Homework Problem Set 1

Arthur Li

Washington University in St. Louis
BME 572
Instructor: Barani Raman

Date Assigned: 1/15/2020
Date Due: 2/10/2020

I, Arthur Li, hereby certify that this report is my original work. Special thanks to Pratyush Ramakrishna, Andrew Van, and Dan Fu for clarifying concepts and providing suggestions on debugging.

Arthur Li

Problem 1

Methods (Raman):

(a) Implement the Hodgkin-Huxley (HH) Model

1. Implement the equations below using Euler's integration technique in MATLAB.

$$I = C_M \frac{\mathrm{d}V}{\mathrm{d}t} + \bar{g}_K n^4 (V - V_K) + \bar{g}_{Na} m^3 h (V - V_{Na}) + \bar{g}_l (V - V_l),$$

where

$$\mathrm{d}n/\mathrm{d}t = \alpha_n(1-n) - \beta_n n,$$
$$\mathrm{d}m/\mathrm{d}t = \alpha_m(1-m) - \beta_m m,$$
$$\mathrm{d}h/\mathrm{d}t = \alpha_h(1-h) - \beta_h h,$$

$$\alpha_n = 0{\cdot}01\ (V+10) \Big/ \left(\exp\frac{V+10}{10} - 1\right),$$
$$\beta_n = 0{\cdot}125\ \exp\ (V/80),$$
$$\alpha_m = 0{\cdot}1\ (V+25) \Big/ \left(\exp\frac{V+25}{10} - 1\right),$$
$$\beta_m = 4\ \exp\ (V/18),$$
$$\alpha_h = 0{\cdot}07\ \exp\ (V/20),$$
$$\beta_h = 1 \Big/ \left(\exp\frac{V+30}{10} + 1\right).$$

2. Define the following constants: V = -65-V, C = 1 microF/cm2, Leak reversal potential = -61 mV, K reversal potential = -77mV, Na reversal potential = 55 mv, Leakage conductance (gL = 0.3 mS/cm2), K conductance (gK = 36 mS/cm2, and N conductance (gNa) = 120 mS/cm2.

3. Execute the model in a loop for 100 mS. Inject the current I = 20 uA at time t = 60 mS and turn it off at t = 63 mS.

(b) Implement the Quadratically Fitted HH Model

1. Replace the exponential relation between α_n and V with a quadratic fit.

2. Repeat procedure (a) with the exp relation.

(c) Implement the Reduced HH Model

1. Repeat procedure (a) with the reduced version of the HH model below

$$\text{Original} \quad C\dot{V} = I - \overbrace{g_K n^4(V - E_K)}^{I_K} - \overbrace{g_{Na} m^3 h(V - E_{Na})}^{I_{Na}} - \overbrace{g_L(V - E_L)}^{I_L}$$

$$\text{Reduced}\, C\dot{V} = I - \overbrace{g_K n^4(V - E_K)}^{I_K} - \overbrace{g_{Na} m_\infty^3(V)(0.89 - 1.1n)(V - E_{Na})}^{\text{instantaneous } I_{Na}} - \overbrace{g_L(V - E_L)}^{I_L}$$

, where m was replaced with its steady state value and h becomes 0.89-1.1n.

For (a) to (c), the following figures were plotted on 0 to 100 mS time interval:

(i) Evolution of the HH variables m,n,h following an action potential.

(ii) Relative evolution of sodium and potassium conductances

(iii) Relative evolution of capacitive, leak, sodium and potassium currents

(iv) Demonstrate the threshold and rebound spiking behavior of the neuron

The figures and code are shown in Result and Problem 1 Code sections respectively.

Result:

Figure 1.1: Plots from (a) HH Model

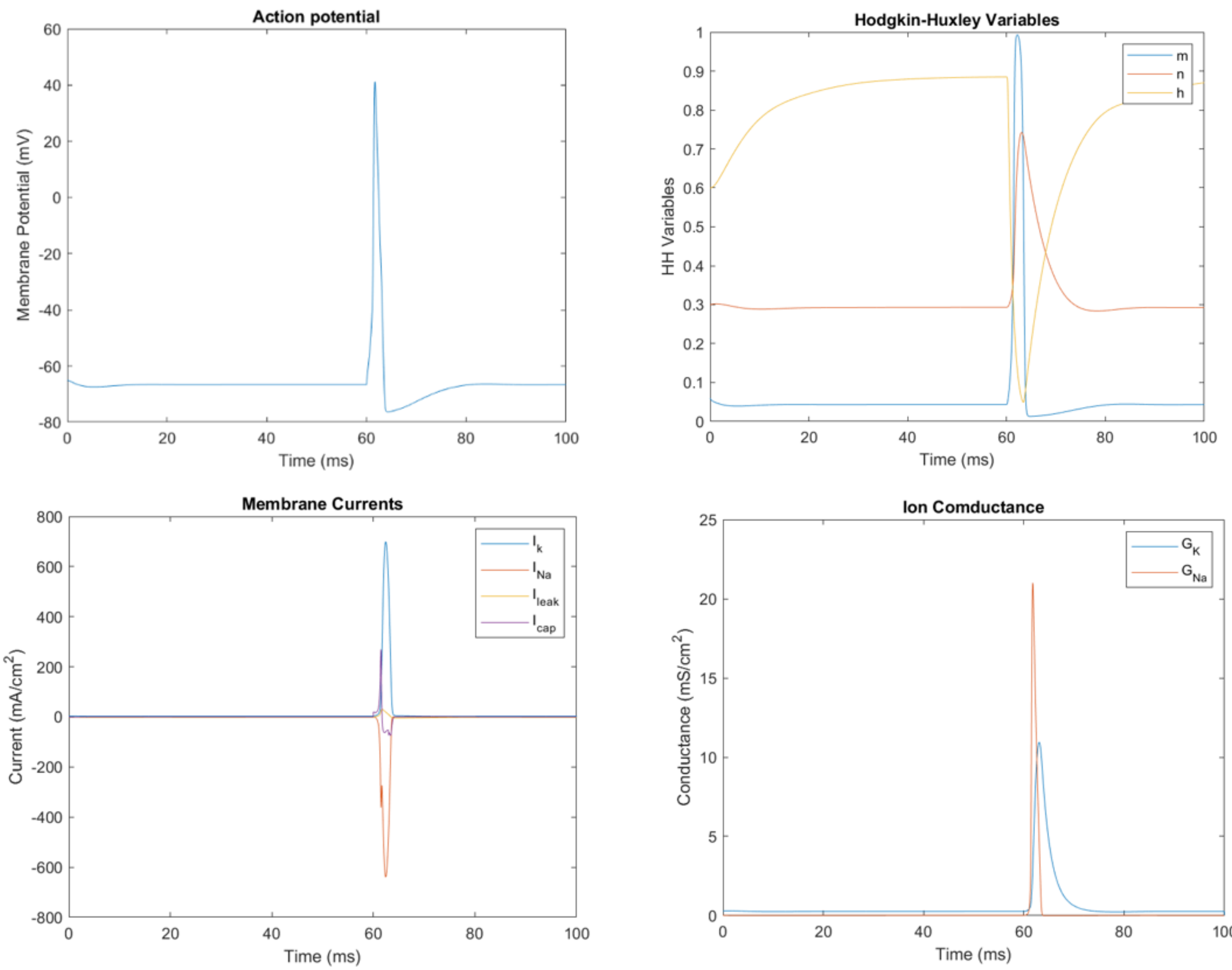


Figure 1.2: Plots from (b) Quadratically Fitted HH Model

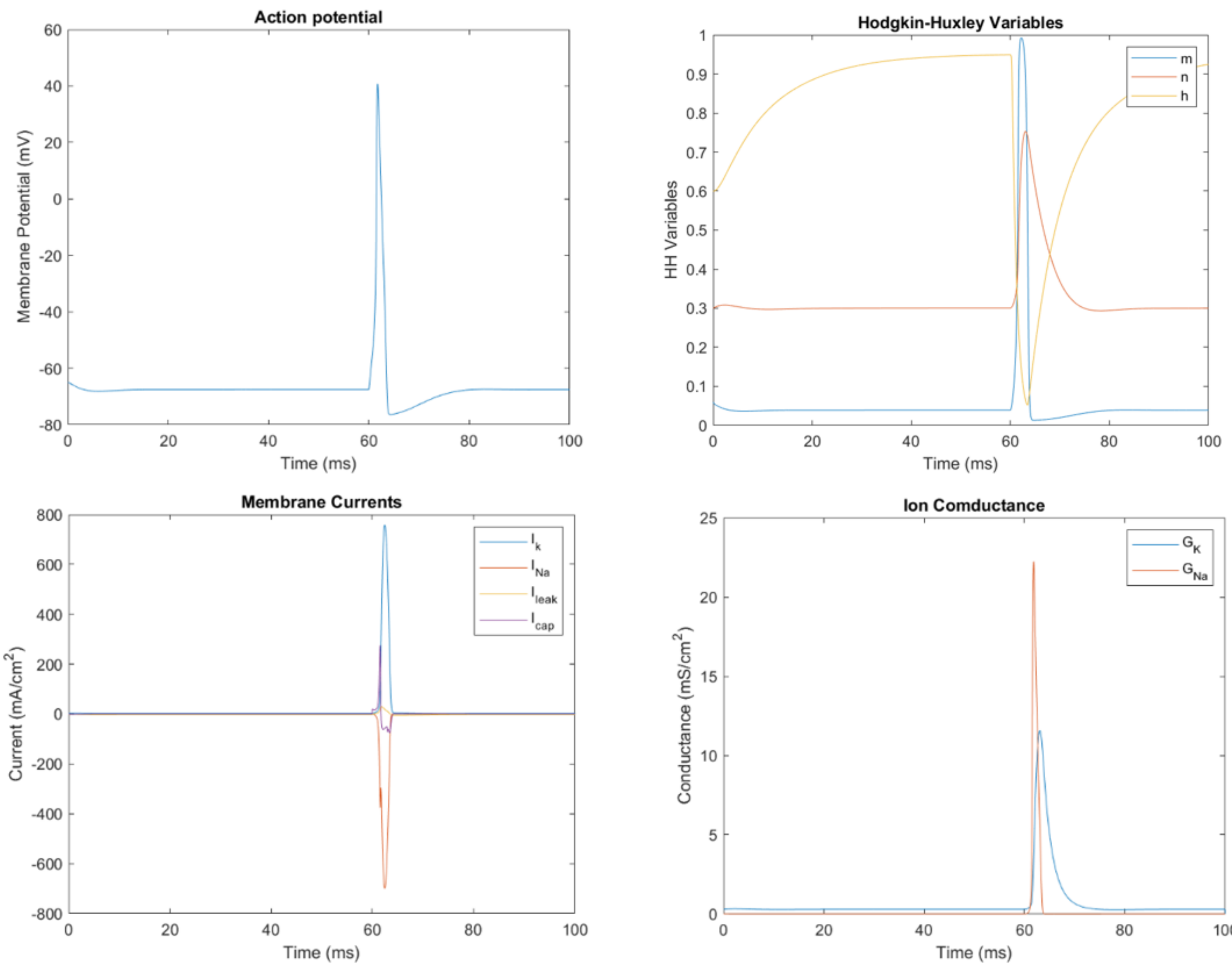
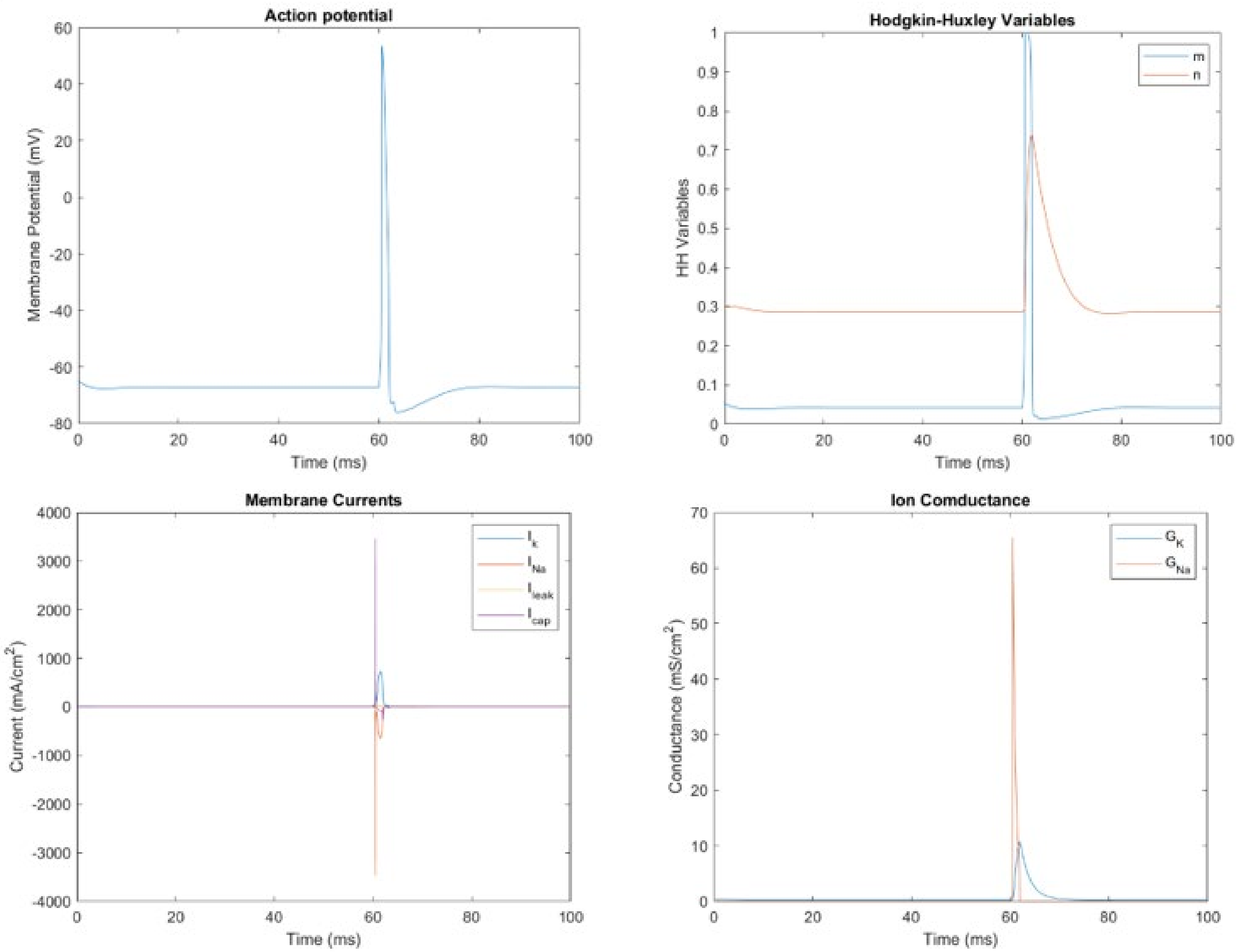


Figure 1.3: Plots from (c) Reduced HH Model



Discussion

(a) As shown in Figure 1.1-Action Potential, the HH model nicely simulate the cell membrane potential of neuron as it resembles the threshold nature of action potential. In Figure 1.1-Hodgkin-Huxley Variables, we can observe that the variables change rapidly at t = 60 ms, which produce the behavior of the action potential seen in Figure 1.1-Action Potential. Overall, m and n plots have the similar shape with that of the action potential while h has the opposite one. As shown in Figure 1.1-Membrane Currents, there appear symmetrical curve shapes between Ik & INa as well as Ileak & Icap, where the sudden change in currents is due to the injection of current at t=60ms, and the 4 currents restore when stop injecting current at t=63ms. Figure 1.1-Ion Conductance indicates the potassium channels have a higher conductance than sodium, which is as expected.

(b) Comparing from Figure 1.1 and 1.2, we observe no obvious difference between the original HH model and the quadratically fitted model. It implies the curve fit is satisfactory and does not cause deviation from the original.

