# CODE DOCUMENTATION FOR DyNAMO (Dynamic Nanoscale Axonal Microtubule Organization Model)

## Table of Contents

# 1. Variable Description:

The variables used in defining the microtubular system are described in the following table:

## *1.1* *DATA MEMBERS FOR MICROTUBULAR SYSTEM:*

| Sl. No. | Variable Name | Variable Data type | Variable Description |
|---|---|---|---|
| | MICROTUBULAR SYSTEM | | |
| 1 | rows_count | int | A variable that stores the number of parallel microtubular tracks. |
| 2 | cols_count | int | A variable that stores the number of sites in each microtubular track. |
| 3 | segments | int | A variable that stores the number of microtubule segments considered in multisegmented microtubular system. (only for multisegements) |
| 4 | vel_a | int | A variable that stores the velocity of faster motor (A) BDNF Kinesin 3. |
| 5 | vel_b | int | A variable that stores the velocity of slower motor (B) Non-BDNF Kinesin 1. |
| 6 | minimum | int | A variable that temporarily stores the velocity of motor under consideration. |
| 7 | process_a | int | A variable that stores the processivity of motor A (Kin3(F)) (no of sites). |
| 8 | process_b | int | A variable that stores the processivity of motor B (Kin1(S)) (no of sites). |
| 9 | lifetime | int | A variable that stores the lifetime of the BDNF and Non-BDNF motor in a productive reservoir. |
| 10 | association_rate | int | A variable that stores the generation rate of motor_cargo pairs (given as percentage for number of iterations a motor will bind out of total_runtime) |
| 11 | gap | int | A variable that stores the gap between two subsequent motors in a track |
| 12 | reservoir_length | int | A variable that stores the length of productive reservoir queues present at each site of microtubular system. |
| 13 | stagger | float | A variable that stores the percentage of blockage of staggered microtubular system (only for Scenarios 3-8) |

| | | | |
|---|---|---|---|
| 14 | no_of_input_iter | int | A variable that stores current iteration time instant. |
| 15 | total_runtime | int | A variable that stores the total simulation time instant. |
| | DATA STORAGE MATRICES | | |
| 1 | input_array | int | int variable array that stores the motor cargo pairs generated with the given association_rate(α) |
| 2 | particle | int | matrix that stores the position of motors (0-unoccupied, 1-motor_a, 2-motor_b) at any instant of time. |
| 3 | velocity | int | matrix that stores the corresponding velocity for motors that are present in particle matrix. |
| 4 | span | int | matrix that stores the corresponding processive sites spanned for motors that are present in particle matrix. |
| 5 | lifetime_cells | int | matrix that stores the corresponding time spent by each motor in the productive reservoir. |
| 6 | detachment_a | int | matrix that stores cumulative number of motor A (Kin3(F)) detached into productive reservoir at each instant of time. |
| 7 | detachment_b | int | matrix that stores cumulative number of motor B (Kin1(S)) detached into productive reservoir at each instant of time. |
| 8 | reattachment_a | int | matrix that stores cumulative number of motor A (Kin3(F)) reattached from productive reservoir at each instant of time. |
| 9 | reattachment_b | int | matrix that stores cumulative number of motor B (Kin1(S)) reattached from productive reservoir at each instant of time. |
| 10 | detachment_matrix_a | int | matrix that stores position of sites from where motor A (Kin3(F)) have detached into productive reservoir at each instant of time. |
| 11 | detachment_matrix_b | int | matrix that stores position of sites from where motor B (Kin1(S)) have detached into productive reservoir at each instant of time. |
| 12 | reattachment_matrix_a | int | matrix that stores position of sites from where motor A (Kin3(F)) have reattached from productive reservoir at each instant of time. |
| 13 | reattachment_matrix_b | int | matrix that stores position of sites from where motor B (Kin1(S)) have reattached from productive reservoir at each instant of time. |

| | | | |
|---|---|---|---|
| 14 | reservoir_a | int | matrix that represents productive reservoir queues of finite length (reservoir_length) for motor A (Kin3(F)) at each site of microtubular track. |
| 15 | reservoir_b | int | matrix that represents productive reservoir queues of finite length (reservoir_length) for motor B (Kin1(S)) at each site of microtubular track. |
| 16 | productive_a or waiting_a | int | matrix that stores cumulative number of motor A (Kin3(F)) waiting in productive reservoirs. |
| 17 | productive_b or waiting_b | int | matrix that stores cumulative number of motor B (Kin1(S)) waiting in productive reservoirs. |
| 18 | non_productive_a or leakage_a | int | matrix that stores cumulative number of motor A (Kin3(F)) leaked into non-productive reservoirs. |
| 19 | non_productive_b or leakage_b | int | matrix that stores cumulative number of motor B (Kin1(S)) leaked into non-productive reservoirs. |
| 20 | heatmap_a or heatmap_b | int | matrix that stores the number of times a particular site in microtubular track is occupied by motor A (Kin3(F)) or motor B (Kin1(S)). |
| | METRIC CALCULATION VARIABLES | | |
| 1 | motor_cargo_a | int | A variable that stores the number of motor A (Kin3(F)) attached on the minus end of microtubule tracks. |
| 2 | motor_cargo_b | int | A variable that stores the number of motor B (Kin1(S)) attached on the minus end of microtubule tracks. |
| 3 | throughput_a | int | A variable that stores the number of motor A (Kin3(F)) delivered from the plus end of microtubule tracks. |
| 4 | throughput_b | int | A variable that stores the number of motor B (Kin1(S)) delivered from the plus end of microtubule tracks. |
| 5 | lateral_a | int | A variable that stores the number of times motor A (Kin3(F)) that laterally move from one microtubule track to another. |
| 6 | lateral_b | int | A variable that stores the number of times motor B (Kin1(S)) that laterally move from one microtubule track to another. |
| 7 | wait_a | int | A variable that stores the number of motor A (Kin3(F)) waiting in the productive reservoir. |

| | | | |
|---|---|---|---|
| 8 | wait_b | int | A variable that stores the number of motor B (Kin1(S)) waiting in the productive reservoir. |
| 9 | leak_a | int | A variable that stores the number of motor A (Kin3(F)) leaked into the non-productive reservoir. |
| 10 | leak_b | int | A variable that stores the number of motor B (Kin1(S)) leaked into the non-productive reservoir. |
| 11 | detach_a | int | A variable that stores the number of motor A (Kin3(F)) detached from microtubular track into productive reservoir. |
| 12 | detach_b | int | A variable that stores the number of motor B (Kin1(S)) detached from microtubular track into productive reservoir. |
| 13 | reattach_a | int | A variable that stores the number of motor A (Kin3(F)) reattached from productive reservoir into microtubular track. |
| 14 | reattach_b | int | A variable that stores the number of motor B (Kin1(S)) reattached from productive reservoir into microtubular track. |
| 15 | detach_process_a | int | A variable that stores the number of motor A (Kin3(F)) detached from microtubular track into productive reservoir due to processivity. |
| 16 | detach_process_b | int | A variable that stores the number of motor B (Kin1(S)) detached from microtubular track into productive reservoir due to processivity. |

Excel worksheets that stores the different metrics are:

A. **Scenario 1 Outputs.xlsx** - The worksheet stores the number of motors attached in the minus end of microtubular track (motor_cargo), motors delivered from plus end of microtubular track (throughput), number of times motors have laterally moved into adjacent track (lateral), motors that have detached from microtubular track into productive reservoir (detach) and motors that have reattached from productive reservoir into microtubular track (reattach).

