a) Learning Pod

Table of Contents

POD members: Meghan, Raymond Struggle: I did not know how to write the for loop for to simulating neuron behavior Help from POD: My POD mates helped me to figure out the logic to write the for loop to simulate the neuron behavior. Then I was able to apply a similar logic to implement the neuron behavior with noise and graph them to compare the difference.

Euler's Method

Explain Euler’s method as an approach to approximate the solution of ordinary differential equations. You may use diagrams such as flow charts or pseudocode.

```
% Euler's method approximates the solution by using a series of steps
% along the function's tangent line to estimate the function's values.
% It starts from an initial point and progresses step-by-step, calculating
% the next point using the derivative at the current point.

% The equations of Euler's method for solving an ODE
%           dy/dx = f(x,y), y(x_0) = y_0
% where f(x,y) is a function that can be evaluated at any point (x,y) and
% y(x_0) = y_0 is the initial condition at x_0

% Here is the psuedocode for Euler's Method:

% Initialize x to the initial x value, x_0
% Initialize y to the initial y value, y_0
% Specify the step size, h (a small number)
% Specify the number of steps, N or the ending value of x, x_end
%
% FOR i FROM 0 TO N-1
%     slope = f(x, y)          // Calculate the slope at the current point
%     y = y + slope * h        // Update y using the slope and step size
%     x = x + h                // Update x by adding the step size
% END FOR
```

Tutorial 2.1 1a

```
% i) Define parameters
E_L = -70e-3; % Leak potential (E_L) in volts
```

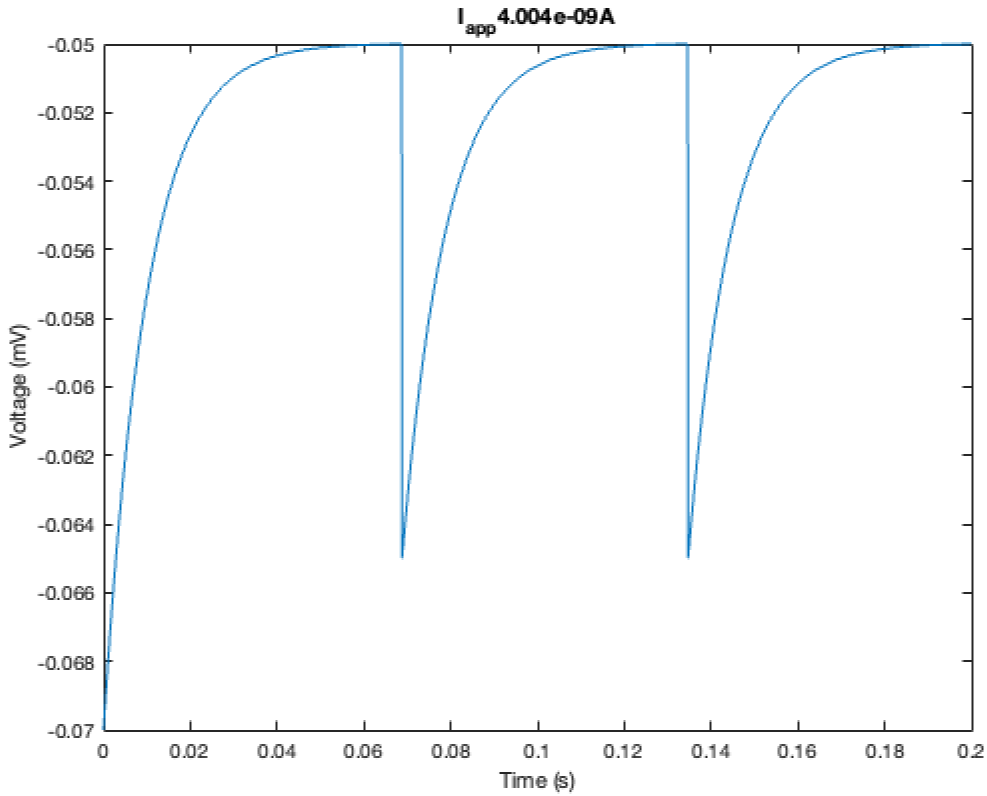
```
R_m = 5e6; % Membrane resistance (R_m) in ohms
C_m = 2e-9; % Membrane capacitance (C_m) in farads
V_th = -50e-3; % Spike threshold (V_th) in volts
V_reset = -65e-3; % Reset potential (V_reset) in volts

% ii) Create a time vector
delta_t = 0.0001; % integration 0.1 ms
tmax = 2; % Max time in seconds
t = 0:delta_t:tmax; % Time vector

% iii) Create a vector for the membrane potential, V
V = zeros(size(t));

% iv) Set the initial value of V to E_L
V(1) = E_L;

% v) Create a vector for the applied current, I_app
I_0 = 0; % A
I_app_temp = zeros(size(t)); % Applied current array

% vi) Set up the for loop to integrate through time
for k = 2:length(t)
    % vii) Update the membrane potential using the Forward Euler method
    dVdt = (1/C_m) * ((E_L - V(k-1)) * (1/ R_m) + I_app_temp(k));
    V(k) = V(k-1) + dVdt * delta_t;

    % viii) Check if the membrane potential is above threshold
    if (V > V_th)
        V(k) = V_reset; % Reset the membrane potential
    end % end if

end % end for loop
```

