Table of Contents

a) Learning Pod 1
b) Compare and contrast the modifications to the ALIF model 1
c) Tutorial 2.2 Q1 2
Tutorial 2.2 Q2 7
Tutorial 2.2 Q3 12
Tutorial 2.3 Q1 Part a 17
Tutorial 2.3 Q1 Part b 19
Tutorial 2.3 Q2 Part a 21
Tutorial 2.3 Q2 Part b 24

a) Learning Pod

```
% POD members: Meghan, Raymond
% Struggle: I had difficulty finding a way to implement refractory period,
% I did not know how to check for refractory period in the for loop
% Help from POD: We discussed the logic to implement refractory period in
% the POD, we wrote the pseudocode to try to make our logic clear. I was
% able to implement the refractory period with the help from my POD.
```

b) Compare and contrast the modifications to the ALIF model

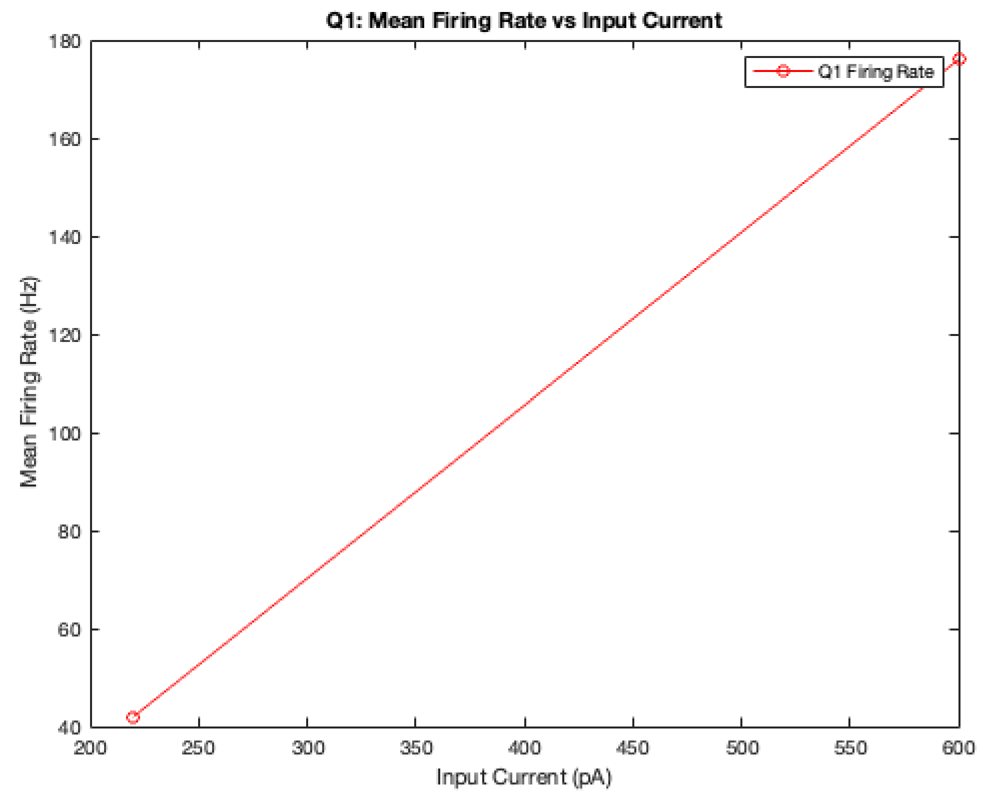
```
% Differences in mechanisms:
% The LIF model is one of the simplest forms of a neuron model that
% includes a membrane potential that leaks over time towards a baseline
% or rest potential, influenced by a linear resistance and capacitance.
% When the membrane potential reaches a threshold, the neuron fires
% (generates a spike), and the membrane potential is reset to a lower
% value.
% The AELIF model extends the LIF model by introducing an adaptation
% mechanism through an additional current termed I_SRA (Spike Rate
% Adaptation). This current increases following a spike and decays
% exponentially, affecting the neuron's excitability. It also includes an
% exponential term that makes the model more biologically realistic,
% allowing it to better capture the sharp upswing of action potentials
% observed in real neurons.


% Difference in terms and equations:
% LIF: $$ C_m \frac{dV_m}{dt} = \frac{(E_L - V_m)}{R_m} + I_{app} $$\
% V is the membrane potential
% E_L is the leak potential
% R_m is the membrane resistance
% C_m is the membrane capacitance
% I_{app} is the applied current

% AELIF:
```

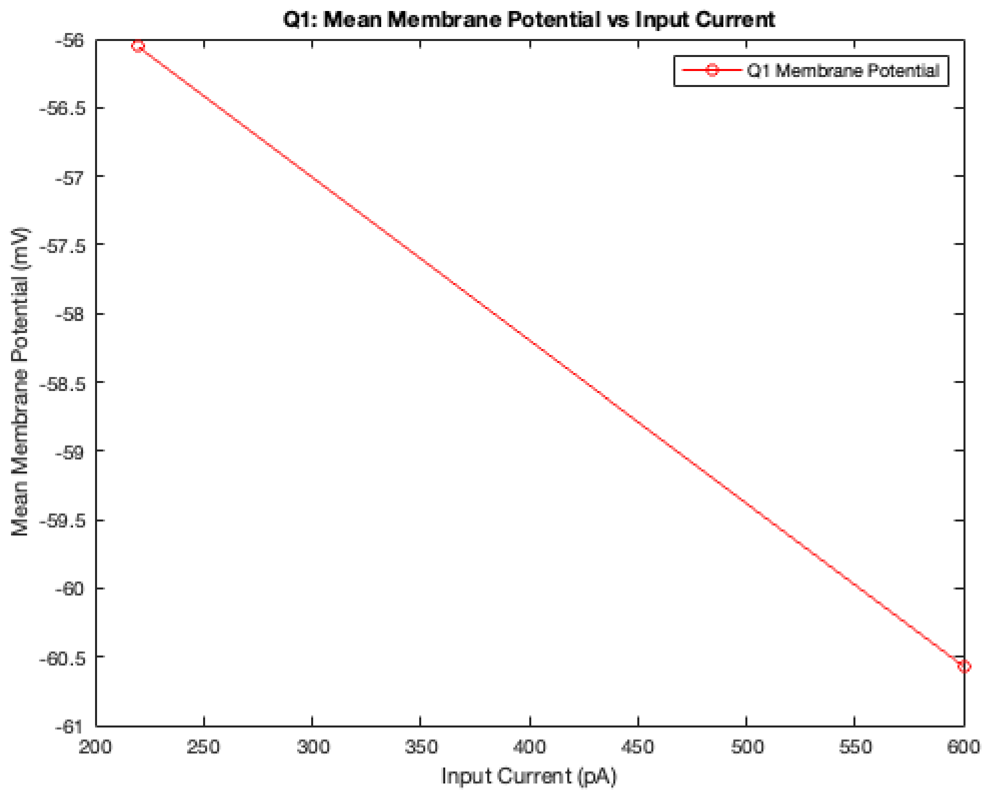
```
% C_m \frac{dV_m}{dt} = G_L \left[ E_L - V_m + \Delta_{th} \exp
\left( \frac{V_m - V_{th}}{\Delta_{th}} \right) \right] - I_{SRA} + I_{app}
% \Delta_{th} introduces an exponential component to the spike mechanism
% G_L is the leak conductance
% I_{SRA} is the adaptation current


% Difference in model behavior changes
% LIF: The response of the LIF model to constant input currents is
% relatively linear and predictable, without accommodation or adaptation
% over time. The spike frequency is directly related to the input current
% without any form of frequency adaptation.
% The AELIF model shows adaptation to prolonged stimuli. After an initial
% response to a sustained input current, the firing rate decreases over
% time due to the increasing adaptation current. Exhibits more complex
% behaviors like bursting and adaptation, which are closer to biological
% neuron behaviors. The exponential term allows the neuron to respond more
% aggressively as the potential approaches the threshold.


% In summary, the AELIF model incorporates additional complexity and
% biological realism into the basic LIF framework, allowing it to simulate
% more sophisticated behaviors observed in real neurons, such as adaptation
% and the sharp spike upswing. These enhancements make the AELIF model
% suitable for more detailed studies of neuronal behavior under various
% input conditions, providing insights into the dynamics of adaptation
% and firing patterns in response to different stimuli.
```

