```
% Tutorial 2.3
% Created by Jonathan Lindbloom, 2/23/2019
% Email: jlindbloom@smu.edu

clear all
close all

% Define parameters.
global e_leak r_membrane c_membrane v_threshold v_reset e_k
 delta_g_sra tau_sra g_leak a delta_th b v_max
e_leak = -75e-3;
r_membrane = 100e6;
c_membrane = 100e-12;
g_leak = 10e-9;
v_threshold = -50e-3;
v_reset = -80e-3;
v_max = -37e-3;
e_k = -80e-3;
delta_g_sra = 1e-9;
tau_sra = 200e-3;
a = 2e-9;
delta_th = 2e-3;
b = 0.02e-9;

% Create time vector.
global dt
dt = 0.001;
t = 0:dt:1.5;

% Create membrane potential and conductance vectors.
v = zeros(1, length(t));
g = zeros(1, length(t));

% Initialize membrane potential and conductance vectors.
v(1) = e_leak;
g(1) = 0;
```

Question 1 %%%

```
% Create current vector for 1.a.
I_app = zeros(1, length(v));
index_start = int64(length(I_app)/3);
index_end = int64(length(I_app)*(2/3));
for i = 1:length(I_app)
   if i >= index_start
       if i <= index_end
           I_app(i) = 500e-12;
       end
   end
end

% Run simulation for 1.a.
```

```
[v1sim, g1sim] = q1sim(v, g, t, I_app);

% Plot figure for 1.a.
f1 = figure;
figure(f1);
subplot(3,1,1);
plot(t, I_app);
ylim([-1e-10 6e-10]);
ylabel('Applied Current (A)');
subplot(3,1,2);
plot(t, v1sim);
ylim([-0.085 -0.045]);
ylabel('Membrane Potential (V)');
subplot(3,1,3);
plot(t, g1sim);
ylim([-0.1e-8 1e-8]);
ylabel('Conductance (S)');
xlabel('Time (s)');

% Create time vector.
dt = 0.001;
t = 0:dt:5;

% Create membrane potential and conductance vectors.
v = zeros(1, length(t));
g = zeros(1, length(t));

% Initialize membrane potential and conductance vectors.
v(1) = e_leak;
g(1) = 0;

% Run simulations for varying appplied current.
I_var = 200e-12:20e-12:500e-12;
initial_rates = zeros(1, length(I_var));
steady_rates = zeros(1, length(I_var));
for i = 1:length(I_var)
    I_app = I_var(i)*ones(1, length(v));
    [vsim, gsim] = q1sim(v, g, t, I_app);
    search = search_potential(vsim, -80e-3);
    search = find(search);
    if length(search) == 0
        initial_rates(i) = 0;
        steady_rates(i) = 0;
    else
        ISI_first = (search(2) - search(1))*dt;
        initial_rates(i) = 1/ISI_first;
        ISI_last = (search(length(search)) -
 search(length(search)-1))*dt;
        steady_rates(i) = 1/ISI_last;
    end
end

f2 = figure;
figure(f2);
```

```
plot(I_var, steady_rates);
hold on;
scatter(I_var, initial_rates);
title('Spike Rate vs. I_{app}');
xlabel('I_{app} (A)');
ylabel('Spike Rate (Hz)');
legend('Final Rate', 'Initial Rate');

% Comments:
% As the applied current increases beyond the threshold, both the
 initial and steady firing
% rates increase - however, the initial firing rate increases much
 faster
% than the steady rate. This is due to the adaptation current term
 included
% in this LIF model, which causes the model to mimic spike-rate
 adaptation.
```

