# -\*- coding: utf-8 -\*-

"""

Created on Mon Aug 22 2016

Python 3.x targeted

@author: Chris Currin & Kira Dusterwald

"""

import numpy as np

import copy

from deferred\_update import UpdateType

from constants import F

from common import default\_radius, default\_length, \

clo, ko, nao, xo\_z, oso, \

default\_p, \

gk, gna, gcl, \

ck, cna, \

pw, vw, km, \

RTF, RT

from sim\_time import TimeMixin, Time

import simulator

import time

nan = float("nan")

class Compartment(TimeMixin):

"""

"""

def \_\_init\_\_(self, name, radius=default\_radius, length=default\_length,

nai=0.033, ki=0.1038, cli=0.0052,

z=-0.85, gx = 0e-9, pkcc2=2e-3/F, p=default\_p, stretch\_w=False):

self.unique\_id = str(time.time())

self.name = name

self.r = radius # in um

self.r1 = radius

self.L = length # in um

self.pkcc2 = pkcc2 # strength of kcc2

self.z = z # intracellular charge of impermeant anions

self.w = np.pi \* self.r \*\* 2 \* self.L # initial volume in liters

self.w2 = self.w # temporary volume parameter

self.sa = 2 \* np.pi \* self.r \* self.L

self.stretch\_w=stretch\_w

#self.Ar = 4e6

#self.Ar=2.0/self.r # area constant (F and H method)

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

self.Ar = self.sa / self.w

# (F/C\*area scaling constant)

self.FinvCAr = F / (self.C \* self.Ar)

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

self.nai = nai

self.ki = ki

if cli == 0:

# setting chloride that is osmo- and electro-neutral initially.

self.cli = (oso + (self.nai + self.ki) \* (1 / self.z - 1)) / (1 + self.z)

else:

self.cli = cli

if self.ki == 0:

self.xi = 155.858e-3

self.ki = self.cli-self.z\*self.xi-self.nai

else:

self.xi = (self.cli - self.ki - self.nai) / self.z

# default conductance of impermeant anions

self.gx = gx

if self.xi < 0 or self.cli < 0:

raise Exception("""Initial choice of either ki or nai resulted in negative concentration of

intracellular ion - choose different starting values.""")

# intracellular osmolarity

self.osi = self.nai + self.ki + self.cli + self.xi

if self.osi != oso:

print("Compartment {} not osmo-neutral".format(self.name))

self.nao = nao

self.ko = ko

self.clo = clo

# define step attributes for t=0

# define constant element of pump rate

self.p = p

# voltage

self.V = self.FinvCAr \* (self.nai + (self.ki + (self.z \* self.xi) - self.cli))

# pump rate

self.jp = self.p \* (self.nai / nao) \*\* 3

# kcc2

# self.jkcc2 = (gk \* self.pkcc2 \* (self.ki \* self.clo - self.ki \* self.cli)) # Fraser and Huang

self.ek = RTF \* np.log(self.ko / self.ki)

self.ecl = RTF \* np.log(self.cli / self.clo)

self.jkcc2 = self.pkcc2 \* (self.ek - self.ecl) # Doyon

# delta(anions of a fixed charge)

self.ratio = 0.98

self.xm = self.xi \* self.ratio

self.xi\_temp = self.xi \* (1 - self.ratio)

self.xmz = self.z

self.xz = self.z

self.absox = self.xi \* self.w

# ramp kcc2

self.jkccup = None

# for plotting of changes

self.dnai = 0

self.dki = 0

self.dcli = 0

self.dxi = 0

self.dz = 0

# register component with simulator

simulator.Simulator.get\_instance().register\_compartment(self)

def step(self, \_time: Time = None):

"""

perform a time step for the compartment

1) update voltage (V)

2) update cubic pump rate

3) update KCC2 flux rate

4) solve ionic flux equations for t+dt from t

5) increment ionic concentrations

6) update volume

7) correct ionic concentrations due to volume change

:param \_time: time object for reference

"""

if \_time is None:

raise ValueError("{} has no time object specified".format(self.\_\_class\_\_.\_\_name\_\_))

# update voltage

self.xi = self.xm + self.xi\_temp

self.V = self.FinvCAr \* (self.nai + self.ki - self.cli + self.z \* self.xi)

