Brett Settle

CS 115 Computer Simulation

Diffusion Simulation

Question:
Do the calcium diffusion sites (puffs) provide satisfactory information about the calcium activity in a cell, or are there invisible, individual receptor sites in the cell that create the uniform calcium wave we see in diffusion?

Method:

I am basing my initial simulation parameters on the paper "Hindered cytoplasmic diffusion of inositol trisphosphate restricts its cellular range of action" by George D. Dickinson, Kyle L. Ellefsen, Silvina Ponce Dawson, John E. Pearson, and Ian Parker, Science Signaling Nov 2016
Paper available at: http://stke.sciencemag.org/content/9/453/ra108

Calcium ions diffuse from sites within the cell, and release IP3 ions as they do, creating a "puff" of luminescence that can be used to measure the strength and duration of calcium release. While most sites are large enough to measure and track, some hidden sites have been recognized throughout the cell, altering the visible diffusion of calcium. My model simulates this effect by randomly generating sites with fractional amplitudes in between normal sites and comparing the resulting wave.

Fig 1. Puff Fluorescence Trace at Opening

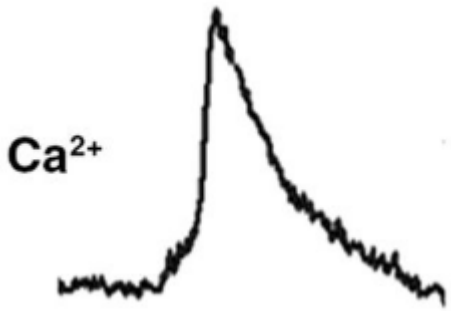
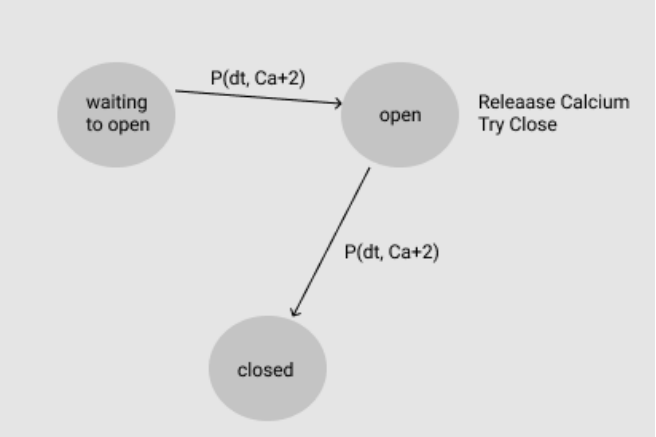


Fig 2. Model of Puff Entities:



Simulation

My simulation would generate a cell (as a rectangle represented by a pixel mesh grid) with uniform IP3 distribution, and randomly generate N puff sites across the cell. The activation of puffs can be modeled by an exponential distribution of time and calcium, where most puffs excite within the first hundred milliseconds. Puff sites open and generate uniform calcium at every time step, then close by some exponential distribution. Puffs can open very briefly, or remain open for up to 300 milliseconds. Best results were achieved using a Poisson distribution with mean of ~100 ms. As the calcium from the puff diffuses, it increases the concentration within the cell, increasing the likelihood of other sites becoming excited. This chain reaction of events has been seen to create a uniform "wave" effect.

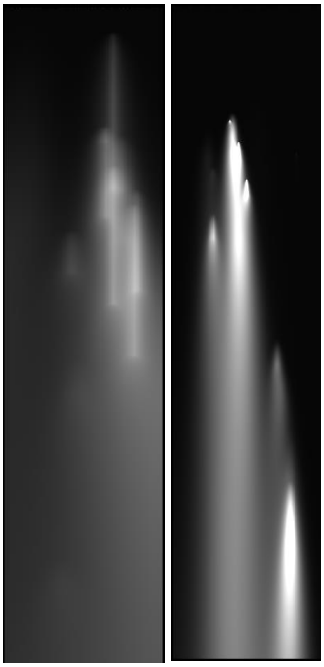
Model:

