

ECE C143A/C243A, Spring 2023

Department of Electrical and Computer Engineering
University of California, Los Angeles

Homework #3

Prof. J.C. Kao
TAs T. Monsoor, R.R. Gore, D. Singla

Due Friday, 5 May 2023, uploaded to Gradescope.

Covers material up to Poisson Processes III.

100 points total.

1. (13 points) You are recording the activity of a neuron, which is spiking according to a Poisson process with rate λ . At some point during your experiment, the recording equipment breaks down and begins dropping spikes randomly with probability p .
 - (a) (10 points) Let the random variable M be the number of recorded spikes with the broken equipment. Show that the distribution of M is $\text{Poisson}((1 - p)\lambda s)$. (Hint: If N is a random variable denoting the number of actual spikes, what is $\mathbf{Pr}(M = m|N = n)$?)
 - (b) (1 points) What is the rate of the Poisson process in part (a)?
 - (c) (2 points) What is the distribution on the number of spikes dropped within a τ second interval?
2. (35 points) Homogeneous Poisson process
Complete the Jupyter notebook `hw3p2.ipynb`. Print your notebook code and output to a pdf as your submission for this question.
3. (22 points) Inhomogeneous Poisson process
Complete the Jupyter notebook `hw3p3.ipynb`. Print your notebook code and output to a pdf as your submission for this question.
4. (30 points) Real neural data
Complete the Jupyter notebook `hw3p4.ipynb`. Print your notebook code and output to a pdf as your submission for this question.

12) Neuron spiking according to Poisson process with rate λ .

→ recording equip. breaks, starts dropping spikes randomly with probability P

if N is a RV denoting the actual # of spikes, what is $\Pr(M=m | N=n)$

↑
RV # of recorded spikes

$$\Pr(N=n) = \frac{(\lambda s)^n e^{-\lambda s}}{n!}$$

$$\Pr(M=m | N=n) = \frac{n!}{(n-m)! m!} (1-P)^m P^{n-m}$$

$$\Pr(M=m | N=n) = \frac{\Pr(M=m, N=n)}{\Pr(N=n)}$$

$$\Pr(M=m | N=n) \Pr(N=n) = \underbrace{\Pr(M=m, N=n)}_{/}$$

$$P(M=m) = \text{sum across } P(M=m, N=n)$$

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

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

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

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

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

I need: $\frac{[(\lambda s)(1-p)]^m e^{-\lambda s}(1-p)}{m!}$

Sep-

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

↙
sum over poisson
distribution = 1

$$P(M=m) = \frac{\lambda^m [1-p]^m e^{-\lambda} (1-p)}{m!}$$

QED

b) $(1-p)\lambda$

c) $\sim \text{Poisson}(p\lambda\tau)$

2c) There should be 800 points on plot



go through each trial and

① count the # of spikes

associate the λ to the angle of the test

② $\frac{1}{\# \text{spikes}} = \lambda$

↓ because I second

2d) for each reach angle, plot normalised distribution



count in a class
divided by the total
of observations



1. need total # of spikes

Spike-counts \rightarrow each element in spike counts has total spike count for that angle and trial

totalspikes needs to be a

$$\begin{bmatrix} 0_1 \\ 0_2 \\ \vdots \\ 0_8 \end{bmatrix}$$

such that you can normalise based on that value

row, column

$8 \times 1 \rightarrow$

spikecounts [con, rep]

8×100

so I need to go column by column.

totalspikes [con] = add spike counts.

totalspikes [k] = $\sum_{n=0}^{99}$ spikecounts [k, n]

totalspikes [0] = $\sum_{n=0}^{99}$ spikecounts [0, n]

first row all columns.

$$\text{totalspikes}[0] = \text{np.sum}(\text{spikecounts}[0, :])$$

\uparrow
con

\uparrow
con

$$\therefore \text{totalspikes}[con] = \text{np.sum}(\text{spikecounts}[con, :])$$

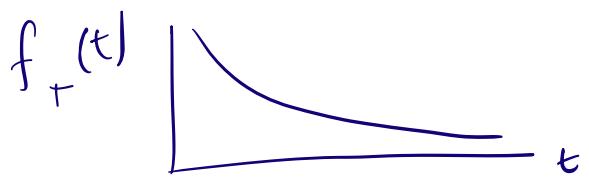
→ this did not work.

scipy.stats.poisson → "fitting" a POISSON distribution to empirical distribution → trying to find the poisson distribution that best describes the empirical distr.
↓
find the PARAMETERS
↓
in this case mean

2e) given there are 8 points, one for each angle,
I have to take mean and variance of spikes-count [con]

variance →

2f). for each angle, plot normalised dist. of ISIs
I have an 8 [] with spike times → each has its own trial with spike times



$$f_+(t) = \begin{cases} \lambda e^{-\lambda t} & \text{if } t \geq 0 \\ 0 & \text{else} \end{cases}$$

np.diff()

