## Importing modules

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import gridspec

from pylab import rcParams

rcParams['figure.figsize'] = 8,8

from plotting import clcolor, kcolor, xcolor,nacolor,wcolor

## Constants, fixed parameters

### Constants

R=26.725\*1e-3 #E: R in this context is not the gas constant, it's RT/F

F=96485.0 # R (RT/F) in Volts, where F is Faraday's constant in C/mol, and T is 37 deg C

vw=0.018 # partial molar volume of water, dm3/mol

pw=0.0015 # osmotic permeability, biological membrane, dm s

km=6\*10\*\*(-7) # extensional rigidity of RBC at 23 deg, N/dm

km2=2.5\*10\*\*(1)

density=1.0 # kg/dm3 = g/ml --> assume close to 1 (density of water)

hp=1e-3

hydrop=0

qpump=6.13\*1e-5 #picoamperes

kd=15\*1e-3 #M Kd (Raimondo 2012)

vmax=5\*1e-3 #M/s Vmax (Raimondo 2012)

### Cell dimensions

rad=5\*1e-5 # radius in um convert to dm

rad0=rad

length=25\*1e-5 # length in um converted to dm

### Conductances

gna=2e-3/F #E: why are conductances/F?

gk=7e-3/F

gcl=2e-3/F # gna,gk,gcl: conductances in mS/cm^2 conv to S/dm^2

gkcc=2e-3/F # gkcc conductance

gamma=gna/gk

beta=1.0/(gk\*gcl+gkcc\*gk+gcl\*gkcc)

### Pump related variable

ck=2

cna=3 # cna,ck: pump (ATPase) stoichiometries

P=range(-70000,-38000)

default\_p=-1

default\_P=-40456 # P\_effective x10^5

### Concentrations

nao=145e-3

clo=119e-3

ko=3.5e-3 # nao,clo,ko: extracellular concentrations (mM converted to M)

z=-0.85 # intracellular (and extracellular) charge of impermeant anions

n=200 # points to plot

## Simulation

### Calling function

def plm(p=(10\*\*(default\_p))/(F),graph=0,pkcc=gkcc,gx=0,xt=100000,os\_init=ose,clinit=5.163e-3,toff=150000,ton=150000,tt=200,xinit=154.962e-3,two=0,xe=xe,f4d=0,ke=ke,n=1800,k\_init=122.873e-3,na\_init=14.002e-3,tk=100000,ratio=0.98,xend=120,osmofix=False,paratwo=False,moldelt=1e-13,xflux=0,z=z,dz=0,Zx=-1,ztarget=-100,length=length,areascale=1,rad=rad,title='fig.eps',neww=0,ls='-',a0=0,a1=0,a2=0,os\_choose=0,f1d=False,hamada=0,kccmodel=0,vmax=vmax,lin=0):

#### example of how plm is called in figure 1b

endcl=plm(clinit=init\_cl[i],tt=1800,osmofix=False,k\_init=0, hamada=0)

#### example of how plm is called in figure 1c

offpump=plm(graph=g,ton=3000,toff=9000,tt=12000,title='f1c.eps',neww=new,ls=l,a0=g1,a1=g2,a2=g3)

#### example of how plm is called in figure 1d

a=plm(p=q,tt=time,graph=0,k\_init=0,xinit=30e-3,clinit=120e-3,na\_init=140e-

### Create plotting arrays

Vm=[]

K=[]

Na=[]

Cl=[]

W=[]

X=[]

time=[]

Cl2=[]

X2=[]

Na2=[]

K2=[]

z\_delt=[]

xe\_delt=[]

gkcc\_delt=[]

kflux=[]

naflux=[]

clflux=[]

Xflux=[]

wflux=[]

### Creating timing variables

dt=1e-3 # zero time, dt time step

ts=tt/n # plotting timestep

ctr=1 # counter for plotting points

t=0 # real time

sw=1 # switch for ATPase action

### Alterations to volumes FinCAr

w=np.pi\*rad\*\*2\*length # initial volume in liters

sa=2\*np.pi\*rad\*(length)