(c) Comparing from Figure 1.1 and 1.3, we observe the plots from the reduced HH model are overall taller and narrower than the original model given the same injected current. Also, it's worth to notice that potassium and sodium currents from the reduced model respond to the injected currents a lot faster than the original.

```
%Problem 1 (a) Code
%Implement the HH model in Matlab.
step_num = 100000;

% Initiliza arrays
n = zeros(1,step_num);
m = zeros(1,step_num);
h = zeros(1,step_num);
V = zeros(1,step_num);

ConducK = zeros(1,step_num+1);
ConducNa = zeros(1,step_num+1);
I_k = zeros(1,step_num+1);
I_Na = zeros(1,step_num+1);
I_leak = zeros(1,step_num+1);
I_cap = zeros(1,step_num+1);

% Define constants
Cm = 1;
I = 0;
Vl = -61;
Vk = -77;
VNa = 55;
gL = 0.3;
gK = 36;
gNa = 120;

dt=0.001; % step's size
T = 0:100000;

% Define init values
n(1) = 0.3;
m(1) = 0.06;
h(1) = 0.6;
V(1) = -65;
ConducK(1) = (gK*(n(1)^4));
ConducNa(1) = (gNa*(m(1)^3)*h(1));

I_k(1) = ConducK(1) * (V(1)-Vk);
```

```
I_Na(1) = ConducNa(1) * (V(1)-VNa);
I_leak(1) = gL * (V(1)-Vl);


for t=1:(length(T)-1)
    if t==60000
        I=20;
    end

    if t==63000
        I=0;
    end

    %Define alpha and betas
    an = (0.01 * ((-65-V(t))+10))/(exp(((-65-
V(t))+10)/10)-1);
    bn = 0.125*exp((-65-V(t))/80);

    am = 0.1 * ((-65-V(t)) + 25)/(exp(((-65-
V(t))+25)/10)-1);
    bm = 4*exp((-65-V(t))/18);

    ah = 0.07 * exp((-65-V(t))/20);
    bh = 1/(exp(((-65-V(t))+30/10))+1);

    % Conductances
    ConducK(t) = gK*(n(t)^4);
    ConducNa(t) = gNa*(m(t)^3)*h(t);

    % Currents
    I_k(t) = ConducK(t) * (V(t)-Vk);
    I_Na(t) = ConducNa(t) * (V(t)-VNa);
    I_leak(t) = gL * (V(t)-Vl);
    if t>1
        I_cap(t) = (V(t)-V(t-1))/0.001;
    end

    % Membrane potential
```

```
    V(t+1) = V(t)+ ((I - I_k(t) - I_Na(t) -
I_leak(t))/Cm)*dt;

    % HH variables
    n(t+1) = n(t) + (an*(1-n(t)) - bn*n(t))*dt;
    m(t+1) = m(t) + (am*(1-m(t)) - bm*m(t))*dt;
    h(t+1) = h(t) + (ah*(1-h(t)) - bh*h(t))*dt;
end

a=figure(1);
plot(T*0.001,V);
xlabel('Time (ms)');
ylabel('Membrane Potential (mV)');
title('Action potential');
saveas(a,'p1-1-1.png');

b=figure(2);
plot(T*0.001,m);
xlabel('Time (ms)');
ylabel('HH Variables');
title('Hodgkin-Huxley Variables');
hold on
plot(T*0.001,n);
plot(T*0.001,h);
legend("m", "n", "h");
hold off
saveas(b,'p1-1-2.png');

%Current
c=figure(3);
disp(length(T))
disp(length(I_k))
plot(T*0.001,I_k);
title('Membrane Currents');
xlabel('Time (ms)');
ylabel('Current (mA/cm^2)');
hold on;
plot(T*0.001,I_Na);
```

```
plot(T*0.001,I_leak);
plot(T*0.001,I_cap);
legend("I_k", 'I_N_a', "I_l_e_a_k", "I_c_a_p");
hold off;
saveas(c,'p1-1-3.png');

%Conductance
d=figure(4);
plot(T*0.001,ConducK);
title('Ion Comductance');
xlabel('Time (ms)');
ylabel('Conductance (mS/cm^2)');
hold on;
plot(T*0.001,ConducNa);
legend("G_K", "G_N_a")
hold off;
saveas(d,'p1-1-4.png');
```

```
%Problem 1 (b) Code
%Replace the exponential relation between an and V
with a
%quadratic fit and check whether the model is still
valid

step_num = 100000;

% Initiliza arrays
n = zeros(1,step_num);
m = zeros(1,step_num);
h = zeros(1,step_num);
V = zeros(1,step_num);

ConducK = zeros(1,step_num+1);
ConducNa = zeros(1,step_num+1);
I_k = zeros(1,step_num+1);
I_Na = zeros(1,step_num+1);
I_leak = zeros(1,step_num+1);
I_cap = zeros(1,step_num+1);

% Define constants
Cm = 1;
I = 0;
Vl = -61;
Vk = -77;
VNa = 55;
gL = 0.3;
gK = 36;
gNa = 120;

dt=0.001; % step's size
T = 0:100000;

% Define init values
n(1) = 0.3;
m(1) = 0.06;
h(1) = 0.6;
V(1) = -65;
```

```
ConducK(1) = (gK*(n(1)^4));
ConducNa(1) = (gNa*(m(1)^3)*h(1));

I_k(1) = ConducK(1) * (V(1)-Vk);
I_Na(1) = ConducNa(1) * (V(1)-VNa);
I_leak(1) = gL * (V(1)-Vl);

%Polyfit an
po_V = linspace(-110,10,100);
po_an = (0.01*((-65-po_V)+10))./(exp((-65-
po_V+10)/10)-1);
poly = polyfit(-65-po_V, po_an,2);

for t=1:(length(T)-1)
    if t==60000
        I=20;
    end

    if t==63000
        I=0;
    end

    %Define alpha and betas
    %Poly evaluate
    an = poly(1)*((-65-V(t))^2) + poly(2)*(-65-V(t))
+ poly(3);

    bn = 0.125*exp((-65-V(t))/80);

    am = 0.1 * ((-65-V(t)) + 25)/(exp(((-65-
V(t))+25)/10)-1);
    bm = 4*exp((-65-V(t))/18);

    ah = 0.07 * exp((-65-V(t))/20);
    bh = 1/(exp(((-65-V(t))+30/10))+1);

    % Conductances
    ConducK(t) = gK*(n(t)^4);
    ConducNa(t) = gNa*(m(t)^3)*h(t);
```

```
    % Currents
    I_k(t) = ConducK(t) * (V(t)-Vk);
    I_Na(t) = ConducNa(t) * (V(t)-VNa);
    I_leak(t) = gL * (V(t)-Vl);
    if t>1
        I_cap(t) = (V(t)-V(t-1))/0.001;
    end

    % Membrane potential
    V(t+1) = V(t)+ ((I - I_k(t) - I_Na(t) -
I_leak(t))/Cm)*dt;
    n(t+1) = n(t) + (an*(1-n(t)) - bn*n(t))*dt;
    m(t+1) = m(t) + (am*(1-m(t)) - bm*m(t))*dt;
    h(t+1) = h(t) + (ah*(1-h(t)) - bh*h(t))*dt;


end

a=figure(1);
plot(T*0.001,V);
xlabel('Time (ms)');
ylabel('Membrane Potential (mV)');
title('Action potential');
saveas(a,'p1-2-1.png');

b=figure(2);
plot(T*0.001,m);
xlabel('Time (ms)');
ylabel('HH Variables');
title('Hodgkin-Huxley Variables');
hold on
plot(T*0.001,n);
plot(T*0.001,h);
legend("m", "n", "h");
hold off
saveas(b,'p1-2-2.png');

%Current
```

```
c=figure(3);
disp(length(T))
disp(length(I_k))
plot(T*0.001,I_k);
title('Membrane Currents');
xlabel('Time (ms)');
ylabel('Current (mA/cm^2)');
hold on;
plot(T*0.001,I_Na);
plot(T*0.001,I_leak);
plot(T*0.001,I_cap);
legend("I_k", 'I_N_a', "I_l_e_a_k", "I_c_a_p");
hold off;
saveas(c,'p1-2-3.png');

%Conductance
d=figure(4);
plot(T*0.001,ConducK);
title('Ion Comductance');
xlabel('Time (ms)');
ylabel('Conductance (mS/cm^2)');
hold on;
plot(T*0.001,ConducNa);
legend("G_K", "G_N_a")
hold off;
saveas(d,'p1-2-4.png');
```

```matlab
%Problem 1 (c) Code
%Implement the HH model in Matlab.

step_num = 100000;

% Initiliza arrays
n = zeros(1,step_num);
m = zeros(1,step_num);
h = zeros(1,step_num);
V = zeros(1,step_num);

ConducK = zeros(1,step_num+1);
ConducNa = zeros(1,step_num+1);
I_k = zeros(1,step_num+1);
I_Na = zeros(1,step_num+1);
I_leak = zeros(1,step_num+1);
I_cap = zeros(1,step_num+1);

% Define constants
Cm = 1;
I = 0;
Vl = -61;
Vk = -77;
VNa = 55;
gL = 0.3;
gK = 36;
gNa = 120;

dt=0.001; % step's size
T = 0:100000;

% Define init values
n(1) = 0.3;
m(1) = 0.06;
h(1) = 0.6;
V(1) = -65;
ConducK(1) = (gK*(n(1)^4));
ConducNa(1) = (gNa*(m(1)^3)*h(1));
```

```
I_k(1) = ConducK(1) * (V(1)-Vk);
I_Na(1) = CondecNa(1) * (V(1)-VNa);
I_leak(1) = gL * (V(1)-Vl);


for t=1:(length(T)-1)
    if t==60000
        I=30;
    end

    if t==63000
        I=0;
    end

    %Define alpha and betas
    an = (0.01 * ((-65-V(t))+10))/(exp(((-65-
V(t))+10)/10)-1);
    bn = 0.125*exp((-65-V(t))/80);

    am = 0.1 * ((-65-V(t)) + 25)/(exp(((-65-
V(t))+25)/10)-1);
    bm = 4*exp((-65-V(t))/18);

    ah = 0.07 * exp((-65-V(t))/20);
    bh = 1/(exp(((-65-V(t))+30/10))+1);

    % Conductances
    CondecK(t) = gK*(n(t)^4);
    CondecNa(t) = gNa*(m(t)^3)*(0.89-(1.1*n(t)));

    % Currents
    I_k(t) = CondecK(t) * (V(t)-Vk);
    I_Na(t) = CondecNa(t) * (V(t)-VNa);
    I_leak(t) = gL * (V(t)-Vl);
    if t>1
        I_cap(t) = (V(t)-V(t-1))/0.001;
    end

    % Membrane potential
```

```
    V(t+1) = V(t)+ ((I - I_k(t) - I_Na(t) -
I_leak(t))/Cm)*dt;

    % HH variables
    n(t+1) = n(t) + (an*(1-n(t)) - bn*n(t))*dt;
    m(t+1) = am/(am+bm);
end

a=figure(1);
plot(T*0.001,V);
xlabel('Time (ms)');
ylabel('Membrane Potential (mV)');
title('Action potential');
saveas(a,'p1-3-1.png');

b=figure(2);
plot(T*0.001,m);
xlabel('Time (ms)');
ylabel('HH Variables');
title('Hodgkin-Huxley Variables');
hold on
plot(T*0.001,n);
legend("m", "n");
hold off
saveas(b,'p1-3-2.png');

%Current
c=figure(3);
disp(length(T))
disp(length(I_k))
plot(T*0.001,I_k);
title('Membrane Currents');
xlabel('Time (ms)');
ylabel('Current (mA/cm^2)');
hold on;
plot(T*0.001,I_Na);
plot(T*0.001,I_leak);
plot(T*0.001,I_cap);
```

```
legend("I_k", 'I_N_a', "I_l_e_a_k", "I_c_a_p");
hold off;
saveas(c,'p1-3-3.png');

%Conductance
d=figure(4);
plot(T*0.001,ConducK);
title('Ion Comductance');
xlabel('Time (ms)');
ylabel('Conductance (mS/cm^2)');
hold on;
plot(T*0.001,ConducNa);
legend("G_K", "G_N_a")
hold off;
saveas(d,'p1-3-4.png');
```

<u>Problem 2</u>

Methods (Raman):

(a) Implement the Simple Integrate and Fire Neuron (IAF)

1. Implement the equations below using Euler's integration method in MATLAB

$$I(t) = \frac{V(t)}{R} + C\frac{dV(t)}{dt}$$

$$V(t) = \begin{cases} V(t-1) + \dfrac{dv(t)}{dt} \bullet dt & V(t) < V_{THR} \\ \\ V(t) = 70mv & \\ V(t+1) = 0 & V(t) \geq V_{THR} \end{cases}$$

2. Define the following constants: R=10 MOhm, C=1 nF, Vthr=5 mV, Vspk=70 mV, time step (dt) = 1 ms,

3. Generate a representative plot of the membrane voltage as a function of time for a step current injection with I=1 nA for 10 <= t <60 ms and I=0 otherwise.

4. Use sinusoidal currents with frequencies 1, 2, 5, 10, 20, 50, and 100 Hz to run the model for 1 second and plot each with membrane voltage vs. time

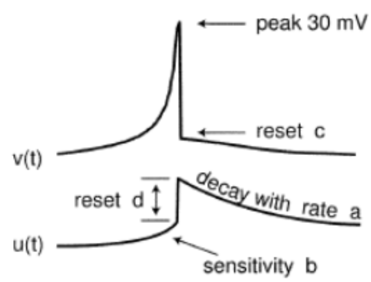
5. Generate a plot of "spike count vs. stimulus frequency," where "spike count" is the total number of spikes generated during the 1 second stimulus interval

(b) Implement the New Model

1. Implement the equations below using Euler's integration method in MATLAB

$$v' = 0.04v^2 + 5v + 140 - u + I$$
$$u' = a(bv - u)$$

if $v = 30$ mV,
then $v \leftarrow c, \quad u \leftarrow u + d$



2. Initialize the following variables: a = 0.02, b=0.2, c=-65, d=8,v = -65, u = b*v.

3. Repeat (a)-3 to (a)-5 with the New Model

(c) Construct the Two-neuron Oscillator using reciprocal inhibition

1. Implement the equations below using Euler's integration method in MATLAB

$$C\frac{dv}{dt} = \frac{-v}{R} - g_{syn}(v - E_{syn}) + I_{inject} \quad \text{voltage update}$$

$$\frac{d\theta}{dt} = \frac{-\theta + v}{\tau_{thresh}} \quad \text{threshold update}$$

$$\frac{dz}{dt} = \frac{-z}{\tau_{syn}} + \frac{g_{peak}}{(\tau_{syn}/e)}u(t) \quad \text{conductance update I}$$

$$\frac{dg}{dt} = \frac{-g}{\tau_{syn}} + z(t) \quad \text{conductance update II}$$

2. Define the following constants

$C = 1$	membrane capacitance (nF)
$R = 10$	membrane resistance ($M\Omega$)
$V_{rest} = 0$	resting membrane potential (mV)
$V_{spk} = 70$	action potential amplitude (mV)
$\tau_{thresh} = 50$	threshold time constant (ms)
$E_{inh} = -15$	synaptic reversal potential (mV)
$\tau_{syn} = 15$	synaptic time constant (ms)
$g_{peak} = 0.1$	peak synaptic conductance (μS)
$T_{\max} = 1500$	total simulation time (ms)
$\Delta t = 1$	integration time step (ms)

Esyn = Einh = -15 mV and e = 2.7183.

