[CODE] Calcium-mediated regulation of astrocytes response in the brain

January 10, 2022

Raul Adell Víctor Jimenez

1 Imports and generic function definition

[ ]:
```python
import matplotlib.pylab as plt
import numpy as np
from numpy import linalg
from scipy.optimize import fsolve
from scipy.integrate import odeint
import random
import math
from scipy.signal import find_peaks
from scipy.signal import hilbert

from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
def fill_between_3d(ax,x1,y1,z1,x2,y2,z2,c,alpha,mode=1):

    """

    Function similar to the matplotlib.pyplot.fill_between function but
    for 3D plots.

    input:

        ax -> The axis where the function will plot.

        x1 -> 1D array. x coordinates of the first line.
        y1 -> 1D array. y coordinates of the first line.
        z1 -> 1D array. z coordinates of the first line.

        x2 -> 1D array. x coordinates of the second line.
        y2 -> 1D array. y coordinates of the second line.
        z2 -> 1D array. z coordinates of the second line.
    modes:
        mode = 1 -> Fill between the lines using the shortest distance between
                    both. Makes a lot of single trapezoids in the diagonals
                    between lines and then adds them into a single collection.

        mode = 2 -> Uses the lines as the edges of one only 3d polygon.

    Other parameters (for matplotlib):

        c -> the color of the polygon collection.
        alpha -> transparency of the polygon collection.

    """

    if mode == 1:

        for i in range(len(x1)-1):

            verts = [(x1[i],y1[i],z1[i]), (x1[i+1],y1[i+1],z1[i+1])] + \
                    [(x2[i+1],y2[i+1],z2[i+1]), (x2[i],y2[i],z2[i])]

            ax.add_collection3d(Poly3DCollection([verts],
                                                 alpha=alpha,
                                                 linewidths=0,
                                                 color=c))

    if mode == 2:

        verts = [(x1[i],y1[i],z1[i]) for i in range(len(x1))] + \
                [(x2[i],y2[i],z2[i]) for i in range(len(x2))]

        ax.add_collection3d(Poly3DCollection([verts],alpha=alpha,color=c))


def hl_envelopes_idx(s, dmin=1, dmax=1, split=False):
    """
    Input :
    s: 1d-array, data signal from which to extract high and low envelopes
    dmin, dmax: int, optional, size of chunks, use this if the size of the␣
↪input signal is too big
    split: bool, optional, if True, split the signal in half along its mean,␣
↪might help to generate the envelope in some cases
    Output :
    lmin,lmax : high/low envelope idx of input signal s
    """

    # locals min
    lmin = (np.diff(np.sign(np.diff(s))) > 0).nonzero()[0] + 1
    # locals max
    lmax = (np.diff(np.sign(np.diff(s))) < 0).nonzero()[0] + 1

    if split:
        # s_mid is zero if s centered around x-axis or more generally mean of␣
 ↪signal
        s_mid = np.mean(s)
        # pre-sorting of locals min based on relative position with respect to␣
 ↪s_mid
        lmin = lmin[s[lmin] < s_mid]
        # pre-sorting of local max based on relative position with respect to␣
 ↪s_mid
        lmax = lmax[s[lmax] > s_mid]

    # global max of dmax-chunks of locals max
    lmin = lmin[[i+np.argmin(s[lmin[i:i+dmin]])
                 for i in range(0, len(lmin), dmin)]]
    # global min of dmin-chunks of locals min
    lmax = lmax[[i+np.argmax(s[lmax[i:i+dmax]])
                 for i in range(0, len(lmax), dmax)]]

    return lmin, lmax

def system(X, p, li):
    c = X[0]; q = X[1]

    m = p/(p+Kp)
    alfq = a*d1*(p+Kp)/(p+d2)
    nn = c/(c+Kn)
    betq = a*c

    dqdt = alfq*(1-q)-betq*q

    Ji = fv*v1*np.power(m,3)*np.power(nn,3)*np.power(q,3)*((Ct-c)/fv-c)
    Jl = fv*v2*((Ct-c)/fv-c)
    Js = v3*np.power(c,2)/(Ks**2+np.power(c,2))

    dcdt = Ji - Js + Jl

    if li is False:
        Y = np.zeros(2)
        Y[0] = dcdt; Y[1] = dqdt
    else:
        Y = []
        Y.append(dcdt); Y.append(dqdt)
    return Y
```

```
def equilibrium(x0,p):
    it = 0; tolk = 1; xk = x0
    while it < 20 and tolk > 10**-4:
        it = it+1
        h = 10**-9
        fk = system(xk, p, False)
        J = np.zeros((2,2))
        J[:, 0] = (system(xk+np.asarray([h, 0]), p, False) - fk)/h
        J[:, 1] = (system(xk+np.asarray([0, h]), p, False) - fk)/h
        deltaxk = np.linalg.solve(J, -fk)
        xk_ant = xk
        xk = xk + deltaxk
        tolk = np.linalg.norm(xk - xk_ant)
    return xk

def ID3R_2D(p, dt, params, initials):
    # Simulation vectors
    c = np.zeros(len(p)); m = c.copy(); nn = c.copy()
    Ji = c.copy(); Jl = c.copy(); Js = c.copy(); Jc = c.copy()
    q = c.copy(); alfq = c.copy(); betq = c.copy()

    Ct = params[0] # Per si la volem canviar
    Kn = params[7]
    d1 = params[9]
    d2 = params[10]


    # Initial conditions
    c[0] = initials[0]
    q[0] = initials[1]
    m[0] = p[0]/(p[0] + Kp)
    nn[0] = c[0]/(c[0] + Kn)
    betq[0] = a*c[0]

    # They are not zero: look at data from references
    Ji[0] = 0
    Jl[0] = 0.4
    Js[0] = -0.4
    Jc[0] = 0

    # CaEr no la considerem com a varialbe, a partir de la quantitat conservada␣
 ↪CaEr = (Ct - fv)/fv

    for i in range(0, len(p)-1):
        m[i] = p[i]/(p[i]+Kp)
        alfq[i] = a*d1*(p[i]+Kp)/(p[i]+d2)
        Ji[i+1] = fv*v1*m[i]**3*nn[i]**3*q[i]**3*((Ct-c[i])/fv-c[i])
        Jl[i+1] = fv*v2*((Ct-c[i])/fv-c[i])
        Js[i+1] = v3*c[i]**2/(Ks**2+c[i]**2)
        Jc[i+1] = Ji[i] - Js[i] + Jl[i]
        c[i+1] = c[i] + Jc[i]*dt

        nn[i+1] = c[i+1]/(c[i+1]+Kn)
        betq[i+1] = a*c[i+1]
        jq = alfq[i]*(1-q[i])-betq[i]*q[i]
        q[i+1] = q[i] + jq*dt  # Crec que això és així

        if c[i] < 0:  # Per a l'estudi inicial més primitiu
            print(i)
            break
    return c, q
```