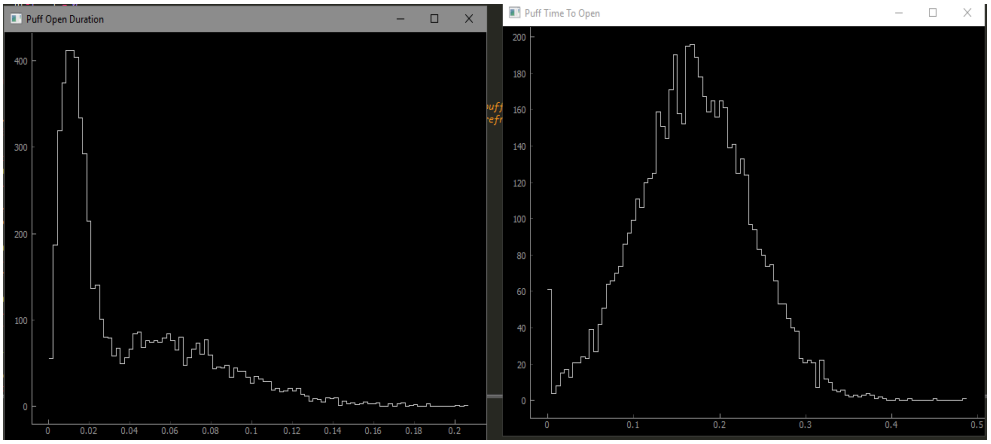
Parameters:

- Cell shape: default to rectangle [width, height]
- Iterations: Number of iteration intervals to simulate
- Number of Clusters: Cluster count within cell, uniformly distributed as X,Y coordinates
- Diffusion coefficient: the coefficient to use in the diffusion equation
- Distribution for initial calcium levels throughout the cell, as well as calcium levels in clusters
- Number of hidden clusters to create within the cell, with default amplitude .1

Results:

Looking at plots from dozens of simulations, the puff open duration histogram is a rough exponential distribution and the time to open histogram follows a single spike in the middle. The kymographs are plotted via a diagonal drawn across the movie. The left kymograph shows a wave without hidden sites, while the right kymograph includes hidden sites and shows a much more controlled increase of calcium over time.





The parameters required a lot of toying to perfect a realistic chain reaction effect. Because puffs open by some exponential distribution, but also are set off early by spikes in calcium, I test the probability at each time interval, as opposed to deciding it on creation of the Puff. Initial calcium in the cell can be set to .05 or 1, so some normalization was required to maintain an increasing probability as time progressed. Even after much toying, the "wave" effect was a rare result.

Conclusion

The difference between the resulting movies (and "waves") is not significant enough to draw any specific conclusion from the simulation. With the miniature "hidden" sites present in the simulation, dispersed between regular receptors, the calcium increase wave appears to be slightly more uniform. The difficulty in measuring how uniform the wave is makes it impossible to quantify how accurate the hidden sites are.

About Code

Developed in python with numpy backend. GUI available with PyQt and pyqtgraph.

- Diffusion.py: main program to simulate multiple movies at once and output puff statistics. Arguments are specified by the command line interface, and default parameters are set within the file
- Puff.py: Puff class for monitoring puff activity and tracking puff entities
- Gen.py: Handles stored models and the actual diffusion equation/simulation process.
- Main.py: GUI interface for simulation, allows for more customization of simulation parameters and interactive visualization post-completion.