Tutorial 2.1 1b

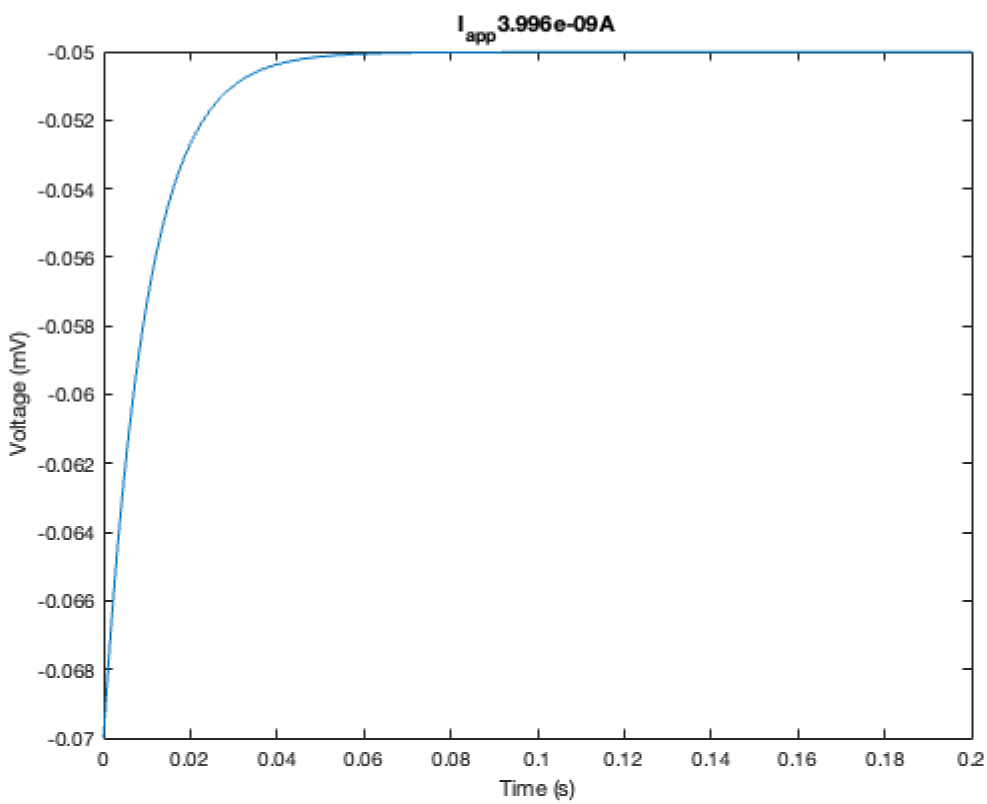
```
% Calculate I_th using equation by hand
% I_th = G_L(V_th - E_L)
% I_th = 1/R_m(V_th - E_L) = 1/5e6(-50e-3 + 70e-3) = 4e-9 amperes
% I_th = 4 nano amperes

% Calculate the threshold current I_th
I_th = (V_th - E_L) / R_m; % in A

% Function to simulate neuron behavior
function V = simulate_neuron(V, I_app, t, E_L, R_m, C_m, V_th, V_reset,
delta_t)
    % Integrate over time
    for k = 2:length(t)
        dVdt = (1/C_m) * (((E_L - V(k-1)) / R_m) + I_app(k));
        V(k) = V(k-1) + dVdt * delta_t;
        % Check for spikes
        if (V(k) > V_th)
            V(k) = V_reset;
```

```
        end
    end
end

% Firing: at I_th
I_app_temp(1:2000) = I_th * 1.001; % set 200 ms threshold
disp("I_app required for a spike: ")
disp(I_th)
% generate V vector using Euler's method
V = simulate_neuron(V, I_app_temp, t, E_L, R_m, C_m, V_th, V_reset, delta_t);

figure
plot(t(1:2000), V(1:2000))
hold on
xlabel("Time (s)")
ylabel("Voltage (mV)")
title(strcat("I_{app}", num2str(I_th*1.001), "A"))
hold off

% No Firing: slightly below I_th
I_app_temp(1:2000) = I_th * 0.999; % slightly less than I_th
V = simulate_neuron(V, I_app_temp, t, E_L, R_m, C_m, V_th, V_reset, delta_t);

figure
plot(t(1:2000), V(1:2000))
hold on
xlabel("Time (s)")
ylabel("Voltage (mV)")
title(strcat("I_{app}", num2str(I_th*0.999), "A"))
hold off

% Firing: slightly above I_th
I_app_temp(1:2000) = I_th * 1.01; % slightly more than I_th
V = simulate_neuron(V, I_app_temp, t, E_L, R_m, C_m, V_th, V_reset, delta_t);

figure
plot(t(1:2000), V(1:2000))
hold on
xlabel("Time (s)")
ylabel("Voltage (mV)")
title(strcat("I_{app}", num2str(I_th*1.01), "A"))
hold off
```

