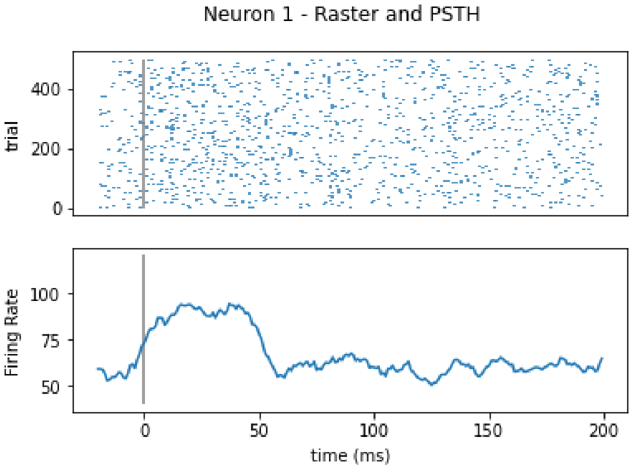
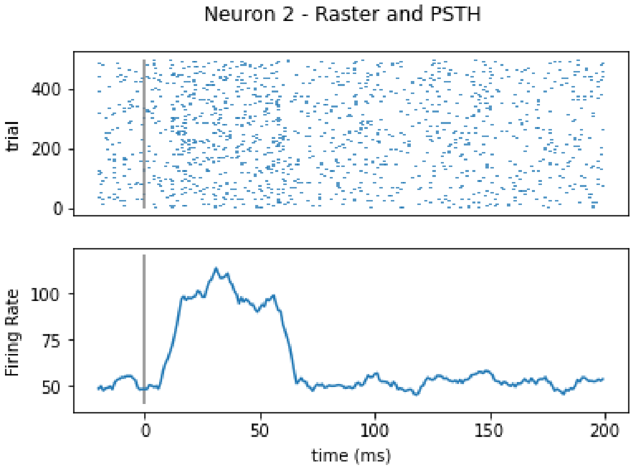
Assignment 4

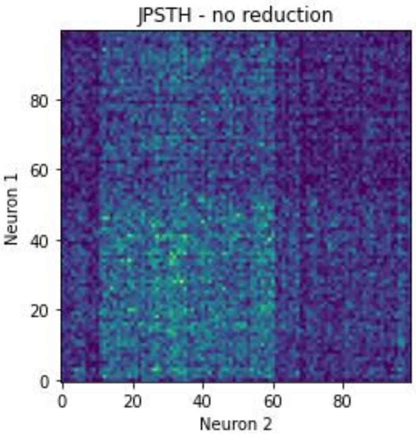
1) Encoding: from stimulus to neuronal activity (PSTH)

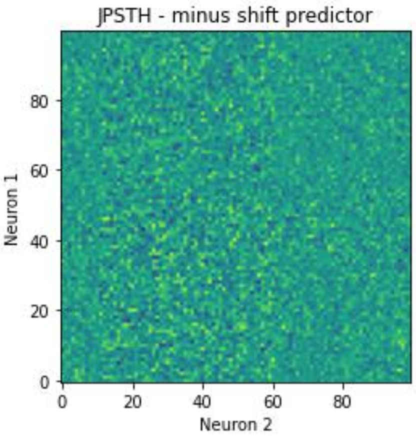
A. I used 20 ms pre-stimulus and 200 ms post-stimulus, just after some visual inspection. I also used a rectangular 10 ms sliding window to smooth the PSTH. From these figures, we can estimate the neuron's baseline firing rate, the delay until the response to the stimulus, and the increase and duration in firing rate post-stimulus.

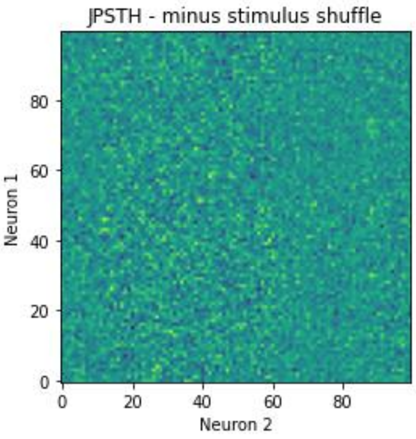




B. I took the first 100 ms after stimulus per neuron, and normalized by the mean, such that the ij position shows the percent of neuron A firing at i and neuron B firing at j for the no reduction JPSTH (i from bottom to top). It is easy to see the clusters in the no reduction JPSTH align nicely with our separate PSTH plots from earlier, and neuron A increases its firing rate immediately post-stimulus while neuron B increases its firing rate about 10 ms post-stimulus, both for about 50 ms. After removing the effect of the stimulus using both methods, I cannot determine any relationship between the neurons (no noticeable diagonals or clusters).