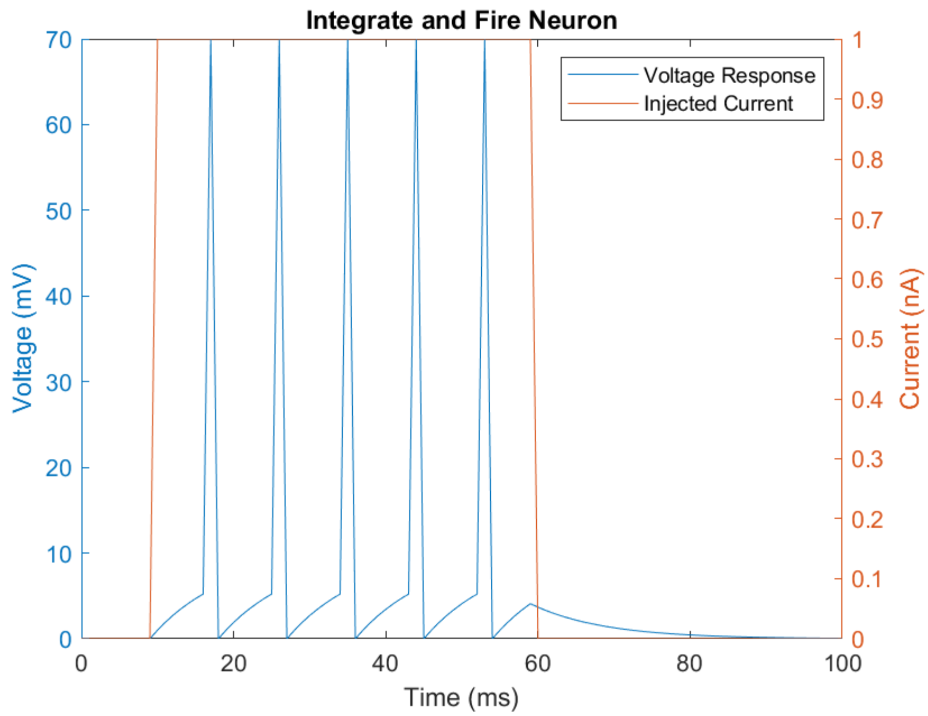
3. Inject 1.1 nA into neuron 1 and 0.9 nA into neuron 2.

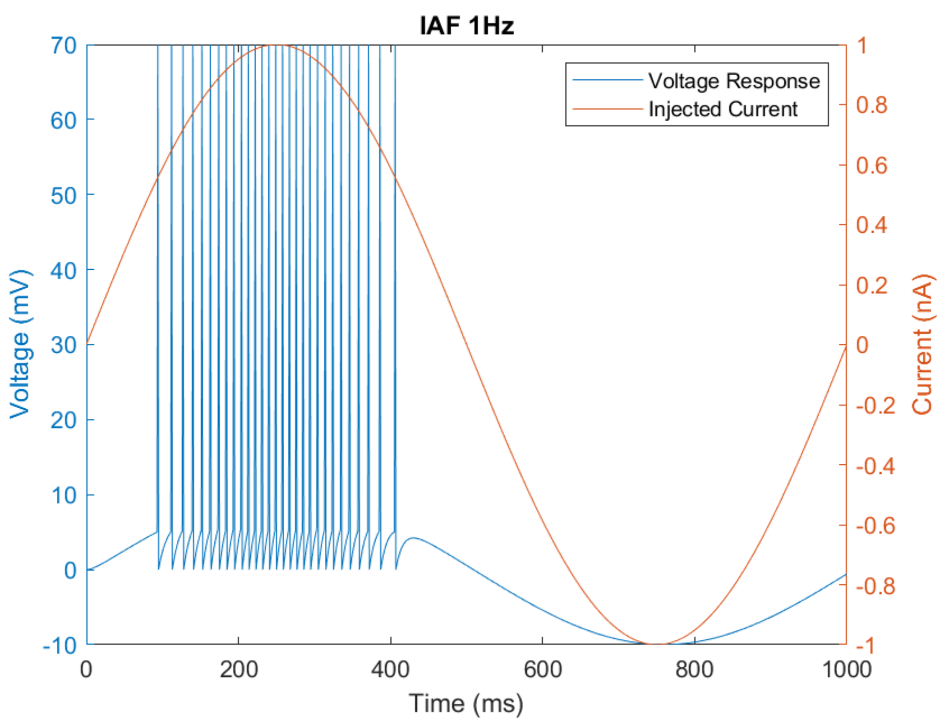
4. Reset the membrane voltage to Einh on the next time step when the neuron fires an action potential.

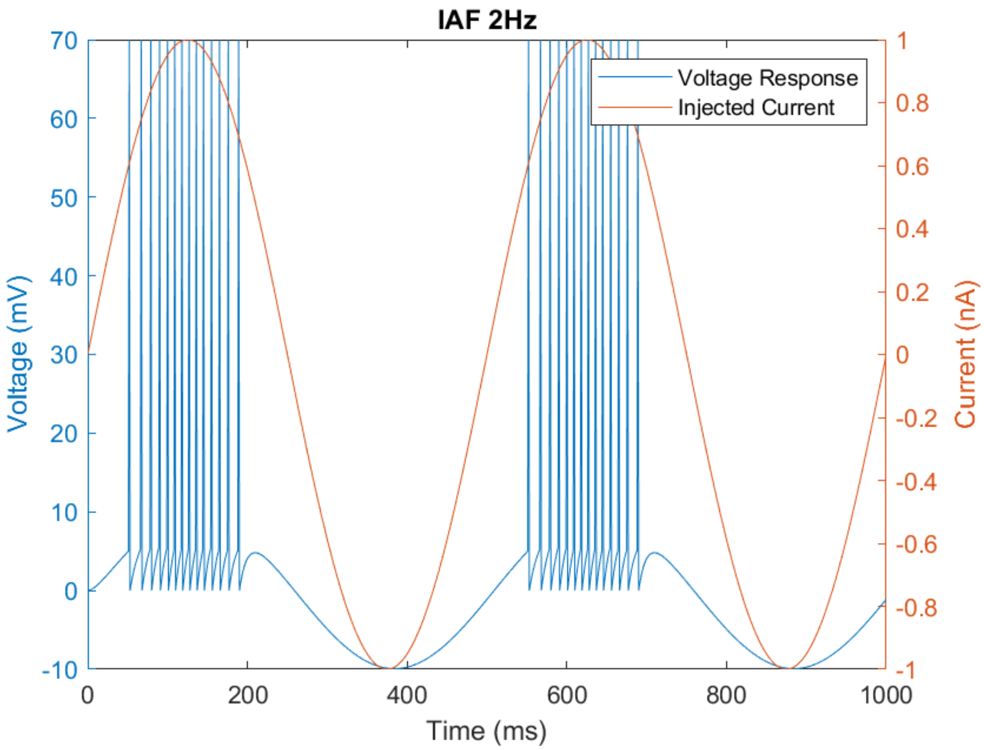
The figures and code are shown in Result and Problem 2 Code sections respectively.

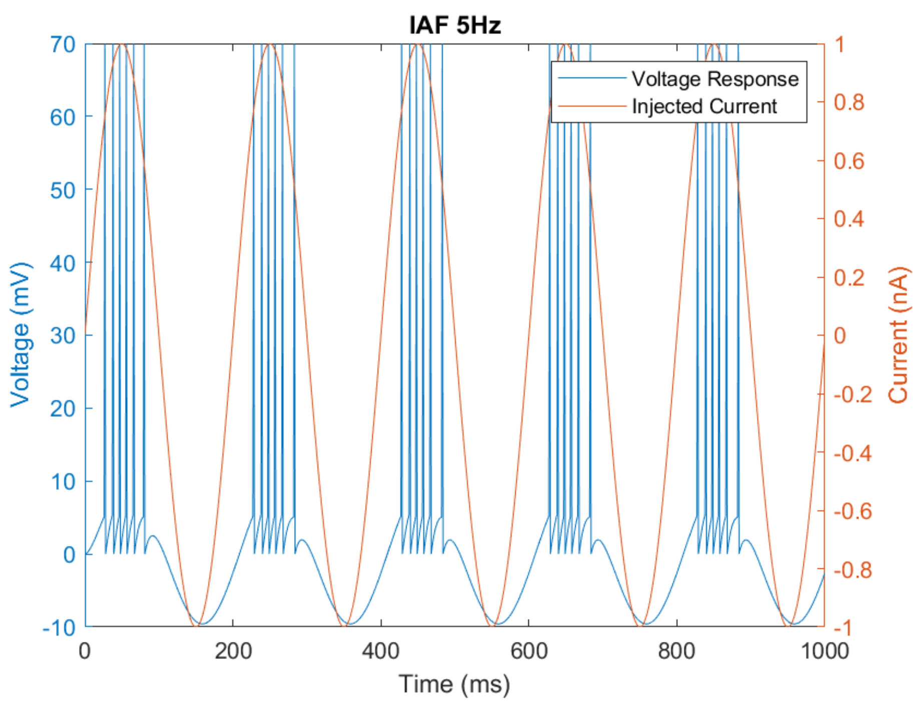
Result

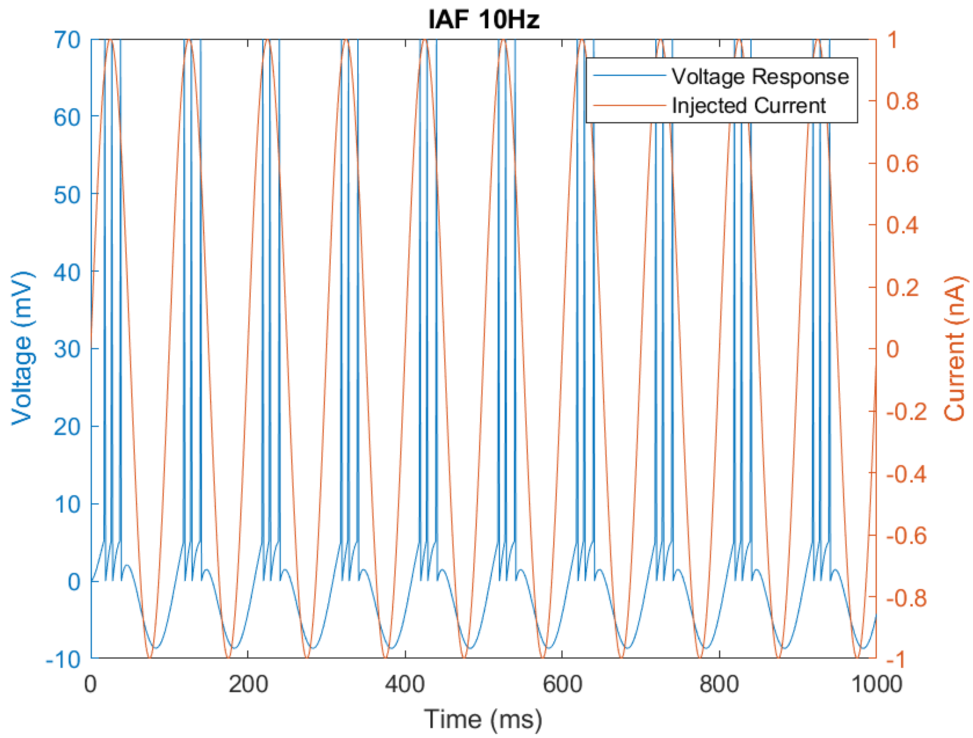
(a) Plots from Integrate and Fire Neuron