|S|

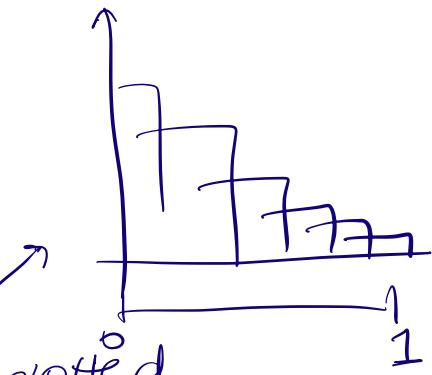
This worked with isi-test

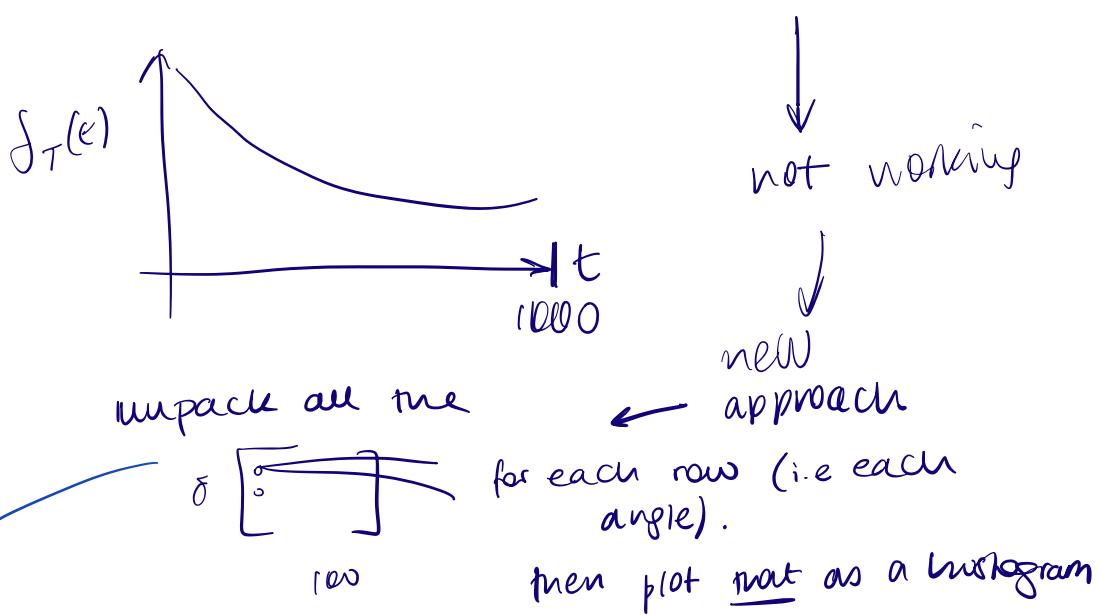
new so
should
be 1000
is it in
ms?

right
now
they
are all plotted

separately. I need to
take mean of each
bin

I because
each trial lasts
1 second so that's
the max value we
can take





spikesunpacked will be

$$\begin{bmatrix} & & \\ & \nearrow & \\ 8 & & \\ & \swarrow & \\ & & \end{bmatrix}$$
here this will be a list of all the spike times for θ_8

← isis_unpacked₈ []

for con in range (num_cons):

isis_unpacked = np.diff (spikesunpacked [con])

HOW TO DO THIS:

new vectors = []

for con in range (num_cons):

 for rep in (num trials):

 new vector [con] += spiketimes [con, rep]

↑ does not work

/

use .ext

2g) spikes unpacked $\left[\begin{array}{c} \circ \\ \vdots \\ \circ \end{array} \right]$ list of all the spike times for that reach angle

isis_array $\left[\begin{array}{c} \circ \\ \vdots \\ \circ \end{array} \right]$ list of all ISI's for that reach angle.

mean_ISI = np.zeros(num_cols)

std_ISI = np.zeros(num_cols)

coeff_v = np.zeros(num_cols)

for con in range(num_cols)

mean_ISI[con] = np.mean(isis_array[con])

std_ISI[con] = np.std(isis_array[con])

coeff_v[con] = std_ISI[con] / mean_ISI[con]

plt.plot(mean_ISI, coeff_v)

Homework 3, Problem 2 on homogeneous Poisson processes

ECE C143A/C243A, Spring Quarter 2023, Prof. J.C. Kao, TAs T. Monsoor, R. Gore, D. Singla

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 referred 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 []:

```
"""
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
import scipy.stats

# Load matplotlib images inline
%matplotlib inline

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

(a) (6 points) Spike trains

For each of the following reaching condition ($s = k \cdot \pi/4$, where $k = 0, 1, \dots, 7$), 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 [ ]: ## 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 conditions
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)

def findfiringrate(angle: int) -> float:
    return r_0 + (r_max-r_0)*np.cos(angle-s_max) # tuning curve

for con in range(num_cons):
    angle = s[con]
    for rep in range(num_trials):
        firingrate = findfiringrate(angle)
        spike_times[con, rep] = nsp.GeneratePoissonSpikeTrain(T, firingrate)
```

```
In [ ]: # ### MY FUNCTION IMPLEMENTATION ## IGNORE

## 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 conditions
# 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)

# def findfiringrate(angle: int) -> float:
#     return r_0 + (r_max-r_0)*np.cos(angle-s_max) # tuning curve

# for con in range(num_cons):
#     firingrate = findfiringrate(s[con])
#     t = 0
#     tau = 1

#     for rep in range(num_trials):
#         #=====
#         list_spiketimes = []

#         while (t < tau):
#             sample = np.random.exponential(scale=(1.0/firingrate))
#             t += sample
#             #print("sample is", sample)
#             list_spiketimes.append(sample)

#         #=====
#         spike_times[con, rep] = list
#         pass
# =====#
# # END YOUR CODE
# =====#
```