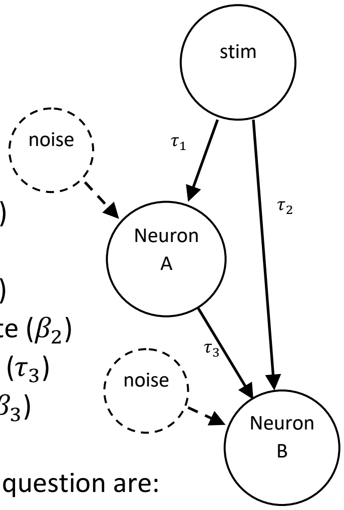






C. The parameters which we care about are:

    a. Neuron A baseline firing rate
    b. Neuron B baseline firing rate (β_0)
    c. Stimulation present (S_t)
    d. Stimulation effect on neuron A firing rate
    e. Lag between stimulation and effect on neuron A (τ_1)
    f. Stimulation effect on neuron B firing rate (β_1)
    g. Lag between stimulation and effect on neuron B (τ_2)
    h. Baseline neuron A spike effect on neuron B firing rate (β_2)
    i. Lag between neuron A spike and effect on neuron B (τ_3)
    j. Stimulation effect on neuron A effect on neuron B (β_3)



The additional parameters which we assume constant for this question are:

    a. Random noise (set $\varepsilon \sim N(0,1)$
    b. Refractory periods (set 3 ms absolute, no relative)
    c. Trial duration (set 1 trial of 500 s, equivalent to 500 trials of 1 s)
    d. Stationarity and homogeneity (no change during trial or between trials)
    e. Adaptation to stimulation (none)
    f. Number of stimulations (set 500)
    g. Duration and intervals of stimulation (set duration 500 ms, random intervals)
    h. Duration of stimulation effect on neurons (set 1 ms per 1 ms of stimulation)
    i. Lags (remain constant)

A simple approach is to model the firing rate of neuron B at any given time point t using the probability of B to fire and the following logistic model:

$$\boldsymbol{Logit(P(B_t)) = \beta_0 + \beta_1 \cdot S_{t-\tau_2} + \beta_2 \cdot A_{t-\tau_3} + \beta_3 \cdot S_{t-\tau_1-\tau_3} \cdot A_{t-\tau_3} + \varepsilon}$$

Where $\boldsymbol{P(B_t) = 0}$ if B fired in the previous 3 ms.

While it is probably more accurate to build a Poisson model, or do something smarter with the firing rate instead of treating it as 1000*probability of firing in every ms (binomial approximation), or even adjust for the almost nonexistent cases where the stimulation excited A but not B, I thought this method could still be pretty good.

The effectiveness of the stimulation on the neurons' activity, to me, means how much the stimulation increases the effect of neuron A on neuron B. In other words, the interesting question is: *after controlling for the effect of the stimulation on A and on B and the baseline excitation relation between A and B, what is the effect of the stimulation on the excitation relation between A and B?* I hope it is clear that in our linear model, this is the interaction term β_3, where the other terms are what we wish to control for. To evaluate effectiveness, we just need to compute the model's predicted rate for B when A fires due to stimulation minus when A fires from baseline.

I wrote a code in python to simulate neurons with various relations and check if my measure actually manages to capture the effectiveness of the stimulation on the neurons' excitation relation. It seems to do so, albeit with a severe underestimation (most of the time), which I did not diagnose. Different lag lengths did not affect the model as far as I could tell. Here is a table of examples:

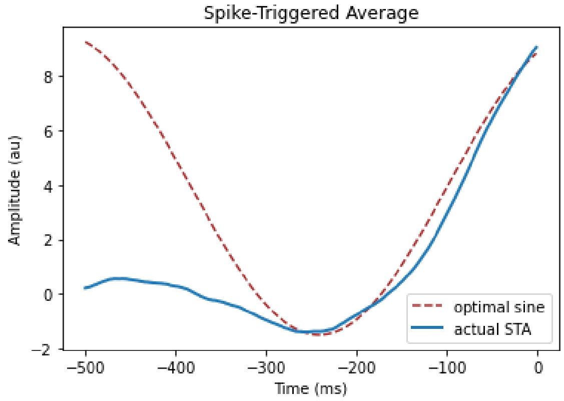
A0	B0	Stim on A	Stim on B	A on B	Stim on A on B	Eff-score
60.00	50.00	10.00	10.00	20.00	10.00	6.15
60.00	50.00	10.00	10.00	20.00	100.00	82.41
20.00	20.00	10.00	10.00	10.00	10.00	4.32
60.00	50.00	30.00	20.00	20.00	50.00	37.66
60.00	50.00	30.00	20.00	0.00	20.00	14.59
60.00	50.00	0.00	0.00	10.00	20.00	15.74
60.00	50.00	100.00	10.00	10.00	20.00	9.18
60.00	50.00	10.00	10.00	500.00	200.00	135.10
60.00	50.00	200.00	10.00	500.00	200.00	212.36
200.00	200.00	10.00	10.00	20.00	20.00	14.99
60.00	50.00	30.00	50.00	20.00	50.00	22.23

Notice that the eff-score tracks the effect of stim on A on b, which is our goal.

It is fairly easy to generalize this method to include duration of effects and adaptation – we just to first convolve the preceding stimulation with the appropriate kernel before putting it in the model. However, calculating the exact kernels while controlling for other variables may be difficult.