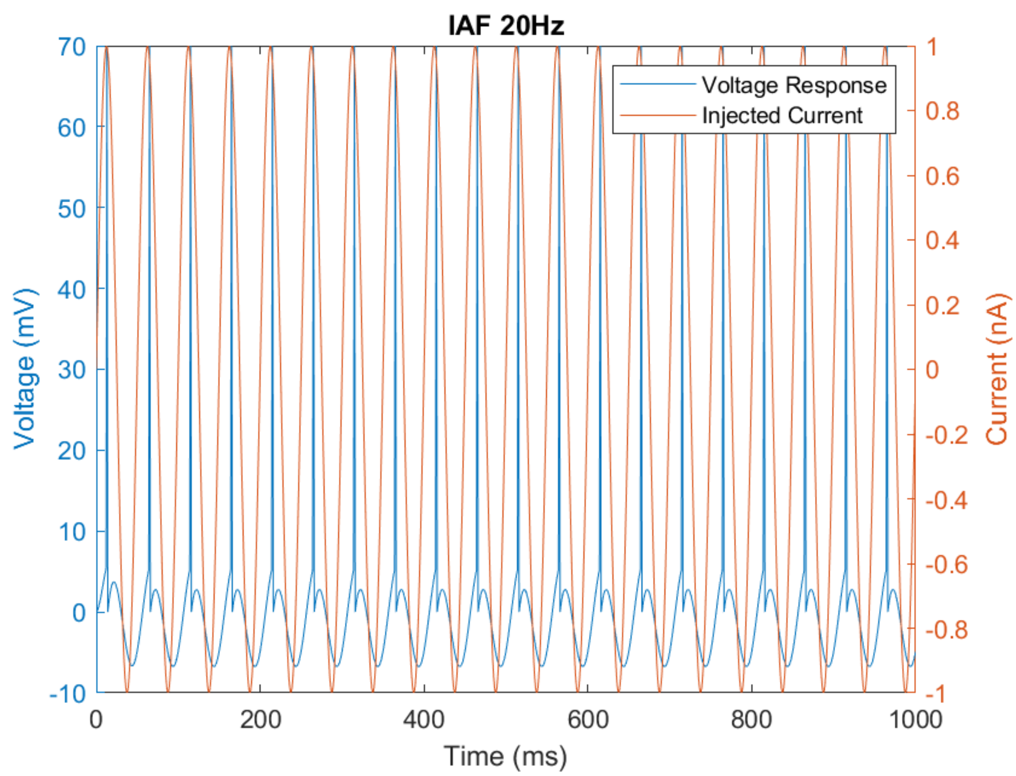


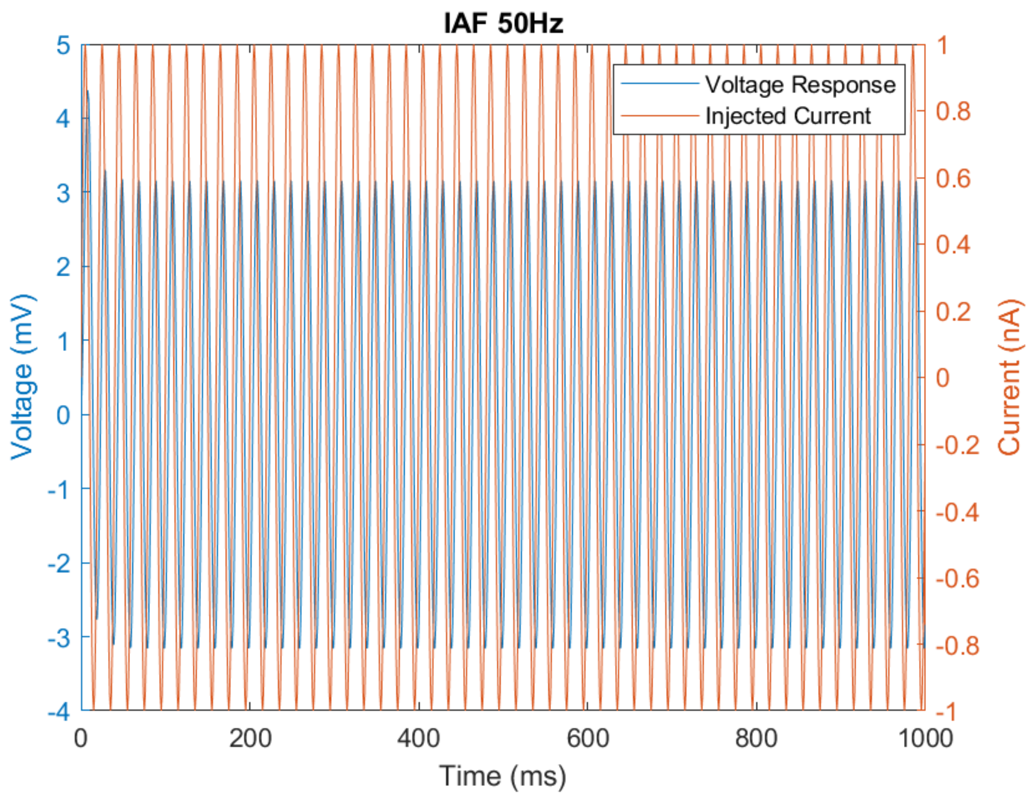


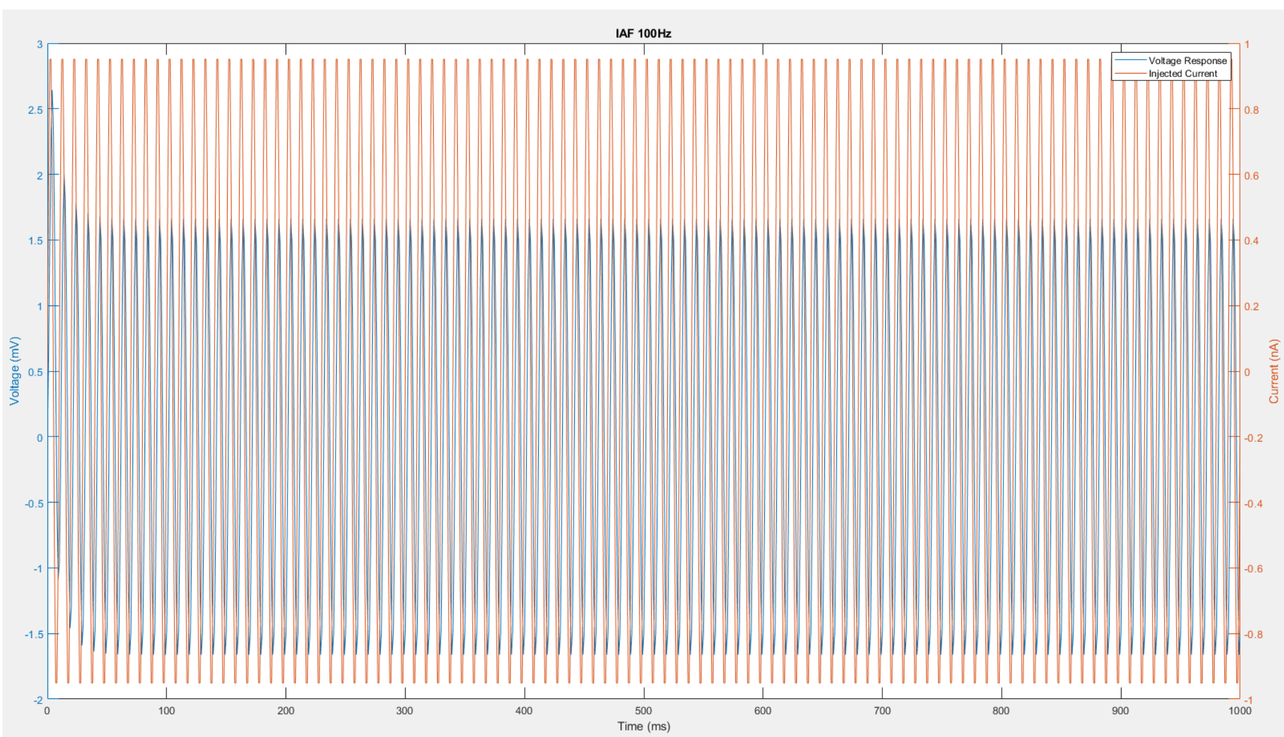

IAF 2Hz
Voltage Response
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
70
60
50
40
30
20
10
0
-10
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
200
400
600
800
1000

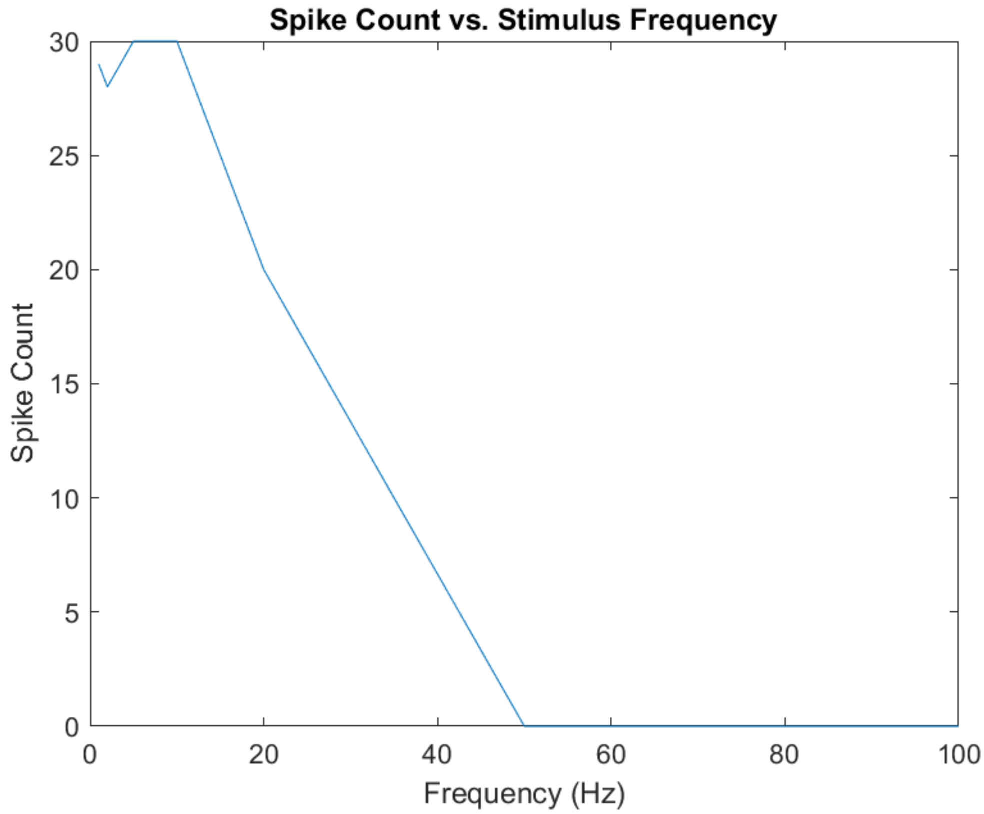

IAF 5Hz
Voltage Response
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
70
60
50
40
30
20
10
0
-10
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
200
400
600
800
1000


IAF 10Hz
Voltage Response
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
70
60
50
40
30
20
10
0
-10
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
200
400
600
800
1000


IAF 20Hz
Voltage Response
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
70
60
50
40
30
20
10
0
-10
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
200
400
600
800
1000

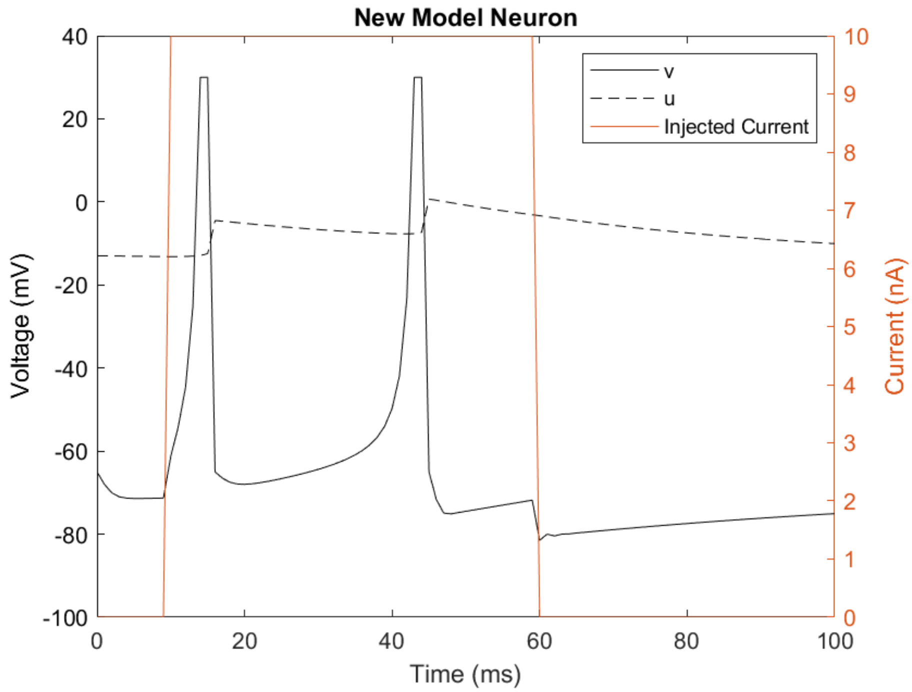

IAF 50Hz
Voltage Response
Injected Current
Voltage (mV)
Current (nA)
Time (ms)

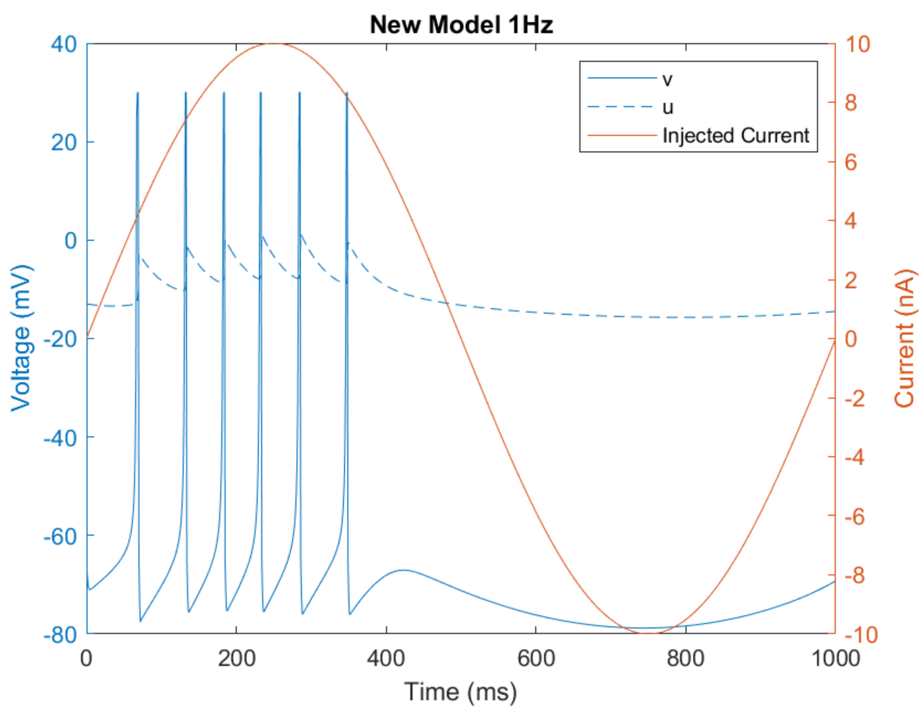

IAF 100Hz
Voltage Response
Injected Current
Voltage (mV)
Current (nA)
Time (ms)