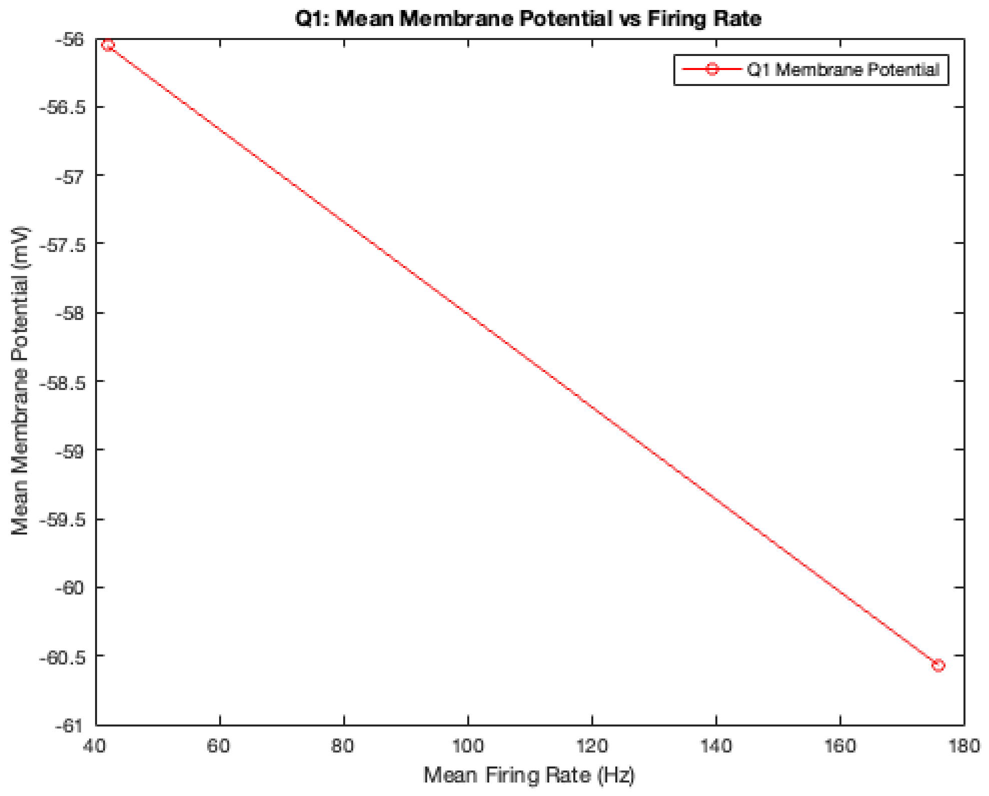
c) Tutorial 2.2 Q1

```
function [V, spike_times] = simulate_neuron_Q1(I_app, t, E_L, G_L, C_m, V_th,
V_reset, delta_t, tau_ref, V_peak)
    V = zeros(length(I_app), length(t)); % Initialize membrane potential
matrix
    spike_times = cell(length(I_app), 1); % Use a cell array to store spike
times for each I_app

    for i = 1:length(I_app)
        last_spike_time = -inf; % Initialize the last spike time
        V(i,1) = E_L; % Set the initial condition for each simulation

        for k = 2:length(t)
            % Check for refractory period
            if (t(k) > last_spike_time + tau_ref)
                dVdt = G_L * (E_L - V(i, k-1)) + I_app(i);
                V(i, k) = V(i, k-1) + dVdt * (delta_t / C_m);
            else
                V(i, k) = V_reset;
            end

            % Check for spikes
            if (V(i, k) >= V_th)
                V(i, k) = V_peak; % Set to V_peak for visualization
                last_spike_time = t(k); % Update last spike time
                spike_times{i} = [spike_times{i}, t(k)]; % Record spike time
            end
        end
    end
end

% Parameters
E_L = -70e-3;       % Leak potential in volts
R_m = 100e6;        % Membrane resistance in ohms
G_L = 1 / R_m;      % Conductance in Siemens
C_m = 0.1e-9;       % Membrane capacitance in farads
V_th = -50e-3;      % Spike threshold in volts
V_reset = -65e-3;   % Reset potential in volts
V_peak = 50e-3;     % Spike peak in volts
tau_ref = 2.5e-3;   % Refractory period in seconds

% Create time vector
delta_t = 0.1e-3;   % Time step in seconds (0.1 ms)
tmax = 2;           % Maximum time in seconds
t = 0:delta_t:tmax; % Time vector from 0 to tmax with steps of delta_t

% Input currents
I_app = [220e-12, 600e-12]; % Example input currents

% Call the simulate_neuron function
[V_Q1, spike_times_Q1] = simulate_neuron_Q1(I_app, t, E_L, G_L, C_m, V_th,
V_reset, delta_t, tau_ref, V_peak);


% Initialize the results arrays for Q1
mean_firing_rates_Q1 = zeros(size(I_app));
mean_membrane_potentials_Q1 = zeros(size(I_app));

% Data Analysis for Q1
for i = 1:length(I_app)
    % Calculate the mean firing rate
    mean_firing_rates_Q1(i) = length(spike_times_Q1{i}) / tmax; % Firing rate
in Hz

    % Calculate the mean membrane potential excluding the time of spikes
    % Here we use V_peak to exclude spike peaks for calculation
    mean_membrane_potentials_Q1(i) = mean(V_Q1(i, V_Q1(i,:) < V_peak));
end

% Plotting for Q1

% Plot mean firing rate as a function of input current for Q1
figure;
plot(I_app * 1e12, mean_firing_rates_Q1, 'r-o'); % Convert A to pA for the
x-axis
xlabel('Input Current (pA)');
ylabel('Mean Firing Rate (Hz)');
title('Q1: Mean Firing Rate vs Input Current');
```

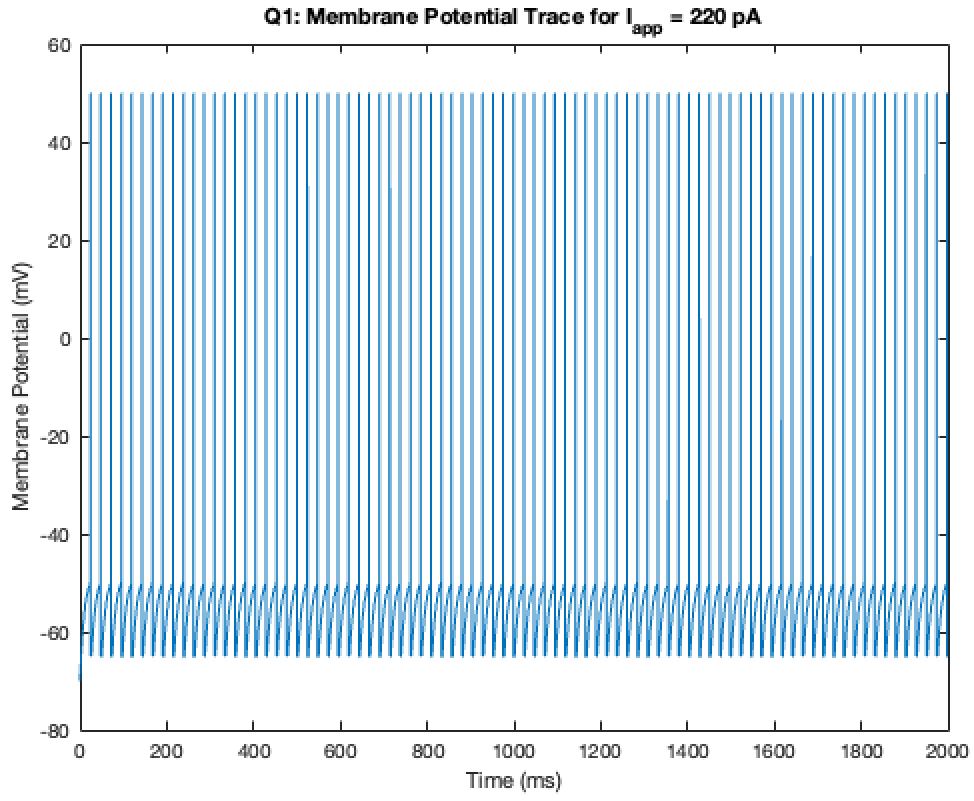
```
legend('Q1 Firing Rate');

% Plot mean membrane potential as a function of input current for Q1
figure;
plot(I_app * 1e12, mean_membrane_potentials_Q1 * 1e3, 'r-o'); % Convert V to
mV for the y-axis
xlabel('Input Current (pA)');
ylabel('Mean Membrane Potential (mV)');
title('Q1: Mean Membrane Potential vs Input Current');
legend('Q1 Membrane Potential');

% Plot mean membrane potential as a function of firing rate for Q1
figure;
plot(mean_firing_rates_Q1, mean_membrane_potentials_Q1 * 1e3, 'r-o'); %
Convert V to mV for the y-axis
xlabel('Mean Firing Rate (Hz)');
ylabel('Mean Membrane Potential (mV)');
title('Q1: Mean Membrane Potential vs Firing Rate');
legend('Q1 Membrane Potential');

% Additionally, plot the membrane potential over time for specific I_app
values for Q1
% Identify the indices for I_app = 220 pA and 600 pA if they are part of your
simulations
indices_of_interest = find(ismember(I_app, [220e-12, 600e-12]));
for i = indices_of_interest
    figure;
    plot(t * 1e3, V_Q1(i,:) * 1e3); % Convert seconds to ms for x-axis and V
to mV for the y-axis
    hold on;
    xlabel('Time (ms)');
    ylabel('Membrane Potential (mV)');
    title(['Q1: Membrane Potential Trace for I_{app} = ' num2str(I_app(i) *
1e12) ' pA']);
    hold off;
end
```


Q1: Mean Firing Rate vs Input Current
Q1 Firing Rate
Mean Firing Rate (Hz)
Input Current (pA)
180
160
140
120
100
80
60
40
200
250
300
350
400
450
500
550
600

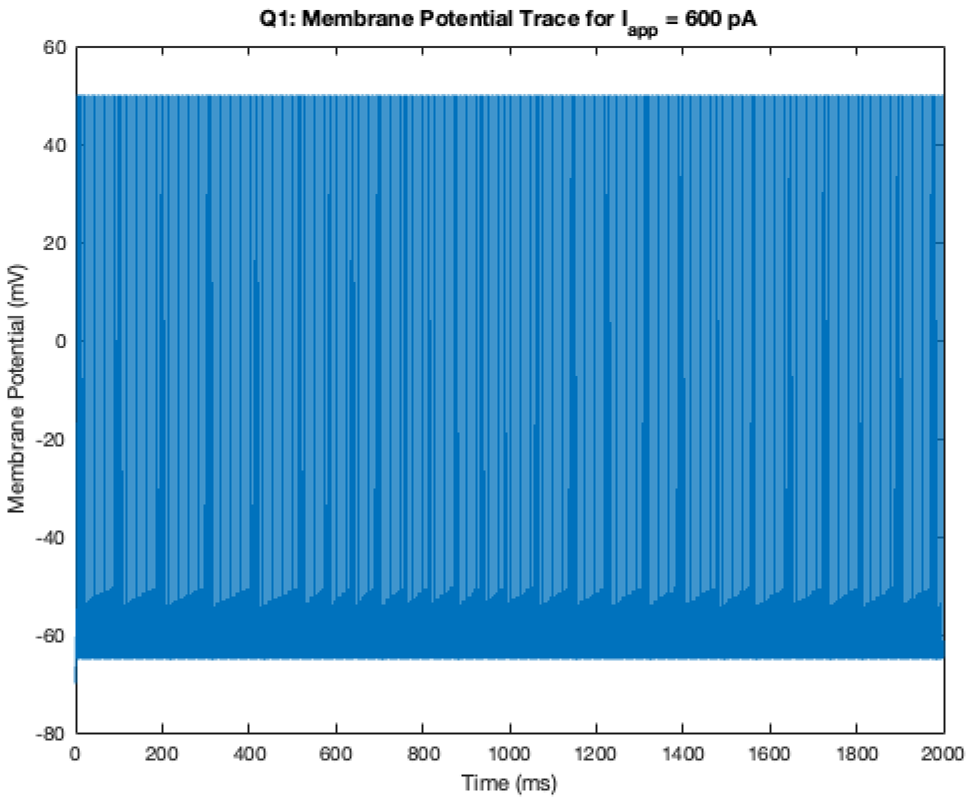

Q1: Mean Membrane Potential vs Input Current
Q1 Membrane Potential
Mean Membrane Potential (mV)
Input Current (pA)
-56
-56.5
-57
-57.5
-58
-58.5
-59
-59.5
-60
-60.5
-61
200
250
300
350
400
450
500
550
600