Code

Diffusion.py

```
import numpy as np
import sys, os
import argparse
from gen import Model, models
from tqdm import *
from glob import glob

def parseArgs():
	parser = argparse.ArgumentParser()
	args = sys.argv[1:]
	parser.add_argument("-width", type=int, help="width in microns")
	parser.add_argument("-height", type=int, help="height in microns")
	parser.add_argument("-dx", type=int, help="mesh width")
	parser.add_argument("-dy", type=int, help="mesh height")
	parser.add_argument("-t", type=int, help="total time to simulate")
	parser.add_argument("-dt", type=int, default=None, help="time interval in microseconds")
	parser.add_argument("-s", type=float, default=1.0, help="sequestration coefficient")
	parser.add_argument('-p', type=int, default=10, help="Sites to generate")
	parser.add_argument("-d", type=float, help="Diffusion Coefficient in square microns per second")
	parser.add_argument("-r", type=int, help="Refresh rate for result movie. Saves every r frames")
	parser.add_argument('-o', type=str, default="out/results.txt",help="output file to save results to")
	parser.add_argument('-n', type=int, default=1, help="Number of simulations to run")
	parser.add_argument('--stop-early', action="store_true", default=False, help="Stop once all puffs are closed")
	parser.add_argument("--hidden", type=int, default=0, help="Hidden sites to generate")

	args = dict(parser.parse_args()._get_kwargs())

	args['x_max'] = args.pop('width')
	args['y_max'] = args.pop('height')
	args['t_max'] = args.pop('t')
	args['sequestration'] = args.pop('s')
	args['puffs'] = args.pop('p')
	args['refresh'] = args.pop('r')
	args['hidden_puffs'] = args.pop('hidden')

	return {k:v for k, v in args.items() if v is not None}

if __name__ == '__main__':
	args = parseArgs()
	n = args.pop('n')

	##### SETTINGS ###
	n = 30
	args['t_max'] = 1000
	args['puffs'] = 30
	args['hidden_puffs'] = 10
	args['stop_early'] = True
```

```
#####

fname = str(args.pop('o'))
if not os.path.exists(os.path.dirname(fname)):
        os.mkdir(os.path.dirname(fname))
if os.path.exists(fname):
        os.remove(fname)

for f in glob('out/*'):
        os.remove(f)

m = Model(**args)
for i in tqdm(range(n)):
        m.genPuffs(len(m.puffs))
        #np.random.choice(m.puffs).open = True
        #frames = (m.nt) // m.refresh + 1
        frames = 1
        movie = np.zeros([frames, m.nx+2, m.ny+2])

        movie[0, 1:-1, 1:-1] = np.random.random([m.nx, m.ny]) * .05
        j = 1
        p = 0
        for im in m.run(movie[0]):
                if j % m.refresh == 0:
                        movie[0] = im
                        #movie[i // m.refresh] = im

                if (100 * j) // m.nt > p:
                        p = (100 * j) // m.nt
                j += 1

        m.export(open(str(fname), 'wb' if i == 0 else 'ab'))
        a = []
        for p in m.puffs:
                a.append(p.concentrations)

        arr = np.array(a)
        np.savetxt(open('out/puffs_%d.txt' % i, 'wb'), arr)
```