2 2D $[Ca^{+2}]([IP_3])$ bifurcation diagram

[ ]:
```
# Simulation parameters
Ct = 2  #uM
fv = 0.18
v1 = 6 #s**-1
v2 = 0.11 #s**-1
v3 = 0.9  #uM /s
Ks = 0.1  #uM
Kp = 0.13  #uM
Kn = 0.08234   #uM
a = 0.2   #uM**-1 s***-1
d1 = 1.049  #uMJi[0] = 0
d2 = 0.9434  #uM
params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]

# Initial conditions:
c0 = 0; q0= 0.25
initials = [c0, q0]

# Time vector for Euler
dt = 0.001
T = 200
tv = np.arange(0, T, dt)

# Now we can build the bifurcation diagram for p constant:
pvals = np.linspace(0,1.25,40)

# 1) Numerically
```

```
osc_c = np.zeros((4, len(pvals))); osc_q = osc_c.copy()
ss_c = np.zeros((2, len(pvals))); ss_q = ss_c.copy()
for i in range(len(pvals)):
    pval = pvals[i]
    p = np.ones(len(tv))*pval   # Es un vector constant
    c, q = ID3R_2D(p, dt, params, initials)
    ss_c[0,i] = pval; ss_c[1,i] = c[-1];
    ss_q[0,i] = pval; ss_q[1,i] = q[-1];

    osc_c[0, i] = pval
    osc_c[1, i] = np.max(c[int(3*len(c)/4):])
    osc_c[2, i] = np.mean(c[int(3*len(c)/4):])
    osc_c[3, i] = np.min(c[int(3*len(c)/4):])

    osc_q[0, i] = pval
    osc_q[1, i] = np.max(q[int(3*len(q)/4):])
    osc_q[2, i] = np.mean(q[int(3*len(q)/4):])
    osc_q[3, i] = np.min(q[int(3*len(q)/4):])

# 2) Seminumerically (Newton to system)
peqs_an = np.zeros((2, len(pvals)))
for i in range(len(pvals)):
    peqs_an[:, i] = equilibrium([ss_c[1, i], ss_q[1, i]], pvals[i])


# 3) Stability by trace-determinant
def jacobian(sys, peq, p):
    J = np.zeros((2,2))
    h = 1e-9
    fk = system(peq, p, False)
    J[:, 0] = (system(peq+np.asarray([h, 0]), p, False) - fk)/h
    J[:, 1] = (system(peq+np.asarray([0, h]), p, False) - fk)/h
    return J

traza = np.zeros(len(pvals))
for i in range(len(pvals)):
    peq = peqs_an[:,i]
    J = jacobian(system, peq, pvals[i])
    traza[i] = np.trace(J)
    vaps, veps = np.linalg.eig(J)

# Ara trobem els valors de p tq tenim bifurcacions, i tmb els seus index
inds = []
for i in range(1,len(pvals)-1):
    left = np.abs(traza[i-1])
    mint = np.abs(traza[i])
    right = np.abs(traza[i+1])
    if left > mint and right > mint:
        inds.append(i)
lowb = inds[0]; highb = inds[1]

plt.figure(1, figsize=(10,6))
pvals_out = np.append(pvals[0:lowb+1], pvals[highb:])
for i in range(0,len(pvals),2):  # AIXO CANVIAR SI CANVIES L'INTERVAL DE LES P
    if i <= lowb or i >= highb+2:
        plt.scatter(ss_c[0,i], ss_c[1,i], color = 'k', s=50)

plt.plot(osc_c[0,lowb:highb+1], osc_c[1,lowb:highb+1], color = 'grey',␣
 ↪linewidth = 3)
plt.plot(osc_c[0,lowb:highb+1], osc_c[3,lowb:highb+1], color = 'grey',␣
 ↪linewidth = 3)
#plt.title(r'Bifurcation diagram $[Ca^{+2}]([IP_3])$', fontsize = 20)
plt.xlabel(r'$[IP_3]$', fontsize = 12)
plt.ylabel(r'$[Ca^{+2}]$', fontsize = 12)


plt.plot(pvals[highb:], peqs_an[0,highb:], color='k', linewidth = 3,␣
 ↪linestyle='--')
plt.plot(pvals[:lowb+1], peqs_an[0,:lowb+1], color='k', linewidth = 3,␣
 ↪linestyle='--', label=r'$(\frac{d[Ca^{+2}]}{dt}, \frac{d[IP_3]}{dt}) = 0$␣
 ↪attractor')
plt.plot(pvals[lowb:highb+1], peqs_an[0,lowb:highb+1], color='red', linewidth =␣
 ↪3, linestyle='--', label=r'$(\frac{d[Ca^{+2}]}{dt}, \frac{d[IP_3]}{dt}) = 0$␣
 ↪repulsor')
plt.fill_between(osc_c[0,lowb:highb+1], osc_c[1,lowb:highb+1], osc_c[3,lowb:
 ↪highb+1], color = 'k', alpha = 0.3, label = r'$[Ca^{+2}]$ oscillations')
plt.fill_between(osc_c[0,:lowb+1], osc_c[1,:lowb+1], osc_c[3,:lowb+1], color =␣
 ↪'k', alpha = 0.1, label = r'$[Ca^{+2}]$ decay ($T_{int} = $' + str(T) + 's)')
plt.fill_between(osc_c[0,highb:], osc_c[1,highb:], osc_c[3,highb:], color =␣
 ↪'k', alpha = 0.1)

# Hopf bifurcation points
plt.scatter(pvals[lowb], peqs_an[0, lowb], color = 'yellow', s=100)
plt.scatter(pvals[highb], peqs_an[0, highb], color = 'yellow', s=100, label =␣
 ↪r'Hopf bifurcation ($Tr(\mathbb{J}) = 0$)')

# Per poder posar la label
plt.scatter(pval, c[-1], color = 'k', s=50,label=r'$[Ca^{+2}]_{SS}$')
plt.text(pvals[int(lowb/3)], 0.15, r'$\Re (\lambda_i) < 0$', fontsize = 14)
plt.text(pvals[int(lowb/3)], 0.12, r'$\Im (\lambda_i) = 0$', fontsize = 14)

plt.text(pvals[int(highb) + int(highb/2)], 0.35, r'$\Re (\lambda_i) < 0$',␣
 ↪fontsize = 14)
```

```
plt.text(pvals[int(highb) + int(highb/2)], 0.32, r'$\Im (\lambda_i) \Im␣
↪(\lambda_j) < 0$', fontsize = 14)
plt.text(0.46, 0.35, r'$\Re (\lambda_i) > 0$', color = 'r', fontsize = 14)
plt.text(0.46, 0.32, r'$\Im (\lambda_i) \neq 0$', color = 'r', fontsize = 14)
plt.legend(loc='best', fontsize = 12)
plt.savefig('C_p.png')
plt.show()
```

3 Calcium behaviour in the presence of step increases p

[ ]:
```
# Faig el temps mes llarg per donar temps a que estabilitzi a cada estat.
dt = 0.001
T = 1000
tv = np.arange(0, T, dt)
N = len(tv)
p = np.zeros(N)
p[int(N/9):int(8*N/9)] += 0.3  # First and last attractors
p[int(2*N/9):int(7*N/9)] += 0.053  #
p[int(3*N/9):int(6*N/9)] += 0.15
p[int(4*N/9):int(5*N/9)] += 0.2

c, q = ID3R_2D(p, dt, params, initials)
plt.figure(figsize=(15,5))
#plt.title(r'$[Ca^{+2}]$ times course in presence of step $[IP_3]$ increases',␣
↪fontsize = 20)
plt.plot(tv, c, 'b', label = r'$[Ca^{+2}]$  ($\mu M$)', linewidth = 2.5)
plt.plot(tv, p, color = 'k', linewidth=1, linestyle='--', label=r'$[IP_3]$ ␣
↪($\mu M$)')
plt.legend(loc="upper right", fontsize=17)
plt.xlabel('time (s)', fontsize=14)
plt.ylabel(r'$[\mu M]$', fontsize=14)
plt.xticks(fontsize = 13)
plt.yticks(fontsize = 13)
```