#### Figure 1D – change starting volume

if f1d==True:

w=w\*154.962e-3/xinit # adjust for starting conditions in F1D (optimisation)

w1=w # initial volume stored for graphing later

Ar=2.0/rad # area constant (F and H method)

if areascale==0 or areascale==1:

Ar=sa/w #This is essentially the same

C=2e-4 # capacitance (F/dm^2)

FinvCAr=F/(C\*Ar) # (F/C\*area scaling constant)

sarest=sa # assuming this is initial SA stored for graphing later

### na,k,cl,x: intracellular starting concentrations

na=na\_init

x=xinit

#cl=((os\_init-na-k)\*z+na+k)/(1+z)

cl=clinit

#x=(cl-na-k)/z

k=k\_init

cle=clo

### Setting all deltas to 0

dw=0

dk=0

dcl=0

dna=0

dx=0

### 

### Alterations depending on whether osmolarity is fixed or not

if osmofix==True:

if xinit==0: #if there are no intracellular anions

x=(os\_init-2\*cl)/(1-z) #make the anion concentration set by this function

xinit=x #Change the starting impermeants to the new value

else:

cl=(os\_init+(z-1)\*x)/2.0 #If there are intracellular anions change the chloride rather.

print(cl)

if k\_init==0: #set the K if it is not given

k=cl-z\*x-na

print("k\_init: "+str(k))

print("osi: "+str(k+cl+x+na))

print("z\_aim: "+str(ztarget) +" with zflux of "+str(Zx))

### Dummy variables for z(charge of imp. Anions) and z(concentration of imp anions)

xm=x\*ratio #intracellular impermeants (154.962e-3) x 0.98

xtemp=x\*(1-ratio) #splitting the intracellular anions according to the ratio

zxm=z # vars are based on 0.85

zx=z # vars are based on 0.85

#### # for f1c --> slow change in ATPase rate

pdinit=-5.0 #new initial pump rate

pd=default\_p #pd = -1 = default\_p

em=(default\_p-pdinit)/(12.0\*10\*\*4)/8 #em= ½ \*12\*10^4

jeffconstant=p\*(na/nao)\*\*3 #Effective pumprate

### Related to anion flux

if two==1:

zx=Zx #Changing charge of anions

zxm=(z\*x-zx\*xtemp)/xm

if paratwo==True:

return (w\*x)

## Start of the simulation loop

**while t < tt:**

### Calculate voltage

V=FinvCAr\*(na+k-cl+z\*x) # voltage Calculation

### update arrays for plotting

if t>=(ctr-1)\*ts:

K.append(1000\*R\*np.log(ke/k)) #Ek Array

K2.append(1000\*k) #K array

Na.append(1000\*R\*np.log(nao/na)) #ENa array

Na2.append(1000\*na) #Na array

Cl.append(1000\*R\*np.log(cl/cle)) #ECl array

Cl2.append(cl\*1000.0) #Cl array

X.append(z\*1000\*R\*np.log(xe/x)) #Ex array

X2.append(1000\*(x)) #X array

W.append(w) #w array

Vm.append(1000\*V) #Vm array

time.append(t) #time arry

z\_delt.append(z) #z array (charge of impermeants)

xe\_delt.append(xe) #Extracellular x

gkcc\_delt.append(pkcc) #gKcc2 array

wflux.append(dw) #dw array

Xflux.append(dx) #dx arry

clflux.append(dcl) #dcl array

naflux.append(dna) #dna array

kflux.append(dk) #dk array

ctr+=1

### various conditional states

#### Figure 3: if the time is during the time of KCC2 changes

if tk+360>t>tk: #KCC2 changes start at time tk and stop at tk+360 (runs for 6 mins)

pkcc+=1e-12 # control switch for gkkc ramp (Fig 3)

vmax+=3.3e-7

#### Switch for anion flux

if dz!=0 and xt<t<xt+420 and xtemp>0 and xm>0:

xtemp+=dz #increase xtemp

xm-=dz # decrease xm

if two==1: # switch to recalculate average charge if needed

z=(zxm\*xm+zx\*xtemp)/(xm+xtemp)

