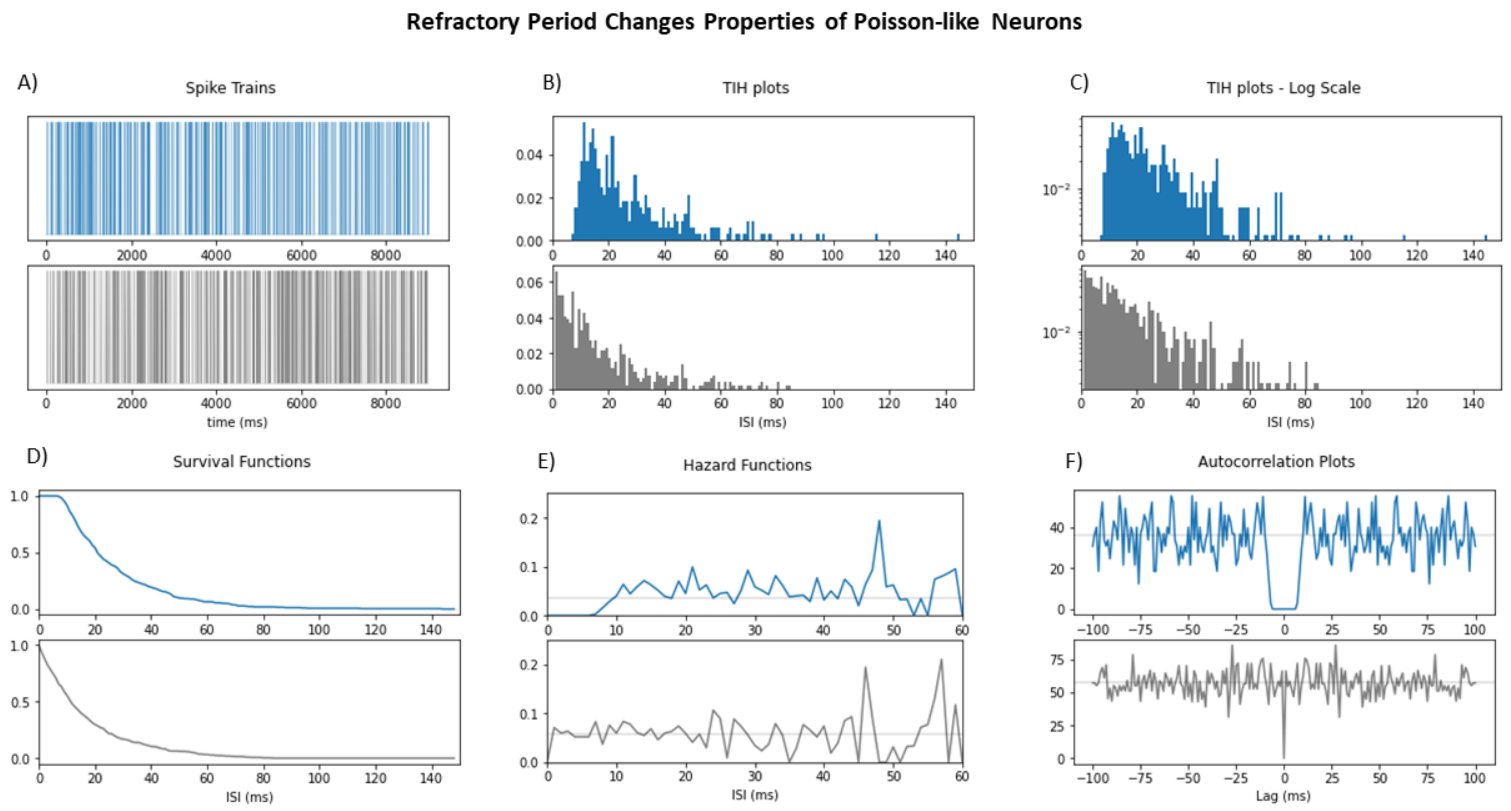
Assignment 3

1) Properties of a single Poisson process with refractory period

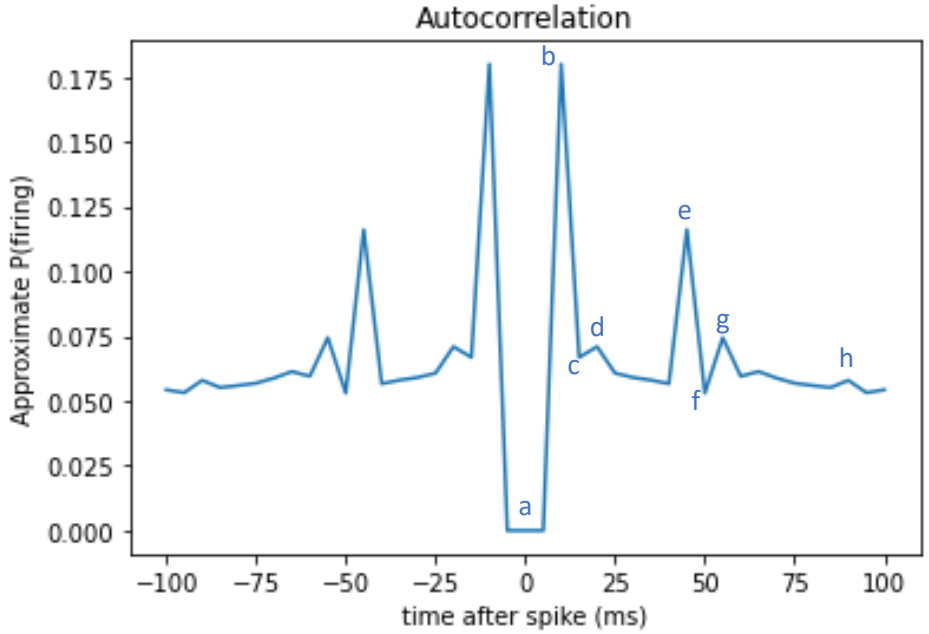


Properties of a Poisson-like neuron with refractory periods (blue) and without refractory periods (gray).

I simulated 9 s long spike trains from 2 Poisson-like neurons with baseline firing rate of 55 hz, one blue neuron (shown in blue) with an absolute refractory period of 5 ms and relative refractory period of 6 ms, and one gray neuron (shown in gray) with no refractory period. **Fig A** shows the 9 s long spike trains. **Fig B** shows the TIH plots, where we can clearly see that the refractory periods of the blue neuron. The y axes denote probability of a spike. **Fig C** shows the TIH plots in log scale, where we can clearly see a downwards linear trend in the gray neuron, proving its ISIs form an exponential distribution, which was to be expected from its Poisson-like property. Additionally, its intercept is near 0.055, which is the probability of firing at each ms. The blue neuron differs somewhat from an exponential distribution, due to its refractory periods. **Fig D** shows the survival functions, where we can clearly see the blue neuron has survival probability of 1 for 5 ms due to the refractory period, and then a gradual drop, while the gray neuron drops immediately, as expected. **Fig E** shows the hazard functions up to 60 ms after spike (they get very noisy past that). We can clearly see the influence of the refractory periods in the probability of firing for the blue neuron. Additionally, the function tends to the mean firing rate (shown in black horizontal line for each neuron), which is to be expected. **Fig F** shows the autocorrelation plots with ±100 ms lag. The y axes denote firing rate. The effect of the refractory periods of the blue neuron is evident around 0. Additionally, the rates tend to the mean firing rate (shown in black horizontal line for each neuron), as to be expected.

2) Extracting the autocorrelation from the hazard function

A. Wrote a function in python to approximate the autocorrelation from the hazard function. The function solves for the survival function and then computes for each time lag (n) the probability of every interval combination happening up to n (up to 3 intervals) by multiplying the survival and hazard functions. The longer the lag the more the function underestimates the true probability. The rough output is as follows:

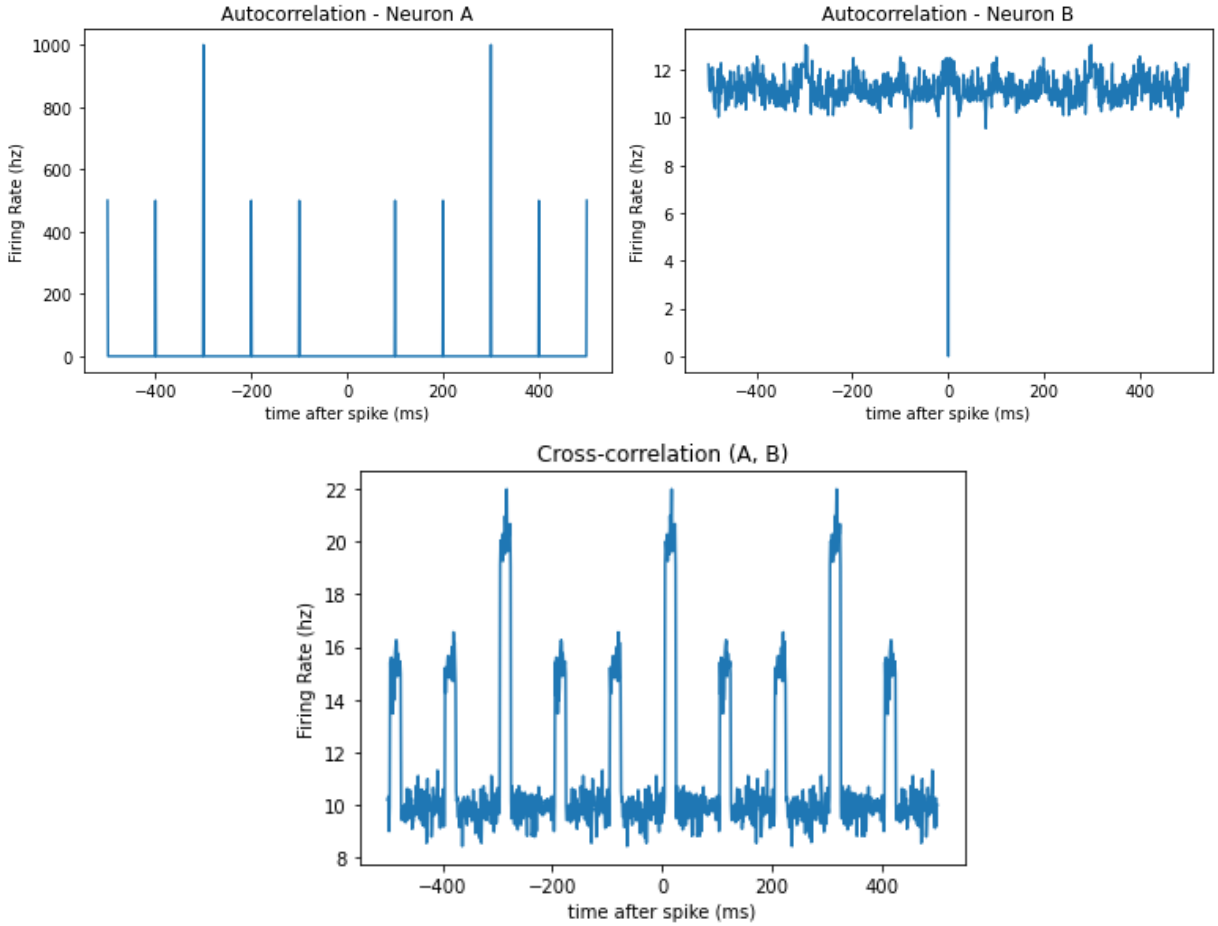


The main interpretation I have is that we see a clear refractory period in the center (a), then a high probability of firing right after(b), then a drop to baseline which is also due to the refractory period following the a spike in b (c), then a slightly increase which is due to the higher probability of a spike 10 ms after b, then a drop to baseline until another increase in 40 ms due to the hazard function (e), which again incurs a drop for the refractory period (f) and an increase 10 ms after (g) (which presumably keep incurring smaller refractory period decreases and 10 ms increases), then a drop to baseline until the last slight increase in 80 ms due to the periodicity of e (h), which also presumably keeps incurring refractory period decreases and 10 ms increases.

B. The neuron has a refractory period, and is partially bursty (since it has high probability of firing 10 ms after spike and 40ish ms after spike), and partially Poisson-like since it has equal chance of firing for most other times.

