Table of Contents

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.
```

Tutorial 2.2 Functions

```
% Tutorial 2.2 Q1: Forced Voltage Clamp
function [ fr, v_mean ] = fvc_model(applied_curr, time_ms, plot_ts)

    % Define Parameters
    E_l = -0.070;           % -70 mV
    R_m = 10e7;             % 100 M Ohms
    C_m = 100e-12;          % 0.1 nF
    V_th = -0.050;          % -50 mV
    V_reset = -0.065;       % -65 mV
    G_l = 1/R_m;            % Inverse of Resistance
    I_app = applied_curr;   % in e-12
    t_ref = .0025;          % 2.5 ms
    V_peak = 0.050;         % 50 mV

    % Time Vectors
    t_max = 2; % 2 s, 2000 ms
    dt = 0.0001; % 0.1 ms steps
    t = 0:dt:t_max; % Range from 0 to 2000 ms in 0.1 ms increments

    v = zeros(size(t));
    v(1) = E_l;
```

```
    i_app = zeros(size(t)) + I_app;
    spikes = zeros(size(t));

    last_spike = -t_ref*10;   % Ensure simulation does not commence in a
                                % refractory period

    for i = 2:length(t)
        dvdt = i_app(i) + G_l * (E_l - v(i-1));
        v(i) = v(i-1) + (dt * dvdt / C_m);

        if ( t(i) < last_spike + t_ref )  % If within tref of last spike
            v(i) = V_reset;                % Clamp voltage to reset
        end

        if v(i) > V_th
            v(i) = V_reset;
            last_spike = t(i);
            spikes(i) = i;
        end
    end

    % Convert V whenever spike occurred to V_peak to sim spiking
    spikes = nonzeros(spikes)';
    v(spikes) = V_peak;

    % Calculate Firing Rate by dividing length of spikes by seconds (2)
    fr = length(spikes) / 2;
    v_mean = mean(v);

    % Plot membrane potential across time given
    if plot_ts
        % Scale time_ms to simulation steps (0.1 ms)
        timesteps = time_ms * 10;
        plot(t(1:timesteps), v(1:timesteps));
        xlabel('Time (ms)');
        ylabel('Membrane Potential (V)');
    end
end


% Tutorial 2.2 Q2: Threshold increase
function [ fr, v_mean ] = ti_model(applied_curr, time_ms, plot_ts)

    % Define Parameters
    E_l = -0.070;           % -70 mV
    R_m = 10e7;             % 100 M Ohms
    C_m = 100e-12;          % 0.1 nF
    V_th0 = -0.050;          % -50 mV
    V_reset = -0.065;       % -65 mV
    G_l = 1/R_m;            % Inverse of Resistance
    I_app = applied_curr;   % in e-12
    t_vth = .001;           % 1 ms
    V_peak = 0.050;         % 50 mV
    V_thmax = 0.200;        % 200 mv
```

```
    % Time Vector
    t_max = 2; % 2 s, 2000 ms
    dt = 0.0001; % 0.1 ms steps
    t = 0:dt:t_max; % Range from 0 to 2000 ms in 0.1 ms increments

    v = zeros(size(t));     % Initialize membrane potential vector
    v(1) = E_l;

    vt = zeros(size(t));    % Initialize membrane potential threshold vector
    vt(1) = V_th0;

    i_app = zeros(size(t)) + I_app;
    spikes = zeros(size(t));

    for i = 2:length(t)
        % Update Membrane Potential
        dvdt = i_app(i) + G_l * (E_l - v(i-1));
        v(i) = v(i-1) + (dt*dvdt/C_m);

        % Update Membrane Potential Threshold
        dvthdt = (V_th0 - vt(i-1)) / t_vth;
        vt(i) = vt(i-1) + (dt*dvthdt);

        if v(i) > vt(i)
            v(i) = V_reset;
            vt(i) = V_thmax;
            spikes(i) = i;
        end
    end

    % Convert V whenever spike occurred to V_peak to sim spiking
    spikes = nonzeros(spikes)';
    v(spikes) = V_peak;

    % Calculate Firing Rate by dividing length of spikes by seconds (2)
    fr = length(spikes) / 2;
    v_mean = mean(v);

    % Plot membrane potential across time given
    if plot_ts
        % Scale time_ms to simulation steps (0.1 ms)
        timesteps = time_ms * 10;
        plot(t(1:timesteps), v(1:timesteps));
        xlabel('Time (ms)');
        ylabel('Membrane Potential (V)');
    end
end