In []:

```
print(spike_times.shape)

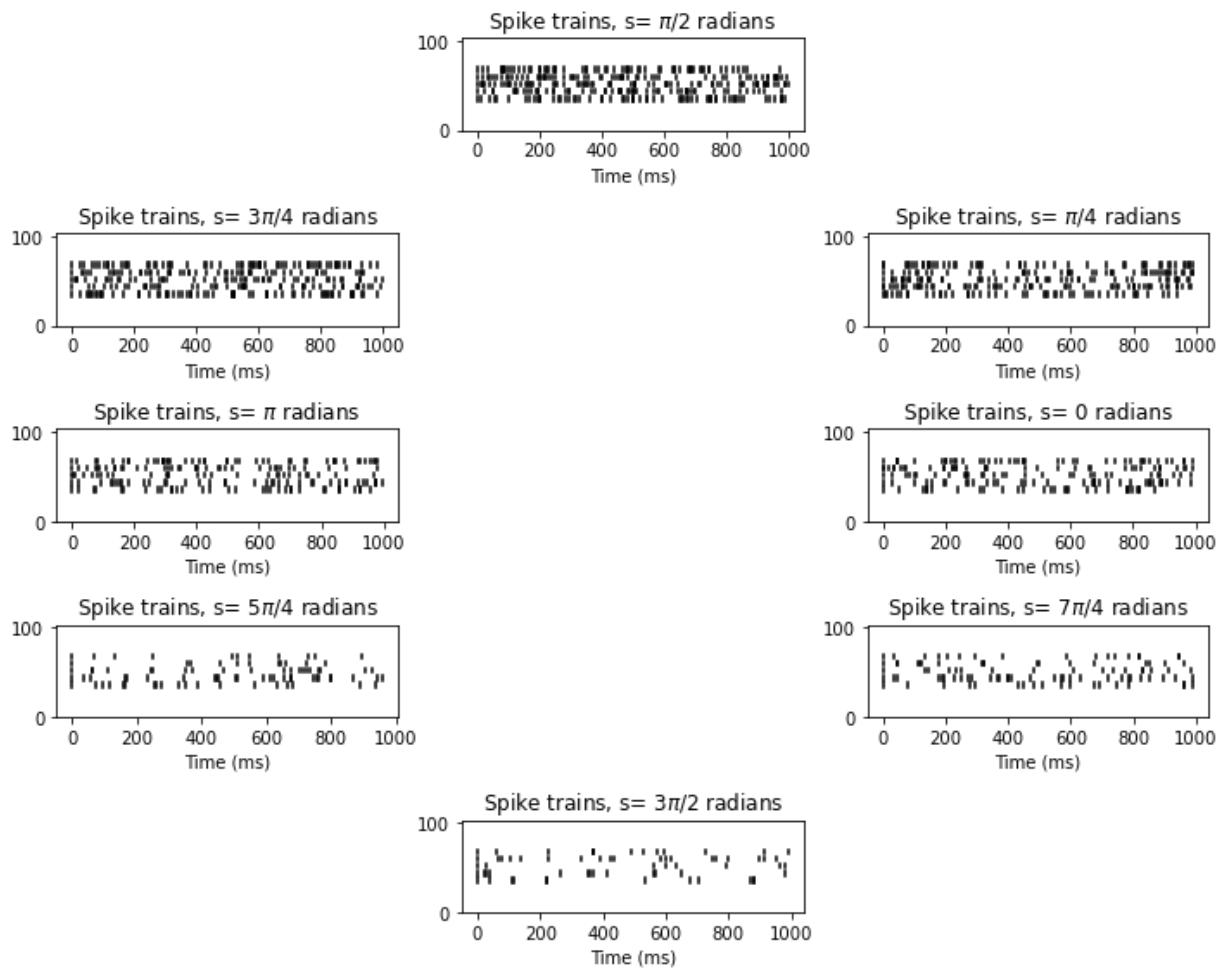
s_labels = ['0', '$\pi/4', '$\pi/2', '3$\pi/4', '$\pi', '5$\pi/4', '3$\pi/2']
num_plot_rows = 5
num_plot_cols = 3
subplot_idx = [9, 6, 2, 4, 7, 10, 14, 12]
num_rasters_to_plot = 5 # per condition

# 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_idx[con])
    nsp.PlotSpikeRaster(spike_times[con, 0:num_rasters_to_plot])

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

(8, 100)



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 [ ]: ## 2b
plt.figure(figsize=(10,8))
print(spike_times[0,1])
bin_width = 20 #ms
```

```

num_bins = int(T / bin_width)

# initialise the arrays to hold the binned spike counts and rates
binned_spike_counts = np.zeros((num_cons, num_trials, num_bins), dtype = int)
binned_spike_rates = np.zeros((num_cons, num_bins))

# compute the bin edges, with small epsilon value to avoid 0
eps = 0.000001
bin_edges = np.arange(num_bins+1) * bin_width + eps

# loop over the angles and trials

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

    for trial in range(num_trials):

        # get spike times for current angle and trial
        spike_train = spike_times[con, trial]

        # bin the spike times
        binned_counts, _ = np.histogram(spike_train, bin_edges)

        # store the binned spike counts
        binned_spike_counts[con, trial] = binned_counts

# compute the average binned spike counts across trials

        binned_spike_rates[con] = np.mean(binned_spike_counts[con], axis = 0)

        bin_time_centers = bin_edges[:-1] + bin_width / 2

        # plot the binned spike counts across trials

        plt.bar(bin_time_centers, binned_spike_rates[con], width=bin_width)

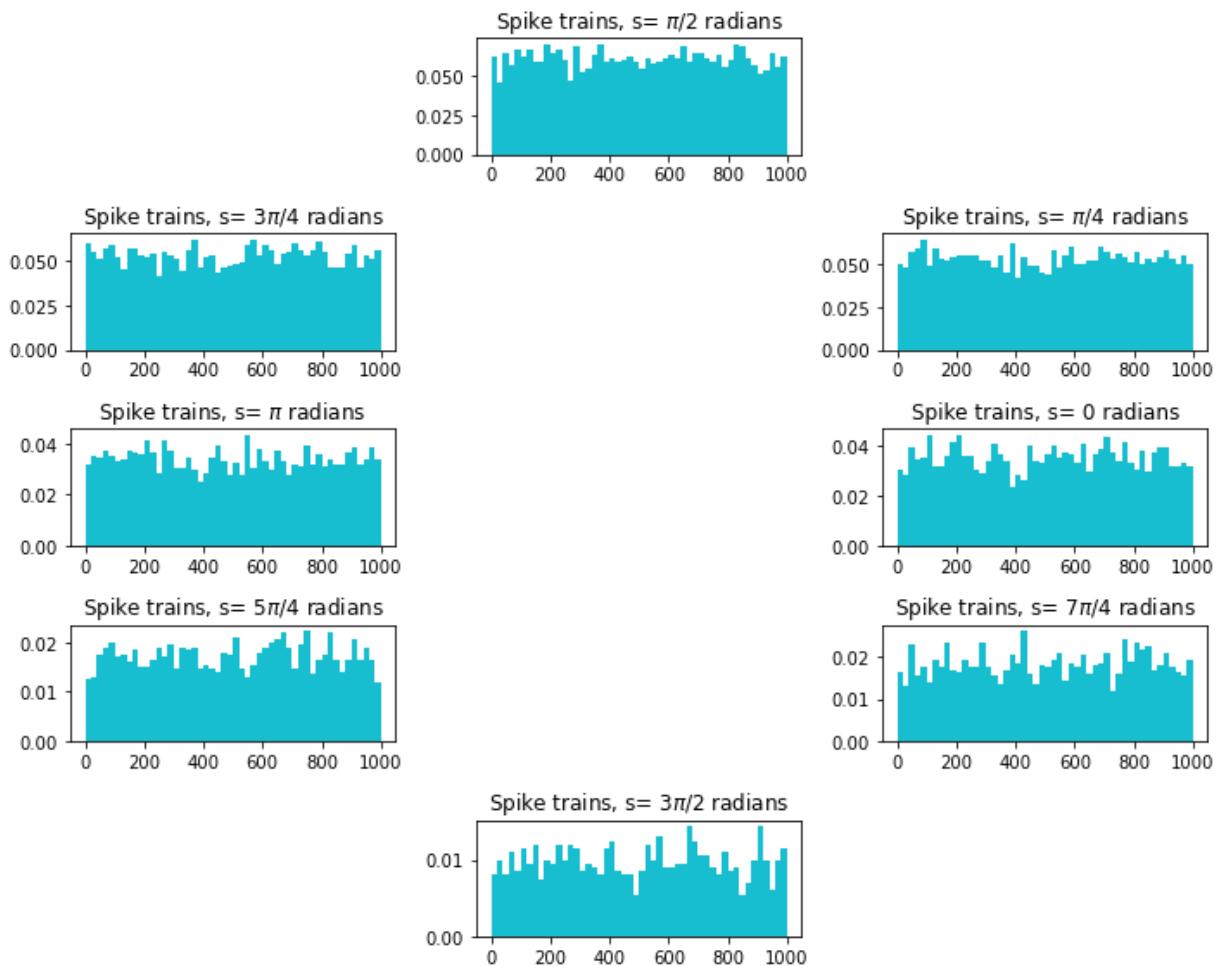
=====
pass

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

plt.show()