Gen.py

```
import numpy as np
from puff import Puff

AMPLITUDE = 1
HIDDEN_AMPLITUDE = .1


data = [[260.916,   63.486],
[268.831, 324.685],[276.746, 421.646],[340.068, 448.359], [303.46, 508.712], [448.901, 546.309], [325.227,  635.354], [359.855, 714.506], [303.46,    832.243], [370.739,   812.455], [411.304,   734.293], [529.041,   735.283], [571.585,   616.556], [563.67, 458.253], [561.691,   393.943], [529.041,   300.940], [390.527,   247.513],  [419.219,   150.552]]

data = np.array(data) // 4 - [20, 0]

def uniform(points, size):
   xs = np.random.uniform(0.15*size[0], size[0]*.85, points)
   ys = np.random.uniform(0.15*size[1], size[1]*.85, points)
   return np.transpose([xs, ys]).astype(int)
```

```
class Model():

    def __init__(self, **kargs):
        self.d = kargs.get('d', 20)
        self.dt = kargs.get('dt', .05)
        self.dx = kargs.get('dx', 166)
        self.dy = kargs.get('dy', 166)
        self.t_max = kargs.get('t_max', 1000)
        self.x_max = kargs.get('x_max', 40000)
        self.y_max = kargs.get('y_max', 40000)
        self.sequestration = kargs.get('sequestration', 1.0)
        self.stop_early = kargs.get('stop_early', False)
        self.refresh = kargs.get('refresh', 50)

        maxDt = self.dx**2*self.dy**2/( 2*self.d*(self.dx**2+self.dy**2) )
        if self.dt > maxDt:
            print("ALERT: time step too large for mesh, setting to %s" % maxDt)
            self.dt = maxDt

        self.nx = len(np.arange(0, self.x_max+self.dx, self.dx))
        self.ny = len(np.arange(0, self.y_max+self.dy, self.dy))
        self.nt = len(np.arange(0, self.t_max+self.dt, self.dt))

        puffs = kargs.get('puffs', 30)
        if isinstance(puffs, int):
            self.genPuffs(puffs, amplitude=AMPLITUDE)
        else:
            self.puffs = puffs
            self.puffCount = len(puffs)

        hidden_puffs = kargs.get('hidden_puffs', 0)
        self.genPuffs(hidden_puffs, amplitude=HIDDEN_AMPLITUDE, reset=False)

    def finished(self):
        return all([p.finished() for p in self.puffs])

    def genPuffs(self, n, amplitude=AMPLITUDE, reset=True):
        puffs = [Puff(a, b, amplitude) for a, b in uniform(n, [self.nx, self.ny])]
        if reset:
            self.puffs = puffs
        else:
            self.puffs.extend(puffs)

        self.puffCount = len(self.puffs)

    def export(self, fname):
        # x y amplitude openDuration
        d = np.zeros([len(self.puffs), 5])
        for i, p in enumerate(self.puffs):
            d[i] = [p.x, p.y, p.amplitude, p.timeToOpen, p.openDuration]

        np.savetxt(fname, d)

    def handlePuffs(self, dt):
        for p in self.puffs:
            #if p.finished():
            #    continue
```

```
            valAtT = self.u[p.x, p.y]
            v = p.update(dt, valAtT)
            self.u[p.x, p.y] += v

    def run(self, im):

        #convert units to micron

        d = self.d  * 1e-6 #/ (1e-3) # micron ** 2 per second
        dt = self.dt * 1e-3
        dx = self.dx * 1e-6
        dy = self.dy * 1e-6
        t_max = self.t_max * 1e-3
        x_max = self.x_max * 1e-6
        y_max = self.y_max * 1e-6

        s = d*dt/(dy*dx)

        x = np.arange(0,x_max+dx,dx)
        y = np.arange(0,y_max+dy,dy)
        t = np.arange(0,t_max+dt,dt)
        r = len(t)
        c = len(y)
        d = len(x)
        self.u = im.copy()
        for n in range(0,r-1): # time
            u = self.u[1:-1, 1:-1] + s * (self.u[2:, 1:-1] - 4*self.u[1:-1, 1:-1] + self.u[:-2, 1:-1] + self.u[1:-1, 2:] + self.u[1:-1, :-2])
            u *= self.sequestration
            u[:, 0] = u[:, 1]
            u[:, -1] = u[:, -2]
            u[0, :] = u[1, :]
            u[-1, :] = u[-2, :]

            self.u[1:-1, 1:-1] = u.copy()
            self.handlePuffs(self.dt)
            yield self.u
            if self.stop_early and self.finished():
                break

d = 20 # um**2/s

models = {'Science Signaling': Model(d=d, dt=.01, dx=50, dy=50, t_max=1000, x_max=8000, y_max=12000, puffs=[Puff(*p) for p
in data], refresh=100),
      'No Hidden Sites': Model(puffs=30),
      'Hidden Sites': Model(puffs=30, hidden_puffs=20),
      }

import pickle, os
def load_models():
    global models
    if os.path.exists('models.p'):
        newModels = pickle.load(open('models.p', 'rb'))
        models.update(newModels)

def save_models():
    modelsNew = {k:v for k, v in models.items() if k not in ('Science Sampling', 'Sample')}
    pickle.dump(models, open('models.p', 'wb'))
```

```
def save_model(name, model):
  global models
  models[name] = model

load_models()

def showMovie(mov):
  import pyqtgraph as pg
  app = pg.Qt.QtGui.QApplication([])
  im = pg.ImageView()
  im.setImage(mov)
  im.show()
  return app, im
```

Puff.py

```
import numpy as np
import random

OPEN_DURATION = 100

class Puff:
	OPEN_RATE = 5
	def __init__(self, x, y, amplitude=1):
		self.x = int(x)
		self.y = int(y)
		self.amplitude = amplitude
		self.open = False
		self.openDuration = 0.
		self.timeToOpen = 0.
		self.concentrations = []
		self.pToggle = Puff.OPEN_RATE
		duration = min(1, amplitude) * OPEN_DURATION
		self.closeTime = min(300, random.expovariate(1/duration))

	def finished(self):
		return self.open == False and self.openDuration > 0

	def tryToggleOpen(self, dt, concentration):
		if np.random.random() < (np.exp(1 + 20 * concentration) * self.pToggle * 1e-5 * dt):
			self.open = not self.open

	def update(self, dt, concentration):
		self.concentrations.append(concentration)
		val = 0
		if self.open:
			self.openDuration += dt
			val = 5 * self.amplitude * dt
			if self.openDuration >= self.closeTime:
				self.open = False
			#self.tryToggleOpen(dt, concentration)
		elif self.openDuration == 0:
			self.tryToggleOpen(dt, concentration)
			if not self.open:
				self.timeToOpen += dt

		return val
```