4 3D $([Ca^{+2}], q)([IP_3])$ bifurcation diagram and trajectories

[ ]:
```
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
# Hopf bifurcation
ax.scatter3D(pvals[highb], peqs_an[0, highb],peqs_an[1, highb], color =␣
↪'yellow', s=100, label = r'Hopf bifurcation ($Tr(\mathbb{J}) = 0$)')
ax.scatter3D(pvals[lowb], peqs_an[0, lowb],peqs_an[1, lowb], color = 'yellow',␣
↪s=100)
# Steady state values
```

```
ax.scatter3D(ss_c[0,:lowb+1], ss_c[1,:lowb+1], ss_q[1,:lowb+1], color='k',␣
 ↪s=60, label=r'$[Ca^{+2}]_{SS}$')
ax.scatter3D(ss_c[0,highb+1:], ss_c[1,highb+1:], ss_q[1,highb+1:], color='k',␣
 ↪s=60)
# Pseudo-numeric equilibrium curves
ax.plot3D(pvals[:lowb+1], peqs_an[0,:lowb+1], peqs_an[1,:lowb+1], linewidth=3,␣
 ↪color='k', linestyle='--')

ax.plot3D(pvals[highb:], peqs_an[0,highb:], peqs_an[1,highb:], linewidth=3,␣
 ↪color='k', linestyle='--',label=r'$(\frac{d[Ca^{+2}]}{dt},␣
 ↪\frac{d[IP_3]}{dt}) = 0$ stable')
ax.plot3D(pvals[lowb:highb+1], peqs_an[0,lowb:highb+1], peqs_an[1,lowb:
 ↪highb+1], linewidth=3, color='r', linestyle='--',␣
 ↪label=r'$(\frac{d[Ca^{+2}]}{dt}, \frac{d[IP_3]}{dt}) = 0$ unstable')


# Inside bifurcation
ax.plot3D(osc_c[0,lowb:highb+1], osc_c[1,lowb:highb+1], osc_q[1,lowb:highb+1],␣
 ↪color = 'grey',linewidth=3, label = r'$[Ca^{+2}]$ oscillations')
ax.plot3D(osc_c[0,lowb:highb+1], osc_c[1,lowb:highb+1], osc_q[3,lowb:highb+1],␣
 ↪color = 'grey',linewidth=3)
ax.plot3D(osc_c[0,lowb:highb+1], osc_c[3,lowb:highb+1], osc_q[1,lowb:highb+1],␣
 ↪color = 'grey',linewidth=3)
ax.plot3D(osc_c[0,lowb:highb+1], osc_c[3,lowb:highb+1], osc_q[3,lowb:highb+1],␣
 ↪color = 'grey',linewidth=3)

# Inside trajectories
count = -1
for bound in [0.40, 0.55, 0.9, 1.2,0.17]:
    count += 1
    ind_mig = np.where(pvals > bound)[0][0]
    pval = pvals[ind_mig]
    p = np.ones(len(tv))*pval

    initials = [peqs_an[0,ind_mig]+1e-5, peqs_an[1,ind_mig]+1e-5]
    c, q = ID3R_2D(p, dt, params, initials)
    if count == 0:  # solo una label
        ax.plot3D(p, c, q, color = 'C1', label = r'$([Ca^{+2}], q)$ outwards')
        ax.scatter3D(p[0], c[0], q[0], s = 20, color = 'lime',␣
 ↪label=r'$([Ca^{+2}], q)_0$ for trajectories')
    else:
        ax.plot3D(p, c, q, color = 'C1')
        ax.scatter3D(p[0], c[0], q[0], s = 20, color = 'lime')

c0 = 0; q0= 0.6
```

```
initials = [c0, q0]
# Ploteamos alguna trayectoria:
count = -1
for pval in [0.2, 0.37, 0.5, 0.8, 1, 0.65, 0.1]:
    count += 1
    p = np.ones(len(tv))*pval
    c, q = ID3R_2D(p, dt, params, initials)
    if count == 0:  # solo una label
        ax.plot3D(p, c, q, color = 'C0', label = r'$([Ca^{+2}], q)$ inwards')
    else:
        ax.plot3D(p, c, q, color = 'C0')
    ax.scatter3D(p[0], c[0], q[0], s = 20, color = 'lime')

# Decays:
# Left
ax.plot3D(osc_c[0,:lowb+1], osc_c[1,:lowb+1], osc_q[1,:lowb+1], color =␣
↪'grey',linewidth=3, linestyle = '--')
ax.plot3D(osc_c[0,:lowb+1], osc_c[1,:lowb+1], osc_q[3,:lowb+1], color =␣
↪'grey',linewidth=3,linestyle = '--')
ax.plot3D(osc_c[0,:lowb+1], osc_c[3,:lowb+1], osc_q[1,:lowb+1], color =␣
↪'grey',linewidth=3,linestyle = '--')
ax.plot3D(osc_c[0,:lowb+1], osc_c[3,:lowb+1], osc_q[3,:lowb+1], color =␣
↪'grey',linewidth=3,linestyle = '--')
# Right
ax.plot3D(osc_c[0,highb:], osc_c[1,highb:], osc_q[1,highb:], color =␣
↪'grey',linewidth=3, linestyle = '--')
ax.plot3D(osc_c[0,highb:], osc_c[1,highb:], osc_q[3,highb:], color =␣
↪'grey',linewidth=3,linestyle = '--')
ax.plot3D(osc_c[0,highb:], osc_c[3,highb:], osc_q[3,highb:], color =␣
↪'grey',linewidth=3,linestyle = '--')
ax.plot3D(osc_c[0,highb:], osc_c[3,highb:], osc_q[1,highb:], color =␣
↪'grey',linewidth=3,linestyle = '--',label = r'$[Ca^{+2}]$ decay')

fill_max = [osc_c[0,lowb:highb+1], osc_c[1,lowb:highb+1], osc_q[1,lowb:highb+1]]
fill_min = [osc_c[0,lowb:highb+1], osc_c[1,lowb:highb+1], osc_q[3,lowb:highb+1]]
fill_between_3d(ax, *fill_max, *fill_min, c='k', alpha=0.2,mode = 1)

fill_max = [osc_c[0,lowb:highb+1], osc_c[3,lowb:highb+1], osc_q[1,lowb:highb+1]]
fill_min = [osc_c[0,lowb:highb+1], osc_c[3,lowb:highb+1], osc_q[3,lowb:highb+1]]
fill_between_3d(ax, *fill_max, *fill_min, c='k', alpha=0.2,mode = 1)



ax.text(0, 0.15, 0.81, r'$\Re (\lambda_i) < 0$', fontsize = 14)
ax.text(0, 0.15,0.79, r'$\Im (\lambda_i) = 0$', fontsize = 14)
ax.text(0.4, 0.32, 0.78,r'$\Re (\lambda_i) > 0$', color = 'k', fontsize = 14)
ax.text(0.4, 0.32,0.76, r'$\Im (\lambda_i) \neq 0$', color = 'k', fontsize = 14)
ax.text(pvals[int(highb) + int(highb/2)], 0.35,0.68, r'$\Re (\lambda_i) < 0$',␣
↪fontsize = 14)
ax.text(pvals[int(highb) + int(highb/2)], 0.35,0.66, r'$\Im (\lambda_i) \Im␣
↪(\lambda_j) < 0$', fontsize = 14)

ax.text(-0.1, 0.15, 0.83, r'attracting node', fontsize = 14)
ax.text(0.32, 0.32, 0.8,r'repulsing focus', color = 'k', fontsize = 14)
ax.text(pvals[int(highb) + int(highb/2)]-0.1, 0.35,0.7, r'attracting focus',␣
↪fontsize = 14)

#ax.set_title("2D Bifurcation and phase portrait", fontsize=20)
ax.set_xlabel(r"$[IP_3]$", fontsize=16)
ax.set_ylabel(r"$[Ca^{+2}]_{SS}$", fontsize=16)
ax.set_zlabel(r"$q$", fontsize=16)
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
ax.zaxis.set_tick_params(labelsize=12)
ax.legend(loc='best', fontsize = 12)
ax.set_xlim3d(0, 1.2)
ax.set_ylim3d(0.08, 0.65)
ax.set_zlim3d(0.55, 0.8)
```