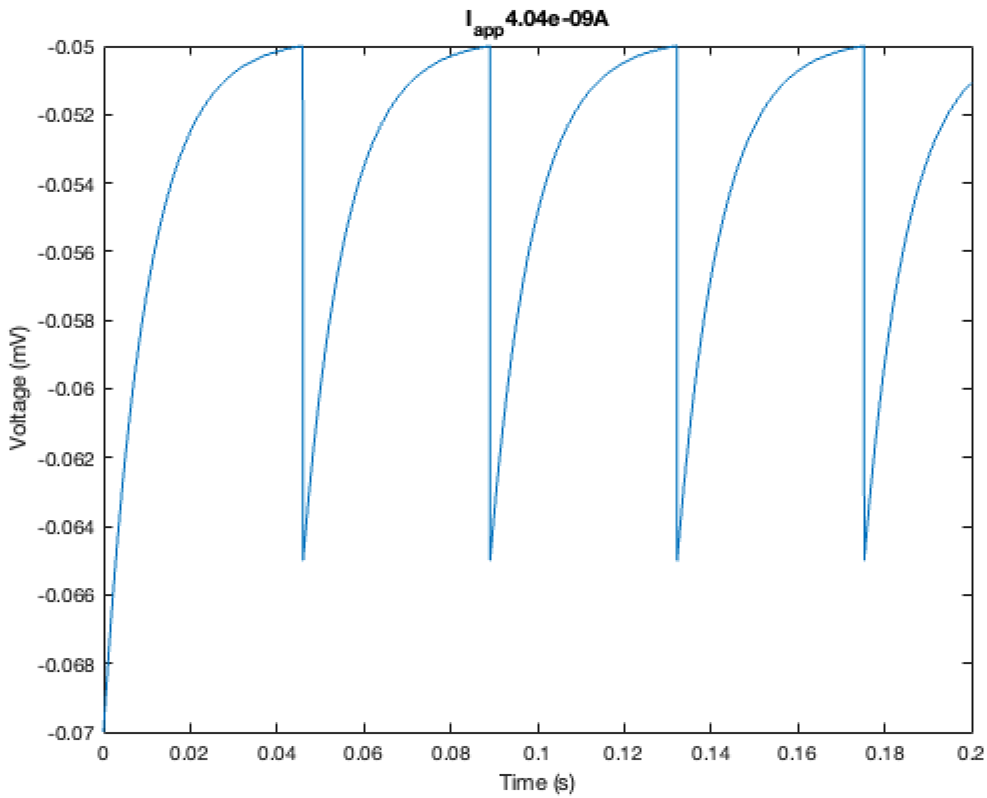
```
I_app required for a spike:
   4.0000e-09
```



