Neuroscience, Learning, Memory, Cognition Course

Computer Assignment 2

Dr. Karbalaiee

Soraya Charkas 99101387

Department of Electrical Engineering
Sharif University of Technology

April 2023

[Revised April 8, 2023]

0.1 FitzHugh-Nagumo

0.1.1 Parameters and Variables role in the equation

To determine the role of each variable and parameter, first we need to explain how we achieved these equations.as we know, to determine the firing of a neuron we need to model a fast and slow mechanis. the fast mechanism is modeled with voltage equation and the slow mechanism is our blocking mechanism which we model with mathematical equations using ω.

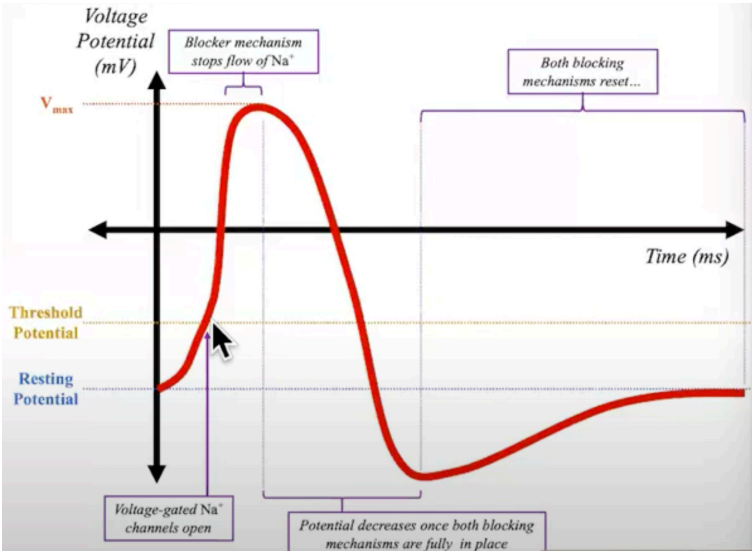

Figure 1: Neuron Firing Model

As we can see in the above picture, we have 3 amount for our voltage:

$$V_{Max}$$

$$V_{Threshold}$$

$$Vrest$$

To ease our job, we scale our voltages as amounts said below:

$$V_{Max} = 1$$

$$V_{rest} = 0$$

$$V_{Threshold} = a$$

If we name our scaled voltages $\tilde{V}$,we can say that there is a linear relation between V and $\tilde{V}$ which is $\tilde{V} = aV + b$.

Solving the equation with three fixed points, we can say that $a = \frac{1}{V_{max} - V_{rest}}$ and $b = \frac{-Vrest}{V_{max} - V_{rest}}$.

The voltage equation:

1-we know that the relation between the amount of voltage and increase in voltage is linear,so the more voltage we have, we expect to have more increase in voltage. this can be written as:

$$\frac{d\tilde{V}}{dt} = \tilde{V}$$

2-there is a maximum voltage which we want our voltage to be equal to the maximum voltage.so we will have:

$$\frac{d\tilde{V}}{dt} = \tilde{V}(1 - \tilde{V})$$

3-we have to consider the threshold voltage as well. if $\tilde{V} < a$,neuron does not fire but if $\tilde{V} > a$ the neuron fires and we have positive feedback.so the equation will be:

$$\frac{d\tilde{V}}{dt} = \tilde{V}(1 - \tilde{V})(\tilde{V} - a)$$

Blocking mechanism:

1-blocking mechanism increases in strength as V increases and ω is the parameter that

models the strength of blocking mechanism so this can be written as:

$$\frac{d\omega}{dt} = \epsilon\tilde{V}$$

2-blocking mechanism gets stronger and we want a battle between the strength of blocking mechanism and $\tilde{V}$. we can model this by:

$$\frac{d\omega}{dt} = \epsilon(\tilde{V} - \gamma\omega)$$

now back to the voltage equation,we need to add $-\omega$ so that the voltage fells the effect of blocking mechanism:

$$\frac{d\tilde{V}}{dt} = \tilde{V}(1 - \tilde{V})(\tilde{V} - a) - \omega$$

but this equation may only model one firing of a neuron, so we add I_{app} to the voltage equation which puts more positive ions into the neuron.the final equations are desired.
to sum up, i have explained each parameter and variable below:

- V = membrane voltage or voltage potential
- ω = strength of blocking mechanism
- I = applied current which puts more positive ion into the neuron
- 0.08 = growth rate of blocking mechanism which is small so the blocking mechanism does not shoot up
- 0.8= rate of difference between ω and $\tilde{V}$ to have a battle between them.
- 0.7 = V_0

0.1.2 Trajectory-python simulation

In this section , we use the given equations and simulate the FitzHugh-Nagumo model in python.