Q1: Mean Membrane Potential vs Firing Rate
Q1 Membrane Potential
Mean Membrane Potential (mV)
Mean Firing Rate (Hz)
-56
-56.5
-57
-57.5
-58
-58.5
-59
-59.5
-60
-60.5
-61
40
60
80
100
120
140
160
180



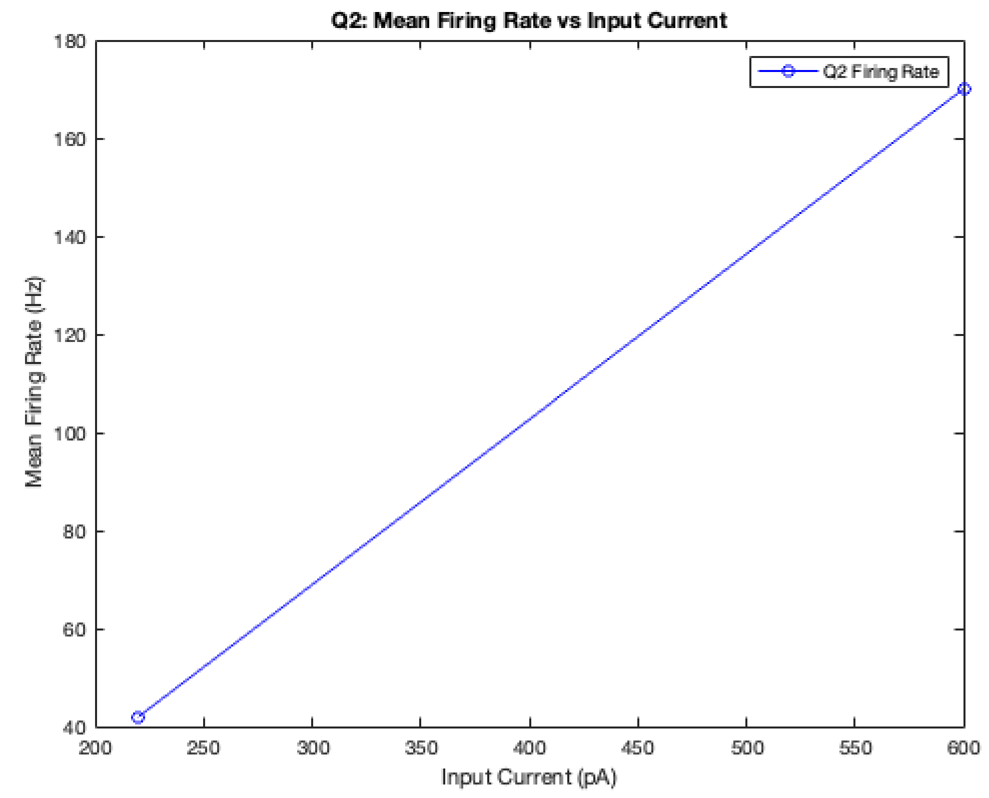
Q1: Membrane Potential Trace for I_{app} = 220 pA
Membrane Potential (mV)
Time (ms)
60
40
20
0
-20
-40
-60
-80
0
200
400
600
800
1000
1200
1400
1600
1800
2000