# update cubic pump rate (dependent on sodium gradient)

self.jp = self.p \* (self.nai / self.nao) \*\* 3

# kcc2

# self.jkcc2 = (gk \* self.pkcc2 \* (self.ki \* self.clo - self.ki \* self.cli)) # Fraser and Huang

if self.jkccup is not None:

self.pkcc2 += self.jkccup

self.jkcc2 = self.pkcc2 \* (self.ek - self.ecl) # Doyon

self.xz -= self.dz

self.z = (self.xmz \* self.xm + self.xz \* self.xi\_temp) / self.xi

# ionic flux equations

# dnai,dki,dcli,dxi: increase in intracellular ion conc during time step dt

dnai = -\_time.dt \* self.Ar \* (gna \* (self.V - RTF \* np.log(self.nao / self.nai)) + cna \* self.jp)

dki = -\_time.dt \* self.Ar \* (gk \* (self.V - RTF \* np.log(self.ko / self.ki)) - ck \* self.jp - self.jkcc2)

dcli = \_time.dt \* self.Ar \* (gcl \* (self.V + RTF \* np.log(self.clo / self.cli)) + self.jkcc2)

# dxi = -\_time.dt \* self.Ar \* self.xz \* (self.gx \* (self.V - RTF / self.xz \* np.log(xo\_z / self.xi\_temp)))

self.dnai = dnai

self.dki = dki

self.dcli = dcli

if self.gx != 0:

dxi = 6e-9\*self.Ar\*\_time.dt

else:

dxi = 0

self.ek = RTF \* np.log(self.ko / self.ki)

self.ecl = RTF \* np.log(self.cli / self.clo)

simulator.Simulator.get\_instance().to\_update\_multi(self, {

'nai': {

"value": dnai,

"type": UpdateType.CHANGE

}, 'ki': {

"value": dki,

"type": UpdateType.CHANGE

}, 'cli': {

"value": dcli,

"type": UpdateType.CHANGE

}, 'xi\_temp': {

"value": dxi,

"type": UpdateType.CHANGE

}

})

self.w2 = self.w + \_time.dt\*(vw\*pw\*self.sa\*(self.osi-oso))

if self.stretch\_w == True:

self.w2=self.w+\_time.dt\*(vw\*pw\*self.sa\*(self.osi-oso-4\*km\*np.pi\*(1-self.r1/self.r)/(RT)))

simulator.Simulator.get\_instance().to\_update(self, self.name, self.update\_values, UpdateType.FUNCTION)

def update\_values(self):

"""

Method for applying deferred update for multiple variables, specifically where there are multiple steps and

variables rely on other variables. This is done after variable (deferred) updates in step.

"""

# intracellular osmolarity

self.xi = self.xm + self.xi\_temp

self.osi = self.nai + self.ki + self.cli + self.xi

# update volume

#w2 = (self.w \* self.osi) / oso # update volume

# correct ionic concentrations by volume change

self.nai = (self.nai \* self.w) / self.w2

self.ki = (self.ki \* self.w) / self.w2

self.cli = (self.cli \* self.w) / self.w2

self.xi\_temp = (self.xi\_temp \* self.w) / self.w2

self.xm = (self.xm \* self.w) / self.w2

self.w = self.w2

# affect volume change into length change

self.update\_radius()

self.absox = self.xi \* self.w

def update\_radius(self):

self.r = np.sqrt(self.w / (np.pi \* self.L))

self.sa = 2 \* np.pi \* self.r \* (self.L)

self.Ar = self.sa / self.w

self.FinvCAr = F / (self.C \* self.Ar)

def update\_length(self):

self.L = self.w / (np.pi \* self.r \*\* 2)

def copy(self, name):

"""

Create a new Compartment identical to this one

:param name: identifier for the new compartment

:return: new Compartment

"""

comp = Compartment(name, radius=self.r, length=self.L, pkcc2=self.pkcc2, z=self.z, nai=self.nai, ki=self.ki,

cli=self.cli, p=self.p)

return comp

def deepcopy(self, name):

"""

Create a new Compartment identical to this one.

To be compared against copy() above

:param name: identifier for the new compartment

:return: new Compartment

"""

comp = copy.deepcopy(self)

comp.name = name

comp.unique\_id = str(time.time())

# register component with simulator

simulator.Simulator.get\_instance().register\_compartment(comp)

return comp

def mols(self,ion):

"""

:return: the mols for a concentration (cli etc) / flux (dcli etc)

"""

return ion\*self.w

def \_\_getitem\_\_(self, item):

return self.\_\_dict\_\_[item]

def \_\_setitem\_\_(self, key, value):

self.\_\_dict\_\_[key] = value

def \_\_str\_\_(self, \*args, \*\*kwargs):

return self.name

def \_\_repr\_\_(self):

return str(self.\_\_dict\_\_)

# return """Compartment({0},radius={1}, length={2}, pkcc2={3}, z={4}, nai={5}, ki={6},

# p={7}, cli={8})

# """.format(self.name,self.r,self.L,self.pkcc2,self.z,self.nai,self.ki,self.p,self.cli)