we determined the phase plot. at first, we have to define our FHN model using the equations above.after that, we have to define three different functions for each plot.one for plotting the phase plane, one for plotting trajectory and one for plotting W and V based on t. the functions are as below:

```python
#we are using the equation we got in part 1-1 v(v-1)(a-v)
a = -1
b, r = 0.08, 0.8
f = lambda v: v*(a-v)*(v-1)
dvdt = lambda v, w, I: f(v)-w+I
dwdt = lambda v, w: b*(v-r*w+0.7)
#our FHN model
def FHN(v, w, dt, I, ttot):
    niter = int(ttot/dt)
    vhist = []
    whist = []
    for i in range(niter):
        v, w = (v + dvdt(v, w, I)*dt), (w + dwdt(v, w)*dt)
        vhist.append(v)
        whist.append(w)
    return vhist, whist
```

Figure 2: FHN model in python

```python
#plotting v and w based on t
def plot_fig(y, title="", ylim=(), x=None, xlim=()):
    plt.figure()
    plt.title(title)
    color = ["b", "r"]
    if not x:
        for i in y:
            plt.plot(i)
    else:
        for i,j in enumerate(y):
            plt.plot(x, j, color=color[i])
    if ylim: plt.ylim(ylim)
    if xlim: plt.xlim(xlim)
    plt.xlabel("Time")
    plt.ylabel("Voltage")
    plt.legend(["V(t)", "W(t)"])
    plt.grid()
    plt.show()
#plotting nullcline function
def plot_nullclines(I, xlim, ylim, positions):
    color = ['lightsteelblue', 'gold', 'tomato']*3
    v = np.linspace(xlim[0], xlim[1], 100)
    w = np.linspace(xlim[0], xlim[1], 100)
    v_mesh, w_mesh = np.meshgrid(v, w)
    v_vel = dvdt(v_mesh, w_mesh, I)
    w_vel = dwdt(v_mesh, w_mesh)
    vnc, wnc = nullclines(I, v)
    plt.figure()
    plt.plot(v, vnc, 'b')
    plt.plot(v, wnc, 'r')
    plt.legend(['v nullcline', 'w nullcline'])
    plt.ylim(ylim[0], ylim[1])
    title = "Phase Plot for I=" + str(round(I, 2))
    plt.title(title)
    plt.xlabel('v-values')
    plt.ylabel('w-vaues')
    if positions:
        for i in range(len(positions)):
            plt.streamplot(v_mesh, w_mesh, v_vel, w_vel, density=2, start_points=[positions[i]], color=color[i], int
    else:
        plt.streamplot(v_mesh, w_mesh, v_vel, w_vel, color=color[0])
    plt.grid()
    plt.show()
```

Figure 3: plotting functions

and the plots are going to be as below.first we start with the phase plain:

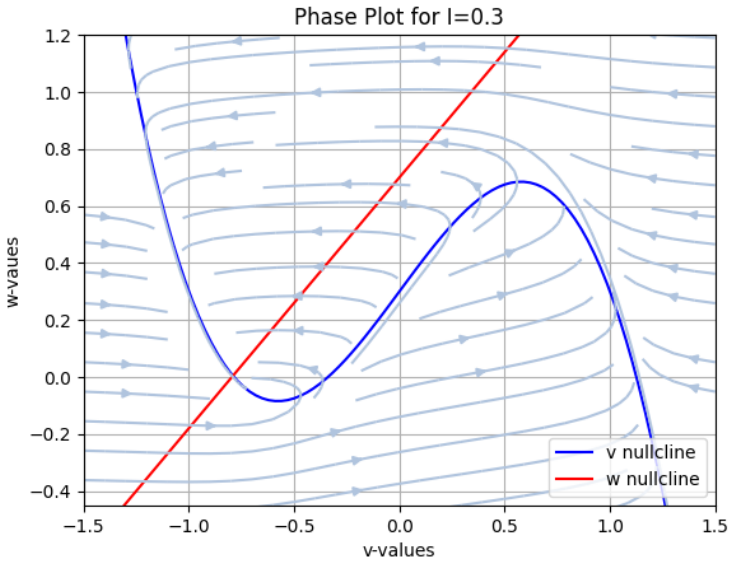

Figure 4: phase plain

as said in the question, we should determine the w,v-t plot for different initial values of v and w . i plotted this figure for two different values which are written above the figures:

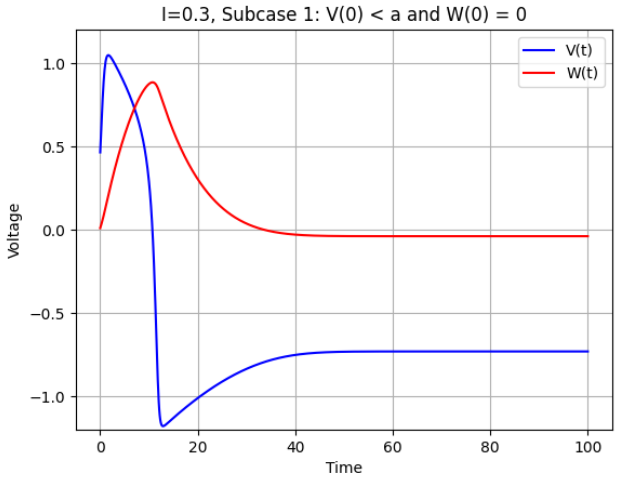

Figure 5: w,v-t

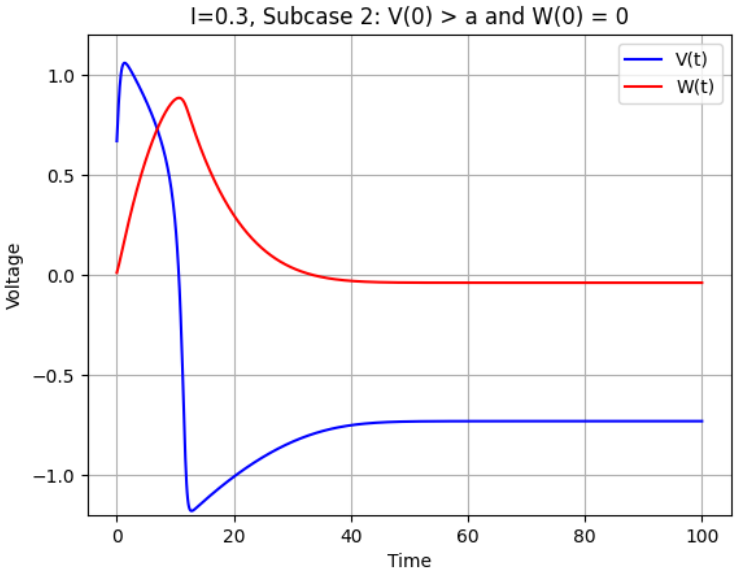

Figure 6: w,v-t

and finally, the trajectory w-v:

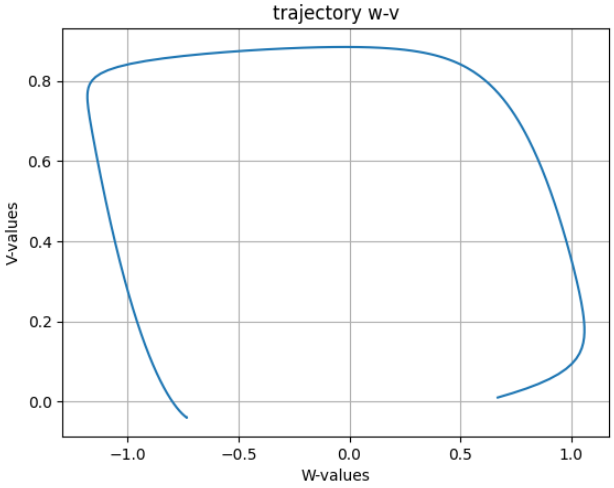

Figure 7: trajectory

0.1.3 stability

Due to the plotted figures,we expect that our system has an only balanced point.we have to determine the value and the type of balanced point using the jacobian matrix. first,we determine the value of our points:

$\frac{dv}{dt} = v - v^3 - \omega + 0.3$, $\frac{d\omega}{dt} = 0.8(v + 0.7 - 8\omega)$
writing v and ω in terms of each other, we will achieve to these points:

$V = -0.73$ and $\omega = 0.04$

$$j(v, w) = \begin{pmatrix} \frac{\partial V}{\partial v} & \frac{\partial V}{\partial \omega} \\ \frac{\partial \omega}{\partial v} & \frac{\partial \omega}{\partial \omega} \end{pmatrix}$$

using the points we got above,we will reach this Eigenvalues:

$\lambda = -0.3 \pm 0.07j$ so both Eigenvalues are negative,which means our balanced point's type is attractive.we can see that our calculations correspond to our simulation results.

0.1.4 stationary

we now have to run our simulation for different values of I.i used the first equation which i named FHN and added a loop for different values of I from 0-10 and plotted V-I ad W-I. as we expected, around the value of 0.3 which we determined at the previous sections that we will have an only balanced point, the values of V and W are changing. in fact, they get stuck into a limit cycle.the figures are as below:

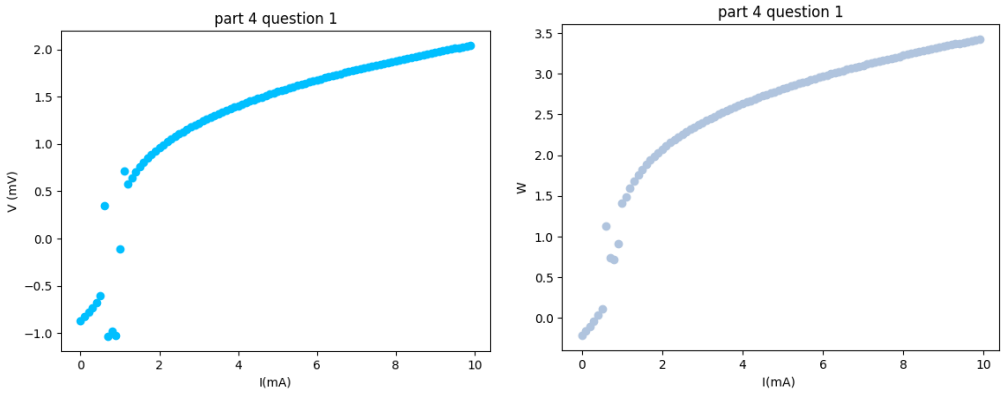

Figure 8: stationary

0.1.5 limit cycle

the answer to this question is yes.as we saw in previous part,near the value of I=0.3,we get an odd response from our simulation and the reason of this is because that our model is stuck into a limit cycle.we can see that for bigger values of I :

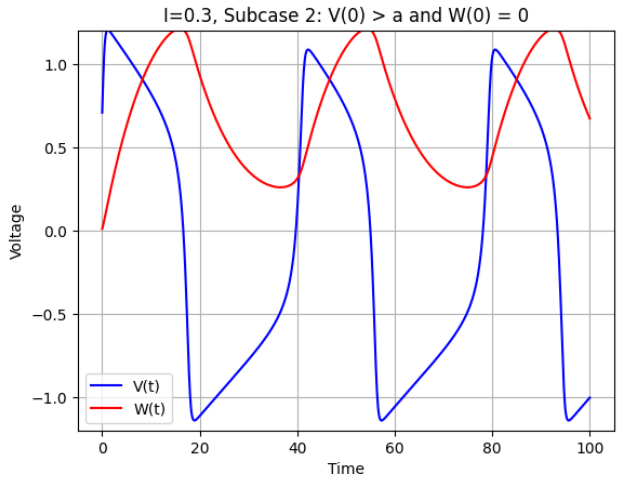

Figure 9: limit cycle

0.1.6 limit cycle:a»1

to answer this question,we can run our simulation for a»1:

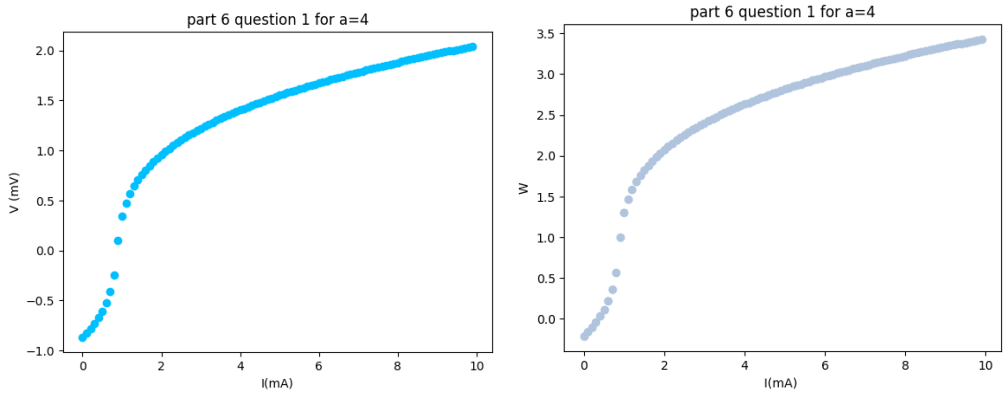

Figure 10: limit cycle

it is obvious that for values o a such that a»1,our neuron model doesn't have a limit cycle.we have to look back to the first part of this question to get our answer.as we said a is growth rate of blocking mechanism which is small so the blocking mechanism does not shoot up , if we increase it, the blocking mechanism shoots up.