```

```
[ 0.          29.37663443  99.43395052 139.36342453 156.54212912
 197.27728062 202.19766896 268.44125135 274.87619869 291.76224511
 334.26861788 352.912235  366.60167652 442.54985728 455.67918855
 481.67457743 528.46632611 643.63825215 670.58999163 675.55315465
 675.99598196 676.34749813 690.53090848 718.9115343  760.14514394
 791.99616938 843.47495505 850.20835556 853.34409685 862.59885437
 885.92781867 891.03989836 892.92497101 906.83628892 947.67472374
 987.43406424]
```



(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 [ ]: ## 2c
spike_counts = np.zeros((num_cons, num_trials)) # each element in spike_count
#=====#
for con in range(num_cons):
    for rep in range(num_trials):
        spike_counts[con, rep] = len(spike_times[con, rep])

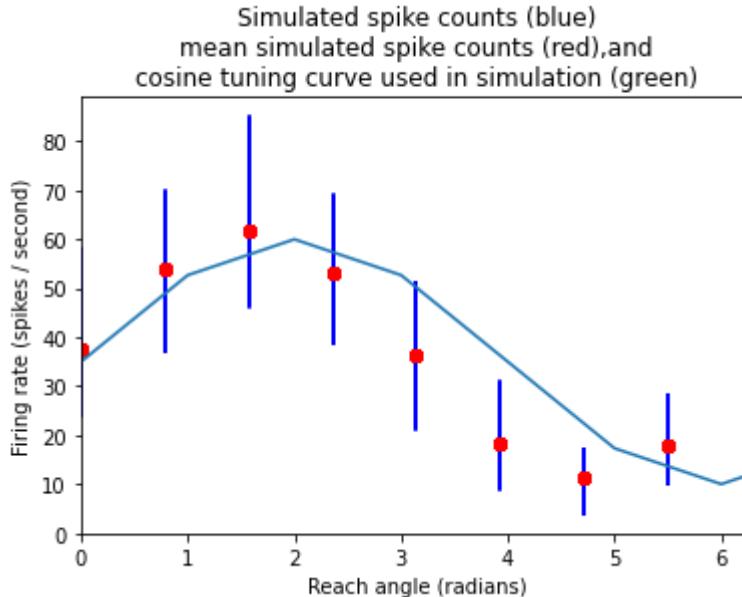
for n in range(len(s)):
    plt.plot([s[n]]*len(spike_counts[n]), spike_counts[n], "b")

for n in range(len(s)):
    mean_frs = np.mean(spike_counts, axis=1)
    plt.plot(s, mean_frs, "r", marker = "o", linestyle = "none")

plt.plot(tuning)
```

```
#=====
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[]: (0.0, 6.283185307179586)



