Neuron simulation through Hodgkin-Huxley and Leaky Integrate-and-Fire models

Zarif Ahmed

April 2025

Abstract

In this report, we simulate and analyze two neuron models: the Hodgkin-Huxley (HH) model and the Leaky Integrate-and-Fire (LIF) model. The HH model gives a detailed representation of action potential generation based on voltage-gated sodium and potassium channels, while LIF is a more simplified model that uses boundary conditions to generate action potential. We implemented both models using the Forward Euler method in MATLAB and explored their responses to varying input currents. For both models, we found the spike threshold and rheobase currents and observed a Type II firing and Type I firing for HH and LIF model respectively. Finally, we compared the computational runtimes of both models to find which is easier to scale. Our findings show that while the HH model gives more detailed information on the roles of ion channels in generating action potentials, the LIF model is able to generate reasonable simulations while being less computationally expensive.

Introduction

To computationally model a brain or parts of the brain we first need to understand how neurons transmit information. There are many models that have been made to help to simulate neurons. The two we will be looking at in this report are the Hodgkin-Huxley model and the Leaky Integrate-and-Fire model. The HH model describes the generation of action potentials through

sodium and potassium ion channels. In contrast, LIF is simpler model which generates action potential through the use of boundary conditions. In this report, we simulate both models with varying input currents and compare the results to understand the difference between both models. The equations and the parameter values of both the Hodgkin-Huxley model and the Leaky Integrate-and-Fire model are provided below.

The Hodgkin-Huxley model:

$$C\frac{dV}{dt} = -G_L(V - V_L) - G_{\mathrm{Na}}m^3h(V - E_{\mathrm{Na}}) - G_Kn^4(V - E_K) + I_e$$

- V = Membrane potential.
- C = Membrane capacitance.
- G_L = Leak channel conductance.
- G_{Na} = Sodium channel conductance.
- G_K = Potassium conductance.
- V_L = Reversal potentials for leak channel.
- E_{Na} = Reversal potentials for sodium ion.
- E_K = Reversal potentials potassium ion.
- m = Sodium channel activation gate.
- h = Sodium channel inactivation gate.
- n = Potassium channel activation gate.
- I_e = External current.

Each gating variable obey the first-order ODEs:

$$\frac{dx}{dt} = \alpha_x(V)(1 - x) - \beta_x(V)x$$

where x is the given gating variable. Transition rates $\alpha(V)$ and $\beta(V)$ are given by:

$$\alpha_m(V) = \frac{0.1(V+40)}{1-e^{-0.1(V+40)}}, \quad \beta_m(V) = 4e^{-0.0556(V+65)}$$

$$\alpha_h(V) = 0.07e^{-0.05(V+65)}, \quad \beta_h(V) = \frac{1}{1+e^{-0.1(V+35)}}$$

$$\alpha_n(V) = \frac{0.01(V+55)}{1-e^{-0.1(V+55)}}, \quad \beta_n(V) = 0.125e^{-0.0125(V+65)}$$

Unless otherwise specified, model parameters are given as follows for all experiments:

- $G_{Na} = 400$ nS.
- $G_K = 200$ nS.
- $G_L = 2$ nS.
- $E_{Na} = 99$ mV.
- $E_K =$ -85 mV.
- $V_L =$ -65 mV.
- $C = 2$ pF.
- $I_e =$ Specified for each result.

Initial Conditions:

- $V(0) = V_L$.
- $m(0) = m_\infty(V(0))$.
- $h(0) = h_\infty(V(0))$.
- $n(0) = n_\infty(V(0))$.

where,

$$m_\infty(V) = \frac{\alpha_m(V)}{\alpha_m(V) + \beta_m(V)}$$

and analogous for $h_\infty(V)$ and $n_\infty(V)$

The Leaky Integrate-and-Fire (LIF) model simplifies the neuron's membrane dynamics and is defined by:

$$C\frac{dV}{dt} = -G_L(V - V_L) + I_e$$

For our model, a spike is said to occur when $V(t) \geq V_{\text{spk}}$ where V_{spk} = -45 mV, after which the potential is reset to V_r = -65 mV and held there for a refractory period τ_{arp} = 2 ms. Afterwards the dynamics of the Eq resumes.

Unless otherwise specified, model parameters are given as follows for all experiments:

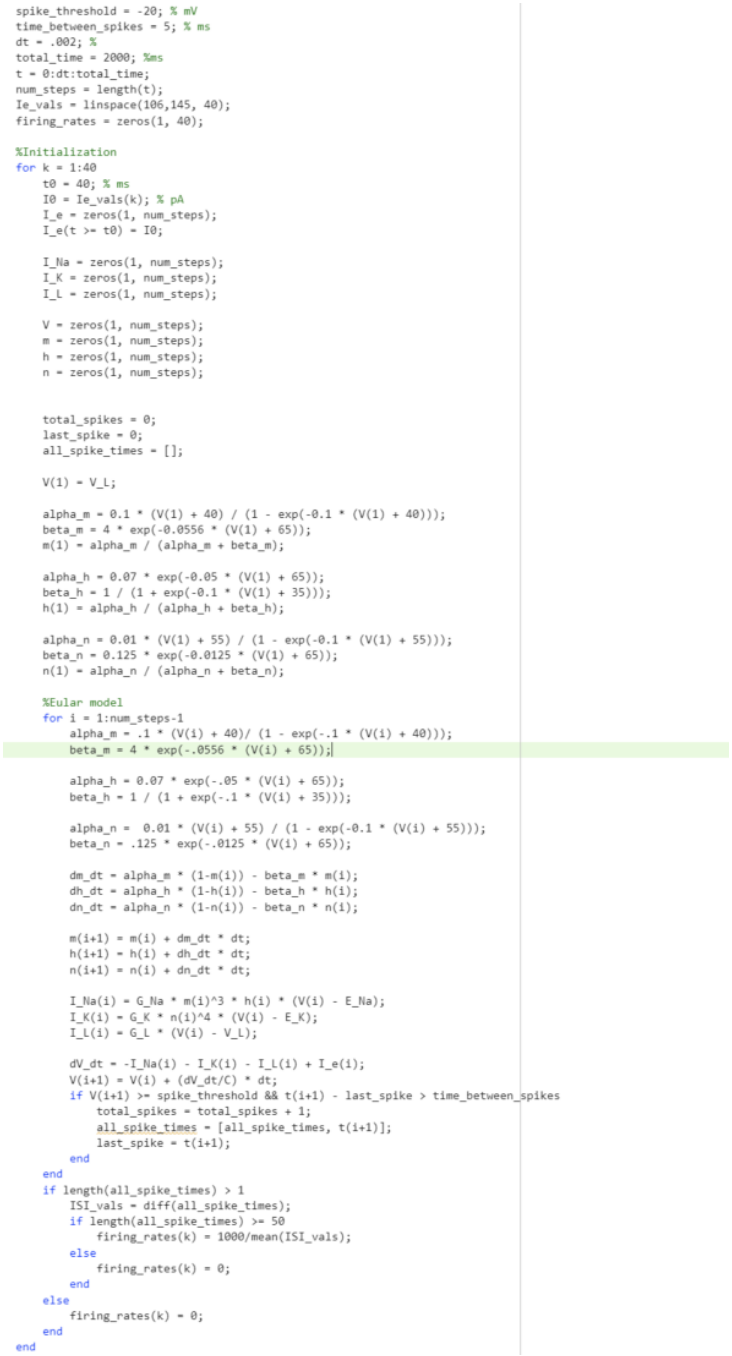
- G_L = 50 nS.
- V_L = -65 mV.
- C = 1 nF.
- I_e = Specified for each result.

With initial condition V(0) = V_L.

We ues MATLAB to implement these models with the help of the Forward Euler method. The MATLAB codes are shown in Figures 1, 2, 3 and 4.

```
% Model Parameters
G_Na = 400; %nS
G_K = 200; %nS
G_L = 2; %nS
E_Na = 99; %mV
E_K = -85; %mV
V_L = -65; %mV
C = 2; %pF

dt = .01; %
total_time = 200; %ms
t = 0:dt:total_time;
num_steps = length(t);
Ie_vals = 101:1:140;
firing_rates = zeros(1, 40);

t0 = 40; % ms
I0 = 200; % pA
I_e = zeros(1, num_steps);
I_e(t >= t0) = I0;
I_Na = zeros(1, num_steps);
I_K = zeros(1, num_steps);
I_L = zeros(1, num_steps);

V = zeros(1, num_steps);
m = zeros(1, num_steps);
h = zeros(1, num_steps);
n = zeros(1, num_steps);

spike_threshold = -20; % mV
time_between_spikes = 5; % ms
total_spikes = 0;
last_spike = 0;
all_spike_times = [];
all_spike_voltages = [];

%Initialization
V(1) = V_L;

alpha_m = 0.1 * (V(1) + 40) / (1 - exp(-0.1 * (V(1) + 40)));
beta_m = 4 * exp(-0.0556 * (V(1) + 65));
m(1) = alpha_m / (alpha_m + beta_m);

alpha_h = 0.07 * exp(-0.05 * (V(1) + 65));
beta_h = 1 / (1 + exp(-0.1 * (V(1) + 35)));
h(1) = alpha_h / (alpha_h + beta_h);

alpha_n = 0.01 * (V(1) + 55) / (1 - exp(-0.1 * (V(1) + 55)));
beta_n = 0.125 * exp(-0.0125 * (V(1) + 65));
n(1) = alpha_n / (alpha_n + beta_n);

tic
%Eular model
for i = 1:num_steps-1
    alpha_m = .1 * (V(i) + 40)/ (1 - exp(-.1 * (V(i) + 40)));
    beta_m = 4 * exp(-.0556 * (V(i) + 65));

    alpha_h = 0.07 * exp(-.05 * (V(i) + 65));
    beta_h = 1 / (1 + exp(-.1 * (V(i) + 35)));

    alpha_n =  0.01 * (V(i) + 55) / (1 - exp(-0.1 * (V(i) + 55)));
    beta_n = .125 * exp(-.0125 * (V(i) + 65));

    dm_dt = alpha_m * (1-m(i)) - beta_m * m(i);
    dh_dt = alpha_h * (1-h(i)) - beta_h * h(i);
    dn_dt = alpha_n * (1-n(i)) - beta_n * n(i);

    m(i+1) = m(i) + dm_dt * dt;
    h(i+1) = h(i) + dh_dt * dt;
    n(i+1) = n(i) + dn_dt * dt;

    I_Na(i) = G_Na * m(i)^3 * h(i) * (V(i) - E_Na);
    I_K(i) = G_K * n(i)^4 * (V(i) - E_K);
    I_L(i) = G_L * (V(i) - V_L);

    dV_dt = -I_Na(i) - I_K(i) - I_L(i) + I_e(i);
    V(i+1) = V(i) + (dV_dt/C) * dt;
    if V(i+1) >= spike_threshold && t(i+1) - last_spike > time_between_spikes
        total_spikes = total_spikes + 1;
        all_spike_times = [all_spike_times, t(i+1)];
        all_spike_voltages = [all_spike_voltages, V(i+1)];
        last_spike = t(i+1);
    end
end
toc
I_m = I_Na + I_K + I_L;
```


Figure 1: MATLAB implementation of Hodgkin-Huxley model



```
spike_threshold = -20; % mV
time_between_spikes = 5; % ms
dt = .002; %
total_time = 2000; %ms
t = 0:dt:total_time;
num_steps = length(t);
Ie_vals = linspace(106,145, 40);
firing_rates = zeros(1, 40);

%Initialization
for k = 1:40
    t0 = 40; % ms
    I0 = Ie_vals(k); % pA
    I_e = zeros(1, num_steps);
    I_e(t >= t0) = I0;

    I_Na = zeros(1, num_steps);
    I_K = zeros(1, num_steps);
    I_L = zeros(1, num_steps);

    V = zeros(1, num_steps);
    m = zeros(1, num_steps);
    h = zeros(1, num_steps);
    n = zeros(1, num_steps);


    total_spikes = 0;
    last_spike = 0;
    all_spike_times = [];

    V(1) = V_L;

    alpha_m = 0.1 * (V(1) + 40) / (1 - exp(-0.1 * (V(1) + 40)));
    beta_m = 4 * exp(-0.0556 * (V(1) + 65));
    m(1) = alpha_m / (alpha_m + beta_m);

    alpha_h = 0.07 * exp(-0.05 * (V(1) + 65));
    beta_h = 1 / (1 + exp(-0.1 * (V(1) + 35)));
    h(1) = alpha_h / (alpha_h + beta_h);

    alpha_n = 0.01 * (V(1) + 55) / (1 - exp(-0.1 * (V(1) + 55)));
    beta_n = 0.125 * exp(-0.0125 * (V(1) + 65));
    n(1) = alpha_n / (alpha_n + beta_n);

    %Eular model
    for i = 1:num_steps-1
        alpha_m = .1 * (V(i) + 40)/ (1 - exp(-.1 * (V(i) + 40)));
        beta_m = 4 * exp(-.0556 * (V(i) + 65));

        alpha_h = 0.07 * exp(-.05 * (V(i) + 65));
        beta_h = 1 / (1 + exp(-.1 * (V(i) + 35)));

        alpha_n =  0.01 * (V(i) + 55) / (1 - exp(-0.1 * (V(i) + 55)));
        beta_n = .125 * exp(-.0125 * (V(i) + 65));

        dm_dt = alpha_m * (1-m(i)) - beta_m * m(i);
        dh_dt = alpha_h * (1-h(i)) - beta_h * h(i);
        dn_dt = alpha_n * (1-n(i)) - beta_n * n(i);

        m(i+1) = m(i) + dm_dt * dt;
        h(i+1) = h(i) + dh_dt * dt;
        n(i+1) = n(i) + dn_dt * dt;

        I_Na(i) = G_Na * m(i)^3 * h(i) * (V(i) - E_Na);
        I_K(i) = G_K * n(i)^4 * (V(i) - E_K);
        I_L(i) = G_L * (V(i) - V_L);

        dV_dt = -I_Na(i) - I_K(i) - I_L(i) + I_e(i);
        V(i+1) = V(i) + (dV_dt/C) * dt;
        if V(i+1) >= spike_threshold && t(i+1) - last_spike > time_between_spikes
            total_spikes = total_spikes + 1;
            all_spike_times = [all_spike_times, t(i+1)];
            last_spike = t(i+1);
        end
    end
    if length(all_spike_times) > 1
        ISI_vals = diff(all_spike_times);
        if length(all_spike_times) >= 50
            firing_rates(k) = 1000/mean(ISI_vals);
        else
            firing_rates(k) = 0;
        end
    else
        firing_rates(k) = 0;
    end
end
```

Figure 2: Hodgkin-Huxley model implementation with action potential detection and fire rate calculation

```matlab
% Model Parameters
C = 1;% nF
G_L = 50;% nS
V_L = -65;% mV
V_spk = -45;% mV
V_r = -65;% mV
t_arp = 2;% ms
I0 = 1.1; %nA

% Simulation Parameters
dt = 0.01;% ms
total_time = 200;% ms
t = 0:dt:total_time;
num_steps = length(t);

I_e = zeros(1, num_steps);
t0 = 40;
I_e(t >= t0) = I0;
V = zeros(1, num_steps);
V(1) = V_L;
refractory_time = 0;
tic
for i = 1:num_steps-1
    if refractory_time > t(i)
        V(i+1) = V_r;
    else
        dV_dt = (-G_L/1000 * (V(i) - V_L) + I_e(i)) / C;
        V(i + 1) = V(i) + (dV_dt * dt);
        if V(i+1) >= V_spk
            refractory_time = t(i+1) + t_arp;
            V(i+1) = V_r;
        end
    end
end
toc
```

Figure 3: MATLAB implementation of Leaky Integrate-and-Fire model

```
dt = 0.2;% ms
total_time = 2000;% ms
t = 0:dt:total_time;
num_steps = length(t);
Ie_vals = linspace(0,2, 100);
firing_rates = zeros(1, 100);


for k = 1:100
    I_e = zeros(1, num_steps);
    t0 = 40;
    I_e(t >= t0) = Ie_vals(k);
    V = zeros(1, num_steps);
    V(1) = V_L;
    refractory_time = 0;
    all_spike_times = [];

    for i = 1:num_steps-1
        if refractory_time > t(i)
            V(i+1) = V_r;
        else
            dV_dt = (-G_L/1000 * (V(i) - V_L) + I_e(i)) / C;
            V(i + 1) = V(i) + (dV_dt * dt);
            if V(i+1) >= V_spk
                refractory_time = t(i+1) + t_arp;
                V(i+1) = V_r;
                all_spike_times = [all_spike_times, t(i)];
            end
        end
    end
    if length(all_spike_times) > 1
        ISI_vals = diff(all_spike_times);
        firing_rates(k) = 1000/mean(ISI_vals);
    else
        firing_rates(k) = 0;
    end
end
```

Figure 4: Leaky Integrate-and-Fire model implementation with fire rate calculation

Results and Discussion

We begin with an initial simulation of the Hodgkin-Huxley model with a runtime of 200 ms and time step, dt, of .01 ms. The external current applied is 0 pA from 0 ms to 39 ms and 200 pA starting from 40 ms till end of the simulation at 200 ms. We plot the membrane potential, membrane current, sodium conductance, potassium conductance and the external current into 5 subplots. The plots are shown in Figure 5.

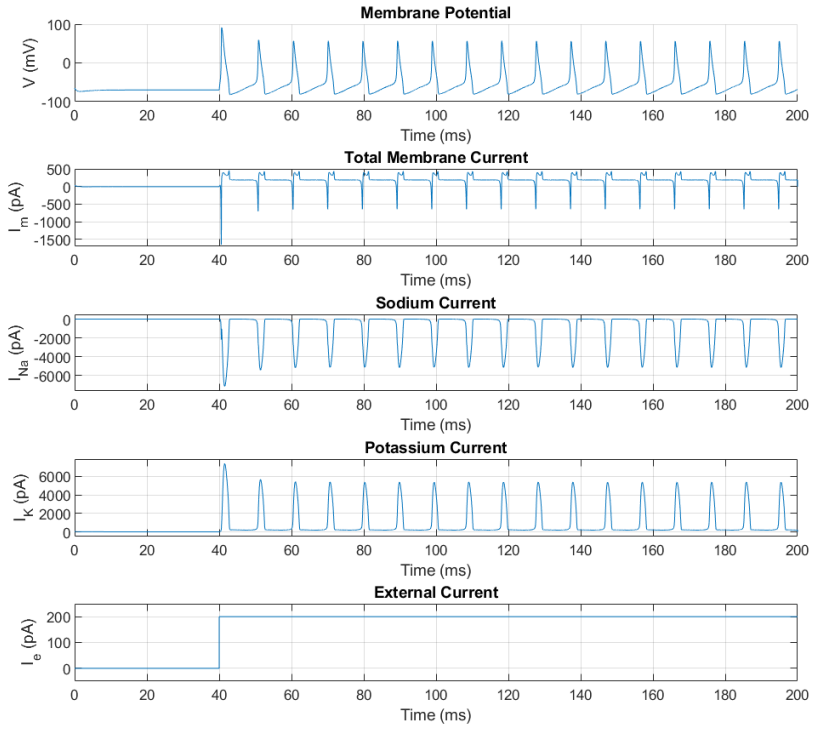


Figure 5: HH model simulation results with I0 = 200 pA at time $\geq$ 40ms and Forward Euler method time step of .01 ms

The above Figure demonstrates a consistent repetitive firing. The sodium conductance peaks rapidly to initiate spiking of the membrane potential through depolarization. When the membrane potential reaches the spike threshold, the potassium channels open up and repolarize the membrane

back to resting potential. We can modify our code to detect each 'spike'. To do so, we define a voltage threshold for which we say that a spike that has been emitted, in our case that threshold is -20 mV. The resulting code is given in Figure 2. Figure 6 illustrates the dynamics of one single action potential, marking the time at which V_{spk} was reached.

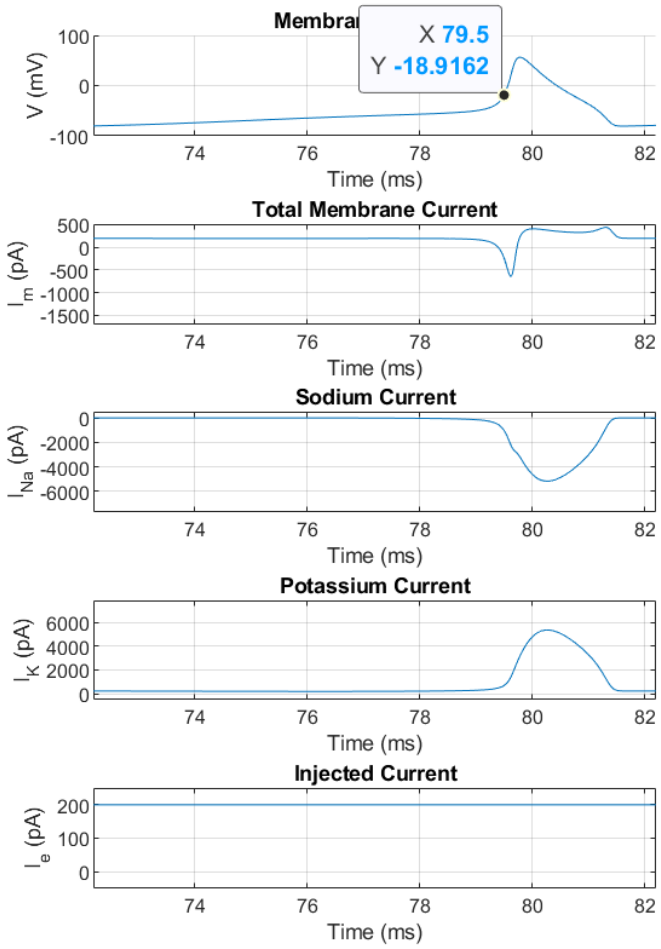


Figure 6: Zoomed in view of a single action potential at time between 72.2 ms and 82.2 ms. V_{spk} occurs at around 79.5 ms.

Using the spike detection mechanism we built, we found that the min-

imum input current I_1 required to generate one action potential is 19 mV. Figure 7 gives the plot when $I_e = 19$ pA at time ≥ 40 ms.

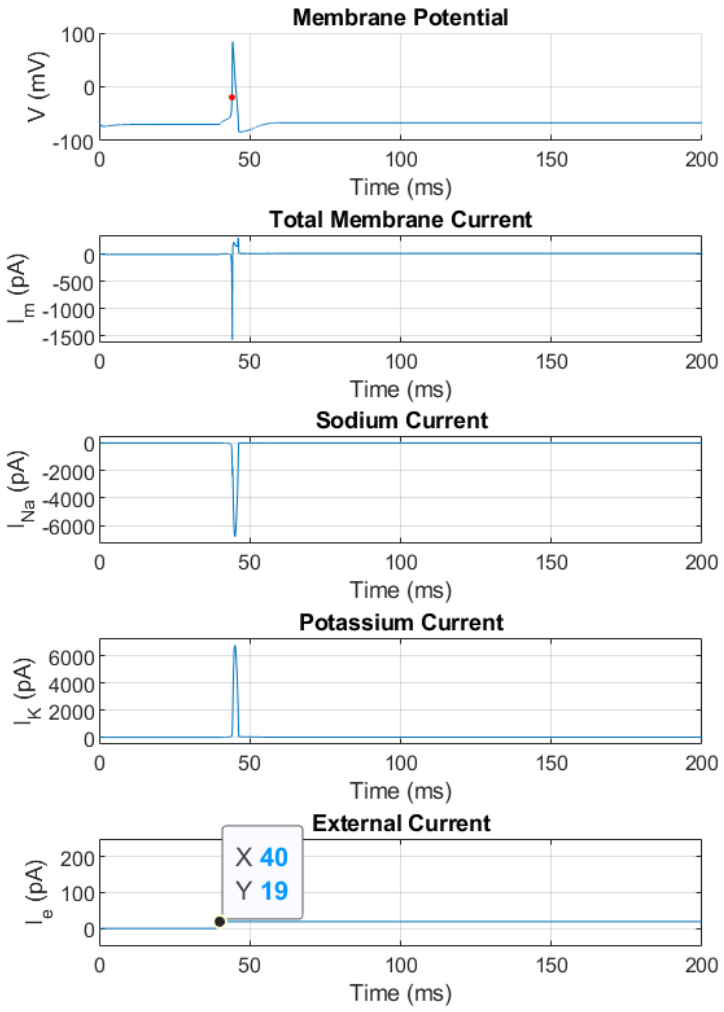


Figure 7: Firing of single action potential when an external current of 19 pA is introduced to the system

By increasing input current beyond I_1, eventually 2, 3 or more action potentials will generate but then the neuron stops firing again. Rheobase

current is the minimum current above which repetitive firing ignites. In our simulation, the rheobase current, I_{rh}, is 109 pA. Figure 8 gives the plot when $I_e = I_{rh}$. We extended the simulation runtime to 1000 ms to ensure that we achieved repetitive firing and that the activation potential does not stop firing later on.

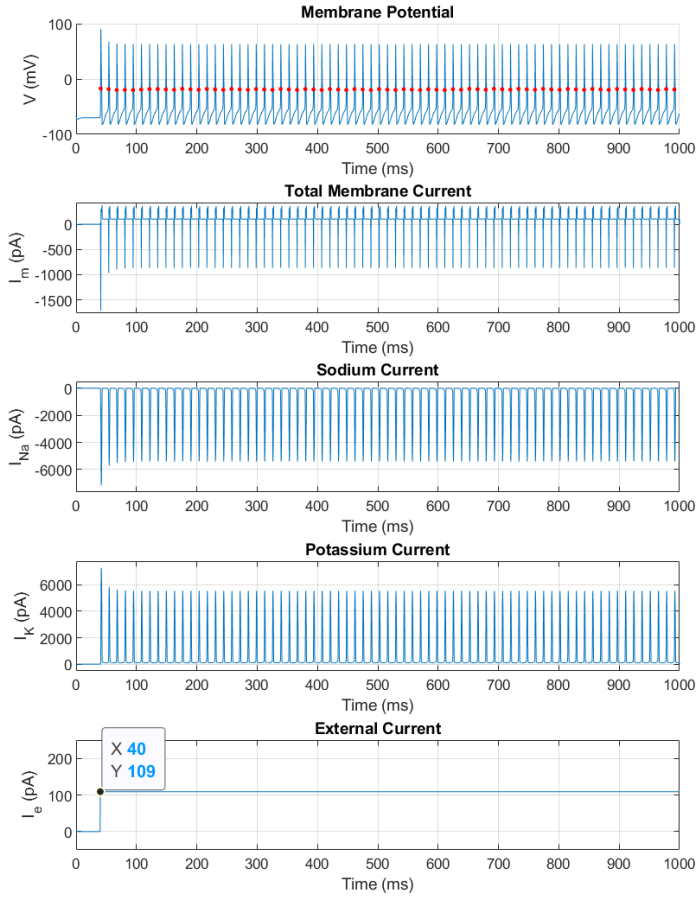


Figure 8: Result of simulation when input current is equal to rheobase current at 109 pA when time $\geq$ 40 ms

To further investigate the relationship between firing rate and input current in HH model, we ran simulations for 40 different input current val-

ues with increasing amplitude. The current values ranged from just below I=$I_{rh}$to significantly above it, specifically from 106 pA to 145 pA. For all input current where the neuron emits only a few action potentials before stopping, the firing rate was set to zero. To calculate the firing rate for each current value, we find all interspike intervals, time between two consecutive spike, and divided 1000 ms with the mean ISI to get spikes per second. The resulting f-I curve is given in Figure 9.

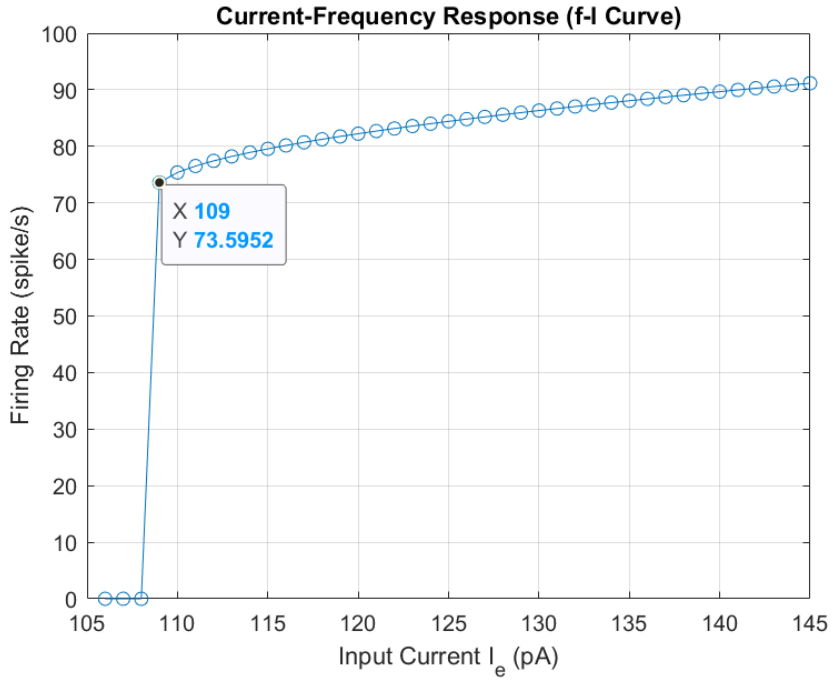


Figure 9: f-I curve generated for input currents between 106 pA and 145 pA

The f-I curve shows that the firing rate remains at zero for input currents below the rheobase. At the rheobase current of approximately 109 pA, there is a sudden jump in firing rate. This is reminiscent of a Type II firing rate. After the jump, the firing rate increases linearly with current. The curve indicates that the actual rheobase is likely somewhere between 108 pA and 109 pA. While reducing the interval between input current can give a more accurate estimation of the rheobase, we cannot get the exact rheobase value without an analytical solution to the model equation. The discontinuous nature of the firing rate suggests that the HH model is not an efficient model for an encode graded input (spanning a continuous range of input currents).

Because there is a huge jump at the rheobase, there is no continues increase in firing rate until we reach the rheobase. As such, ecnoding low-intensity signals can be more difficult for the HH model since we have a firing rate of zero below the rheobase and at rheobase we make a discontinuous jump from 0 to a very high firing rate.

We repeat the previous steps for the LIF model with $I_e = 1.1$ nA, simulating for 200 ms using a time step, dt, of .1 ms. The resulting plots of the membrane potential and the input current are given in Figure 10.

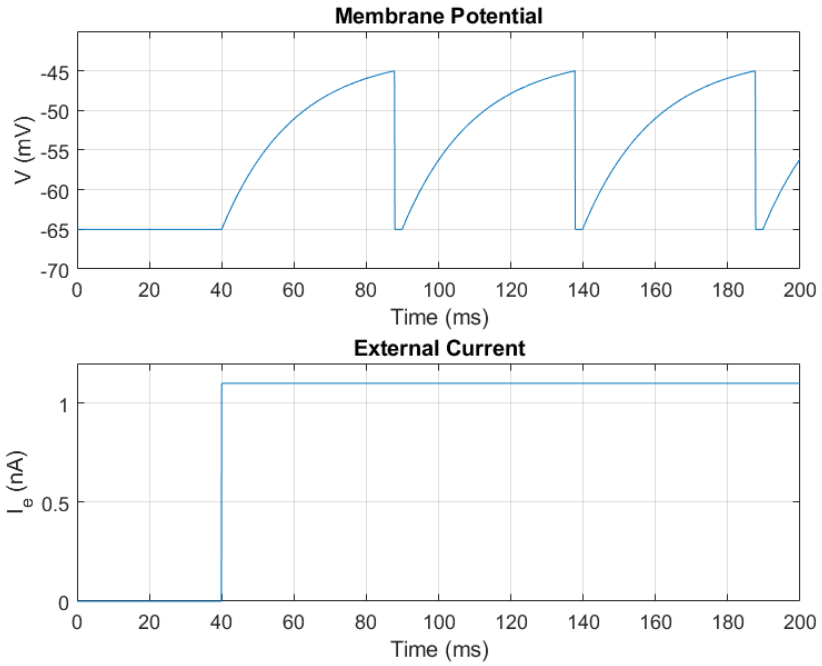


Figure 10: Simulation results of LIF model with input current of 1.1 nA at time $\geq$ 40 ms

We make an f-I curve for the LIF model with 100 currents at regular intervals between 0 nA and 2 nA. We use the same method as the HH model to calculate the firing rate. The curve generated for the model is shown in Figure 11.

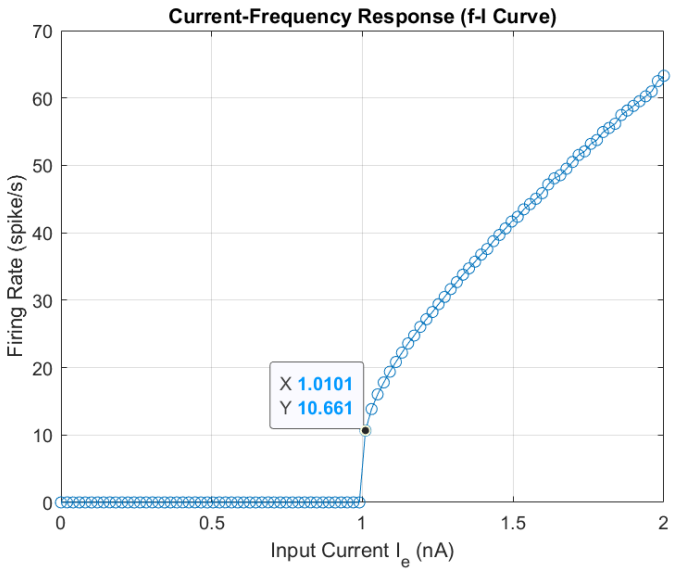


Figure 11: F-I curve of the LIF model with input current ranging from 0 nA to 2 nA

The f-I curve for demonstrates Type i firing, where the firing rate gradually increases from 0 as the input current increases. This is different from the HH f-I curve where the firing rate would jump from zero (below the bifurcation) to a finite positive value as shown in Figure 9. In addition, unlike the HH model, it is not possible for the LIF model to have only a few action potentials and then stop firing. This is because in the LIF model, the membrane potential increases linearly in response to constant input. If the external current is below the rheobase, the membrane current stabilizes at a voltage below the threshold. However, once rheobase current is applied, the membrane potential will always reach the threshold, leading to repetitive firing without ever stopping. The theoretical rheobase current for a LIF model can be found using the equation $G_L(V_{spk} - V_L)$. With our given model parameters, the theoratical rheobase current is 1 nA. The LIF f-I curve shown in Figure 11 seems to match the theoretical rheobase current; in the curve, the first firing rate above 0 occurs at 1.0101 nA.

The HH mode fails to work at larger time step sizes beyond .01. When the simulation is run with a dt =.1 ms and dt = .05ms, the simulation results are thrown off completely. One action potential is generated in both cases but then the membrane potential seems to keep increasing infinity. At 200 pA input current, the plot generated for a simulation with dt=.05ms is shown

in Figure 12.

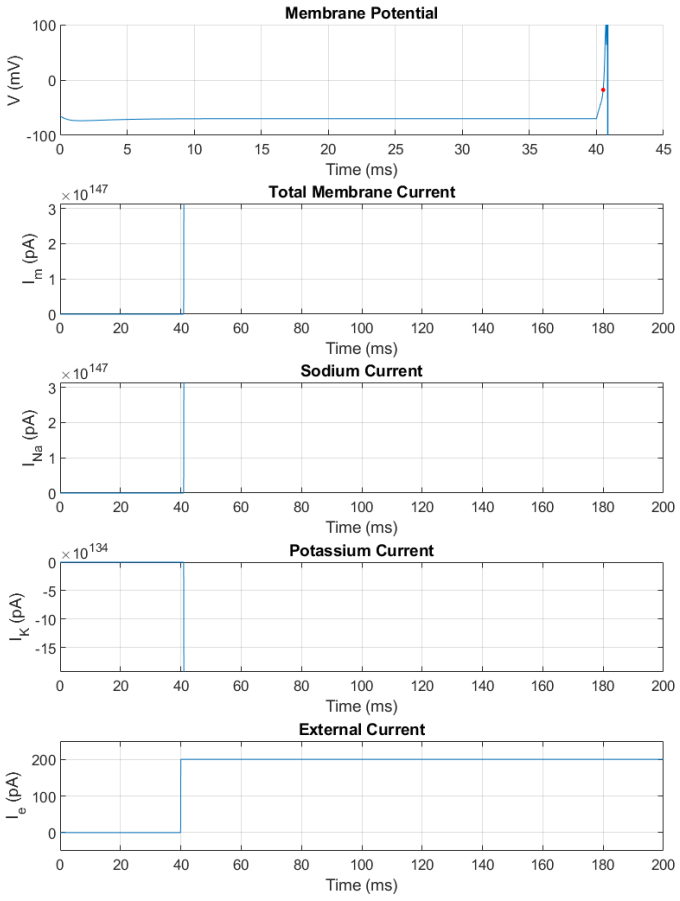


Figure 12: Simulation result of HH model at 200 pA when dt=.05ms

In contrast, if we increase the step size dt for the LIF model to .2ms, the model still is able to work relatively well for lower input currents. Figure 13 gives f-I curve of the LIF model when dt = .2 ms.

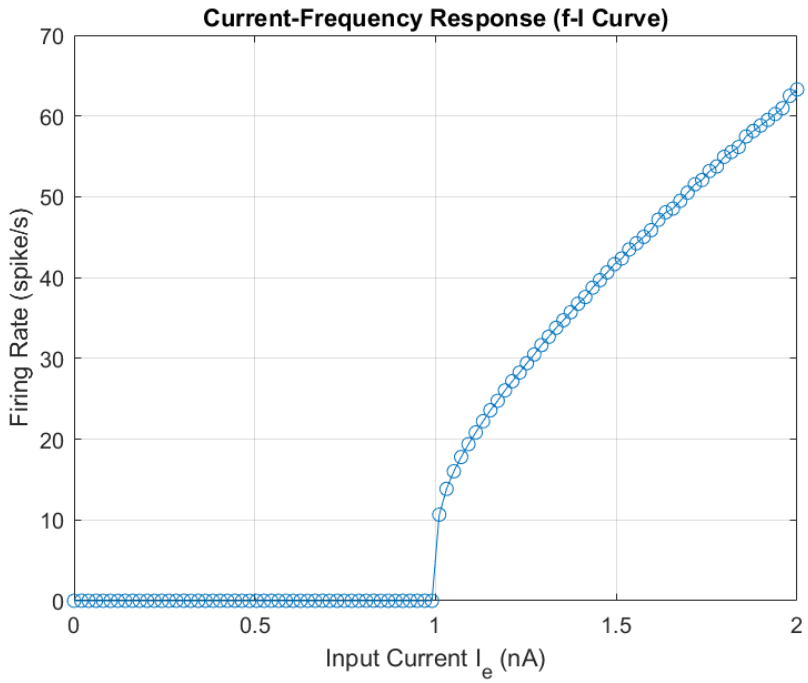


Figure 13: LIF f-I curve when dt = .2 ms

From the LIF f-I curve we observe that at higher input currents, the curve becomes less smooth and instead makes sharper changes between firing rates. Despite this, the LIF model is still able to generate a reasonable simulation for neuron behavior even with relatively larger step sizes. The Hodgkin-Huxley model on the otherhand requires smaller time steps, dt, to maintain a stable simulation. To further demonstrate the efficiency of the LIF model vs the HH model, we can use the tic/toc functions provided by matlab to calculate the runtime of each model. Each model was simulated using two different time steps: dt = .01 ms and dt = .002 ms. In all cases. the simulations were run once with a single input current over a fixed time window of 200 ms. At dt = .01 ms the LIF model took .002018 seconds to complete while the HH model took .011821 seconds to complete. At dt = .002 ms, the LIF model took .002805 seconds while the HH model took .041825 seconds. In both cases, the HH model required an order of magnitude more time to complete the simulation. As we see from these results, the HH model has a significantly more computationally expensive. This cost will increase exponentially if the model is scaled up to simulate large networks of neurons.

Conclusion

The Hodgkin-Huxley and Leaky Integrate-and-Fire models both play important roles in computational neuroscience. They each offer their own advantages when it comes to detail vs computational efficiency. The HH model accurately represents the roles of ion channels in producing action potentials but requires smaller time steps and longer runtimes, which makes it computationally expensive. Its f–I curve shows Type II firing, with a sudden jump in firing rate at the rheobase, which limits its ability to encode graded input at lower input currents.

On the other hand, the LIF model is a much simpler model which handles spikes and recovery periods through the use of boundary conditions rather than modeling ion channel behavior. It demonstrates Type I firing where there is a gradual increase in firing rate. Additionally, the LIF model remains stable at larger time steps and completes simulations much faster than the HH model. This makes it better for simulating large networks of neurons where computational efficiency matters.

In summary, the HH model better for detailed single-neuron simulation, while the LIF model is easier to scale for large neural network models.