0.2 Winner takes all

simulation in this part are similar to the previous part, we have to define a function for the winner takes all model and plot the phase plane and trajectory for different initial values. the written functions are as below:

```
#defining S(x)
def S(x):
    M = 100
    sig = 120
    N = 2
    if (x >= 0):
        return ((M* (x**N)) / ((sig**N)+(x**N)) )
    else:
        return 0
#initial values
tau = 220
e1i = 1
e2i = 0
k1 = 120
k2 = 120
#winner takes all function
def WTA(x, t = 0):
    E1, E2 = x
    dE1 = (-1*E1 + S( (k1-3*E2)))/tau
    dE2 = (-1*E2 + S( (k2-3*E1)))/tau
    return np.array ([dE1, dE2])
#initial conditions
time = np.arange(0 ,1000, 0.1)
e1 = np.zeros (len (time) )
e1[0] = e1i
e2 = np. zeros (len (time) )
e2[0] = e2i
rangex = (-25,75)
rangey = (-25,75)
points = 30
x= np.linspace (rangex[0], rangex[1], points)
y = np.linspace (rangey[0], rangey[1], points)
grid = np.meshgrid(x, y)
diff = np.zeros ( (points, points, 2))
for nx in range (points):
    for ny in range(points):
        df = WTA([grid[0][nx,ny], grid[1][nx,ny]])
        diff[nx, ny, 0] = df[0]
        diff[nx, ny, 1] = df[1]
plt.streamplot(grid[0],grid[1],diff[:,:,0],diff[:,:,1],color='lightsteelblue')
for i in range(0 , len(trange)-1):
    e1[i+1] = e1[i] + dt* (-1 * e1[i] + S((k1 - 3*e2[i])))/tau
    e2[i+1] = e2[i] + dt* (-1 * e2[i] + S((k2 - 3*e1[i])))/tau
plt.plot (e1, e2,'k')
plt.plot (e1[len(trange)-1] ,e2[len(trange)-1] , 'o')
plt.plot (e1[0] , e2[0] , 'x')
plt.contour(grid[0], grid[1], diff[:, :, 0], [0], colors ='b')
plt.contour (grid[0], grid[1], diff[:, :, 1], [0], colors = 'r')
plt.grid()
plt.xlabel ('E1 (mV)')
plt.ylabel ('E2 (mV)')
plt.show()
```

Figure 11: functions

0.2.1 nullcline and limit cycle for different values

for E1=E2=0 we have:

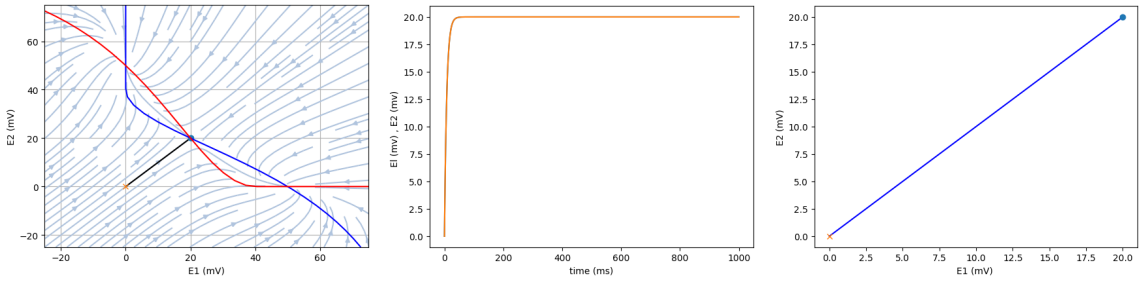

Figure 12: limit cycle for E1=E2=0

for E1=1 and E2=0

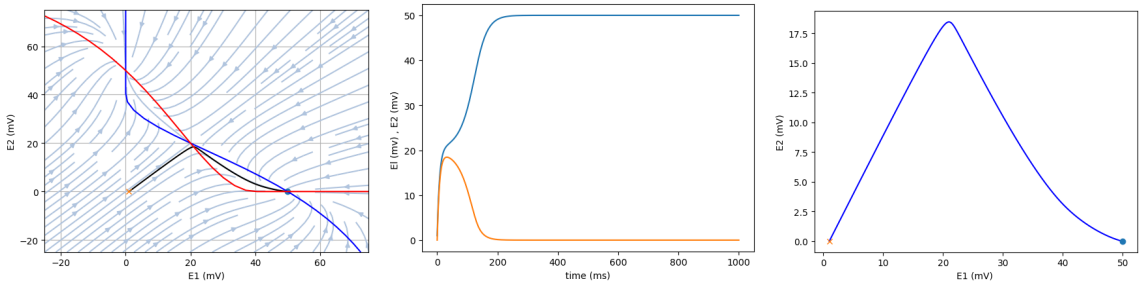

Figure 13: limit cycle for E1=1,E2=0

for E1=0 and E2=1

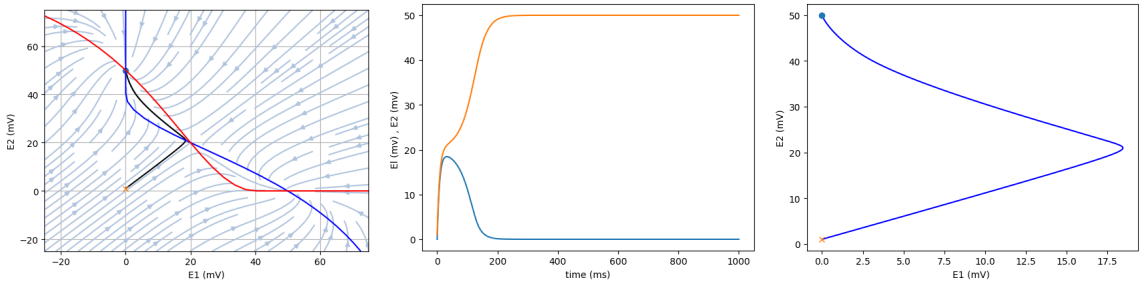

Figure 14: limit cycle for E1=0,E2=1

we can see that the balanced point is around E1=0 and E2=50.the type of balanced point is the gravitation type