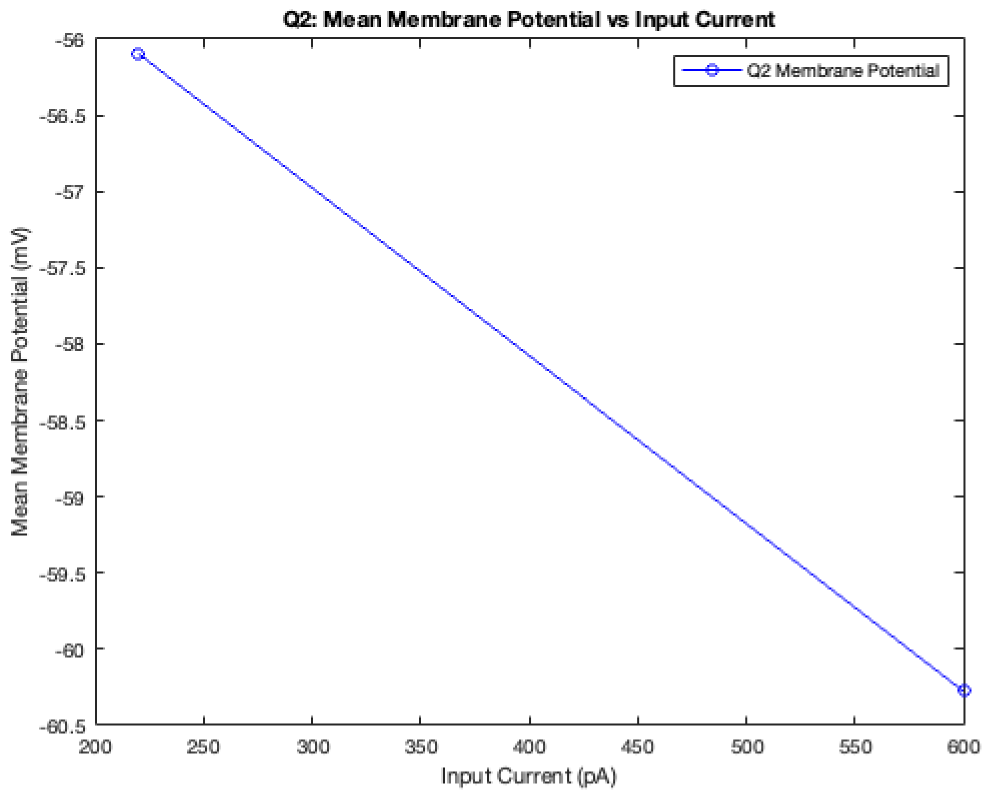


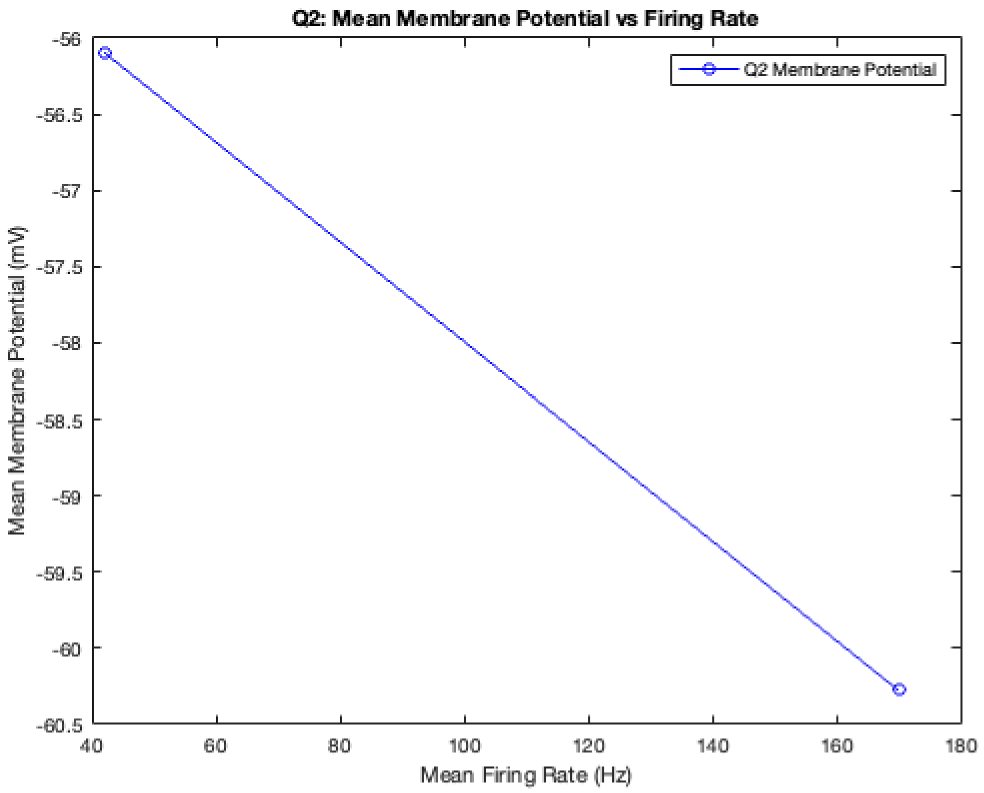
Tutorial 2.2 Q2

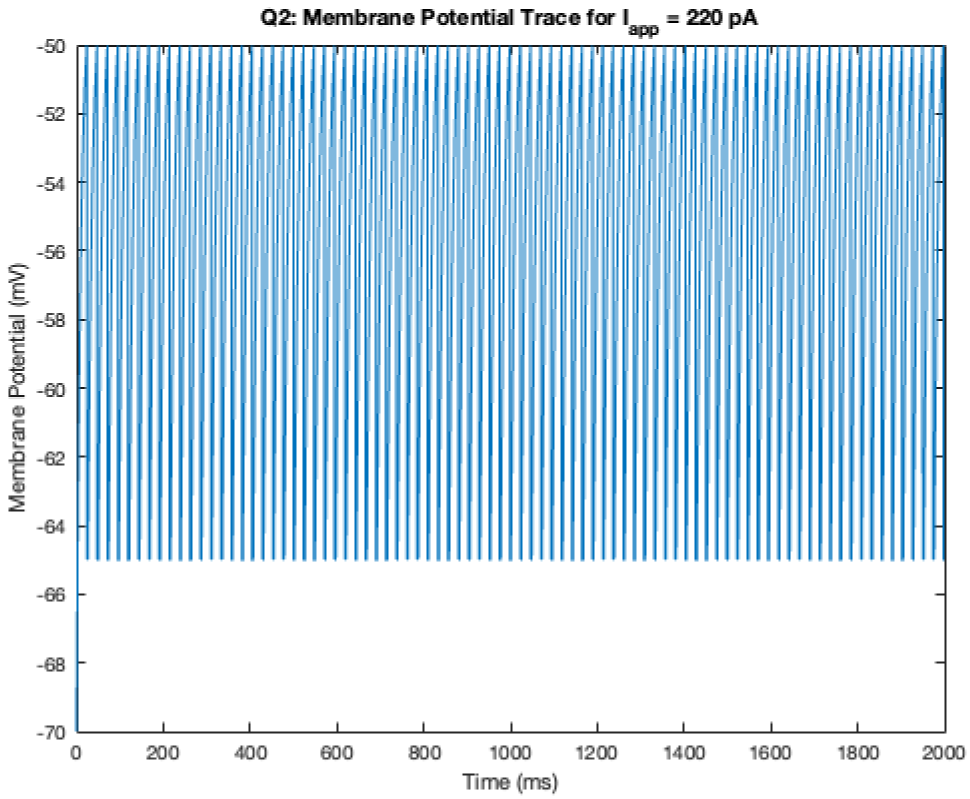
```
function [V_Q2, spike_times_Q2, V_th_Q2] = simulate_neuron_Q2(I_app, t, E_L,
G_L, C_m, V_reset, delta_t, tau_ref, ~, tau_V_th, V_th_baseline, V_th_max)
    V_Q2 = zeros(length(I_app), length(t)); % Initialize membrane potential
matrix for Q2
    V_th_Q2 = V_th_baseline * ones(length(I_app), length(t)); % Initialize
dynamic threshold matrix
    spike_times_Q2 = cell(length(I_app), 1); % Use a cell array to store
spike times for each I_app

    for i = 1:length(I_app)
        last_spike_time_Q2 = -inf; % Initialize the last spike time for Q2
        V_Q2(i,1) = E_L; % Set the initial condition for V for each simulation
        V_th_Q2(i,1) = V_th_baseline; % Set the initial condition for V_th
for each simulation

        for k = 2:length(t)
            % Update V_m(t) if we're past the refractory period
            if (t(k) > last_spike_time_Q2 + tau_ref)
                dVdt_Q2 = G_L * (E_L - V_Q2(i, k-1)) + I_app(i);
                V_Q2(i, k) = V_Q2(i, k-1) + dVdt_Q2 * (delta_t / C_m);
            else
                V_Q2(i, k) = V_reset;
            end
            % Update V_th(t)
            dV_th_dt_Q2 = (V_th_baseline - V_th_Q2(i, k-1)) / tau_V_th;
            V_th_Q2(i, k) = V_th_Q2(i, k-1) + dV_th_dt_Q2 * delta_t;

            % Check for spikes
            if (V_Q2(i, k) >= V_th_Q2(i, k))
                V_Q2(i, k) = V_reset; % Reset membrane potential
                V_th_Q2(i, k) = V_th_max; % Increase threshold
                last_spike_time_Q2 = t(k); % Update last spike time for Q2
                spike_times_Q2{i} = [spike_times_Q2{i}, t(k)]; % Record spike
time
            end
        end
    end
end

% Parameters for Q2
V_th_baseline_Q2 = -50e-3; % Baseline threshold in volts for Q2
V_th_max_Q2 = 200e-3;      % Maximum threshold after spike in volts for Q2
tau_V_th_Q2 = 1e-3;        % Time constant for threshold decay in seconds for
Q2

% Call the simulate_neuron_Q2 function
[V_Q2, spike_times_Q2, V_th_Q2] = simulate_neuron_Q2(I_app, t, E_L, G_L,
C_m, V_reset, delta_t, tau_ref, V_peak, tau_V_th_Q2, V_th_baseline_Q2,
V_th_max_Q2);

% Data analysis for Q2
mean_firing_rates_Q2 = zeros(size(I_app));
mean_membrane_potentials_Q2 = zeros(size(I_app));

% Compute mean firing rates and membrane potentials
for i = 1:length(I_app)
    mean_firing_rates_Q2(i) = length(spike_times_Q2{i}) / tmax;  % Firing
rate in Hz
    mean_membrane_potentials_Q2(i) = mean(V_Q2(i, V_Q2(i,:) < V_th_max_Q2));
% Exclude spike peaks
end

% Plotting for Q2

% Plot mean firing rate as a function of input current for Q2
figure;
plot(I_app * 1e12, mean_firing_rates_Q2, 'b-o');  % Convert current to pA for
the x-axis
xlabel('Input Current (pA)');
ylabel('Mean Firing Rate (Hz)');
title('Q2: Mean Firing Rate vs Input Current');
legend('Q2 Firing Rate');

% Plot mean membrane potential as a function of input current for Q2
figure;
plot(I_app * 1e12, mean_membrane_potentials_Q2 * 1e3, 'b-o');  % Convert V to
```

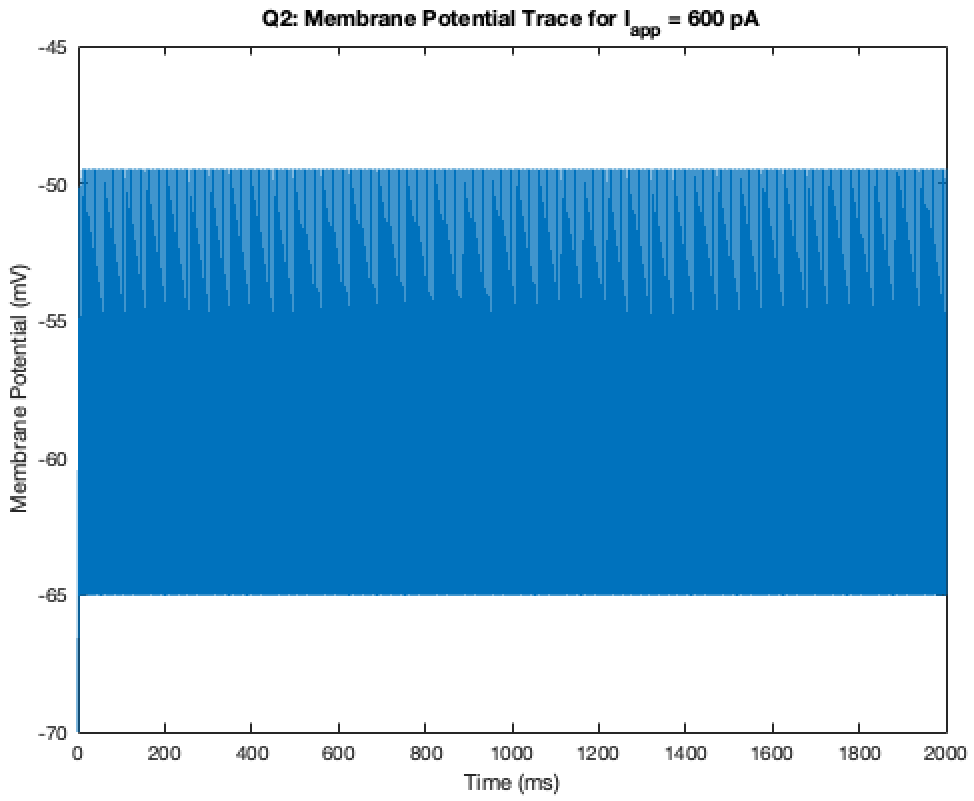
```
mV for the y-axis
xlabel('Input Current (pA)');
ylabel('Mean Membrane Potential (mV)');
title('Q2: Mean Membrane Potential vs Input Current');
legend('Q2 Membrane Potential');

% Plot mean membrane potential as a function of firing rate for Q2
figure;
plot(mean_firing_rates_Q2, mean_membrane_potentials_Q2 * 1e3, 'b-o');  %
Convert V to mV for the y-axis
xlabel('Mean Firing Rate (Hz)');
ylabel('Mean Membrane Potential (mV)');
title('Q2: Mean Membrane Potential vs Firing Rate');
legend('Q2 Membrane Potential');

% Additionally, plot the membrane potential as a function of time for
specific I_app values
% (e.g., 220 pA and 600 pA, but you need to find their indices in the I_app
array)
for i = 1:length(I_app)
    figure;
    plot(t * 1e3, V_Q2(i,:) * 1e3);  % Convert seconds to ms for the x-axis
and V to mV for the y-axis
    hold on;
    xlabel('Time (ms)');
    ylabel('Membrane Potential (mV)');
    title(['Q2: Membrane Potential Trace for I_{app} = ' num2str(I_app(i) *
1e12) ' pA']);
    hold off;
end
```


Q2: Mean Firing Rate vs Input Current
Q2 Firing Rate
Mean Firing Rate (Hz)
Input Current (pA)
180
160
140
120
100
80
60
40
200
250
300
350
400
450
500
550
600


Q2: Mean Membrane Potential vs Input Current
Q2 Membrane Potential
Mean Membrane Potential (mV)
Input Current (pA)
-56
-56.5
-57
-57.5
-58
-58.5
-59
-59.5
-60
-60.5
200
250
300
350
400
450
500
550
600


