameters r_0, $r_{\max}$, and $s_{\max}$. (Hint: this can be done using linear regression; refer to Homework # 2.) Plot the resulting tuning curve of this neuron in green on the same plot.

In [4]:
```
#======================================================#
# YOUR CODE HERE:
# Tuning curve. Please use the following colors for plot:
# Firing rates(blue);Mean firing rate(red); Cosine tuning curve(green)
#======================================================#
X = []
for con in s:
```

```
  for trial in range(num_trials):
    X.append(con)

Y = []
Mean_rates =[]
for con in range(num_cons):
  avg = []
  for rep in range(num_trials):
    spikes_in_trial = len(spike_times[con, rep])
    spike_rate = spikes_in_trial/.5
    Y.append(spike_rate)
    avg.append(spike_rate)
  Mean_rates.append(np.mean(avg))
plt.scatter(X,Y, c= 'b')
plt.scatter(s, Mean_rates, color = 'r')

def curve(x, r0, rmax, smax):
  return r0 + (rmax-r0)*np.cos(x-smax)

constants, pcov = curve_fit(curve, s, Mean_rates)
r0 = constants[0]
print(f'r0 is {r0}')
rmax = constants[1]
print(f'rmax is {rmax}')
smax = constants[2]
print(f'smax is {smax}')

theta_trend = np.arange(0, 6.5, .2)
Y_fit = []
for theta in theta_trend:
  Y_fit.append(r0 + (rmax-r0)*np.cos(theta-smax))
plt.plot(theta_trend, Y_fit, c = 'g')








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

```
r0 is 7.041337138664717
rmax is 2.6625133665060803
smax is 1.062486805375495
```

Out[4]:
```
Text(0.5, 1.0, 'Firing rates, mean firing rate and tuning curve')
```

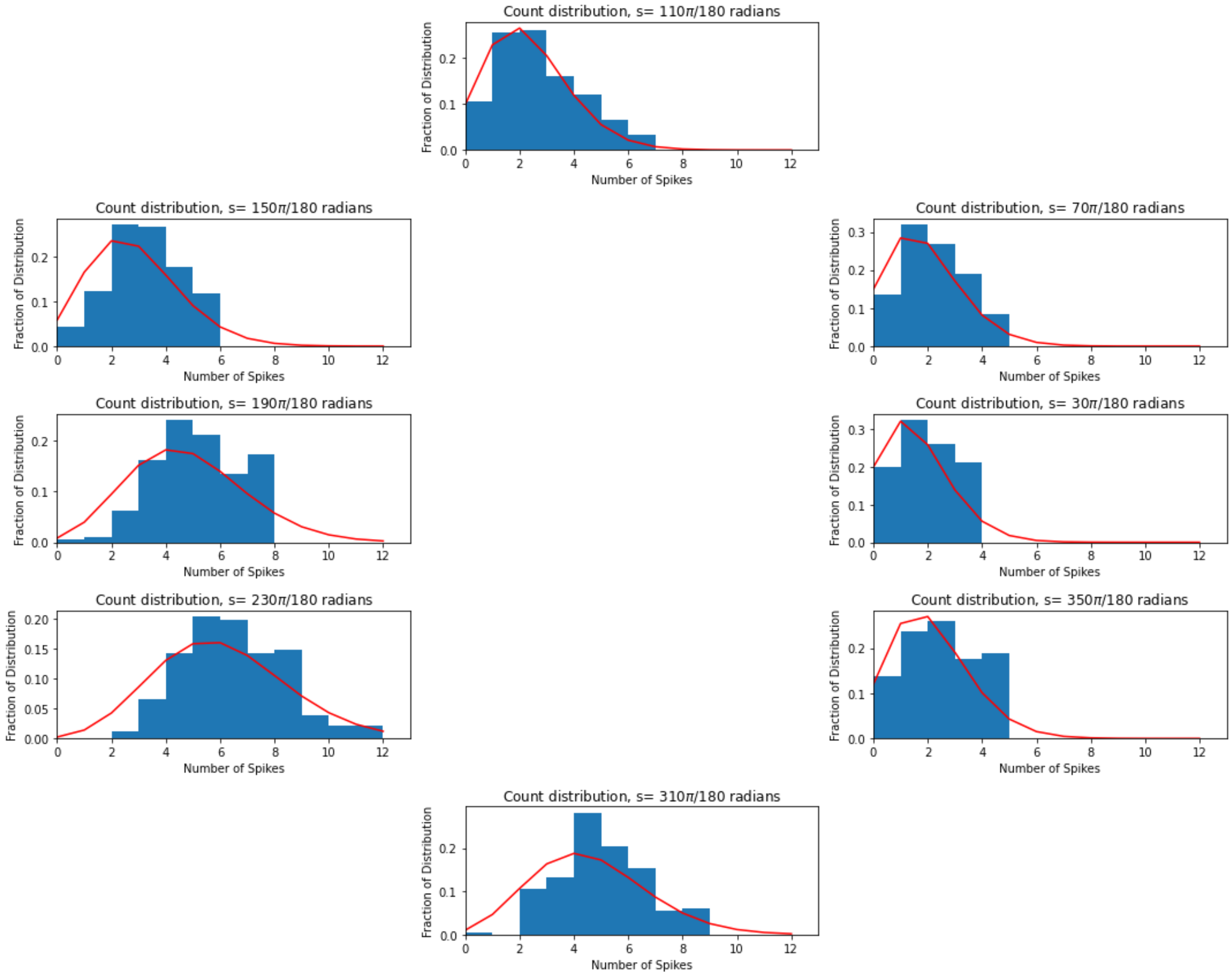


(d) (6 points) Count distribution

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 [5]:
```
plt.figure(figsize=(15,12))
max_count = 13
spike_count_bin_centers =  np.arange(0,max_count,1)
mean_counts = []
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.
    #=====================================================#
    counts = []
    for trial in range(num_trials):
      spikes_in_trial = len(spike_times[con, trial])
      counts.append(spikes_in_trial)
    emp_mean = np.mean(counts)
    mean_counts.append(emp_mean)
    #=====================================================#
    # END YOUR CODE
    #=====================================================#