Main.py

```
import random
```

```
import numpy as np
import threading
from qtpy import QtWidgets, QtCore
import pyqtgraph as pg
from pyqtgraph.console import ConsoleWidget
from gen import models, uniform, save_model, Model, save_models, HIDDEN_AMPLITUDE, AMPLITUDE
from puff import Puff


class PuffTable(pg.TableWidget):
    def __init__(self):
        pg.TableWidget.__init__(self)
        self.setEditable(True)
        self.cellChanged.connect(self.changedCell)
        self.puffs = []

    def changedCell(self, a):
        if self.item(a, 0) is None or self.item(a, 1) is None or self.item(a, 2) is None:
            return
        self.puffs[a].x = self.item(a, 0).value
        self.puffs[a].y = self.item(a, 1).value
        self.puffs[a].amplitude = self.item(a, 2).value

    def addPuff(self, puff):
        self.puffs.append(puff)
        self.updateTable()

    def updateTable(self):
        data = [[p.x, p.y, p.amplitude] for p in self.puffs]
        self.setData(data)
        self.setHorizontalHeaderLabels(["X", "Y", "Amplitude"])

    def setPuffs(self, puffs):
        self.removeAll()
        self.puffs = puffs
        self.updateTable()

    def getPuffs(self):
        return self.puffs

    def removeAll(self):
        self.puffs = []
        self.setData([])

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        centralWidget = QtWidgets.QWidget()
        self.setCentralWidget(centralWidget)
        self.running = False

        self._makeMenuBar()

        self.imageview = pg.ImageView()
        self.imageview.view.setAutoPan(False)
        self.imageview.setMinimumWidth(650)
        self.console = ConsoleWidget()
        self.console.setMaximumWidth(400)
```

```
self.optionsWidget = QtWidgets.QWidget()

def errCall(err, a):
    self.running = False
    self.imageview.setHistogramRange(0, 2)
    self.imageview.setLevels(0, 2)
np.seterrcall(errCall)
np.seterr(over='call')

opsL = QtWidgets.QFormLayout()
header = QtWidgets.QLabel("Diffusion Simulation")
opsL.addRow(header)
self.optionsWidget.setMaximumWidth(600)
self.optionsWidget.setMinimumWidth(400)
self.top_left_label = pg.LabelItem("", justify='right')
self.imageview.ui.graphicsView.addItem(self.top_left_label)

def timeChanged(m, b):
    self.top_left_label.setText("Frame %d, %s ms" % (m, m * self.dtSpin.value() * self.refreshSpin.value()))

self.imageview.sigTimeChanged.connect(timeChanged)
def viewKeyPress(ev):
    pg.ViewBox.keyPressEvent(self.imageview.view, ev)

self.imageview.view.keyPressEvent = viewKeyPress

def saveModel():
    puffs = self.puffs
    model = self.getModel()
    model.puffs = puffs
    name = QtWidgets.QInputDialog.getText(self, "Enter a model name", "Enter a model name")
    if name is not None and len(name) > 0:
        save_model(name, m)

self.startButton = QtWidgets.QPushButton("Start")
self.startButton.pressed.connect(self.start)
self.saveModelButton = QtWidgets.QPushButton("Save Model")
self.saveModelButton.pressed.connect(saveModel)
self.progressBar = QtWidgets.QProgressBar()
self.progressBar.setRange(0, 100)
self.progressBar.setValue(0)
buttonBox = QtWidgets.QWidget()
buttonLayout = QtWidgets.QGridLayout()
buttonLayout.addWidget(self.progressBar, 0, 0, 1, 3)
buttonLayout.addWidget(self.saveModelButton, 1, 0)
buttonLayout.addWidget(self.startButton, 1, 1)
buttonBox.setLayout(buttonLayout)

self.imageWidget = self._makeImageWidget()
self.modelWidget = self._makeModelWidget()
self.puffWidget = self._makePuffWidget()

separator = QtWidgets.QWidget()
sepLayout = QtWidgets.QHBoxLayout()
self.infoLabel = QtWidgets.QLabel()
sepLayout.addWidget(self.infoLabel)
sepLayout.addStretch()
separator.setLayout(sepLayout)
```

```
    def modelSelected():
        data = combo.currentData()
        self.widthSpin.setValue(data.x_max)