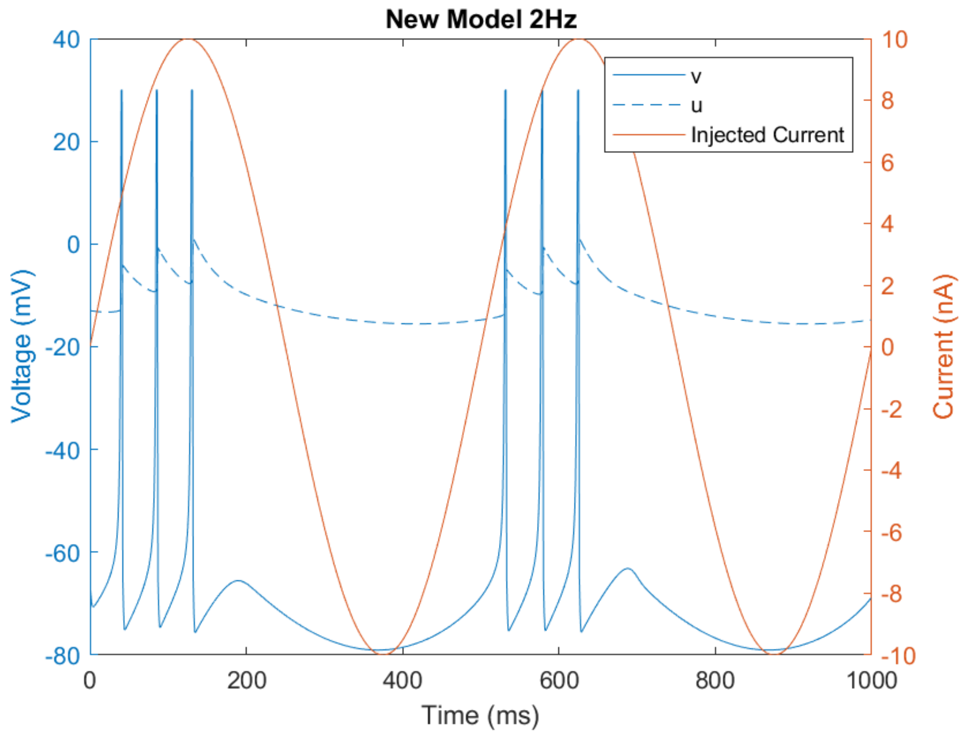


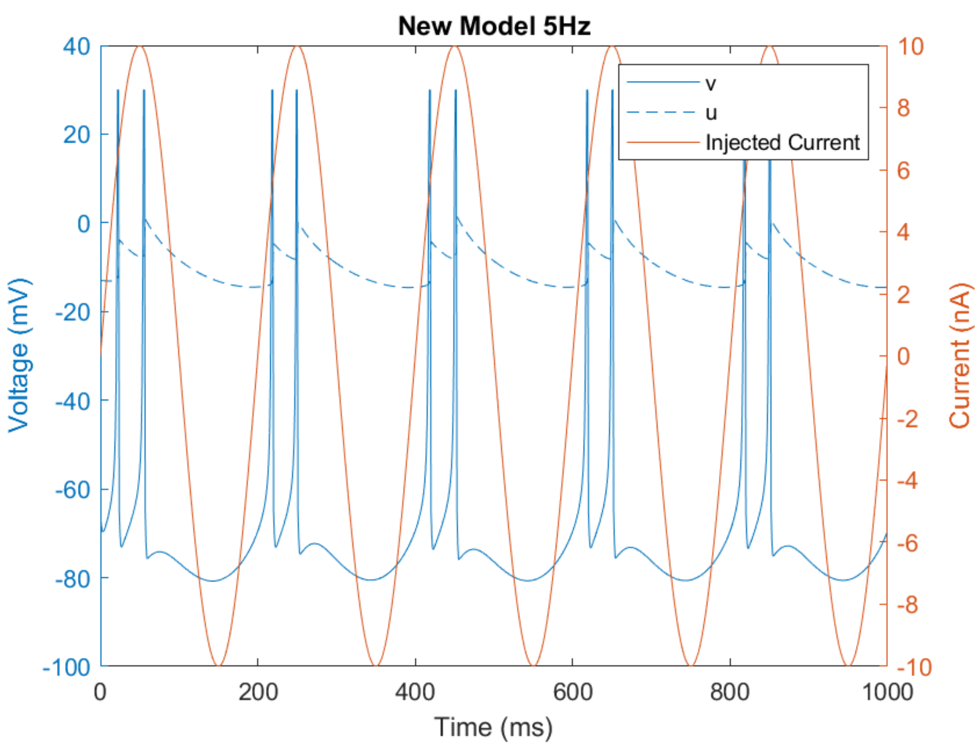
* Count for IAF model

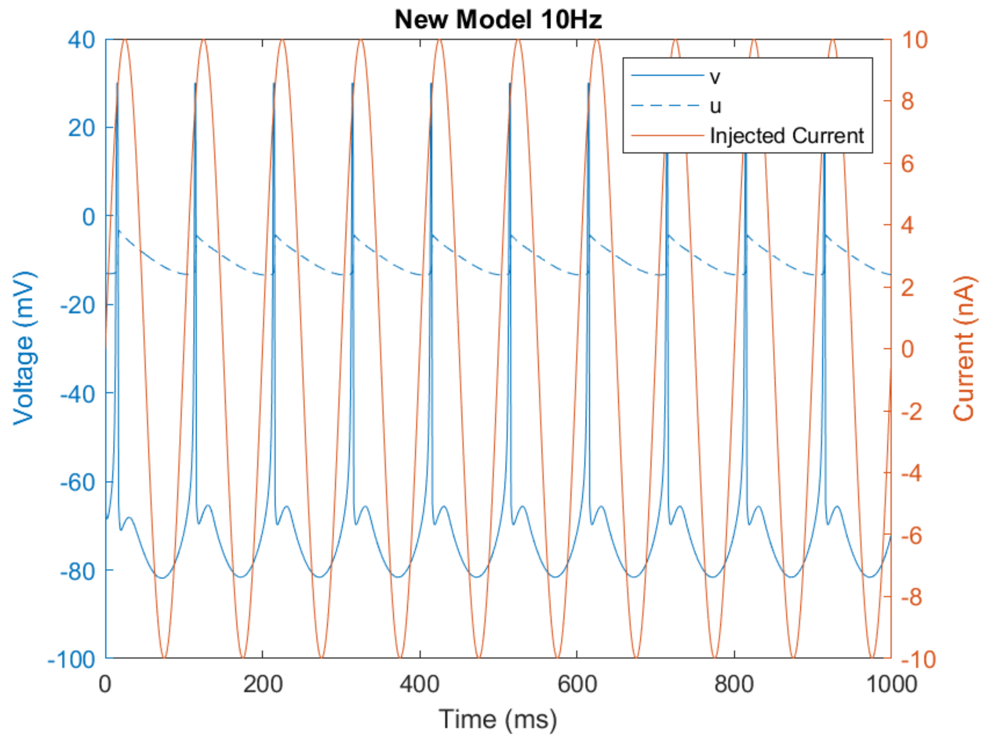
(b) Plots from the New Model

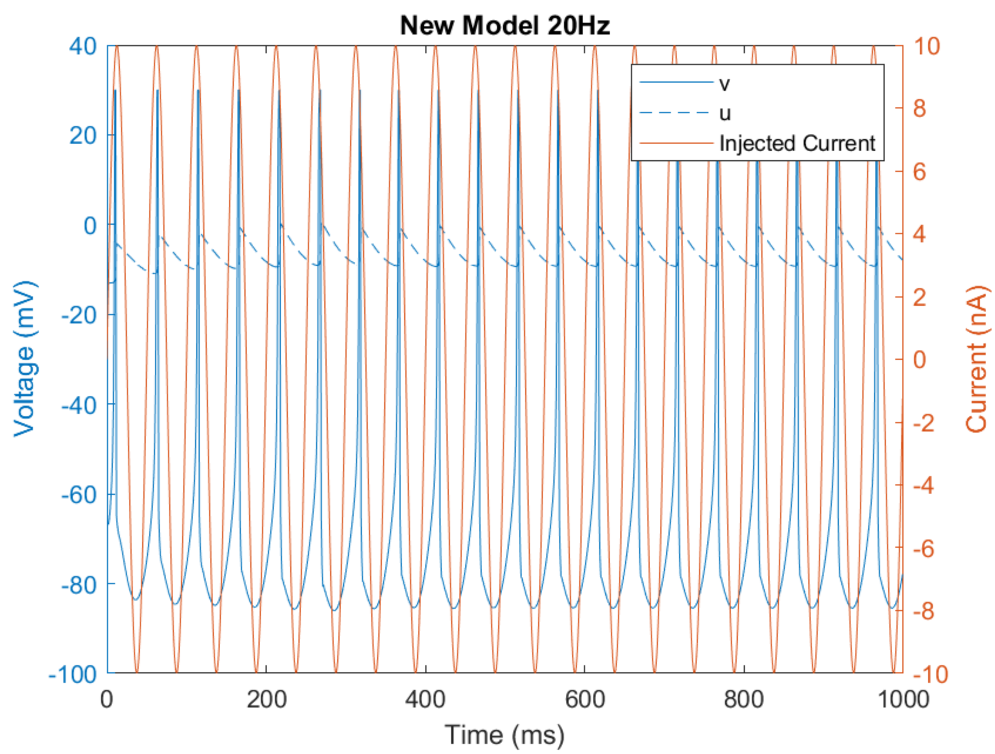


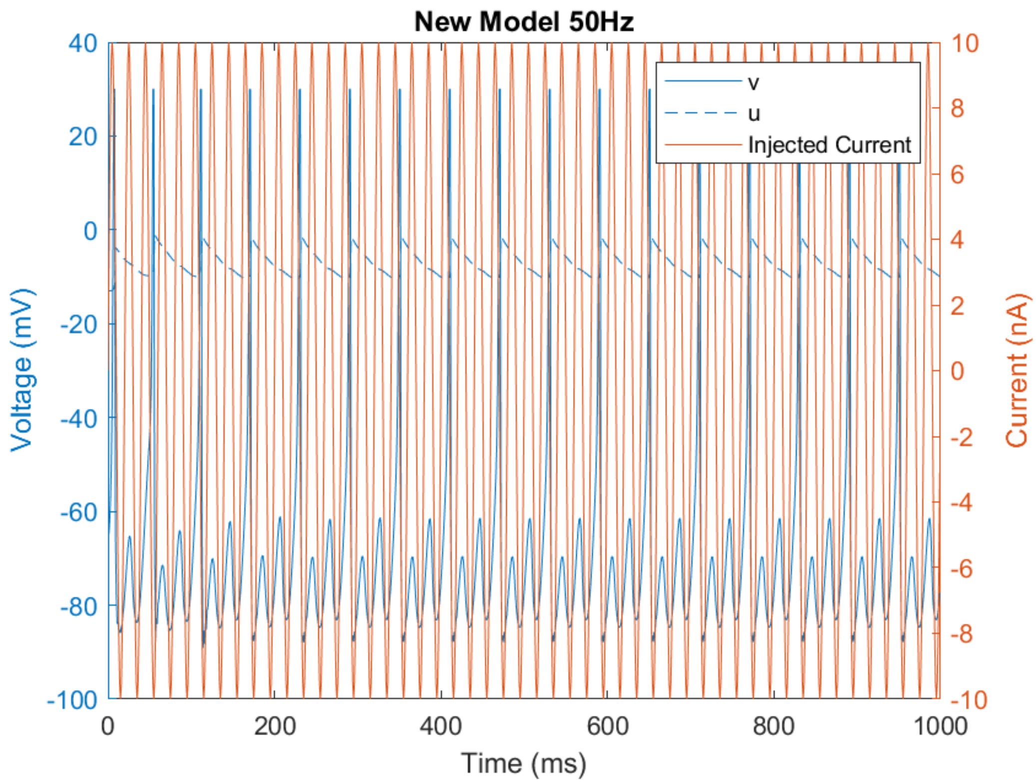


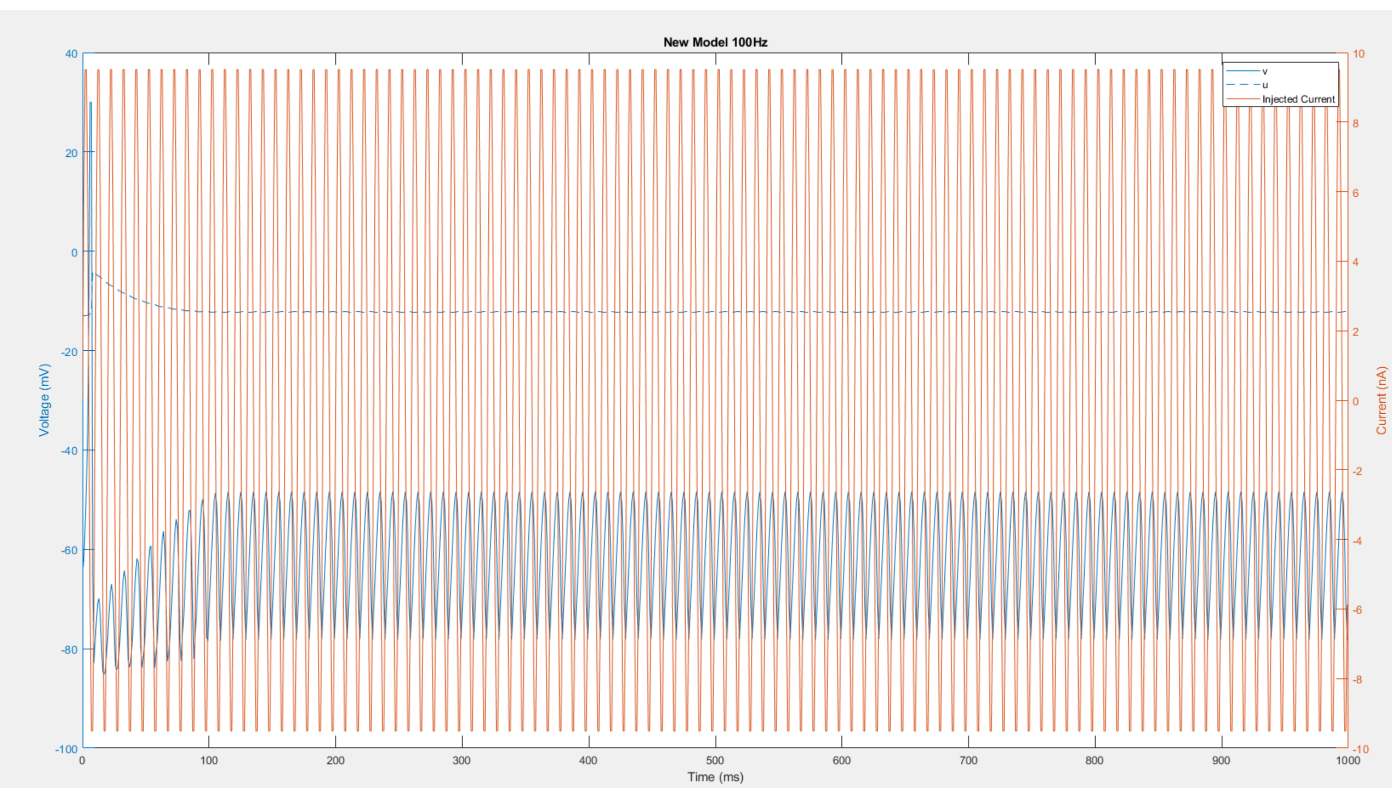

New Model 2Hz
v
u
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
40
20
0
-20
-40
-60
-80
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
200
400
600
800
1000

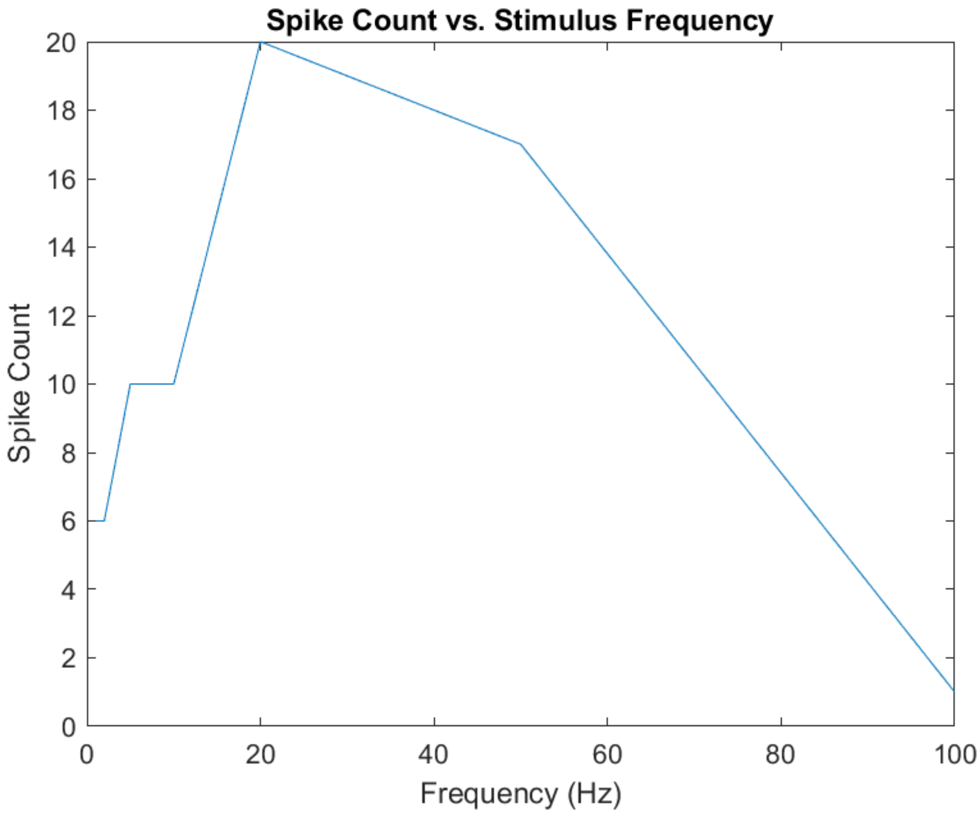

New Model 5Hz
v
u
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
40
20
0
-20
-40
-60
-80
-100
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
200
400
600
800
1000


New Model 10Hz
v
u
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
40
20
0
-20
-40
-60
-80
-100
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
200
400
600
800
1000


New Model 20Hz
v
u
Injected Current
Voltage (mV)
Current (nA)
Time (ms)
40
20
0
-20
-40
-60
-80
-100
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
200
400
600
800
1000

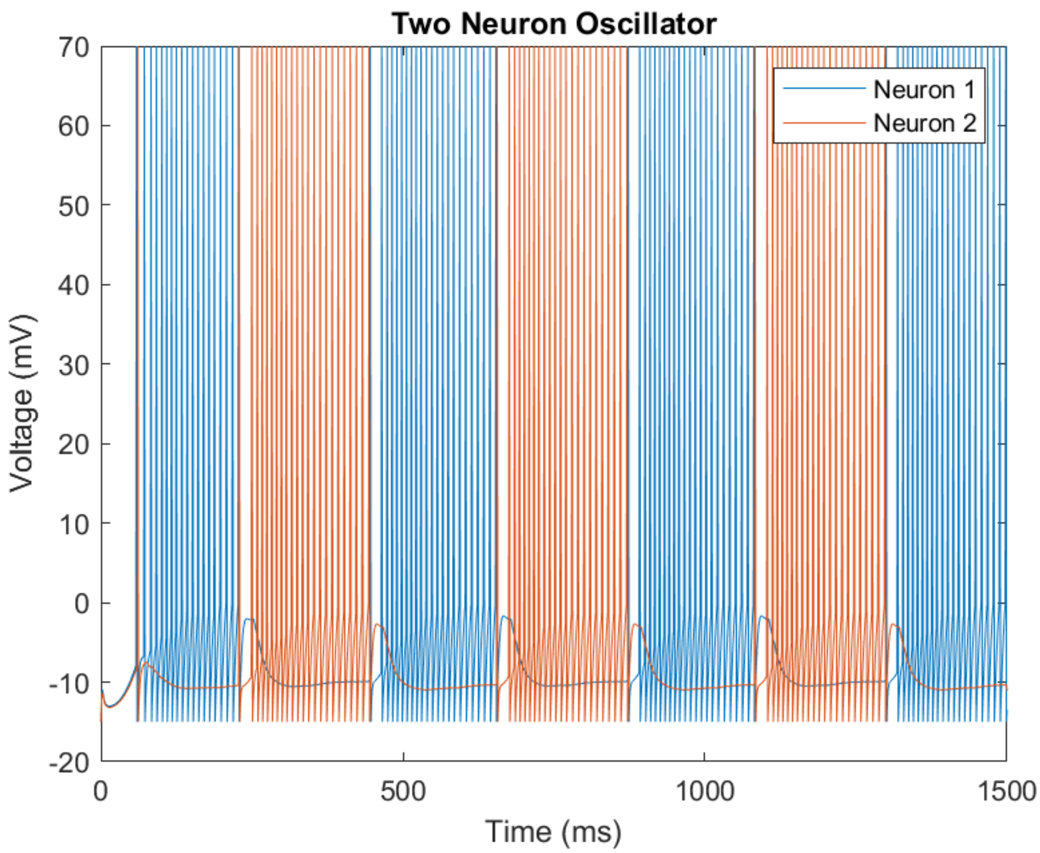

New Model 50Hz
v
u
Injected Current
Voltage (mV)
Current (nA)
Time (ms)


New Model 100Hz
v
u
Injected Current
Voltage (mV)
Current (nA)
Time (ms)



* For the new model

(c) Plot of the Two Neuron Oscillator



Discussion

(a) "The integrate-and-fire neuron model is one of the most widely used models for analyzing the behavior of neural systems. It describes the membrane potential of a neuron in terms of the synaptic inputs and the injected current that it receives. A spike is generated when the membrane potential reaches a threshold, which is resulted from the injection of current in a period of time" (Budelli, 1). This can be observed from both the plots with constant and sinusoidal currents as spikes are produced when current is turned on. Also, observing the IAF plots with sinusoidal currents, when frequency gets higher and higher, less spikes are produced since the voltages under high frequency currents are lower than that of action potential. From the Spike Count vs. Frequency, we know IAF looks like a low pass filter with a cutoff frequency of 10 Hz, approximately, which is as expected.