B. **Scenario 1 Productive and Non-Productive.csv** – The worksheet stores cumulative number of motors waiting in productive reservoir at each corresponding site (productive) for each track individually. The worksheet also stores the cumulative number of motors leaked into non-productive reservoir at each corresponding site (non-productive) for each track individually. Each corresponding reservoir at each site is represented by productive[row][column] or non-productive[row][column].

The different graphs plotted are:

A. **Particle_*.png –** The particle matrix is plotted as pcolor plot with the following color combination. No color represents vacant site. Red color signifies Motor A - Kin3 (F) and Blue color signifies Motor B - Kin1(S)

B. **Scenario 1(0-1000) Waiting motors_*.png –** The graph is plotted as number of motors waiting in productive reservoirs at each site respectively along the entire microtubule.

C. **Scenario 1(0-1000) Leakage motors_*.png –** The graph is plotted as number of motors leaked into non-productive reservoirs at each site respectively along the entire microtubule.

D. **Scenario 1(0-500) Waiting motors_*.png –** The graph is plotted as number of motors waiting in productive reservoirs at each site respectively along the first half of microtubule track. (first 500 sites)

E. **Scenario 1(0-500) Leakage motors_*.png –** The graph is plotted as number of motors leaked into non-productive reservoirs at each site respectively along the first half of microtubule track. (first 500 sites)

F. **Scenario 1(500-1000) Waiting motors_*.png –** The graph is plotted as number of motors waiting in productive reservoirs at each site respectively along the latter half of microtubule track. (last 500 sites)

G. **Scenario 1(500-1000) Leakage motors_*.png –** The graph is plotted as number of motors leaked into non-productive reservoirs at each site respectively along the latter half of microtubule track. (last 500 sites)

H. **Reservoir1/Productive_*.png -** The graph is plotted as number of motors waiting and leaked in productive reservoirs at each site respectively along the entire microtubule at every dynamic interval.

I. **Reservoir2/Productive_*.png -** The graph is plotted as cumulative number of motors waiting and leaked in productive reservoirs at each site respectively along the entire microtubule after every time step.

   **The graphs are plotted at interval of 100 iterations (or 4 secs) and are combined serially with frame duration of 200ms to form .gif outputs.**

J. **1 Heatmap.png –** The graph shows a color contour map of the number of times a particular site in microtubular track is occupied in total_runtime.

K. There are several ***.gif** which shows the motor movement along axon and productive reservoir dynamics.

## *1.2* *MEMBER FUNCTIONS:*

### *1.2.1* *Motor-cargo Pair Generation:*

The member function of motor-cargo pair generation takes in generation rate "association rate" ($\alpha_a$, $\alpha_b$) for (A)BDNF Kinesin 3 and (B)Non-BDNF Kinesin 1 respectively. The association rate is taken as integer variable. It is given as percentage of total iterations a particular motor will attach. (e.g. 10% of 15000 iterations means = 1500 motor a/b will join within 15000 iterations of total runtime). The function returns input_array as output which stores motor_a and motor_b at random positions.

Function declaration for motor cargo generation is:
**def poisson_input (input_array, count_a, count_b, association_rate)**

where,
**input_array** = array of length of total runtime where each position is occupied by 1500 motor A, 1500 motor B and remaining empty positions.
**association_rate** = motor_cargo generation rate (as percentage)

**Each iteration** = 40 ms

**Total_runtime** = 15000*40ms = 600000 ms = 600 secs

**Motor_generation_rate / association_rate** = 1500 motors in 600 secs = 2.5 motors/sec

### *1.2.2* *Motor-cargo loading on minus end of microtubular track:*

In each instant of time, corresponding site of input_array is taken into consideration. Either (A)BDNF Kinesin 3 or (B)Non-BDNF Kinesin 1 or vacant site is loaded from the input array into the minus end of one of the microtubular tracks. The loaded motor occupying the particle matrix is updated with all other parameters like velocity, span (reset as 1) and lifetime (reset to 0). The function returns a loaded particle matrix and stores count of motors loaded in variables motor_cargo_a and motor_cargo_b respectively.

Function declaration for motor cargo loading on minus end is:
**def initial_association (no_of_input_iter, input_array, rows_count, motor_cargo_a, motor_cargo_b)**

**return variables =** no_of_input_iter, motor_cargo_a, motor_cargo_b

where,
**no_of_input_iter** = variable that stores the current iteration time instant.

**motor_cargo_a / motor_cargo_b =** variable that stores the number of motor A (Kin3(F)) and motors B loading on microtubular track within that time instant.

### *1.2.3 Motor reattachment from productive reservoir into microtubular track: (LK model)*

At every time instant, the motors present in the productive reservoirs have a scope to reattach back into the microtubular tracks. The motors present in the front of productive reservoir queues are checked of their current lifetime. If the motors are waiting less than their given lifetime, then the motors can reattach on one of the nine neighboring sites if available or vacant. If the motors have crossed their lifetime waiting in the productive reservoir are leaked into non-productive reservoirs.

Function declaration for motor reattachment from productive reservoir:

**def lateral_association (rows_count, cols_count, reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b, reattachment_matrix_a, reattachment_matrix_b, leakage_a, leakage_b, time)**

**return variables =** reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b, reattachment_matrix_a, reattachment_matrix_b, leakage_a, leakage_b

where,
**rows_count, cols_count** = give the corresponding site under consideration

**reservoir_a / reservoir_b** = current state of limited length productive reservoirs queues located at each site.

**reattachment_a / reattachment_b** = matrix that stores the number of motors that have reattached from productive reservoir into each corresponding microtubular track.

**reattach_a / reattach_b** = variable that stores total cumulative number of motors that have reattached from productive reservoir into microtubular track.

**reattachment_matrix =** binary matrix that stores the corresponding location from where one motor is leaked out. The matrix stores value just for one time instant and is reset in each time iteration.

**leakage_a / leakage_b** = matrix that keeps tally of total number of motors leaked form each corresponding productive reservoir into non-productive reservoir.

**time** = variable that temporarily stores the current time instant

### *1.2.4 Motor traversing along the same track or movement into lateral track: (TASEP model)*

The motor present on microtubular track (particle) traverse from one site to another on availability of sufficient gap. If a motor faces crowding, then the motor gets detached from microtubular track into productive reservoirs. If provision for lateral movement are provided for motors, then before detachment, motor gets a chance to laterally move into adjacent track if vacant.

But during detachment if the productive reservoir is full then the motor present at top of queue is popped and directly pushed into non-productive reservoir while the detached motor is pushed at the rear of productive reservoir.

