```
#!/usr/bin/env python
# coding: utf-8

# The principle behind the confocal image simulation - FRAP simulation
# This program intends to simulate the scenario immediately after the
bleaching event, where a whole nucleus is photobleached.

#_____________________________________________________________________
# Importing Packages.

import numpy as np

#_____________________________________________________________________
# Image properties.

image_size = 40 # px (The image will be square 40 x 40 pixels)
pixel_size = 0.2 # µm/pixel
boundary = image_size * pixel_size
radius = 3 ; # µm
dwell_time = 0.001 # s
psf_width = 0.3 # µm (Width of the point spread function in focus)
psf_height = 1.5 #
diff_const = 0.1 # µm^2/s (diffusion coefficient of mobile particles)
step_time = 0.001 # s
B = 1e4 # Brightness, Hz

Nparticles = 2000

center_pos = [4, 4,4] # the centre of the sphere

#_____________________________________________________________________
# Generate initial positions of particles, which are outside of
nucleus.

start_pos = np.zeros((Nparticles,3))
for n in range(Nparticles):
    temp = start_pos[n,:]
    while temp[0]**2 + temp[1]**2 + temp[2]**2 == 0:
        x = np.random.rand(3) * boundary
        if ((x[0] - 4)**2 + (x[1] - 4)**2 + (x[2] - 4)**2) > radius**2
and (x[0]**2 + x[1]**2 + x[2]**2) > 0:
            start_pos[n,:] = x

#_____________________________________________________________________
# Calculating the pixel intensity.
# The pixel intensity is dependent on the distance from the optical
axis.

def GaussianBeam( start_pos, beam_pos, psf_width, psf_height):
    if start_pos.shape[0] == 2:
```

```
        GB = B*step_time*np.exp(- 2* ((start_pos -
beam_pos)**2).sum()/ psf_width**2)
    else:
        GB = B*step_time*np.exp(- 2* ((start_pos[0:2] -
beam_pos[0:2])**2).sum()/ psf_width**2) * np.exp(-2*((start_pos[2]-
beam_pos[2])**2/psf_height**2))

    return GB

#_____________________________________________________________________
# More parameters for the movement of particles.

pout = 0.01 #  flow rate from nucleus to cytoplasm

pin = 0.04 # flow rate from cytoplasm to nucleus

steps = 60000 # number of steps in the simulation

pre_pos = np.zeros((steps+1,3,Nparticles)) # a 3D matrix storing the
previous position of particles
pre_pos[0,:,:] = np.transpose(start_pos)
depth = np.zeros((steps,Nparticles)) # the distance form the particle
to the center

# the size of step along x,y,z coordinate
track =
np.random.normal(loc=0,scale=np.sqrt(2*diff_const*step_time),size=(ste
ps,3,Nparticles))

loca = np.zeros((steps,3,Nparticles))

#_____________________________________________________________________
# Movement for each particle during each step.

for n in range(Nparticles):
    for i in range(steps):

        depth[i,n] = np.sqrt(((pre_pos[i,:,n] - center_pos)**2).sum())
        forwd = np.sqrt(((pre_pos[i,:,n] + track[i,:,n] -
center_pos)**2).sum())

        if depth[i,n]  <= radius: # radius = image_size * pixel_size /
4

            if forwd <= radius:
                loca[i,:,n] = pre_pos[i,:,n] + track[i,:,n]

            else:

                proba = np.random.rand()
```

```
                if proba >= 0 and proba <= pout :
                    loca[i,:,n] = pre_pos[i,:,n] + track[i,:,n]

                else:
                    loca[i,:,n] = pre_pos[i,:,n]
        else:
            if forwd >= radius:
                x  = pre_pos[i,0,n] + track[i,0,n]
                y  = pre_pos[i,1,n] + track[i,1,n]
                z  = pre_pos[i,2,n] + track[i,2,n]

                if  x > boundary or x < 0:
                     loca[i,0,n] = pre_pos[i,0,n]
                else:
                    loca[i,0,n] = pre_pos[i,0,n] + track[i,0,n]

                if y > boundary or y < 0:
                    loca[i,1,n] = pre_pos[i,1,n]
                else:
                    loca[i,1,n] = pre_pos[i,1,n]+ track[i,1,n]

                if z > boundary or z < 0:
                    loca[i,2,n] = pre_pos[i,2,n]
                else:
                    loca[i,2,n] = pre_pos[i,2,n] + track[i,2,n]

            else:

                proba = np.random.rand()
                if proba >= 0 and proba <= pin :
                    loca[i,:,n] = pre_pos[i,:,n] + track[i,:,n]

                else:
                    loca[i,:,n] = pre_pos[i,:,n] # - track[i,:,n]

        pre_pos[i+1,:,n] = loca[i,:,n]

#_____________________________________________________________________
# Calculate the intensity array at t = start and t = end for the
centre z slice.
z_slice = [19, 19]
kk = [0, steps - image_size*image_size]  # the index for the start of
the scanning

image_array = np.zeros((image_size,image_size,len(z_slice)))
image_array_mobile = np.zeros((image_size,image_size,len(z_slice)))


for n in range(Nparticles):
```

```
    for k in range(0,2): # two images, one at the begninning, one at
the end
        for j in range(image_array.shape[1]): # x
            for i in range(image_array.shape[0]): # y
                beam_pos = np.array([i,j,19]) * pixel_size # only scan
the Z/2

                particle_pos = loca[ i + image_size * j + kk[k] ,:,n]
                image_array[i,j,k] +=
GaussianBeam(particle_pos,beam_pos,psf_width,psf_height)
image_array_mobile = np.array(image_array)
```