(b) Observing the plots from the New Model, we know the patterns of v(t) are close to IFA. This is because "the model combines the Hodgkin–Huxley-type dynamics and integrate-and-fire neurons" (Izhikevich, 2). Also, u(t) are from spike activity of the presynaptic unit, which is as expected(Raman). From the Spike Count vs. Frequency, we know the new model looks like a band pass filter with cutoff frequencies of 20 and 50 Hz, approximately.

(c) Observing from the Two Neuron Oscillator plot, we see the symmetry of the model is broken since neuron 1 was injected with slightly more current than neuron 2, which makes neuron 1 curve leads neuron 2 (take less time to begin firing). Also, the neurons do not fire forever because the voltage of one neuron is set to Einh when the voltage of the other neuron is greater or equal to the theta function, which produces the reciprocal inhibition.

%Problem 2 Code

```
%Problem 2
clear all;
% Run the models aksed to implement
integrate_and_fire_neuron();
sinusoidal_IAF();
new_model();
sinusoidal_new_model();
two_neuron_oscillator();

%(a) Implement a simple integrate and fire neuron
(IAF)
function integrate_and_fire_neuron
    %Initialize constants
    R=10;
    C=1;
    Vthr=5;
    Vspk=70;

    step_num = 100; % # of interations
    dt=1; %step size
    T = 0:step_num; %Time array

    % Initiliza array
    V = zeros(1,step_num);
    V(1)=0;

    I_arr=[0];
    skip=0;
    for t=1:(length(T)-1)
        %Get current
        if t >= 10 && t < 60
            I = 1;
        else
            I = 0;
        end
        I_arr=[I_arr I];
        %Get V
```

```
        if skip == 1 %Skip if v was calculated (0)
            skip=0;
            continue;
        end

        if V(t)<Vthr %If V below threshold
            V(t+1) = V(t)+((I-(V(t)/R))/C)*dt;
        else %If V above threshold
            V(t+1) = Vspk;
            V(t+2) = 0;
            skip=1;
        end
    end

    %Plot the model
    a=figure(1);

    title('Integrate and Fire Neuron');
    yyaxis left;
    xlabel('Time (ms)');
    plot(T, V)
    ylabel('Voltage (mV)');

    yyaxis right;
    ylabel('Current (nA)');
    hold on
    plot(T, I_arr);
    legend('Voltage Response', 'Injected Current')
    saveas(a,'p2-1.png');
    hold off;
end

function sinusoidal_IAF
    %Initialize constants
    R=10;
    C=1;
    Vthr=5;
    Vspk=70;
```

```
    freq_arr = [1, 2, 5, 10, 20, 50, 100];
    spike_count_arr = zeros(1, length(freq_arr)); %
For conting spikes at each freq

    step_num = 1000; % # of interations
    dt=1; %Time step size
    T = 0:step_num; %Time array

    % Initiliza array
    V = zeros(1,step_num);
    V(1)=0;

    skip=0;
    for f_idx = 1:(length(freq_arr))
        f = freq_arr(f_idx);
        I_arr=[0];
        for t=1:(length(T)-1)
            %Get sin current
            if t >= 0 && t < 1000
                I = sin(2*pi*f*(t*dt/1000));
            else
                I = 0;
            end

            I_arr=[I_arr I];
            %Get V
            if skip == 1 %Skip if v was calculated
(0)
                skip=0;
                continue;
            end

            if V(t)<Vthr %If V below threshold
                V(t+1) = V(t)+((I-(V(t)/R))/C)*dt;

            else %If V above threshold, spike
                V(t+1) = Vspk;
                V(t+2) = 0;
                % increase spike at freq by 1
```

```
spike_count_arr(f_idx)=spike_count_arr(f_idx)+1;
                skip=1;
            end
        end

        %Plot V vs. t at a freq
        b=figure(f_idx+1);

        title('IAF ' + string(freq_arr(f_idx)) +
'Hz');
        yyaxis left;
        xlabel('Time (ms)');
        plot(T, V)
        ylabel('Voltage (mV)');
        hold on
        yyaxis right;
        plot(T, I_arr);
        ylabel('Current (nA)');
        legend('Voltage Response', 'Injected
Current')
        saveas(b,string(f_idx)+'p2-2.png');
        hold off;
    end
    % Plot spike count vs. stimulus frequency
    c=figure(length(freq_arr)+2);
    plot(freq_arr, spike_count_arr)
    xlabel('Frequency (Hz)');
    ylabel('Spike Count');
    title('Spike Count vs. Stimulus Frequency');
    saveas(c,'p2-3.png');

end

%Additional problems only for BME 572
%(b)
function new_model
    step_num = 100; % # of interations
```

```
dt=1; %Time step size
T = 0:step_num; %Time array

%Define constants
a=0.02;
b=0.2;
c=-65;
d=8;

% Initiliza arrays
v = zeros(1,step_num);
u = zeros(1,step_num);
v(1) =-65;
u(1)=b*v(1);

I_arr=[0];
reset=0;
for t=1:(length(T)-1)
    %Get sin current
    if t >= 10 && t < 60
        I = 10;
    else
        I = 0;
    end
    I_arr=[I_arr I];
    %Reset
    if reset==1
        v(t+1) = c;
        u(t+1) = u(t)+d;
        % increase spike at freq by 1
        reset=0;
    else
        u(t+1) = u(t)+(a*(b*v(t)-u(t)))*dt;
        if v(t)>=30
            v(t)=30;
            v(t+1)=30;
            reset=1;
        else
```

```
                v(t+1) =
v(t)+(0.04*(v(t)^2)+5*v(t)+140-u(t)+I)*dt;
            end
        end
    end

    %Plot V vs. t
    d=figure(9);
    xlabel('Time (ms)');
    yyaxis left;
    plot(T, v)
    ylabel('Voltage (mV)')
    hold on
    plot(T, u);
    yyaxis right;
    ylabel('Current (nA)');
    plot(T, I_arr);

    legend("v", "u", 'Injected Current')
    title('New Model Neuron');
    saveas(d,'p2-4.png');
    hold off;
end

function sinusoidal_new_model
    step_num = 1000; % # of interations
    dt=1; %Time step size
    T = 0:step_num-1; %Time array

    %Define constants
    a=0.02;
    b=0.2;
    c=-65;
    d=8;

    % Initilize arrays
    v = zeros(1,step_num);
    u = zeros(1,step_num);
    v(1) =-65;
```

```
    u(1)=b*v(1);

    freq_arr = [1, 2, 5, 10, 20, 50, 100];
    spike_count_arr = zeros(1, length(freq_arr)); %
For conting spikes at each freq

    reset=0;
    for f_idx = 1:(length(freq_arr))
        f = freq_arr(f_idx);
        I_arr=[0];
        for t=1:(length(T)-1)
            %Get sin current
            if t >= 0 && t < 1000
                I = 10*sin(2*pi*f*(t*dt/1000));
            else
                I = 0;
            end
            I_arr=[I_arr I];

            %Reset
            if reset==1
                v(t+1) = c;
                u(t+1) = u(t)+d;
                % increase spike at freq by 1

                reset=0;

            else
                u(t+1) = u(t)+(a*(b*v(t)-u(t)))*dt;
                if v(t)>=30

spike_count_arr(f_idx)=spike_count_arr(f_idx)+1;

                    v(t)=30;
                    v(t+1)=30;
                    reset=1;
                else
                    v(t+1) =
v(t)+(0.04*(v(t)^2)+5*v(t)+140-u(t)+I)*dt;
```

```
                end
            end
        end

        %Plot V vs. t at a freq
        e=figure(length(freq_arr)+f_idx+3);
        xlabel('Time (ms)');
        yyaxis left;
        plot(T, v)
        ylabel('Voltage (mV)');
        hold on
        plot(T, u);
        yyaxis right;
        plot(T, I_arr);
        ylabel('Current (nA)');
        legend("v", "u", 'Injected Current')
        title('New Model ' + string(freq_arr(f_idx))
+ 'Hz');
        hold off;
        saveas(e,string(f_idx)+'p2-5.png');
    end

    % Plot spike count vs. stimulus frequency
    f=figure(2*length(freq_arr)+4);
    plot(freq_arr, spike_count_arr)
    xlabel('Frequency (Hz)');
    ylabel('Spike Count');
    title('Spike Count vs. Stimulus Frequency');
    saveas(f,string(f_idx)+'p2-6.png');
end


%(c)Construct a two-neuron oscillator using
reciprocal inhibition
function two_neuron_oscillator
    step_num = 1500; % # of interations
    dt=1; %Time step size
    T = 0:step_num; %Time array
```

```
%Define constants
C=1;
R=10;
Vrest=0;
Vspk=70;
tauthre=50;
Einh=-15;
tausyn=15;
gpeak=0.1;

%Initialize an array of structs for 2 neurons
neuron_arr=[[] []];
for i=1:2
    neuron_arr(i).v = zeros(1,step_num);
    neuron_arr(i).v(1) = Einh;
    neuron_arr(i).theta = zeros(1,step_num);
    neuron_arr(i).theta(1) = Vrest;
    neuron_arr(i).z = zeros(1,step_num);
    neuron_arr(i).z(1) = 0.2;
    neuron_arr(i).g = zeros(1,step_num);
    neuron_arr(i).g(1) = 0.2;
end

reset_arr = [0 0];
for t=1:(length(T)-1)
    %Get currents
    if t >= 0 && t < 1500
        I1=1.1;
        I2=0.9;
    else
        I1=0;
        I2=0;
    end

    for i=1:(length(neuron_arr))
        j=2;
        I=I1;
        if i == 2
```

```
                I=I2;
                j = 1;
            end

            % Euler's method
            neuron_arr(i).v(t+1) = neuron_arr(i).v(t) + (((-neuron_arr(i).v(t)/R) - neuron_arr(i).g(t)*(neuron_arr(i).v(t) - Einh) + I)/C)*dt;
            neuron_arr(i).theta(t+1) = neuron_arr(i).theta(t) + ((-neuron_arr(i).theta(t) + neuron_arr(i).v(t))/tauthre)*dt;
            neuron_arr(i).z(t+1) = neuron_arr(i).z(t) + (-neuron_arr(i).z(t)/tausyn) + (gpeak/(tausyn/exp(1)))*(neuron_arr(j).v(t)==Vspk)*dt;
            neuron_arr(i).g(t+1) = neuron_arr(i).g(t) + ((-neuron_arr(i).g(t)/tausyn) + neuron_arr(i).z(t))*dt;

            if reset_arr(i) == 1
                neuron_arr(i).v(t+1) = Einh;
                reset_arr(i)=0;
            end

            if neuron_arr(i).v(t+1) >= neuron_arr(i).theta(t+1) %when fires
                neuron_arr(i).v(t+1) = Vspk;
                reset_arr(i) = 1;
            end
        end
    end

    %Plot V vs. t
    g=figure(19);
    plot(T, neuron_arr(1).v)
    xlabel('Time (ms)');
    ylabel('Voltage (mV)');
```

```
    title('Two Neuron Oscillator');
    hold on;
    plot(T, neuron_arr(2).v);
    legend("Neuron 1", "Neuron 2")
    hold off;
    saveas(g, 'p2-7.png');
end
```

Problem 3

Methods (Raman):

(a) Implement a linear-nonlinear model (LN model) of a neuron.

1. Process 'Spikes.txt' and 'Stimulus.txt' and load the variables in MATLAB. LN model is built using the first four trials, and the fifth trial was used for testing.

2. Find S the stimulus matrix during the previous 2 s (binned in 100 ms time bins to check overlap, first row in S: 0-2s; second row in S: 0.1-2.1s…).

3. Find R the firing rate response matrix in the current time bin (binned in 100ms time bin to count spikes).

4. Implement W the linear filter using the equation below.

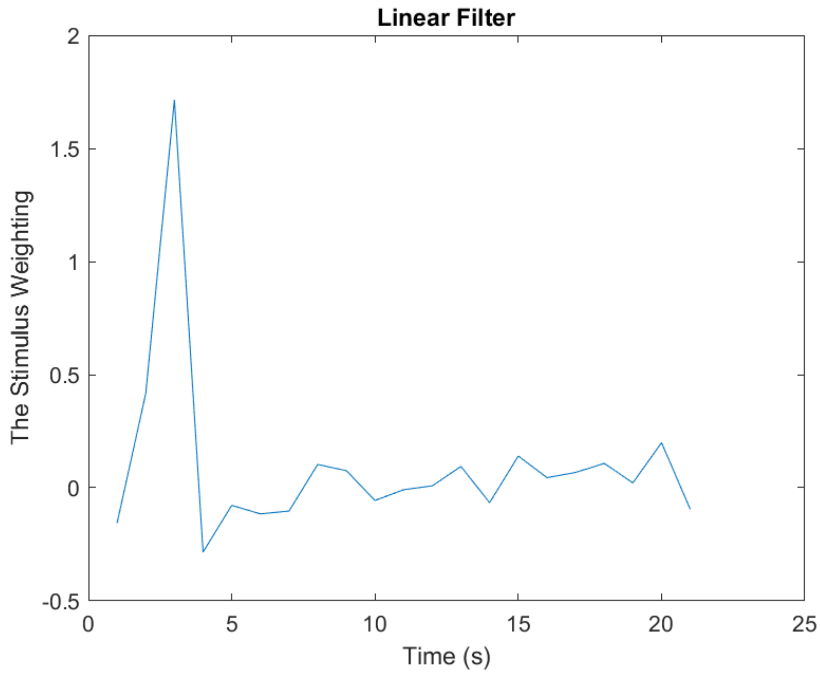
$$W = (S^T S)^{-1} R$$

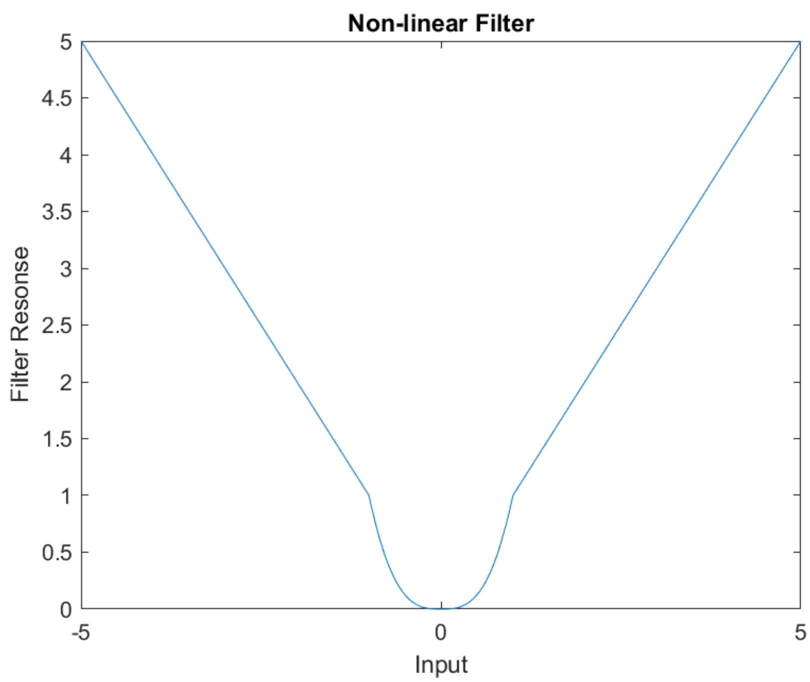
5. Implement a nonlinear filter that takes absolute value and remove noises in a MATLAB function.