5 3D $([Ca^{+2}])([IP_3], C_{tot})$ bifurcation diagram

[ ]:
```
# Els faig mes generals, depenent del parametre Ct
def system(X, p, Ct, li):
    c = X[0]; q = X[1]

    m = p/(p+Kp)
    alfq = a*d1*(p+Kp)/(p+d2)
    nn = c/(c+Kn)
    betq = a*c

    dqdt = alfq*(1-q)-betq*q

    Ji = fv*v1*np.power(m,3)*np.power(nn,3)*np.power(q,3)*((Ct-c)/fv-c)
    Jl = fv*v2*((Ct-c)/fv-c)
    Js = v3*np.power(c,2)/(Ks**2+np.power(c,2))

    dcdt = Ji - Js + Jl

    if li is False:
        Y = np.zeros(2)
        Y[0] = dcdt; Y[1] = dqdt
    else:
        Y = []
        Y.append(dcdt); Y.append(dqdt)
    return Y

def equilibrium(x0,p,Ct):
    it = 0; tolk = 1; xk = x0
    while it < 20 and tolk > 10**-4:
        it = it+1
        h = 10**-9
        fk = system(xk, p, Ct,False)
        J = np.zeros((2,2))
        J[:, 0] = (system(xk+np.asarray([h, 0]), p, Ct, False) - fk)/h
        J[:, 1] = (system(xk+np.asarray([0, h]), p, Ct, False) - fk)/h
        deltaxk = np.linalg.solve(J, -fk)
        xk_ant = xk
        xk = xk + deltaxk
        tolk = np.linalg.norm(xk - xk_ant)
    return xk

def jacobian2(sys, peq, p, Ct):
    J = np.zeros((2,2))
    h = 1e-10
    fk = system(peq, p, Ct, False)
    J[:, 0] = (system(peq+np.asarray([h, 0]), p, Ct, False) - fk)/h
    J[:, 1] = (system(peq+np.asarray([0, h]), p, Ct, False) - fk)/h
    return J


def build_Ca_Ct_p(dt, pvals, Ctots, initials, params, color, lab):
    data_Cp = np.zeros((len(Ctots), len(pvals))); data_Qp = data_Cp.copy()
    data_Cp_min = data_Cp.copy()
    for i in range(len(Ctots)):
        params[0] = Ctots[i]
        for j in range(len(pvals)):
            pval = pvals[j]
            p = np.zeros(len(tv))
            p[:] = pval  # es constant pel primer apartat
            c,q = ID3R_2D(p, dt, params, initials)
            data_Cp[i,j] = np.max(c[int(3*len(c)/4):])
            data_Qp[i,j] = np.max(q[int(3*len(c)/4):])
            data_Cp_min[i,j] = np.min(c[3*int(len(c)/4):])

    # 2) Seminumerically (Newton to system)
    an_Cp = data_Cp.copy()
    an_Qp = an_Cp.copy()
    for i in range(len(Ctots)):
        for j in range(len(pvals)):
            an_Cp[i, j] = equilibrium([data_Cp[i, j], data_Qp[i,␣
↪j]],pvals[j],Ctots[i])[0]
            an_Qp[i, j] = equilibrium([data_Cp[i, j], data_Qp[i,␣
↪j]],pvals[j],Ctots[i])[1]

    # 3) Stability by trace-determinant
    traza = np.zeros((len(Ctots),len(pvals)))
    for i in range(len(Ctots)):
        for j in range(len(pvals)):
            J = jacobian2(system, [an_Cp[i, j], an_Qp[i, j]], pvals[j],␣
↪Ctots[i])
            traza[i,j] = np.trace(J)
            vaps, veps = np.linalg.eig(J)


    # Index dels punts de bifurcacio analitics
    inds = np.zeros((len(Ctots),2))
    for i in range(len(Ctots)):
        inds_p = []
        for j in range(1,len(pvals)-1):
            left = np.abs(traza[i,j-1])
            mint = np.abs(traza[i,j])
            right = np.abs(traza[i,j+1])
            if left > mint and right > mint:
                inds_p.append(j)

        if len(inds_p) == 2:
            inds[i,0] = inds_p[0]; inds[i,1] = inds_p[1]
        elif len(inds_p) == 1:
            inds[i,0] = inds_p[0]; inds[i,1] = np.nan
        else:
            inds[i,0] = np.nan; inds[i,1] = np.nan


    # Index dels punts de oscilacio i decay
    err_Cp = np.abs(an_Cp - data_Cp)
    p_open = np.zeros(len(Ctots)); p_close = p_open.copy()
    for i in range(len(Ctots)):
        pth = []
        minbool = True
        maxbool = True
        for j in range(len(pvals)):
            if err_Cp[i,j] > 1e-3:
                pth.append(j)
        if len(pth) != 0:
            if min(pth)-1 < 0:
                minbool = False
            if max(pth)+1 >= len(pvals):
                maxbool = False

            if minbool is True and maxbool is True:
                p_open[i] = pvals[min(pth)-1]; p_close[i] = pvals[max(pth)+1]
            elif minbool is False and maxbool is True:
                p_open[i] = np.nan; p_close[i] = pvals[max(pth)+1]
            elif minbool is True and maxbool is False:
                p_open[i] = pvals[min(pth)-1]; p_close[i] = np.nan
            else:
                print('Nonsense')
        else:
            p_open[i] = np.nan; p_close[i] = np.nan

    # EL QUE A MI M'AGRADA
    for i in range(len(Ctots)):
        in_low = np.where(pvals==p_open[i])[0][0]
        try:
            in_high = np.where(pvals==p_close[i])[0][0] + 1
        except IndexError:
            in_high = len(pvals)-1
        if inds[i, 0] < 100 and inds[i, 1] < 100: # != np.nan , HOPF

            # Numeric oscillations:
            ax.plot3D(pvals[int(inds[i,0]):int(inds[i,1])+1],␣
↪data_Cp[i,int(inds[i,0]):int(inds[i,1])+1], Ctots[i], color = color[1],␣
↪linewidth = 3)
            if lab[0] == 0:
                ax.plot3D(pvals[int(inds[i,0]):int(inds[i,1])+1],␣
↪data_Cp_min[i,int(inds[i,0]):int(inds[i,1])+1], Ctots[i], color = color[1],␣
↪linewidth = 3, label=r'$[Ca^{+2}]$ oscillations')
                lab[0] = 1
            else:
                ax.plot3D(pvals[int(inds[i,0]):int(inds[i,1])+1],␣
↪data_Cp_min[i,int(inds[i,0]):int(inds[i,1])+1], Ctots[i], color = color[1],␣
↪linewidth = 3)

            # Decays:
            #Right
            ax.plot3D(pvals[int(inds[i,1]):in_high], data_Cp[i,int(inds[i,1]):
↪in_high], Ctots[i], color = color[1], linestyle='--', linewidth = 3)
            ax.plot3D(pvals[int(inds[i,1]):in_high],␣
↪data_Cp_min[i,int(inds[i,1]):in_high], Ctots[i], color =␣
↪color[1],linestyle='--', linewidth = 3)
            #Left:
            ax.plot3D(pvals[in_low:int(inds[i,0])], data_Cp[i,in_low:
↪int(inds[i,0])], Ctots[i], color = color[1], linestyle='--', linewidth = 3)
```

```
            if lab[1] == 0:
                ax.plot3D(pvals[in_low:int(inds[i,0])], data_Cp_min[i,in_low:
↪int(inds[i,0])], Ctots[i], color = color[1],linestyle='--', linewidth = 3,␣
↪label = r'$[Ca^{+2}]$ decay')
                lab[1] = 1
            else:
                ax.plot3D(pvals[in_low:int(inds[i,0])], data_Cp_min[i,in_low:
↪int(inds[i,0])], Ctots[i], color = color[1],linestyle='--', linewidth = 3)

            # Analytical SS
            ax.plot3D(pvals[0:int(inds[i,0])+1],an_Cp[i,0:
↪int(inds[i,0])+1],Ctots[i],linewidth=1, color=color[0], linestyle='--')
            if lab[2] == 0:
                ax.plot3D(pvals[int(inds[i,0]):int(inds[i,1])+1], an_Cp[i,␣
↪int(inds[i,0]):int(inds[i,1])+1],Ctots[i],linewidth=1, color='r',␣
↪linestyle='--', label=r'$(\frac{d[Ca^{+2}]}{dt}, \frac{d[IP_3]}{dt}) = 0$␣
↪repulsor')
                lab[2] = 1
            else:
                ax.plot3D(pvals[int(inds[i,0]):int(inds[i,1])+1], an_Cp[i,␣
↪int(inds[i,0]):int(inds[i,1])+1],Ctots[i],linewidth=1, color='r',␣
↪linestyle='--')
            if lab[3] == 0:
                ax.plot3D(pvals[int(inds[i,1]):],an_Cp[i,int(inds[i,1]):
↪],Ctots[i],linewidth=1, color=color[0], linestyle='--',label =␣
↪r'$(\frac{d[Ca^{+2}]}{dt}, \frac{d[IP_3]}{dt}) = 0$ attractor')
                lab[3] = 1
            else:
                ax.plot3D(pvals[int(inds[i,1]):],an_Cp[i,int(inds[i,1]):
↪],Ctots[i],linewidth=1, color=color[0], linestyle='--')

            # Fill middle
            Ct_const = np.ones(len(pvals[int(inds[i,0]):
↪int(inds[i,1])+1]))*Ctots[i]
            fill_max = [pvals[int(inds[i,0]):
↪int(inds[i,1])+1],data_Cp[i,int(inds[i,0]):int(inds[i,1])+1], Ct_const]
            fill_min = [pvals[int(inds[i,0]):
↪int(inds[i,1])+1],data_Cp_min[i,int(inds[i,0]):int(inds[i,1])+1], Ct_const]
            fill_between_3d(ax, *fill_max, *fill_min, c=color,alpha=0.6,mode =␣
↪1)

            # Fill decay
            # Right
            Ct_const = np.ones(len(pvals[int(inds[i,0]):in_low+1]))*Ctots[i]
            fill_max = [pvals[int(inds[i,0]):in_low],data_Cp[i,int(inds[i,0]):
↪in_low+1], Ct_const]
```

```
            fill_min = [pvals[int(inds[i,0]):
↪in_low+1],data_Cp_min[i,int(inds[i,0]):in_low+1], Ct_const]
            fill_between_3d(ax, *fill_max, *fill_min, c=color[0],alpha=0.3,mode␣
↪= 1)
            # Left
            Ct_const = np.ones(len(pvals[int(inds[i,1]):in_high+1]))*Ctots[i]
            fill_max = [pvals[int(inds[i,1]):
↪in_high+1],data_Cp[i,int(inds[i,1]):in_high+1], Ct_const]
            fill_min = [pvals[int(inds[i,1]):
↪in_high+1],data_Cp_min[i,int(inds[i,1]):in_high+1], Ct_const]
            fill_between_3d(ax, *fill_max, *fill_min, c=color[0],alpha=0.3,mode␣
↪= 1)

        elif type(inds[i, 1]) is not int and inds[i, 0] < 100:  # Cambio de␣
