UNIVERITY OF TEHRAN

SINGLE SPIKING NEURON MODEL AND ANALYSIS

IMPLEMENTATION OF SINGLE SPIKING NEURON MODELS, VARIATIONS OF LIF MODELS AND THEIR BEHAVIORS ANALYSIS

AUTHOR

ARASH NIKZAD

Student No. 610301195

SUPERVISOR

MOHAMMAD GANJTABESH

TEHRAN, MARCH 2024

Contents

1 Introduction

These days with the focus on artificial intelligence and their huge impact on our daily lives, the study of how our brains work is a crucial question, humanity is craving to answer. With help of Computational Neuroscience, we are able to model the functionality of our brain (to some extent) and run analysis on these models, figuring out the mysteries of our brain one by one. The first step in modeling the complex and unkown model of our brain is to go back to its very beginning, their building blocks, **neurons**. If we can find a sufficent single model of a neuron, we can procced by leveling up our system to synapses and neural populations and finlly we can hope for a model for our brains. In this report, we tried to implement the first single spiking neuron model (**LIF model**) and its few variations and analyse its behavior on different input currents and its parameters.

2

LIF Model Implementation

In this section, we introduce the dynamics of a simple LIF model and its variants including Exponential LIF (**ELIF**) and Adaptive Exponential LIF (**AELIF**) in structure of **`pymoNNtorch`** framework.Providing sudo-code and its initial parameters. In the next section we see its behavior on various input currents and different noise input.

2.1 simple LIF model

The LIF model is simply an adaptation of the **passive membrance** equations as a computational model which works linearly in accordance to the input, simply by adding a threshold of θ for a spike and change of dynamics into firing an action potential and reseting its potential $u(t)$.

Leaky Integrate-and-Fire model dynamics

$$\tau \frac{du}{dt} = -(u - u_{rest}) + R.I(t) \tag{2.1}$$

$$if\ u(t) = \theta \longrightarrow Fire\ Spike + Reset\ (u = u_{rest}) \tag{2.2}$$

With this Model we hope to get an accurate timing of spikes in a real neuron rather than a descriptive model of biological and biophysical behaviors of a neuron. Below there is an implementation of a simple LIF model in `pymoNNtorch`, consisting of `initialize` function and `forward` function.

Listing 2.1: *initialize function*

```
1 class LIF_Behavior(Behavior) :
2     ...
3     def initialize(self, neuron) :
4         super().initialize(neuron)
5
6         neuron.cnt = 0
7
```

```
8         self.threshold = -10
9         neuron.reset = -80
10        neuron.rest = -65
11        neuron.tau = 10
12        neuron.dt = 0.1
13        neuron.R = 10
14
15        neuron.voltage = neuron.vector("ones") * neuron.rest
```

Listing 2.2: *forward function*

```
1     class LIF_Behavior(Behavior) :
2         ...
3         def forward(self, neuron) :
4             I = input_current(2, neuron.cnt, hasNoise=False)
5             neuron.cnt += 1
6
7             firing = neuron.voltage >= self.threshold
8             neuron.spike = firing.byte()
9             neuron.voltage[firing] = neuron.reset
10
11            dV = (
12                -(neuron.voltage - neuron.rest)
13                + neuron.R*I
14                ) / neuron.tau
15            neuron.voltage += dV * neuron.dt
```

2.2 Exponential LIF (ELIF)

Due to LIF model's simplicity, it can not account for highly nonlinear activities which are a common theme is neurons interaction, so in order to make our model more accurate we need to add nonlinearity to its dynamics. We can use various functions to make it nonlinear but, two good options are **exponential** and **quadratic** functions. In this report, we only have implemented the exponential LIF, that can account for nonlinear activities as well as giving us power to implement real spiking mechanism and overshooting instead of simply reseting after reaching the threshold potential.

Exponential LIF model dynamics

$$\tau \frac{du}{dt} = -(u - u_{rest}) + \Delta_T exp(\frac{u - \theta_{rh}}{\Delta_T}) + R.I(t) \tag{2.3}$$

$$if\ u(t) = \theta_{rh} \rightarrow Fire\ Spike + Reset\ (u = u_{rest}) \tag{2.4}$$

The Implementation of Exponential LIF (**ELIF**) is the same as basic LIF only applying new attributes for the exponential part and adding the change of dynamic in its differential equation.

Listing 2.3: *change in "initialize" function*

```
    # Exponential parameters

    self.rh_threshold = -10
    neuron.delta = 5
    neuron.spike_zone = False
```

Listing 2.4: *change in "forward" function*

```
def forward(self, neuron) :
    ...
    firing = torch.BoolTensor(
        [(neuron.voltage >= self.rh_threshold and not neuron.spike_zone)]
    )

    if(firing) :
        neuron.spike_zone = True
    elif(not firing and neuron.voltage < self.rh_threshold) :
        neuron.spike_zone = False

    neuron.spike = firing.byte()

    reset_need = neuron.voltage >= self.threshold
    neuron.voltage[reset_need] = neuron.reset

    dV += + neuron.delta*torch.exp(torch.tensor([(neuron.voltage - self.rh_threshold)
```

2.3 Adaptive ELIF (AELIF)

One important capability of a real neuron is the ability to adapt to its input, meaning most neuron change behavior when getting the same input current over and over again, e.g. facing an input current cuasing a spike does not get all neurons of all types into **bursting** phase, instead most of them adapt to the input current and the spike intervals extends to the point that it would not fire an action potential in response to the given input, which gives us the idea of **adaptive** spiking neurons. This can be achived simply by adding a adaptive variable (or variables) `w`, and adding a new dynamic for it based on number of spikes of a neuron.

Adaptive ELIF model dynamics

$$\tau_m \frac{du}{dt} = -(u - u_{rest}) + \Delta_T exp(\frac{u - \theta_{rh}}{\Delta_T}) - Rw + R.I(t) \tag{2.5}$$

$$\tau_w \frac{dw}{dt} = a(u - u_{rest}) - w + b\tau_w \sum_{t^f} \delta(t - t^f) \tag{2.6}$$

$$if\ u(t) = \theta_{rh} \rightarrow Fire\ Spike + Reset\ (u = u_{rest}) \tag{2.7}$$

There are few new variable including `a, b,` τ_w, which are constants, for subthreshold and spike-triggered adaptations, these variables can be fine-tuned when using a real data. the changes of implementation in both `initialize` and `forward` functions of **ELIF** model will be listed below.

Listing 2.5: *changes in "initialize" function*

```
def initialize(self, neuron) :
    ...
    # Adaptability
    # ------------------------------------------------------------

    neuron.a = 0.5 # sub-threshold adaptation variable
    neuron.b = 1 # spike-triggered adaptation variable

    neuron.w = 0 # adaptive variable
    neuron.adpt_tau = 50 # time-constant for w
```

Listing 2.6: *changes in "forward" function*

```
def forward(self, neuron) :
    ...
    dV -= neuron.R*neuron.w # voltage potential changes
    dW = ( # dynamics of adaptive variable itself
            + neuron.a*(neuron.voltage - neuron.rest)
            - neuron.w
            + neuron.b*neuron.adpt_tau*(firing.byte())
        ) / neuron.adpt_tau

    neuron.w += dW * neuron.dt
```

In the next section, we are going to see these models performances on various input currents, including step function, trigonometric functions and random customized functions with, or without, noise.

3

Model behavior on input current

In this section we focus on behavior of the models implemented in Chapter 2 when getting different input currents and various noisy input, also because we do not have any given data from a neuron, there are numerous settings for our model's parameters to play around with.

3.1 LIF model

We are going to input, three diffrent types of currents, with, or without noise and see the changes of potential and our models spikes responding to each input current. These are the following input currents, which will be used in our experiment.

- **step function** : $I(t) = (t > 200) * 20$
- **trigonometric function** : $I(t) = 3\sin^2(2\pi t)$
- **random function** : $I(t)$ is just random noise

Here, we have the implementation of our `input current` function, used to simulate the input of our model in each iteration using the number of iterations as `cnt`, the power of each current as a multiplier `pw`, the type of current to use as `mode` and finally whether to have noise or not identified by parameter `hasNosie`.

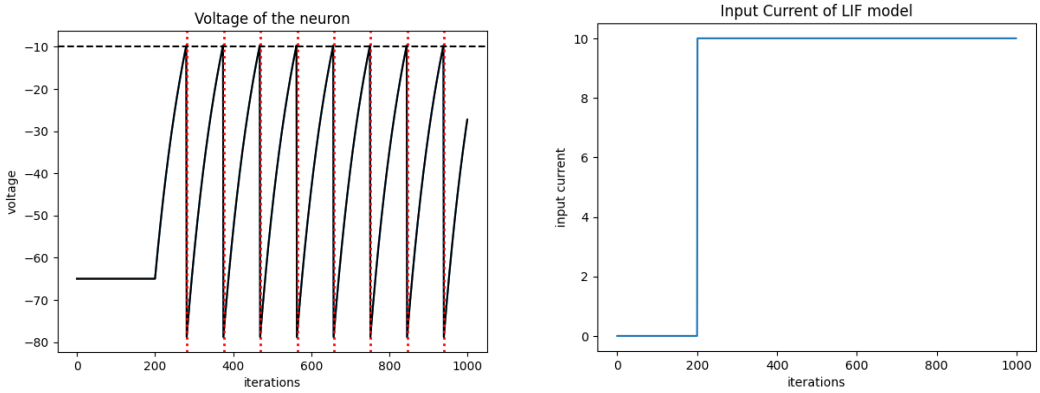
Listing 3.1: *input current function implementation*

```
1  def input_current(mode, cnt, hasNoise = False, pw = 20) :
2      if(mode == 1) :
3          return (math.sin(cnt/360 * np.pi))**2 * pw + hasNoise * random.random()
4      elif(mode == 2) :
5          return (cnt > 200) * pw + hasNoise * random.random()
6      elif(mode == 3) :
7          return random.random() * pw + 0.5 + hasNoise * random.random()
8      elif(mode == 4) :
9          a, res, state = np.random.rand(5) * self.pw, 0, 1
10         for i in a :
11             res += state * i, state *= -1
```

```
12        return res
```

3.1.1 step function

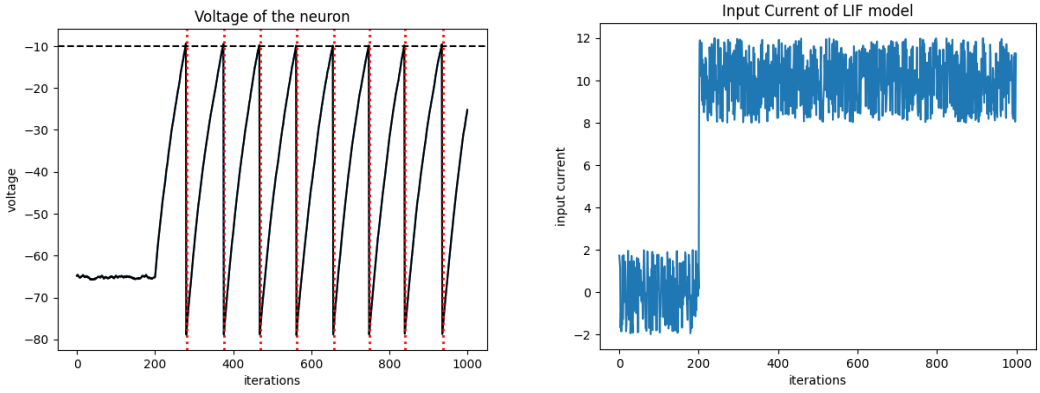
Here we present the plot of both our input current and the model potential at each step in 1000 iterations each corresponding to `dt = 1ms`. Initial value of parameters of the model are set the same as it was shown in the implementation of the model in Chapter 2.