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

Your answer: I think there is a slight phase shift that I have to look into. But once this phase shift is sorted, yes it does

(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 [ ]: ##2d

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.

    spike_count_means = np.mean(spike_counts, axis=1)
```

```
mean_sc = spike_count_means[con]

fit_poisson = scipy.stats.poisson(mean_sc)
poisson_pmf = fit_poisson.pmf(spike_count_bin_centers)

# plt.bar(spike_count_bin_centers, binned_spike_rates[con], width=bin_width)
plt.hist(spike_counts[con], spike_count_bin_centers, density = True)

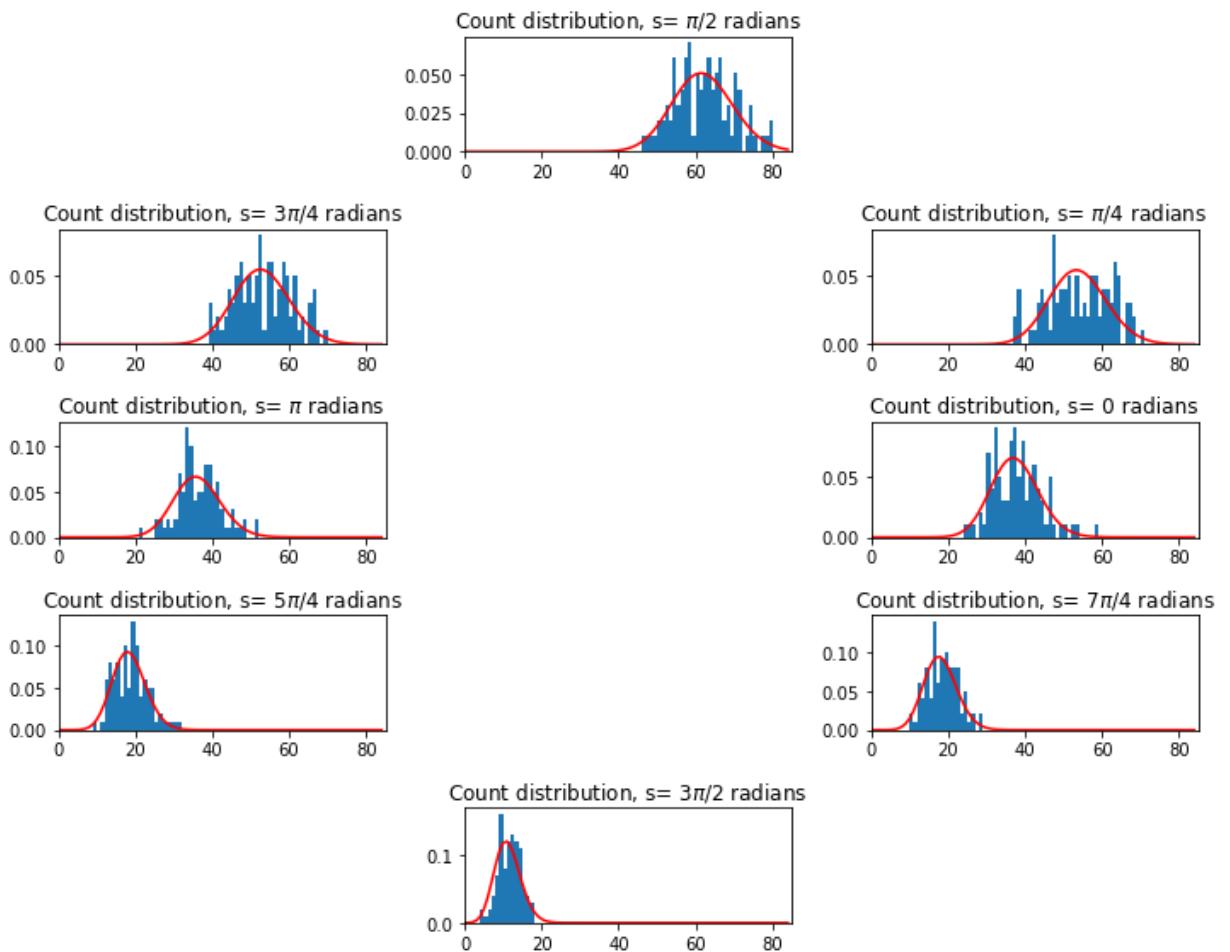
# plt.bar(spike_count_bin_centers, )
plt.plot(poisson_pmf, "r")

#####
# pass

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

#####
# YOUR CODE HERE:
# Plot the empirical count distribution, and on top of it
# plot your fit Poisson distribution.
#####
# 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

(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 [ ]: ## 2e

spike_count_means = np.mean(spike_counts, axis=1)
print(spike_count_means)