3) Cross-correlation

Wrote a function to randomly simulate such neuron pairs in python (for much longer than the given experiment conditions since what we care about is what happens at the limit, ignoring noise from under-sampling). The rough output is as follows:

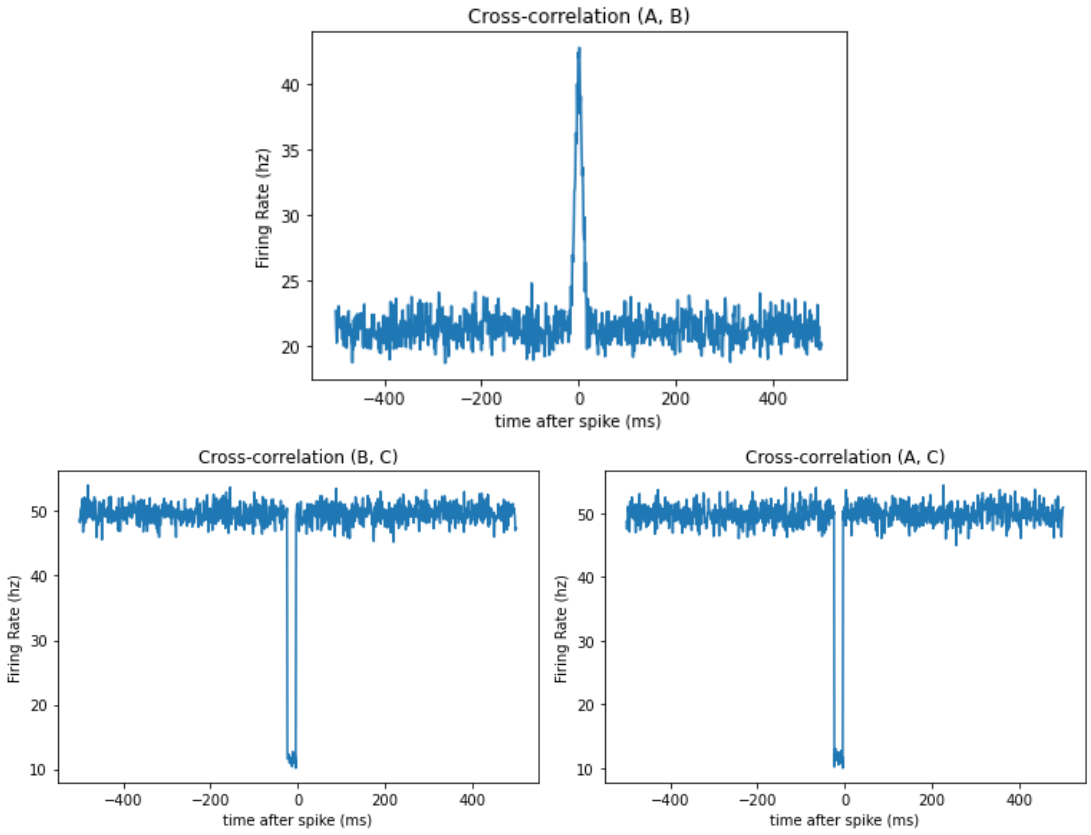


It is pretty clear to see the autocorrelation of neuron A to be so systematically periodic, since the intervals between spikes are fixed. The autocorrelation of neuron B should generally be flat as it is Poisson-like (at firing rate of 10 hz), but since every 100 ms or 200 ms lags the rate increases for a short while, we do see a small increase in firing rate dispersed around these lags (and their multiples).

The cross correlation most clearly shows the relationship between neurons A and B, as we clearly see neuron B is usually at around 10 hz firing rate, except for right after spikes in neuron A, which occur always at 0 and 300 ms, and half the time at 100 ms, 200 ms, 400 ms, and 500 ms, and thus neuron B increases its firing rate accordingly (4 ms after spike for 20 ms). Since A is periodic, we can also see the negative lags to be the same, except they show the interval when B fired before A.