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.1: *LIF on step function without noise*

As expected after a current of sufficient power to make a neuron fire an action potential, it will go into a **bursting phase** with the same frequency. Now we want to see the voltage changes with noise but still using the base function as step function.



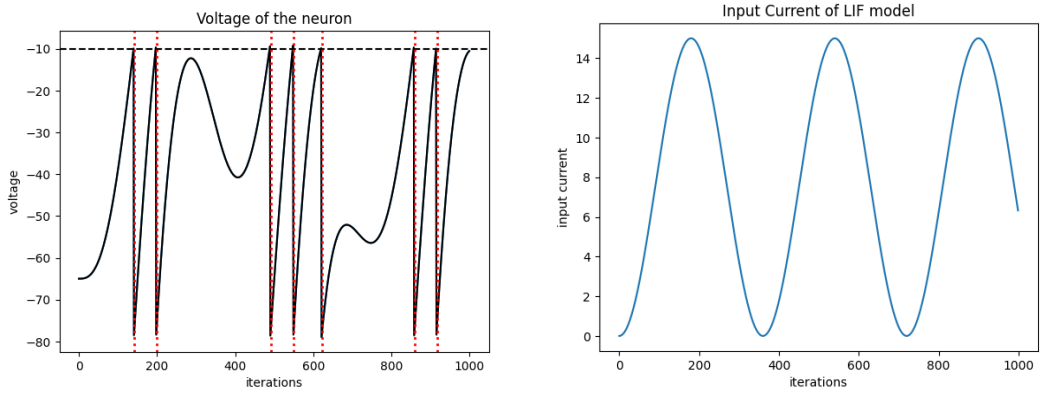
(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.2: *LIF on step function with noise*

As we can see, the the potential of our model has more fluctuation due to the very noisy input, however due to power of steady input current, there is not much of difference in its behavior except the spiking frequency has weakened.

3.1.2 trigonometric function

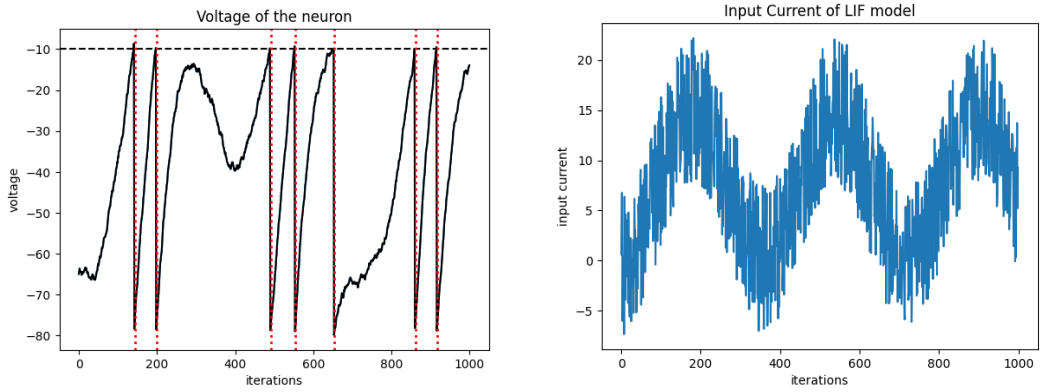
Due to the periodic attribute of trigonometric function, we can expect a different behavior from our model comparing to using a constant input current.



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.3: *LIF on trigonometric function with no noise*

As we can see, its behavior is totally different from step function, however we can still see that in a period which the sin() function peaks, we still get into a **bursting phase** but we can obviously see its recurring behavior which spikes with around the same frequency, now it is time to add some noise and analyse its behavior.



(a) *LIF potential per iteration* **(b)** *input current per iteration*

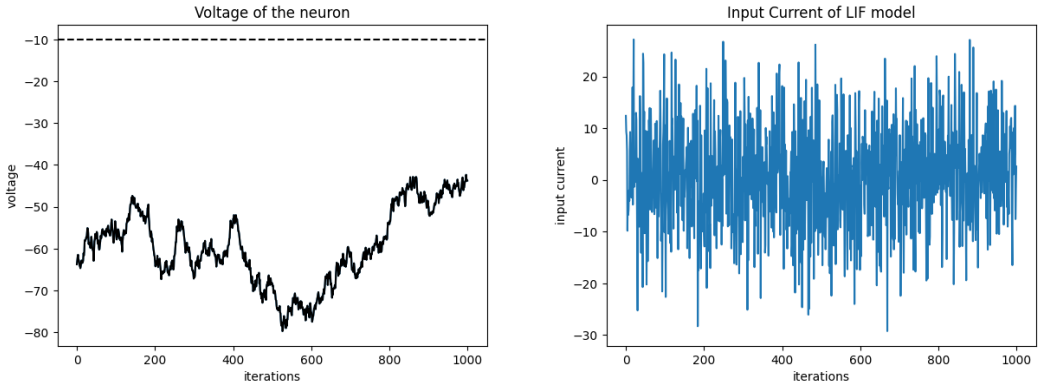
Figure 3.4: *LIF on trigonometric function with noise*

Here we increased the noise a little bit more compare to the first experiment. We can see that its fluctuation as increased considerably but still the final spiking pattern has not changed, only the frequency and its sub-threshold regime.

3.1.3 random function

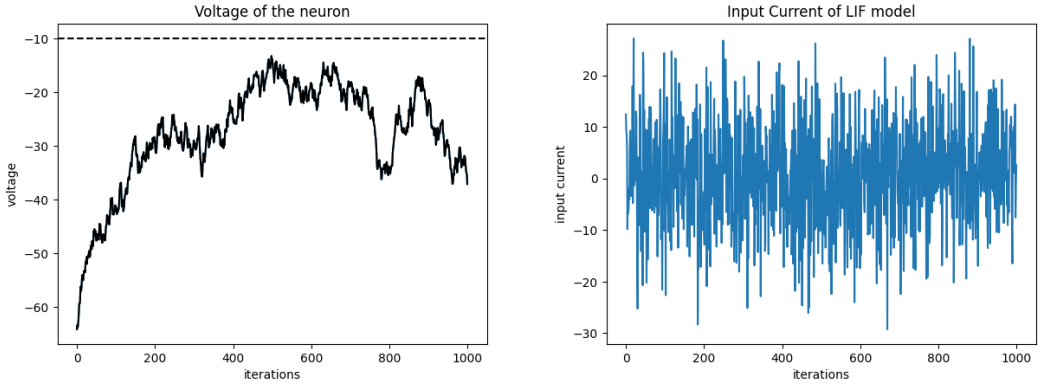
Now that we want to enter a random input current into our model we expect various behaviors of the neuron as most of real-world input pattern does not have a specific pattern or origination, so our model's performance on these types of input is an important

topic to consider. Here we tried different ranges of `pw` and more importantly three different ranges of random noise to see the model's behavior.



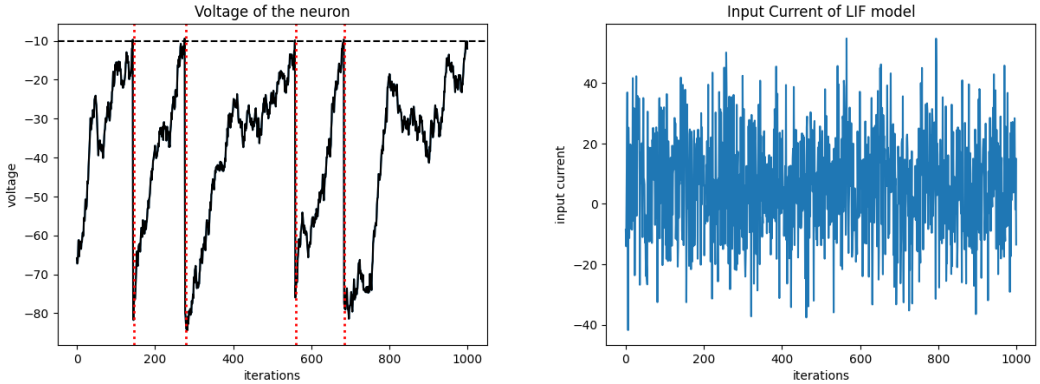
(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.5: *LIF on random function with low current*



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.6: *LIF on random function with mid current*



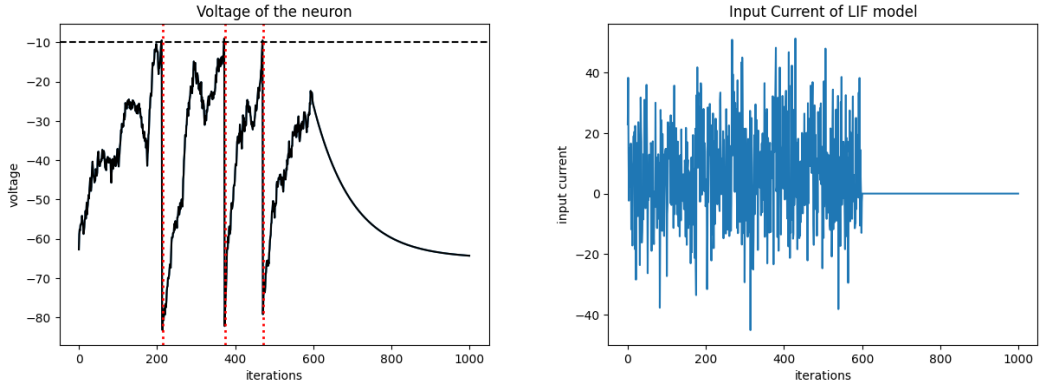
(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.7: *LIF on random function with high current*

As we can see, in Figure 3.5, due to the fact that the random input current, moves

around $(-10, +10)mA$, the model's potential never reaches or even gets close to the threshold. The second one, is no different in the sense that it did not fire a spike in this period, nevertheless, it came close to reaching the threshold and firing an action potential. The third input current, on the ther hand, is different. This time due to the high volume of input current and because most of its values are in range $(-10, +30)mA$ it reached the threshold and fired an spike multiple times. Of course its fluctuation is no where near the other two experiments but still our model has a good control and natural behavior on a random noisy input which is what we expect from a our LIF model.

finally one thing to consider it the **leakage** of our neuron that we have not tested yet, meaning, once the input cuts off its potential has to decay to its original u_{rest}. in order to test this feature, we started with a random input and after 600 iterations we cut of the input current.