#### Figure 4 changes

if f4d!=0:

if xt+400>t>xt:

xe+=f4d\*6e-5

cle-=f4d\*6e-5 # Figure 4D (balance the charge differences) --> can adjust the ratio at \* for interest

### 

### Alterations to pump rate

jp=p\*(na/nao)\*\*3 # cubic pump rate update (dependent on sodium gradient)

if hamada!=0: # hamada is a type of pump model

jp=qpump\*hamada\*(1.62/(1+(0.0067/na)\*\*3)+1.0/(1+(0.0676/na)\*\*3))/F

if lin!=0: # lin is a type of pump model

jp=p\*(na/nao)

#### # Figure 6

if neww==4 or neww==5:

jp=jeffconstant

if (toff>t) and (t>ton):

if pd>pdinit:

pd-=em

p=(10\*\*(pd))/F

elif t>toff:

if pd<default\_p:

pd+=em

p=(10\*\*(pd))/F # ATPase ramp

# kcc2

if kccmodel==1:

externals=cle\*ke/k

fix=np.log(externals/0.056)

jkcc2=-0.117\*np.log(externals/cl)/fix\*vmax\*cl/(kd+cl)/F #Raimondo

elif kccmodel==2:

jkcc2=51.55\*pkcc\*(ke\*cle-k\*cl) #Fraser and Huang

elif kccmodel==3:

jkcc2=0.011125\*0.3\*(ke\*cle-k\*cl)/(0.000054\*((1+ke\*cle/0.000054)\*(1+ke/0.009)\*(1+cle/0.006)+(1+k\*cl/0.000054)\*(1+k/0.009)\*(1+cl/0.006)))/F

else:

jkcc2=pkcc\*(K[ctr-2]-Cl[ctr-2])/1000.0 #Doyon

# ionic flux equations

dna=-dt\*Ar\*(gna\*(V-R\*np.log(nao/na))+cna\*jp\*sw)

dk=-dt\*Ar\*(gk\*(V-R\*np.log(ke/k))-ck\*jp\*sw-jkcc2)

dcl=dt\*Ar\*(gcl\*(V+R\*np.log(cle/cl))+jkcc2) #dna,dk,dcl: increase in intracellular ion conc during time step dt

dx=-dt\*Ar\*zx\*(gx\*(V-R/zx\*np.log(xe/(xtemp))))

na+=dna

k+=dk

cl+=dcl # increment concentrations

# anion flux switches

if xend==0 and (t>xt):

if (np.abs(x\*w-xinit\*w1)<moldelt) and (abs((np.abs(z)-np.abs(ztarget)))>0.001) and (min(z,zx)<=ztarget<=max(z,zx)):

if xflux==0:

xtemp+=dx

tt=t+180

else:

xtemp+=xflux

dx=xflux

tt=t+1000

else:

if (min(z,zx)<=ztarget<=max(z,zx)):

print('anions stopped diffusing at '+str(t))

xend=1

dx=0

else:

if xflux!=0 and tt<1000 and xtemp>0 and (min(zxm,zx)<=ztarget<=max(zxm,zx)):

xtemp-=xflux

tt=t+50

dx=-xflux

else:

print( 'anions stopped diffusing at '+str(t))

xend=1

dx=0

if xt+xend>t>xt:

if xflux!=0:

xtemp+=xflux/10

dx=xflux/10

else:

xtemp+=dx/10

dx=dx/10

elif xend!=0:

dx=0

elif t<xt:

dx=0

# update volume (usual method)

x=xm+xtemp

osi=na+k+cl+x # intracellular osmolarity

ose=nae+ke+cle+xe+xe1\*0.8

dw=dt\*(vw\*pw\*sa\*(osi-ose))

w2=w+dw

# other volume updates (incorporating hydrostatic pressure - various options considered)

if neww==1:

w2=w+dt\*(vw\*pw\*sa\*(osi-ose)+hp\*dt/density\*km\*(sarest-sa)/sarest)

elif neww==2:

w2=w+dt\*(vw\*pw\*sa\*(osi-ose-os\_choose))

elif neww==3 or neww==5:

hydrop=4.0\*km2\*np.pi\*(rad/rad0-1)/(R\*F)

w2=w+dt\*(vw\*pw\*sa\*(osi-ose-hydrop))