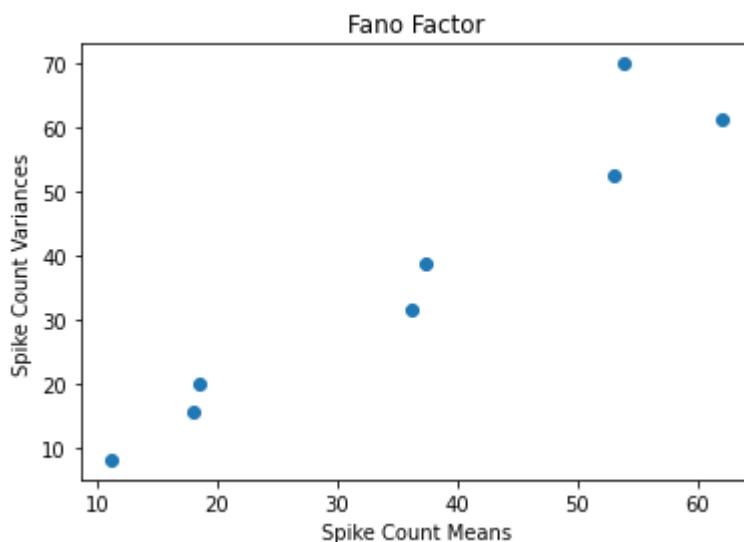
spike_count_vars = np.var(spike_counts, axis = 1)

plt.plot(spike_count_means, spike_count_vars, Marker = "o", linestyle = "none"
plt.xlabel("Spike Count Means")
plt.ylabel("Spike Count Variances")
plt.title("Fano Factor")
plt.show()

[37.42 53.86 61.92 52.98 36.14 18.44 11.21 18.08]
```

```
<ipython-input-9-9a8e4e20b62b>:8: MatplotlibDeprecationWarning: Case-insensitive properties were deprecated in 3.3 and support will be removed two minor releases later
```

```
    plt.plot(spike_count_means, spike_count_vars, Marker = "o", linestyle = "none" )
```



Question:

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

Your answer: Yes

(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 [ ]: ## 2f FINAL ANSWER

## initial test for understanding

plt.figure(figsize=(10,8))
spikesunpacked = [[] for i in range(num_cons)]
isis_array = np.zeros(num_cons, dtype = list)
# let max ISI = 600 based on analysis of data

isis_max = 600
bin_centres_2f = np.arange(0, isis_max, 10)

for con in range(num_cons) :
    for rep in range(num_trials) :

        plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])

        spikesunpacked[con].extend(spike_times[con, rep])
        isis_array[con] = np.diff(spikesunpacked[con])
```

```

plt.hist(isis_array[con], bins = bin_centres_2f, density = True)
mean_isi = np.mean(isis_array[con])
fit_exp = scipy.stats.expon(mean_isi)
expon_pdf = fit_exp.pdf(bin_centres_2f)

# I recognise I have not scaled the exponential correctly - to do: correct
plt.plot(bin_centres_2f ,expon_pdf)

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

```

<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

```

plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

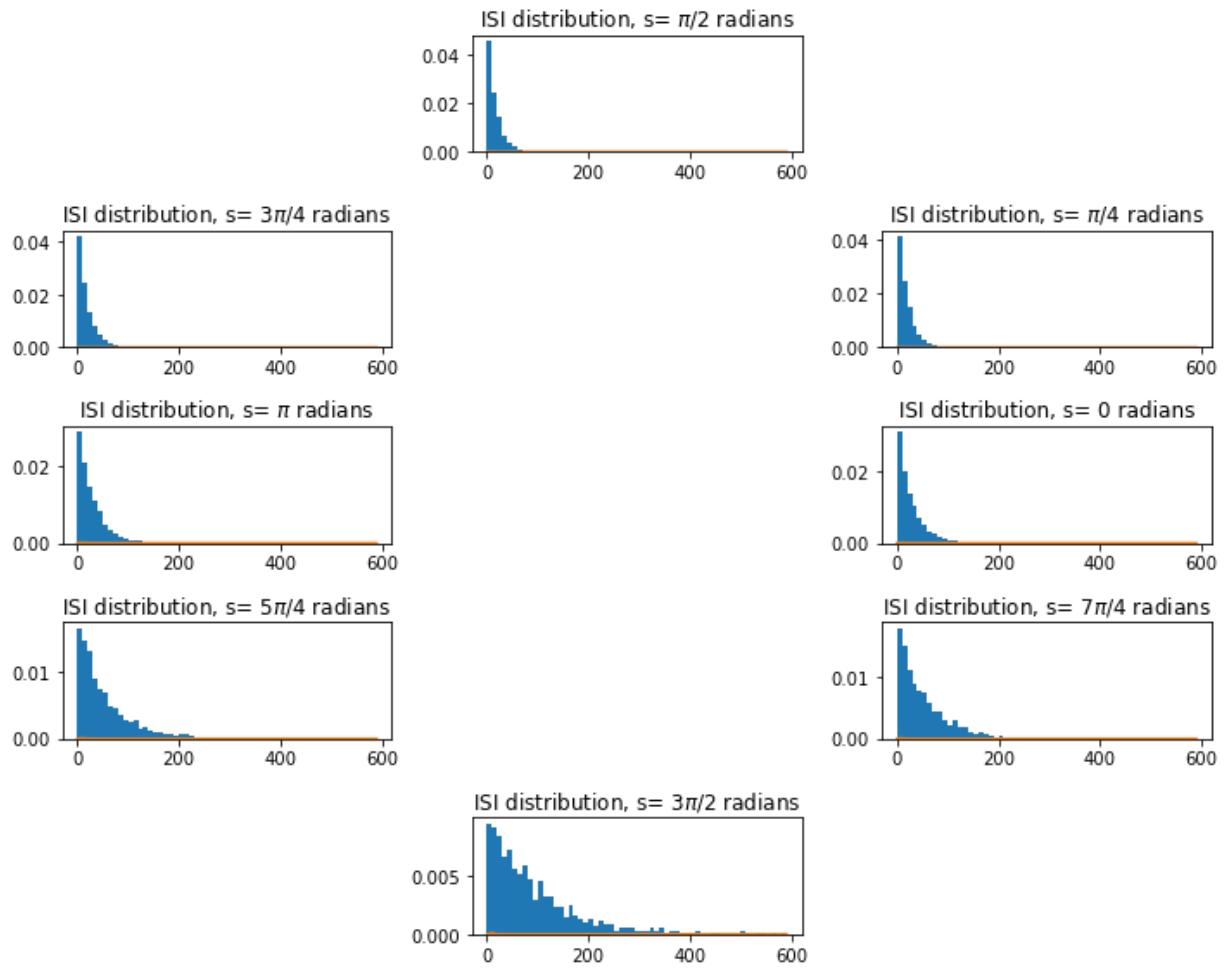
plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])
<ipython-input-16-b9a2ae7900f0>:19: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

plt.subplot(num_plot_rows,num_plot_cols,subplot_indx[con])