Q2: Mean Membrane Potential vs Firing Rate
Q2 Membrane Potential
Mean Membrane Potential (mV)
Mean Firing Rate (Hz)
-56
-56.5
-57
-57.5
-58
-58.5
-59
-59.5
-60
-60.5
40
60
80
100
120
140
160
180


Q2: Membrane Potential Trace for I_{app} = 220 pA
Membrane Potential (mV)
Time (ms)
-50
-52
-54
-56
-58
-60
-62
-64
-66
-68
-70
0
200
400
600
800
1000
1200
1400
1600
1800
2000

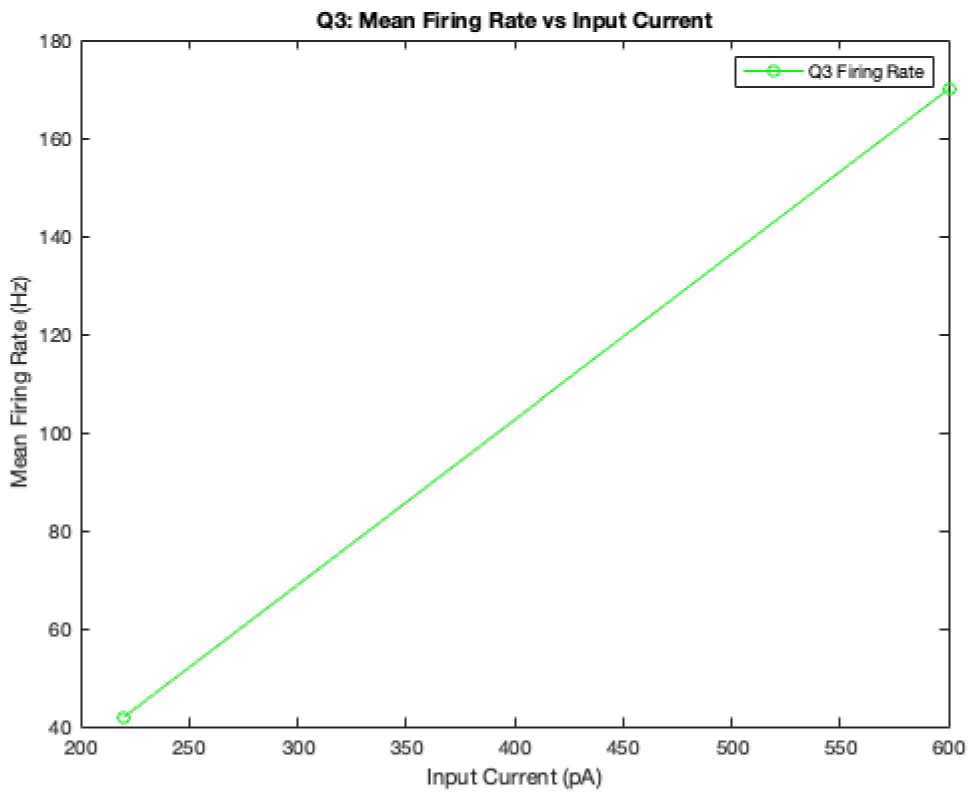


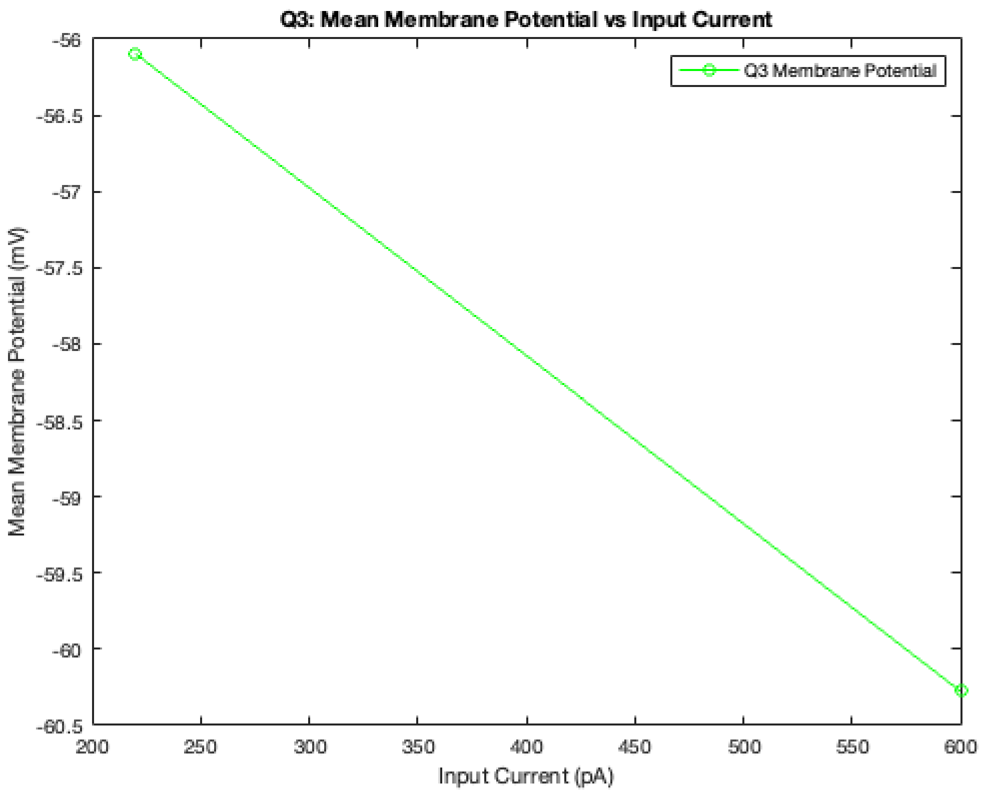
Tutorial 2.2 Q3

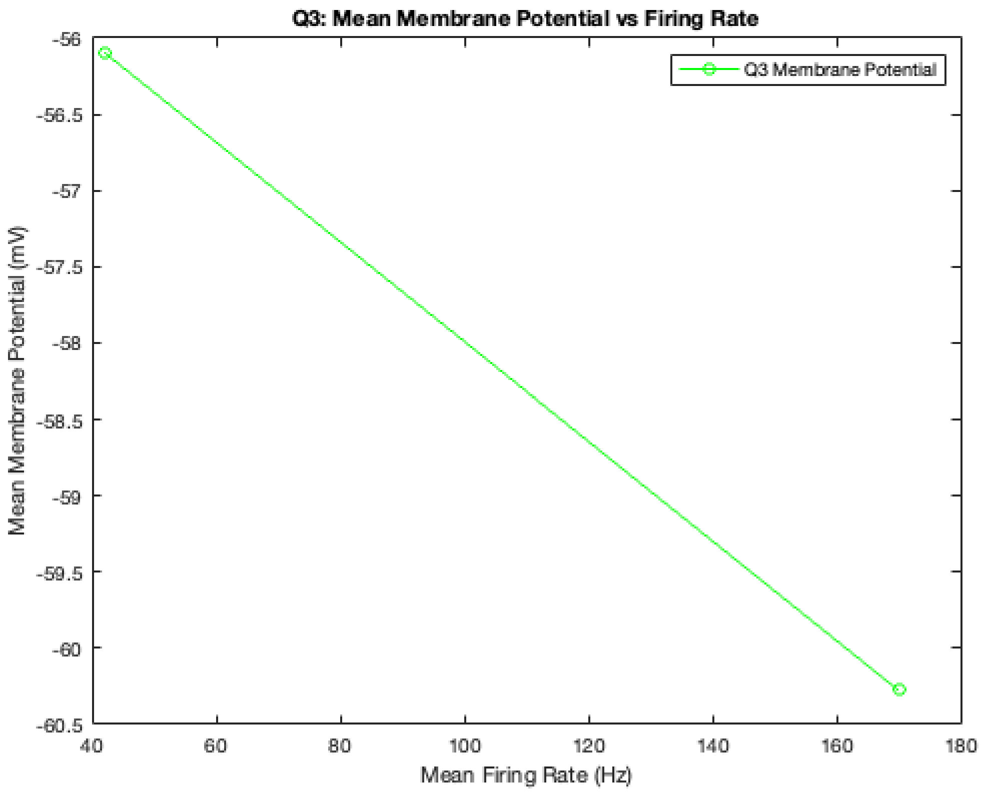
```
function [V, spike_times, V_th, G_ref] = simulate_neuron_Q3(I_app, t, E_L,
G_L, C_m, V_reset, delta_t, tau_ref, ~, tau_V_th, V_th_baseline, V_th_max,
E_K, tau_G_ref, Delta_G)
    V = zeros(length(I_app), length(t)); % Initialize membrane potential
matrix
    V_th = V_th_baseline * ones(length(I_app), length(t)); % Initialize
dynamic threshold matrix
    G_ref = zeros(length(I_app), length(t)); % Initialize refractory
conductance matrix
    spike_times = cell(length(I_app), 1); % Use a cell array to store spike
times for each I_app

    for i = 1:length(I_app)
        last_spike_time = -inf; % Initialize the last spike time
        V(i,1) = E_L; % Set the initial condition for each simulation
        V_th(i,1) = V_th_baseline; % Set the initial condition for V_th for
each simulation

        for k = 2:length(t)
            % Update V_m(t) if we're past the refractory period
            if (t(k) > last_spike_time + tau_ref)
                dVdt = G_L * (E_L - V(i, k-1)) + G_ref(i, k-1) * (E_K - V(i,
k-1)) + I_app(i);
```