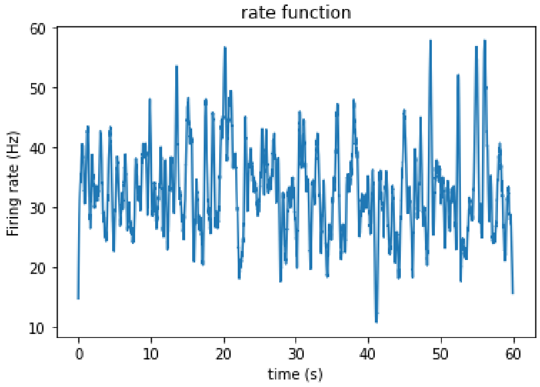
2) Encoding: GLM: optimal kernel and STA

A. Calculated the STA in code and fit a close to optimal sine wave for this neuron (just by eye). The sine wave frequency I found is 1.75 hz with a phase shift of -pi/2+0.5.

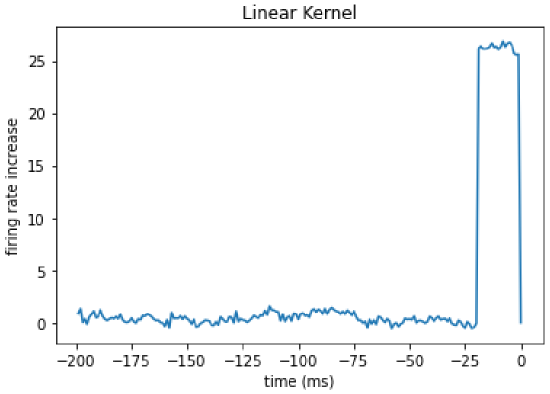


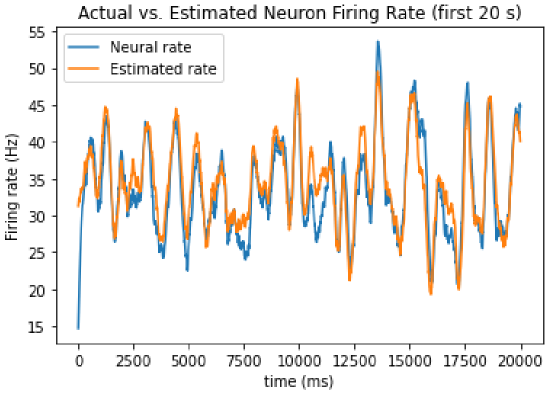
$\mathbf{4 + 5.5 \cdot \cos(1.75 \cdot 2 \cdot \pi + 0.5)}$

B. The best window in my opinion is a gaussian (size 0.5 sec and std of 0.15 sec). This is because it smoothed out the neuron's firing rate but did not lose too much information. Since we are also looking to fit a GLM to this r(t), it is important not to use too large windows which can obfuscate local correlations between stimulus and firing rate.

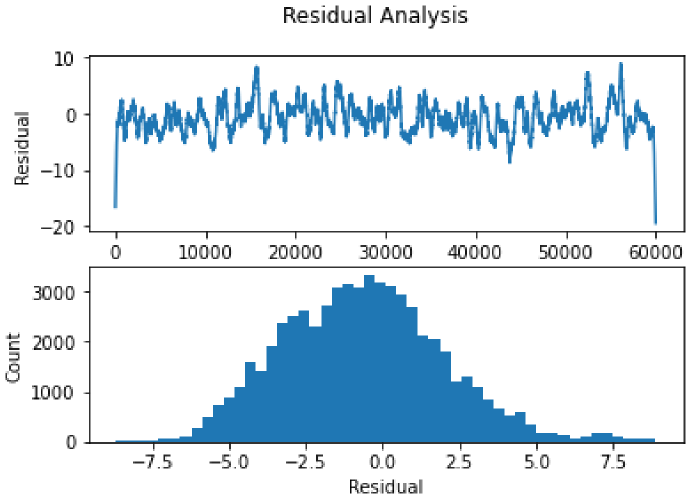


Using what we know about white-noise stimuli, I calculated the optimal linear kernel by a normalized sum of the stimuli and rate for each lag, and it can be seen that the neuron simply sums the stimulus presented for the preceding 20 ms (left figure). I then used the kernel (up to 20 ms backwards), the stimuli, and the gaussian window to estimate the firing rate (right figure). The RMSE is 2.839, which I think is small.





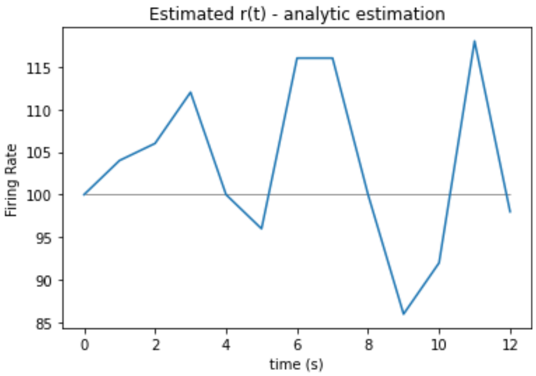
I may have misunderstood the question, but I find it a bit pointless to try and optimize the kernel further by changing its size – it's extremely obvious visually that the kernel is using the last 20 ms and then it is around 0 for any further lag, meaning that taking larger kernels will not improve much than the zero-padding which occurs anyway, and obviously smaller kernels will be less accurate. While theoretically a non-linear kernel could have provided a better fit, we see from r(t) that this specific neuron is not so bounded from the reaction to the stimuli (eliminating our need for rectifiers or other static non-linearity), as it doesn't reach rate of 0 for example. I also think it is pretty clear from the visual inspection of the residuals that they are mostly normally distributed and there is no clear non-linear trend we missed.