```



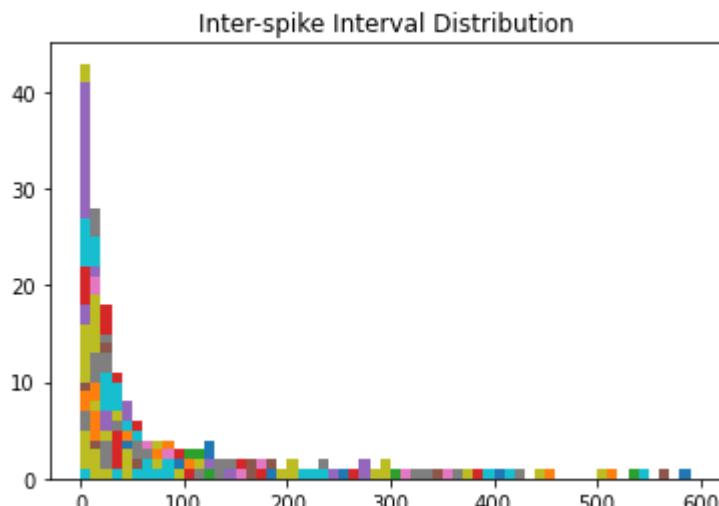
In []:

`## 2f`

```
## INITIAL TEST FOR UNDERSTANDING

for con in range(num_cons) :
    for rep in range(num_trials):

        isi_test = np.diff(spike_times[con,rep])
        bins2f = np.arange(0, 600, 10)
        # print(isi_test)
        plt.hist(isi_test, bins2f)
        plt.title('Inter-spike Interval Distribution')
```



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 []:

```
#2g

# coefficient of variation = standard deviation / mean

mean_isi = np.zeros(num_cons)
std_isi = np.zeros(num_cons)
coef_v = np.zeros(num_cons)
mean_2_isi = np.zeros(num_cons)
std_2_isi = np.zeros(num_cons)

print(isis_array.shape)
print(isis_array[0])
print("-----")
print(np.mean(isis_array[0]))


for con in range(num_cons):
    mean_isi[con] = np.mean(isis_array[con])
    std_isi[con] = np.std(isis_array[con])
    coef_v[con] = std_isi[con] / (mean_isi[con])

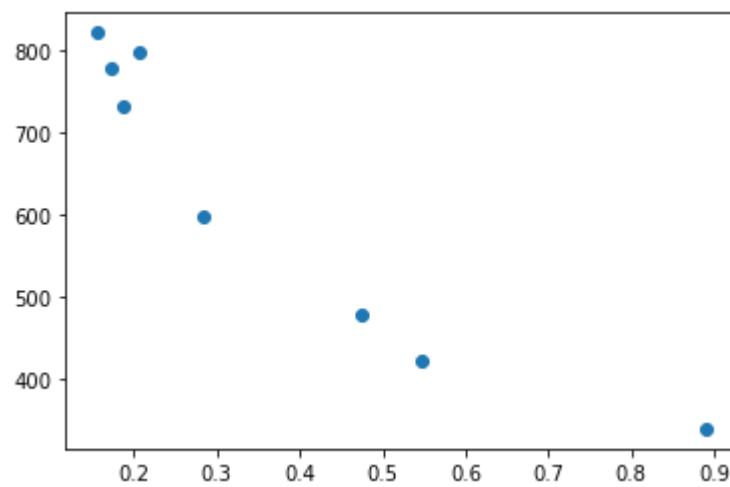
print(mean_isi)
print(std_isi)

plt.plot(mean_isi, coef_v, marker = 'o', linestyle = 'none')
#plt.plot(coef_v, mean_isi, marker = 'x', linestyle = 'none')

print("testesttest:", mean_isi)

## this unity should be reversed -- If I had time what I would do is: create
# each list would be a list of ISIs
# I would take the mean and standard deviation of each list, creating a new a
# then I would take the mean and standard deviation across each condition, an
```

```
(8,
[52.82817918 78.89446167 6.18168684 ... 11.51698786 30.78871957
 9.73491553]
-----
0.20633657176808395
[0.20633657 0.17327531 0.15438427 0.18618816 0.28345941 0.54789421
 0.88992617 0.47339882]
[164.75331609 134.75643613 126.84782776 136.11361454 168.97998842
 231.05507818 301.02627751 225.70712557]
testesttest: [0.20633657 0.17327531 0.15438427 0.18618816 0.28345941 0.5478942
 1
 0.88992617 0.47339882]
```



Question:

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

Your answer:

Not right now, but I know what I need to do to make it look like unity.