# correct ionic concentrations and surface area by volume change

na=(na\*w)/w2

k=(k\*w)/w2

cl=(cl\*w)/w2

x=(x\*w)/w2

xm=(xm\*w)/w2

xtemp=(xtemp\*w)/w2

w=w2

sa=2\*np.pi\*rad\*(length)

### # methods of updating Ar constant (dependent on how the surface area changes for volume, by radius or length)

if areascale==1:

rad=np.sqrt(w/(np.pi\*length))

Ar=sa/w

FinvCAr=F/(C\*Ar)

elif areascale==0:

length=w/(np.pi\*rad\*\*2)

t+=dt

### #plot if asked

if graph==1:

gs = gridspec.GridSpec(3, 1, height\_ratios=[1.5, 1, 1])

plt.figure()

a0=plt.subplot(gs[0])

a0.plot (time,Cl2,color=clcolor)

a0.plot(time,K2,color=kcolor)

a0.plot(time,X2,color=xcolor)

a0.plot(time,Na2,color=nacolor)

a1=plt.subplot(gs[1])

a1.plot(time,Vm,'k')

a1.plot (time,Cl,color=clcolor)

a1.plot(time,K,color=kcolor)

a2=plt.subplot(gs[2])

a2.plot(time,W,color=wcolor,label='relative volume')

#plt.savefig('kcl\_concs.eps')

plt.show()

if graph==2:

a0.plot (time,Cl2,color=clcolor,linestyle=ls)

a0.plot(time,K2,color=kcolor,linestyle=ls)

a0.plot(time,X2,color=xcolor,linestyle=ls)

a0.plot(time,Na2,color=nacolor,linestyle=ls)

a1.plot(time,Vm,'k')

a1.plot (time,Cl,color=clcolor,linestyle=ls)

a1.plot(time,K,color=kcolor,linestyle=ls)

a2.plot(time,W,color=wcolor,label='relative volume',linestyle=ls)

#plt.savefig(title)

plt.show()

print('na', na, 'k', k, 'cl', cl, 'x', x, 'vm', V, 'cle', cle, 'ose', ose, 'osi', osi, 'deltx', x\*w-xinit\*w1)

print('w', w, 'radius', rad, 'z', z)

print('ecl', Cl[-1])

return na, k, cl, x, V, Na[-1], K[-1], Cl[-1], X[-1], Vm[-1], W, time, Na, K, Cl, X, Vm, Cl2, Na2, K2, X2, w, z\_delt, xe\_delt, gkcc\_delt, a0, a1, a2, naflux, kflux, clflux, wflux, Xflux, np.log10(jp\*F), osi, ose

# FIGURE 1B

def f1b(init\_cl=[1e-3,15e-3,50e-3,90e-3],ham=0):

leg=[]

print("Figure 1B")

plt.figure()

for i in range(len(init\_cl)):

endcl=plm(clinit=init\_cl[i],tt=1800,osmofix=False,k\_init=0, hamada=ham)

plt.subplot(2,1,1)

plt.plot(endcl[11][13:-1],endcl[17][13:-1],color=clcolor,linestyle=sym[i])

plt.subplot(2,1,2)

plt.plot(endcl[11][13:-1],endcl[10][13:-1],'k'+sym[i])

leg.append(str(init\_cl[i]\*1000)+' mM')

plt.savefig('f1bham.eps')

plt.show()

return

# FIGURE 1C

def f1c(new=0,l='-',g1=0,g2=0,g3=0):

g=1

if new!=0:

g=2

print("\nFigure 1C")

offpump=plm(graph=g,ton=3000,toff=9000,tt=12000,title='f1c.eps',neww=new,ls=l,a0=g1,a1=g2,a2=g3)

return offpump

# FIGURE 1D

def f1d(time=25000,ham=0,l=0):