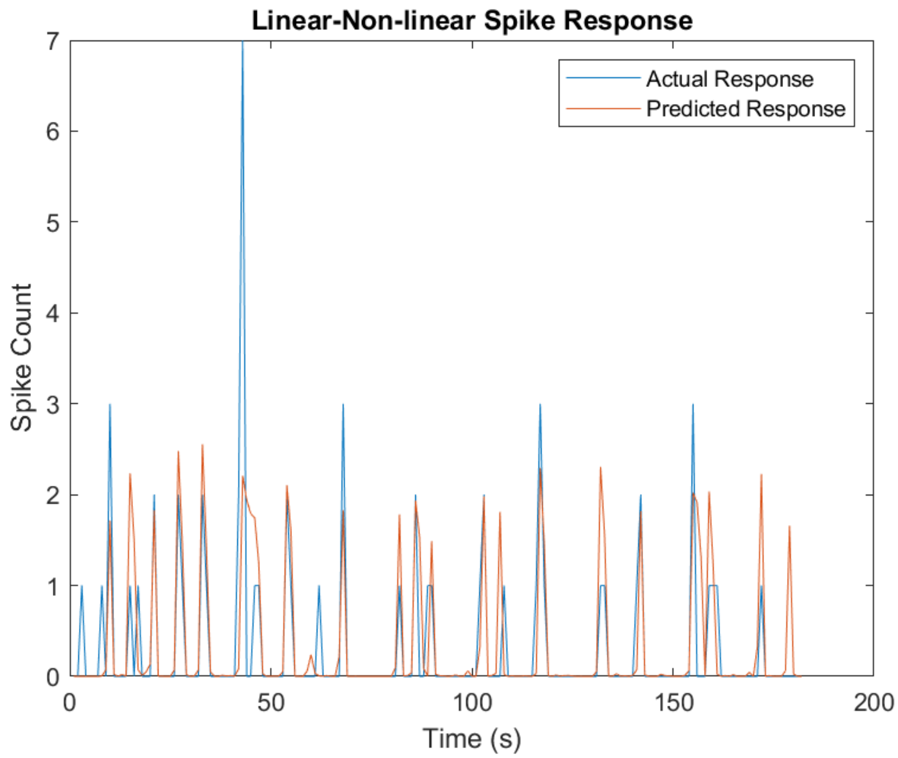
6. Plot linear filter, non-linear filter, and LN spike response vs. time in seconds.

The figures and code are shown in Result and Problem 3 Code sections respectively.

Result:


Linear Filter
The Stimulus Weighting
Time (s)
2
1.5
1
0.5
0
-0.5
0
5
10
15
20
25


Non-linear Filter
Filter Resonse
Input
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-5
0
5



Linear-Non-linear Spike Response
Actual Response
Predicted Response
Spike Count
Time (s)
0
1
2
3
4
5
6
7
50
100
150
200

Discussion and Limitations:

(a) As shown in the Linear Filter plot, the linear filter simulates the behavior of the action potential (at 0 to 5). As shown in the Non-linear Filter, the filter is a combination of an absolute value function and a square function (from t= -1 to 1) that removes the noises. As shown in the Linear-non-linear Response plot, a reasonable prediction match is generated. However, the perdition is not very precise as the predicted spike counts are sometimes different than the actual, which might be resulted from insufficient training set. To improve the prediction in the future, we might consider using more data to train the model.

%Problem 3 Code

```
#Python code for processing the file
with open('stimulus.txt', 'r') as f:
    stimulus = []
    for l in f:
        pair = []
        for s in l.rstrip().split('\t'):
            pair.append(float(s)/1000)
        stimulus.append(pair)

with open('spikes.txt', 'r') as f:
    spikes = []
    for l in f:
        spikes.append(float(l.rstrip()))
print(stimulus)

trial_count = 1
spk=[]
start=0
end=0
s_idx=0

for s in spikes:
    s_idx+=1
    if s >= trial_count * 20:
        start = end
        end = s_idx
        spk.append([s2 - (trial_count-1) * 20 for s2
in spikes[start:end]])
        trial_count += 1
    if trial_count == 5:
        spk.append([s2 - (trial_count-1) * 20 for s2
in spikes[end:len(spikes)]])
        break

spike_matrix = []
for t in spk:
    spike_matrix.append(t + [0]*(126-len(t)))
```

```
print(spike_matrix)

%MATLAB code for matrix manipulation
clear all;
% Load var generated by python
spikes=[0.129200000000000,0.582000000000000,0.661333
333333333,1.00626666666667,1.02606666666667,1.353600
00000000,1.77693333333333,1.81120000000000,1.9263333
3333333,2.35786666666667,2.78120000000000,2.78526666
666667,2.79253333333333,2.83573333333333,2.852133333
33333,2.87073333333333,2.88686666666667,3.3682000000
0000,3.37573333333333,3.37973333333333,3.39780000000
000,3.90273333333333,3.92420000000000,3.928866666666
67,3.93180000000000,3.96800000000000,4.5224000000000
0,4.53073333333333,4.54240000000000,4.54613333333333
,4.56606666666667,5.04860000000000,5.16746666666667,
5.17440000000000,5.19286666666667,5.19673333333333,5
.20173333333333,5.27273333333333,5.94013333333333,6.
10286666666667,6.11186666666667,6.12753333333333,6.1
3886666666667,6.19233333333333,6.24713333333333,6.27
366666666667,6.38273333333333,6.39046666666667,6.416
86666666667,6.45600000000000,6.51220000000000,6.5470
0000000000,7.27273333333333,7.27653333333333,7.31786
666666667,7.32786666666667,7.33560000000000,8.580066
66666667,8.58420000000000,8.60700000000000,8.6706000
0000000,9.99853333333333,10.0020666666667,10.0080666
666667,10.0159333333333,10.0470000000000,10.32540000
00000,10.4550000000000,10.4700000000000,10.493066666
6667,10.4994666666667,10.5166000000000,10.5429333333
333,10.5539333333333,10.5776000000000,10.81113333333
33,10.8234000000000,10.8490000000000,10.886133333333
3,10.9134666666667,12.1143333333333,12.1202666666667
,12.1246666666667,12.1496666666667,12.1698000000000,
12.5571333333333,12.5704666666667,12.5762666666667,1
2.6574666666667,13.5414666666667,13.5490666666667,13
.5837333333333,13.5963333333333,13.6255333333333,15.
0019333333333,15.0077333333333,15.0205333333333,15.0
268000000000,15.0414666666667,15.0538666666667,15.10
```

35333333333,16.0228666666667,16.0291333333333,16.032
2666666667,16.0371333333333,16.1108666666667,17.3598
000000000,17.3728666666667,17.3820000000000,17.39100
00000000,17.4372666666667,17.5041333333333,17.758733
3333333,17.7836000000000,17.7915333333333,17.7958000
000000,17.8824000000000,19.0038666666667,19.01206666
66667,19.0165333333333,19.0198000000000,19.065333333
3333,19.0793333333333,19.7089333333333,19.7422000000
000,20.3374666666667;0.501533333333299,0.65060000000
0001,0.957733333333302,1.32013333333330,1.4388666666
6670,1.86900000000000,2.52446666666670,2.78726666666
670,2.79446666666670,2.80773333333330,3.258133333333
30,3.35633333333330,3.80960000000000,3.8879333333333
0,3.89320000000000,3.90040000000000,3.90746666666670
,4.51493333333330,4.51973333333330,4.52513333333330,
4.61500000000000,5.15660000000000,5.16180000000000,5
.16820000000000,5.17346666666670,5.19893333333330,5.
23753333333330,5.26100000000000,6.08873333333330,6.0
9486666666670,6.11706666666670,6.13946666666670,6.15
260000000000,6.24720000000000,6.36926666666670,6.459
33333333330,7.24860000000000,7.25353333333330,7.2802
0000000000,7.30620000000000,7.35306666666670,8.56660
000000000,8.57413333333330,8.59473333333330,8.637866
66666670,9.72513333333330,10.0222000000000,10.468600
0000000,10.4902666666667,10.5106000000000,10.5316666
666667,10.5668000000000,10.8012666666667,10.81053333
33333,11.8282666666667,12.0979333333333,12.106066666
6667,12.1210666666667,12.1258000000000,12.1296000000
000,12.5530666666667,12.5622666666667,13.54273333333
33,13.5581333333333,13.5623333333333,13.566866666666
7,13.5820666666667,13.5959333333333,13.6009333333333
,15.0035333333333,15.0435333333333,15.0560666666667,
15.1118666666667,15.1456000000000,16.0215333333333,1
6.0276000000000,16.0334000000000,17.3617333333333,17
.3736666666667,17.4246000000000,17.7869333333333,17.
7942000000000,17.8102000000000,18.9914000000000,19.0
066000000000,19.0296000000000,19.0980000000000,19.74
62666666667,19.7645333333333,20.5322000000000,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0;1.66700000000000,1.75446666666670,1.89
60000000000,2.16620000000000,2.82106666666670,2.890
13333333330,2.90840000000000,3.47373333333330,3.9324
0000000000,3.93686666666670,4.06273333333330,4.52380
000000000,4.53080000000000,4.54040000000000,4.548133
33333330,4.57700000000000,5.18833333333330,5.1950000
0000000,5.21953333333330,5.22520000000000,5.26566666
666670,5.77406666666670,6.11233333333330,6.117400000
00000,6.39553333333330,6.53553333333330,7.3034000000
0000,7.96180000000000,8.59566666666670,10.0340666666
667,10.0772000000000,10.4836000000000,10.50473333333
33,10.5244000000000,12.1250666666667,12.140133333333
3,12.1564666666667,12.1731333333333,12.5992666666667
,12.6319333333333,13.5502000000000,13.5639333333333,
13.5732000000000,13.6056666666667,13.6370000000000,1
3.6550000000000,15.0085333333333,15.0148000000000,15
.0554000000000,15.0704666666667,15.0913333333333,15.
1051333333333,16.0544666666667,16.0799333333333,17.7
992666666667,19.0472666666667,19.0658666666667,19.08
40666666667,19.1080666666667,19.1174666666667,19.130
8000000000,19.7194000000000,19.7706000000000,21.5884
666666667,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;2.31520000000000,2.787
53333333330,2.83166666666670,4.46720000000001,4.5165
3333333330,4.54086666666670,4.59053333333330,5.17473
333333329,5.17786666666670,5.20646666666670,6.100600
00000000,6.11620000000001,6.19940000000000,6.2180666
6666670,6.36953333333329,6.41113333333330,8.51540000
000000,8.54233333333330,8.59253333333329,8.648266666
66670,9.94360000000000,9.98300000000000,9.9996666666
6670,10.0304000000000,10.0994000000000,10.5030000000
000,10.5952666666667,10.8004666666667,12.09700000000
00,13.0512666666667,13.2520666666667,13.574466666666
7,13.5849333333333,13.6566666666667,15.1266666666667
,16.0411333333333,16.0793333333333,16.9275333333333,
17.1841333333333,17.3448666666667,17.3582000000000,1

```
9.7018000000000,20.9705333333333,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;2.1995333
3333331,2.6415333333330,2.8018000000000,2.80646666
666669,2.8141333333330,3.3820666666670,3.587200000
00000,3.9026000000001,3.9229333333330,4.5274000000
0000,4.5314666666670,4.6470666666670,5.18473333333
330,5.1914666666670,5.2157333333330,6.087266666666
69,6.0934666666670,6.1002000000000,6.1128666666667
0,6.1177333333331,6.1306666666670,6.1561333333330
,6.1701333333330,6.1885999999999,6.4614666666670,
6.5692000000000,7.2586666666670,7.2680666666670,7
.3662666666670,8.0952666666670,8.6033333333340,8.
6560000000001,8.6905333333331,10.0180000000000,10.
4714666666667,10.4888000000000,10.7950666666667,10.8
007333333333,12.0237333333333,12.1114666666667,12.12
53333333333,12.6418666666667,13.4626666666667,13.542
8000000000,13.5566000000000,13.5934666666667,13.6324
000000000,15.0462666666667,15.1038000000000,15.91013
3333333,16.0206666666667,16.0454000000000,17.349400
0000000,17.3798000000000,17.3926666666667,17.7865333
333333,17.8490000000000,17.9537333333333,19.00933333
33333,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

stimulus=[2.7157072000000,2.7595368930000;3.291774
32800000,3.3026632100000;3.8316785630000,3.8794657
6600000;4.4427883050000,4.5206185710000;5.09332367
600000,5.1747721360000;6.0214589620000,6.114255111
00000;6.2891042250000,6.3751291600000;6.4424845410
0000,6.4855823000000;7.1845216770000,7.25039379700
000;8.5023359680000,8.5781509220000;9.910267094000
00,9.9377174800000;10.3804959300000,10.471929040000
0;10.7049164000000,10.7661449200000;12.0269137600000
,12.0937748600000;12.4599461900000,12.4910433600000;
13.4641099400000,13.5235003300000;14.9177523300000,1
```

```
5.0115948300000;15.9486560300000,15.9888238000000;17
.2671894700000,17.3361872600000;17.3810876900000,17.
4263590700000;17.6729015400000,17.7393598700000;18.9
191597000000,18.9920769100000;19.6213921700000,19.66
71387200000];

%Find S
S_bin = get_bin(0.0, 0.1, 0.1, 21);
sti_bool=[];
for b=S_bin.'
    sti_check = 0;
    for st = stimulus.'
        if b(2) >= st(1) && b(1) <= st(2)
            sti_check = 1;
            break;
        end
    end

    if sti_check == 1
        sti_bool=[sti_bool 1];
    else
        sti_bool=[sti_bool 0];
    end
end

S=[];
for t = get_bin(0, 20, 1, 201).'
    S=[S; sti_bool(t(1)+1:t(2)+1)];
end

% Get R
R=[];
[m, n] = size(spikes);

for t = get_bin(1.9, 2.0, 0.1, 20.1).'
    r_count=zeros(1,m);
    s_idx=1;
    for trial=spikes.'
        for s=trial.'
```

```
            if s == 0
                break;
            end

            if t(2) >= s && t(1) <= s
                r_count(s_idx) = r_count(s_idx)+1;
            end
        end
        s_idx=s_idx+1;
    end
    R=[R; r_count];
end

%Find W
W1 = (pinv(S))*R(:, 1);
W2 = (pinv(S))*R(:, 2);
W3 = (pinv(S))*R(:, 3);
W4 = (pinv(S))*R(:, 4);
W=(W1+W2+W3+W4)./4;
prediction = NL_filter(S*W);

a=figure(1);
plot(1:21, flip(W));
xlabel('Lag (s)');
ylabel('Weighting of the Stimulus ');
title('Linear Filter');
saveas(a, 'p3-1.png');

b=figure(2);
fplot(@(x) NL_filter(x), [1, 21]);
xlabel('Time (s)');
ylabel('Filter Resonse');
title('Non-linear Filter');
saveas(b, 'p3-2.png');

c=figure(3);
plot(1:182, R(:, 5));
xlabel('Time (s)');
ylabel('Spike Count');
```

```
title('Linear-Non-linear Spike Response');
hold on;
plot(1:182, prediction);
legend("Actual Response", "Predicted Response")
hold off;
saveas(c, 'p3-3.png');

% Generate the bin
function bin_arr = get_bin(init_start, init_end,
incre, stop)
    bin_arr=[];
    while 1
        if init_end >= (stop + incre)
            break;
        end
        bin_arr = [bin_arr; [init_start init_end]];
        init_start=init_start+incre;
        init_end=init_end+incre;
    end
end


function arr_new=NL_filter(arr)
    for i=1:length(arr)
        if arr(i) < 0
            arr(i) = (-arr(i));
        end

        if arr(i) < 1
            arr(i) = arr(i)^3;
        end

    end
    arr_new=arr;
end
```

Problem 4

Methods (Raman):

(a) Create a spike train using inhomogeneous Poisson process.