% Tutorial 2.2 Q3: Refractory Conductance with Threshold Increase
function [ fr, v_mean ] = rc_model(applied_curr, time_ms, plot_ts)

    % Define Parameters
```

```
E_l = -0.070;           % -70 mV
R_m = 10e7;             % 100 M Ohms
C_m = 100e-12;          % 0.1 nF
V_th0 = -0.050;         % -50 mV
G_l = 1/R_m;            % Inverse of Resistance
I_app = applied_curr;   % in e-12
t_vth = .001;
V_peak = 0.050;         % 50 mV
V_thmax = 0.200;        % 200 mv

E_k = -0.080;           % -80 mV
t_gref = 0.0002;         % 0.2 ms
delta_g = 2e-6;         % 2 microSiemens

% Time Vectors
t_max = 2; % 2 s, 2000 ms
dt = 0.0001; % 0.1 ms steps
t = 0:dt:t_max; % Range from 0 to 2000 ms in 0.1 ms increments

v = zeros(size(t));     % Initialize membrane potential vector
v(1) = E_l;

vt = zeros(size(t));    % Initialize membrane potential threshold vector
vt(1) = V_th0;

g_ref = zeros(size(t));
g_ref(1) = 0;

i_app = zeros(size(t)) + I_app;
spikes = zeros(size(t));

for i = 2:length(t)
    % Update Potassium Refractory Conductance
    g_ref(i) = g_ref(i-1)*exp(-dt/t_gref);

    % Update Membrane Potential Threshold
    dvthdt = (V_th0 - vt(i-1)) / t_vth;
    vt(i) = vt(i-1) + (dt*dvthdt);

    Vss = ( i_app(i) + G_l*E_l + g_ref(i)*E_k)/(G_l + g_ref(i));
    taueff = C_m/(G_l + g_ref(i));

    v(i) = Vss + ( v(i-1)-Vss)*exp(-dt/taueff);

    % Spike conditional
    if v(i) > vt(i)
        vt(i) = V_thmax;
        g_ref(i) = g_ref(i) + delta_g;
        spikes(i) = i;
    end
end

% Convert V whenever spike occurred to V_peak to sim spiking
spikes = nonzeros(spikes);
```

```
    v(spikes) = V_peak;

    % Calculate Firing Rate by dividing length of spikes by seconds (2)
    fr = length(spikes) / 2;
    v_mean = mean(v);

    % Plot membrane potential across time given
    if plot_ts
        % Scale time_ms to simulation steps (0.1 ms)
        timesteps = time_ms * 10;
        plot(t(1:timesteps), v(1:timesteps));
        xlabel('Time (ms)');
        ylabel('Membrane Potential (V)');
    end
end
```

Tutorial 2.2 Q1, Q2, Q3 Plots and Explanation

```
I_app = 100:600;        % Range of applied currents
I_app = I_app * 1e-12;  % in e-12

% Q1 vectors Forced Voltage Clamp model
fr_vec_q1 = zeros(size(I_app));
vm_vec_q1 = zeros(size(I_app));

% Q2 vectors Threshold Increase Model
fr_vec_q2 = zeros(size(I_app));
vm_vec_q2 = zeros(size(I_app));

% Q3 vectors Refractory Conductance with Threshold Increase Model
fr_vec_q3 = zeros(size(I_app));
vm_vec_q3 = zeros(size(I_app));

% Obtain mean FR and Membrane Potential for Q1 model
for i = 1:length(I_app)
    [fr_vec_q1(i), vm_vec_q1(i)] = fvc_model(I_app(i), 1, false);
    [fr_vec_q2(i), vm_vec_q2(i)] = ti_model(I_app(i), 1, false);
    [fr_vec_q3(i), vm_vec_q3(i)] = rc_model(I_app(i), 1, false);
end

% F1: FR_IC - Mean Firing rate as a Function of Input Current
figure;
hold on;
plot(I_app, fr_vec_q1);
plot(I_app, fr_vec_q2);
plot(I_app, fr_vec_q3);
xlabel('I_{App} (A)');
ylabel('Firing Rate (Hz)');
title('Mean Firing Rate vs Input Current');
legend('FVC', 'TI', 'RC', 'Location', [.15 .82 .1 .1]);

% F2: MP_IC - Mean Membrane Potential as a Function of Input Current
figure;
```

```
hold on;
plot(I_app, vm_vec_q1);
plot(I_app, vm_vec_q2);
plot(I_app, vm_vec_q3)
xlabel('I_{App} (A)');
ylabel('Membrane Potential (V)');
title('Mean Membrane Potential vs Input Current');
legend('FVC', 'TI', 'RC');