↪estabilidad
            # Ahora todo es decay:
            # Line:
            ax.plot3D(pvals[in_low:in_high+1], data_Cp[i,in_low:in_high+1],␣
↪Ctots[i], color = color[1], linestyle='--', linewidth = 3)
            ax.plot3D(pvals[in_low:in_high+1], data_Cp_min[i,in_low:in_high+1],␣
↪Ctots[i], color = color[1],linestyle='--', linewidth = 3)
            # Fill:
            Ct_const = np.ones(len(pvals[in_low:in_high+1]))*Ctots[i]
            fill_max = [pvals[in_low:in_high+1],data_Cp[i,in_low:in_high+1],␣
↪Ct_const]
            fill_min = [pvals[in_low:in_high+1],data_Cp_min[i,in_low:
↪in_high+1], Ct_const]
            fill_between_3d(ax, *fill_max, *fill_min, c=color[1],alpha=0.3,mode␣
↪= 1)

            # Analytical SS
            ax.plot3D(pvals[0:int(inds[i,0])+1],an_Cp[i,0:
↪int(inds[i,0])+1],Ctots[i],linewidth=1, color='r', linestyle='--')
            ax.plot3D(pvals[int(inds[i,0]):], an_Cp[i, int(inds[i,0]):
↪],Ctots[i],linewidth=1, color=color[0], linestyle='--')

        else:  # Los dos son NaN, no hay cambio de estabilidad
            # Analytical SS
            ax.plot3D(pvals, an_Cp[i,:],Ctots[i],linewidth=1, color=color[0],␣
↪linestyle='--')


    # Els scatters al final aviam si es veuen millor
    scattered = np.zeros((3, len(Ctots)))
    for i in range(len(Ctots)):
        if inds[i, 0] < 100 and inds[i, 1] < 100: # != np.nan
```

```
            scattered[0,i] = pvals[int(inds[i,0])]
            scattered[1,i] = pvals[int(inds[i,1])]
            scattered[2,i] = Ctots[i]
            ax.scatter3D(pvals[int(inds[i,0])], an_Cp[i, int(inds[i,0])],␣
↪Ctots[i], color = 'yellow', s = 60) # p, C, Ctot
            if lab[4] == 0:
                ax.scatter3D(pvals[int(inds[i,1])], an_Cp[i, int(inds[i,1])],␣
↪Ctots[i], color = 'yellow', s = 60, label = r'Hopf bifurcation␣
↪($Tr(\mathbb{J}) = 0$)')
                lab[4] = 1
            else:
                ax.scatter3D(pvals[int(inds[i,1])], an_Cp[i, int(inds[i,1])],␣
↪Ctots[i], color = 'yellow', s = 60)

        elif type(inds[i, 1]) is not int and inds[i, 0] < 100:
            if lab[5] == 0:
                ax.scatter3D(pvals[int(inds[i,0])], an_Cp[i, int(inds[i,0])],␣
↪Ctots[i], color = 'lime', s = 60, label = r'Stability change')
                lab[5] = 1
            else:
                ax.scatter3D(pvals[int(inds[i,0])], an_Cp[i, int(inds[i,0])],␣
↪Ctots[i], color = 'lime', s = 60)




    ax.set_xlabel(r"$[IP_3]$", fontsize=16)
    ax.set_ylabel(r"$[Ca^{+2}]_{SS}$", fontsize=16)
    ax.set_zlabel(r"$C_{tot}$", fontsize=16)
    ax.xaxis.set_tick_params(labelsize=12)
    ax.yaxis.set_tick_params(labelsize=12)
    ax.zaxis.set_tick_params(labelsize=12)
    ax.legend(loc = 'upper right')
    return data_Cp, data_Qp, scattered

# Simulation parameters
Ct = 2  #uM
fv = 0.18
v1 = 6 #s**-1
v2 = 0.11 #s**-1
v3 = 0.9  #uM /s
Ks = 0.1  #uM
Kp = 0.13  #uM
Kn = 0.08234   #uM
a = 0.2   #uM**-1 s***-1
d1 = 1.049  #uMJi[0] = 0
d2 = 0.9434  #uM
```

```
params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]

# Initials
c0 = 0; q0= 0.25
initials = [c0, q0]

# Time vector for Euler
dt = 0.001
T = 200
tv = np.arange(0, T, dt)

# 1) Now we can build the bifurcation diagram for p constant:
pvals = np.linspace(0,2.25,30)
Ctots = np.linspace(1, 3.5, 8)

fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
data_Cp_o, data_Qp_o, scattered_o = build_Ca_Ct_p(dt, pvals, Ctots, initials,␣
 ↪params, ['k', 'grey'], np.zeros(6))


# 2) Provant amb diferents parametres
# Per fer el seguent apartat, redueixo les lamines Ctot
pvals = np.linspace(0,2,30)
Ctots = np.linspace(1.5, 3, 4)
# Parametres anteriors
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
lab = np.zeros(6)  # Per no repetir les labels
build_Ca_Ct_p(dt, pvals, Ctots, initials, params, ['k', 'grey'], lab)

# Si canvio els parametres
Kn = 0.1
d1 = 1.0
d2 = 0.9
params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]
data_Cp_par, data_Qp_par, scattered_par = build_Ca_Ct_p(dt, pvals, Ctots,␣
 ↪initials, params, ['green', 'green'], lab)


# Per fer un analisi d'estabilitat més extens:
Ct_long = np.linspace(0.8,4,100)
pvals_long = np.linspace(0,2,100)
an_Cp = np.zeros((len(Ct_long), len(pvals_long)))
an_Qp = an_Cp.copy()
an_Cp_par = an_Cp.copy(); an_Qp_par = an_Cp.copy()
for i in range(len(Ct_long)):
    for j in range(len(pvals_long)):
        # Ara falta la initial guess per trobar els punts d'equilibri:
        # Fare servir les dades que tinc amb les p's del primer cas
        ind_rel_Ct = int(i/len(Ct_long)*data_Cp_o.shape[0])
        ind_rel_p = int(j/len(pvals_long)*data_Cp_o.shape[1])
        # Primers parametres
        Kn = 0.08234    #uM
        d1 = 1.049  #uMJi[0] = 0
        d2 = 0.9434  #uM
        params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]
        an_Cp[i, j] = equilibrium([data_Cp_o[ind_rel_Ct, ind_rel_p],␣
↪data_Qp_o[ind_rel_Ct, ind_rel_p]], pvals_long[j],Ct_long[i])[0]
        an_Qp[i, j] = equilibrium([data_Cp_o[ind_rel_Ct, ind_rel_p],␣
↪data_Qp_o[ind_rel_Ct, ind_rel_p]], pvals_long[j],Ct_long[i])[1]

        # # For the parameters:
        Kn = 0.1
        d1 = 1.0
        d2 = 0.9
        params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]
        ind_rel_Ct = int(i/len(Ct_long)*data_Cp_par.shape[0])
        ind_rel_p = int(j/len(pvals_long)*data_Cp_par.shape[1])
        an_Cp_par[i, j] = equilibrium([data_Cp_par[ind_rel_Ct, ind_rel_p],␣
↪data_Qp_par[ind_rel_Ct, ind_rel_p]], pvals_long[j],Ct_long[i])[0]
        an_Qp_par[i, j] = equilibrium([data_Cp_par[ind_rel_Ct, ind_rel_p],␣
↪data_Qp_par[ind_rel_Ct, ind_rel_p]], pvals_long[j],Ct_long[i])[1]



# 3) Stability by trace-determinant
traza = np.zeros((len(Ct_long),len(pvals_long)))
traza_par = traza.copy()
for i in range(len(Ct_long)):
    for j in range(len(pvals_long)):
        J = jacobian2(system, [an_Cp[i, j], an_Qp[i, j]], pvals_long[j],␣
↪Ct_long[i])
        traza[i,j] = np.trace(J)

        J_par = jacobian2(system, [an_Cp_par[i, j], an_Qp_par[i, j]],␣
↪pvals_long[j], Ct_long[i])
        traza_par[i,j] = np.trace(J_par)

# Index dels punts de bifurcacio analitics
pv = np.zeros((len(Ct_long), 2))
pv_par = np.zeros((len(Ct_long), 2))
for i in range(len(Ct_long)):
    inds_p = []
    for j in range(1,len(pvals_long)-1):
        left = np.abs(traza[i,j-1])
        mint = np.abs(traza[i,j])
        right = np.abs(traza[i,j+1])
        if left > mint and right > mint:
            inds_p.append(j)
    if len(inds_p) == 2:
        pv[i,0] = pvals_long[inds_p[0]]; pv[i,1] = pvals_long[inds_p[1]]

    inds_p_par = []
    for j in range(1,len(pvals_long)-1):
        left = np.abs(traza_par[i,j-1])
        mint = np.abs(traza_par[i,j])
        right = np.abs(traza_par[i,j+1])
        if left > mint and right > mint:
            inds_p_par.append(j)
    if len(inds_p_par) == 2:
        pv_par[i,0] = pvals_long[inds_p_par[0]]; pv_par[i,1] =␣
 ↪pvals_long[inds_p_par[1]]



# Ara elimino els zeros:
ind_zero = np.where(pv[:,0] == 0)
ps0 = []; ps1=[]; Cts = []
count = 0
for i in range(len(pv[:,0])):
    if i != ind_zero[0][count]:
        ps0.append(pv[i,0])
        ps1.append(pv[i,1])
        Cts.append(Ct_long[i])
    else:
        if len(ind_zero[0]) > count:
            count +=1

ind_zero = np.where(pv_par[:,0] == 0)
ps_par0 = []; ps_par1 = []; Cts_par = []
count = 0
for i in range(len(pv_par[:,0])):
    if i != ind_zero[0][count]:
        ps_par0.append(pv_par[i,0])
        ps_par1.append(pv_par[i,1])
        Cts_par.append(Ct_long[i])
    else:
        if len(ind_zero[0])-1 > count:
            count +=1
```

```
plt.figure(figsize = (7,5))
plt.plot(ps0, Cts,linewidth=2, color ='k')
plt.plot(ps1, Cts, linewidth=2, color ='k',label=r'$K_n = 0.08234$, $d_1 = 1.
↪049$, $d_2 = 0.9434$  [$\mu M$]')
plt.plot(ps_par0, Cts_par,linewidth=2, color ='g')
plt.plot(ps_par1, Cts_par,linewidth=2, color ='g',label=r'$K_n = 0.9$, $d_1 = 1.
↪0$, $d_2 = 0.9$  [$\mu M$]')
plt.legend(loc='best')
#plt.fill_between(ps_par0, )
plt.xlabel(r'$[IP_3]$', fontsize = 12)
plt.ylabel(r'$C_{tot}$', fontsize = 12)

# I els scattered d'abans
for i in range(len(scattered_o[0,:])):
    if scattered_o[0,i] != 0:
        plt.scatter(scattered_o[0,i], scattered_o[2,i], color = 'yellow', s =␣
↪40)
        plt.scatter(scattered_o[1,i], scattered_o[2,i], color = 'yellow', s =␣
↪40)
        plt.scatter(scattered_par[0,i], scattered_par[2,i], color = 'lime', s =␣
↪40)
        plt.scatter(scattered_par[1,i], scattered_par[2,i], color = 'lime', s =␣
↪40)
```