A motor can also detach if a motor has taken subsequent processive steps during transport. On crossing processive limit the motor is pushed into productive reservoir.

Function declaration for motor transport function:
**def transport (rows_count, cols_count, vel_a, vel_b, process_a, process_b, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a, detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b, time):**

**return variables =** throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a, detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b

where,
**throughput_a / throughput_b =** variable that keeps count of total number of motors delivered from the plus end of microtubular track.

**reservoir_a / reservoir_b** = current state of limited length productive reservoirs queues located at each site.

**detachment_a / detachment_b** = matrix that stores the number of motors that have detached from microtubular track into productive reservoir.

**detach_a / detach_b** = variable that stores the total cumulative number of motors that have detached from microtubular track into productive reservoir.

**detachment_matrix =** binary matrix that temporarily stores the corresponding location from where one motor has detached.

**leakage_a / leakage_b** = matrix that keeps tally of total number of motors leaked form each corresponding productive reservoir into non-productive reservoir.

**lateral_a / lateral_b =** variable that keeps count the number of times motor has laterally moved into adjacent track.

### *1.2.5* *Main function:*

The main function runs the lateral_association (reattachment), transport (TASEP) and initial_association (attachment/loading) correspondingly at each time instant for total simulation runtime.

The different metric values are stored in **"Scenario 1 Outputs.xlsx"** at each time instant. After every 100 iterations, the current state of productive and non-productive reservoirs is stored in **"Scenario 1 Productive and Non-Productive.csv".**

The different graphs as mentioned above are plotted and saved into different corresponding folders. These photos are later clubbed into corresponding .gif file with frame duration of 200ms.

The heatmap is plotted at end of main function.

# 2. CODEs for each part of model:

## *2.1* *IMPORTED LIBRARIES:*

```
import xlsxwriter                          // writing excel files (.xlsx)
import random                              // define random variables
import matplotlib.pyplot as plt            // plotting functions
import math                                // math function
import numpy as np                         // array function
import seaborn as sns                      // database function
import csv                                 // writing .csv files
from timeit import default_timer as timer  // time functions
import pdb                                 // python debugger
import os                                  // operating system functions
import glob                                // gif image sorting
import pandas as pd                        // excel read commands

from queue import Queue                    // queuing functionalities

from matplotlib.colors import LogNorm      //plot colours
from PIL import Image, ImageDraw, ImageFont
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
from textwrap import wrap                  // text wrapping
import glob
import pandas as pd
from collections import deque
from timeit import default_timer as timer


//colourmap for pcolor and heatmap plots
from matplotlib.colors import LinearSegmentedColormap
cmap_name = 'my_list'
colors = [(1, 1, 1), (0, 0, 1), (1, 0, 0)]  # White -> Blue -> Red
cm = LinearSegmentedColormap.from_list(cmap_name, colors, N=3)
```

## *2.2* *OPERATING SYSTEM FOLDER GENERATION:*

```
script_dir = os.getcwd()
graphs_dir = os.path.join(script_dir, 'Reservoir1/')

if not os.path.isdir(graphs_dir):
   os.makedirs(graphs_dir)

graphs_dir = os.path.join(script_dir, 'Reservoir2/')

if not os.path.isdir(graphs_dir):
   os.makedirs(graphs_dir)

graphs_dir = os.path.join(script_dir, 'Reservoir_Stagger/')

if not os.path.isdir(graphs_dir):
   os.makedirs(graphs_dir)

graphs_dir = os.path.join(script_dir, 'Reservoir_NonStagger/')

if not os.path.isdir(graphs_dir):
   os.makedirs(graphs_dir)

images_dir = os.path.join(script_dir, 'Images/')

if not os.path.isdir(images_dir):
   os.makedirs(images_dir)

images_dir = os.path.join(script_dir, 'GIFs/')

if not os.path.isdir(images_dir):
   os.makedirs(images_dir)
```

## *2.3* *DATA VARIABLES:*

```
association_rate = 20                          //Motors (a+b) entering the system per sec

rows_count = 3                                 // number of parallel tracks
cols_count = 1000                              // number of sites in each track

# Velocity steps:

# Velocity ratio : Va/Vb (Varying it)          // velocity of motors
vel_a = 5  #1000nm/s
vel_b = 3  #600nm/s


#Keeping processivity of A and B same ie Pa/Pb = 1      // processivity of motors
process_a = 1005
process_b = 150
```

```
#Lifetime                                   //lifetime of motors
lifetime = 1500                             #1 min lifetime

gap = 0
minimum = 0
reservoir_length = 50                       //length of productive reservoir

total_runtime = 15001                       // total simulation runtime
count_a = int(total_runtime/2)
count_b = int(total_runtime/2)
```

## 2.4 *DATA STORAGE METRICS:*

```
#Input queue
input_array = [0 for i in range(count_a + count_b)]

#Association
motor_cargo_a = 0   #no of a type motor cargos
motor_cargo_b = 0   #no of b type motor cargos

#Detachment at the end
throughput_a = 0 #cargo_a output
throughput_b = 0 #cargo_b output

lateral_a = 0
lateral_b = 0
wait_a = 0
wait_b = 0
leak_a = 0
leak_b = 0
detach_a = 0
detach_b = 0
reattach_a = 0
reattach_b = 0
detach_process_a=0
detach_process_b=0

#Motors lost due to detachment
detachment_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
detachment_b = [[0 for j in range(cols_count)] for i in range(rows_count)]

#Motors reused due to reattachment
reattachment_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
reattachment_b = [[0 for j in range(cols_count)] for i in range(rows_count)]


#Heatmaps
heat_map_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
heat_map_b = [[0 for j in range(cols_count)] for i in range(rows_count)]
```

```
#Motor position and parameters
particle = [[0 for j in range(cols_count)] for i in range(rows_count)]
velocity = [[0 for j in range(cols_count)] for i in range(rows_count)]
span = [[0 for j in range(cols_count)] for i in range(rows_count)]
lifetime_cells = [[0 for j in range(cols_count)] for i in range(rows_count)]

#Productive reservoirs
waiting_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
waiting_b = [[0 for j in range(cols_count)] for i in range(rows_count)]

productive_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
productive_b = [[0 for j in range(cols_count)] for i in range(rows_count)]

#Non-Productive reservoirs
leakage_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
leakage_b = [[0 for j in range(cols_count)] for i in range(rows_count)]

non_productive_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
non_productive_b = [[0 for j in range(cols_count)] for i in range(rows_count)]
```

## *2.5* *PRODUCTIVE RESERVOIR DEFINING FUNCTION:*

```
class props(object): #detachment time and lifetime
    def __init__(self, joins, life):
        self.joins = joins
        self.life = life
        return

reservoir_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
reservoir_b = [[0 for j in range(cols_count)] for i in range(rows_count)]

for i in range(rows_count):
    for j in range(cols_count):
        reservoir_a[i][j] = Queue(maxsize = reservoir_length)
        reservoir_b[i][j] = Queue(maxsize = reservoir_length)
```