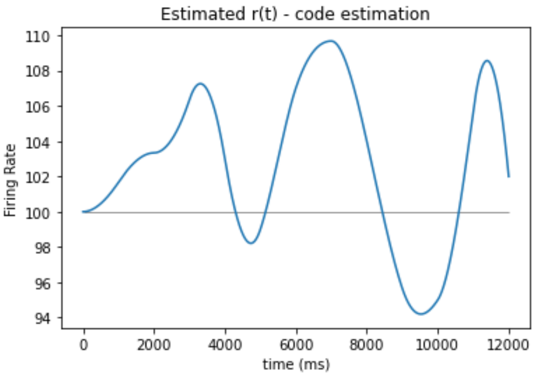


3) Analytical computations for optimal kernel

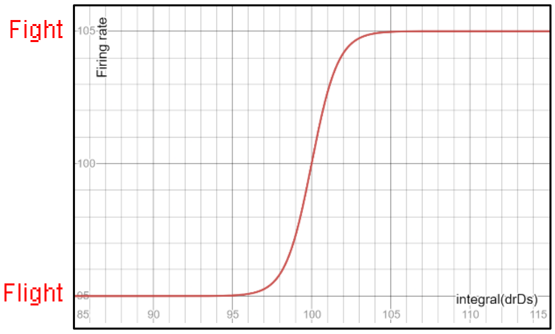
A. r_0 = 100 (arbitrarily) Kernel = [0, …, 0, -2, 2, 4] Stimulus = [0, …, 0, 1, 1, 3, -1, 1, 3, 3, 0, -2, -1, 4, -3]
r(0) = 100 + 0*4 + 0*2 - 0*2 = 100 r(1) = 100 + 1*4 + 0*2 - 0*2 = 104 r(2) = 100 + 1*4 + 1*2 - 0*2 = 106
r(3) = 100 + 3*4 + 1*2 - 1*2 = 112 r(4) = 100 + (-1)*4 + 3*2 - 1*2 = 100 r(5) = 100 + 1*4 + (-1)*2 - 3*2 = 96
r(6) = 100 + 3*4 + 1*2 - (-1)*2 = 116 r(7) = 100 + 3*4 + 3*2 - 1*2 = 116 r(8) = 100 + 0*4 + 3*2 - 3*2 = 100
r(9) = 100 + (-2)*4 + 0*2 - 3*2 = 86 r(10) = 100 + (-1)*4 + (-2)*2 - 0*2 = 92 r(11) = 100 + 4*4 + (-1)*2 - (-2)*2 = 118
r(12) = 100 + (-3)*4 + 4*2 - (-1)*2 = 98

One way to provide an estimation of r(t) is to assume that the functions are discrete and thus calculate the convolution on their pivot points as shown above, and then linearly connect the new points (see left figure). However, we know that this is a wrong assumption, and convolution of linear functions is not necessarily linear itself, so I simulated a code to approximate r(t) with a much better resolution (see right figure).

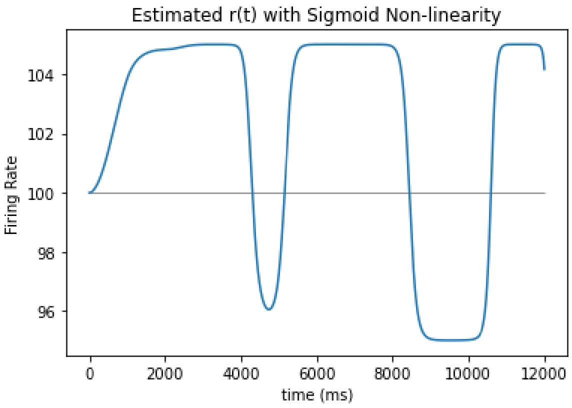




B. Looking at the neuron's response to the stimulus, I think a somewhat intuitive sigmoid fitting would be such that L (product of linear filter) above 104 translates to response A (let's say fight) by firing at 105 spikes per second, and L below 96 to response B (flight) by firing at only 95 spikes per second, which makes our $r_{½}$ value the midpoint 100, conveniently also our baseline firing rate. I also think it is important not to make the slope (beta) too steep, since hypothetically that could mean 1) no stimuli or minor changes from baseline will produce a costly behavioral response, and 2) much higher chance for errors of both types. Thus, the model I implemented[1] is on the left, and the neuron's firing rate in response to the stimuli is on the right:

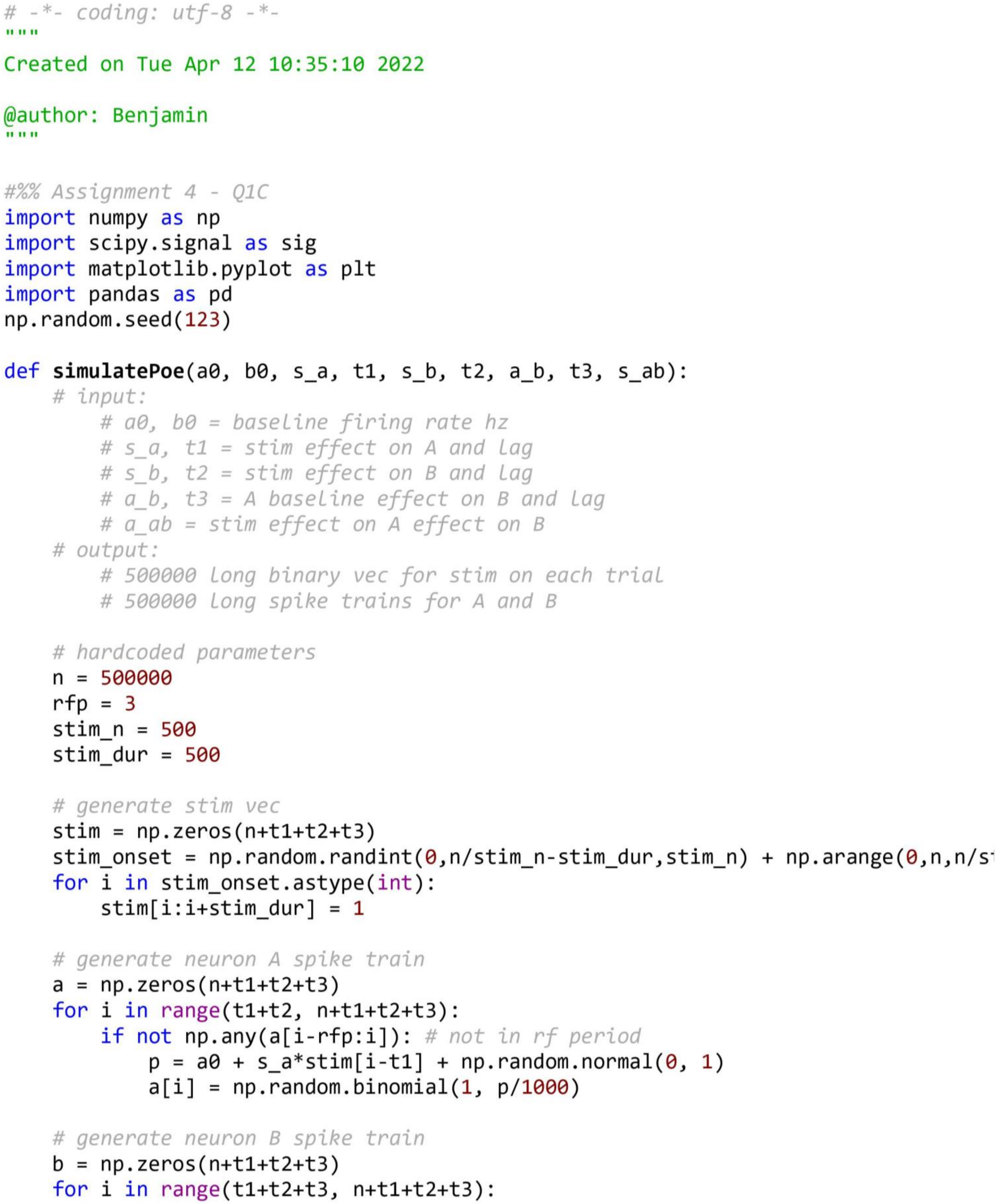


$$r(t) = 95 + 10 \cdot \frac{1}{1 + e^{1.2\cdot(100 - \int drDs)}}$$

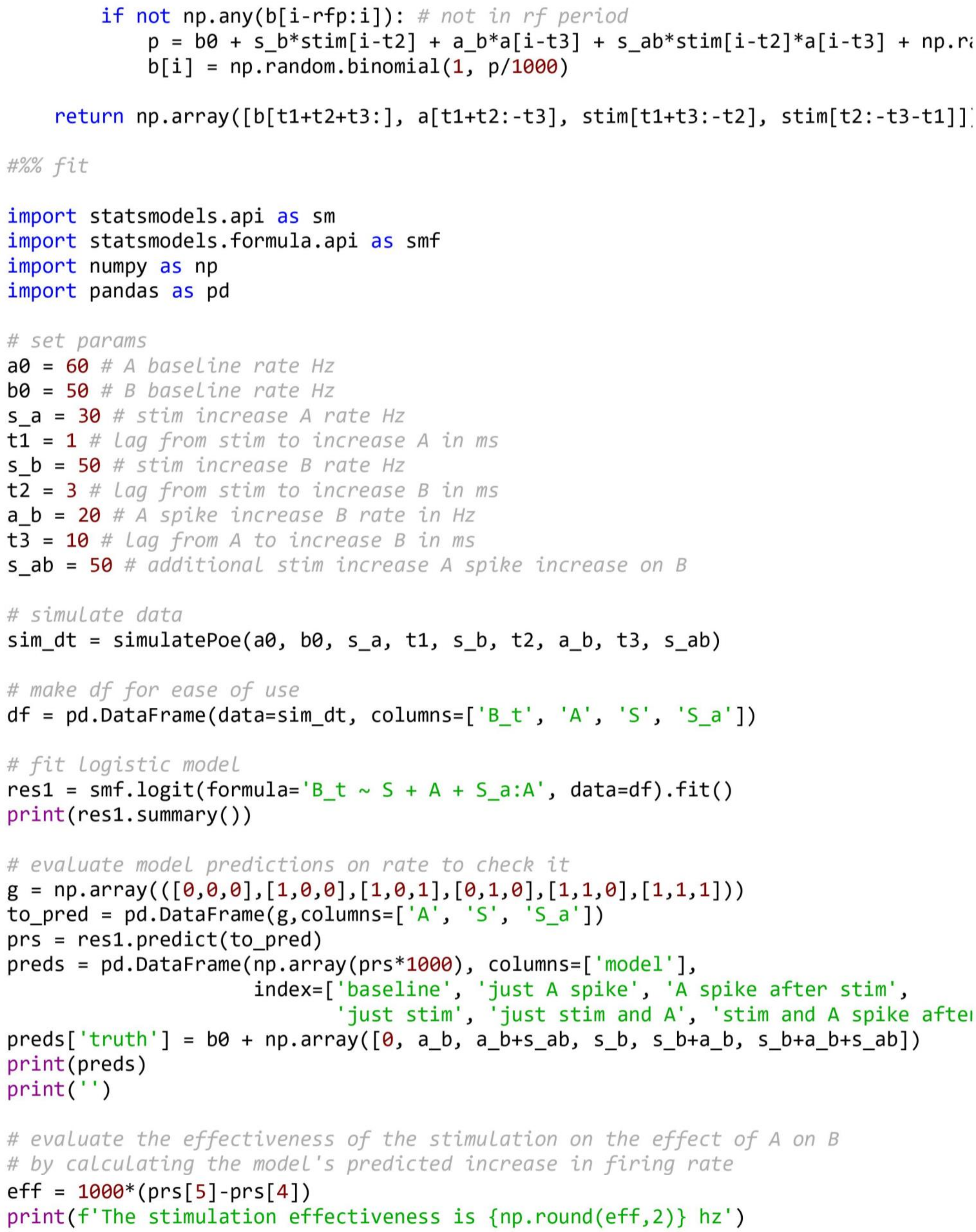


[1] After playing with the parameters in the graphing notebook I created here: https://www.desmos.com/calculator/9xa5mxy4j4. Notice you can press play on the parameters to see how they affect the sigmoid function.

Appendix – Code for 1 C

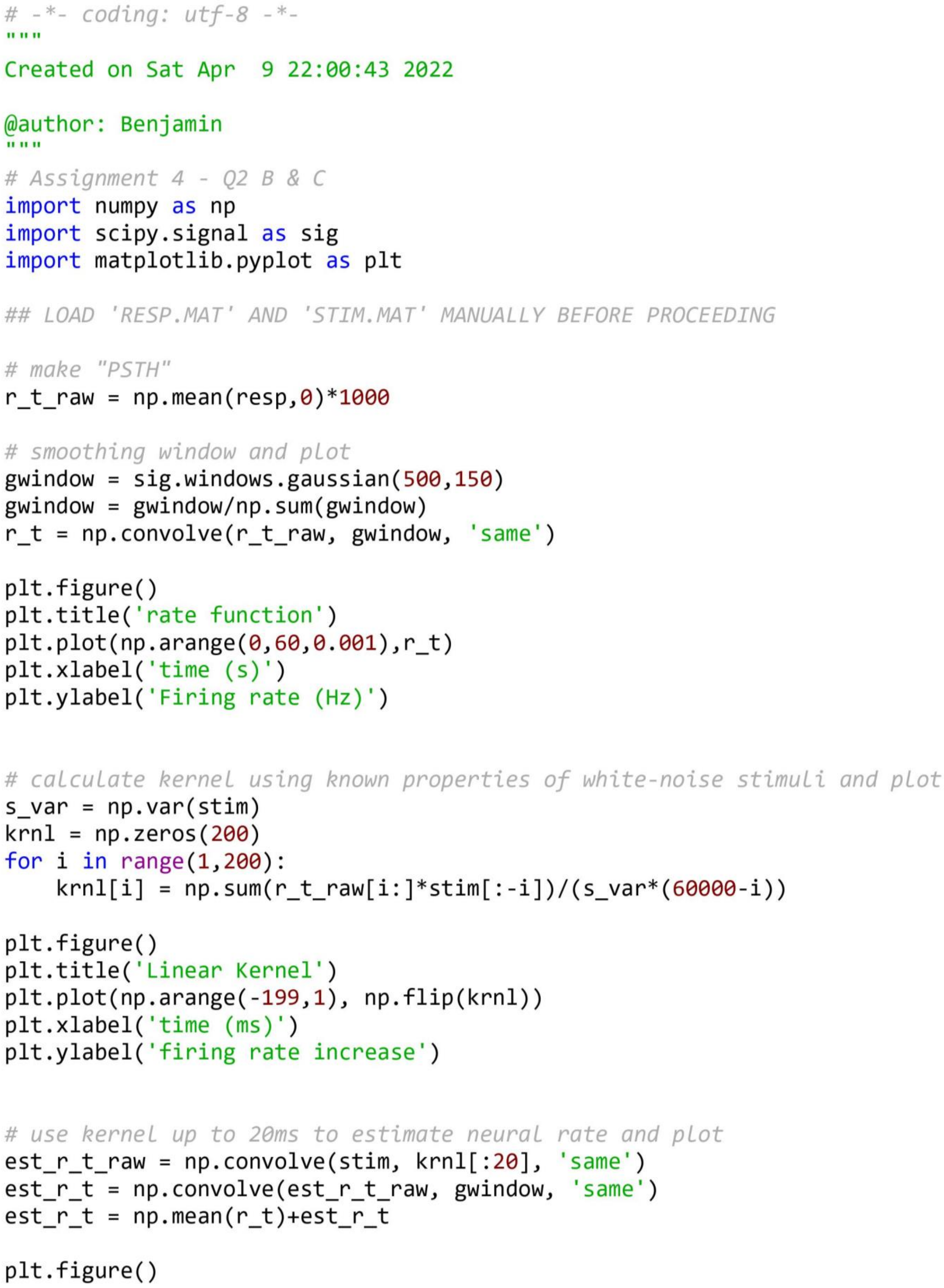


```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 12 10:35:10 2022

@author: Benjamin
"""

#%% Assignment 4 - Q1C
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt
import pandas as pd
np.random.seed(123)

def simulatePoe(a0, b0, s_a, t1, s_b, t2, a_b, t3, s_ab):
    # input:
        # a0, b0 = baseline firing rate hz
        # s_a, t1 = stim effect on A and lag
        # s_b, t2 = stim effect on B and lag
        # a_b, t3 = A baseline effect on B and lag
        # a_ab = stim effect on A effect on B
    # output:
        # 500000 long binary vec for stim on each trial
        # 500000 long spike trains for A and B

    # hardcoded parameters
    n = 500000
    rfp = 3
    stim_n = 500
    stim_dur = 500

    # generate stim vec
    stim = np.zeros(n+t1+t2+t3)
    stim_onset = np.random.randint(0,n/stim_n-stim_dur,stim_n) + np.arange(0,n,n/s
    for i in stim_onset.astype(int):
        stim[i:i+stim_dur] = 1

    # generate neuron A spike train
    a = np.zeros(n+t1+t2+t3)
    for i in range(t1+t2, n+t1+t2+t3):
        if not np.any(a[i-rfp:i]): # not in rf period
            p = a0 + s_a*stim[i-t1] + np.random.normal(0, 1)
            a[i] = np.random.binomial(1, p/1000)

    # generate neuron B spike train
    b = np.zeros(n+t1+t2+t3)
    for i in range(t1+t2+t3, n+t1+t2+t3):
```