% F3: MP_FR - Mean Membrane Potential as a Function of Firing Rate
figure;
hold on;
plot(fr_vec_q1, vm_vec_q1);
plot(fr_vec_q2, vm_vec_q2);
plot(fr_vec_q3, vm_vec_q3);
xlabel('Firing Rate (Hz)');
ylabel('Membrane Potential (V)');
title('Mean Membrane Potential vs Mean Firing Rate');
legend('FVC', 'TI', 'RC');

% Explanation: Explain any trends and any nonmonotonic behavior, as well as
% the differences between the curves in each figure

% Mean firing rate as a function of the input current:
% The mean firing rate typically increases as the input current increases.
% This is due to the fact that higher input currents more frequently or
% easily bring the membrane potential to the threshold level necessary for
% firing a spike. Initially, there is a threshold level of current below
% which no spikes occur (subthreshold currents), followed by a linear or
% exponential increase in firing rate as the current surpasses this
% threshold. Nonmonotonic behavior could be due to factors like adaptation
% or other dynamic mechanisms within the neuron model that affect how the
% neuron responds to prolonged or high levels of input current. As I_app
% increases, the Mean Firing Rate of TI Model increases the fastest, then
% FVC Model, then RC Model.

% Mean membrane potential as a function of the input current:
% The mean membrane potential typically becomes more depolarized (less
% negative) as input current increases. This is because higher currents
% drive the membrane potential closer to or above the threshold more
% frequently or maintain it at a higher baseline between spikes.
% Nonmonotonic behavior could occur if the neuron model includes mechanisms
% like voltage-gated ion channels, which could introduce complex dynamics
% such as bistability or threshold variability based on the current or
% past membrane potentials. As I_app increases, the mean membrane potential
% of all three models first increase together to -0.05V until I_app reaches
% 200 pA, then the mean membrane potential for all three models decreases
% togeher, then TI Model started increasing first, then FVC model, then RC
% Model.

% Mean membrane potential as a function of firing rate:
% The membrane potential to be more depolarized (higher) at higher firing
% rates. This reflects that to sustain a higher firing rate, the neuron's
% membrane potential must be reaching the threshold more frequently,
```

```
% indicating it remains relatively high on average. Nonmonotonic behaviors
% could be introduced if adaptation or other feedback mechanisms affect the
% neuron. For instance, in some neuron models, after a spike, mechanisms
% might hyperpolarize the neuron or increase the threshold, temporarily
% reducing the membrane potential even if the firing rate is high. When
% Firing Rate = 0 Hz, the mean membrane potential for all three models are
% at -50 mV. As the firing rate increases, the membrane potential first
% decreases then starts to gradually increase after around 100-200 Hz. The
% TI Model first starts to increase, then FVC Model, then RC Model.


% F4: MP_T - Q1 Membrane Potential at 220 pA and 600 pA across 100 ms
figure;
hold on;
fvc_model(220e-12, 100, true);
fvc_model(600e-12, 100, true);
xlabel('Time (S)');
ylabel('Membrane Potential (V)');
title('Forced Voltage Clamp Model Simulation at 220 pA and 600 pA');
legend('220 pA', '600 pA');

% F5: MP_T - Q2 Membrane Potential at 220 pA and 600 pA across 100 ms
figure;
hold on;
ti_model(220e-12, 100, true);
ti_model(600e-12, 100, true);
xlabel('Time (S)');
ylabel('Membrane Potential (V)');
title('Threshold Increase Model Simulation at 220 pA and 600 pA');
legend('220 pA', '600 pA');

% F6: MP_T - Q3 Membrane Potential at 220 pA and 600 pA across 100 ms
figure;
hold on;
rc_model(220e-12, 100, true);
rc_model(600e-12, 100, true);
xlabel('Time (S)');
ylabel('Membrane Potential (V)');
title('Refractory Conductance & Threshold Increase Model Simulation at 220 pA
and 600 pA');
legend('220 pA', '600 pA');
```

Tutorial 2.3 Q1 Functions

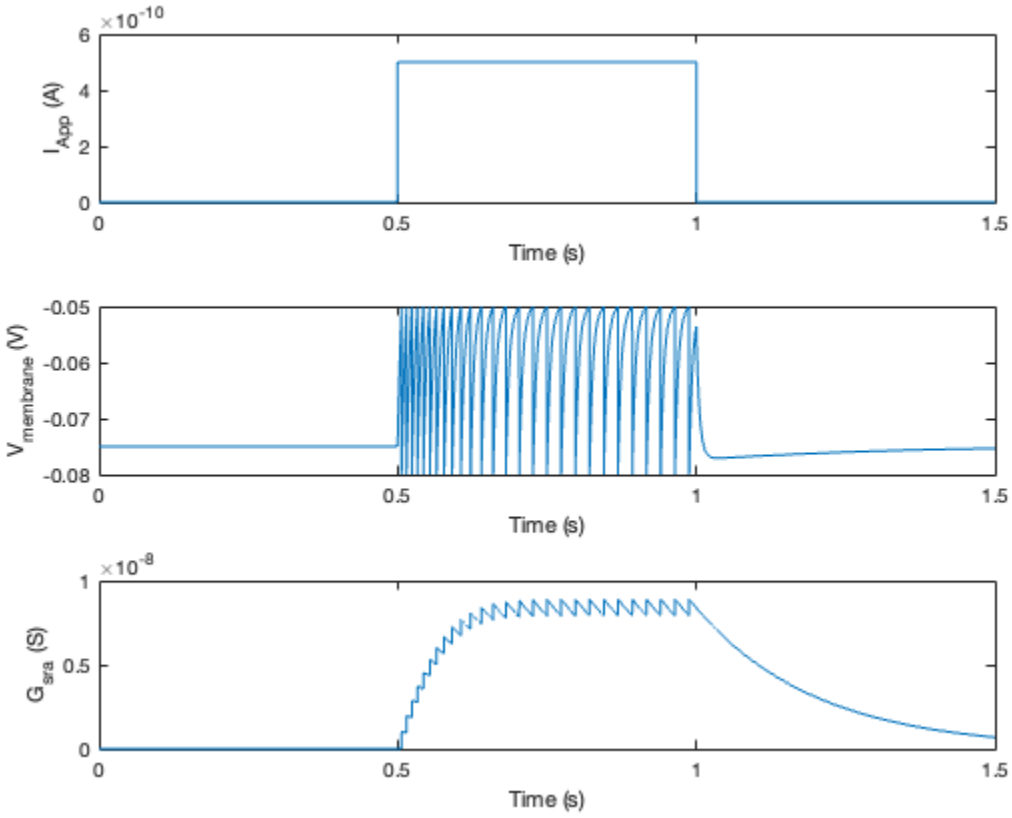
```
% Tutorial 2.3 Q1:

% Calculate firing rate given applied current and other parameters
function [fr] = calc_FR(I_app, C_m, G_l, E_l, V_th, V_reset)
    t_m = C_m* (1/G_l);
    fr = 1 / (t_m * log((E_l + I_app/G_l - V_reset) / (E_l + I_app/G_l -
V_th)));
end
```

```
% LIF model with adaptation function
% Simulate the LIF modelnueron with an adaptation current
% Function returns vectors of applied current, membrane potential, G_sra
% values, time, ISIs, as well as the raw firing rate w/o SRA.
function [I_app, v, G_sra, t_vec, fr, ISIs] = ada_LIF_model(current_pulse,
start_pulse_s, end_pulse_s, time_s)

    % Initialize Parameters
    E_l = -0.075;       % -75 mv
    V_th = -0.050;      % -50 mV
    V_reset = -0.080;   % -80 mV
    R_m = 100e6;        % 100 M Ohms
    C_m = 1e-10;        % 100 pF
    E_k = -0.080;       % -80 mV
    delta_G_sra = 1e-9; % 1 nS
    t_sra = 0.2;        % 200 ms
    G_l = 1/R_m;

    % Initialize time vector
    t_max = time_s;
    dt = 0.0001;
    t_vec = 0:dt:t_max;

    % Initialize other time-dependent vectors
    v = zeros(size(t_vec));
    I_app = zeros(size(t_vec));
    G_sra = zeros(size(t_vec));
    spikes = zeros(size(t_vec));

    % Set initial vector values
    v(1) = E_l;
    G_sra(1) = 0;

    % Alter the I_app vector to incorporate the parameterized pulse
    if start_pulse_s == 0
        start_pulse_s = 1/10000;
    end

    I_app((start_pulse_s*10000):(end_pulse_s*10000)) = current_pulse;

    % For loop to simulate adaptation model
    for i = 2:length(t_vec)
        % Update Potassium conductance term (G)
        dgdt = -G_sra(i-1) / t_sra;
        G_sra(i) = G_sra(i-1) + dt*dgdt;

        % Update membrane potential
        dvdt = (E_l - v(i-1))/R_m + G_sra(i-1)*(E_k - v(i-1)) + I_app(i);
        v(i) = v(i-1) + (dt*dvdt/C_m);

        % When the membrane potential is greater than the threshold, reset it,
        % increment the Potassium conductance term by delta G_sra, and record
        % the spike time
        if v(i) > V_th
```

```
            v(i) = V_reset;
            G_sra(i) = G_sra(i) + delta_G_sra;
            spikes(i) = t_vec(i);
        end
    end

    % Keep only the spiketimes from the spikes vector (remove zeros)
    spikes = nonzeros(spikes)';

    % If more than one spike, assign to the variable ISIs the difference
    % between each pair of spikes, yielding a vector of ISIs. If one or less,
    % record an ISIs of 0.
    if length(spikes) >= 2
        ISIs = diff(spikes);
    else
        ISIs = 0;
    end

    % Calculate Firing Rate
    fr = calc_FR(current_pulse, C_m, G_l, E_l, V_th, V_reset);
end
```

Tutorial 2.3 Q1a Plot

```
% Simulate the adaptation LIF model for 1.5 s with 500 pA applied from 0.5
% s until 1.0 s. Plot three subplots as a function of time
[I_app, v, G_sra, t_vec, ~, ~] = ada_LIF_model(500e-12, 0.5, 1.0, 1.5);

figure;

% Current as a function of time
subplot('Position', [.1 .75 .8 .2]);
plot(t_vec, I_app);
xlabel('Time (s)');
ylabel('I_{App} (A)');

% Membrane Potential as a function of time
subplot('Position', [.1 .425 .8 .2]);
plot(t_vec, v);
xlabel('Time (s)');
ylabel('V_{membrane} (V)');

% Adaptation conductance as a function of time
subplot('Position', [.1 .1 .8 .2]);
plot(t_vec, G_sra);
xlabel('Time (s)');
ylabel('G_{sra} (S)');
hold off;
```



Tutorial 2.3 Q1b Plot

```
% Define Parameters
R_m = 100e6;

% Used for loop below to find range of I_app values that yield firing rate
% of 0-50 Hz. Created 20 steps by subtracting them then dividing by 19.
I_range = 2.5:0.0247:2.97;
I_range = I_range*1e-10;

fr = zeros(size(I_range));
ISIs = zeros(size(I_range));
final_rate = zeros(size(I_range));

for i = 1:length(I_range)
    [~, ~, G_sra, ~, frate, isis] = ada_LIF_model(I_range(i), 0, 5, 5);

    % Obtain initial ISI
    ISIs(i) = 1/isis(1);

    % Obtain calculated firing rate (base rate without adaptation)
    if isreal(frate)
        fr(i) = frate;
    end
```

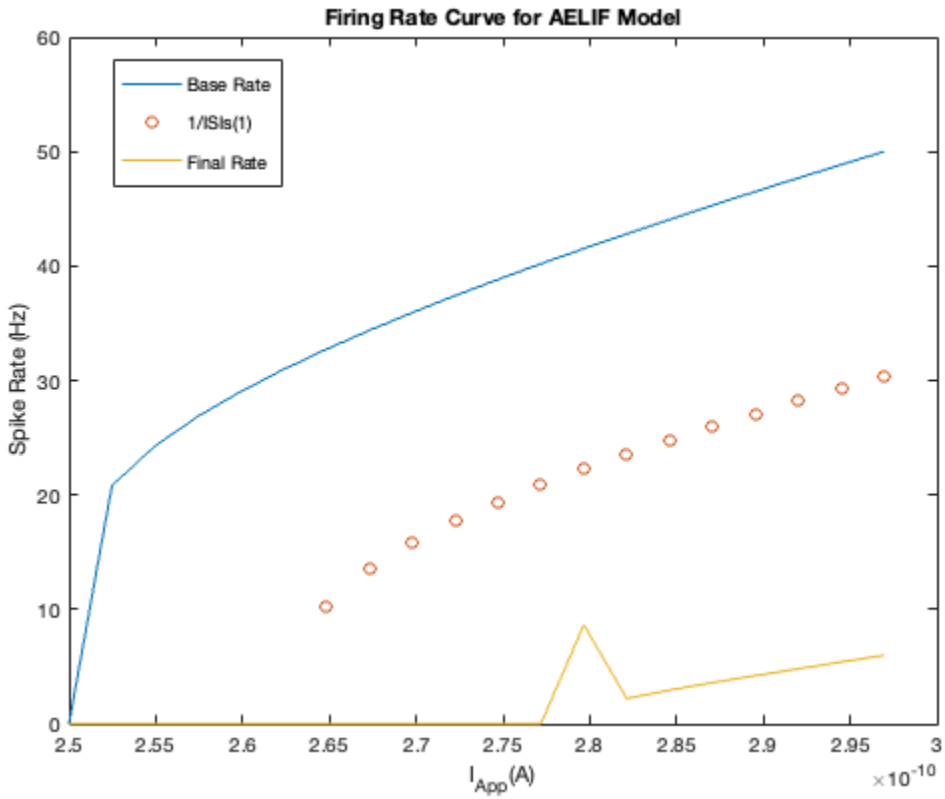
```
    % Obtain final simulated ISI
    if length(isis) > 1
        final_rate(i) = 1/isis(end);
    end
end

% Replace any final rate of Inf (inverse of 0 ISI) with 0
final_rate(final_rate == Inf) = 0;
ISIs(ISIs == Inf) = 0;

figure;
plot(I_range, fr);
hold on;
plot(I_range, ISIs);
plot(I_range, final_rate);
hold off;
xlabel('I_{App} (A)');
ylabel('Spike Rate (Hz)');
legend('Base Rate', '1/ISIs(1)', 'Final Rate');
title('Firing Rate Curve for LIF Model')

% Comment on the result:
% Base Rate: This line represents the minimum firing rate, starting at
% zero, which indicates that below a certain threshold of I_app, the
% neuron does not fire. This threshold is the point at which the input
% current just compensates for the leak currents, allowing the membrane
% potential to reach the threshold voltage necessary for firing a spike.
%
% 1/ISIs (Initial Spike Interval): This curve represents the inverse of the
% initial interspike interval. At lower applied currents (just above the
% firing threshold), the 1/ISI value is lower, indicating longer
% intervals between spikes. As I_app increases, this value generally
% increases, reflecting shorter times between the initial spikes as the
% current drives the membrane potential to the threshold more quickly.
%
% Final Rate: This line reflects the steady-state firing rate, which is the
% firing frequency after the neuron has adapted to the continuous
% application of I_app. Initially, at lower currents, this rate might
% closely follow the 1/ISI curve. However, as the current increases, the
% neuron may start to exhibit adaptation behaviors (not as pronounced in
% basic LIF models without explicit adaptation mechanisms), such as a
% saturation of firing rate due to the limits imposed by the membrane's
% time constant and refractory periods.
```

Tutorial 2.3 Q2 Functions

```
% Function to calculate desired I_app given Firing rate and other parameters

function [I_a] = calc_Iapp(fr, C_m, G_l, E_l, V_th, V_reset)
    t_m = C_m * (1/G_l);

    % I took the firing rate function (pg. 70) and solved for I_app. I broke
    % the solved equation down into three parts below.
```

```
    E = exp( 1 / (fr * t_m));
    G = G_l * (E*E_l - E*V_th - E_l + V_reset);
    I_a = G / (1 - E);
end


% AELIF Model function
% Simulate the AELIF neuron model.
% Function returns vectors of applied current, membrane potential, I_sra
% values, time, ISIs, as well as the raw firing rate w/o SRA.
function [I_app, v, I_sra, t_vec, fr, ISIs] = AELIF_model(current_pulse,
start_pulse_s, end_pulse_s, time_s)

    % Set parameters
    E_l = -0.075;           % -75 mV
    V_th = -0.050;          % -50 mV
    V_reset = -0.080;       % -80 mV
    V_max = 0.050;          % 50 mV
    delta_th = 0.002;       % 2 mV
    G_l = 10e-9;            % 10 nS
    C_m = 100e-12;          % 100 pF
    a = 2e-9;               % 2 nS
    b = 0.02e-9;            % 0.02 nA
    t_sra = 0.200;          % 200 ms

    % Initialize time vector
    t_max = time_s;
    dt = 0.0001;
    t_vec = 0:dt:t_max;

    % Initialize other time-dependent vectors
    v = zeros(size(t_vec));
    I_app = zeros(size(t_vec));
    I_sra = zeros(size(t_vec));
    spikes = zeros(size(t_vec));

    % Set initial vector values
    v(1) = E_l;
    I_sra(1) = 0;

    % Alter the I_app vector to incorporate the parameterized pulse
    if start_pulse_s == 0
        start_pulse_s = 1/10000;
    end

    I_app((start_pulse_s*10000):(end_pulse_s*10000)) = current_pulse;

    % For loop to simulate AELIF model for each time step
    for i = 2:length(t_vec)

        % Update membrane potential at each time step
        dvdt = G_l * (E_l - v(i-1) + delta_th*exp((v(i-1) - V_th)/delta_th))
- I_sra(i-1) + I_app(i);
        v(i) = v(i-1) + (dt*dvdt/C_m);
```

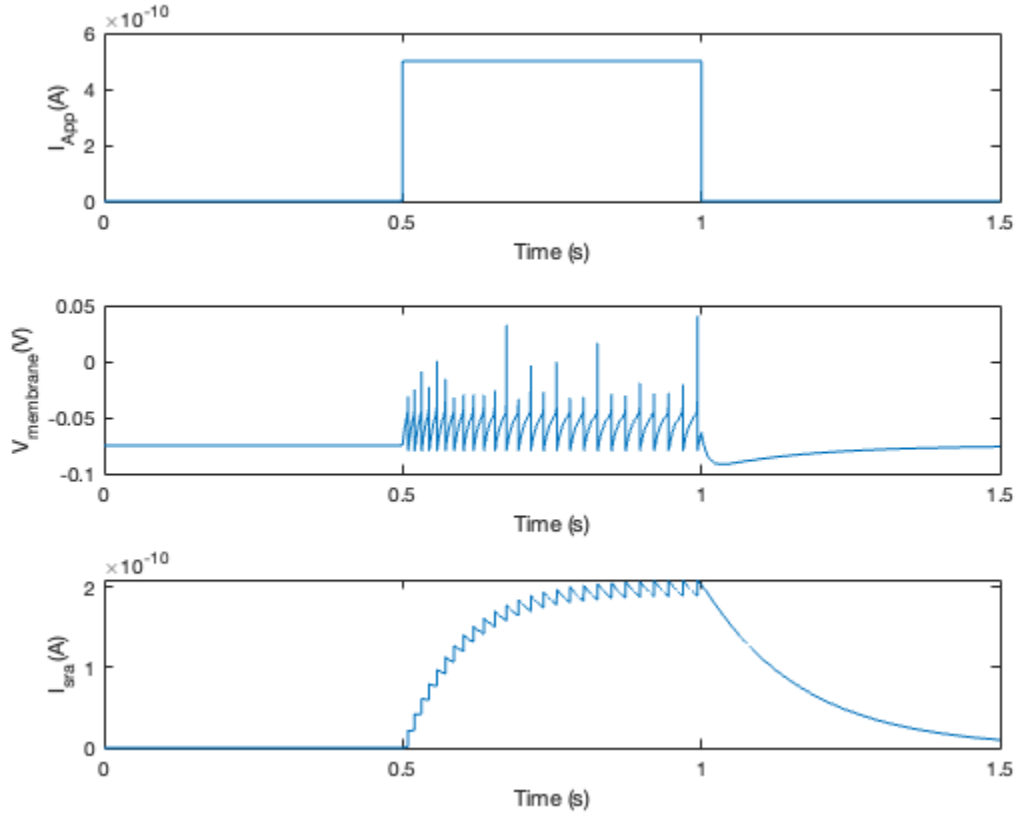
```
        % Update the I_sra at each time step so that it decays over time
        didt = a*(v(i-1) - E_l) - I_sra(i-1);
        I_sra(i) = I_sra(i-1) + (dt*didt/t_sra);

        % When the membrane potential reaches the max, reset membrane
        % potential, increase I_sra by constant b, record spike time
        if v(i) > V_max
            v(i) = V_reset;
            I_sra(i) = I_sra(i) + b;
            spikes(i) = t_vec(i);
        end
    end

    % Keep only the spiketimes from the spikes vector (remove zeros)
    spikes = nonzeros(spikes)';

    % If more than one spike, assign to the variable ISIs the difference
    % between each pair of spikes, yielding a vector of ISIs. If one or less,
    % record an ISIs of 0.
    if length(spikes) >= 2
        ISIs = diff(spikes);
    else
        ISIs = 0;
    end

    % Calculate raw firing rate w/o SRA
    fr = calc_FR(current_pulse, C_m, G_l, E_l, V_th, V_reset);
end
```



Tutorial 2.3 Q2a Plot

```
% Simulate  AELIF model neuron for 1.5 seconds, with a current pulse of
% I_app = 500 pA applied from 0.5 s until 1.0 s. Plot results in a graph,
% using 2 subplots:
%       1) Current as a function of time
%       2) Membrane potential as a function of time
[I_app, v, I_sra, t_vec, fr, firstISI] = AELIF_model(500e-12, 0.5, 1.0, 1.5);

figure;

% Current as a function of time
subplot('Position', [.1 .75 .8 .2]);
plot(t_vec, I_app);
xlabel('Time (s)');
ylabel('I_{App}(A)');

% Membrane Potential as a function of time
subplot('Position', [.1 .425 .8 .2]);
plot(t_vec, v);
xlabel('Time (s)');
ylabel('V_{membrane}(V)');

% I_sra as a function of time
subplot('Position', [.1 .1 .8 .2]);
```

```
plot(t_vec, I_sra);
xlabel('Time (s)');
ylabel('I_{sra}(A)');
hold off;
```



Tutorial 2.3 Q2b Plot

```
% Simulate the AELIF model for 5 seconds with a range of 20 different levels
% of constant applied current such that the steady state firing rate of the
% cell varies from 0 to 50 Hz.
%   - For each applied current calculate the first ISI and the ss ISI. On a
%   graph plot the inverse of the ss ISI against the applied current to
%   produce an f-1 curve. On the same graph plot as individual points the
%   inverse of the ISI.

% Calculate I_app such that firing rate equals 50 Hz, given parameters of
% the AELIF model. Then calculate again such that firing rate equals 0.
E_l = -0.075;           % -75 mV
V_th = -0.050;          % -50 mV
V_reset = -0.080;       % -80 mV
G_l = 10e-9;            % 10 nS
C_m = 100e-12;          % 100 pF
max_fr = 50;            % 50 Hz
min_fr = 0;             % 0 Hz

% Find applied current threshold (Max applied current that yields 0 Hz
```

```
% firing rate)
I_app_max = calc_Iapp(max_fr, C_m, G_l, E_l, V_th, V_reset);
I_threshold = G_l * (V_th - E_l);

% Create a 20-element vector that goes from minimum I_app to maximum I_app,
% such that the firing rate varies from 0 Hz to 50 Hz. First calculate the
% steps to use, then create the vector
I_step = (I_app_max - I_threshold) / 19;
I_app_vec = I_threshold:I_step:I_app_max;

% Initialize firing rate vector (inverse of steady state ISI) and Initial
% ISI vector(first ISI between the first and second of spikes)
fr_vec = zeros(size(I_app_vec));
ISI_i = zeros(size(I_app_vec));
ISI_f = zeros(size(I_app_vec));

% Loop through each I_app in the I_app vector and assign the firing rate
% and initial ISI to their respective vectors
for i = 1:length(I_app_vec)
    [I_app, v, I_sra, t_vec, fr_vec(i), ISIs] = AELIF_model(I_app_vec(i), 0,
5, 5);
    ISI_i(i) = 1/ISIs(1);

    % If more than one ISI, record final ISI
    if (length(ISIs) > 1)
        ISI_f(i) = 1/ISIs(end);
    end
end

% Plot the Firing rate and the Inverse of the initial ISI as a function of
% applied Current
figure;
% Firing rate as a function of applied current w/o SRA
plot(I_app_vec, fr_vec);
hold on;

% Initial ISI
scatter(I_app_vec, ISI_i);

% Final ISI
plot(I_app_vec, ISI_f);

% Label
xlabel('I_{App}(A)');
ylabel('Spike Rate (Hz)');
legend('Base Rate', '1/ISIs(1)', 'Final Rate', 'Location', [0.17 0.75 0.15
0.15]);
title('Firing Rate Curve for AELIF Model')
hold off;

% Comment on the result:
% Base Rate: Similar to the LIF model, the base rate for the AELIF model
% starts at zero and begins to increase as I_app crosses a threshold
% necessary for overcoming the exponential spike initiation dynamics.
```

```
% This threshold might be lower compared to LIF due to the exponential
% factor aiding the membrane potential in reaching the firing threshold
% more rapidly.
%
% 1/ISIs (Initial Spike Interval): In the AELIF model, the initial
% interspike intervals can decrease more sharply with increasing I_app
% due to the exponential component that accelerates the depolarization
% process as the membrane potential approaches the threshold. This results
% in a more pronounced decrease in ISI with increasing current.
%
% Final Rate: The final firing rate in the AELIF model can exhibit
% adaptation and saturation differently from the LIF model. The adaptation
% current in the AELIF model increases with each spike, which can
% significantly affect the neuron's ability to maintain high firing rates
% under continuous high current, leading to a notable saturation or even
% reduction in firing rate at high I_app levels. This adaptation helps
% prevent the runaway excitation that might occur due to the exponential
% term.
```

Published with MATLAB® R2024a