I_{app} 4.004e-09A
Voltage (mV)
Time (s)
-0.05
-0.052
-0.054
-0.056
-0.058
-0.06
-0.062
-0.064
-0.066
-0.068
-0.07
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2



I_{app} 3.996e-09A
Voltage (mV)
Time (s)
-0.05
-0.052
-0.054
-0.056
-0.058
-0.06
-0.062
-0.064
-0.066
-0.068
-0.07
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2

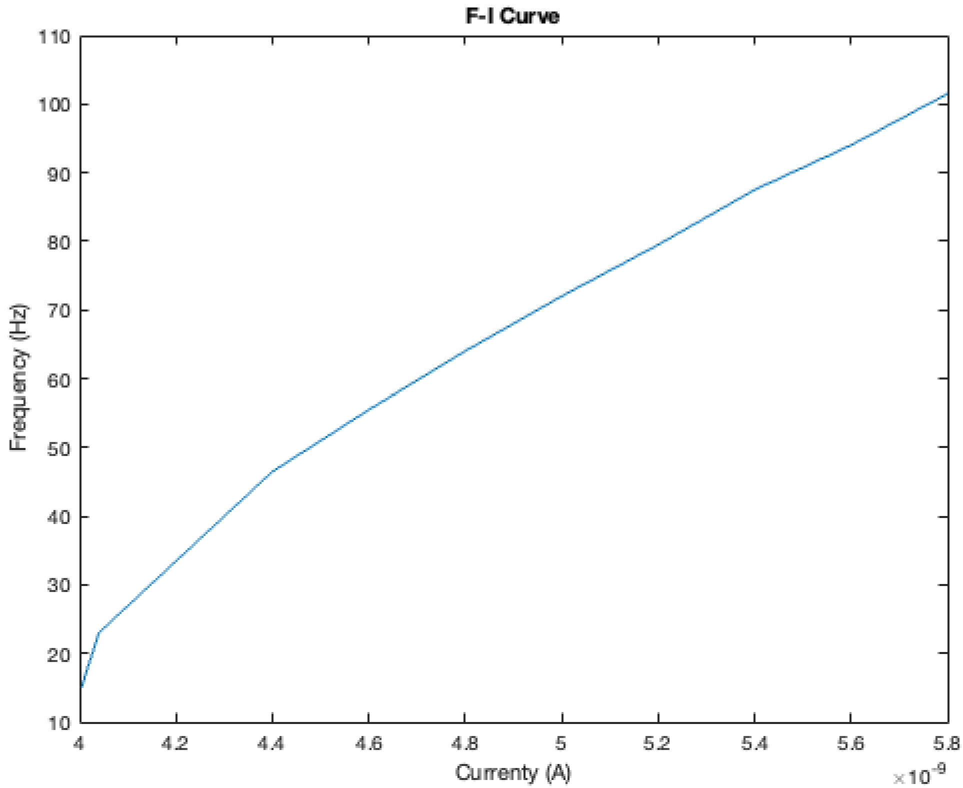


Tutorial 2.1 1c

Make another for .. end loop to use at least 10 different values of I_app, one value for each 2 s simulation (a "trial") such that the average firing rate (f) varies in the range from 0 to 100 Hz. Plot the resulting firing rate as a function of injected current (called the firing-rate curve or f-I curve).