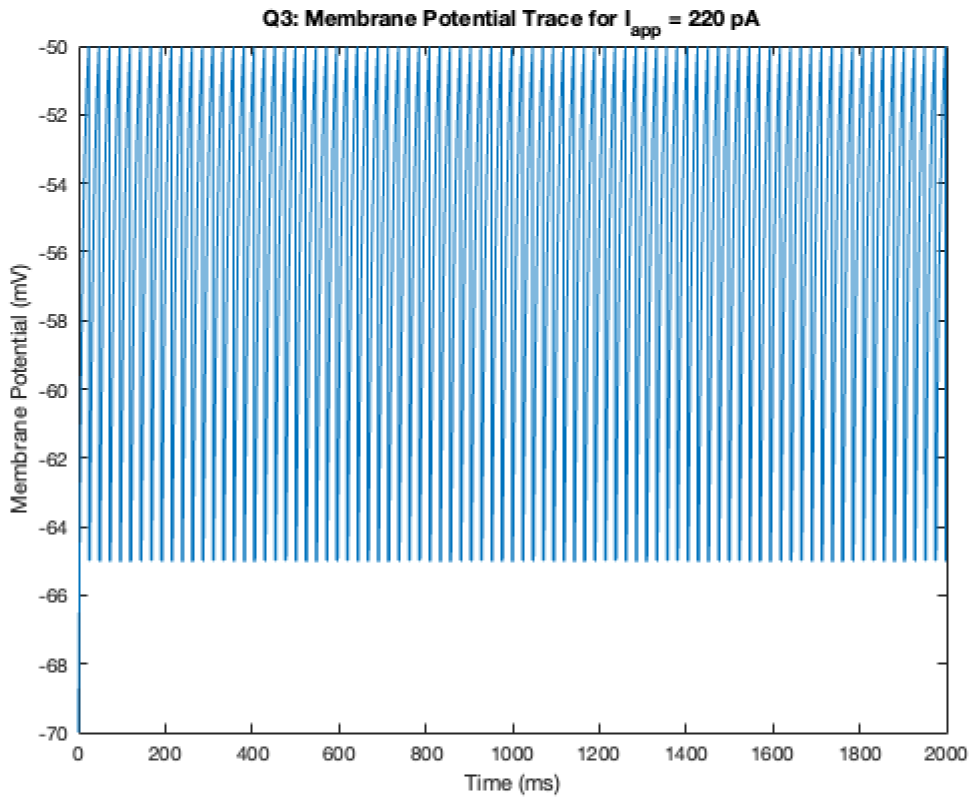
```
                V(i, k) = V(i, k-1) + dVdt * (delta_t / C_m);
            else
                V(i, k) = V_reset;
            end

            % Update G_ref(t)
            dG_ref_dt = -G_ref(i, k-1) / tau_G_ref;
            G_ref(i, k) = G_ref(i, k-1) + dG_ref_dt * delta_t;

            % Update V_th(t)
            dV_th_dt = (V_th_baseline - V_th(i, k-1)) / tau_V_th;
            V_th(i, k) = V_th(i, k-1) + dV_th_dt * delta_t;

            % Check for spikes
            if (V(i, k) >= V_th(i, k))
                V(i, k) = V_reset; % Reset membrane potential
                V_th(i, k) = V_th_max; % Increase threshold
                G_ref(i, k) = G_ref(i, k) + Delta_G; % Increase refractory
conductance
                last_spike_time = t(k); % Update last spike time
                spike_times{i} = [spike_times{i}, t(k)]; % Record spike time
            end
        end
    end
end

% Additional parameters specific to Q3
E_K = -80e-3; % Potassium reversal potential in volts
tau_G_ref = 0.2e-3; % Time constant for G_ref decay in seconds
Delta_G = 2e-6; % Increment in refractory conductance in Siemens (2 micro
Siemens)
tau_V_th = 1e-3;       % Time constant for the decay of the threshold voltage
V_th_baseline = -50e-3; % Baseline threshold voltage in volts
V_th_max = 200e-3;     % Maximum threshold voltage after a spike in volts

% Assume other parameters (like E_L, R_m, C_m, etc.) are already defined in
your script.
% Call the function for Q3
[V_Q3, spike_times_Q3, V_th_Q3, G_ref_Q3] = simulate_neuron_Q3(I_app, t,
E_L, G_L, C_m, V_reset, delta_t, tau_ref, V_peak, tau_V_th, V_th_baseline,
V_th_max, E_K, tau_G_ref, Delta_G);

% Initialize arrays to store results
mean_firing_rates_Q3 = zeros(size(I_app));
mean_membrane_potentials_Q3 = zeros(size(I_app));

% Compute mean firing rates and membrane potentials
for i = 1:length(I_app)
    mean_firing_rates_Q3(i) = length(spike_times_Q3{i}) / tmax; % Firing rate
in Hz
    mean_membrane_potentials_Q3(i) = mean(V_Q3(i, V_Q3(i,:) < V_peak)); %
Exclude spike peaks for calculation
end
```

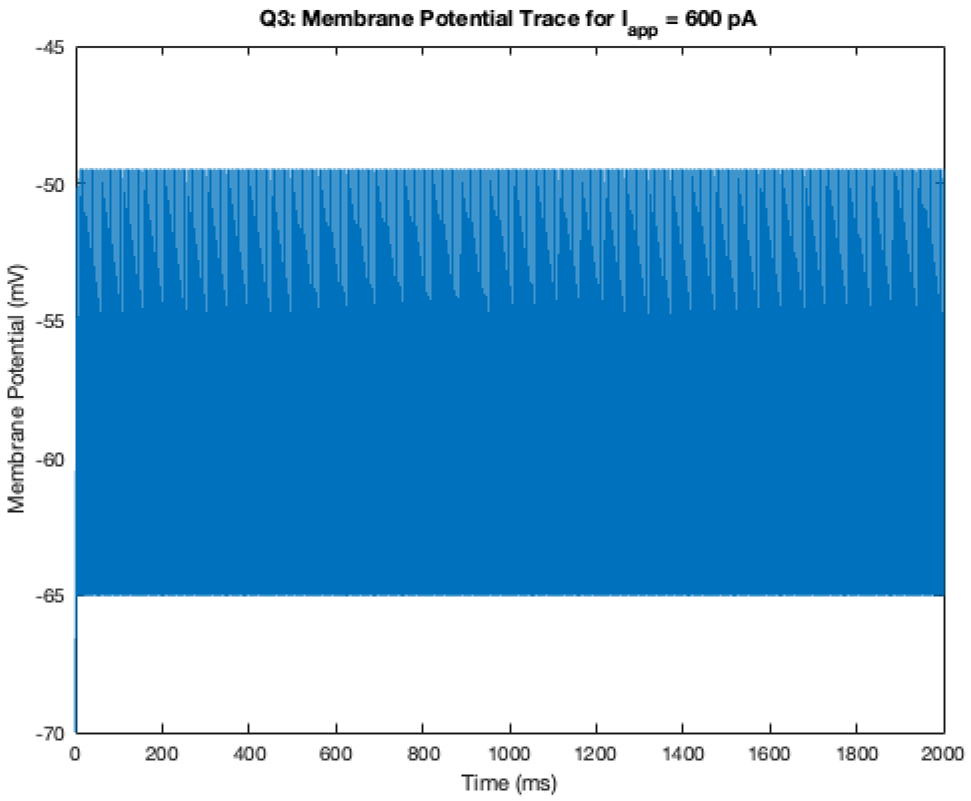
```
% Plot mean firing rate as a function of input current for Q3
figure;
plot(I_app * 1e12, mean_firing_rates_Q3, 'g-o'); % Convert A to pA for the
x-axis
xlabel('Input Current (pA)');
ylabel('Mean Firing Rate (Hz)');
title('Q3: Mean Firing Rate vs Input Current');
legend('Q3 Firing Rate');

% Plot mean membrane potential as a function of input current for Q3
figure;
plot(I_app * 1e12, mean_membrane_potentials_Q3 * 1e3, 'g-o'); % Convert V to
mV for the y-axis
xlabel('Input Current (pA)');
ylabel('Mean Membrane Potential (mV)');
title('Q3: Mean Membrane Potential vs Input Current');
legend('Q3 Membrane Potential');

% Plot mean membrane potential as a function of firing rate for Q3
figure;
plot(mean_firing_rates_Q3, mean_membrane_potentials_Q3 * 1e3, 'g-o'); %
Convert V to mV for the y-axis
xlabel('Mean Firing Rate (Hz)');
ylabel('Mean Membrane Potential (mV)');
title('Q3: Mean Membrane Potential vs Firing Rate');
legend('Q3 Membrane Potential');

% Membrane potential traces ver time for specific I_app values for Q3
indices_of_interest = find(ismember(I_app, [220e-12, 600e-12])); % Example
indices
for i = indices_of_interest
    figure;
    plot(t * 1e3, V_Q3(i,:) * 1e3); % Convert seconds to ms for x-axis and V
to mV for the y-axis
    hold on;
    xlabel('Time (ms)');
    ylabel('Membrane Potential (mV)');
    title(['Q3: Membrane Potential Trace for I_{app} = ' num2str(I_app(i) *
1e12) ' pA']);
    hold off;
end
```


Q3: Mean Firing Rate vs Input Current
Mean Firing Rate (Hz)
Input Current (pA)
Q3 Firing Rate
180
160
140
120
100
80
60
40
200
250
300
350
400
450
500
550
600


Q3: Mean Membrane Potential vs Input Current
Mean Membrane Potential (mV)
Input Current (pA)
Q3 Membrane Potential
-56
-56.5
-57
-57.5
-58
-58.5
-59
-59.5
-60
-60.5
200
250
300
350
400
450
500
550
600