6 Ca^{+2} response to preset dynamics for $[IP_3]$

[ ]:
```
# Tornem als parametres del a) (diagrama de bifurcacio m'agrada mes, i el volem␣
↪recorrer)
Kn = 0.08234   #uM
d1 = 1.049  #uMJi[0] = 0
d2 = 0.9434  #uM
params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]

# Initials
c0 = 0.16; q0= 0.25
initials = [c0, q0]

# 1) P PROPOSADA
dt = 0.001
T = 120
tv = np.arange(0, T, dt)
```

```
# T = 120
# A = 0.375, d rise = 36, r rise = 0.002, and d decay = 120  SP
# A = 0.4, drise=30; rrise=0.02 ; ddec = 90 SP
# A = 0.9; drise = 16; rrise = 0.01; ddec = 120  LL
# A = 0.45; drise = 27; rrise = 0.1; ddec = 90 MP
# A = 0.9; drise = 16; rrise = 0.01; ddec = 90 PT

parsfun = [[0.375,36,0.002,120], [0.9,16,0.01,120],[0.45,27,0.1,90],[0.9,16,0.
 ↪01,90]]
titles = ['Single-Peak response (SP)', 'Long-Lasting response (LL)',␣
 ↪'Multi-Peak response (MP)', 'Plateau response (PT)']
lletres = ['a)', 'b)','c)','d)']
plt.figure(figsize=(40,7))
#plt.suptitle(r'$[Ca^{+2}]$ transients for given $[IP_3](t)$', fontsize = 22)
for i in range(len(parsfun)):
    A = parsfun[i][0] # Maximum amplitude (0.2-0.9 uM)
    ind_stim = int(len(tv)/6)   # per exemple

    drise = parsfun[i][1] # Duration of IP3 increase (1-41 s)
    ind_drise = int((drise/tv[-1])*len(tv))  # per exemple

    rrise = parsfun[i][2] # Rate of expoential growth (0.002-12 s**-1)
    ddec = parsfun[i][3] # Duration of IP3 decline (15-220 s)
    rdec = (-1/ddec)*np.log(0.005/A)
    s_inf = A/(1-np.exp(-rrise*drise))

    p = np.zeros(len(tv))
    diff_t = tv[ind_stim:ind_stim+ind_drise] - tv[ind_stim]
    p[ind_stim:ind_stim+ind_drise] = s_inf*(1-np.exp(-rrise*diff_t))

    diff_t = tv[ind_stim+ind_drise:] - tv[ind_stim] - drise
    p[ind_stim+ind_drise:] = A*np.exp(-rdec*diff_t)

    plt.subplot(1,4,i+1)
    plt.title(titles[i], fontsize = 18)
    plt.text(0.01*tv[-1], 0.9*np.max(p), lletres[i], fontsize=18)
    c, q = ID3R_2D(p, dt, params,initials)
    plt.plot(tv, c, 'b', label = r'$[Ca^{+2}]$  ($\mu M$)', linewidth = 2.5)
    plt.plot(tv, p, color = 'k', linewidth=1, linestyle='--', label=r'$[IP_3]$ ␣
 ↪($\mu M$)')
    if i == len(parsfun)-1:
        plt.legend(loc="upper right", fontsize=17)
    plt.xlabel('time (s)', fontsize=14)
    plt.ylabel(r'$[\mu M]$', fontsize=14)
    plt.xticks(fontsize = 13)
    plt.yticks(fontsize = 13)
```