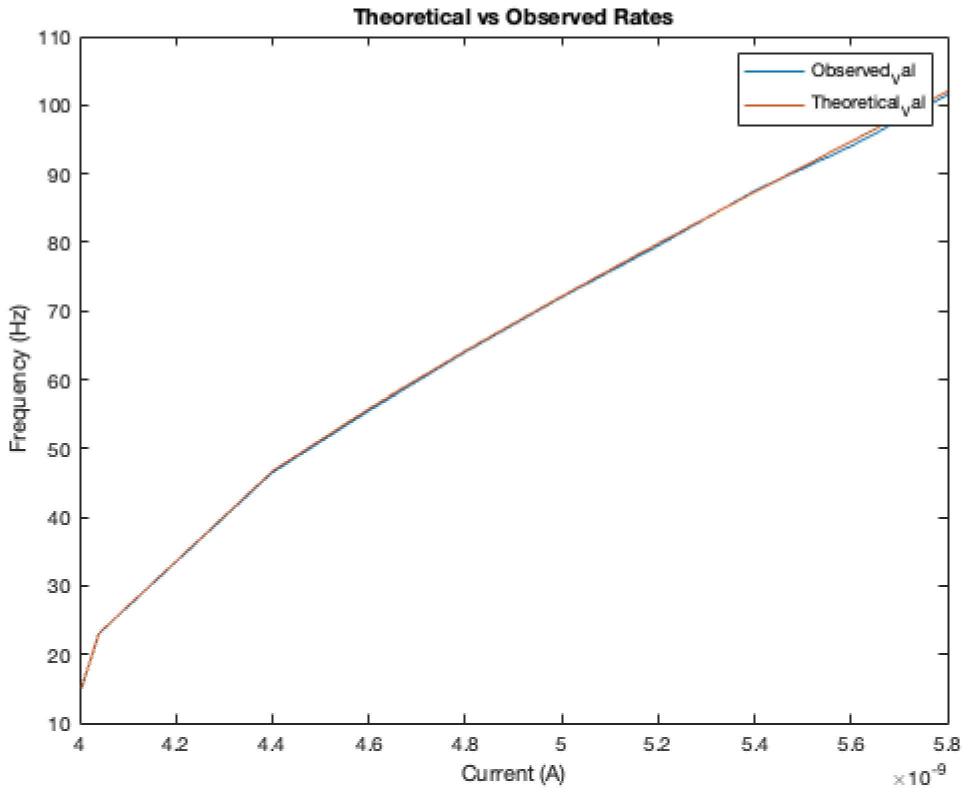
```
% set current ranges
I_app_values = I_th * [1.001, 1.01, 1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4,
1.45];  % Create 10 equally spaced I_app values
obs_val = zeros(size(I_app_values));

for k = 1:length(I_app_values)
    % I is constant
    I_app_temp(:) = I_app_values(k);
    V = simulate_neuron(V, I_app_temp, t, E_L, R_m, C_m, V_th, V_reset,
delta_t);
    % count firing
    obs_val(k) = sum(V == V_reset) / 2;
end

% plot f-i curve
figure
plot(I_app_values, obs_val)
hold on
title("F-I Curve")
xlabel("Currenty (A)")
```

```
ylabel("Frequency (Hz)")
hold off
```