        self.heightSpin.setValue(data.y_max)
        self.timeSpin.setValue(data.t_max)
        self.dxSpin.setValue(data.dx)
        self.dySpin.setValue(data.dy)
        self.dtSpin.setValue(data.dt)
        self.dSpin.setValue(data.d)

        self.sequesterSpin.setValue(data.sequestration)
        puffs = data.puffs
        self.pointSpin.setValue(len(puffs))
        self.puffTable.setPuffs(puffs)
        self.refreshSpin.setValue(data.refresh)

    combo = QtWidgets.QComboBox()
    combo.currentIndexChanged.connect(modelSelected)

    for a, b in models.items():
        combo.addItem(a, b)
    opsL.addRow("Saved Models", combo)
    opsL.addRow(self.imageWidget)
    opsL.addRow(self.puffWidget)
    opsL.addRow(self.modelWidget)
    opsL.addRow(separator)
    opsL.addRow(buttonBox)
    self.optionsWidget.setLayout(opsL)

    self.console.localNamespace.update({'self': self, 'set': self.imageview.setImage})
    layout = QtWidgets.QGridLayout()
    layout.addWidget(self.console, 0, 0)
    layout.addWidget(self.imageview, 0, 1)
    layout.addWidget(self.optionsWidget, 0, 2)
    centralWidget.setLayout(layout)
    self.resize(1850, 600)

    self.mouse = [0, 0]

    self.imageview.scene.sigMouseMoved.connect(self.mouseMoved)
    self.imageview.mousePressEvent = self.mousePressed

    self.puffs = []

    self.scatter = pg.ScatterPlotItem(brush=pg.mkBrush(255, 255, 0))
    self.scatter.sigClicked.connect(self.mousePressed)

    self.imageview.view.addItem(self.scatter)
    self.imageview.view.setMenuEnabled(False)

def _makeMenuBar(self):
    m = self.menuBar()
    fileMenu = m.addMenu("File")
    self.saveResultsAction = fileMenu.addAction("Save Results", self.saveResults)
    fileMenu.addAction("Save Movie", lambda : np.savetxt("movie.txt", self.imageview.image))
    plotMenu = m.addMenu("Plot")
```

```
        def plotTTO():
            vals = [p.timeToOpen for p in self.puffTable.puffs]
            a, b = np.histogram(vals, 50)
            pg.plot(x=b, y=a, stepMode=True, title="Puff Time To Open (ms)")

        def plotOD():
            vals = [p.openDuration for p in self.puffTable.puffs]
            a, b = np.histogram(vals, 50)
            pg.plot(x=b, y=a, stepMode=True, title="Puff Open Duration (ms)")

        def plotPuffs():
            self.plotItem = pg.PlotWidget()
            for p in self.puffs:
                self.plotItem.addItem(pg.PlotDataItem(p.concentrations))
            self.plotItem.show()


        plotMenu.addAction("Plot Time To Open", plotTTO)
        plotMenu.addAction("Plot Open Duration", plotOD)
        plotMenu.addAction("Plot Puffs", plotPuffs)


        m.addAction("Quit", self.close)

    def saveResults(self):
        fname = QtWidgets.QFileDialog.getSaveFileName(self, "Save model results")
        fname = str(fname if type(fname) != tuple else fname[0])
        if fname != '':
            self.model.export(fname)

    def getModel(self):
        dt = self.dtSpin.value()
        dx = self.dxSpin.value()
        dy = self.dySpin.value()
        x_max = self.widthSpin.value()
        y_max = self.heightSpin.value()
        t_max = self.timeSpin.value()
        puffs = self.puffTable.puffs
        sequestration = self.sequesterSpin.value()
        d = self.dSpin.value()
        refresh = self.refreshSpin.value()
        stop_early = self.stopEarlyCheck.isChecked()
        return Model(d=d, dt=dt, dx=dx, dy=dy, t_max=t_max, x_max=x_max, y_max=y_max, puffs=puffs,
sequestration=sequestration, refresh=refresh, stop_early=stop_early)

    def mouseMoved(self, point):
        mouse = self.imageview.getImageItem().mapFromScene(point)
        self.mouse = [int(mouse.x()), int(mouse.y())]

    def mousePressed(self, ev):

        if isinstance(ev, pg.ScatterPlotItem):
            pt = ev.ptsClicked[0]
            for p in self.puffs:
                if p.x == int(pt.pos().x()) and p.y == int(pt.pos().y()):
                    p.open = not p.open

        elif ev.button() == 2:
```

```
            if hasattr(self, 'selectedPoint'):
                print(self.selectedPoint)
            puffs = 0
            for p in self.puffs:
                d = np.linalg.norm(np.subtract(self.mouse, [p.x, p.y]))
                if puffs == 0 and d < 1:
                    return
            self.puffTable.addPuff(Puff(self.mouse[0], self.mouse[1]))
        self.generate(image=False)

    def closeEvent(self, ev):
        print("Saving models...")
        save_models()

    def _makeImageWidget(self):
        w = QtWidgets.QGroupBox("Model Settings")
        layout = QtWidgets.QFormLayout()
        self.widthSpin = QtWidgets.QSpinBox()
        self.widthSpin.setRange(0, 50000)
        self.widthSpin.setValue(10000)
        self.heightSpin = QtWidgets.QSpinBox()
        self.heightSpin.setRange(0, 50000)
        self.heightSpin.setValue(10000)
        self.timeSpin = QtWidgets.QDoubleSpinBox()
        self.timeSpin.setRange(0, 10000)
        self.timeSpin.setDecimals(2)
        self.timeSpin.setValue(2000)

        self.dxSpin = QtWidgets.QSpinBox()
        self.dxSpin.setRange(0, 1000)
        self.dxSpin.setValue(100)
        self.dySpin = QtWidgets.QSpinBox()
        self.dySpin.setRange(0, 1000)
        self.dySpin.setValue(100)
        self.dtSpin = QtWidgets.QDoubleSpinBox()
        self.dtSpin.setDecimals(4)
        self.dtSpin.setValue(.01)

        imageGenStyle = QtWidgets.QWidget()
        l = QtWidgets.QHBoxLayout()
        self.randomRadio = QtWidgets.QRadioButton("Random [0, .05)")
        self.randomRadio.setChecked(True)
        self.onesRadio = QtWidgets.QRadioButton("Ones")
        l.addWidget(self.randomRadio)
        l.addWidget(self.onesRadio)
        imageGenStyle.setLayout(l)

        layout.addRow("Width (micron)", self.widthSpin)
        layout.addRow("Height (micron)", self.heightSpin)
        layout.addRow("Time (ms)", self.timeSpin)
        layout.addRow("X Grid Size (micron)", self.dxSpin)
        layout.addRow("Y Grid Size (micron)", self.dySpin)
        layout.addRow("Time Step (ms)", self.dtSpin)

        layout.addRow(imageGenStyle)
        w.setLayout(layout)
        return w
```

```
def _makePuffWidget(self):
    w = QtWidgets.QGroupBox("Puff Settings")
    layout = QtWidgets.QFormLayout()
    self.pointSpin = QtWidgets.QSpinBox()
    self.pointSpin.setRange(1, 1000)
    self.puffTable = PuffTable()

    self.minPuffSpin = QtWidgets.QSpinBox()
    self.minPuffSpin.setValue(0)

    def generatePuffs():
        m = self.getModel()
        m.genPuffs(self.pointSpin.value(), amplitude=AMPLITUDE)

        m.genPuffs(self.minPuffSpin.value(), amplitude=HIDDEN_AMPLITUDE, reset=False)

        self.puffTable.setPuffs(m.puffs)

    genPointsButton = QtWidgets.QPushButton("Generate Points")
    genPointsButton.pressed.connect(generatePuffs)
    generateButton = QtWidgets.QPushButton("Generate")
    generateButton.pressed.connect(self.generate)
    buttonBox = QtWidgets.QWidget()
    buttonLayout = QtWidgets.QHBoxLayout()
    buttonLayout.addWidget(genPointsButton)
    buttonLayout.addWidget(generateButton)
    buttonBox.setLayout(buttonLayout)

    layout.addRow("Puff Sites", self.pointSpin)
    layout.addRow("Hidden Sites", self.minPuffSpin)
    layout.addRow(self.puffTable)
    layout.addRow(buttonBox)
    w.setLayout(layout)
    return w

def _makeModelWidget(self):
    widg = QtWidgets.QGroupBox("Model Settings")
    layout = QtWidgets.QFormLayout()

    self.dSpin = QtWidgets.QDoubleSpinBox()
    self.dSpin.setRange(0, 1000)
    self.dSpin.setValue(20)
    self.sequesterSpin = QtWidgets.QDoubleSpinBox()
    self.sequesterSpin.setRange(0, 1)
    self.sequesterSpin.setDecimals(4)