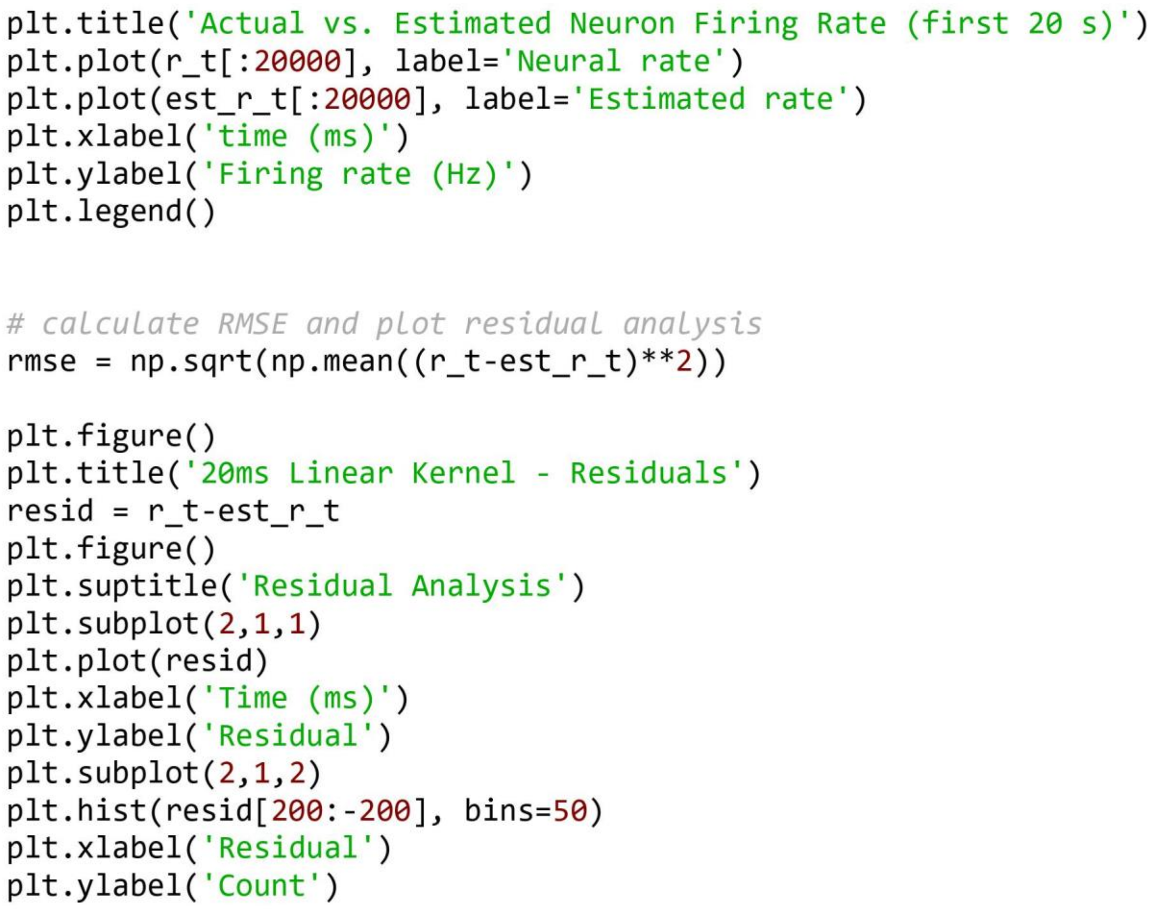


```
        if not np.any(b[i-rfp:i]): # not in rf period
            p = b0 + s_b*stim[i-t2] + a_b*a[i-t3] + s_ab*stim[i-t2]*a[i-t3] + np.ra
            b[i] = np.random.binomial(1, p/1000)

    return np.array([b[t1+t2+t3:], a[t1+t2:-t3], stim[t1+t3:-t2], stim[t2:-t3-t1]]

#%% fit

import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd

# set params
a0 = 60 # A baseline rate Hz
b0 = 50 # B baseline rate Hz
s_a = 30 # stim increase A rate Hz
t1 = 1 # lag from stim to increase A in ms
s_b = 50 # stim increase B rate Hz
t2 = 3 # lag from stim to increase B in ms
a_b = 20 # A spike increase B rate in Hz
t3 = 10 # lag from A to increase B in ms
s_ab = 50 # additional stim increase A spike increase on B

# simulate data
sim_dt = simulatePoe(a0, b0, s_a, t1, s_b, t2, a_b, t3, s_ab)

# make df for ease of use
df = pd.DataFrame(data=sim_dt, columns=['B_t', 'A', 'S', 'S_a'])

# fit logistic model
res1 = smf.logit(formula='B_t ~ S + A + S_a:A', data=df).fit()
print(res1.summary())

# evaluate model predictions on rate to check it
g = np.array(([0,0,0],[1,0,0],[1,0,1],[0,1,0],[1,1,0],[1,1,1]))
to_pred = pd.DataFrame(g,columns=['A', 'S', 'S_a'])
prs = res1.predict(to_pred)
preds = pd.DataFrame(np.array(prs*1000), columns=['model'],
                     index=['baseline', 'just A spike', 'A spike after stim',
                            'just stim', 'just stim and A', 'stim and A spike afte
preds['truth'] = b0 + np.array([0, a_b, a_b+s_ab, s_b, s_b+a_b, s_b+a_b+s_ab])
print(preds)
print('')

# evaluate the effectiveness of the stimulation on the effect of A on B
# by calculating the model's predicted increase in firing rate
eff = 1000*(prs[5]-prs[4])
print(f'The stimulation effectiveness is {np.round(eff,2)} hz')
```