Tutorial 2.1 1d

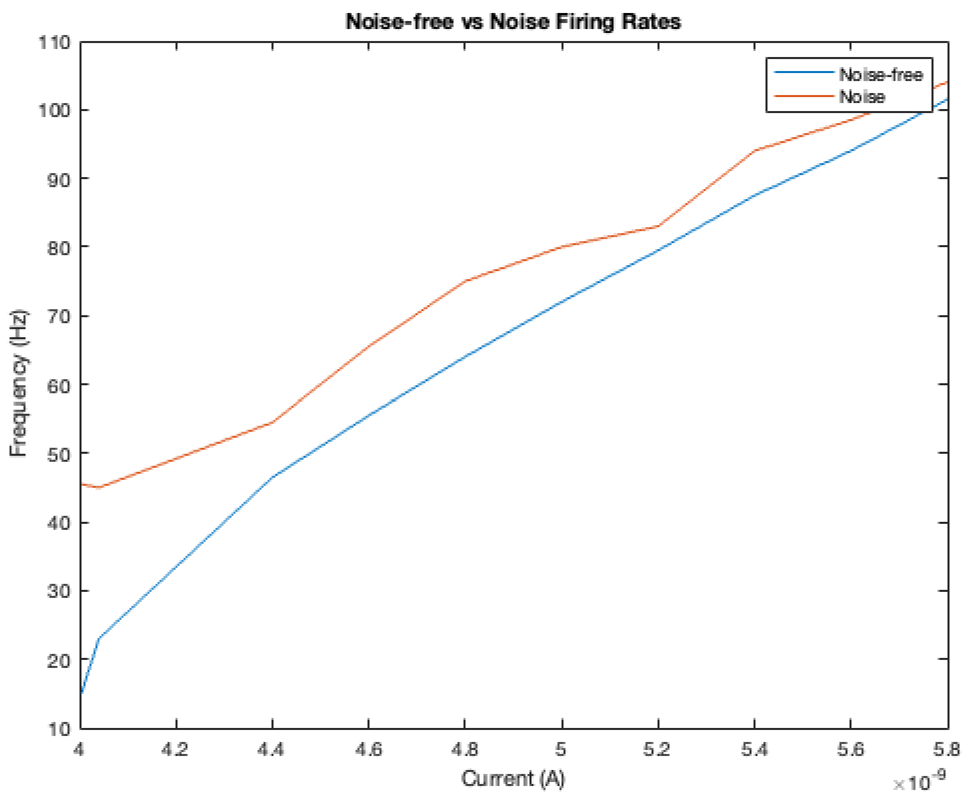
```
thr_val = zeros(size(I_app_values));

for k = 1:length(I_app_values)
    I_const = I_app_values(k);
    % input formula
    thr_val(k) = 1 / (C_m * R_m * log((E_L + I_const * R_m - V_reset) ...
        / (E_L + I_const * R_m - V_th)));
end

% Plotting both the simulated and theoretical firing rates
figure;
plot(I_app_values, obs_val)
hold on
plot(I_app_values, thr_val)
legend('Observed_Val', "Theoretical_Val")
title("Theoretical vs Observed Rates")
xlabel("Current (A)")
ylabel("Frequency (Hz)")
hold off
```



Tutorial 2.1 2a

```
% Modify the simulate_neuron function to include a noise term
function V = simulate_neuron_noise(V, I_app, t, E_L, R_m, C_m, V_th, V_reset,
delta_t, sigma_I)
    % set noise vector
    noise_vec = randn(size(t)) * sigma_I * sqrt(delta_t);
    % Integrate over time
    for k = 2:length(t)
        dVdt = (1/C_m) * (((E_L - V(k-1)) / R_m) + I_app(k));
        V(k) = V(k-1) + (dVdt * delta_t) + noise_vec(k);

        % Check for spikes
        if (V(k) > V_th)
            V(k) = V_reset;
        end
    end
end

sigma_I = 0.05;  % Standard deviation of the current noise
obs_val_noise_2a = zeros(size(I_app_values));

for k = 1:length(I_app_values)
    I_app_temp(:) = I_app_values(k);
    V = simulate_neuron_noise(V, I_app_temp, t, E_L, R_m, C_m, V_th, ...
```