4) Common input cross-correlations analysis

Wrote a function to randomly simulate such neuron dynamics in python. I decided that the neurons will all be Poisson-like with firing rate of 50 hz, and each spike from C decreases the firing rates of A and B to 5 hz. The rough output is as follows:



The cross-correlations of (A,C) is identical and easy to explain – conditional on A firing, it is much less likely that C fired in the period of 24ms-4ms previously (since, had C fired then, A is much less likely to fire). The cross-correlation of (B,C) is identical for the same reason.
The cross-correlation of (A,B) also shows a certain effect – conditional on A firing, it is less likely that C fired shortly previously, which means that it is more likely that B fired at a similar time. Therefore, A and B are highly correlated around 0, at their 50hz firing rate, but the inhibitory effect of C lowers the correlation in different lags. Essentially, we found what we expected to find – a high correlation caused by a confounding variable.

It is probably important to note that had C been a neuron with any periodic behavior, we would have seen periodic decreases in the corresponding lags in the cross-correlations (A,C) and (B,C), and periodic increases in the corresponding lags in the cross-correlation (A,B), for the reasons explained earlier (except duplicated in time due to periodicity).

Appendix – Code 1A

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)

def refPeriod(): # function used after spike to see what happens in next 11 ms
    k = np.zeros(5)
    for i in np.arange(6):
        if np.random.binomial(1, 0.055*i/6) == 1:
            k = np.append(k,1)
            return k
        k = np.append(k,0)
    return k

spk_trn = np.zeros(1) # w/ ref period
while len(spk_trn) < 9000: # go through all ms
    if spk_trn[-1] == 1: # if spike occured during relative ref period
        spk_trn = np.append(spk_trn, refPeriod()) # go into next ref period
    elif np.random.binomial(1, 0.055) == 1: # else, see if spike occured now
        spk_trn = np.append(spk_trn, 1)
        spk_trn = np.append(spk_trn,refPeriod())
    else: # or didn't occur now
        spk_trn = np.append(spk_trn, 0)
spk_trn = spk_trn[1:9001]

spk_trn2 = np.zeros(9000) # w/o ref period
for i in range(9000):
    if np.random.binomial(1, 0.055) == 1:
        spk_trn2[i] = 1

ISIs = np.diff(np.flatnonzero(spk_trn))
ISIs2 = np.diff(np.flatnonzero(spk_trn2))

plt.figure()
plt.suptitle('Spike Trains')
plt.subplot(2,1,1)
plt.plot(range(9000),spk_trn, lw = 0.2)
ax = plt.gca()
ax.axes.yaxis.set_ticks([])
plt.subplot(2,1,2)
plt.plot(range(9000), spk_trn2, lw = 0.2, color = 'gray')
plt.xlabel('time (ms)')
ax = plt.gca()
ax.axes.yaxis.set_ticks([])
plt.show()

plt.figure()
plt.suptitle('TIH plots')
plt.subplot(2,1,1)
plt.hist(ISIs, bins = np.arange(0,150), density = True)
plt.xlim((0,150))
plt.subplot(2,1,2)
plt.hist(ISIs2, bins = np.arange(0,150), density = True, color='gray')
plt.xlim((0,150))
plt.xlabel('ISI (ms)')
plt.show()

plt.figure()
plt.suptitle('TIH plots - Log Scale')
plt.subplot(2,1,1)
plt.hist(ISIs, bins = np.arange(0,150), density = True, log = True)
plt.xlim((0,150))
plt.subplot(2,1,2)
plt.hist(ISIs2, bins = np.arange(0,150), density = True, log = True, color='gray')
```

```
plt.xlim((0,150))
plt.xlabel('ISI (ms)')
plt.show()

z = plt.hist(ISIs, bins = np.arange(0,150), density = True)[0] # make pdf
z2 = plt.hist(ISIs2, bins = np.arange(0,150), density = True)[0] # make pdf
s = 1 - np.cumsum(z) # make survivor function
s2 = 1 - np.cumsum(z2) # make survivor function

plt.figure()
plt.suptitle('Survival Functions')
plt.subplot(2,1,1)
plt.plot(s)
plt.xlim((0,150))
plt.subplot(2,1,2)
plt.plot(s2, color = 'gray')
plt.xlim((0,150))
plt.xlabel('ISI (ms)')
plt.show()