    #=====================================================#
    # YOUR CODE HERE:
    #   Plot the empirical distribution of spike counts and the
    #   Poission distribution you just calculated
    #=====================================================#
    def Poisson_func(N, poisson_dis):
      """Returns Y probability values for a given Poison dist given Number and lamda_t
      return ((poisson_dis ** N) * (np.exp(-poisson_dis)) / (np.math.factorial(N)))

    plt.xlabel('Number of Spikes')
```

```
    plt.ylabel('Fraction of Distribution')
    plt.hist(counts, bins = range(0, max(counts), 1), density = True)
    fit_x = np.arange(0, 13, 1)
    fit_y = []
    for x in fit_x:
      fit_y.append(Poisson_func(x, emp_mean))
    plt.plot(fit_x, fit_y, color = 'r')

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

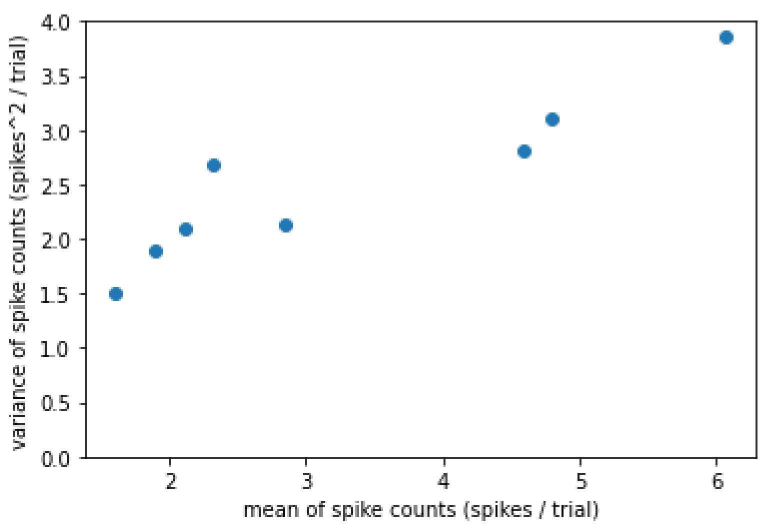
Your answer:

Because the rate is likely not a honogenous exponential process so it will not fit pefectly in a poisson distribution.

(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 [6]:
```
## 4e

#====================================================#
# YOUR CODE HERE:
# Plot the mean and variance of spike counts on the axes
#====================================================#
var = []
mean = []

for con in range(num_cons):
  num_spikes = []
  for trial in range(num_trials):
    spikes_in_trial = len(spike_times[con, trial])
    num_spikes.append(spikes_in_trial)
  var.append(np.var(num_spikes))
  mean.append(np.mean(num_spikes))

plt.scatter(mean, var)
#====================================================#
# END YOUR CODE
#====================================================#
plt.xlabel('mean of spike counts (spikes / trial)')
plt.ylabel('variance of spike counts (spikes^2 / trial)')
plt.ylim(0,4)
plt.show()
```



Question:

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

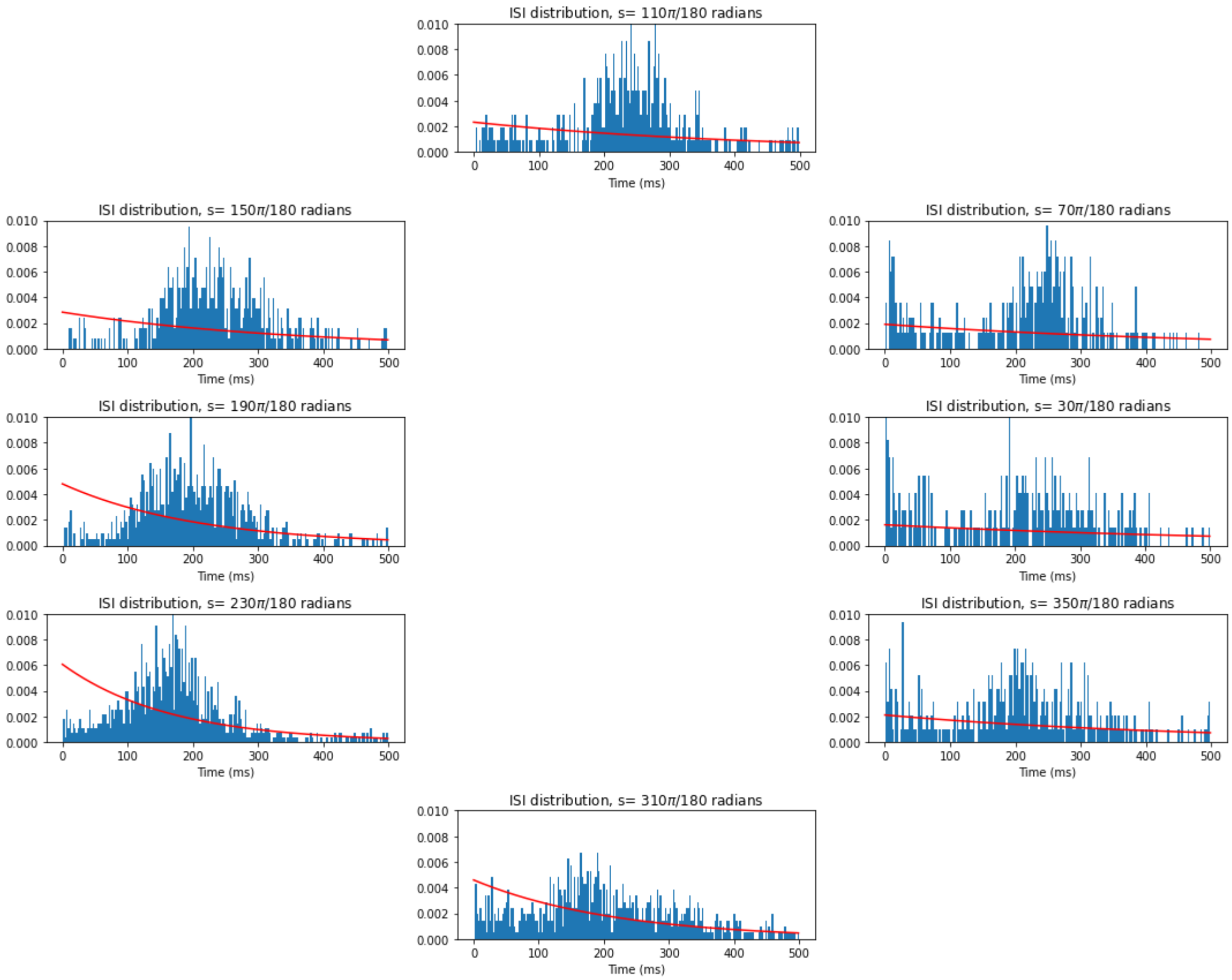
Your answer:

Somewhat but not completly. Becuase this is discretized, the variance seems to be somewhat higher than would be expected for a poisson distribution especially at the lower number of counts.

(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=(15,12))
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.
    #=====================================================#
    ISI = []
    for trial in range(num_trials):
        prev_time = 0
        for spike_time in spike_times[con, trial]:
          isi = spike_time - prev_time
          ISI.append(isi)
    plt.hist(ISI, bins =200, density = True)

    X = np.arange(0, 500, 1)
    Y = []
    lamda = mean_counts[con]/1000
    for x in X:
      Y.append(lamda*np.exp(-1*lamda*x))
    plt.plot(X,Y, c= 'r')
    plt.ylim([0, .01])
    plt.xlabel("Time (ms)")

    #=====================================================#
    # 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:

Becuase the refactory period of the neurons limits the smallest isi's from occuring which is where the idealized exponential distribution has the highest concentraition of isi's.