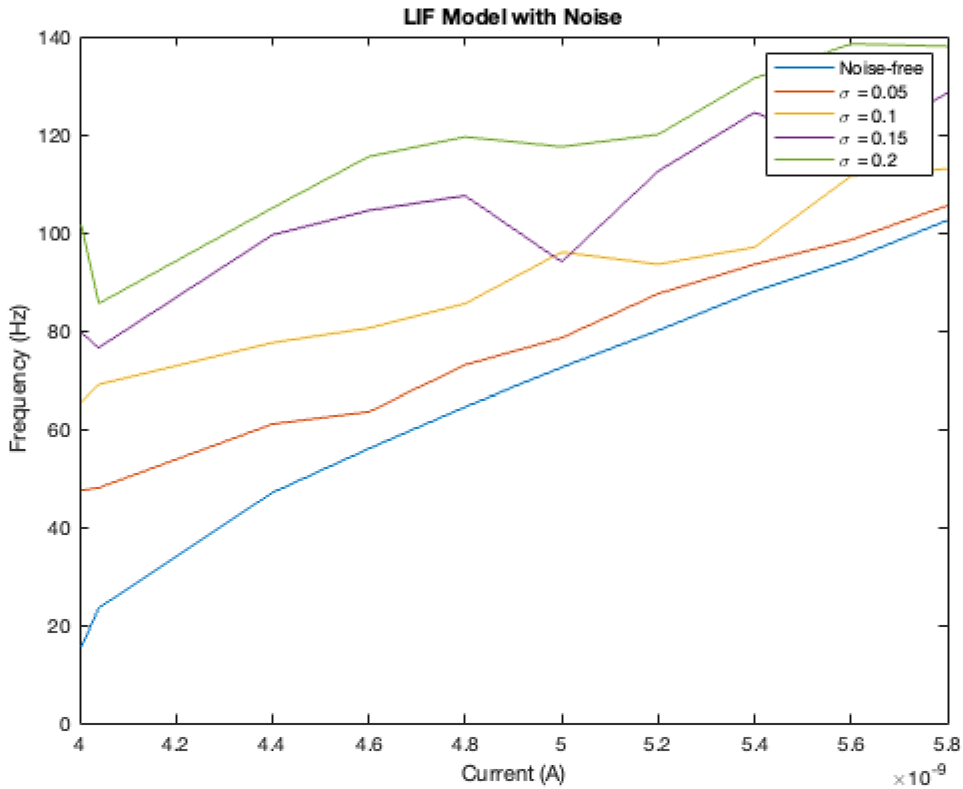
```
        V_reset, delta_t, sigma_I);
    obs_val_noise_2a(k) = (sum(V == V_reset) / 2);
end

% Plotting the membrane potential with noise
figure;
plot(I_app_values, obs_val)
hold on
plot(I_app_values, obs_val_noise_2a)
legend("Noise-free", "Noise")
title("Noise-free vs Noise Firing Rates")
xlabel("Current (A)")
ylabel("Frequency (Hz)")
hold off
```



Tutorial 2.1 2b

```
V = zeros(size(t));
I_app_temp = zeros(size(t));

sigma_I = [0, 0.05, 0.1, 0.15, 0.2];
obs_val_noise_2b = zeros(5, length(I_app_values));

for k = 1:length(sigma_I)
    for i = 1:length(I_app_values)
        I_app_temp(:) = I_app_values(i);
```