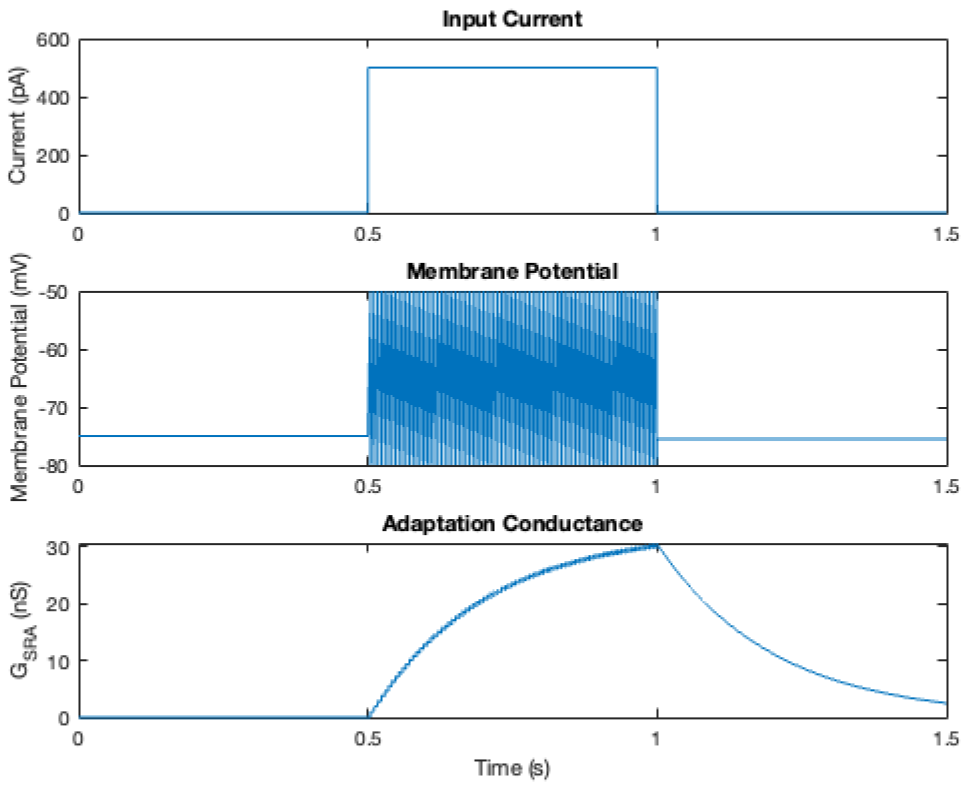

Q3: Mean Membrane Potential vs Firing Rate
Q3 Membrane Potential
Mean Membrane Potential (mV)
Mean Firing Rate (Hz)
-56
-56.5
-57
-57.5
-58
-58.5
-59
-59.5
-60
-60.5
40
60
80
100
120
140
160
180


Q3: Membrane Potential Trace for I_{app} = 220 pA
Membrane Potential (mV)
Time (ms)
-50
-52
-54
-56
-58
-60
-62
-64
-66
-68
-70
0
200
400
600
800
1000
1200
1400
1600
1800
2000



Tutorial 2.3 Q1 Part a

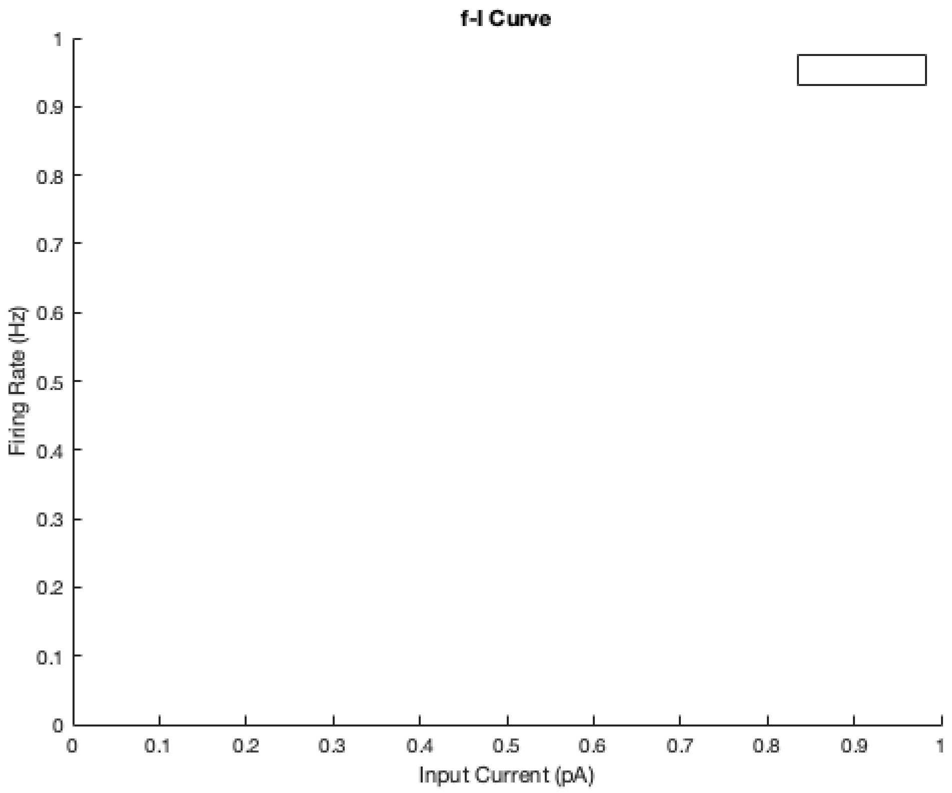
```
function [t, V, G_SRA, I_app] = simulate_LIF_with_adaptation(T, dt, I_app,
params)
    % Unpack parameters
    E_L = params.E_L;       % Leak reversal potential
    V_th = params.V_th;     % Spike threshold
    V_reset = params.V_reset;  % Reset potential
    R_m = params.R_m;       % Membrane resistance
    C_m = params.C_m;       % Membrane capacitance
    E_K = params.E_K;       % Potassium reversal potential
    Delta_G_SRA = params.Delta_G_SRA;  % Change in G_SRA at spike
    tau_SRA = params.tau_SRA;  % Time constant for SRA decay

    % Time vector
    t = 0:dt:T;

    % Initialize variables
    V = E_L * ones(size(t));   % Membrane potential
    G_SRA = zeros(size(t));    % Spike Rate Adaptation conductance

    % Simulation loop
    for i = 2:length(t)
        % Update the membrane potential
        dVdt = (E_L - V(i-1)) / R_m + G_SRA(i-1) * (E_K - V(i-1)) +
I_app(i) / C_m;
        V(i) = V(i-1) + dt * dVdt;

        % Update the adaptation conductance
        dG_SRA_dt = -G_SRA(i-1) / tau_SRA;
        G_SRA(i) = G_SRA(i-1) + dt * dG_SRA_dt;

        % Spike event
        if V(i) >= V_th
            V(i) = V_reset;
            G_SRA(i) = G_SRA(i) + Delta_G_SRA;
        end
    end
end

% Parameters
params = struct();
params.E_L = -75e-3;          % Leak reversal potential in volts
params.V_th = -50e-3;         % Spike threshold in volts
params.V_reset = -80e-3;      % Reset potential in volts
params.R_m = 100e6;           % Membrane resistance in ohms
params.C_m = 100e-12;         % Membrane capacitance in farads
params.E_K = -80e-3;          % Potassium reversal potential in volts
params.Delta_G_SRA = 1e-9;    % Change in G_SRA at spike in Siemens
params.tau_SRA = 200e-3;      % Time constant for SRA decay in seconds

% Simulation settings
T = 1.5;                      % Total time in seconds
dt = 1e-4;                    % Time step in seconds
t = 0:dt:T;                   % Time vector
I_app = zeros(size(t)) + 500e-12;  % Current in amperes
I_app(t < 0.5 | t > 1.0) = 0;  % Current pulse from 0.5 s to 1.0 s

% Run the simulation
[t, V, G_SRA, I_app] = simulate_LIF_with_adaptation(T, dt, I_app, params);

% Plotting
figure;
subplot(3,1,1);
plot(t, I_app * 1e12);
ylabel('Current (pA)');
title('Input Current');

subplot(3,1,2);
plot(t, V * 1e3);
ylabel('Membrane Potential (mV)');
title('Membrane Potential');

subplot(3,1,3);
plot(t, G_SRA * 1e9);
ylabel('G_{SRA} (nS)');
xlabel('Time (s)');
title('Adaptation Conductance');
```



Tutorial 2.3 Q1 Part b

```
% Parameters (same as part a)
params = struct();
params.E_L = -75e-3;           % Leak reversal potential in volts
params.V_th = -50e-3;          % Spike threshold in volts
params.V_reset = -80e-3;       % Reset potential in volts
params.R_m = 100e6;            % Membrane resistance in ohms
params.C_m = 100e-12;          % Membrane capacitance in farads
params.E_K = -80e-3;           % Potassium reversal potential in volts
params.Delta_G_SRA = 1e-9;     % Change in G_SRA at spike in Siemens
params.tau_SRA = 200e-3;       % Time constant for SRA decay in seconds

% Simulation settings for part b
T = 5;                         % Total time in seconds
dt = 1e-4;                     % Time step in seconds
t = 0:dt:T;                    % Time vector

% Define the range of applied currents from 0 pA to 500 pA
I_app_values = linspace(0, 500e-12, 20); % Applied current values in amperes

% Arrays to store the interspike intervals
first_ISIs = zeros(size(I_app_values));
steady_state_ISIs = zeros(size(I_app_values));
```