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.8: *LIF on random function and its leaky feature*

Fortunately, we can see that after we cut off the input current, the neuron's potential decays back to the resting potential as expected.

3.2 ELIF model

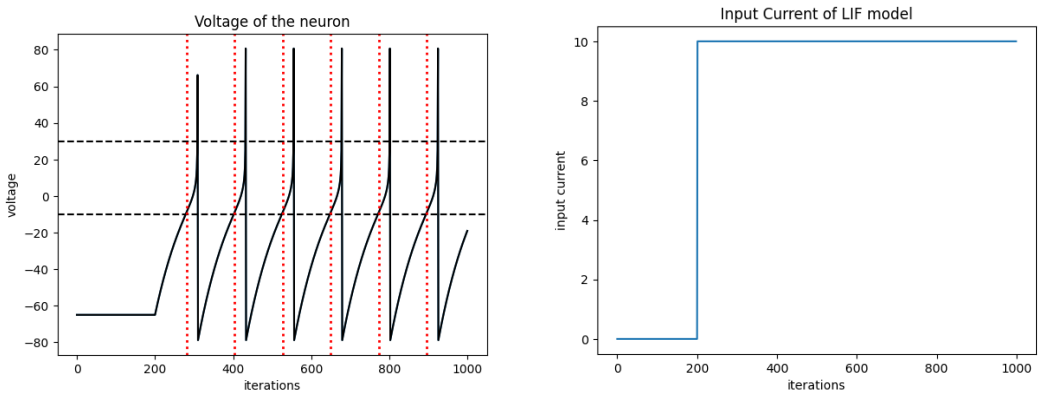
Now, it is time to see the behavior of our exponential LIF on these input currents. We can analyse its behavior independantly and also we can compare it to LIF and see the changes of behavior in both spikes and sub-threshold regime.

3.2.1 step function

Our first input current as before is the step function, we still expect a behavior of bursting of some sort however this time we are looking for spikes in a more natural way.

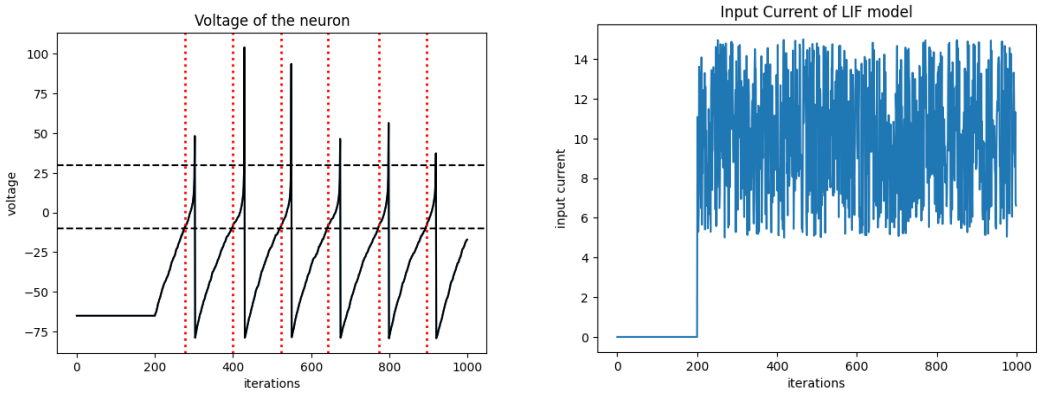
We can see in Figure 3.9 that our expectations are met, also it is nice to see that with use of the sudden changes in the exponential function after θ_{rh} we can mimic firing of a spike considerably well. Now its time to see ELIF behavior on noisy data.

Again,as we can see in Figure 3.10, no particular changes happened due to the fact that our noise is still weaker than the constant current entering the system so the rate of



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.9: *ELIF on step function with no noise*



(a) *LIF potential per iteration* **(b)** *input current per iteration*

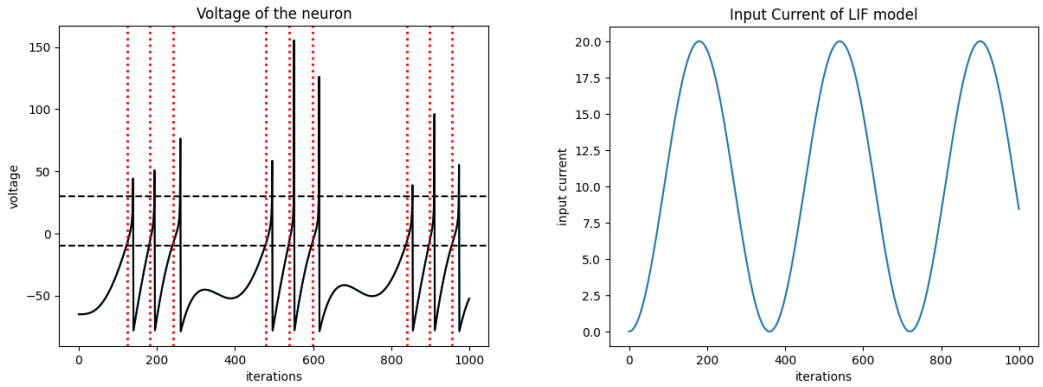
Figure 3.10: *ELIF on step function with noise*

changes is always positive resulting in somewhat the same result only this time with more fluctuation of course.

3.2.2 trigonometric function

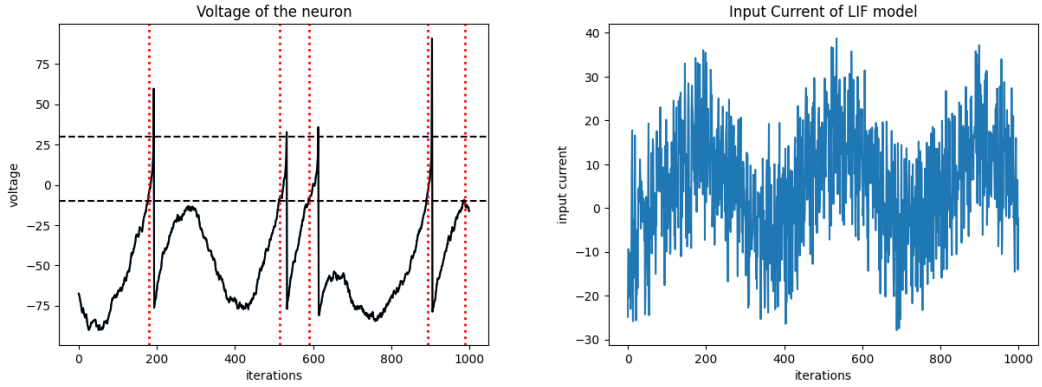
Same as LIF, no particular difference are expected to be noticed, simply trying to see whether the model's behavior matches our expectations.

Nothing special for further analyse on the trigonometric input current without noise, however we can see the effects of a heavy noise on Figure 3.12, beside the fluctuation and even changes of firing frequency, one important point to grasp is the final spike which has only reached the θ_{rh} but could not fire an after spike which was not a feature we had while using simle LIF, but now, with help of the exponential LIF we were able to achieve that.



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.11: *ELIF on trigonometric function with no noise*

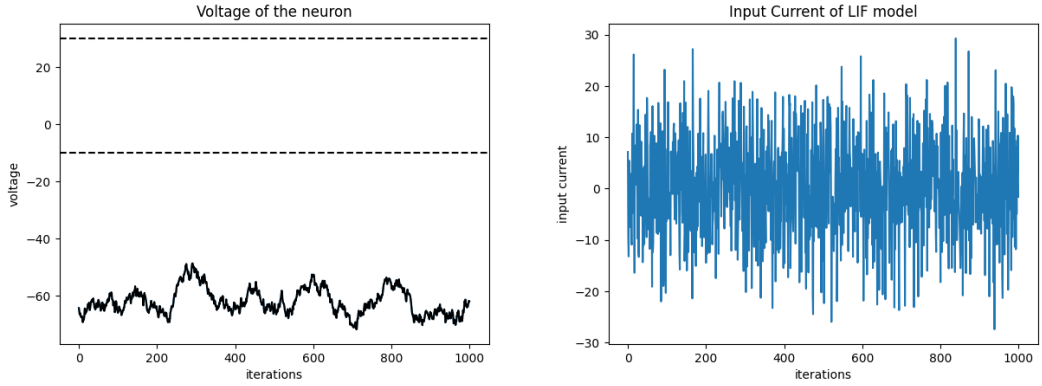


(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.12: *ELIF on trigonometric function with noise*

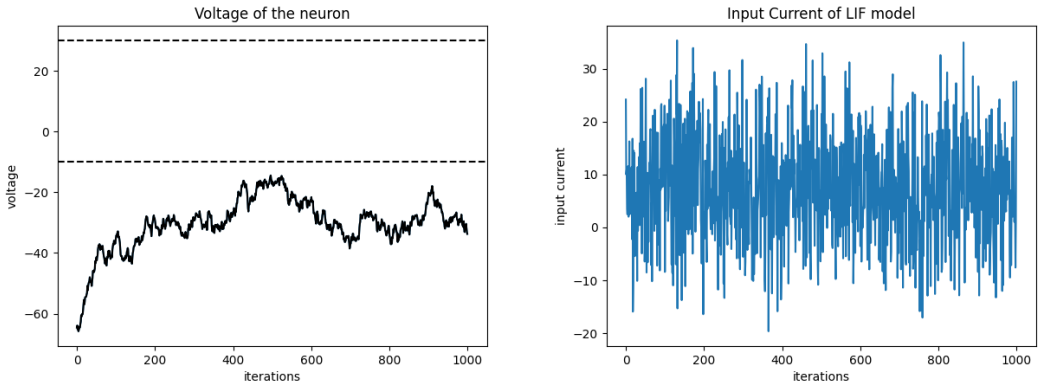
3.2.3 random function

Here again we are going to see the performance of our model, facing a random input current. again we are going to see its changes of potential in three levels of noise.



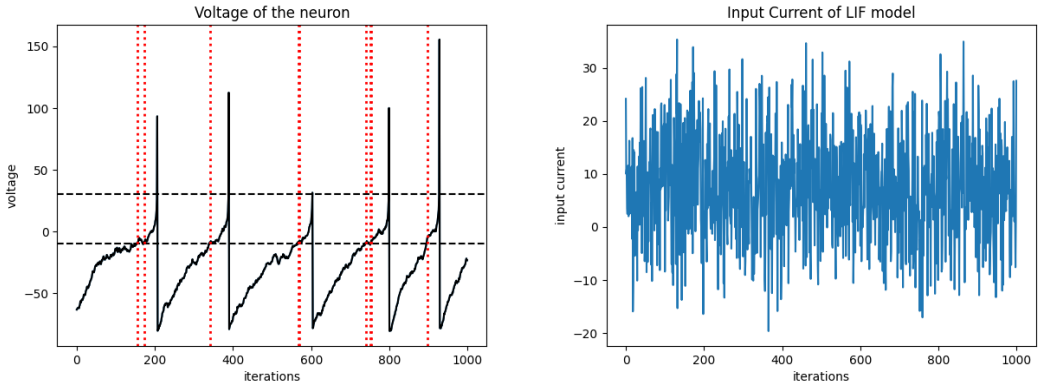
(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.13: *ELIF on random function with low current*



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.14: *ELIF on random function with mid current*



(a) *LIF potential per iteration* **(b)** *input current per iteration*

Figure 3.15: *ELIF on random function with high current*

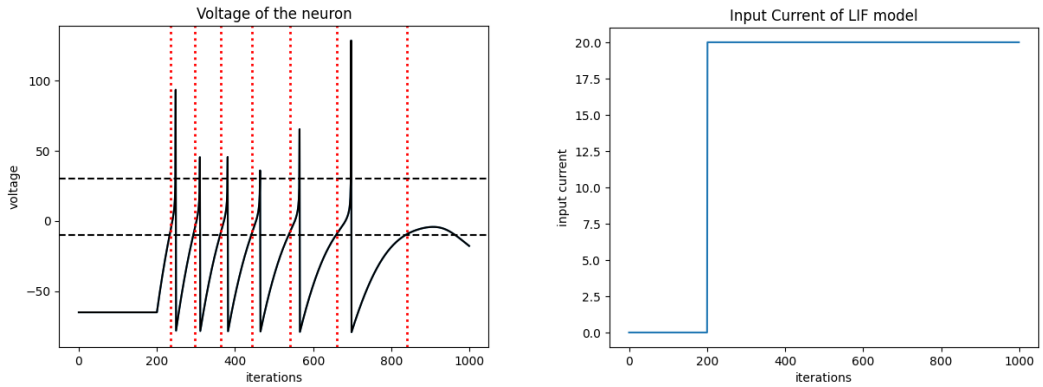
Well, we can say that our model does whatever we told it to do, however, one important topic is that after using two thresholds, θ_{rh} and θ_{spike}, we can see in Figure 3.15, there are two sets of spikes, really close to each other and both being recognised as spikes. In real-life neurons or even more high-level neuron models, we do not see such activity, the reason behind this is an action called **refractory** in neurons which prevents any spikes in a short period of time after an action potential has fired. We will implement this feature in Chapter 5.

3.3 AELIF model

Working with an adaptive model like AELIF, gives us important features and different behaviors comparing to two previous models that we discussed. Our crucial expectation from this model is its adaptability particularly on a steady input of current which will test and see whether we have achived that or not.

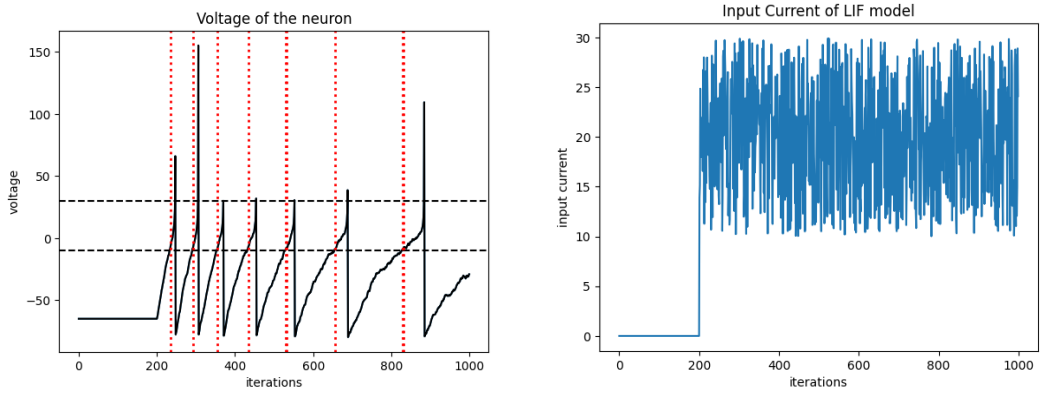
3.3.1 step function

Unlike the other two previous models where their behavior on step function was trivial and of low importance, here one can argue that AELIF behavior on a constant input is the critical reaction we are seeking to achieve.



(a) *model potential per iteration* **(b)** *input current per iteration*

Figure 3.16: *AELIF on step function with no noise*



(a) *model potential per iteration* **(b)** *input current per iteration*

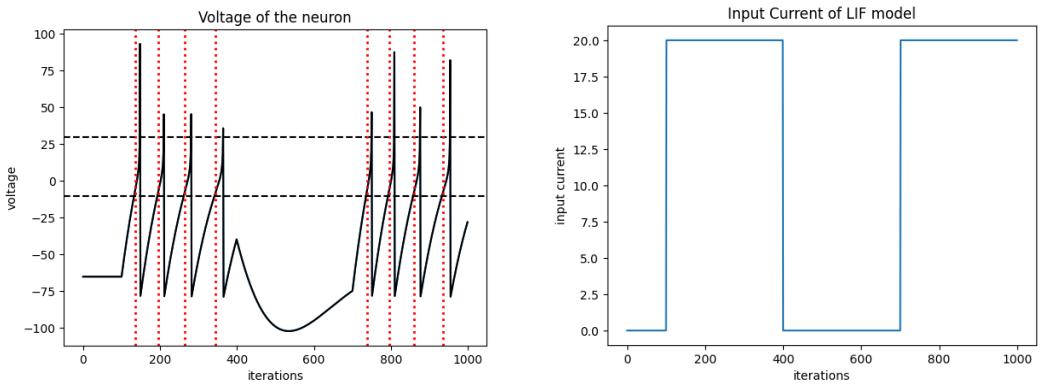
Figure 3.17: *AELIF on step function with noise*

Fortunately, the model does what we wanted all along. Here we can see the obviously changes of spike frequency on a constant input current with, or without noise.

3.3.2 adaptability, both ways?

One thing we should test is that, real neurons are apdative both ways, meaning, after the frequency has changed and the same input does not trigger an action potential this adaptability decays over time. So we expect our model to go back to its original behavior after a period of no or low input current.

Here we notice, after a period of time the intervals are back to normal and the firing frequency has reset due to the lack of input current which is the desired outcome.

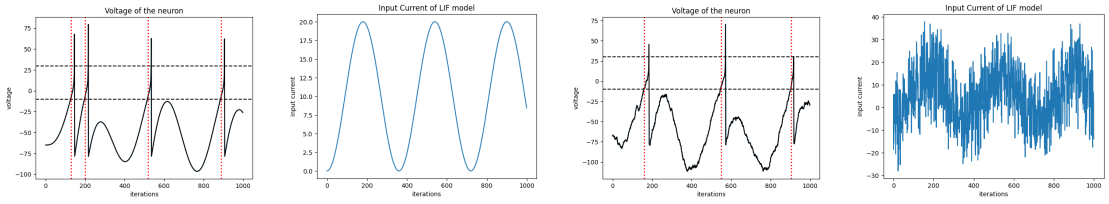


(a) *model potential per iteration* **(b)** *input current per iteration*

Figure 3.18: *AELIF and its adaptability*

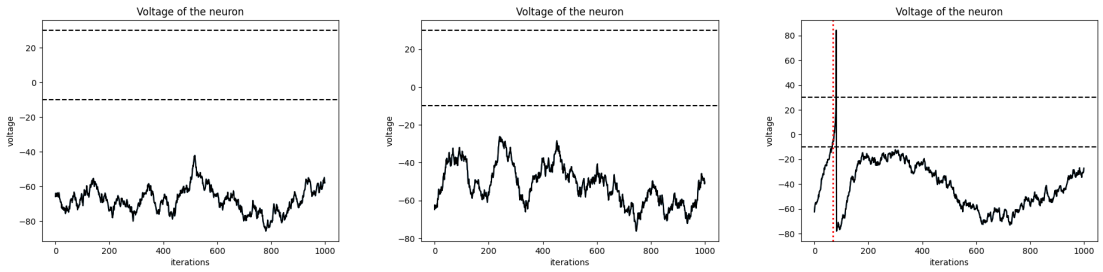
3.3.3 trigonometric and random function

In this section we do not expect any significant behavior of AELIF compared to the last two models except its adaptability which we discussed in the previous section so we combined the results of these input currents into a single section.



(a) *model potential per iteration* **(b)** *input current per iteration* **(c)** *input current per iteration* **(d)** *input current per iteration*

Figure 3.19: *AELIF on trigonometric function*



(a) *model potential with low current* **(b)** *model potential with mid current* **(c)** *model potential with high current*

Figure 3.20: *AELIF on random function*

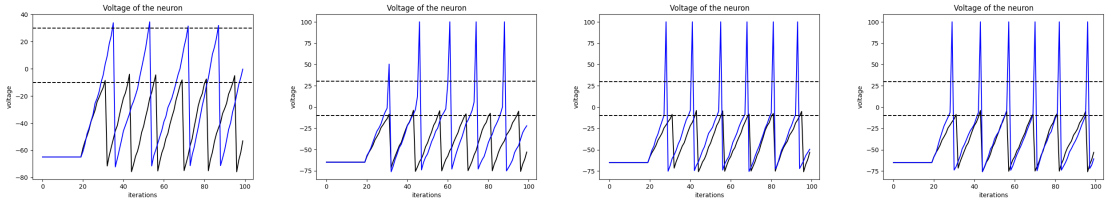
Here we discussed the behavior of each model on different input current with different levels of power and noise. In the section we are going to evaluate the model performance by different metric and analyse their results.

3.4 more on input current

Here we focus on other interesting behaviors of neurons when facing different input currents and comparing their behavior to one another to have a better understanding of their differences.

3.4.1 LIF vs ELIF, any difference?

We know from Chapter 2, the only basic difference in dynamics of LIF and ELIF is the **exponential term**, so we expect that as $\Delta t \rightarrow 0$, the more our ELIF gets close to basic LIF model.



(a) $\Delta_T = 10$ **(b)** $\Delta_T = 1$ **(c)** $\Delta_T = 0.01$ **(d)** $\Delta_T = 0.0001$

Figure 3.21: *LIF vs ELIF on $Delta_T$ term*

Here as expected we can see that, as $\Delta t \rightarrow 0$, the more the ELIF gets close to LIF, the only difference it the spike shooting which we implemented in ELIF but not LIF, if we get rid of that feature we can see that the results would be pretty much identical.

3.4.2 noise sensitivity

One important factor in neurons is their noise sensitivity, meaning how their behavior change when facing with an input current with, or without noise. In this part, we are going to compare the behaviors of LIF and ELIF on noisy data to see their behaviors.

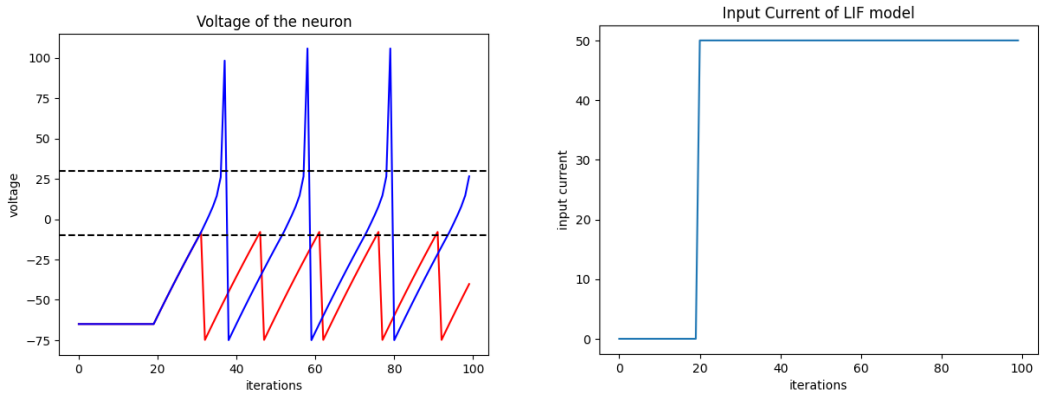


Figure 3.22: *LIF vs ELIF model spike pattern without noise*

Here we can see that the effects of noise are greater in LIF in comparison to ELIF, and so, we can say that LIF is more sensitive to noise than ELIF, but is there a theoretical

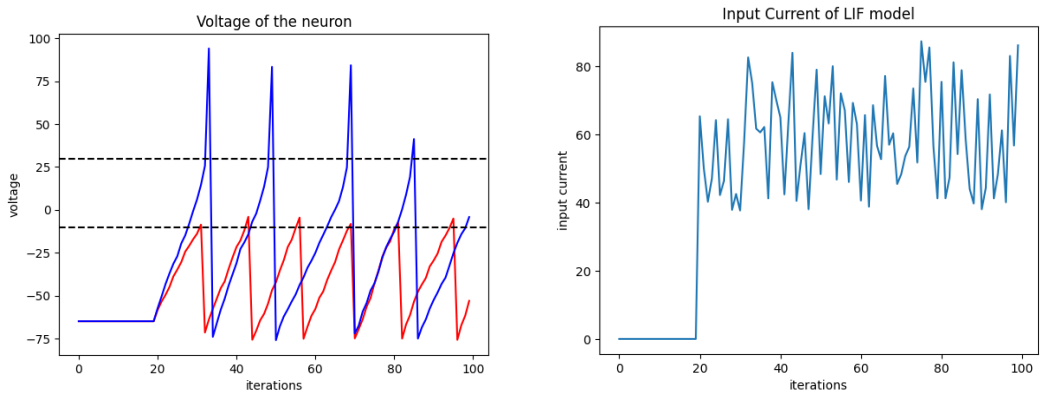


Figure 3.23: *LIF vs ELIF model spike pattern with noise*

hypothesis to back our experiment results?

In fact yes there is! In simple terms, generally, A dynamical system it less influenced by a parameter `X`, the more `X`-independant terms is has in its dynamics. So basically due to the fact that the only difference in dynamics of LIF and ELIF are in a single **exponential term** in ELIF which is in fact is independant of **input current** $I(t)$ is it logical to expect that ELIF is more stable when facing noise.

4

Model Analysis

In this section, we focus on ways to analyse our models even further. There are several ways to analyse a model's behavior or performance. because we do not have a test data, using performance-based analysis would not be feasible. Our main focus here, is on more correlation-based analysis like I-F and tau-F plots as well as correlation coeficient between input current and membrance potential of the neuron.

4.1 I-F curve

In neuroscience, a frequency-current curve (fI or F-I curve) is the function that relates the net synaptic current (I) flowing into a neuron to its firing rate (F) Because the f-I curve only specifies the firing rate rather than exact spike times, it is a concept suited to the **rate coding** rather than **temporal coding** model of neuronal computation. With this curve, we hope to achieve a basic understanding of how the input current rate effects the general spiking of pattern of our LIF models.

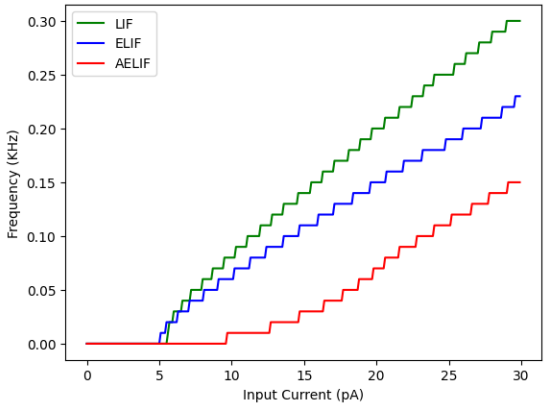


Figure 4.1: *I-F curve on LIF model*

Here, we can observe all of the frequencies of our models, one important aspect to observe is the fact that our Adaptive LIF frequency is lower than the other two models which is something to expect due to its adaptability. On the other hand, there are several

factors missing while only using F-I curve analysis, so we try to use other methods in order to get better understanding of different aspects of our models.

4.2 membrance potential vs. time coefficient

One important constant in our model is the its time coefficient τ_m. Here we try to see its effects on both membrance potential and spike frequency.

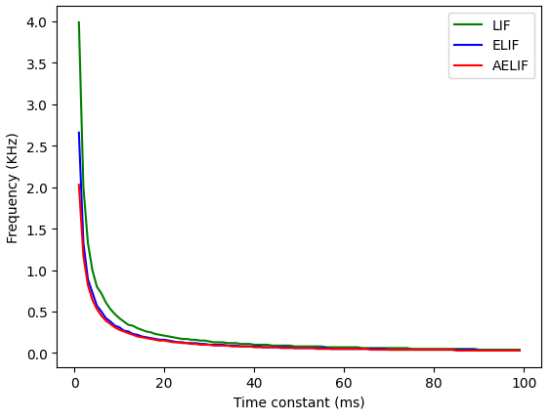


Figure 4.2: *τ-F curve on LIF model*

Here as expected, we can see its inverse effect on membrance potential and more specifically on its spiking frequency rate. The higher the time constant τ gets the lower the firing rate becomes and its aligned with our theoretical knowledge because we know that on simple terms the time constant controls how long and high an effect on input (whether outside current or from a pre-synaptic neuron) goes in the neurons own membrance potential, the lower it is, the higher the input effects the neuron's potential thus, triggering more spikes.

4.3 Model memory

Here we want to test which of the three models that we implemented has better memory with the same configuration and more importantly, **same time-constant**. In order to test this, we are going to enter a low amber input current not to fire a spike but only to change the membrance potential to our models in a short period of time and see how long the effects of them last in the membrance potential of each model.

We can see that in terms of memory, LIF and ELIF are arguable the same, however the adaptive LIF model works differently due to its *subthreshold adaptation variable* the effects does not remain the same as the other two but it gives a different dynamics which gives us the ability to have different firing patterns even when there are no input currents.

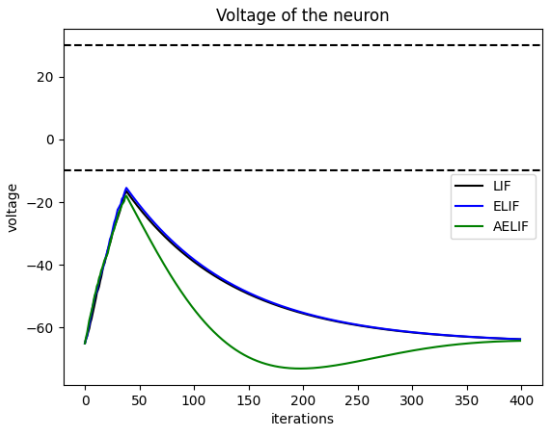


Figure 4.3: *memory of each model*

4.4 Other methods

There are other various aspects to consider like I-V plots and SNR (signal-to-noise ratio) which in some studies show that the correlation between inputs are somewhat effective in LIF performance, meaning it works generally better on inputs with low correlation comparing to other high-level neurons like HH model. on the other hand some other experiments can be performed regarding the variance of spikes and membrance potential comparing to a test data which is out of the scope of this report.

In the next section we implement yet another behavior on our simple LIF models to seal one of the problems we faced in previous analysis which was firing action potentials in after-spike intervals which can not be seen in biological neurons.

5

Refractory Period

Refractoriness is the fundamental property of any object of autowave nature (especially excitable medium) not responding to stimuli, if the object stays in the specific *refractory state*. In physiology, a refractory period is a period of time during which an organ or cell is incapable of repeating a particular action, or (more precisely) the amount of time it takes for an excitable membrane to be ready for a second stimulus once it returns to its resting state following an excitation.

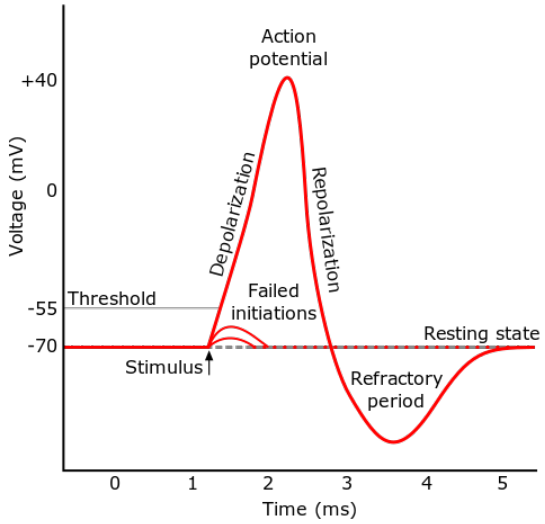


Figure 5.1: *action potential refractory period*

This concept helps neurons not to fire another spike after an action potential has been fired for a period of time called *refractory period*, however as you can see its dynamics is quite different from the dynamics of our previously implemented models, so it begs the question how can we implement this behavior into our models?

5.1 No Input Current!

The simplest way one can face this problem is to simply not let any input current into the model in the refractory period and simply let other factors of dynamics change its

behavior without the interference of outside input current. which gives us the following dynamics of our first refractory based spiking neuron, **Refractory AELIF model**.

Refractory AELIF model dynamics

if **not** in *refractory-period* :

$$\tau_m \frac{du}{dt} = -(u - u_{rest}) + \Delta_T exp(\frac{u - \theta_{rh}}{\Delta_T}) - Rw + \textbf{R.I(t)} \tag{5.1}$$

else :

$$\tau_m \frac{du}{dt} = -(u - u_{rest}) + \Delta_T exp(\frac{u - \theta_{rh}}{\Delta_T}) - Rw \tag{5.2}$$

$$\tau_w \frac{dw}{dt} = a(u - u_{rest}) - w + b\tau_w \sum_{t^f} \delta(t - t^f) \tag{5.3}$$

$$if\ u(t) = \theta_{rh} \rightarrow Fire\ Spike + Reset + \textbf{activate(refractory)}\ (u = u_{rest}) \tag{5.4}$$

With this dynamics we can add a few adjustments to our previous AELIF model to add Refractoriness to our model. the changes of implementation is as followed in both `initialize` and `forward` functions.

Listing 5.1: *changes in "initialize" function*

```
def initialize(self, neuron) :
    ...
    # Refractory parameters
    # ------------------------------------------------------------

    neuron.T = 0 # number of iterations for refractory period
    neuron.refractory_period = False
    neurons.refractory_iter = self.parameter("ref_iter", 20)
```

Listing 5.2: *changes in "forward" function*

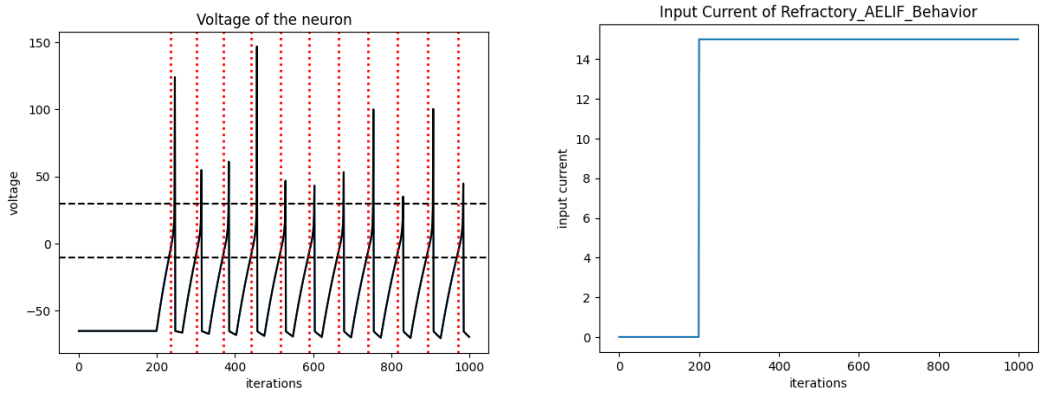
```
def forward(self, neuron) :
    ...
    if(reset_need) :
      neuron.T = neurons.refractory_iter
      neuron.refractory_period = True

      neuron.T = max(neuron.T - 1, 0)
      if(neuron.T == 0) :
        neuron.refractory_period = False

    dV = (
```

```
12      -(neuron.voltage - neuron.rest)
13      + neuron.delta*torch.exp(torch.tensor([(neuron.voltage - self.rh_threshold) / n
14      + neuron.R*neuron.I * (not neuron.refractory_period) # refractory_period
15      - neuron.R*neuron.w
16      ) / neuron.tau
```

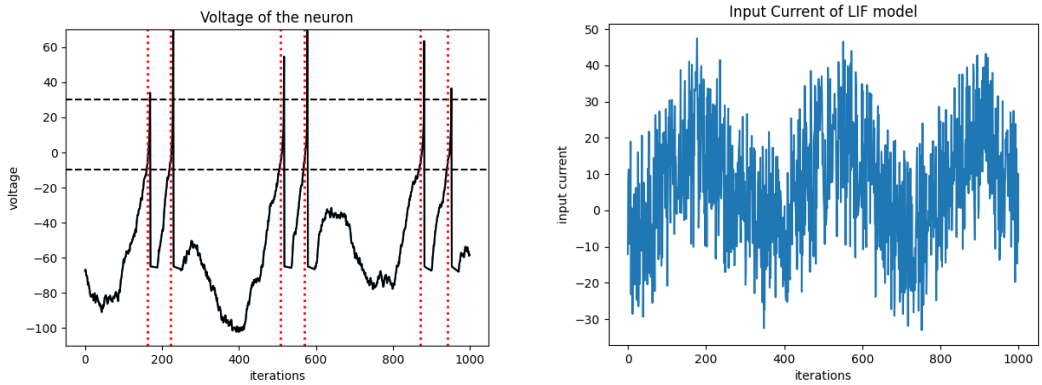
One even simpler way was to simply put changes of $u(t)$ to zero while in refractory period however with this implementation we allow some sort of **undershooting** in after-spike period which as we can see its a known phenomenon in a real neuron. Now we can test our model's behavior on input currents to see how it handles the refractory period.



(a) *Refractory AELIF potential per iteration* **(b)** *input current per iteration*

Figure 5.2: *Refractory AELIF on constant function with no noise*

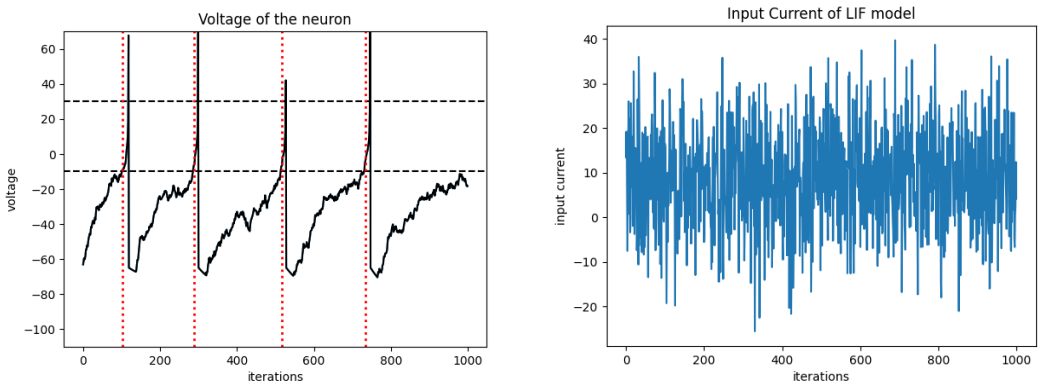
As we can see it works just fine with constant input current with no noise, however as we saw in Chapter 3, The problem of close spike firing arose when dealing with noisy input so this time test our model on more dynamic and noisy input current.



(a) *Refractory AELIF potential per iteration* **(b)** *input current per iteration*

Figure 5.3: *Refractory AELIF on trigonometric function with noise*

Fortunately, our model works just fine in both trigeometric current with noise and also just on random noise getting into the input.



(a) *Refractory AELIF potential per iteration* **(b)** *input current per iteration*

Figure 5.4: *Refractory AELIF on random noise*

5.2 spike shooting process

As we saw in the Chapter 2, we implemented the depolarization of an action potential with our **Exponential LIF model** using both `rh-threshold` and `threshold` in 2.2, with these and using the refreactory period we are hoping to get closer to what we expected from a real neuron to some extent using our generalized LIF models. which can be shown in the following figure.

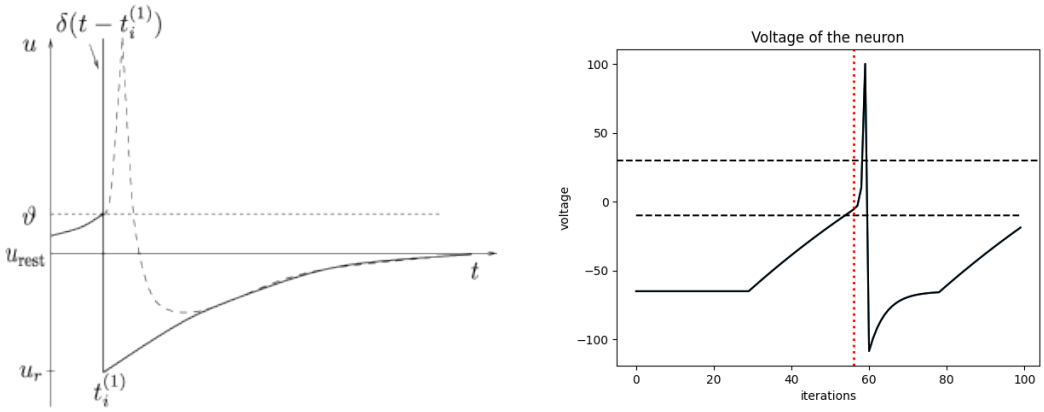


Figure 5.5: *spike shooting process*

Here we can see that our single neuron model works just fine and close enough to a real neuron activity which is what we hope to achieve. With this feature we can now expect a better accuracy in predicting spike patterns of a real neuron when facing with the same input current.

5.3 Adaptive Threshold

There is one more feature to add to our *Refractory AELIF* model, which is an adaptive threshold. Neurons not only show refractoriness after each spike but also exhibit

adaptation which builds up over hundreds of milliseconds. A simple leaky integrate-and-fire model does not perform well at predicting the exact spike times of a real neuron. However, if adaptation (and refractoriness) is added to the neuron model, the prediction works surprisingly well. The question we have to ask ourselves is "How to implement adaptive threshold?". A straightforward way to add adaptation is to make the firing threshold of the neuron model dynamic is as follows.

after each spike the threshold θ is increased by an amount ϵ, while during a quiescent period the threshold approaches its stationary value θ_0.

Adaptive threshold dynamics

$$\tau_{adapt} \frac{d}{dt}\theta(t) = -(\theta(t) - \theta_0) + \epsilon \sum_f \delta(t - t^{(f)}) \tag{5.5}$$

where τ_{adapt} is the time constant of adaptation (a few hunderd milliseconds) and $t^{(f)} = t^{(1)}, t^{(2)}, \ldots$ are the firing times of the neuron.

The implementation is nothing we have not seen, it has simply added a new dynamic to our model and calculating it for every iteration using euler method of approximation. Below is the changes in both `initialize` and `forward` functions to add the *adaptive threshold* to our refractory AELIF model.

Listing 5.3: *changes in "initialize" function*

```
def initialize(self, neuron) :
    ...
    # Adaptive Threshold
    # ------------------------------------------------------------

    neuron.tau_thresh_adpt = 100 # adaptive threshold time constant
    neuron.eps = 5 # epsilon changes of threshold when spiking
```

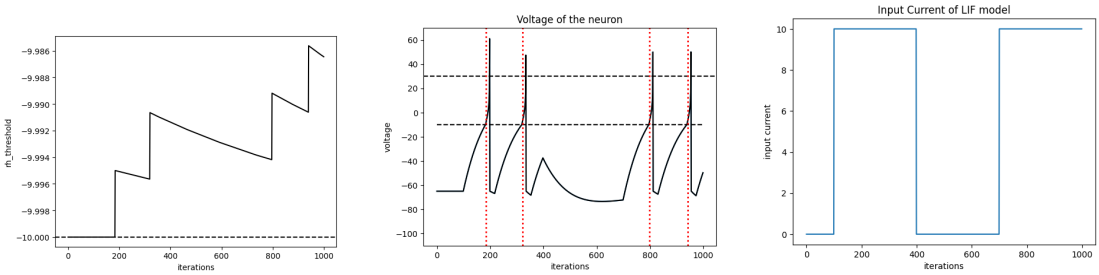
Listing 5.4: *changes in "forward" function*

```
def forward(self, neuron) :
    ...
    dTh = (
        - (neuron.rh_threshold - neuron.base_threshold)
        + neuron.eps * neuron.spike
    ) / neuron.tau_thresh_adpt

    neuron.rh_threshold += dTh * neuron.dt
```

Now lets see its effect on a step function input current without any noise for better understanding, keep in mind that the adaptive threshold works really slow in time

due to the fact that in real neurons this threshold changes over a long period of time comapring to other dynamics of a neuron so we run our model with `dt = 10` and for `iter = 1000` iterations in order to see its effects more vividly.



(a) θ_{th} *per iteration* **(b)** *voltage per iteration* **(c)** *input current per iteration*

Figure 5.6: *adaptive threshold RAELIF model*

As we can see, after each spike there is a subtle change in θ_{rh} threshold which in a short period of time may not have a huge impact but we can see its dynamics are closer to a real neuron than our previous models, also we can see that in the range where we cut out the input current the threshold is decaying back to its stationary state θ_0 which is what we expect.

So far, we have developed different variations of LIF model and have tested and analysed its behavior on different input currents with or without noise. Finally we have implemented our yet strongest model that has various features and dynamics of a real neuron. In order to have a better understanding of each and every parameter of our model and how their changes effect our membrance potential, we have to test out our model with different set of parameters and see their behaviors which is the topic of next and final chapter of this report.

6

In-depth analysis

In the final section of this report we try to have an in-depth analysis of our models and try to see the effects of each and every parameter of our models. As we developed more and more features into our model we have got various model to choose from, here in order to see the effects of each parameter on our model on an equal level for each of them we decided to use a single model for all of our experiments. The chosen model is the final one we implemented in Chapter 5, the **adaptive Refractory AELIF**, due to the fact that it has all the behaviors of our previous models combined, on the other hand its complexity might be an issue in seeing the effects of each parameter individually, however with increasing the rate of changes in each parameter we can sense the effects vividly.

First, in order to get a picture of what we are going to work with we have provided a list of the parameters we are going to analysis and a short description of what we expect them to do.

- **`tau`,** τ_m : main time constant of LIF
- **`R`** : the resistor in our RC-circut.
- **`delta`,** Δ_T : the sharpness parameter of exponential LIF
- **`a, b`** : subthreshold adaptation and spike-triggered adaptation parameters
- **`adapt-tau`,** τ_w : time constant of the adaptive variable w
- **`refractory-iter`** : number of iterations for a refractory period to last
- **`tau-thresh-adpt`,** τ_θ : time constant for adaptive threshold
- **`eps`,** ϵ : change rate of adaptive threshold for each spike

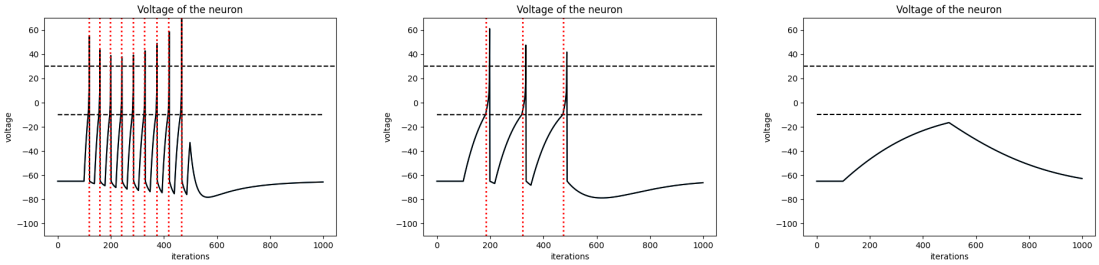
Now with this list at hand, we can start our experiments and see the whether the results of the experiments match our expectations.

6.1 LIF base parameters

Here are the main parameters of an LIF model which are a part of every variation of Intergrate-and-Fire models.

6.1.1 main time constant `tau`

As the nature of time constant `tau`, we expect it have inverse effect on membrane potential.



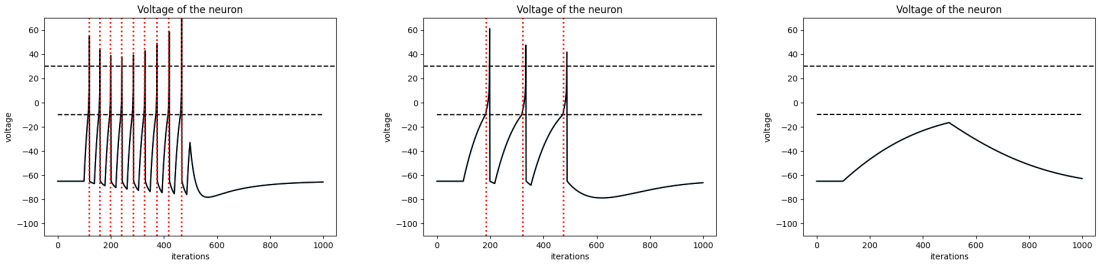
(a) $\tau_m = 2$ **(b)** $\tau_m = 10$ **(c)** $\tau_m = 50$

Figure 6.1: *effects of τ_m on membrane potential*

We can see what our theory works just fine in real life, as we increased `tau`, the effects of input effected the model in longer period of time and prevented the membrane potential to pile up, causing an action potential. So in order to have highly sensitive neurons their time constant τ_m must be low, on the other hand, if we are looking for a neuron to keep the effects for longer period of time but not be active as much having higher time constant might be the solution.

6.1.2 Resistor `R`

In practice it is simply a linear coeffient of input current entering the neuron, so its relation should be direct and linear to our membrance potential.



(a) $R = 2$ **(b)** $R = 10$ **(c)** $R = 50$

Figure 6.2: *effects of R on membrane potential*

Here we can see the direct effect of `R` on the membrance potential and causing the model to spike. For its linear effect on better way of visualization is to use R-F (Resistor-Frequency) plot.

We can notice that its effect is "somewhat" linear, but why it is not a straight line as we might expect? The reason lies in the fact that first, because not every little change on `R` would result in the change of number of spikes we have a sort of staircase behavior, on the other hand, due to models complexity and numerous non-linear dynamics we can not expect to see a complete linear behavior.

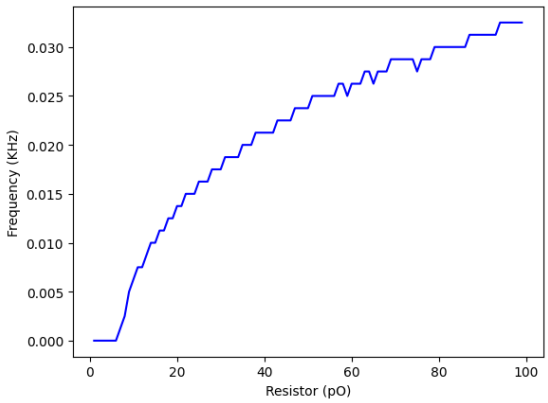


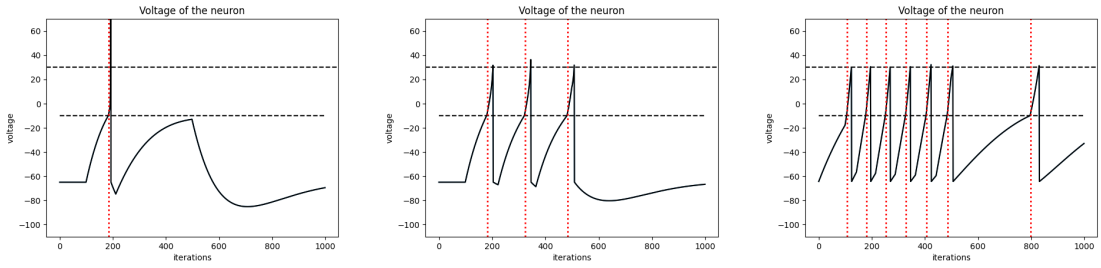
Figure 6.3: *R-F plot*

6.2 Exponential parameters

Now that we covered the basic parameters of an LIF model, here we focus on parameters of exponential part of nonlinear lif model (ELIF).

6.2.1 Sharpness parameter `delta`

The sharpness parameter works both as a coeffient of the exponential term and also a parameter in the exponential itself, working as a direct coeffient and also having inverse effect as the parameter in the exponential, this makes the analysis a bit harder. It is better to see the results on our model and analyse its behavior.



(a) $\Delta_T = 1.5$ **(b)** $\Delta_T = 10$ **(c)** $\Delta_T = 100$

Figure 6.4: *effects of Δ_T on membrane potential*

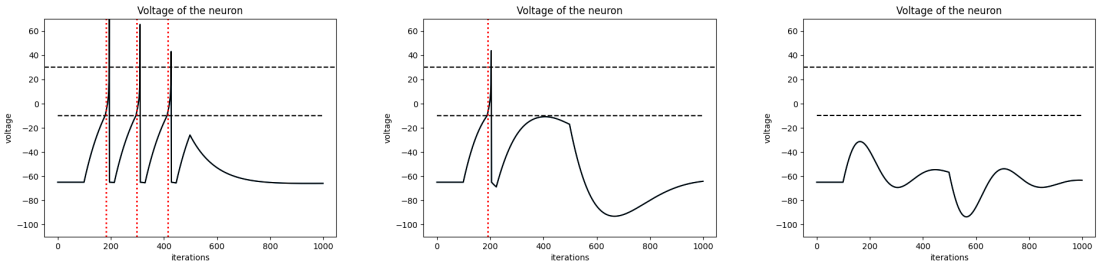
Here we can see that it has a direct effect on our membrance potential and spike rate, But in case $\Delta_T = 100$ we can see an abnormal behavior! Even though we cut of the input current after the $500th$ iterations, there is a spike even after this period, the reasoning behind it is that due to high value of `delta` in the last case, it has a huge effect in the dynamics of membrance potential $u(t)$ that can overcome the effects of decay term, making the membrance potential increase gradually and finally firing an action potential. So in order to have a behavior close to the common neuron we should avoid setting Δ_T too high, however this feature might be useful in simulating neurons which have spike patterns when they do not recieve an input.

6.3 Adaptive LIF paramters

Now it is time for the parameters envolving adaptive LIF models which change behavior when facing a spike and prevent bursting.

6.3.1 subthreshold adaptation variable `a`

The adaptation dynamics consists of three main parts, one of them is coeffient of decay term which is `a`.



(a) $a = 0.01$ **(b)** $a = 0.2$ **(c)** $a = 1.5$

Figure 6.5: *effects of subthreshold adaptation variable on membrane potential*

It is rather obvious that first of all it range of effectiveness is rather small in comparison to previous parameters. Also it can be seen that even as small as 1.5, it can completely overcome the effects of normal input current and prevent action potentials which is what we expect because the larger it becomes, the bigger the adaptation variable gets which results in a huge negative effect on our membrance potential. The neurons with high `a`, are really resistant to change of memebrance potential.

6.3.2 spike-triggered adaptation variable `b`

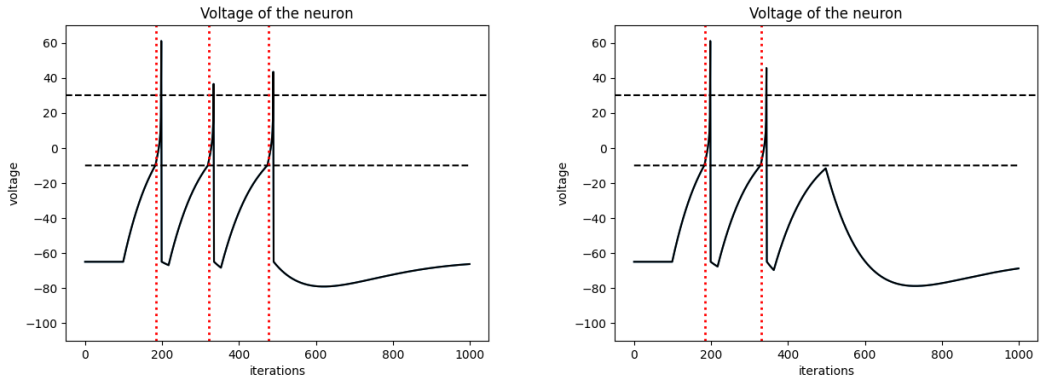
The spike-triggered adaptation variable, `b`, is responsible for the effects of dynamics of the adaptation variable once a spike has been fired, Its performance is highly related to the number of spikes the neuron makes.

Here we can see that it definitely has an inverse effect on membrance potential and a direct effect on the adaptation variable in Figure 6.6 but its effects are not as big as the previous parameters which is normal due to the fact that in a single simulation of a neuron the number of spikes will not reach a considerable number to see its effect more clearly.

6.3.3 adaptive time constant `adaptive-tau`

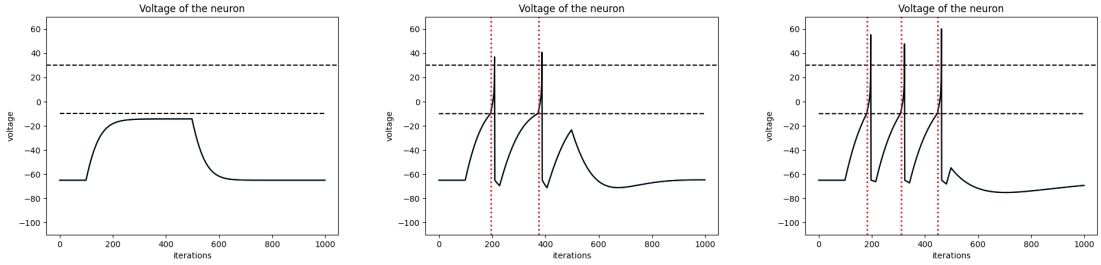
The time constant work as usual. This time however, it would have direct effect on membrance potential due to the fact that it directly controls the adaptation variable which has a negative effect on membrance potential.

The only thing worth mentioning is that it does not have a linear and more importantly steady effect on our membrance potential, meaning, because it only effects the adaptation



(a) $b = 1$ *with* $I.pw = 10$ **(b)** $a = 100$ *with* $I.pw = 10$

Figure 6.6: *effects of spike-triggered adaptation variable on membrane potential*



(a) $\tau_{adpt} = 2$ **(b)** $\tau_{adpt} = 10$ **(c)** $\tau_{adpt} = 50$

Figure 6.7: *effects of* τ_{adpt} *on membrane potential*

variable and it decays back to its original value, after a threshold its changes would not have an effect on our membrance potential and its effect rate is rapidly decreasing.

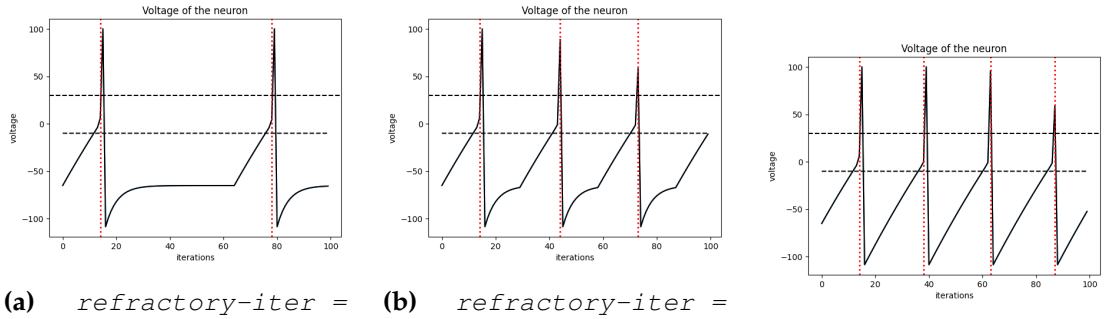
6.4 Refractory parameters

The refractory parameters are the ones controling the spike interval and prevent spikes from firing in a short period of time, the basic expectation is to learn how to control the changes of refractory interval and how to control the chnages of threshold.

6.4.1 refractory period `refractory-iter`

The main parameter of refractory LIF model is `refractory-iter`. It controls the interval in each the refractory period is active and we ought to keep it in a short interval between 10 to 20 iterations, of course the number of iterarions heavily depends on out time resolution `dt`, but it is safe to say that with our configuration, 10 to 20 iteration would work just fine.

As we can see in Figure 6.8, the middle figure with `refractory-iter = 15` looks the best comparing to the other two and particularly, with `refractory-iter = 2`, we can see that the refractory feature has close to no effect on our model.

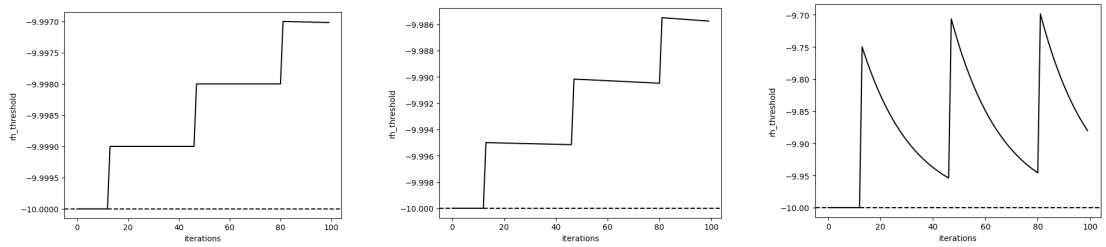


(a) `refractory-iter = 50` **(b)** `refractory-iter = 15` **(c)** `refractory-iter = 2`

Figure 6.8: *effects of* `refractory-iter` *on membrane potential*

6.4.2 adaptive threshold time constant `tau-thresh-adpt` and `eps`

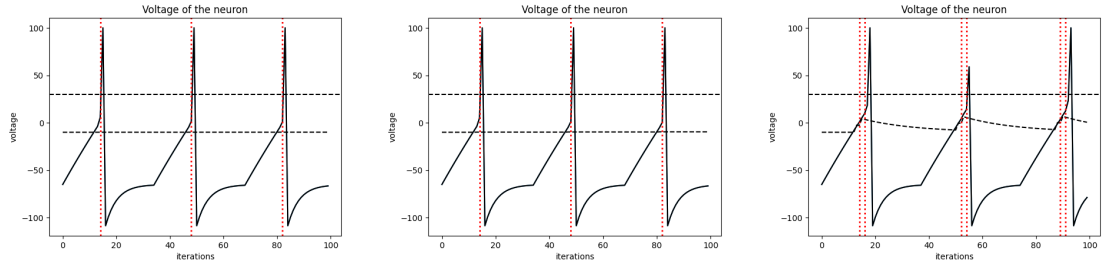
The same as all other time constants, we expect it to control how fast the effects are applied to the model and how long its effects remain but we should keep in mind that this parameter is related to **adaptive threshold**, so because in biological neurons the threshold rarely changes, we have to keep in mind that the normal value of this time constant should be relatively high, around few hundred milliseconds.



(a) $\tau_{adpt} = 500$ **(b)** $\tau_{adpt} = 100$ **(c)** $\tau_{adpt} = 2$

Figure 6.9: *effects of* $\tau_{adpt} = 500$ *on threshold*

As we can see, the lower the τ_{adpt} gets, the higher the changes and the lower the "memory" of our adaptive threshold would be. In order to see the effects of different values of τ_{adpt} more clearly on membrance potential and firiing pattern we increase `eps` which controls the rate of changes in adaptive threshold after each spike to have a better understanding of its effects.



(a) $\tau_{adpt} = 500$ **(b)** $\tau_{adpt} = 100$ **(c)** $\tau_{adpt} = 2$

Figure 6.10: *effects of* $\tau_{adpt} = 500$ *on spike pattern*

Here we can see that specially low τ_{adpt}, would result in a very strange and abnormal behavior for neuron's threshold which is clearly, not what we are looking for, so it is better to keep τ_{adpt} around few hundreds as biological data suggests.

7

Conclusion

In this report, we implemented one of the most simple, yet important single spiking neuron models and its variations, LIF (Leaky Intergrate-and-Fire). We added features and made it more complex step by step. Next, we analysed its behavior on different input current and also we went in-depth to understand each and every parameter's effect on our model. Now that we have a better understanding of how a single neuron work and how we can simulate its behavior to some extent, we now are ready to dive into more complex behaviors of nervous system. Our next step would be to simulate the connection between neurons (the synapses) and explore the relation between these single neurons as a whole, getting closer to having a neural population.

This page intentionally left blank.

UNIVERITY OF TEHRAN

SINGLE SPIKING NEURON MODEL AND ANALYSIS

ARASH NIKZAD
Student No. 610301195

TEHRAN, MARCH 2024