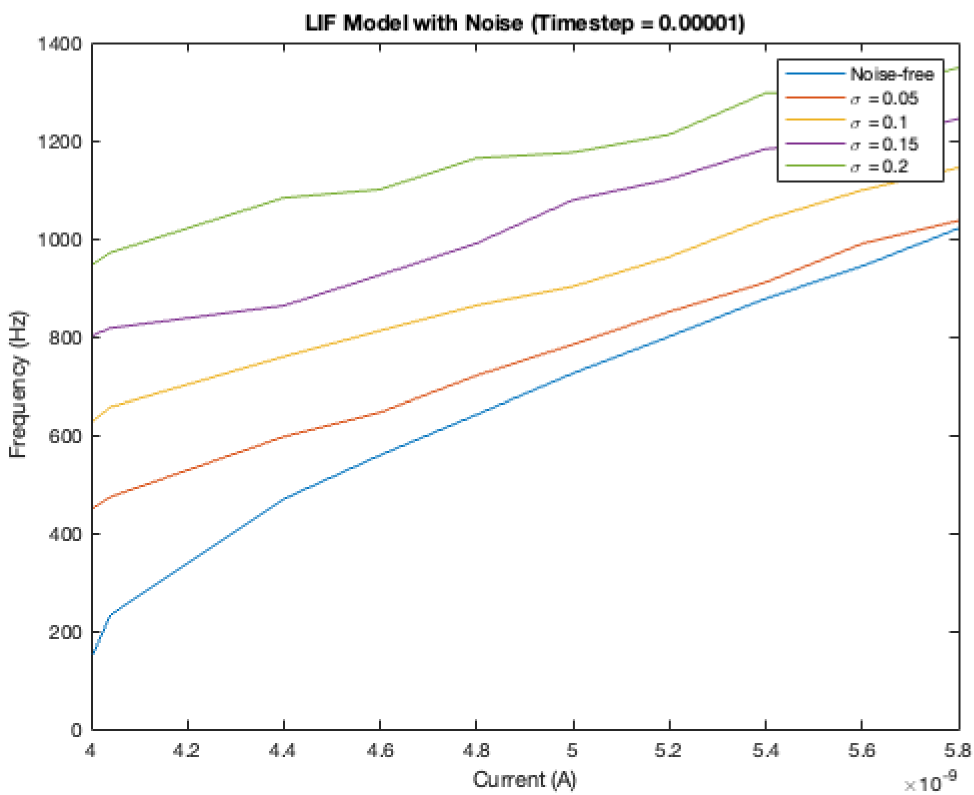
```
        V = simulate_neuron_noise(V, I_app_temp, t, E_L, R_m, C_m, V_th,
V_reset, delta_t, sigma_I(k));
        obs_val_noise_2b(k,i) = sum(V == V_reset) / 2;
    end
end

for k = 1:length(sigma_I)
    plot(I_app_values, obs_val_noise_2b(k,:))
    hold on
end
legend("Noise-free", "\sigma = 0.05", "\sigma = 0.1", "\sigma = 0.15",
"\sigma = 0.2")
title("LIF Model with Noise")
xlabel("Current (A)")
ylabel("Frequency (Hz)")
hold off

% Explanation:
% This code simulates the neuronal response over a range of I_app values for
two levels of noise
% sigma_I. By plotting the firing-rate curves for each level of noise, we
observe how noise influences
% neuronal firing. Higher sigma_I values typically cause more variability in
the membrane potential,
% which can lead to either more frequent spiking at lower thresholds due to
random depolarizations
% exceeding the spike threshold or erratic behavior that inhibits consistent
spiking. This plot will
% help visualize such effects, showing the transition from deterministic to
stochastic responses as noise increases.
```



Tutorial 2.1 2c

```
dt = 0.00001;
t = 0:dt:2;
V = zeros(size(t));
I_app_temp = zeros(size(t));

obs_val_noise_exp = zeros(5, length(I_app_values));

for k = 1:length(sigma_I)
    for i = 1:length(I_app_values)
        I_app_temp(:) = I_app_values(i);

        V = simulate_neuron_noise(V, I_app_temp, t, E_L, R_m, C_m, V_th,
V_reset, delta_t, sigma_I(k));
        obs_val_noise_exp(k,i) = sum(V == V_reset) / 2;
    end
end

for k = 1:length(sigma_I)
    plot(I_app_values, obs_val_noise_exp(k,:))
    hold on
end
legend("Noise-free", "\sigma = 0.05", "\sigma = 0.1", "\sigma = 0.15",
"\sigma = 0.2")
```

```
title("LIF Model with Noise (Timestep = 0.00001)")
xlabel("Current (A)")
ylabel("Frequency (Hz)")
hold off

% Explanation:
% This code segment tests the sensitivity of the neuron model to changes in
the timestep (dt).
% By reducing dt by a factor of ten, we increase the resolution of the
integration, which can potentially
% lead to more accurate calculations of membrane potential changes and spike
detection. This finer temporal
% granularity might reveal dynamics that were not captured with the larger
dt, such as more accurate spike timing
% and responses to fast changes in input current. Comparing the results with
the original dt will show if
% significant differences occur, indicating the influence of dt on the
simulation outcomes.
```



Published with MATLAB® R2024a