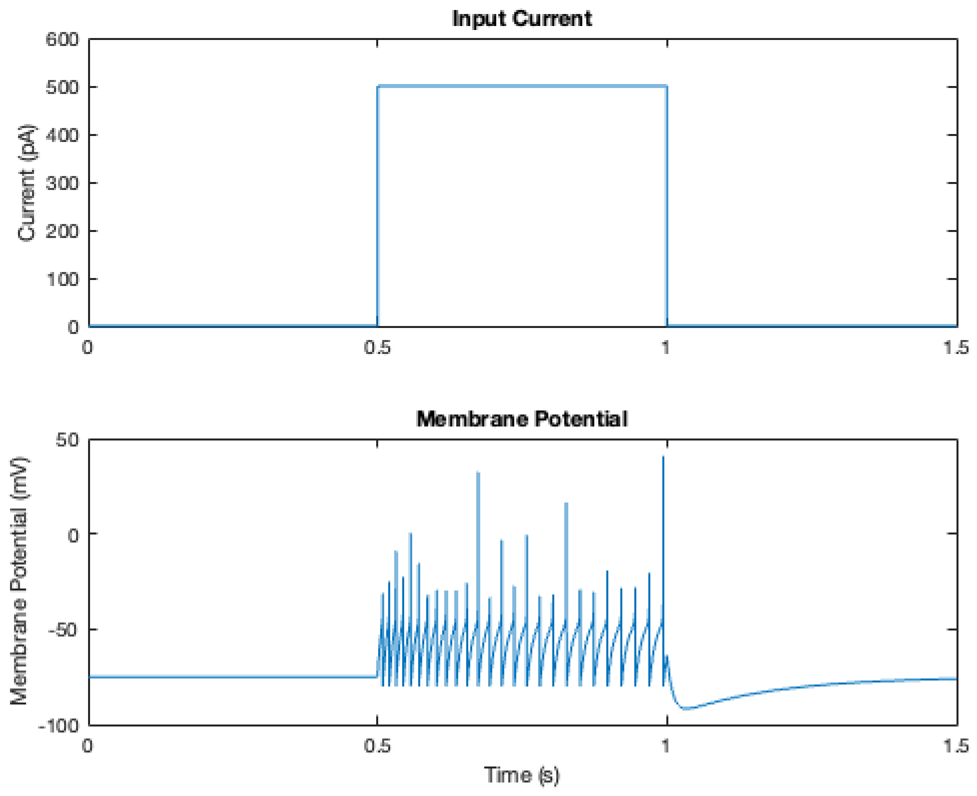
```
% Simulate for each current level
for idx = 1:length(I_app_values)
    I_app = I_app_values(idx) * ones(size(t));  % Constant current

    % Run the simulation
    [~, V, ~, ~] = simulate_LIF_with_adaptation(T, dt, I_app, params);

    % Detect spikes
    spikes = find(V(1:end-1) < params.V_th & V(2:end) >= params.V_th);
    spike_times = t(spikes);

    if length(spike_times) > 1
        % Calculate interspike intervals
        ISIs = diff(spike_times);
        first_ISIs(idx) = ISIs(1);
        steady_state_ISIs(idx) = mean(ISIs(end-4:end));  % Average the last
few ISIs for steady state
    else
        first_ISIs(idx) = NaN;
        steady_state_ISIs(idx) = NaN;
    end
end

% Plotting the f-I curve
figure;
hold on;
% Check and plot only valid indices where ISI is not NaN and not Inf
valid_ss = ~isnan(1 ./ steady_state_ISIs) & ~isinf(1 ./ steady_state_ISIs);
valid_fi = ~isnan(1 ./ first_ISIs) & ~isinf(1 ./ first_ISIs);

plot(I_app_values(valid_ss) * 1e12, 1 ./ steady_state_ISIs(valid_ss), 'b-o',
'DisplayName', 'Steady-state ISI');
plot(I_app_values(valid_fi) * 1e12, 1 ./ first_ISIs(valid_fi), 'rx',
'DisplayName', 'First ISI');

xlabel('Input Current (pA)');
ylabel('Firing Rate (Hz)');
title('f-I Curve');
legend show;
hold off;
```

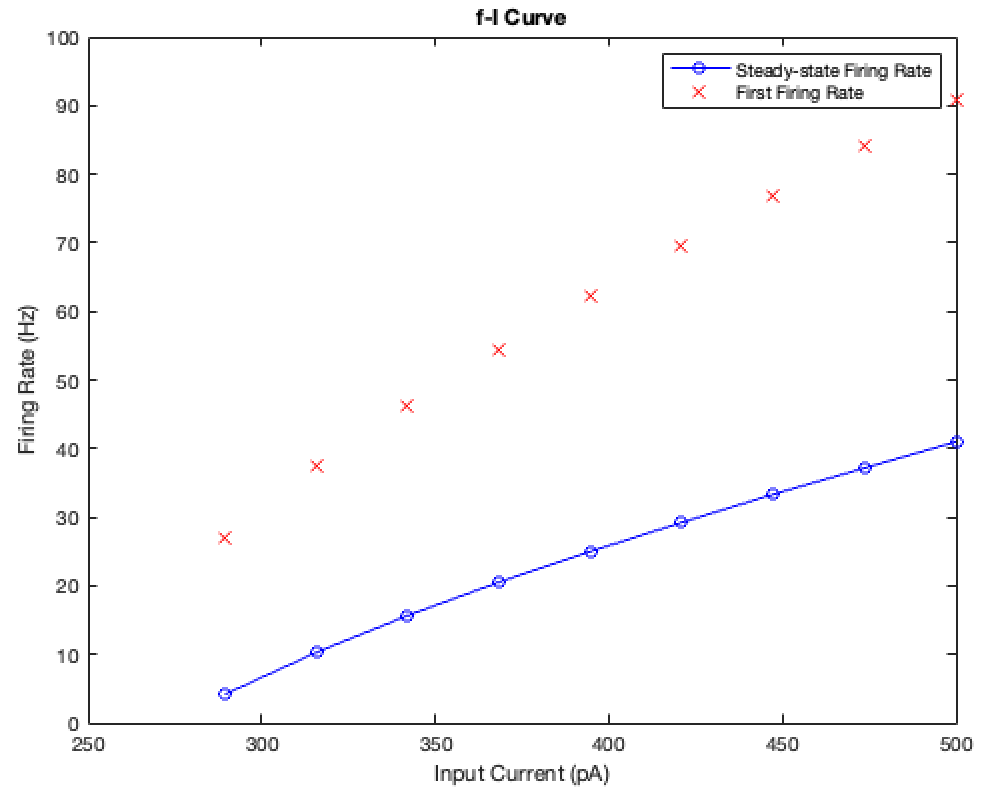


Tutorial 2.3 Q2 Part a

```
function [V, I_SRA, spike_times] = simulate_AELIF(T, dt, I_app, params)
    % Initialize variables
    t = 0:dt:T;
    V = params.E_L * ones(1, length(t));
    I_SRA = zeros(1, length(t));
    spike_times = [];

    % Simulation loop
    for i = 2:length(t)
        % Membrane potential dynamics
        exp_term = params.Delta_th * exp((V(i-1) - params.V_th) /
params.Delta_th);
        dVdt = (params.G_L * (params.E_L - V(i-1) + exp_term) - I_SRA(i-1) +
I_app(i)) / params.C_m;
        V(i) = V(i-1) + dt * dVdt;

        % Adaptation current dynamics
        dISRA_dt = (params.a * (V(i-1) - params.E_L) - I_SRA(i-1)) /
params.tau_SRA;
        I_SRA(i) = I_SRA(i-1) + dt * dISRA_dt;

        % Spike condition
        if V(i) > params.V_max
```

```
            V(i) = params.V_reset;
            I_SRA(i) = I_SRA(i) + params.b;
            spike_times = [spike_times, i * dt];  % Record spike times
        end
    end
end

% Define Parameters
params = struct();
params.E_L = -75e-3;         % Resting membrane potential in volts
params.V_th = -50e-3;        % Threshold potential in volts
params.V_reset = -80e-3;     % Reset potential in volts
params.Delta_th = 2e-3;      % Slope factor in volts
params.G_L = 10e-9;          % Leak conductance in Siemens
params.C_m = 100e-12;        % Membrane capacitance in Farads
params.a = 2e-9;             % Subthreshold adaptation in Siemens
params.b = 0.02e-9;          % Spike-triggered adaptation in Amperes
params.tau_SRA = 200e-3;     % Adaptation time constant in seconds
params.V_max = 50e-3;        % Spike cut-off potential in volts

% Simulation settings
T = 1.5;                      % Total simulation time in seconds
dt = 1e-4;                    % Time step in seconds
t = 0:dt:T;                   % Time vector
I_app = zeros(1, length(t));  % Initialize I_app for entire simulation
duration
I_app(t >= 0.5 & t <= 1.0) = 500e-12;  % Current pulse from 0.5s to 1.0s

[V, I_SRA, ~] = simulate_AELIF(T, dt, I_app, params);


% Plotting
figure;
subplot(2,1,1);
plot(t, I_app * 1e12);
ylabel('Current (pA)');
title('Input Current');

subplot(2,1,2);
plot(t, V * 1e3);
ylabel('Membrane Potential (mV)');
xlabel('Time (s)');
title('Membrane Potential');
```


Input Current
Current (pA)
Membrane Potential
Membrane Potential (mV)
Time (s)


f-I Curve
Steady-state Firing Rate
First Firing Rate
Firing Rate (Hz)
Input Current (pA)

Tutorial 2.3 Q2 Part b

```
% Simulation settings
T = 5;
dt = 1e-4;
t = 0:dt:T;  % Ensure t is defined here or globally accessible
I_app_values = linspace(0, 500e-12, 20);  % Applied current range

first_ISIs = [];
steady_state_ISIs = [];

for I_app_const = I_app_values
    I_app = I_app_const * ones(1, length(t));  % Make sure it spans the full
simulation duration
    [~, ~, spike_times] = simulate_AELIF(T, dt, I_app, params);

    if length(spike_times) > 1
        ISIs = diff(spike_times);
        first_ISIs(end+1) = ISIs(1);
        steady_state_ISIs(end+1) = mean(ISIs(end-4:end));  % Average last few
ISIs
    else
        first_ISIs(end+1) = NaN;
        steady_state_ISIs(end+1) = NaN;
    end
end


% Plotting f-I Curve
figure;
plot(I_app_values * 1e12, 1 ./ steady_state_ISIs, 'b-o', 'DisplayName',
'Steady-state Firing Rate');
hold on;
plot(I_app_values * 1e12, 1 ./ first_ISIs, 'rx', 'DisplayName', 'First Firing
Rate');
xlabel('Input Current (pA)');
ylabel('Firing Rate (Hz)');
title('f-I Curve');
legend;
hold off;
```

Published with MATLAB® R2024a