0.2.2 E1/E2-t

for E1=1 and E2=0.5

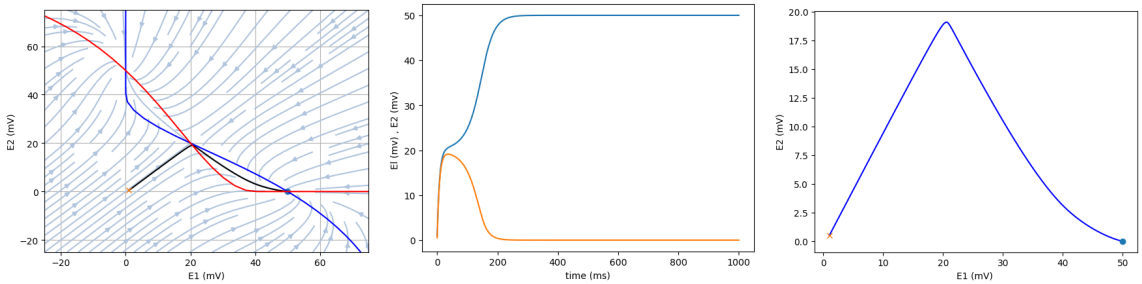

Figure 15: E1/E2-t for E1=1,E2=0.5

for E1=0.49 and E2=0.51

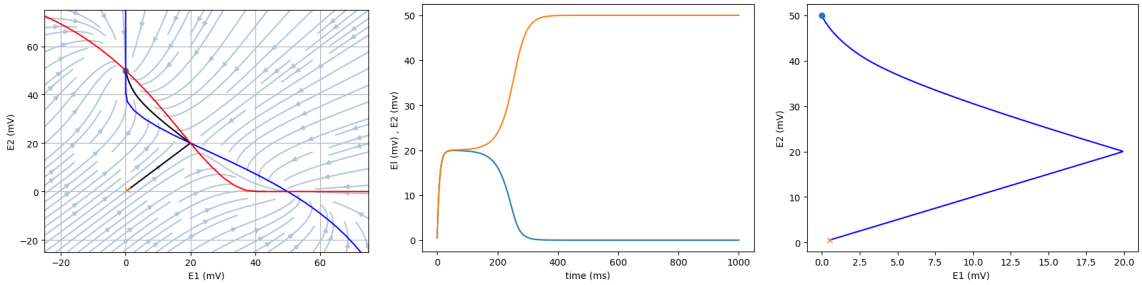

Figure 16: E1/E2-t for E1=0.49,E2=0.51

the balanced point only depends to the primary voltage of the neurons.as you can see,when we only add 0.01 to the voltage of a neuron, the balanced point changes and causes the other neuron to have advantages.i think that's why this model is called winner takes all.

0.2.3 stability

as shown in figures above, we are expecting to have 3 balanced points at E1=50,E2=0 ,E1=E2=20 and E1=0,E2=50.we have to prove this using calculations.

$$20\frac{dE_1}{dt} = -E_1 + (120 - 3E_2)$$

$$20\frac{dE_2}{dt} = -E_2 + (120 - 3E_1)$$

$$S(x) = \frac{x^2}{144 + .01x^2} => E_1 = \frac{(120 - 3E_2)^2}{144 + .01(120 - 3E_2)^2}, E2 = \frac{(120 - 3E_1)^2}{144 + .01(120 - 3E_1)^2}$$

solving this equations, we will get the points below:

$$E_1 = 50, E_2 = 0$$

$$E_1 = E_2 = 20$$

$$E_1 = 0, E_2 = 50$$

now we have to find the type of our balanced points using jacobian matrix:

$$j(E_1, E_2) = \begin{pmatrix} \frac{\partial E_1}{\partial e_1} & \frac{\partial E_1}{\partial e_2} \\ \\ \frac{\partial E_2}{\partial e_1} & \frac{\partial E_2}{\partial e_2} \end{pmatrix}$$

for the first point we have : $\lambda = -1, -1$ this means that our balanced point is attraction type.
for the second point we have : $\lambda = -97, 95$ this means that our balanced point is repulsion type.
for the third point we have : $\lambda = -1, -1$ this means that our balanced point is attraction type. the results are as desired and match our simulation results.

