Table of Contents

a) Learning Pod

```
% POD Members: Meghan, Raymond

% We met on <May 03, Friday>, for about <2h>

% We discussed current-clamp protocol and wrote pseudocode for the function
% implementation.

% I got clarifications on variable declaration and logic on writing the
% function.

% I contributed to clarify by helping writing the pseudocode for the
% function below and implementing the function.

% b) current-clamp protocol
% Current: Begins with a baseline current of zero.
% Change: At 100 ms, there's a sudden increase to a current of 0.22 nA.
% Duration: The elevated current lasts for 100 ms.
% Return: After this period, the current returns to zero and remains there
%          for the rest of the simulation.

% c) current-clamp protocol
% Current: Starts at zero.
% Change: Features multiple pulses of 0.22 nA, each pulse lasting 5 ms.
% Pattern: These pulses are separated first by 16 ms and then by 18 ms
%       delays, repeating this pattern throughout the simulation period.
% Regular Spacing: The pulses are evenly spaced with consistent intervals
%       and durations.

% d) current-clamp protocol
% Current: Constant baseline current of 0.6 nA throughout the entire period.
% Change: There are interruptions in the current, where it drops to 0 nA.
% Duration of Interruptions: Each interruption lasts for 5 ms.
% Spacing: Interruptions are spaced by 20 ms intervals.

% e) current-clamp protocol
% Current: A baseline of 0.65 nA is maintained initially.
% Spike: At 100 ms, there's a spike where the current increases to 1 nA.
% Duration of Spike: The spike lasts for 5 ms.
```

```
% Post-Spike: After the spike, the current returns to the baseline of
%               0.65 nA for the remainder of the period.

% f) current-clamp protocol
% Current: Similar to part e, but the constant baseline is 0.7 nA.
% Spike: There is a single spike to 1 nA at 100 ms.
% Duration of Spike: This spike also lasts for 5 ms.
% Post-Spike: The current then returns to the baseline of 0.7 nA.


% Pseudocode for the function
% Function hhsim(t, i_applied, v_init, m_init, h_init, n_init)
%     // Variable Definitions
%     Declare global variables dt, g_leak, g_na, g_k, e_na, e_k, e_leak,
c_membrane
%
%     // Check for initial conditions
%     // Flow Chart: Decision blocks for initial values
%     If v_init is not provided
%         Set v_init to -61e-3  // Default value for v_init
%     If m_init is not provided
%         Set m_init to 0      // Default value for m_init
%     If h_init is not provided
%         Set h_init to 0      // Default value for h_init
%     If n_init is not provided
%         Set n_init to 0      // Default value for n_init
%
%     // Initialize simulation vectors for v, m, h, and n
%     Initialize v_sim, m_sim, h_sim, n_sim arrays with zeroes, length same
as t
%
%     // Set initial values to the first element of each vector
%     v_sim[1] = v_init
%     m_sim[1] = m_init
%     h_sim[1] = h_init
%     n_sim[1] = n_init
%
%     // Simulation loop
%     // Flow Chart: Loop for stepping through time
%     For each time step n from 1 to length(t)-1
%         // Compute the voltage update for v_sim
%         term1 = g_leak * (e_leak - v_sim[n])
%         term2 = g_na * m_sim[n]^3 * h_sim[n] * (e_na - v_sim[n])
%         term3 = g_k * n_sim[n]^4 * (e_k - v_sim[n])
%         v_sim[n+1] = v_sim[n] + dt * ((term1 + term2 + term3 +
i_applied[n]) / c_membrane)
%
%         // Update gating variables m, h, and n
%         // Flow Chart: Update m_sim using alpha and beta equations
%         alpha = (10^5 * (-v_sim[n] - 0.045)) / (exp(100 * (-v_sim[n] -
0.045)) - 1)
%         beta = 4 * 10^3 * exp((-v_sim[n] - 0.07) / 0.018)
%         term1 = alpha * (1 - m_sim[n])
%         term2 = -beta * m_sim[n]
```

```
%          m_sim[n+1] = m_sim[n] + dt * (term1 + term2)
%
%          // Flow Chart: Update h_sim using alpha and beta equations
%          alpha = 70 * exp(50 * (-v_sim[n] - 0.07))
%          beta = (10^3) / (1 + exp(100 * (-v_sim[n] - 0.04)))
%          term1 = alpha * (1 - h_sim[n])
%          term2 = -beta * h_sim[n]
%          h_sim[n+1] = h_sim[n] + dt * (term1 + term2)
%
%          // Flow Chart: Update n_sim using alpha and beta equations
%          alpha = (10^4 * (-v_sim[n] - 0.06)) / (exp(100 * (-v_sim[n] -
0.06)) - 1)
%          beta = 125 * exp((-v_sim[n] - 0.07) / 0.08)
%          term1 = alpha * (1 - n_sim[n])
%          term2 = -beta * n_sim[n]
%          n_sim[n+1] = n_sim[n] + dt * (term1 + term2)
%
%      // Return computed simulation results
%      Return v_sim, m_sim, h_sim, n_sim
% End Function


% Issue encountered: One difficulty is the precise implementation of
% timing and duration for applied currents. This involves converting
% real-time durations into discrete time steps accurately. Errors can
% easily arise during this conversion, particularly if the simulation's
% time step (dt) does not neatly divide the intended pulse durations or
% delays.
```

Tutorial 4.1

```
% Define parameters.
global g_leak g_na g_k e_na e_k e_leak c_membrane
g_leak = 30e-9;
g_na = 12e-6;
g_k = 3.6e-6;
e_na = 45e-3;
e_k = -82e-3;
e_leak = -60e-3;
c_membrane = 100e-12;

% Setup time vector.

global dt
dt = 0.0001e-3;
t = 0:dt:0.35;

% Create vector for applied current in part b.
I_app = zeros(1, length(t));
for i = 1:((100/dt)/1000)
   I_app(floor(((100/dt)/1000)+i)) = 0.22e-9;
end
```

```
% Run simulation.
[v_sim, m_sim, h_sim, n_sim] = hhsim(t, I_app);
```

Generate plots for part b.

```
f1 = figure
figure(f1)
subplot(2,1,1)
plot(t, I_app)
xlabel("Time (seconds)")
ylabel("Applied Current")
title("Part b: Applied Current vs. Time")
ylim([-0.2e-10, 2.5e-10])
subplot(2,1,2)
plot(t, v_sim)
xlabel("Time (seconds)")
ylabel("Membrane Potential")
title("Part b: Membrane Potential vs. Time")
ylim([-0.1, 0.05])
```

```
f1 =

  Figure (1) with properties:

      Number: 1
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties
```

Use ChatGPT to create the following

```
% Write a MATLAB function: Set up a total duration of 0.35 s. Produce a
% vector for a variable, Iapp that has a baseline of zero and steps up to
% 0.22 for a duration of 100 ms beginning at a time of 100 ms.

function Iapp = createCurrentProfile()
    % Define simulation parameters
    totalTime = 0.35;   % Total duration in seconds
    dt = 0.001;         % Time step in seconds (1 ms)
    amplitude = 0.22;   % Current amplitude

    % Create time vector from 0 to totalTime with a step of dt
    t = 0:dt:totalTime;

    % Initialize Iapp to zero for all time points
    Iapp = zeros(size(t));

    % Find indices for start and end of current step
```

```
    startIdx = find(t >= 0.1, 1, 'first');
    endIdx = find(t >= 0.2, 1, 'first');

    % Set Iapp to amplitude between startIdx and endIdx
    Iapp(startIdx:endIdx) = amplitude;

    % Optionally plot Iapp to visualize the current profile
    figure;
    plot(t, Iapp);
    xlabel('Time (s)');
    ylabel('Iapp (A)');
    title('Current Profile');
end

% testing functions:

function testInitialConditions()
    Iapp = createCurrentProfile();
    t = 0:0.001:0.35;
    initialCondition = all(Iapp(t < 0.1) == 0);
    disp(['Test Initial Conditions: ', num2str(initialCondition)]);
end

function testCurrentStep()
    Iapp = createCurrentProfile();
    t = 0:0.001:0.35;
    currentStepUp = all(Iapp(t >= 0.1 & t < 0.2) == 0.22);
    currentStepDown = all(Iapp(t >= 0.2) == 0);
    disp(['Test Current Step Up: ', num2str(currentStepUp)]);
    disp(['Test Current Step Down: ', num2str(currentStepDown)]);
end

function testDurationOfCurrentStep()
    Iapp = createCurrentProfile();
    t = 0:0.001:0.35;
    startIndex = find(Iapp == 0.22, 1, 'first');
    endIndex = find(Iapp == 0.22, 1, 'last');
    duration = t(endIndex) - t(startIndex) + 0.001;  % Adding dt because
inclusive
    correctDuration = abs(duration - 0.1) < 1e-5;  % Small tolerance for
floating-point comparison
    disp(['Test Duration of Current Step: ', num2str(correctDuration)]);
end

function testEndConditions()
    Iapp = createCurrentProfile();
    t = 0:0.001:0.35;
    endCondition = all(Iapp(t >= 0.2) == 0);
    disp(['Test End Conditions: ', num2str(endCondition)]);
end

% By running the testing functions in the command window, I find that the
% generated function is correct. Running the test initial conditions, I got
% the output true that the displays that initial conditions are met,
```

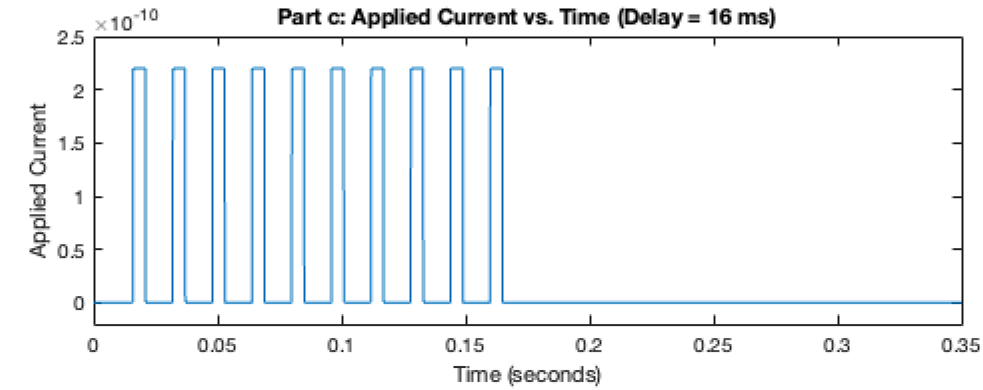
```
% indicating that "Iapp" is zero before 100 ms. By running test current
% step, I get the output true for both the step up and step down, showing
% "Iapp" is 0.22 between 100 ms and 200 ms and 0 otherwise. By running test
% duration of current step, I get output true, confirming that the duration
% of the step at 0.22 lasts exactly 100ms. By running test end conditions,
% I got output true indicating that "Iapp" returns to zero after 200 ms and
% stays zero until the end of the simulation.
```

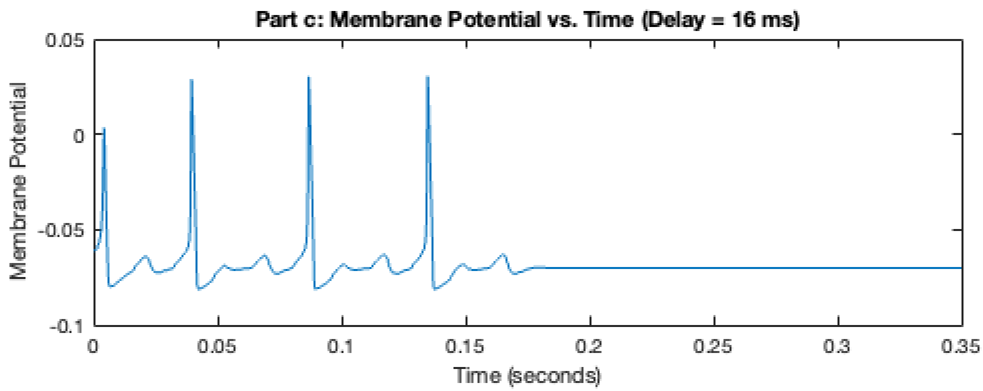
Part (c), run twice with two different ms delays.

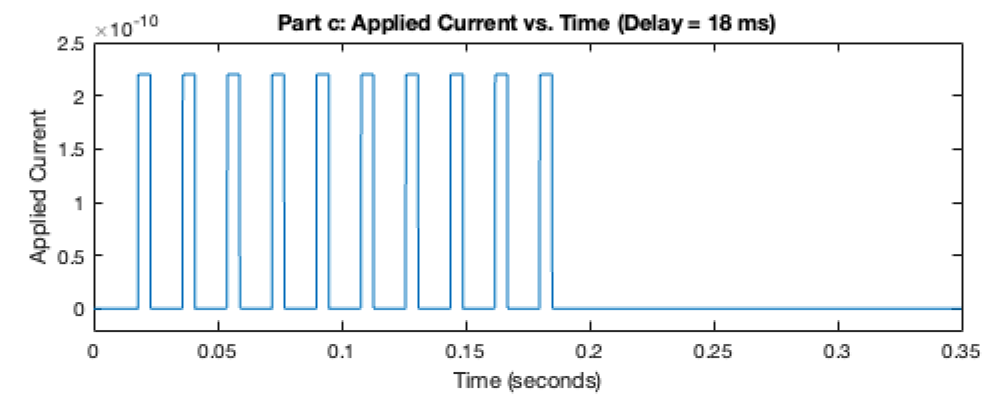
```
% Create vector for applied current.
I_app = zeros(1, length(t));
delay = 16; % number in milliseconds.
delay_index = floor((delay/dt)/1000);
for i = 1:10
   left = i*delay_index;
   right = left + floor((5/dt)/1000);
   I_app(left:right) = 0.22e-9;
end

% Run simulation/
[v_sim, m_sim, h_sim, n_sim] = hhsim(t, I_app);

% Generate plots for part c.
f2 = figure
figure(f2)
subplot(2,1,1)
plot(t, I_app)
xlabel("Time (seconds)")
ylabel("Applied Current")
title("Part c: Applied Current vs. Time (Delay = 16 ms)")
ylim([-0.2e-10, 2.5e-10])
subplot(2,1,2)
plot(t, v_sim)
xlabel("Time (seconds)")
ylabel("Membrane Potential")
title("Part c: Membrane Potential vs. Time (Delay = 16 ms)")

% Create vector for applied current.
I_app = zeros(1, length(t));
delay = 18; % number in milliseconds.
delay_index = floor((delay/dt)/1000);
for i = 1:10
   left = i*delay_index;
   right = left + floor((5/dt)/1000);
   I_app(left:right) = 0.22e-9;
end

% Run simulation/
[v_sim, m_sim, h_sim, n_sim] = hhsim(t, I_app);

% Generate plots for part c.
f3 = figure
```

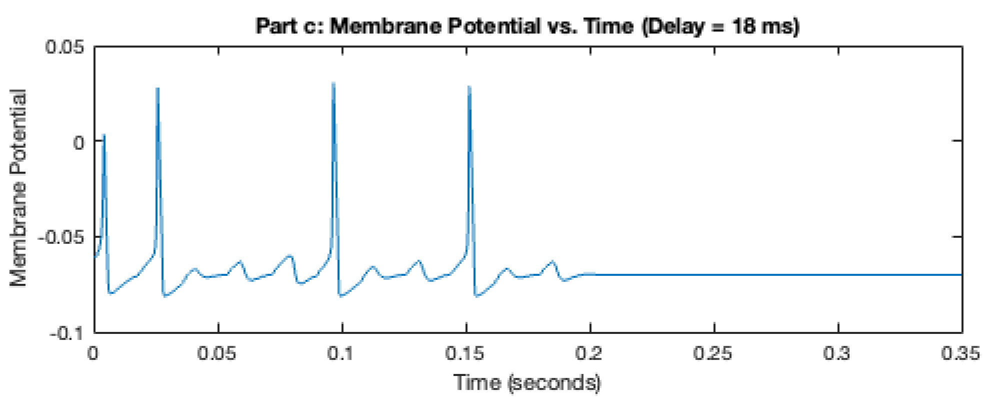
```
figure(f3)
subplot(2,1,1)
plot(t, I_app)
xlabel("Time (seconds)")
ylabel("Applied Current")
title("Part c: Applied Current vs. Time (Delay = 18 ms)")
ylim([-0.2e-10, 2.5e-10])
subplot(2,1,2)
plot(t, v_sim)
xlabel("Time (seconds)")
ylabel("Membrane Potential")
title("Part c: Membrane Potential vs. Time (Delay = 18 ms)")
```

```
f2 = 

  Figure (2) with properties:

      Number: 2
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties


f3 = 

  Figure (3) with properties:

      Number: 3
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties
```


Part c: Applied Current vs. Time (Delay = 16 ms)
×10^-10
Applied Current
Time (seconds)
0
0.5
1
1.5
2
2.5
0.05
0.1
0.15
0.2
0.25
0.3
0.35


Part c: Membrane Potential vs. Time (Delay = 16 ms)
Membrane Potential
Time (seconds)
0.05
0
-0.05
-0.1
0.1
0.15
0.2
0.25
0.3
0.35

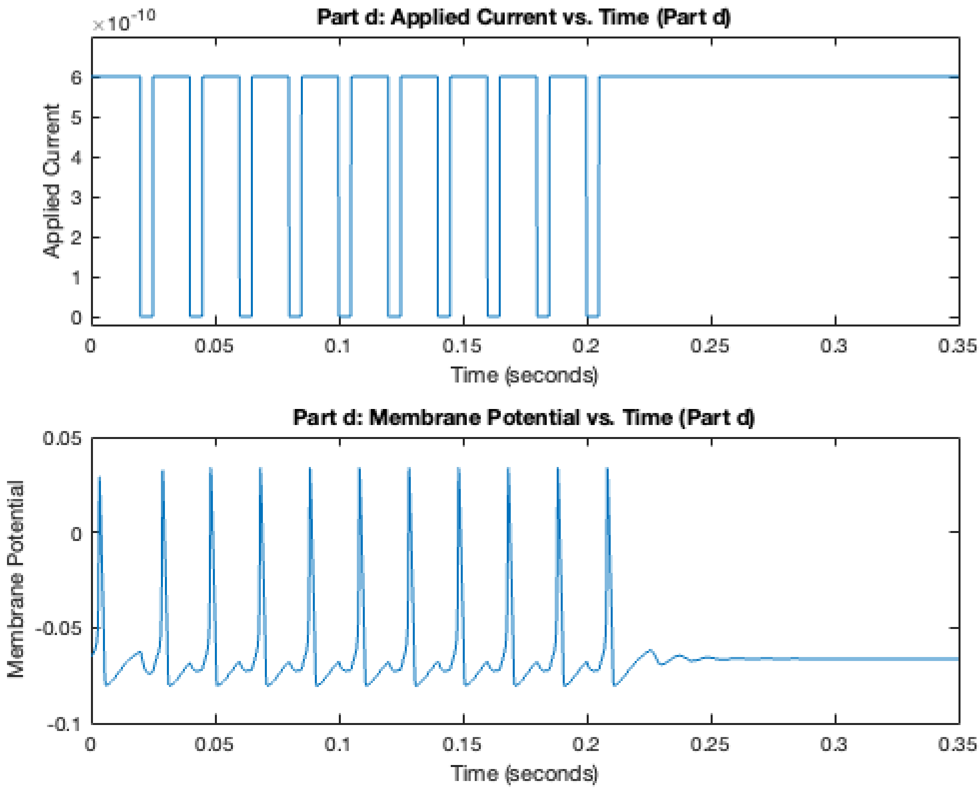

Part c: Applied Current vs. Time (Delay = 18 ms)
×10^-10
Applied Current
Time (seconds)
0
0.5
1
1.5
2
2.5
0.05
0.1
0.15
0.2
0.25
0.3
0.35


Part c: Membrane Potential vs. Time (Delay = 18 ms)
Membrane Potential
Time (seconds)
0.05
0
-0.05
-0.1
0.1
0.15
0.2
0.25
0.3
0.35

Part d.

```
% Create vector for applied current.
I_app = zeros(1, length(t)) + 0.6e-9;
delay = 20; % number in milliseconds.
delay_index = floor((delay/dt)/1000);
for i = 1:10
   left = i*delay_index;
   right = left + floor((5/dt)/1000);
   I_app(left:right) = 0.0;
end

% Run simulation/
[v_sim, m_sim, h_sim, n_sim] = hhsim(t, I_app, -0.065, 0.05, 0.5, 0.35);

% Generate plots for part d.
f4 = figure
figure(f4)
subplot(2,1,1)
plot(t, I_app)
xlabel("Time (seconds)")
ylabel("Applied Current")
title("Part d: Applied Current vs. Time (Part d)")
ylim([-0.2e-10, 7e-10])
subplot(2,1,2)
plot(t, v_sim)
xlabel("Time (seconds)")
ylabel("Membrane Potential")
title("Part d: Membrane Potential vs. Time (Part d)")
```

```
f4 = 

  Figure (4) with properties:

      Number: 4
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties
```

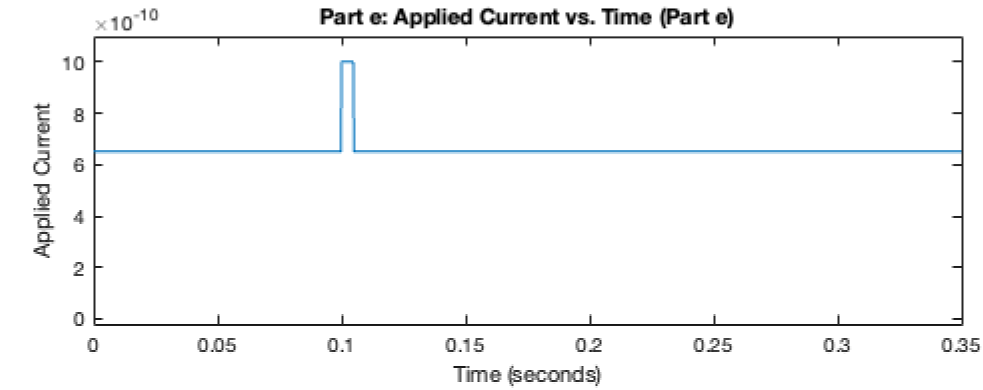


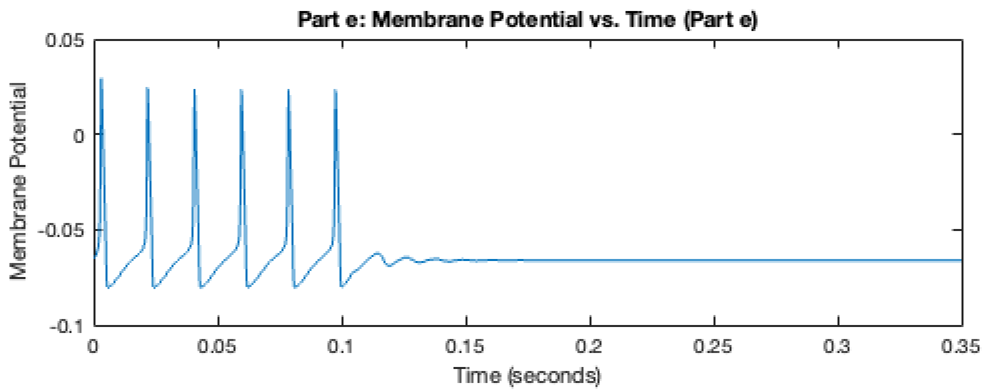
Part e.

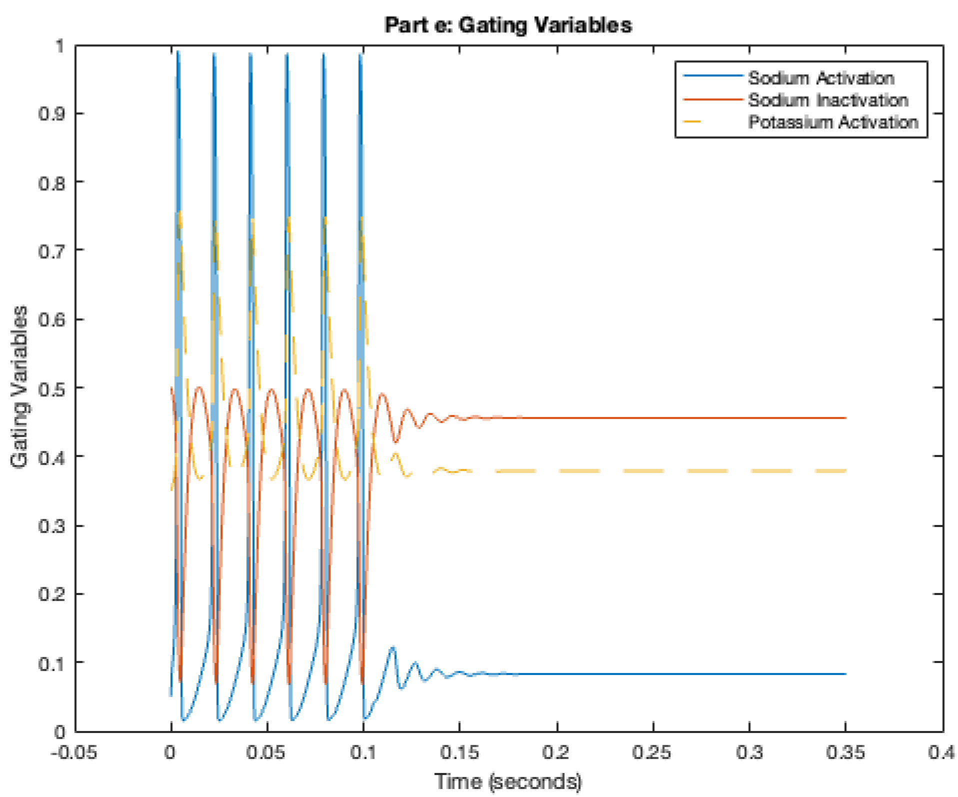
```
% Create vector for applied current.
I_app = zeros(1, length(t)) + 0.65e-9;
delay = 20; % number in milliseconds.
delay_index = floor((delay/dt)/1000);
for i = 1:1
   left =  floor((100/dt)/1000);
   right = left + floor((5/dt)/1000);
   I_app(left:right) = 1e-9;
end

% Run simulation/
[v_sim, m_sim, h_sim, n_sim] = hhsim(t, I_app, -0.065, 0.05, 0.5, 0.35);

% Generate plots for part e.
f5 = figure
figure(f5)
subplot(2,1,1)
plot(t, I_app)
xlabel("Time (seconds)")
ylabel("Applied Current")
title("Part e: Applied Current vs. Time (Part e)")
ylim([-0.2e-10, 11e-10])
subplot(2,1,2)
```

```
plot(t, v_sim)
xlabel("Time (seconds)")
ylabel("Membrane Potential")
title("Part e: Membrane Potential vs. Time (Part e)")

f5half = figure
figure(f5half)
plot(t, m_sim)
hold on
plot(t, h_sim, '-')
hold on
plot(t, n_sim, '--')
xlabel("Time (seconds)")
ylabel("Gating Variables")
title("Part e: Gating Variables")
legend("Sodium Activation", "Sodium Inactivation", "Potassium Activation")
xlim([-0.05, 0.4])
saveas(f5half,"Part_e_activation_variables.png")
```

```
f5 =

  Figure (5) with properties:

      Number: 5
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties


f5half =

  Figure (6) with properties:

      Number: 6
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties
```


Part e: Applied Current vs. Time (Part e)
×10^-10
Applied Current
Time (seconds)


Part e: Membrane Potential vs. Time (Part e)
Membrane Potential
Time (seconds)


Part e: Gating Variables
Sodium Activation
Sodium Inactivation
Potassium Activation
Gating Variables
Time (seconds)

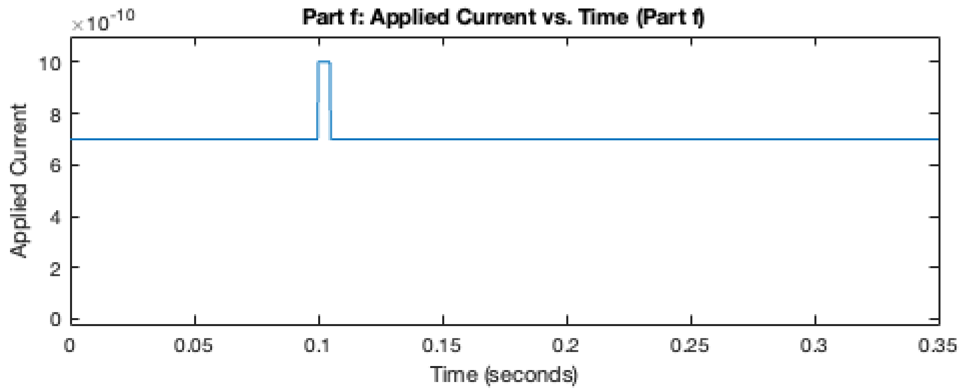
Part f.

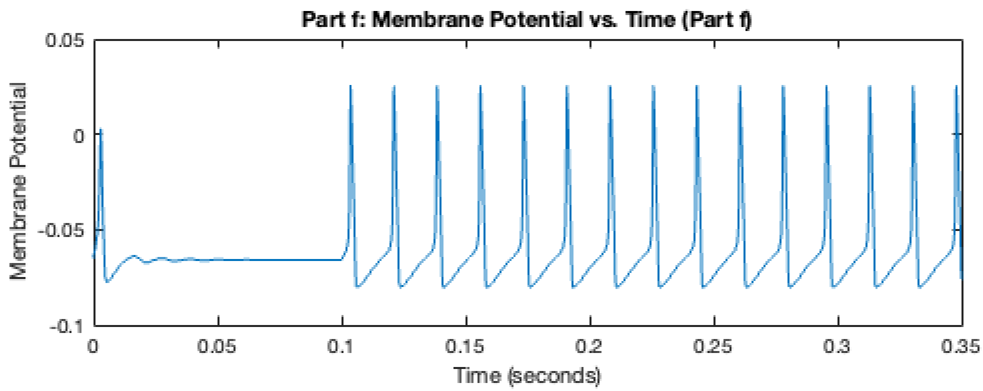
```
% Create vector for applied current.
I_app = zeros(1, length(t)) + 0.7e-9;
delay = 20; % number in milliseconds.
delay_index = floor((delay/dt)/1000);
for i = 1:1
   left =  floor((100/dt)/1000);
   right = left + floor((5/dt)/1000);
   I_app(left:right) = 1e-9;
end

% Run simulation/
[v_sim, m_sim, h_sim, n_sim] = hhsim(t, I_app, -0.065, 0.00, 0.00, 0.00);

% Generate plots for part f.
f6 = figure
figure(f6)
subplot(2,1,1)
plot(t, I_app)
xlabel("Time (seconds)")
ylabel("Applied Current")
title("Part f: Applied Current vs. Time (Part f)")
ylim([-0.2e-10, 11e-10])
subplot(2,1,2)
plot(t, v_sim);
xlabel("Time (seconds)")
ylabel("Membrane Potential")
title("Part f: Membrane Potential vs. Time (Part f)")

f7 = figure
figure(f7)
plot(t, m_sim)
hold on
plot(t, h_sim, '-')
hold on
plot(t, n_sim, '--')
xlabel("Time (seconds)")
ylabel("Gating Variables")
title("Part f: Gating Variables")
legend("Sodium Activation", "Sodium Inactivation", "Potassium Activation")
xlim([-0.05, 0.4])
saveas(f7,"Part_f_activation_variables.png")


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

% Simulates the Hodgkin-Huxley model given the input time vector and
% applied current.
function [v_sim, m_sim, h_sim, n_sim] = hhsim(t, i_applied, v_init, m_init,
h_init, n_init)
    global dt g_leak g_na g_k e_na e_k e_leak c_membrane
```

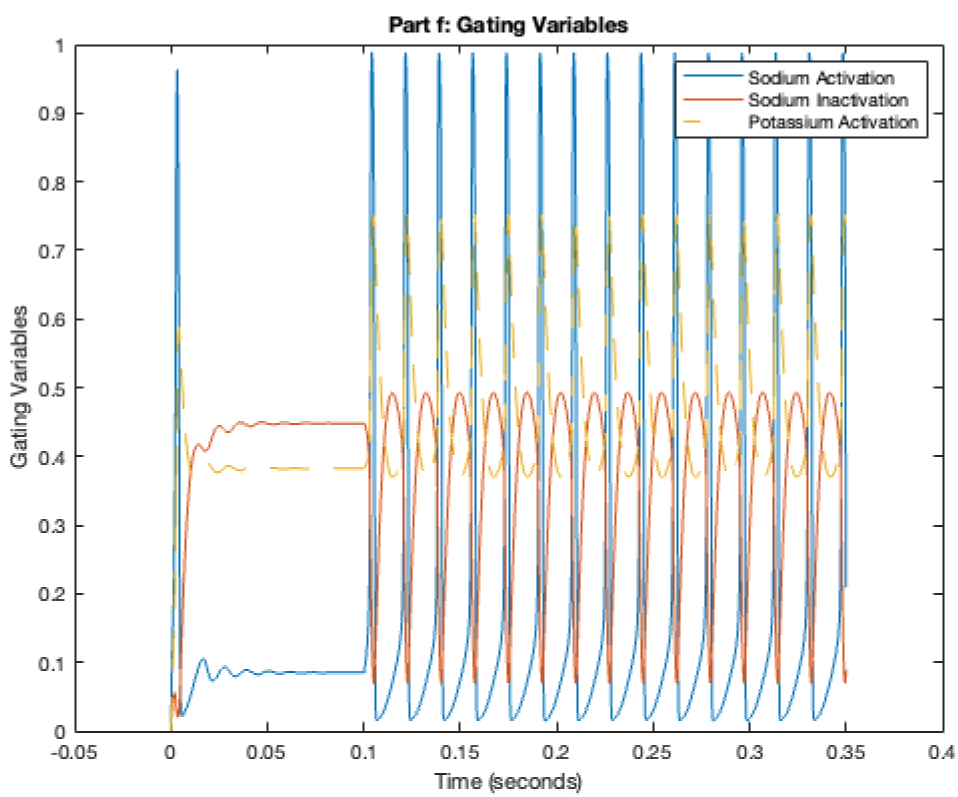
```
    % Default parameters if not inputted.
    if (~exist('v_init'))
        v_init = -61e-3;
    end
    if (~exist('m_init'))
        m_init = 0;
    end
    if (~exist('h_init'))
        h_init = 0;
    end
    if (~exist('n_init'))
        n_init = 0;
    end

    % Setup vectors.
    v_sim = zeros(1, length(t));
    m_sim = zeros(1, length(t));
    h_sim = zeros(1, length(t));
    n_sim = zeros(1, length(t));

    v_sim(1) = v_init;
    m_sim(1) = m_init;
    h_sim(1) = h_init;
    n_sim(1) = n_init;

    % March forward in time.
    for n = 1:(length(t)-1)
        % Update v_sim.
        term1 = g_leak*(e_leak-v_sim(n));
        term2 = g_na*(m_sim(n)^3)*(h_sim(n))*(e_na - v_sim(n));
        term3 = g_k*(n_sim(n)^4)*(e_k - v_sim(n));
        v_sim(n+1) = v_sim(n) + (dt*((term1 + term2 + term3 + i_applied(n))/
c_membrane));

        % Update m_sim.
        alpha = (((10^5)*(-v_sim(n)-0.045))/(exp(100*(-v_sim(n)-0.045))-1));
        beta = (4*(10^3))*exp((-v_sim(n) - 0.07)/(0.018));
        term1 = alpha*(1-m_sim(n));
        term2 = -beta*m_sim(n);
        m_sim(n+1) = m_sim(n) + (dt*(term1 + term2));

        % Update h_sim.
        alpha = 70*exp(50*(-v_sim(n)-0.07));
        beta = ((10^3) / (1 + exp(100*(-v_sim(n)-0.04))));
        term1 = alpha*(1-h_sim(n));
        term2 = -beta*h_sim(n);
        h_sim(n+1) = h_sim(n) + (dt*(term1 + term2));

        % Update n_sim.
        alpha = (((10^4)*(-v_sim(n)-0.06))/(exp(100*(-v_sim(n)-0.06))-1));
        beta = 125*exp(((-v_sim(n)-0.07)/(0.08)));
        term1 = alpha*(1-n_sim(n));
        term2 = -beta*n_sim(n);
```

```
        n_sim(n+1) = n_sim(n) + (dt*(term1 + term2));
    end
end
```

```
f6 =

  Figure (7) with properties:

      Number: 7
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties


f7 =

  Figure (8) with properties:

      Number: 8
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [476 360 560 420]
       Units: 'pixels'

  Use GET to show all properties
```


Part f: Applied Current vs. Time (Part f)
×10^-10
Applied Current
Time (seconds)


Part f: Membrane Potential vs. Time (Part f)
Membrane Potential
Time (seconds)


Part f: Gating Variables
Sodium Activation
Sodium Inactivation
Potassium Activation
Gating Variables
Time (seconds)

Published with MATLAB® R2024a