Appendix – Code for 2 B&C



```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr  9 22:00:43 2022

@author: Benjamin
"""
# Assignment 4 - Q2 B & C
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt

## LOAD 'RESP.MAT' AND 'STIM.MAT' MANUALLY BEFORE PROCEEDING

# make "PSTH"
r_t_raw = np.mean(resp,0)*1000

# smoothing window and plot
gwindow = sig.windows.gaussian(500,150)
gwindow = gwindow/np.sum(gwindow)
r_t = np.convolve(r_t_raw, gwindow, 'same')

plt.figure()
plt.title('rate function')
plt.plot(np.arange(0,60,0.001),r_t)
plt.xlabel('time (s)')
plt.ylabel('Firing rate (Hz)')


# calculate kernel using known properties of white-noise stimuli and plot
s_var = np.var(stim)
krnl = np.zeros(200)
for i in range(1,200):
    krnl[i] = np.sum(r_t_raw[i:]*stim[:-i])/(s_var*(60000-i))

plt.figure()
plt.title('Linear Kernel')
plt.plot(np.arange(-199,1), np.flip(krnl))
plt.xlabel('time (ms)')
plt.ylabel('firing rate increase')


# use kernel up to 20ms to estimate neural rate and plot
est_r_t_raw = np.convolve(stim, krnl[:20], 'same')
est_r_t = np.convolve(est_r_t_raw, gwindow, 'same')
est_r_t = np.mean(r_t)+est_r_t

plt.figure()
```



```
plt.title('Actual vs. Estimated Neuron Firing Rate (first 20 s)')
plt.plot(r_t[:20000], label='Neural rate')
plt.plot(est_r_t[:20000], label='Estimated rate')
plt.xlabel('time (ms)')
plt.ylabel('Firing rate (Hz)')
plt.legend()


# calculate RMSE and plot residual analysis
rmse = np.sqrt(np.mean((r_t-est_r_t)**2))

plt.figure()
plt.title('20ms Linear Kernel - Residuals')
resid = r_t-est_r_t
plt.figure()
plt.suptitle('Residual Analysis')
plt.subplot(2,1,1)
plt.plot(resid)
plt.xlabel('Time (ms)')
plt.ylabel('Residual')
plt.subplot(2,1,2)
plt.hist(resid[200:-200], bins=50)
plt.xlabel('Residual')
plt.ylabel('Count')
```