Question 2 %%%

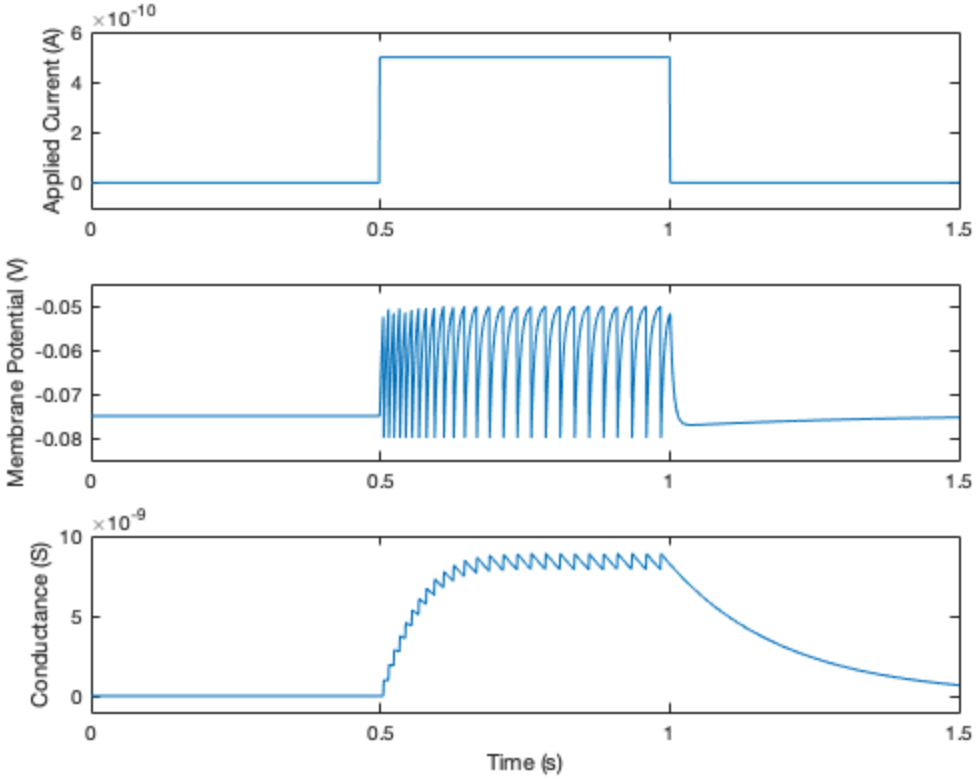
```
% Create time vector.
dt = 0.0001;
t = 0:dt:1.5;

% Create membrane potential and current vectors.
v = zeros(1, length(t));
i_sra = zeros(1, length(t));

% Initialize membrane potential and current vectors.
v(1) = e_leak;
i_sra(1) = 0;

% Create current vector for 2.a.
I_app = zeros(1, length(v));
index_start = int64(length(I_app)/3);
index_end = int64(length(I_app)*(2/3));
for i = 1:length(I_app)
   if i >= index_start
       if i <= index_end
           I_app(i) = 500e-12;
       end
   end
end

% Run simulation for 2.a.
[v2sim, i_sra2sim] = q2sim(v, i_sra, t, I_app);

% Plot figure for 2.a.
f3 = figure;
figure(f3);
subplot(3,1,1);
plot(t, I_app);
ylim([-1e-10 6e-10]);
```