    self.sequesterSpin.setValue(.95)
    self.refreshSpin = QtWidgets.QSpinBox()
    self.refreshSpin.setRange(1, 1000)
    self.refreshSpin.setValue(100)
    self.stopEarlyCheck = QtWidgets.QCheckBox()
    self.stopEarlyCheck.setChecked(True)

    layout.addRow("Diffusion Coefficient (micron**2 / s)", self.dSpin)
    layout.addRow("Sequestration Coefficient", self.sequesterSpin)
    layout.addRow("Interval Refresh Rate", self.refreshSpin)
    layout.addRow("Quit when all puffs close", self.stopEarlyCheck)
    widg.setLayout(layout)
    return widg
```

```
def generate(self, image=True):
    m = self.getModel()

    if image:
        self.Z = np.zeros((m.nx+2,m.ny+2), dtype=np.float64)
        z = self.Z[1:-1,1:-1]
        if self.onesRadio.isChecked():
            z += 1
        elif self.randomRadio.isChecked():
            z += 0.05*np.random.random((m.nx,m.ny))

    self.puffs = []
    points = []
    sizes = []
    colors = []
    for puff in self.puffTable.puffs:
        if image:
            puff.openDuration = 0
        points.append([puff.x+.5, puff.y+.5])
        sizes.append(puff.amplitude)
        colors.append((255, 255, 0) if not puff.open else (255, 0, 0))

    pens = [pg.mkPen(pen) for pen in colors]
    brushes = [pg.mkBrush(brush) for brush in colors]
    self.puffs = self.puffTable.puffs

    if len(self.puffs) > 0:
        sizes = 5 + 4 * (np.array(sizes) / np.max(sizes))
        self.scatter.setPoints(pos=points, size=sizes, pen=pens, brush=brushes)
    if image:
        self.showImage(self.Z)

def start(self):
    if not self.running and self.startButton.text() == 'Stop':
        self.startButton.setText("Start")
        return
    if self.running:
        self.running = False
        return

    self.model = self.getModel()
    m = self.model
    refreshRate = self.refreshSpin.value()
    frames = (m.nt) // refreshRate + 1
    movie = np.zeros([frames, m.nx+2, m.ny+2])

    movie[0] = self.imageview.getImageItem().image
    i = 1
    p = 0
    self.startButton.setText("Stop")
    self.running = True
    for im in m.run(movie[0]):
        if not self.running:
            break
        if i % refreshRate == 0:
            movie[i // refreshRate] = im
```

```
            if (100 * i) // m.nt > p:
                p = (100 * i) // m.nt
                self.progressBar.setValue(p)
            QtWidgets.qApp.processEvents()
            i += 1

        self.progressBar.setValue(100)
        self.showImage(movie)

        self.running = False
        self.startButton.setText("Start")
        self.saveResultsAction.setEnabled(True)

    def keyPressEvent(self, ev):
        if ev.text() == 'c':
            from pyqtgraph.console import ConsoleWidget
            self.console = ConsoleWidget()
            self.console.localNamespace['self'] = self
            self.console.show()

    def closeEvent(self, ev):
        QtWidgets.QWidget.closeEvent(self, ev)
        self.running = False

    def showImage(self, V, **kargs):
        self.imageview.setImage(V, autoLevels=False, **kargs)
        self.imageview.setLevels(0, min(1.5, 1.5 * V.max()))
        self.imageview.setHistogramRange(0, 2)

    def openPuff(self, puff, clicked=False):
        if clicked and self.running:
            return
        puff.open = not puff.open

if __name__ == '__main__':
  app = QtWidgets.QApplication([])
  mw = MainWindow()
  mw.show()
  app.exec_()
```

Output

```
Puff opens at 48, will remain open for 40
Puff opens at 51, will remain open for 42
Puff opens at 67, will remain open for 26
Puff closes at 89
Puff closes at 93
Puff closes at 94
Puff closes at 96
Puff opens at 257, will remain open for 65
Puff opens at 267, will remain open for 245
Puff opens at 289, will remain open for 31
Puff opens at 312, will remain open for 178
```