## *2.6* *WORKSHEET CREATION:*

### *2.6.1* Output File:

```
workbook = xlsxwriter.Workbook('Scenario 1 Outputs.xlsx')

worksheet9 = workbook.add_worksheet()            #Outputs

worksheet9.write(0, 0, "Time")
worksheet9.write(0, 1, "Input a")
worksheet9.write(0, 2, "Input b")
worksheet9.write(0, 3, "Output a")
```

```
worksheet9.write(0, 4, "Output b")
worksheet9.write(0, 5, 'Lateral a')
worksheet9.write(0, 6, 'Lateral b')
worksheet9.write(0, 7, 'Detach a')
worksheet9.write(0, 8, 'Detach b')
worksheet9.write(0, 9, 'Reattach a')
worksheet9.write(0, 10, 'Reattach b')
worksheet9.write(0, 11, "Process Detach a")
worksheet9.write(0, 12, "Process Detach b")
```

### *2.6.2* Productive and Non-productive Reservoirs:

```
column =
"Site,Waiting_A_Track_1,Waiting_B_Track_1,Waiting_A_Track_2,Waiting_B_Track_2
,Waiting_A_Track_3,Waiting_B_Track_3,Leakage_A_Track_1,Leakage_B_Track_1,Le
akage_A_Track_2,Leakage_B_Track_2,Leakage_A_Track_3,Leakage_B_Track_3\n"

dir_1 = "Scenario 1 Productive and Non-Productive.csv"
csv1 = open(dir_1, "a", newline='')
writer1 = csv.writer(csv1, dialect='excel')
csv1.write(column)

dir_2 = "Scenario 1 Productive and Non-Productive (0-500).csv"
csv2 = open(dir_2, "a", newline='')
writer2 = csv.writer(csv2, dialect='excel')
csv2.write(column)

dir_3 = "Scenario 1 Productive and Non-Productive (500-1000).csv"
csv3 = open(dir_3, "a", newline='')
writer3 = csv.writer(csv3, dialect='excel')
csv3.write(column)
```

## *2.7* *DIFFERENT FUNCTIONS:*

### *2.7.1* Motor-cargo pair generation:

```
res_in_a_len = int((association_rate*total_runtime)/100)
res_in_b_len = int((association_rate*total_runtime)/100)
res_in_len = total_runtime-(res_in_a_len-res_in_b_len)

res_in_l = []
input_res = Queue(maxsize = total_runtime)
res_in_l = deque(maxlen = total_runtime)


for i in range(res_in_len):
    res_in_l.append(0)

for i in range(res_in_a_len):
    res_in_l.append(1)
```

```
for i in range(res_in_b_len):
    res_in_l.append(2)

random.shuffle(res_in_l)

for i in range(total_runtime):
    input_res.put(res_in_l[i])
```

### *2.7.2* Motor-cargo loading on minus end of microtubular track:

```
def initial_association(particle, velocity, span, lifetime_cells, input_res, motor_cargo_a,
motor_cargo_b):

    start = timer()

    channels = list(range(0,rows_count))                    # All channels
    random.shuffle(channels)
    c1 = channels[0]
    channels.remove(channels[0])
    c2 = channels[0]
    channels.remove(channels[0])
    c3 = channels[0]

    if(particle[c1][0] == 0):
        particle[c1][0] = input_res.get()
        span[c1][0] = 1
        lifetime_cells[c1][0] = 1

        if(particle[c1][0] == 1):
            motor_cargo_a += 1
            velocity[c1][0] = vel_a

        elif(particle[c1][0] == 2):
            motor_cargo_b += 1
            velocity[c1][0] = vel_b


    elif(particle[c2][0] == 0):
        particle[c2][0] = input_res.get()
        span[c2][0] = 1
        lifetime_cells[c2][0] = 1

        if(particle[c2][0] == 1):
            motor_cargo_a += 1
            velocity[c2][0] = vel_a

        elif(particle[c2][0] == 2):
            motor_cargo_b += 1
            velocity[c2][0] = vel_b
```

```
    elif(particle[c3][0] == 0):
        particle[c3][0] = input_res.get()
        span[c3][0] = 1
        lifetime_cells[c3][0] = 1

        if(particle[c3][0] == 1):
            motor_cargo_a += 1
            velocity[c3][0] = vel_a

        elif(particle[c3][0] == 2):
            motor_cargo_b += 1
            velocity[c3][0] = vel_b

    end = timer()
    print("z take", end - start)
    #pdb.set_trace()
    return particle, velocity, span, lifetime_cells, input_res, motor_cargo_a,motor_cargo_b
```

### *2.7.3* Motor reattachment from productive reservoir into microtubular track: (LK model

```
start = timer()
    i=0
    j=0
    k=0
    temp = 0
    track = 0
    site = 0
    flag = 0
    flag2 = 0
    step = 0
    step1 = 0
    for i in range(rows_count):
        for j in range(cols_count):
            if(reservoir_a[i][j].qsize() != 0 and reservoir_b[i][j].qsize() == 0):
                flag = 0
            elif(reservoir_a[i][j].qsize() == 0 and reservoir_b[i][j].qsize() != 0):
                flag = 1
            elif(reservoir_a[i][j].qsize() != 0 and reservoir_b[i][j].qsize() != 0):
                flag = random.randint(0,1)
            else:
                flag = 10

            if(flag == 0):
                t1 = reservoir_a[i][j].queue[0].joins
                t2 = reservoir_a[i][j].queue[0].life
                                        //if motors A have reached beyond lifetime, then
                                        they are pushed into non-productive reservoir
```

```
    if(time-t1+t2 > lifetime):
        #remove the element as leakage
        print("Leak A")
        reservoir_a[i][j].get()
        leakage_a[i][j] += 1

                                    //motors A can reattach into any one of the nine
                                    neighboring sites

else:
        channels = list(range(0,rows_count))
        channels.remove(i)
        random.shuffle(channels)
        track1 = channels[0]
        track2 = channels[1]
        track = j
        if(j==cols_count):
            if(particle[i][j] == 0): site = j; track = i;
            elif(particle[i][j-1] == 0): site = j-1; track = i;
            elif(particle[track1][j] == 0): site = j; track = track1;
            elif(particle[track1][j-1] == 0): site = j-1; track = track1;
            elif(particle[track2][j] == 0): site = j; track = track2;
            elif(particle[track2][j-1] == 0): site = j-1; track = track2;
            else: continue;
        else:
            if(particle[i][j+1] == 0): site = j+1; track = i;
            elif(particle[track1][j+1] == 0): site = j+1; track = track1;
            elif(particle[track2][j+1] == 0): site = j+1; track = track2;
            elif(particle[i][j] == 0): site = j; track = i;
            elif(particle[track1][j] == 0): site = j; track = track1;
            elif(particle[track2][j] == 0): site = j; track = track2;
            elif(particle[i][j-1] == 0): site = j-1; track = i;
            elif(particle[track1][j-1] == 0): site = j-1; track = track1;
            elif(particle[track2][j-1] == 0): site = j-1; track = track2;
            else:  continue;


        particle[track][site] = 1
        velocity[track][site] = vel_a
        span[track][site] = 1

        t = reservoir_a[i][j].get()
        lifetime_cells[i][site] = time-t.joins+t.life
        #print("Reattach A")
        reattachment_a[track][site] += 1
        reattachment_matrix_a[track][site] = 1
        reattach_a += 1

elif(flag == 1):
    t1 = reservoir_b[i][j].queue[0].joins
    t2 = reservoir_b[i][j].queue[0].life
```

```
                                        //if motors B have reached beyond lifetime, then
                                        they are pushed into non-productive reservoir

    if(time-t1+t2 > lifetime):
        #remove the element as leakage
        print("Leak B")
        reservoir_b[i][j].get()
        leakage_b[i][j] += 1

                                        //motors B can reattach into any one of the nine
                                        neighboring sites

    else:
        channels = list(range(0,rows_count))
        channels.remove(i)
        random.shuffle(channels)
        track1 = channels[0]
        track2 = channels[1]
        track = j
        if(j==cols_count):
            if(particle[i][j] == 0): site = j; track = i;
            elif(particle[i][j-1] == 0): site = j-1; track = i;
            elif(particle[track1][j] == 0): site = j; track = track1;
            elif(particle[track1][j-1] == 0): site = j-1; track = track1;
            elif(particle[track2][j] == 0): site = j; track = track2;
            elif(particle[track2][j-1] == 0): site = j-1; track = track2;
            else: continue;
        else:
            if(particle[i][j+1] == 0): site = j+1; track = i;
            elif(particle[track1][j+1] == 0): site = j+1; track = track1;
            elif(particle[track2][j+1] == 0): site = j+1; track = track2;
            elif(particle[i][j] == 0): site = j; track = i;
            elif(particle[track1][j] == 0): site = j; track = track1;
            elif(particle[track2][j] == 0): site = j; track = track2;
            elif(particle[i][j-1] == 0): site = j-1; track = i;
            elif(particle[track1][j-1] == 0): site = j-1; track = track1;
            elif(particle[track2][j-1] == 0): site = j-1; track = track2;
            else:  continue;

        particle[track][site] = 2
        velocity[track][site] = vel_b
        span[track][site] = 1

        t=reservoir_b[i][j].get()
        lifetime_cells[i][site] = time-t.joins+t.life
        #print("Reattach B")
        reattachment_b[track][site] += 1
        reattachment_matrix_b[track][site] = 1
        reattach_b += 1
else:
    continue
```

```
    end = timer()
    print("x take", end - start)
    #pdb.set_trace()
    return particle, velocity, span, lifetime_cells, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, reattachment_matrix_a, reattachment_matrix_b,
leakage_a, leakage_b
```

### *2.7.4* Motor traversing along the same track or movement into lateral track: (TASEP model)

```
def transport(particle, velocity, span, lifetime_cells, rows_count, cols_count, vel_a,
vel_b, process_a, process_b, throughput_a, throughput_b, reservoir_a, reservoir_b,
detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a,
detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b, detach_process_a,
detach_process_b, time):
    start=timer()

    for i in range(rows_count):
        gap=0                                                    //check gap between motors
        minimum=0
        for j in range(cols_count-1,-1,-1):
            if(particle[i][j] == 0):
                gap += 1
                continue
            else:
                #Output
                if(particle[i][j] == 1):
                    if(gap < vel_a):
                        minimum = 0
                    elif(gap >= vel_a):
                        minimum = vel_a

                #Constant velocity
                elif(particle[i][j] == 2):
                    if(gap < vel_b):
                        minimum = 0
                    elif(gap >= vel_b):
                        minimum = vel_b

                                                    //if sufficient gap is present then motor moves with
                                                    constant velocity along the microtubular track

                #Movement
                if(minimum > 0):
                    if(particle[i][j] == 1):
                        if((j + vel_a) >= cols_count-1):          //delivery of motor A
                            throughput_a += 1
                            velocity[i][j] = 0
```

```
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

        else:                                   //motor A move from one site to another
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 1
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

            if(span[i][j+minimum] >= process_a):
                #detachment if motor exceeds its processivity
                velocity[i][j+minimum] = 0
                particle[i][j+minimum] = 0
                span[i][j+minimum] = 0
                print('y1')
                if(reservoir_a[i][j].qsize()==reservoir_length):
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1

                reservoir_a[i][j].put(props(time,lifetime_cells[i][j+minimum]))

                lifetime_cells[i][j+minimum]=0
                detachment_a[i][j+minimum] += 1
                detachment_matrix_a[i][j+minimum] = 1
                detach_a +=1

    elif(particle[i][j] == 2):
        if((j + vel_b) >= cols_count-1):                //delivery of motor B
            throughput_b += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

        else:                                   //motor B move from one site to another
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 2
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

            if(span[i][j+minimum] >= process_b):
                #detachment if motor exceeds its processivity
```

```
                velocity[i][j+minimum] = 0
                particle[i][j+minimum] = 0
                span[i][j+minimum] = 0
                print('y2')
                if(reservoir_b[i][j].qsize()==reservoir_length):
                    reservoir_b[i][j].get()
                    leakage_b[i][j] += 1

                reservoir_b[i][j].put(props(time,lifetime_cells[i][j+minimum]))

                lifetime_cells[i][j+minimum]=0
                detachment_b[i][j+minimum] += 1
                detachment_matrix_b[i][j+minimum] = 1
                detach_b +=1

#Detachment or Lateral Movement
elif(minimum == 0):
    if(particle[i][j] == 1):
        # Lateral association will happen to only cargos lost at initial stages
        channels = list(range(0,rows_count))        # All channels
        channels.remove(i)
        random.shuffle(channels)
        y1 = channels[0]
        channels.remove(channels[0])
        y2 = channels[0]

        if((j + vel_a) >= cols_count-1):
            throughput_a += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            lateral_a += 1
        else:                                       //Lateral movement of motor A
            if(particle[y1][j+vel_a] == 0):
                velocity[y1][j+vel_a] = vel_a
                particle[y1][j+vel_a] = 1
                span[y1][j+vel_a] = span[i][j] + vel_a
                lifetime_cells[y1][j+vel_a] = lifetime_cells[i][j]
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_a += 1

            elif(particle[y2][j+vel_a] == 0):
                velocity[y2][j+vel_a] = vel_a
                particle[y2][j+vel_a] = 1
                span[y2][j+vel_a] = span[i][j] + vel_a
                lifetime_cells[y2][j+vel_a] = lifetime_cells[i][j]
                velocity[i][j] = 0
```

```
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_a += 1

            else:                                   //Detachment of motor A
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                #print('y3')
                if(reservoir_a[i][j].qsize()==reservoir_length):
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1

                reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

                lifetime_cells[i][j]=0
                detachment_a[i][j] += 1
                detachment_matrix_a[i][j] = 1
                detach_a += 1

        elif(particle[i][j] == 2):
            channels = list(range(0,rows_count))        # All channels
            channels.remove(i)
            random.shuffle(channels)
            y1 = channels[0]
            channels.remove(channels[0])
            y2 = channels[0]

            if((j + vel_b) >= cols_count-1):
                throughput_b += 1
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_b += 1
            else:                                   //Lateral movement of motor B
                if(particle[y1][j+vel_b]== 0):
                    velocity[y1][j+vel_b] = vel_b
                    particle[y1][j+vel_b] = 2
                    span[y1][j+vel_b] = span[i][j] + vel_b
                    lifetime_cells[y1][j+vel_b] = lifetime_cells[i][j]
                    velocity[i][j] = 0
                    particle[i][j] = 0
                    span[i][j] = 0
                    lifetime_cells[i][j] = 0
                    lateral_b += 1

                elif(particle[y2][j+vel_b]== 0):
                    velocity[y2][j+vel_b] = vel_b
                    particle[y2][j+vel_b] = 2
```

```
                    span[y2][j+vel_b] = span[i][j] + vel_b
                    lifetime_cells[y2][j+vel_b] = lifetime_cells[i][j]
                    velocity[i][j] = 0
                    particle[i][j] = 0
                    span[i][j] = 0
                    lifetime_cells[i][j] = 0
                    lateral_b += 1

                else:                                   //Detachment of motor B
                    velocity[i][j] = 0
                    particle[i][j] = 0
                    span[i][j] = 0
                    #print('y4')
                    if(reservoir_b[i][j].qsize()==reservoir_length):
                        reservoir_b[i][j].get()
                        leakage_b[i][j] += 1

                    reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))

                    lifetime_cells[i][j]=0
                    detachment_b[i][j] += 1
                    detachment_matrix_b[i][j] = 1
                    detach_b += 1


            gap = 0

    end = timer()
    print("y take", end - start)
    #pdb.set_trace()
    return particle, velocity, span, lifetime_cells, throughput_a, throughput_b, reservoir_a,
reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a,
detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b, detach_process_a,
detach_process_b
```

## *2.8* *MAIN FUNCTION:*

```
for iter in range(total_runtime):                   //repeated simulation of functions

    start = timer()
    detachment_matrix_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
    detachment_matrix_b = [[0 for j in range(cols_count)] for i in range(rows_count)]

    reattachment_matrix_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
    reattachment_matrix_b = [[0 for j in range(cols_count)] for i in range(rows_count)]


    particle, velocity, span, lifetime_cells, reservoir_a, reservoir_b, reattachment_a,
    reattachment_b, reattach_a, reattach_b, reattachment_matrix_a
    reattachment_matrix_b, leakage_a, leakage_b = lateral_association(particle, velocity,
```

```
span, lifetime_cells, rows_count, cols_count, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, reattachment_matrix_a,
reattachment_matrix_b, leakage_a, leakage_b, iter)

particle, velocity, span, lifetime_cells, throughput_a, throughput_b, reservoir_a,
reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a,
detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b, detach_process_a,
detach_process_b = transport(particle, velocity, span, lifetime_cells, rows_count,
cols_count, vel_a, vel_b, process_a, process_b, throughput_a,
throughput_b,reservoir_a, reservoir_b, detachment_a, detachment_b,
detach_a,detach_b, detachment_matrix_a, detachment_matrix_b, leakage_a,
leakage_b, lateral_a, lateral_b, detach_process_a, detach_process_b, iter)

particle, velocity, span, lifetime_cells, input_res, motor_cargo_a, motor_cargo_b =
initial_association(particle, velocity, span, lifetime_cells, input_res, motor_cargo_a,
motor_cargo_b)

print(motor_cargo_a,motor_cargo_b)
print(throughput_a,throughput_b)


//save metrics to .xlsx file

worksheet9.write(iter+1, 0, iter+1)
worksheet9.write(iter+1, 1, motor_cargo_a)
worksheet9.write(iter+1, 2, motor_cargo_b)
worksheet9.write(iter+1, 3, throughput_a)
worksheet9.write(iter+1, 4, throughput_b)
worksheet9.write(iter+1, 5, lateral_a)
worksheet9.write(iter+1, 6, lateral_b)
worksheet9.write(iter+1, 7, detach_a)
worksheet9.write(iter+1, 8, detach_b)
worksheet9.write(iter+1, 9, reattach_a)
worksheet9.write(iter+1, 10, reattach_b)
worksheet9.write(iter+1, 11, detach_process_a)
worksheet9.write(iter+1, 12, detach_process_b)


waiting_a = [[0 for j in range(cols_count)] for i in range(rows_count)]
waiting_b = [[0 for j in range(cols_count)] for i in range(rows_count)]
wait_a = 0
wait_b = 0
leak_a = 0
leak_b = 0

//productive and non-productive reservoir
for i in range(rows_count):
    for j in range(cols_count):
        waiting_a[i][j] = reservoir_a[i][j].qsize()
        waiting_b[i][j] = reservoir_b[i][j].qsize()
        productive_a[i][j] += waiting_a[i][j]
```

```
            productive_b[i][j] += waiting_b[i][j]
            non_productive_a[i][j] += leakage_a[i][j]
            non_productive_b[i][j] += leakage_b[i][j]
            wait_a += waiting_a[i][j]
            wait_b += waiting_b[i][j]
            leak_a += leakage_a[i][j]
            leak_b += leakage_b[i][j]

    d_a = [0 for j in range(cols_count)]
    d_b = [0 for j in range(cols_count)]
    r_a = [0 for j in range(cols_count)]
    r_b = [0 for j in range(cols_count)]
    w_a = [0 for j in range(cols_count)]
    w_b = [0 for j in range(cols_count)]
    l_a = [0 for j in range(cols_count)]
    l_b = [0 for j in range(cols_count)]

    for j in range(cols_count):
        for i in range(rows_count):
            d_a[j] += detachment_a[i][j]
            d_b[j] += detachment_b[i][j]
            r_a[j] += reattachment_a[i][j]
            r_b[j] += reattachment_b[i][j]
            w_a[j] += reservoir_a[i][j].qsize()
            w_b[j] += reservoir_b[i][j].qsize()
            l_a[j] += leakage_a[i][j]
            l_b[j] += leakage_b[i][j]

    for j in range(cols_count):
        d_a[j] = d_a[j]/total_runtime
        d_b[j] = d_b[j]/total_runtime
        r_a[j] = r_a[j]/total_runtime
        r_b[j] = r_b[j]/total_runtime

    print("Done")
    #column9 = "Site, Waiting A Track 1, Waiting B Track 1, Waiting A Track 2, Waiting
B Track 2, Leakage A Track 1, Leakage B Track 1, Leakage A Track 2, Leakage B Track
2"

    //heatmap matrix
    for i in range(rows_count):
        for j in range(cols_count):
            temp = int(j/10)
            temp1 = int(j/5)
            if(particle[i][j] == 1):
                heat_map_a[i][j] += particle[i][j]
                heat_map_a_2[i][temp] += particle[i][j]
                heat_map_a_3[i][temp1] += particle[i][j]
            elif(particle[i][j] == 2):
                heat_map_b[i][j] += particle[i][j]
                heat_map_b_2[i][temp] += particle[i][j]
```

```
                heat_map_b_3[i][temp1] += particle[i][j]


    end = timer()
    print("\n Run: "+ str(iter+1), (end - start))

    if(iter>0 and iter%100==0):

        t = str(iter+1).zfill(5)
        #-----------
        #Particle Graphs
        #----------

        time=(iter)*time_stamp
        temp=particle[0][0]
        particle[0][0] = 2
        plt.pcolor(particle, linewidths = 0.2, cmap=cm)
        plt.title("\n".join(wrap("Movement of Kinesin motors (Scenario 1 - With
Processivity) After time %0.2f"%round(time,2) + ' secs', 60)), fontsize=14)
        plt.xlabel('Lattice Sites')
        plt.ylabel('Track')
        custom_lines = [Patch(facecolor='white', edgecolor='k',label='Empty Space'),
                        Patch(facecolor='blue', edgecolor='k',label='Motor A (faster)'),
                        Patch(facecolor='red', edgecolor='k',label='Motor B (slower)')]
        plt.legend(handles=custom_lines,loc='center left', bbox_to_anchor=(1, 0.5))
        plt.yticks((0.5, 1.5, 2.5),('1', '2', '3'))
        plt.gcf().text(.1, 0.0, "Data Values: (With Processivity)", fontsize=12)
        plt.gcf().text(.1, -0.04, "Association rate: 10 motors/sec", fontsize=10)
        plt.gcf().text(.1, -0.08, "Processivity::   Motor A: 8000 nm       Motor B:
"+str(process_b*8)+" nm", fontsize=10)
        plt.gcf().text(.1, -0.12, "Input::     Motor A: " +str(motor_cargo_a) + "     Motor B: "
+str(motor_cargo_b), fontsize=10)
        plt.gcf().text(.1, -0.16, "Output::  Motor A: " +str(throughput_a) + "     Motor B: "
+str(throughput_b), fontsize=10)
        plt.gcf().text(.1, -0.20, "Productive Reservoir::         Motor A: " +str(wait_a) + "
Motor B: " +str(wait_b), fontsize=10)
        plt.gcf().text(.1, -0.24, "Non-productive Reservoir::  Motor A: " +str(leak_a) + "
Motor B: " +str(leak_b), fontsize=10)
        plt.tight_layout()
        plt.savefig('Images/Particle_' + t + '.png',  dpi=100, bbox_inches = 'tight')
        plt.draw()
        plt.close()
        print("Graph 1 Done")

        particle[0][0] = temp

        //Productive Reservoir Plots
        plt.figure(1)
        fig, (ax0, ax1) = plt.subplots(ncols=1, nrows = 2, figsize=(8,5.5),
                    gridspec_kw={"height_ratios":[1,1]})
```

```
        ax0.plot(cols1, w_a, alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols1, w_b, alpha=0.5, color="Red", label="Motor B")
        ax0.set_ylim(bottom=0)
        ax0.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax0.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax0.set_title("\n".join(wrap('No of motors waiting in the productive reservoir', 60)),
fontsize=14)
        ax0.legend(loc="upper right")

        ax1.plot(cols1, l_a, alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols1, l_b, alpha=0.5, color="Red", label="Motor B")
        ax1.set_ylim(bottom=0)
        ax1.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax1.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax1.set_title("\n".join(wrap('No of motors leaked from the productive reservoir',
60)), fontsize=14)
        ax1.legend(loc="upper right")

        fig.suptitle("\n".join(wrap('Dynamic state of Productive Reservoir Metrics (Scenario
1 - With Processivity) after time %0.2f'%round(time,2) + ' secs', 60)), fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(top=0.8, hspace = 0.7)
        plt.gcf()
        plt.savefig('Reservoir1/Productive_'+t+'.png',  dpi=100, bbox_inches = 'tight')
        #plt.show()
        plt.close()
        print("Graph 2 Done")


        plt.figure(2)
        fig, (ax2, ax3) = plt.subplots(ncols=1, nrows = 2, figsize=(8,5.5),
                        gridspec_kw={"height_ratios":[1,1]})

        ax2.plot(cols1, d_a, alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols1, d_b, alpha=0.5, color="Red", label="Motor B")
        ax2.set_ylim(bottom=0)
        ax2.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax2.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=13)
        ax2.set_title("\n".join(wrap('No of times motors detached into reservoir from lattice
site', 60)), fontsize=14)
        ax2.legend(loc="upper right")

        ax3.plot(cols1, r_a, alpha=0.6, color="Blue", label="Motor A")
        ax3.plot(cols1, r_b, alpha=0.5, color="Red", label="Motor B")
        ax3.set_ylim(bottom=0)
        ax3.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax3.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=13)
        ax3.set_title("\n".join(wrap('No of times motors reattached from reservoir into
lattice site', 60)), fontsize=14)
        ax3.legend(loc="upper right")
```

```
        fig.suptitle("\n".join(wrap('Cumulative state of Productive Reservoir Metrics
(Scenario 1 - With Processivity) after time %0.2f'%round(time,2) + ' secs', 60)),
fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(top=0.8, hspace = 0.7)
        plt.gcf()
        plt.savefig('Reservoir2/Productive_'+t+'.png',  dpi=100, bbox_inches = 'tight')
        #plt.show()
        plt.close()
        print("Graph 3 Done")


    //writing the metrics of reservoirs in .csv file

    if(iter>0 and iter%100==0):
        csv1 = open(dir_1, "a", newline='')
        writer1 = csv.writer(csv1, dialect='excel')
        for i in range(cols_count):
            strings = list()
            strings.append(str((i+1)))
            strings.append(str(productive_a[0][i]/iter))
            strings.append(str(productive_b[0][i]/iter))
            strings.append(str(productive_a[1][i]/iter))
            strings.append(str(productive_b[1][i]/iter))
            strings.append(str(productive_a[2][i]/iter))
            strings.append(str(productive_b[2][i]/iter))
            strings.append(str(non_productive_a[0][i]/iter))
            strings.append(str(non_productive_b[0][i]/iter))
            strings.append(str(non_productive_a[1][i]/iter))
            strings.append(str(non_productive_b[1][i]/iter))
            strings.append(str(non_productive_a[2][i]/iter))
            strings.append(str(non_productive_b[2][i]/iter))
            strings.append("\n")
            writer1.writerow(strings)
        #pdb.set_trace()
        csv1.close()

        csv2 = open(dir_2, "a", newline='')
        writer2 = csv.writer(csv2, dialect='excel')
        for i in range(0,500):
            strings = list()
            strings.append(str((i+1)))
            strings.append(str(productive_a[0][i]/iter))
            strings.append(str(productive_b[0][i]/iter))
            strings.append(str(productive_a[1][i]/iter))
            strings.append(str(productive_b[1][i]/iter))
            strings.append(str(productive_a[2][i]/iter))
            strings.append(str(productive_b[2][i]/iter))
            strings.append(str(non_productive_a[0][i]/iter))
            strings.append(str(non_productive_b[0][i]/iter))
            strings.append(str(non_productive_a[1][i]/iter))
```

```
            strings.append(str(non_productive_b[1][i]/iter))
            strings.append(str(non_productive_a[2][i]/iter))
            strings.append(str(non_productive_b[2][i]/iter))
            strings.append("\n")
            writer2.writerow(strings)
        #pdb.set_trace()
        csv2.close()

        csv3 = open(dir_3, "a", newline='')
        writer3 = csv.writer(csv3, dialect='excel')
        for i in range(500,cols_count):
            strings = list()
            strings.append(str((i+1)))
            strings.append(str(productive_a[0][i]/iter))
            strings.append(str(productive_b[0][i]/iter))