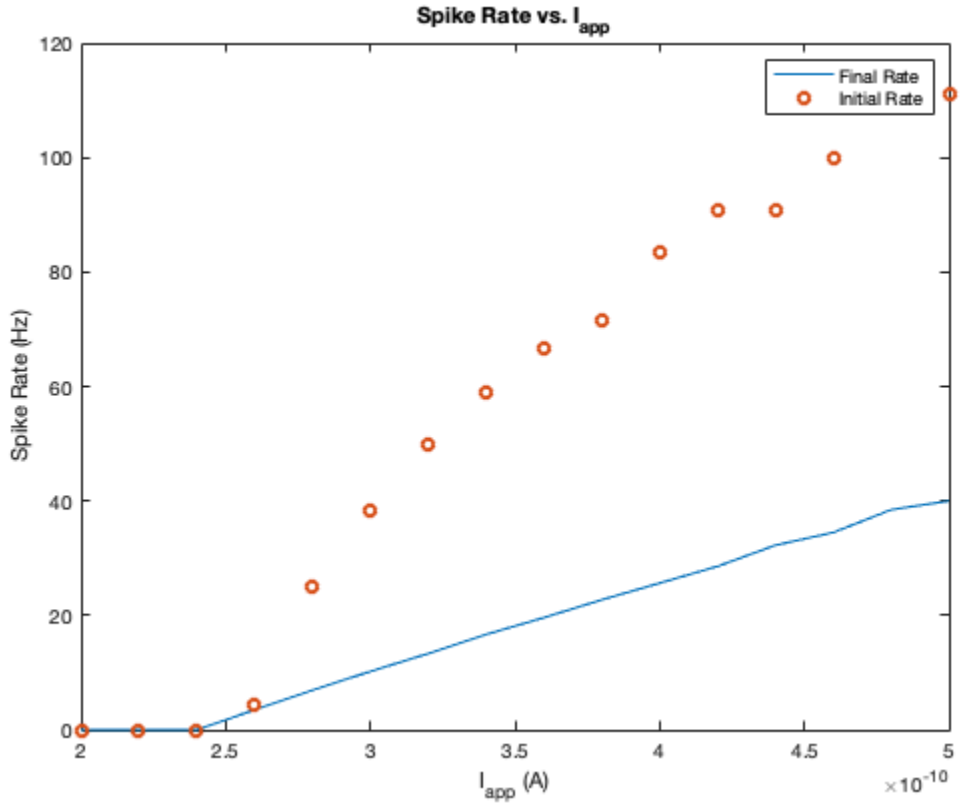
```
ylabel('Applied Current (A)');
subplot(3,1,2);
plot(t, v2sim);
ylabel('Membrane Potential (V)');
ylim([-0.1 -.03]);
subplot(3,1,3);
plot(t, i_sra2sim);
ylabel('I_{SRA} (A)');
ylim([-0.5e-10 2.5e-10]);


% Create time vector.
dt = 0.001;
t = 0:dt:5;

% Create membrane potential and current vectors.
v = zeros(1, length(t));
i_sra = zeros(1, length(t));

% Initialize membrane potential and current vectors.
v(1) = e_leak;
i_sra(1) = 0;

% Run simulations for varying appplied current.
v_max = -50e-3;
I_var = 200e-12:20e-12:500e-12;
initial_rates = zeros(1, length(I_var));
steady_rates = zeros(1, length(I_var));
for i = 1:length(I_var)
    I_app = I_var(i)*ones(1, length(v));
    [vsim, i_srasim] = q2sim(v, i_sra, t, I_app);
    search = search_potential(vsim, -80e-3);
    search = find(search);
    if (length(search) == 0) || (length(search) == 1)
        initial_rates(i) = 0;
        steady_rates(i) = 0;
    else
        ISI_first = (search(2) - search(1))*dt;
        initial_rates(i) = 1/ISI_first;
        ISI_last = (search(length(search)) -
 search(length(search)-1))*dt;
        steady_rates(i) = 1/ISI_last;
    end
end

f4 = figure;
figure(f4);
plot(I_var, steady_rates);
hold on;
scatter(I_var, initial_rates);
title('Spike Rate vs. I_{app}');
xlabel('I_{app} (A)');
ylabel('Spike Rate (Hz)');
legend('Final Rate', 'Initial Rate');
```

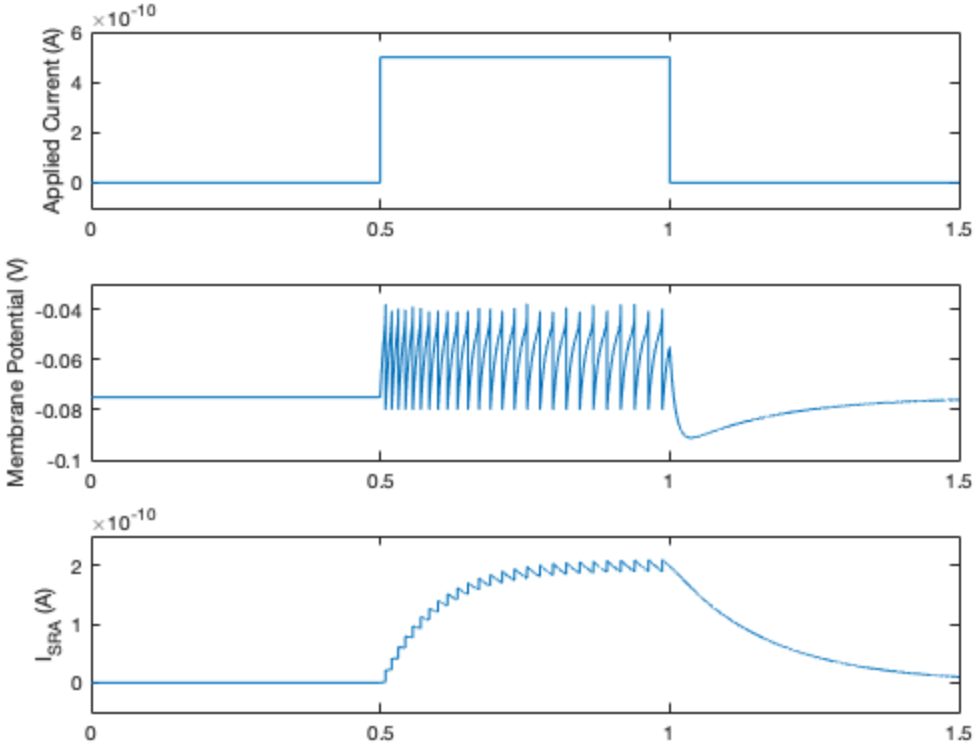
```
% Comments:
% This AELIF model is similar to the previous LIF model, except that
 it
% appears that a greater applied current is needed to induce spikes.
% Qualitatively, the f-I curves produced by the two models are
 extremely
% similar. The main difference in this AELIF model is that we see a
% variations of the threshold and points of inflection in the plots of
% membrane potential vs. time.


%%%%%%%%%%%%%%%%%%%
% Save all figures.
saveas(f1, 'LIF_with_adaptation_current.png');
saveas(f2, 'LIF_spike_rate_vs_applied_current.png');
saveas(f3, 'AELIF_with_adaptation_current.png');
saveas(f4, 'AELIF_spike_rate_vs_applied_current.png');


%%%%%%%%%%%%%%%%%%%%%%
% Function Definitions:

function value = F(v, g , t, i_applied)
global e_leak r_membrane c_membrane e_k
% Evaluates the RHS function of the ODE with given v, t, and I_app.
% Used for generating the RK4 solution.
value = ((e_leak - v)/(r_membrane*c_membrane)) + ((g*(e_k - v))/
c_membrane) + ((i_applied)/c_membrane);
end

function value = G(g)
global tau_sra
% Evaluates the RHS function of the conductance ODE with given g.
% Used for generating the RK4 solution.
value = -g/tau_sra;
end

function value = F2(v, i_sra , t, i_applied)
global e_leak c_membrane g_leak v_threshold delta_th
% Evaluates the RHS function of the ODE with given v, t, i_sra, and
 I_app.
% Used for generating the RK4 solution.
value = ( ( g_leak*((e_leak - v) + (delta_th*exp((v - v_threshold)/
delta_th)) ) ) -i_sra + i_applied)/c_membrane;
end

function value = G2(v, i_sra)
global tau_sra e_leak a
% Evaluates the RHS function of the conductance ODE with given v and
 i_sra.
% Used for generating the RK4 solution.
value = ((a*(v - e_leak)) - i_sra)/tau_sra;
end
```

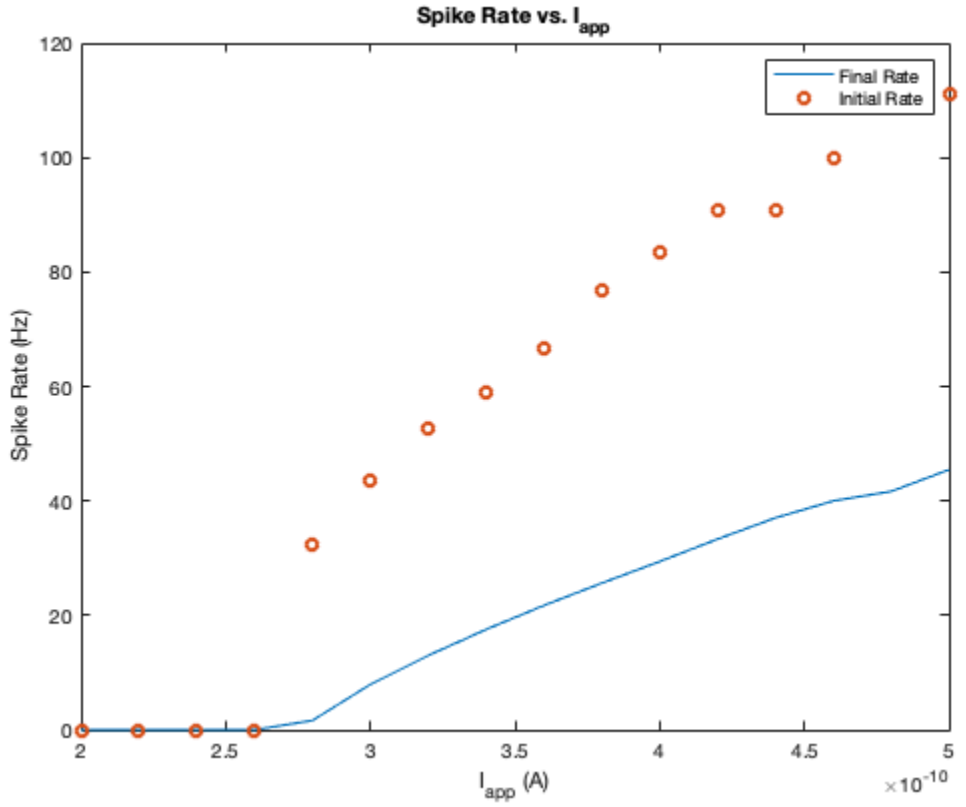
```
function [v_simulated, g_simulated] = q1sim(v, g, t, i_applied)
% Simulates the membrane potential with given inputs. If no noise
% fluctuations desired, pass 0 as the sigma argument.
global v_threshold v_reset dt delta_g_sra
v_simulated = zeros(1, length(v));
v_simulated(1) = v(1);
g_simulated = zeros(1, length(g));
g_simulated(1) = g(1);
temp = length(v)-1;
for n = 1:temp
    % Update membrane potential.
    K0 = dt*(F(v_simulated(n), g_simulated(n), t(n), i_applied(n)));
    K1 = dt*(F(v_simulated(n) + K0/2, g_simulated(n), t(n) + dt/2,
 i_applied(n)));
    K2 = dt*(F(v_simulated(n) + K1/2, g_simulated(n), t(n) + dt/2,
 i_applied(n)));
    K3 = dt*(F(v_simulated(n) + K2, g_simulated(n), t(n) + dt,
 i_applied(n)));
    K4 = (K0 + 2*K1 + 2*K2 + K3)/6;

    % Update conductance.
    k0 = dt*(G(g_simulated(n)));
    k1 = dt*(G(g_simulated(n) + k0/2));
    k2 = dt*(G(g_simulated(n) + k1/2));
    k3 = dt*(G(g_simulated(n) + k2));
    k4 = (k0 + 2*k1 + 2*k2 + k3)/6;

    if (v_simulated(n) + K4) >= v_threshold
        v_simulated(n+1) = v_reset;
        g_simulated(n+1) = g_simulated(n) + delta_g_sra;
    else
        v_simulated(n+1) = v_simulated(n) + K4;
        g_simulated(n+1) = g_simulated(n) + k4;
    end
end
end

function [v_simulated, i_sra_simulated] = q2sim(v, i_sra, t,
 i_applied)
% Simulates the membrane potential with given inputs. If no noise
% fluctuations desired, pass 0 as the sigma argument.
global v_threshold v_reset dt b v_max
v_simulated = zeros(1, length(v));
v_simulated(1) = v(1);
i_sra_simulated = zeros(1, length(i_sra));
i_sra_simulated(1) = i_sra(1);
temp = length(v)-1;
for n = 1:temp
    % Update membrane potential.
    K0 = dt*(F2(v_simulated(n), i_sra_simulated(n), t(n),
 i_applied(n)));
    K1 = dt*(F2(v_simulated(n) + K0/2, i_sra_simulated(n), t(n) +
 dt/2, i_applied(n)));
```

```
    K2 = dt*(F2(v_simulated(n) + K1/2, i_sra_simulated(n), t(n) +
 dt/2, i_applied(n)));
    K3 = dt*(F2(v_simulated(n) + K2, i_sra_simulated(n), t(n) + dt,
 i_applied(n)));
    K4 = (K0 + 2*K1 + 2*K2 + K3)/6;

    % Update conductance.
    k0 = dt*(G2(v_simulated(n), i_sra_simulated(n)));
    k1 = dt*(G2(v_simulated(n), i_sra_simulated(n) + k0/2));
    k2 = dt*(G2(v_simulated(n), i_sra_simulated(n) + k1/2));
    k3 = dt*(G2(v_simulated(n), i_sra_simulated(n) + k2));
    k4 = (k0 + 2*k1 + 2*k2 + k3)/6;

    if (v_simulated(n) + K4) >= v_max
        v_simulated(n+1) = v_reset;
        i_sra_simulated(n+1) = i_sra_simulated(n) + b;
    else
        v_simulated(n+1) = v_simulated(n) + K4;
        i_sra_simulated(n+1) = i_sra_simulated(n) + k4;
    end
end
end

function count = fire_count(v, v_exceed)
% Counts the number of spikes by counting the number of times the
% membrane potential exceeds a given value. v is a vector, v_exceed
% is a scalar.
count = 0;
for n = 1:length(v)
    if v(n) > v_exceed
        count = count + 1;
    end
end
end

function search_results = search_potential(v, search_val)
global dt
% Finds the time between the first two occurences of a membrane
% potential equal to v_equal.
search_results = zeros(1, length(v));
for n = 1:length(v)
   if v(n) == search_val
       search_results(n) = 1;
   end
end
end
```


×10^-10
Applied Current (A)
6
4
2
0
0
0.5
1
1.5
Membrane Potential (V)
-0.05
-0.06
-0.07
-0.08
×10^-9
Conductance (S)
10
5
Time (s)


Spike Rate vs. I_app
Final Rate
Initial Rate
Spike Rate (Hz)
120
100
80
60
40
20
2
2.5
3
3.5
4
4.5
5
I_app (A)
×10^-10


×10^-10
Applied Current (A)
6
4
2
0
0
0.5
1
1.5
Membrane Potential (V)
-0.04
-0.06
-0.08
-0.1
×10^-10
I_SRA (A)
2
1
0


Spike Rate vs. I_app
Final Rate
Initial Rate
Spike Rate (Hz)
120
100
80
60
40
20
0
2
2.5
3
3.5
4
4.5
5
I_app (A)
×10^-10

Published with MATLAB® R2018a