Homework 3, Problem 3 on inhomogeneous Poisson processes

ECE C143A/C243A, Spring Quarter 2023, Prof. J.C. Kao, TAs T. Monsoor, R. Gore, D. Singla

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 referred as $s(t)$ equation in the questions.

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

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

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

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

```
Out[ ]: 1.3161906679489954
```

```
In [ ]: ## 3a
num_trials = 100 # number of total spike trains
```

```
num_rasters_to_plot = 5 # number of spike trains to plot

spike_trains = []

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

#=====#
# YOUR CODE HERE:
# Generate the spike times for 100 trials of an inhomogeneous
# Poisson process. Plot 5 example spike rasters.

def tuning_curve(s):
    return r_0 + (r_max - r_0) * np.cos(s - s_max)

def reach_angle(t):
    return t**2 * np.pi

for i in range(num_trials):
    spikes = []
    t = 0

    while t < T:
        fr = tuning_curve(reach_angle(t))
        t += np.random.exponential(1.0/r_max * 1000)

        if t < T:
            spikes.append(t)

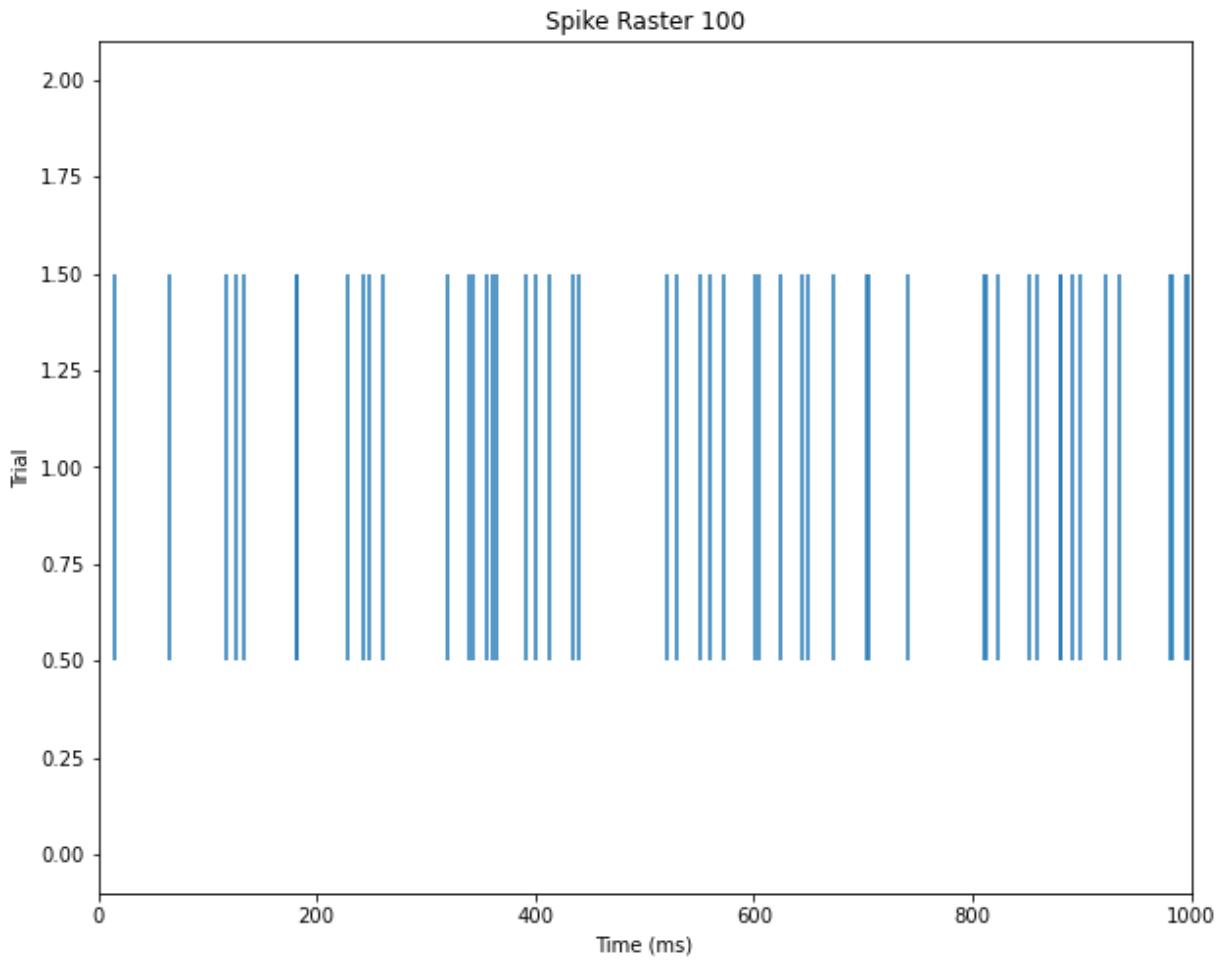
    spike_trains.append(spikes)

plt.eventplot(spike_trains[i])
plt.xlim([0, T])
plt.xlabel('Time (ms)')
plt.ylabel('Trial')
plt.title(f'Spike Raster {i+1}')

plt.show()

#=====
# pass

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



(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 []:

```
# 3b
bin_width = 20 # (ms)
#=====
# YOUR CODE HERE:
#   Plot the spike histogram
#=====
pass

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

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

Question:

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

Your Answer:

(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 [ ]:
=====
# YOUR CODE HERE:
#   Plot the normalized distribution of spike counts
=====
pass

=====
# 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:

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

=====
# 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: