Foundations of Computational Neuroscience: A Brief Analysis of Different Neuron Models (Hodgkin & Huxley, Leaky Integrate-and-Fire Model, Poisson Neuron Model, FitzHugh-Nagumo Model)

Daniel A. Reyna, BSc MSc
Nezih Nieto, BSc (c)

Abstract - Neuron Models are depictions of the biophysics of the electrophysiological activity in real neural cells. Different models are naturally of use to describe approximations beneath the very desired constraints of the modelled systems. In Systems Neuroscience it is widely accepted that neurons are dynamic in time; therefore, models were created thinking in how the neural activity would behave. Hodgkin & Huxley (H&H), Leaky Integrate-and-Fire (LIF), and Poisson models are some of the more often used depictions for real neurons depending on the scope that a research line, study or lecture requires.

Keywords: H&H Model, LIF Model, Poisson Neuron Model, Systems Neuroscience, Neural Dynamics, FHN Model.

Introduction

Neuron Models are rather useful every time neurophysiological phenomena must be analysed. Such models describe how action potentials are initiated and transmitted onto neurons. They typically consist of a set of four nonlinear ordinary differential equations, which approximate the electrical characteristics of excitable cells such as neurons or innervations throughout the nervous system.

In this paper four of the most famous models are shown and briefly summarised so they can be understood properly with a global insight. The models mentioned in this work are the Leaky tegrate-and-Fire (LIF) Model, the Poisson Model, the Hudgkin and Huxley Model, and the FitzHugh-Nagumo Model.

Thus, the main objective of this work is to determine the importance of knowing the differences between them, so they can be used for specified purposes.

LIF Model

When discussing the LIF model, it is important to understand that its very behaviour aims to model the required energy to pass a threshold, which is originated by an amount of graded potentials, that after being temporally added, they fire the neuron action potential, which means that the membrane was fully charged, so it can fire a pulse that will allow in a presynaptic cell to reach through an axon the next neuron or postsynaptic cell, that also will be charged and behave as a presynaptic cell for the next synapsis via the dendrites of the neural cell.

The current that is thus injected in the membrane could be an addition of impulses, so it would be easy to observe that after receiving several impulses from different synapses, the spike will generate the action potential, that involves a temporal analysis of energy and matter balance between the membran external and internal ionic concentrations. Hence the name is 'integrated', since every received spike is integrated into the response of the post-synaptic cells.

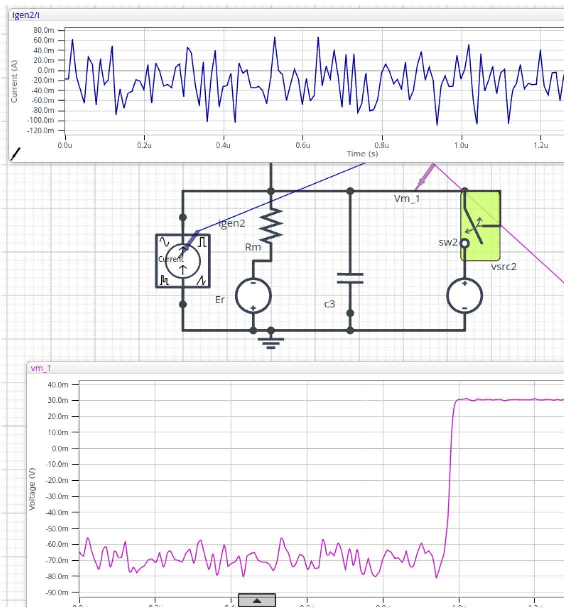

Figure 1. LIF model representation, and measured depolarisation in response to a noise input.

The circuit in *Figure 1* where the capacitor of *20nF* (Cm) models the charges inside the cell, the resistor of *200MΩ* (Rm) models the inverse value of conductance of the ion channels, the value of the electromotive force in equilibrium or rest conditions (Er) of the neuron potential, with value of *-65mV*, finally the injected current is modelled with the current source (Iinj) at the second circuit. Thus the measure *Vm* responds to the membrane potential. The firing is obtained after the voltage-dependent switch that exists normally open closes after the threshold voltage is reached.

The previous analysis can be also deduced as a matter of fact from a series of equations describing the elements in an electrical circuit. So it is important for the model to accomplish feasible representations, such equations are shown as follows:

$$\frac{dV_m}{dt} = aV_m + \frac{1}{C_m} I_{inj}(t)$$

Be *a* the negative inverse of And it must be integrated after changing the form of the equation into:

$$e^{-at}\left[\frac{dV_m}{dt} - aV_m\right] = e^{-at}\left[\frac{1}{C_m} I_{inj}(t)\right]$$

$$\int_{t_0}^{t} e^{-at}\left[\frac{dV_m}{dt} - aV_m\right]dt = \int_{t_0}^{t} e^{-at}\left[\frac{1}{C_m} I_{inj}(t)\right]dt$$

Finally, the integration is solved and the time constant that shows how fast the membrane charges:

$$V_m(t) = R_m I_{inj}(t)\,\left(1 - e^{-\frac{t-t_0}{\tau_m}}\right) - V(t_0)\, e^{-\frac{t-t_0}{\tau_m}}$$

Where the Injected Current is a sum of the noise and a defined current function, such as:

$$V_m(t) = R_m[I_s(t) + I_{noise}(t)]\,\left(1 - e^{-\frac{t-t_0}{\tau_m}}\right) - V(t_0)\, e^{-\frac{t-t_0}{\tau_m}}$$

Such an equation is nevertheless incomplete, and to understand the firing for the model, a restart charge must be introduced (observable as the voltage-dependent switch). So the total Injected Current shall *leak* some input amount over time. As it is shown in the following graphs that are a result of modelling via python (**Appendix A**).

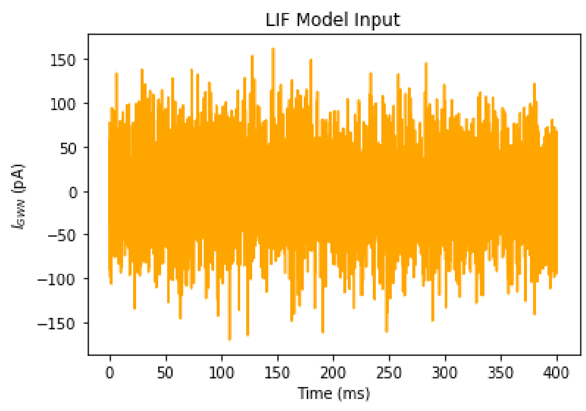

Figure 2. Noise and Active Injected Current Input.

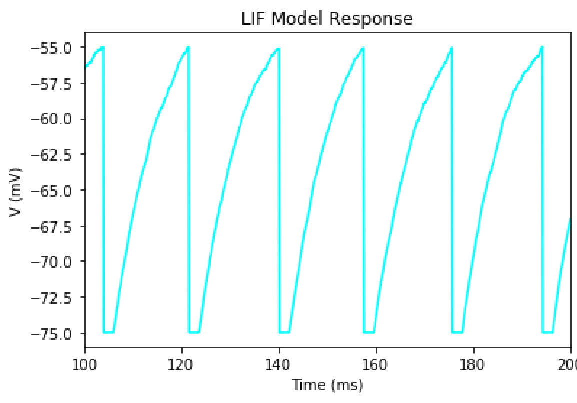

Figure 3. LIF Response to Current Input. That are characterised after an integration of the input noise, reordered in time, and finally becoming outputs after reaching a treshold.

After having observed the two representations of the model, it is evident that the

LIF model will behave periodically by integrating inputs in ordered spikes over time, firing spikes after reaching the threshold. Such phenomena attempts to model the so-called action potentials, that are a consequence of arithmetically sumed .

Hodgkin & Huxley Model

In the Hodgkin & Huxley (HH) model, the conducting components are a function of the potential (V_m) across the cell membrane and the equilibrium potential (E) of the ions. The equilibrium potential can be derived from the Nernst-Einstein relation.

$$\varphi = \frac{RT}{F^2\rho}$$

Where φ is the rate of ions that travel across the membrane, that depends on the value of the membrane resistivity ρ. The currents through the conducting elements can be expressed as:

$$I_{ion} = g_{ion}(V_m - E_{ion})$$

Hodgkin and Huxley's results showed that g_{Na} and g_K are functions of time as well as voltage, but that the conductance of other ions are constant. Depolarisation of the axon membrane causes a transient increase in sodium (Na) conductance and a smaller non-inactive increase in potassium (K) conductance. The time dependence of this conductance can be represented by an activation coefficient x which represents the probability that a gate in the channel is open (if the conductance is considered to be represented by the opening of many individual channels). The conductance for a time-dependent channel must be written in terms of its activation coefficient x ($0 \leq x \leq 1$) and a maximum conductance.

In the *HH* model the circuit will have thus variating circuits for every current that the membrane supports.

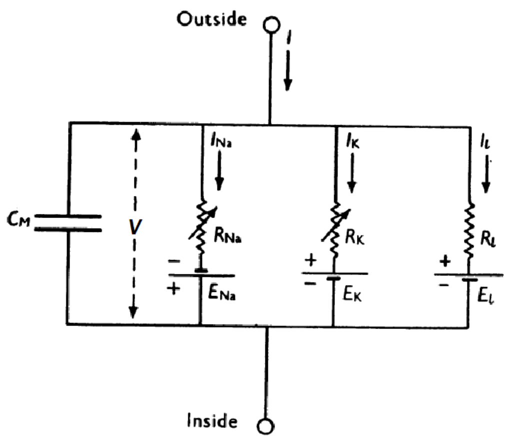

Figure 4. Hodgkin & Huxley Model depicted as an electrical circuit.

As a simple way to observe the spikes, and to exemplify how this model works a program in python (**Appendix B**) showed the figures below:

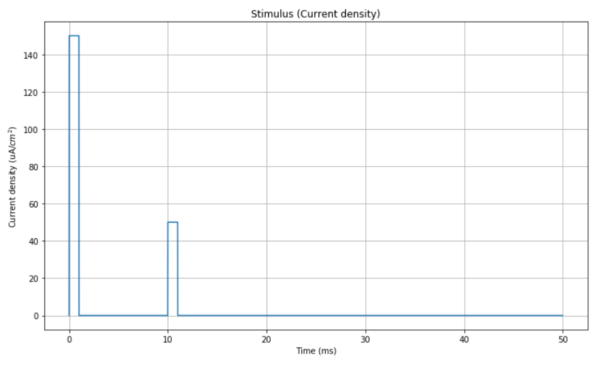

Figure 5. HH model input.

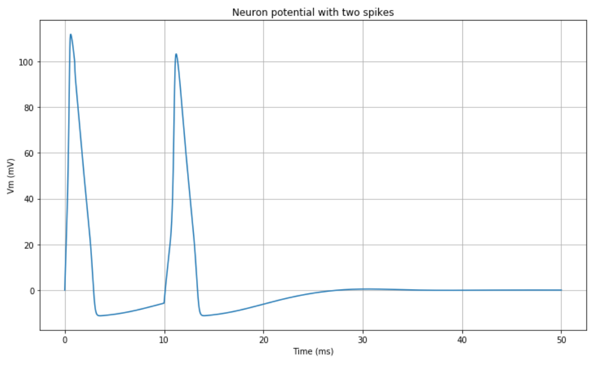

Figure 6. HH model response to the previous input, they are characterised by single pulses in the input that depolarises the cell, then the currents change de direction to repolarise the membrane, finally the action potential looks for equilibrium and resting state potential.

Even though the HH model can be approximated as variated conductances in for each ion channels set, it would be more accurate to understand it as a set of transistors or operational amplifiers arrange, since it is known that the system will show an hysteretic response, also understood as the memory of the system that models different curves for each value that will depend on the previous values of the membrane potential. This phenomenon is similar to the noise integration in the LIF model, but it is also more similar to the real behaviour of a neuron. The derivation of a model is shown in *Figure 7;* hysteresis of the system is observed in Figure *8.*

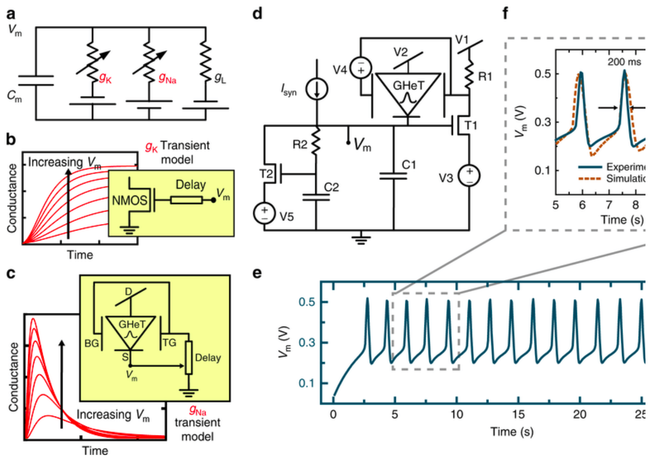

Figure 7. Obtaining more accurate models for Hodgkin and Huxley theoretical propositions.

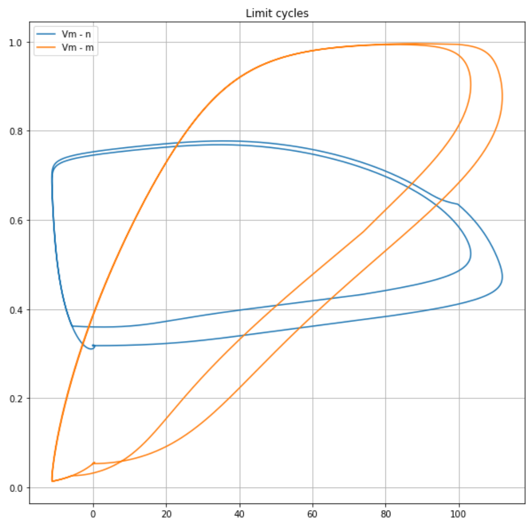

Figure 8. Membrane Potential and Hysteresis in the Hodgkin & Huxley model.

That is to say, that our system will be rule by the conservation of energy principle, and so are the HH Model Equations that are to be reviewed as follows:

$$C\frac{dV_m}{dt} = -g_L(V_m - E_L) - g_{K^+}n^4(V_m - E_{K^+}) - g_{Na^+}m^3h(V_m - E_{Na^+})$$

Describes the differential equation obtained from the Nernst Equilibrium Equation. Where the $g_{L,K,Na}$ are the respective conductances for each ion channel, $E_{L,K,Na}$ are the reversal potentials, and *n, m,* and *h* are values that consider the several number of ions in the membrane system dynamics that the HH model describes.

$$\frac{dn}{dt} = \alpha_n(n - 1) + \beta_n n$$

For potassium ions.

$$\frac{dm}{dt} = \alpha_m(m - 1) + \beta_m m$$

$$\frac{dh}{dt} = \alpha_h(h - 1) + \beta_h h$$

For sodium ions.

Poisson Neuron Model

The Poisson Neuron Model provides an instantaneous firing rate; i.e. the instantaneous probability of firing at any instant, and the output is a stochastic function of the input. Unlike the LIF Model, the Poisson Model does not use the noise per se, yet it takes every input as a part of the probability function. In part because of its simplicity, the model is widely used especially in *in vivo* single unit electrophysiological studies.

In the figures below, a distribution of inputs is generated by using a python code (**Appendix C**):

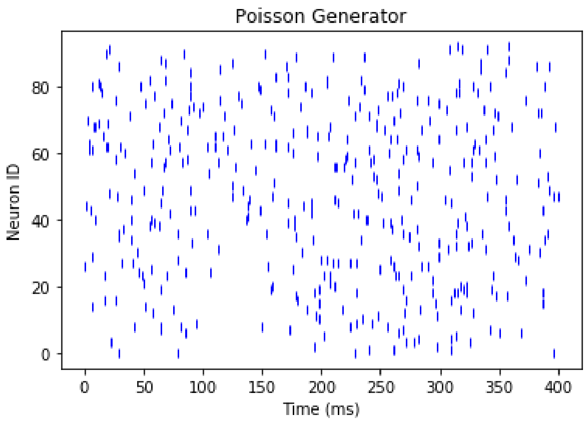

Figure 9. Poisson Distribution Generated via Python Script.

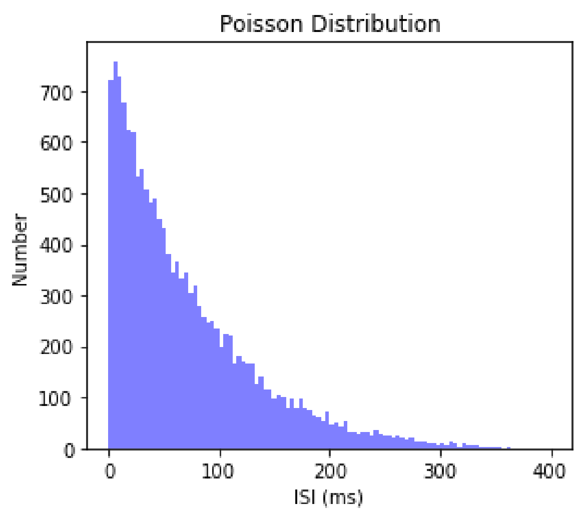

Figure 10. Poisson Distribution ordered over time.

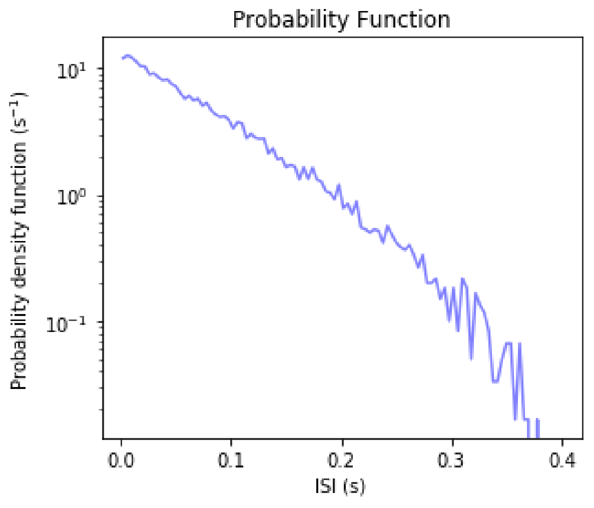

Figure 11. Poisson Distribution Probability Density (the integral of the distribution).

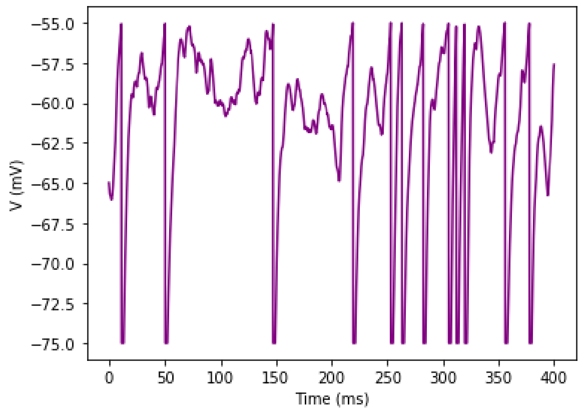

Figure 12. Integrate-and-fire response to the Poisson Distribution spikes.

The model therefore is supposed to be a better depiction of the actual behaviour of the neuron, since it considers a response that mitigates the noise in a distribution as an input, unlike the LIF model. Nonetheless, this model is quite inefficient whether it is required for response prediction.

Comparing the models

Some of the criteria to decide which model would be better to be used would depend on how they present the outputs, on the difficulty to be modelled, on the accuracy respect to the neuron behaviour, on how the models can be used for dimensional analysis, and on how much information they provide.

Criteria	LIF	Poisson	HH
Determinism	Deterministic	Stochastic	Deterministic
Conceptualisation	Affordable	Complex	Reducible
Accuracy	Better for controlled systems	Better for single units	Better at every scale
Adaptability	Flexible	Robust	Flexible because of reduced models
Spatial Analysis	Limited	Scalable	Scalable
Considerations	Given by the model	Assumed by Distributed Input and the model	Broader, includes dynamics of ion channels in complete version

Table 1. Advantages and Disadvantages of each model. In green is shown the best option for the criteria considered in the table above from a computational implementation scope.

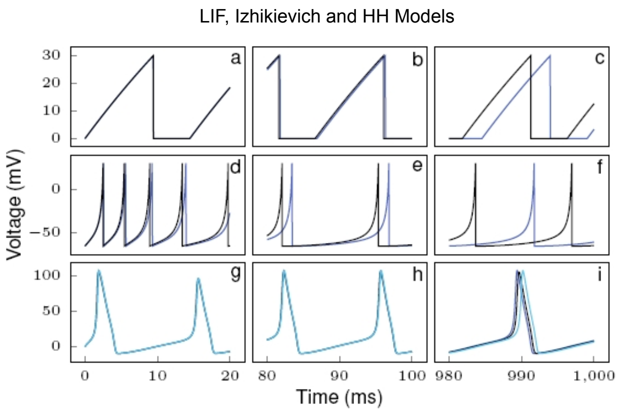

Figure 13. Comparison of LIF models and HH models. The first one is a noise integrating LIF (no Poisson), the second a HH reduced model (Izhikievich Model), and the last one is the Hudgkin and Huxley Model.

Reduced Models: FitzHugh-Nagumo Model

As a consequence of the complexity of the HH model, some simplifications or didactic models have been developed for a better understanding of the work of Alan Lloyd Hodgkin and Andrew Huxley. That is how FitzHugh-Nagumo was invented by using a Bonhoeffer–Van der Pol oscillator.

This model does not provide a very accurate description of the biophysical reality of nerve cells, but rather provides a mathematical insight into the mechanism of neuronal excitability. To interpret the dynamics of the FHN system in biophysical terms, the variable v is translated as the voltage across the membrane, the parameter I_{ext} represents the current applied to the nerve cell, and the variable ω as a system recovery variable with no specific biophysical meaning that includes a coefficient a that include the value of the time constant of the system.

$$\nu' = g(\nu) - \omega + RI_{ext}$$

$$\omega' = \nu - a\,\omega$$

This reduced model can be also plotted in a python script, in this case using the MIT spine dependency (**Appendix D**):

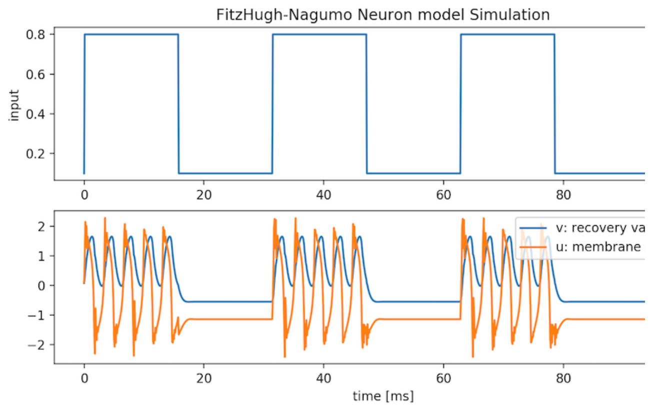

Figure 14. FitzHugh-Nagumo model with a response to a digital pulse train input.

As it can be seen, there is no such a thing as the best model, but there are for sure variations of the models so they can approximate a certain behaviour, considering different conditions and aiming to fit specific requirements.

Conclusions

Among the different models exposed in this paper, it is easy to conclude that every model has its own characteristics and satisfies certain necessities that the implementation may have. Moreover, the understanding of every model helps the reader to identify when and for what purpose a model should be used.

Either it be to present reliable models so they can be used for research purposes or to use a simple representation as a didactic technique, they all involve the understanding of the neurobiology of systems neuroscience, and how dynamics of the human brain shall be considered and not taken for granted as simple as it seem without detailed observations.

References

[1] Ardila U, William, Avendano, Luis Enrique, Orozco G., Alvaro A. Modelo de hodkin y huxley. Scientia et técnica. Año 06, No. 13.

[2] Ardila U., Willliam, Lopez A., Carlos Alberto, Orozco G., Alvaro Angel. Modelo de la membrana nerviosa y simulación de Fitzhugh-Nagumo. Scientia et técnica. Año 06, No. 14.

[3] Hodgkin, A. L. y Huxley, A. F. (1952d), A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiology, 117 (4), 500-544; Aug.

[4] Katchalsky, A., and P. F. CURRAN. 1965. Nonequilibrium Thermodynamics in Biophysics. Harvard University Press, Cambridge, Mass. 133.

[5] Beck, Megan & Shylendra, Ahish & Sangwan, Vinod & Guo, Silu & Rojas, William & Yoo, Hocheon & Bergeron, Hadallia & Su, Katherine & Trivedi, Amit & Hersam, Mark. (2020). Spiking neurons from tunable Gaussian heterojunction transistors. Nature Communications. 10.1038/s41467-020-15378-7.

[6] Kostova T.; Ravindran R.; Schonbek, M. FitzHugh-Nagumo Revisited:
Types of Bifurcations, Periodical Forcing and Stability Regions by a Lyapunov Functional, University of California. International Journal of Bifurcation and Chaos. pp. 4-19. 2003.

[7] Rocsoreanu, C.; Georgescu, A.; Giurgiteanu, N. The FitzHugh-Nagumo
Model, Kluwer Academic Publishers, Netherlands. ISBN 0-7923-6427-9.
2000.

[8] Koyama, S., & Kass, R. E. (2008). Spike train probability models for stimulus-driven leaky integrate-and-fire neurons. Neural computation, 20(7), 1776–1795. https://doi.org/10.1162/neco.2008.06-07-540

[9] Softky, W. and Koch, C. (1993). The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps. J. Neuroscience . , 13:334-350.

[10] C Allen and C F Stevens. An evaluation of causes for unreliability of synaptic transmission. Proc. Natl. Acad. Sci., 91:10380–10383, 1994

[11] Lapicque, L. (1907). Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. J. Physiol. Pathol. Gen., 9:620-635.

[12] Stein, R. B. (1965). A theoretical analysis of neuronal variability. Biophys. J., 5:173-194.

[13] Ermentrout, G. B. (1996). Type I membranes, phase resetting curves, and synchrony. Neural Computation, 8(5):979-1001.

[14] Fourcaud-Trocme, N., Hansel, D., van Vreeswijk, C., and Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating input. J. Neuroscience, 23:11628-11640.

[15] Badel, L., Lefort, S., Berger, T., Petersen, C., Gerstner, W., and Richardson, M. (2008). Biological Cybernetics, 99(4-5):361-370.

[16] Latham, P. E., Richmond, B., Nelson, P., and Nirenberg, S. (2000). Intrinsic dynamics in neuronal networks. I. Theory. J. Neurophysiology, 83:808-827.

[17] F Rieke, D Warland, R de Ruyter van Steveninck, and W Bialek. Spikes: Exploring the Neural Code. MIT Press, Cambridge, MA, 199

Appendix A
(https://colab.research.google.com/github/johanjan/MOOC-HPFEM-source/blob/master/LIF_ei_balance_irregularity.ipynb#scrollTo=oMqaksMrfl1z)

```
import matplotlib.pyplot as plt
# import matplotlib
import numpy as np
# import numpy
import time
# import time
import ipywidgets as widgets
# interactive display
from scipy.stats import pearsonr
# import pearson correlation

fig_w, fig_h = (6, 4)
plt.rcParams.update({'figure.figsiz
e': (fig_w, fig_h)})

def default_pars( **kwargs):
   pars = {}

   ### typical neuron parameters###
   pars['V_th']    = -55. # spike
threshold [mV]
   pars['V_reset'] = -75. #reset
potential [mV]
   pars['tau_m']   = 10. # membrane
time constant [ms]
   pars['g_L']     = 10. #leak
conductance [nS]
   pars['V_init']  = -65. # initial
potential [mV]
   pars['V_L']     = -75. #leak
reversal potential [mV]
   pars['tref']    = 2. #
refractory time (ms)

   ### simulation parameters ###
   pars['T'] = 400. # Total
duration of simulation [ms]
   pars['dt'] = .1  # Simulation
time step [ms]

   ### external parameters if any
###
   for k in kwargs:
       pars[k] = kwargs[k]

   pars['range_t'] = np.arange(0,
pars['T'], pars['dt']) # Vector of
discretized time points [ms]

   return pars
def run_LIF(pars, I):
   '''
   Simulate the LIF dynamics with
external input current

   Expects:
   pars       : parameter
dictionary
   I          : input current [pA].
The injected current here can be a
value or an array

   Returns:
   rec_spikes : spike times
   rec_v      : mebrane potential
   '''

   # Set parameters
   V_th, V_reset = pars['V_th'],
pars['V_reset']
   tau_m, g_L = pars['tau_m'],
pars['g_L']
   V_init, V_L = pars['V_init'],
pars['V_L']
   dt, range_t = pars['dt'],
pars['range_t']
   Lt = range_t.size
   tref = pars['tref']
   # Initialize voltage and current
   v = np.zeros(Lt)
   v[0] = V_init
   I = I * np.ones(Lt)
   tr = 0.
   # simulate the LIF dynamics
   rec_spikes = []                #
record spike times
   for it in range(Lt-1):
     if tr >0:
           v[it] = V_reset
           tr = tr-1
     elif v[it] >= V_th:
#reset voltage and record spike
event
           rec_spikes.append(it)
           v[it] = V_reset
           tr = tref/dt
     #calculate the increment of
the membrane potential
     dv = (-(v[it]-V_L) +
I[it]/g_L) * (dt/tau_m)

     #update the membrane potential
     v[it+1] = v[it] + dv

   rec_spikes =
np.array(rec_spikes) * dt

   return v, rec_spikes

def my_GWN(pars, sig,
myseed=False):
   '''
   Expects:
   pars        : parameter
dictionary
   sig         : noise amplitute
   myseed      : random seed. int or
boolean

   Returns:
   I           : Gaussian white
noise input
   '''

   # Retrieve simulation parameters
   dt, range_t = pars['dt'],
pars['range_t']
   Lt = range_t.size

```

```
   # set random seed
   if myseed:
       np.random.seed(seed=myseed)
   else:
       np.random.seed()

   #generate GWN
   I =  sig * np.random.randn(Lt)  /
np.sqrt(dt/1000.)

   return I
pars = default_pars()
sig_ou = .5
I_GWN = my_GWN(pars, sig=sig_ou,
myseed=1998)
plt.title('LIF Model Input')
plt.plot(pars['range_t'], I_GWN,
'b', color='orange')
plt.xlabel('Time (ms)')
plt.ylabel(r'$I_{GWN}$ (pA)');

v, rec_spikes = run_LIF(pars, I
=I_GWN+250.)
plt.title('LIF Model Response')
plt.plot(pars['range_t'], v, 'b',
color="cyan")
plt.xlabel('Time (ms)')
plt.ylabel('V (mV)');
plt.xlim(100,200)
```

Appendix B
(https://gist.github.com/giuseppebonaccorso/60ce3eb3a829b94abf64ab2b7a56aaef)

```
import matplotlib.pyplot as plt
import numpy as np

from scipy.integrate import odeint

# Set random seed (for
reproducibility)
np.random.seed(1000)

# Start and end time (in
milliseconds)
tmin = 0.0
tmax = 50.0

# Average potassium channel
conductance per unit area (mS/cm^2)
gK = 36.0

# Average sodoum channel
conductance per unit area (mS/cm^2)
gNa = 120.0

# Average leak channel conductance
per unit area (mS/cm^2)
gL = 0.3

# Membrane capacitance per unit
area (uF/cm^2)
Cm = 1.0

# Potassium potential (mV)
VK = -12.0

# Sodium potential (mV)
VNa = 115.0

# Leak potential (mV)
Vl = 10.613

# Time values
T = np.linspace(tmin, tmax, 10000)
```

```
# Potassium ion-channel rate
functions

def alpha_n(Vm):
    return (0.01 * (10.0 - Vm)) /
(np.exp(1.0 - (0.1 * Vm)) - 1.0)

def beta_n(Vm):
    return 0.125 * np.exp(-Vm /
80.0)

# Sodium ion-channel rate functions

def alpha_m(Vm):
    return (0.1 * (25.0 - Vm)) /
(np.exp(2.5 - (0.1 * Vm)) - 1.0)

def beta_m(Vm):
    return 4.0 * np.exp(-Vm / 18.0)

def alpha_h(Vm):
    return 0.07 * np.exp(-Vm / 20.0)

def beta_h(Vm):
    return 1.0 / (np.exp(3.0 - (0.1
* Vm)) + 1.0)
 # n, m, and h steady-state values

def n_inf(Vm=0.0):
    return alpha_n(Vm) /
(alpha_n(Vm) + beta_n(Vm))

def m_inf(Vm=0.0):
    return alpha_m(Vm) /
(alpha_m(Vm) + beta_m(Vm))

def h_inf(Vm=0.0):
    return alpha_h(Vm) /
(alpha_h(Vm) + beta_h(Vm))
 # Input stimulus
def Id(t):
    if 0.0 < t < 1.0:
        return 150.0
    elif 10.0 < t < 11.0:
        return 50.0
```

```
   return 0.0
 # Compute derivatives
def compute_derivatives(y, t0):
   dy = np.zeros((4,))

   Vm = y[0]
   n = y[1]
   m = y[2]
   h = y[3]

   # dVm/dt
   GK = (gK / Cm) * np.power(n, 
4.0)
   GNa = (gNa / Cm) * np.power(m, 
3.0) * h
   GL = gL / Cm

   dy[0] = (Id(t0) / Cm) - (GK * 
(Vm - VK)) - (GNa * (Vm - VNa)) - 
(GL * (Vm - Vl))

   # dn/dt
   dy[1] = (alpha_n(Vm) * (1.0 - 
n)) - (beta_n(Vm) * n)

   # dm/dt
   dy[2] = (alpha_m(Vm) * (1.0 - 
m)) - (beta_m(Vm) * m)

   # dh/dt
   dy[3] = (alpha_h(Vm) * (1.0 - 
h)) - (beta_h(Vm) * h)

   return dy
 # State (Vm, n, m, h)
Y = np.array([0.0, n_inf(), 
m_inf(), h_inf()])

# Solve ODE system
# Vy = (Vm[t0:tmax], n[t0:tmax], 
m[t0:tmax], h[t0:tmax])
Vy = odeint(compute_derivatives, Y, 
T)

# Input stimulus
```

```
Idv = [Id(t) for t in T]

fig, ax = plt.subplots(figsize=(12, 
7))
ax.plot(T, Idv)
ax.set_xlabel('Time (ms)')
ax.set_ylabel(r'Current density 
(uA/$cm^2$)')
ax.set_title('Stimulus (Current 
density)')
plt.grid()

# Neuron potential
fig, ax = plt.subplots(figsize=(12, 
7))
ax.plot(T, Vy[:, 0])
ax.set_xlabel('Time (ms)')
ax.set_ylabel('Vm (mV)')
ax.set_title('Neuron potential with 
two spikes')
plt.grid()


# Trajectories with limit cycles
fig, ax = plt.subplots(figsize=(10, 
10))
ax.plot(Vy[:, 0], Vy[:, 1], 
label='Vm - n')
ax.plot(Vy[:, 0], Vy[:, 2], 
label='Vm - m')
ax.set_title('Limit cycles')
ax.legend()
plt.grid()
```

Appendix C
(https://colab.research.google.com/github/johanjan/MOOC-HPFEM-source/blob/master/LIF_ei_balance_irregularity.ipynb#scrollTo=oMqaksMrfl1z)

```
import matplotlib.pyplot as plt
# import matplotlib
import numpy as np
# import numpy
import time
# import time
```

```
import ipywidgets as widgets
# interactive display
from scipy.stats import pearsonr
# import pearson correlation

fig_w, fig_h = (6, 4)
plt.rcParams.update({'figure.figsiz
e': (fig_w, fig_h)})

def Poisson_generator(pars, rate,
n, myseed=False):

    '''
    Generates poisson trains
    Expects:
    pars       : parameter
dictionary
    rate       : noise amplitute
[Hz]
    n          : number of Poisson
trains
    myseed     : random seed. int or
boolean

    Returns:
    pre_spike_train : spike train
matrix, ith row represents whether
                      there is a
spike in ith spike train over time
                      (1 if spike, 0
otherwise)
    '''

    # Retrieve simulation parameters
    dt, range_t = pars['dt'],
pars['range_t']
    Lt = range_t.size

    # set random seed
    if myseed:
        np.random.seed(seed=myseed)
    else:
        np.random.seed()

```

```
    # generate uniformly distributed
random variables
    u_rand = np.random.rand(n, Lt)

    # generate Poisson train
    poisson_train = 1. *
(u_rand<rate*dt/1000.)

    return poisson_train

pars = default_pars()
pre_spike_train =
Poisson_generator(pars, rate=10,
n=100)
spT =
pre_spike_train[pre_spike_train.sum
(axis=1)>1.,:]

for i in range(spT.shape[0]):
    t_sp = pars['range_t'][spT[i,
:]>0.5] #spike times
    plt.plot(t_sp,
i*np.ones(len(t_sp)), 'b|')
plt.title('Poisson Generator')
plt.xlabel('Time (ms)')
plt.ylabel('Neuron ID');

pars = default_pars()
pars['T'] = 1000.
pre_spike_train =
Poisson_generator(pars, rate=10,
n=5000)
spT =
pre_spike_train[pre_spike_train.sum
(axis=1)>1.,:]
id_o =
np.arange(pars['range_t'].size)
t_isi_temp = id_o[spT[0, :]>0.5]
t_isi =
np.diff(t_isi_temp)*pars['dt']
for i in range(spT.shape[0]-1):
    t_isi_temp = id_o[spT[i+1,
:]>0.5]
    t_isi = np.concatenate((t_isi,
np.diff(t_isi_temp)*pars['dt']))
```

```
plt.figure(figsize=(9., 4.))


plt.subplot(1,2,1)
plt.title('Poisson Distribution')
isi_count, isi_bin =
np.histogram(t_isi, bins =
np.arange(0, 400.1, 4.))
isi_bin = 0.5 * (isi_bin[1:] +
isi_bin[:-1])
plt.bar(isi_bin, isi_count,
width=4.0, color='b', alpha=0.5)
#plt.legend(loc='upper right')
plt.xlabel('ISI (ms)')
plt.ylabel('Number')

plt.subplot(1,2,2)
plt.title('Probability Function')
isi_count_normalize =
isi_count/isi_count.sum()/(isi_bin[
1]-isi_bin[0]) * 1000. #note the
units
plt.semilogy(isi_bin/1000.,
isi_count_normalize, 'b',
alpha=0.5)
plt.xlabel('ISI (s)')
plt.ylabel(r'Probability density
function (s$^{-1}$)')
plt.tight_layout()


def run_LIF_cond(pars, I_inj,
pre_spike_train_ex,
pre_spike_train_in):
   '''
   conductance-based LIF dynamics

   Expects:
   pars                   : parameter
dictionary
   I_inj                  : injected
current [pA]. The injected current
here can be a value or an array
   pre_spike_train_ex : spike train
input from presynaptic excitatory
neuron
   pre_spike_train_in : spike train
input from presynaptic inhibitory
neuron

   Returns:
   rec_spikes : spike times
   rec_v      : mebrane potential
   gE         : postsynaptic
excitatory conductance
   gI         : postsynaptic
inhibitory conductance
   '''

   # Retrieve parameters
   V_th, V_reset = pars['V_th'],
pars['V_reset']
   tau_m, g_L = pars['tau_m'],
pars['g_L']
   V_init, V_L = pars['V_init'],
pars['V_L']
   gE_bar, gI_bar = pars['gE_bar'],
pars['gI_bar']
   VE, VI = pars['VE'], pars['VI']
   tau_syn_E, tau_syn_I =
pars['tau_syn_E'],
pars['tau_syn_I']
   tref = pars['tref']
   dt, range_t = pars['dt'],
pars['range_t']
   Lt = range_t.size

   # Initialize
   tr = 0.
   v = np.zeros(Lt)
   v[0] = V_init
   gE = np.zeros(Lt)
   gI = np.zeros(Lt)
   I = I_inj * np.ones(Lt) #ensure
I has length Lt

   if pre_spike_train_ex.max() ==
0:
    pre_spike_train_ex_total =
np.zeros(Lt)
   else:
     pre_spike_train_ex_total =
pre_spike_train_ex.sum(axis=0) *
np.ones(Lt)

   if pre_spike_train_in.max() ==
0:
     pre_spike_train_in_total =
np.zeros(Lt)
   else:
     pre_spike_train_in_total =
pre_spike_train_in.sum(axis=0) *
np.ones(Lt)

   # simulation
   rec_spikes = [] # recording
spike times
   for it in range(Lt-1):
     if tr >0:
           v[it] = V_reset
           tr = tr-1
     elif v[it] >= V_th:
#reset voltage and record spike
event
           rec_spikes.append(it)
           v[it] = V_reset
           tr = tref/dt
     #update the synaptic
conductance
     gE[it+1] = gE[it] -
(dt/tau_syn_E)*gE[it] +
gE_bar*pre_spike_train_ex_total[it+
1]
     gI[it+1] = gI[it] -
(dt/tau_syn_I)*gI[it] +
gI_bar*pre_spike_train_in_total[it+
1]

     #calculate the increment of
the membrane potential
     dv = (-(v[it]-V_L) -
(gE[it+1]/g_L)*(v[it]-VE) - \
(gI[it+1]/g_L)*(v[it]-VI) +
I[it]/g_L) * (dt/tau_m)

     #update membrane potential
     v[it+1] = v[it] + dv

   rec_spikes =
np.array(rec_spikes) * dt

   return v, rec_spikes, gE, gI
pars = default_pars()
pars['gE_bar']    = 1.2 #[nS]
pars['VE']        = 0. #[mV]
pars['tau_syn_E'] = 5. #[ms]
pars['gI_bar']    = 1.6 #[nS]
pars['VI']        = -80. #[ms]
pars['tau_syn_I'] = 10. #[ms]

pre_spike_train_ex =
Poisson_generator(pars, rate=10,
n=80)
pre_spike_train_in =
Poisson_generator(pars, rate=10,
n=20) # 4:1

v, rec_spikes, gE, gI =
run_LIF_cond(pars, 0,
pre_spike_train_ex,
pre_spike_train_in)

plt.figure(figsize=(15., 4))
plt.title('Poisson Model Response')
plt.subplot(1,3,1)
plt.plot(pars['range_t'], v, 'b')
plt.xlabel('Time (ms)')
plt.ylabel('V (mV)');

plt.tight_layout()
```

Appendix D
(https://github.com/HiroshiARAKI/spine)

```python
from spine import FitzHughNagumo

import numpy as np
import matplotlib.pyplot as plt


if __name__ == '__main__':
   # init experimental time and
time-step
   time = 100
   dt = 0.1

   # create Hodgkin-Huxley Neuron
   neu = FitzHughNagumo(time, dt)

   # Input data (sin curve)
   input_data = np.sin(0.2 *
np.arange(0, time, dt))
   input_data = np.where(input_data
> 0, 0.8, 0.1)

   v, u = neu.calc_v(input_data)

   # plot
   x = np.arange(0, time, dt)
   plt.subplot(2, 1, 1)
   plt.title('FitzHugh-Nagumo
Neuron model Simulation')
   plt.plot(x, input_data)
   plt.ylabel('input')

   plt.subplot(2, 1, 2)
   plt.plot(x, v, label='v:
recovery variable')
   plt.plot(x, u, label='u:
membrane potential')
   plt.xlabel('time [ms]')

   plt.legend()
   plt.show()
```