h = z/s # make hazard function
h[max(ISIs)] = 0
h2 = z2/s2 # make hazard function
h2[max(ISIs2)] = 0

plt.figure()
plt.suptitle('Hazard Functions')
plt.subplot(2,1,1)
plt.plot(h)
plt.xlim((0,60))
plt.ylim((0,0.25))
plt.axhline(np.mean(spk_trn), color = 'black', lw = 0.2)
plt.subplot(2,1,2)
plt.plot(h2, color = 'gray')
plt.xlim((0,60))
plt.ylim((0,0.25))
plt.axhline(np.mean(spk_trn2), color = 'black', lw = 0.2)
plt.xlabel('ISI (ms)')
plt.show()

acs = np.zeros(201) # make auto-correlations
for i in range(1,101):
    acs[100+i] = np.sum(np.logical_and(spk_trn[:-i], spk_trn[i:]))/np.sum(spk_trn[:-i])
acs[100] = 0
acs[:100] = np.flip(acs[101:])
acs = acs/0.001
acs2 = np.zeros(201)
for i in range(1,101):
    acs2[100+i] = np.sum(np.logical_and(spk_trn2[:-i], spk_trn2[i:]))/np.sum(spk_trn2[:-i])
acs2[100] = 0
acs2[:100] = np.flip(acs2[101:])
acs2 = acs2/0.001

plt.figure()
plt.suptitle('Autocorrelation Plots')
plt.subplot(2,1,1)
plt.plot(np.arange(start=-100, stop=101), acs)
plt.axhline(np.mean(spk_trn)*1000, color = 'black', lw = 0.2)
plt.subplot(2,1,2)
plt.plot(np.arange(start=-100, stop=101), acs2, color = 'gray')
plt.axhline(np.mean(spk_trn2)*1000, color = 'black', lw = 0.2)
plt.xlabel('Lag (ms)')
plt.show()
```

Appendix – Code 4B

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)

n = 1000000
p = 50/1000
t = np.arange(-500,500+1)

# Poisson-like, 50 Hz, C reduces A and B to 5 hz during 4ms-24ms after spike
C = np.zeros(n) # spike train C
for i in range(n):
    C[i] = np.random.binomial(1,p)

A = np.zeros(n) # spike train A
for i in range(24, n):
    p = 5/1000 if np.any(C[i-24:i-4]) else 50/1000
    A[i] = np.random.binomial(1,p)

B = np.zeros(n) # spike train B
for i in range(24, n):
    p = 5/1000 if np.any(C[i-24:i-4]) else 50/1000
    B[i] = np.random.binomial(1,p)

AB_cc = np.zeros(1000+1) # CC(A,B)
for i in range(-500,500):
    AB_cc[500+i] = np.sum(np.logical_and(A[500:-500], B[i+500:-500+i]))/np.sum(A[500:-500])
AB_cc = AB_cc*1000
AB_cc[-1] =  AB_cc[-2]

plt.figure()
plt.title('Cross-correlation (A, B)')
plt.plot(t, AB_cc)
plt.xlabel('time after spike (ms)')
plt.ylabel('Firing Rate (hz)')
plt.show()

AC_cc = np.zeros(1000+1) # CC(A,C)
for i in range(-500,500):
    AC_cc[500+i] = np.sum(np.logical_and(A[500:-500], C[i+500:-500+i]))/np.sum(A[500:-500])
AC_cc = AC_cc*1000
AC_cc[-1] =  AC_cc[-2]

plt.figure()
plt.title('Cross-correlation (A, C)')
plt.plot(t, AC_cc)
plt.xlabel('time after spike (ms)')
plt.ylabel('Firing Rate (hz)')
plt.show()

BC_cc = np.zeros(1000+1) # CC(B,C)
for i in range(-500,500):
    BC_cc[500+i] = np.sum(np.logical_and(B[500:-500], C[i+500:-500+i]))/np.sum(A[500:-500])
BC_cc = BC_cc*1000
BC_cc[-1] =  BC_cc[-2]

plt.figure()
plt.title('Cross-correlation (B, C)')
plt.plot(t, BC_cc)
plt.xlabel('time after spike (ms)')
plt.ylabel('Firing Rate (hz)')
plt.show()
```