1. Start with a homogeneous Poisson process with a λ max = max(λ (t)), where λ (t) is the predicted firing rate of the neuron (prediction generated from the LN model).

2. Thin the homogeneous Poisson process by accepting a spike by generating an uniform random number U[0,1] (rand in matlab) and if $U[0,1] \leqslant \lambda(t)/\lambda max$. 1000 spike trains were produced.

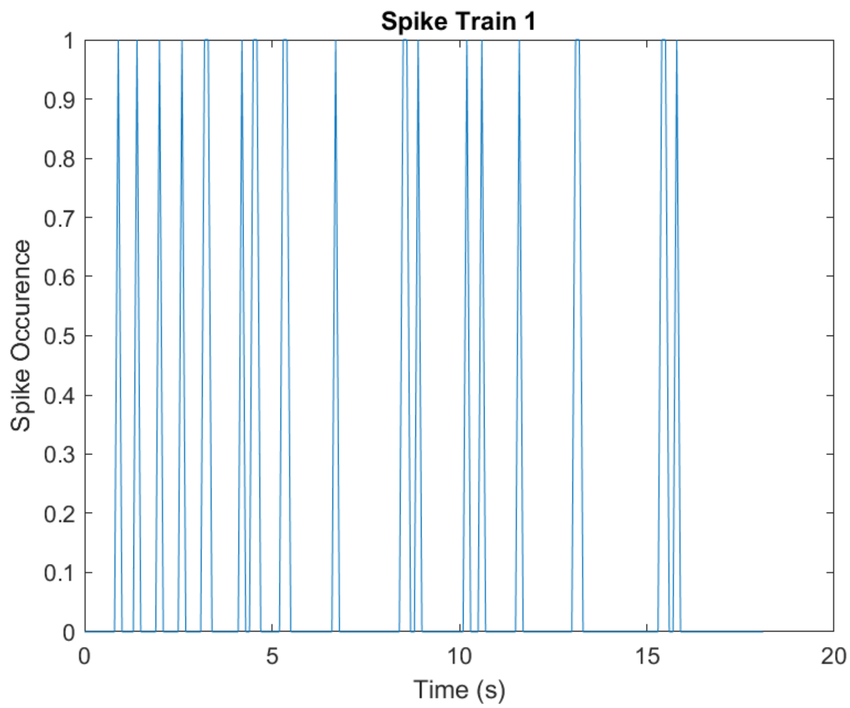
3. Find ISI distributions by counting the time between spike and spike.

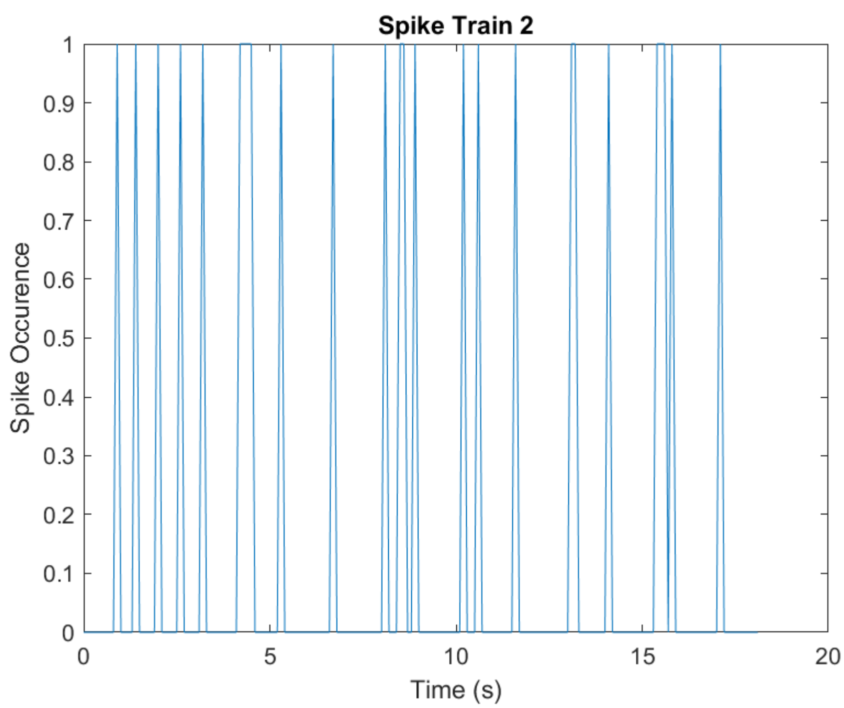
4. Plot the first 4 spike trains and ISI distributions.

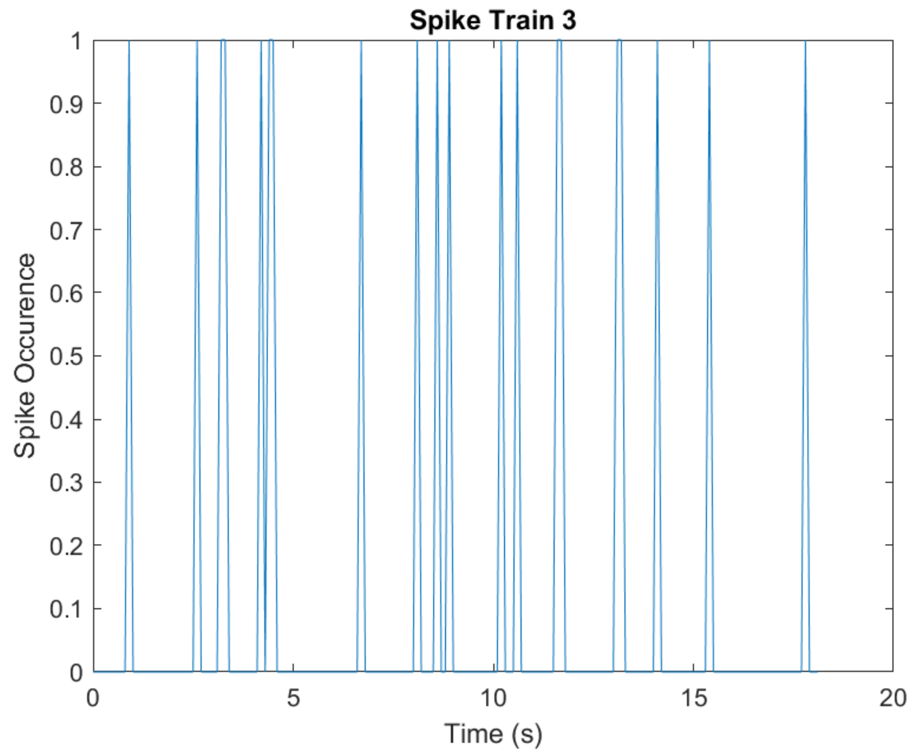
5. Compute the Fano Factor, and Coefficient of Variation of the ISIs by var(spike_trains)/mean(spike_trains) and std(ISI_ distributions)/mean(ISI_ distributions) respectively.

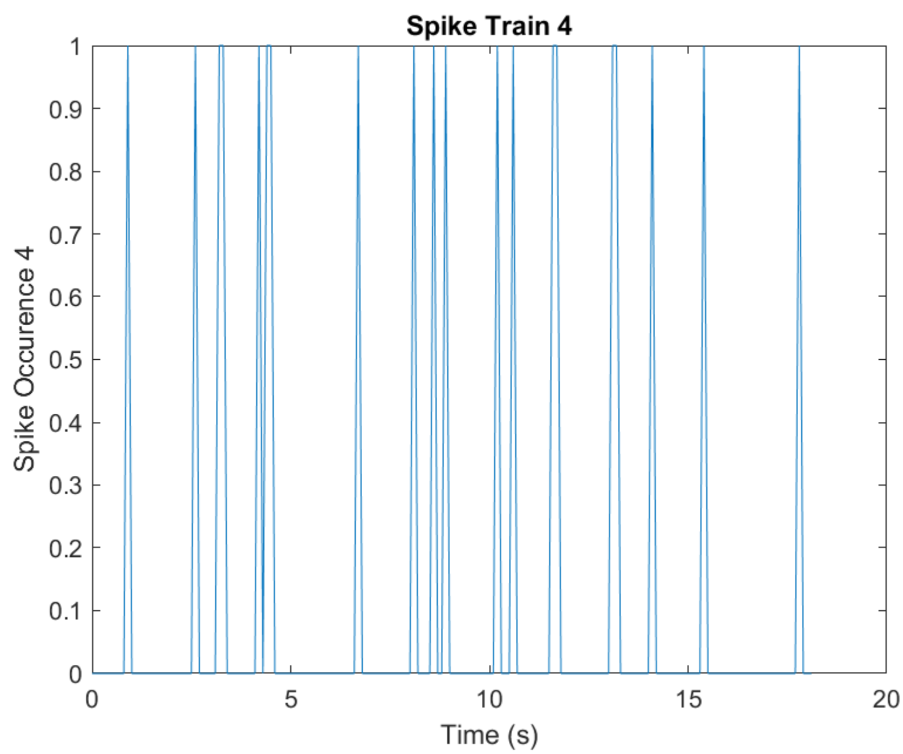
The figures, code are shown in Result and Problem 4 Code sections respectively.

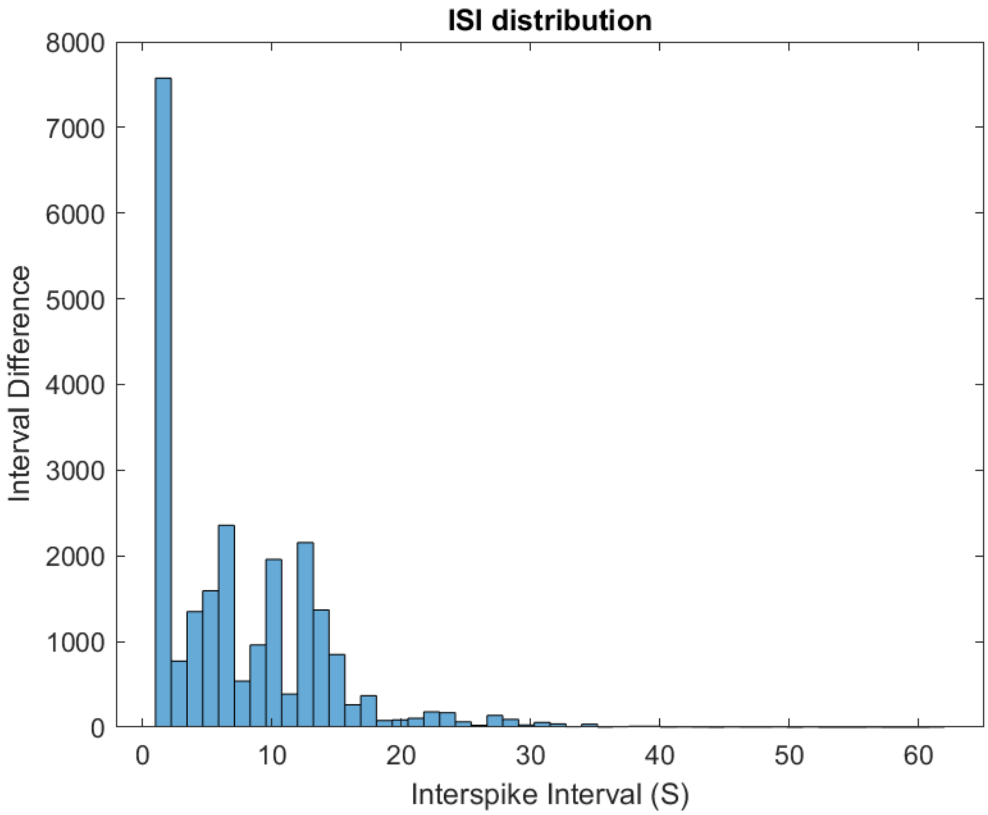
Result:


Spike Train 1
Spike Occurence
Time (s)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20


Spike Train 2
Spike Occurence
Time (s)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20


Spike Train 3
Spike Occurence
Time (s)
0
5
10
15
20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Spike Train 4
Spike Occurence 4
Time (s)
0
5
10
15
20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



The Fano Factor is: 0.86259

Coefficient of Variation for ISI is: 0.87069

Discussion:

As shown in the plots of Spike Trains 1-4, the spike trains seem reasonable as they basically follow spike rate prediction from problem 3. As shown in the ISI distribution plot, the ISI is similar to the Poisson distribution (follow exponential decrease), which is also supported by the Fano factor = 0.86259 and Coefficient of Variation = 0.87069, close to 1 (Budelli, 5).

```
%Problem 4 Code
lambda_max = max(prediction);
seq=[0:0.1:18.1];
spike_trains = zeros(length(prediction),100);
for t = 1:1000
    spike_trains(:,t) = rand(length(prediction),1)
<= (prediction)/lambda_max;
end

% Plot 4 spike trains
a=figure(1);
plot(seq, spike_trains(:,1));
xlabel('Time (s)');
ylabel('Spike Occurence');
title('Spike Train 1');
saveas(a, 'p4-1.png');

b=figure(2);
plot(seq, spike_trains(:,2));
xlabel('Time (s)');
ylabel('Spike Occurence');
title('Spike Train 2');
saveas(b, 'p4-2.png');

c=figure(3);
plot(seq, spike_trains(:,3));
xlabel('Time (s)');
ylabel('Spike Occurence');
title('Spike Train 3');
saveas(c, 'p4-3.png');

d=figure(4);
plot(seq, spike_trains(:,3));
xlabel('Time (s)');
ylabel('Spike Occurence 4');
title('Spike Train 4');
```

```
saveas(d, 'p4-4.png');

% Get ISI
ISI=[];
for t = 1:1000
    spike_col = spike_trains(:,t)';
    t_diff=[];
    interval_count = 0;
    t_pre = 0;
    for s = spike_col
       interval_count=interval_count+1;
       if s == 1
           t_diff=[t_diff (interval_count-t_pre)];
           t_pre=interval_count;
       end
    end
    ISI=[ISI t_diff];
end

%ISI distribution
e=figure(5);
histogram(ISI, 50);
xlabel('Interspike Interval (S)');
ylabel('Interval Difference');
title('ISI distribution');
saveas(e, 'p4-5.png');

%Fano
fano_factor = var(spike_trains)/mean(spike_trains);
disp("The Fano Factor is: " + string(fano_factor));

%Coefficient of Variation
coefficient_of_variation = std(ISI)/mean(ISI);
disp("Coefficient of Variation for ISI is: " +
string(coefficient_of_variation));
```

Works Cited

Budelli, R., et al. “Two-Neurons Network.” *SpringerLink*, Springer-Verlag, 4 Dec. 1991, link.springer.com/article/10.1007/BF00243285.

Izhikevich, E.m. “Simple Model of Spiking Neurons.” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, 6 Nov. 2003, pp. 1569–1572., doi:10.1109/tnn.2003.820440.

Raman, B. “HW1_BME572.pdf”