7 Study of feedbackless neuron-astrocyte model

[ ]:
```
from scipy.optimize import fsolve

def find_r(Hmean, r, pss):
    # Time vector for IP3:
    dt = 10**-3
    T = 2
    tv = np.arange(0, T, dt)

    # References [4], en segons:
    tip3 = 0.14
    peq = 0.16
    p = np.zeros(len(tv))
    for i in range(len(tv)-1):
        p[i+1] = p[i] + dt*((1/tip3)*(peq - p[i]) + r*Hmean)

    pfix = np.mean(p[int(3*len(tv)/8):int(len(tv)/2)])
    return pfix - pss


parameters = []
psss = [0.2, 0.5, 1]; T0s = [0,15,15]; Ts = [35,30,70]
for i in range(3):
    pss = psss[i]
    Hmeans = np.zeros(100)
    rs = np.linspace(T0s[i],Ts[i],100)

    for i in range(len(rs)):
        Hmeans[i] = fsolve(find_r, 15, args=(rs[i], pss))

    plt.figure()
    plt.title('Possibles valors de r amb $p_{SS} = $' + str(pss))
    plt.plot(rs, Hmeans)
    plt.xlabel('r')
    plt.ylabel(r'$\theta_{mean}$')

    # indmin = np.where(Hmeans == min(Hmeans))[0][0]
    # print('')
    # print(r'Per $p_{SS} = $' + str(pss))
    # print('La $\theta_{mean}$ minima es = ' + str(Hmeans[indmin]))
    # print(r'La r correspondent a $\theta_{mean}^{max}$ es: ' + str(rs[indmin]))
    # print('')
    # Pels valors habituals de Iext tenim \theta_{mean} ~ 0.1

    # 0.098 es el valor que trobem al model real amb Iext = 15 uA
    ind01 = np.where(Hmeans > 0.098)[0][-1]
    print('Triem:')
    print('r = ' + str(rs[ind01]))
    print(r'\theta_{mean} = ' + str(Hmeans[ind01]))

    parameters.append((rs[ind01], Hmeans[ind01]))


# I ara els verifiquem amb H senzilla:

# Time vector for IP3:
dt = 10**-3
T = 175
tv = np.arange(0, T, dt)

for i in range(3):
    Hmean = np.zeros(len(tv))
    Hmean[:int(3*len(tv)/4)] = parameters[i][1]
    r = parameters[i][0]

    # References [4], en segons:
    tip3 = 0.14
    peq = 0.16
    p = np.zeros(len(tv))
    for i in range(len(tv)-1):
        p[i+1] = p[i] + dt*((1/tip3)*(peq - p[i]) + r*Hmean[i])

    c, q = ID3R_2D(p, dt, params, initials)

    plt.figure()
    plt.plot(tv, Hmean, label = 'Vmean')
    plt.plot(tv, c, label='c')
    plt.plot(tv, q, label='q')
    plt.plot(tv, p, label='p')
    plt.legend(loc='best')
```

8 Neuron voltage, part 1 of feedbackless model

[ ]:
```
from mpl_toolkits.axes_grid.inset_locator import (inset_axes, InsetPosition,
                                                  mark_inset)

parameters = [(2.8282828282828283, 0.10211292767248088),
 (24.848484848484485, 0.0980460711727733),
 (61.111111111111114, 0.09843463261370276)]

def heavy(V):
    if V >= 50 and V != np.nan:  # in mV
```

```
        y = 1
    else:
        y = 0
    return y


def hodgkin_huxley(t, Iext):
    # EXTERNAL CURRENT MODEL___________________________________________
    p = np.zeros(len(t)); V = p.copy(); ncu = p.copy(); mcu = p.copy(); hcu = p.
 ↪copy()
    Iastro = p.copy()

    vk = -12.0 #mv
    vna = 115.0 #mv
    vl = 10.6 #mv
    gna = 120 #mS/cm^2
    gk = 36.0 #mS/cm^2
    gl = 0.3 #mS/cm^2
    Cm = 1 #uF/cm^2
    Cm = 1 #uF/cm^2

    V[0] = 0;    dt = t[1] - t[0];

    for i in range(len(t)-1):
        alfm = 0.1*(25-V[i])/(np.exp((25-V[i])/10)-1)
        betm = 4*np.exp(-V[i]/18)
        alfh = 0.07*np.exp(-V[i]/20)
        beth = 1/(np.exp((30-V[i])/10)+1)
        alfn = 0.01*(10 - V[i])/(np.exp((10-V[i])/10)-1)
        betn = 0.125*np.exp(-V[i]/80)

        mcu[i+1] = mcu[i] + dt*(alfm*(1-mcu[i]) - betm*mcu[i])
        ncu[i+1] = ncu[i] + dt*(alfn*(1-ncu[i]) - betn*ncu[i])
        hcu[i+1] = hcu[i] + dt*(alfh*(1-hcu[i]) - beth*hcu[i])

        '''
        if c[i+1]*10**3 > 196.69:
            Iastro[i] = 2.11 * heavy(np.log(c[i+1]*10**3-196.69)+50) *np.
 ↪log(c[i+1]*10**3-196.69)
        '''

        Jv = -(gk/Cm)*(ncu[i]**4)*(V[i]-vk) -(gna/
 ↪Cm)*(mcu[i]**3)*hcu[i]*(V[i]-vna) - (gl/Cm)*(V[i]-vl) + (Iext[i] +␣
 ↪Iastro[i])/Cm
        V[i+1] = V[i] + dt*Jv
    return V
```

```
# Initials
c0 = 0.16; q0= 0.25
initials = [c0, q0]

# Simulation parameters
Ct = 2  #uM
fv = 0.18
v1 = 6 #s**-1
v2 = 0.11 #s**-1
v3 = 0.9  #uM /s
Ks = 0.1  #uM
Kp = 0.13  #uM
a = 0.2   #uM**-1 s***-1

Kn = 0.1
d1 = 1.049
d2 = 0.9434
params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]

# Time vector for IP3:
dt = 10**-3
T = 175
tv = np.arange(0, T, dt)


# Time vector for Hodgkin-Huxley:
T_hh = len(tv)
t_hh = np.arange(0, T_hh, dt)

# Intensities
Iext0 = 15
Iext = np.zeros(len(t_hh))
Iext[:int(3*len(t_hh)/4)] = Iext0

V = hodgkin_huxley(t_hh, Iext)
```

9 Astrocyte with mean of neuron heaviside voltage

[ ]:
```
Hmean = np.zeros(len(tv))
for j in range(0, len(tv)-1):
    count = 0
    aa = np.asarray(V[(j)*10**3:(j+1)*10**3])
    count = (aa > 50).sum()
    Hmean[j] = count/(10**3)

print('')
print(r'\theta_{mean} en SS = ' + str(np.mean(Hmean[int(1*len(tv)/4):
↪int(3*len(tv)/4)])) + ' ~ 0.1')
print('')

# References [4], en segons:
tip3 = 0.14
peq = 0.16

# Per dos valors de r
for i in range(3) :
    r = parameters[i][0]
    p = np.zeros(len(tv))
    for j in range(len(tv)-1):
        p[j+1] = p[j] + dt*((1/tip3)*(peq - p[j]) + r*Hmean[j])

    c, q = ID3R_2D(p, dt, params, initials)

    fig, ax1 = plt.subplots()
    ax1.set_title('r = ' + str(round(r,3)))
    ax1.plot(tv, p, label=r'$[IP_3]$')
    ax1.plot(tv, Hmean*0.05/max(Hmean), color ='blue', label = 'V (mean>50,␣
↪normalized)')  # Normalitzat
    ax1.plot(tv, Iext[::1000]*(0.025/max(Iext)), label = r'$I_{ext}$␣
↪(normalized)')  # Normalitzat
    ax1.plot(tv, c, label=r'$[Ca^{+2}]$')
    ax1.set_xlabel('time (s)')
    ax1.set_ylabel(r'$[\mu M]$')
    ax1.legend(loc='upper left')

    ax2 = plt.axes([0,0,0.5,0.5])
    ip = InsetPosition(ax1, [0.65,0.63,0.3,0.3])
    ax2.set_axes_locator(ip)
    ax2.plot(t_hh[:int(5e4)], V[:int(5e4)])
    ax2.legend('V', fontsize=6)


    # plt.figure()
    # plt.plot(t_hh, Iext, label='Iext')
    # plt.plot(t_hh, V, label='V')
    # plt.legend(loc='best')
    # plt.show()

    pfix = np.mean(p[int(3*len(tv)/8):int(len(tv)/2)])
    print(r'Tenim $p_{SS}$ = ' + str(pfix))
    # Vull la Hmean en línterval estacionari
    print(r'Obtenim $\theta_{mean}$ = ' + str(np.mean(Hmean[int(1*len(tv)/4):
↪int(3*len(tv)/4)])))
```

```
    print('')
```

10 Full 3D model with feedback implemented

```
[ ]: rs = [2.8282828282828283, 24.848484848484850, 61.111111111111114]

def heavy(V):
    if V >= 50 and V != np.nan:  # in mV
        y = 1
    else:
        y = 0
    return y

def ID3R_3D(t,Iext, params, r):
    # Some parameters
    # References [4]
    #tip3 = 0.0001400e3 # (s**-1)
    tip3 = 0.14
    #peq = 160.0e-3 #uM
    peq = 0.16
    #r = 0.2 #uM/s segons fig 2 [4]

    Ct = params[0] # Per si la volem canviar
    Kn = params[7]
    d1 = params[9]
    d2 = params[10]


    # Simulation vectors
    c = np.zeros(len(t)); m = c.copy(); nn = c.copy()
    Ji = c.copy(); Jl = c.copy(); Js = c.copy(); Jc = c.copy()
    q = c.copy(); alfq = c.copy(); betq = c.copy()
    p = c.copy(); V = c.copy(); ncu = c.copy(); mcu = c.copy(); hcu = c.copy()
    Iastro = c.copy()

    # Initial conditions
    p[0] = 0  # Ref [5]
    c[0] = 0.1  # Ref [4]
    q[0] = 0.25  # invent
    m[0] = p[0]/(p[0] + Kp)
    nn[0] = c[0]/(c[0] + Kn)
    betq[0] = a*c[0]

    if c[0]*10**3 > 196.69:
        Iastro[0] = 2.11 * heavy(np.log(c[0]*10**3-196.69)+50) *np.
↪log(c[0]*10**3-196.69)
```

```
# They are not zero: look at data from references
Ji[0] = 0
Jl[0] = 0.4  # refs
Js[0] = -0.4  # refs
Jc[0] = 0

vk = -12.0 #mv
vna = 115.0 #mv
vl = 10.6 #mv
gna = 120 #mS/cm^2
gk = 36.0 #mS/cm^2
gl = 0.3 #mS/cm^2
Cm = 1 #uF/cm^2
Cm = 1 #uF/cm^2

V[0] = 0;    dt = t[1] - t[0];

for i in range(0, len(p)-1):
    m[i] = p[i]/(p[i]+Kp)
    alfq[i] = a*d1*(p[i]+Kp)/(p[i]+d2)

    Ji[i+1] = fv*v1*m[i]**3*nn[i]**3*q[i]**3*((Ct-c[i])/fv-c[i])
    Jl[i+1] = fv*v2*((Ct-c[i])/fv-c[i])
    Js[i+1] = v3*c[i]**2/(Ks**2+c[i]**2)
    Jc[i+1] = Ji[i] - Js[i] + Jl[i]
    c[i+1] = c[i] + Jc[i]*dt*10**-3

    nn[i+1] = c[i+1]/(c[i+1]+Kn)
    betq[i+1] = a*c[i+1]
    jq = alfq[i]*(1-q[i])-betq[i]*q[i]
    q[i+1] = q[i] + jq*dt*10**-3

    p[i+1] = p[i] + dt * 10**-3 * ((1/tip3)*(peq - p[i]) + r*heavy(V[i]))

    # EXTERNAL CURRENT MODEL___________________________________________

    alfm = 0.1*(25-V[i])/(np.exp((25-V[i])/10)-1)
    betm = 4*np.exp(-V[i]/18)
    alfh = 0.07*np.exp(-V[i]/20)
    beth = 1/(np.exp((30-V[i])/10)+1)
    alfn = 0.01*(10 - V[i])/(np.exp((10-V[i])/10)-1)
    betn = 0.125*np.exp(-V[i]/80)

    mcu[i+1] = mcu[i] + dt*(alfm*(1-mcu[i]) - betm*mcu[i])
    ncu[i+1] = ncu[i] + dt*(alfn*(1-ncu[i]) - betn*ncu[i])
    hcu[i+1] = hcu[i] + dt*(alfh*(1-hcu[i]) - beth*hcu[i])
```

```
        if c[i+1]*10**3 > 196.69:
            Iastro[i] = 2.11 * heavy(np.log(c[i+1]*10**3-196.69)+50) *np.
 ↪log(c[i+1]*10**3-196.69)

        Jv = -(gk/Cm)*(ncu[i]**4)*(V[i]-vk) -(gna/
 ↪Cm)*(mcu[i]**3)*hcu[i]*(V[i]-vna) - (gl/Cm)*(V[i]-vl) + (Iext[i] +␣
 ↪Iastro[i])/Cm
        V[i+1] = V[i] + dt*Jv

        if c[i] < 0:  # Per a l'estudi inicial més primitiu
            print(i)
            break
    return c, q, p, V, Jv

# Simulation parameters
Ct = 2  #uM
fv = 0.18
v1 = 6 #s**-1
v2 = 0.11 #s**-1
v3 = 0.9  #uM /s
Ks = 0.1  #uM
Kp = 0.13  #uM
a = 0.2   #uM**-1 s***-1

Kn = 0.1
d1 = 1.049
d2 = 0.9434
params = [Ct,fv,v1,v2,v3,Ks,Kp,Kn,a,d1,d2]

# time vector for this section
dt = 10**-3
T = 100*10**3          ␣
 ↪#------------------------------------------------------------------------------------
tv = np.arange(0, T, dt)

# Intensities
Iext = np.zeros(len(tv))
Iext[:int(len(tv)/2)] = 15


for i in range(len(rs)):

    c,q,p,V,Iastro = ID3R_3D(tv,Iext, params, rs[i])

    fig, ax1 = plt.subplots()
    ax1.set_title('r = ' + str(round(rs[i])))
    ax1.plot(tv[::1000]/10**3 , p[::1000], label=r'$[IP_3]$')
    ax1.plot(tv[::1000]/10**3 , V[::1000]*0.05/max(V), label = 'V␣
↪(normalized)')  # Normalitzat
    ax1.plot(tv[::1000]/10**3 , Iext[::1000]*(0.025/max(Iext)), label =␣
↪r'$I_{ext}$ (normalized)')  # Normalitzat
    ax1.plot(tv[::1000]/10**3 , c[::1000], label=r'$[Ca^{+2}]$')
    ax1.set_xlabel('time (s)')
    ax1.set_ylabel(r'$[\mu M]$')
    ax1.legend(loc='upper left')

    ax2 = plt.axes([0,0,0.5,0.5])
    ip = InsetPosition(ax1, [0.65,0.63,0.3,0.3])
    ax2.set_axes_locator(ip)
    ax2.plot(tv[:int(5e4)], V[:int(5e4)])
    ax2.legend('V', fontsize=6)
```