T=[-7000,-6000,-5000,-4500,-4000,-3500,-3000,-2000,-1000,0,1000,2000]

#T=[-2000,0,1000,2000]

ti=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

# numerical solutions

if ham==0:

for k in T:

q=10\*\*(k/1000.0)/F

# optimisations (alternative is to start sims at any values and allow each to run for a long time)

if k==-7000:

a=plm(p=q,tt=time,graph=0,k\_init=0,xinit=30e-3,clinit=120e-3,na\_init=140e-3,f1d=True,lin=l)

elif k==-6000:

a=plm(p=q,tt=time,graph=0,k\_init=0,xinit=36e-3,clinit=113e-3,na\_init=140e-3,f1d=True,lin=l)

else:

a=plm(p=q,tt=time,graph=0,k\_init=0,xinit=75e-3,clinit=75e-3,na\_init=135e-3,f1d=True,lin=l)

for i in range(25):

ti[i].append(a[i])

else:

L=[]

for a in T:

L.append((a-3500.0)/4.5) #y=(x-3500)/4.5

for k in T:

q=10\*\*(k/1000.0-default\_p)

# optimisations (alternative is to start sims at any values and allow each to run for a long time)

if k==-7000:

a=plm(p=q,tt=time,graph=1,k\_init=0,xinit=30e-3,clinit=120e-3,na\_init=140e-3,f1d=True,hamada=q)

elif k==-6000:

a=plm(p=q,tt=time,graph=1,k\_init=0,xinit=33e-3,clinit=118e-3,na\_init=142e-3,f1d=True,hamada=q)

elif -3000>k:

a=plm(p=q,tt=time,graph=1,k\_init=0,xinit=45e-3,clinit=105e-3,na\_init=140e-3,f1d=True,hamada=q)

elif k==-3000:

a=plm(p=q,tt=time,graph=1,k\_init=0,xinit=50e-3,clinit=100e-3,na\_init=140e-3,f1d=True,hamada=q)

elif k==-2000:

a=plm(p=q,tt=time,graph=1,k\_init=0,xinit=120e-3,clinit=35e-3,na\_init=135e-3,f1d=True,hamada=q)

else:

a=plm(p=q,tt=5000,graph=1,k\_init=0,xinit=75e-3,clinit=75e-3,na\_init=135e-3,f1d=True,hamada=q)

for i in range(25):

ti[i].append(a[i])

# parametric solutions

molinit=plm(gx=1e-8,xt=25,tt=100,two=1,paratwo=True,moldelt=0)

para=zplm(molinit=molinit)

# plotting

print( "\nFigure 1D")

gs = gridspec.GridSpec(3, 1, height\_ratios=[1.5, 1, 1])

plt.subplot(gs[0])

plt.plot(para[0],para[8],color=clcolor,linestyle='-')

plt.plot(para[0],para[7],color=kcolor,linestyle='-')

plt.plot(para[0],para[6],color=nacolor,linestyle='-')

plt.plot(para[0],para[9],color=xcolor,linestyle='-')

plt.plot(T,ti[0],'o--',color=nacolor)

plt.plot(T,ti[1],'o--',color=kcolor)

plt.plot(T,ti[2],'o--',color=clcolor)

plt.plot(T,ti[3],'o--',color=xcolor)

plt.subplot(gs[1])

plt.plot(para[0],para[10],'k-')

plt.plot(T,ti[4],'ko--')

plt.subplot(gs[2])

plt.plot(para[0],para[11],color=wcolor,linestyle='-')

plt.plot(T,ti[21],'ko--')

plt.savefig('f1d.eps')

plt.show()

return ti