0.2.4 changing the initial values

repeating 2-1:
for E1=E2=0 we have:

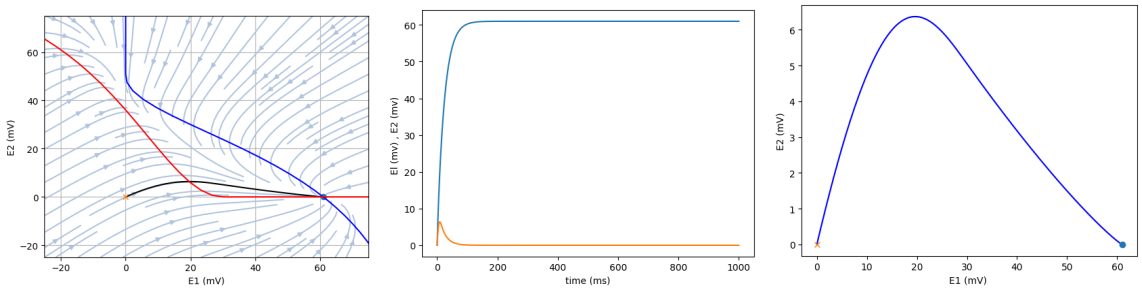

Figure 17: limit cycle for E1=E2=0

for E1=20 and E2=0

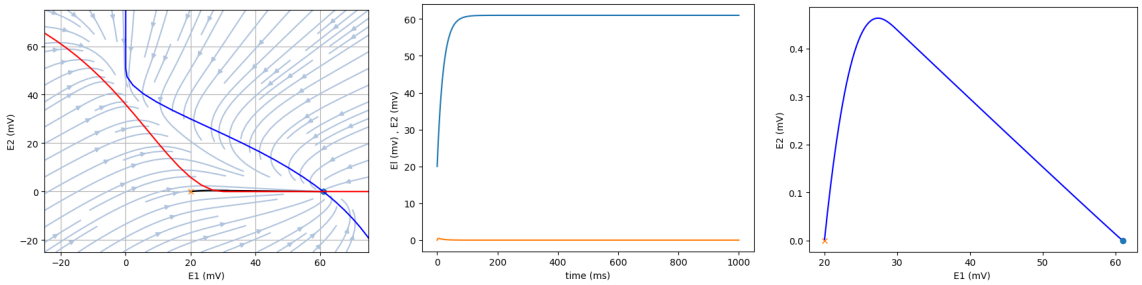

Figure 18: limit cycle for E1=20,E2=0

for E1=0 and E2=20

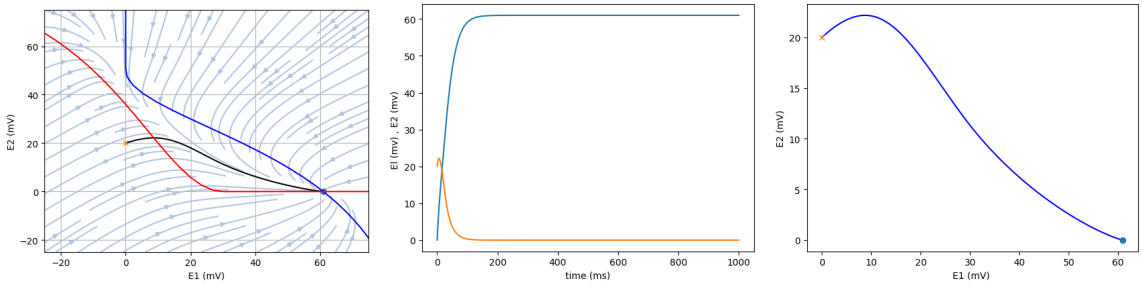

Figure 19: limit cycle for E1=0,E2=20

we had to choose an enormous value for E1 and E2 to see a difference,this shows that because k1>k2, the effect of initial values for E1 and E2 is not that important anymore.because K1>K2, the first neuron always wins.

repeating 2-2:
for E1=0.51 and E2=55.51 we have:

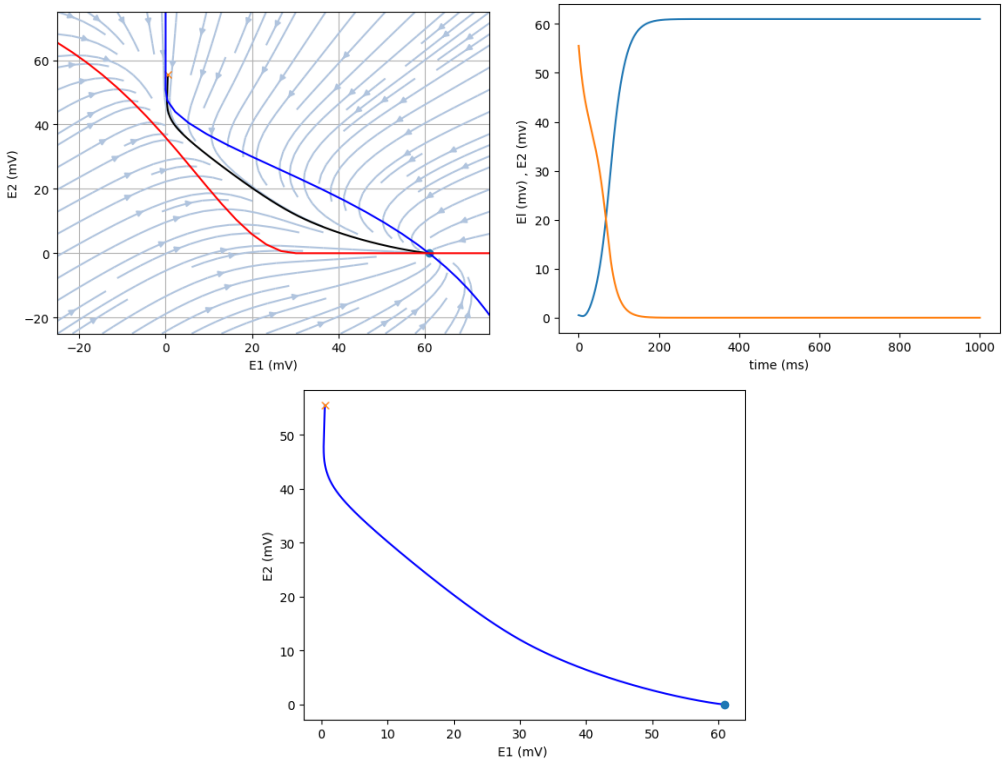

Figure 20: limit cycle for E1=0.51,E2=55.51

although E2=E1+55, we can still see that the effect of K is more important and has caused neuron 1 to win the battle.this approves our conclusion in previous section.
repeating 2-3:
we are expecting to have a balanced point around E1=60 and E2=0 according to our figures. lets compute this.

$$20\frac{dE_1}{dt} = -E_1 + (150 - 3E_2)$$

$$20\frac{dE_2}{dt} = -E_2 + (90 - 3E_1)$$

$$S(x) = \frac{x^2}{144 + .01x^2} => E_1 = \frac{(150 - 3E_2)^2}{144 + .01(150 - 3E_2)^2}, E2 = \frac{(90 - 3E_1)^2}{144 + .01(90 - 3E_1)^2}$$

solving this equations, we will get the points below:

$$E_1 = 61, E_2 = 0$$

now we have to find the type of our balanced points using jacobian matrix:

$$j(E_1, E_2) = \begin{pmatrix} \frac{\partial E_1}{\partial e_1} & \frac{\partial E_1}{\partial e_2} \\ \frac{\partial E_2}{\partial e_1} & \frac{\partial E_2}{\partial e_2} \end{pmatrix}$$

using the jacobian matrix, we will find the eigenvalue as :$\lambda = -1, -1$ so the balanced point is an attraction type.

0.2.5 $\tau = \tau + 20$

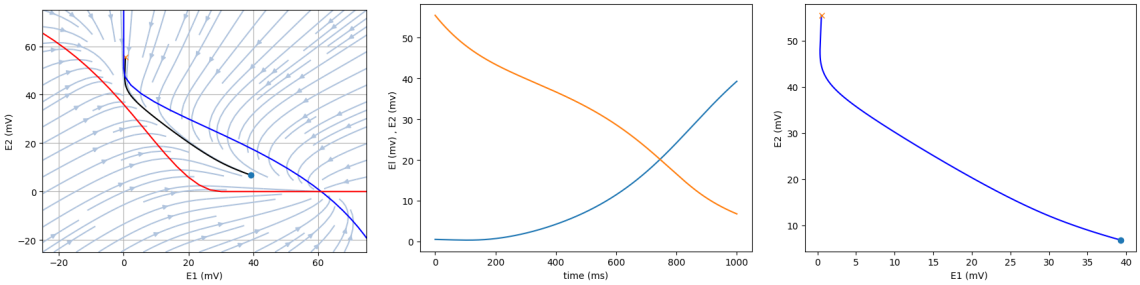

Figure 21: $\tau = \tau + 200$

the balanced point hasn't changed,but the time they reach the fixed point has changed for sure.when we had initialized our τ to 20, the reach time was 100ms. but now, it has become 10 times bigger,we can see that our reach time has become 1000ms.theoretically we would have expected this. because as τ grows,dE_1 and dE_2 become less.which means that they are becoming slower.

0.2.6 conclusion

to conclude, we can say that in this model the only thing that matters is the value of K in each neuron. the neuron with bigger K always wins.other element such as τ only initialize the time that it takes and so on. the only time that other elements are effective is when we are simulating for equal amounts of K for each neuron, as we did in the first part.but except that, K is the most important element in it neuron model.