            strings.append(str(productive_a[1][i]/iter))
            strings.append(str(productive_b[1][i]/iter))
            strings.append(str(productive_a[2][i]/iter))
            strings.append(str(productive_b[2][i]/iter))
            strings.append(str(non_productive_a[0][i]/iter))
            strings.append(str(non_productive_b[0][i]/iter))
            strings.append(str(non_productive_a[1][i]/iter))
            strings.append(str(non_productive_b[1][i]/iter))
            strings.append(str(non_productive_a[2][i]/iter))
            strings.append(str(non_productive_b[2][i]/iter))
            strings.append("\n")
            writer3.writerow(strings)
        #pdb.set_trace()
        csv3.close()
                                                    //pcolor plot of particle matrix
        t = str(iter+1).zfill(5)
        #-----------
        #Particle Graphs
        #-----------
        temp1=iter*0.04
        temp=particle[0][0]
        particle[0][0] = 1
        plt.pcolor(particle, linewidths = 0.2, cmap=cm)
        plt.title("\n".join(wrap("Movement of motors in Scenario 1 at time %0.2f"%temp1 +
' secs', 60)), fontsize=12)
        plt.xlabel('Sites')
        plt.ylabel('Track')
        custom_lines = [Patch(facecolor='white', edgecolor='k',label='Empty Space'),
                        Patch(facecolor='blue', edgecolor='k',label='Motor A (faster)'),
                        Patch(facecolor='red', edgecolor='k',label='Motor B (slower)')]
        plt.legend(handles=custom_lines,loc='center left', bbox_to_anchor=(1, 0.5))
        plt.xlim([0, 1000])
        plt.yticks(np.arange(0, 4, 1))
        plt.gcf().text(.1, 0.01, "Data Values:", fontsize=10)
        plt.gcf().text(.1, -0.04, "Association rate: 1-2 motors/sec", fontsize=8)
        plt.gcf().text(.1, -0.08, "Input::     Motor A: " +str(motor_cargo_a) + "     Motor B: "
```

```
+str(motor_cargo_b), fontsize=8)
        plt.gcf().text(.1, -0.12, "Output::  Motor A: " +str(throughput_a) + "     Motor B: "
+str(throughput_b), fontsize=8)
        plt.gcf().text(.1, -0.16, "Productive Reservoir::         Motor A: " +str(wait_a) + "
Motor B: " +str(wait_b), fontsize=8)
        plt.gcf().text(.1, -0.20, "Non-productive Reservoir::  Motor A: " +str(leak_a) + "
Motor B: " +str(leak_b), fontsize=8)
        plt.tight_layout()
        plt.savefig('Images/Particle_'+t+'.png',  dpi=100, bbox_inches = 'tight')
        plt.draw()
        plt.close()
        print("Graph 1 Done")

        particle[0][0] = temp

        t = str(iter+1).zfill(5)

        #-----------
        #Reservoir Graphs (0-1000)
        #-----------
        temparray1 = [[0 for j in range(cols_count)] for i in range(rows_count)]
        temparray2 = [[0 for j in range(cols_count)] for i in range(rows_count)]
        temparray3 = [[0 for j in range(cols_count)] for i in range(rows_count)]
        temparray4 = [[0 for j in range(cols_count)] for i in range(rows_count)]

        for i in range(rows_count):
            for j in range(cols_count):

                temparray1[i][j] = productive_a[i][j]/iter
                temparray2[i][j] = productive_b[i][j]/iter
                temparray3[i][j] = non_productive_a[i][j]/iter
                temparray4[i][j] = non_productive_b[i][j]/iter

        greek_letterz=[chr(code) for code in range(945,970)]

        #data = pd.read_csv('Scenario 1 Productive and Non-Productive.csv')
        plt.figure(1)
        fig, (ax0, ax1, ax2) = plt.subplots(3, 1)

        ax0.plot(cols1, temparray1[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols1, temparray2[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_xlabel('Corresponding Site', fontsize=15)
        ax0.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax0.set_title('Track 1', fontsize=15)
        ax0.legend(loc="upper right")
        ax0.set_xlim([0, 1000])

        ax1.plot(cols1, temparray1[1], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols1, temparray2[1], alpha=0.5, color="Red", label="Motor B")
        ax1.set_xlabel('Corresponding Site', fontsize=15)
        ax1.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
```

```
        ax1.set_title('Track 2', fontsize=15)
        ax1.legend(loc="upper right")
        ax1.set_xlim([0, 1000])

        ax2.plot(cols1, temparray1[2], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols1, temparray2[2], alpha=0.5, color="Red", label="Motor B")
        ax2.set_xlabel('Corresponding Site', fontsize=15)
        ax2.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax2.set_title('Track 3', fontsize=15)
        ax2.legend(loc="upper right")
        ax2.set_xlim([0, 1000])

        fig.suptitle("\n".join(wrap('Scenario 1 - Mean Distribution (%s)'%greek_letterz[16]
+ ' of Motors in Productive Reservoirs after time %.2f'%temp1 + ' secs', 45)),
fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(bottom=-0.5, right=0.8, top=0.7, hspace = 1.2)
        plt.gcf()
        plt.savefig('Graphs1/Scenario 1(0-1000) Waiting motors_'+t+'.png', bbox_inches =
'tight')
        #plt.show()
        plt.close()

        plt.figure(2)
        fig, (ax0, ax1, ax2) = plt.subplots(3, 1)

        ax0.plot(cols1, temparray3[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols1, temparray4[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_xlabel('Corresponding Site', fontsize=15)
        ax0.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax0.set_title('Track 1', fontsize=15)
        ax0.legend(loc="upper right")
        ax0.set_xlim([0, 1000])

        ax1.plot(cols1, temparray3[1], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols1, temparray4[1], alpha=0.5, color="Red", label="Motor B")
        ax1.set_xlabel('Corresponding Site', fontsize=15)
        ax1.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax1.set_title('Track 2', fontsize=15)
        ax1.legend(loc="upper right")
        ax1.set_xlim([0, 1000])

        ax2.plot(cols1, temparray3[2], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols1, temparray4[2], alpha=0.5, color="Red", label="Motor B")
        ax2.set_xlabel('Corresponding Site', fontsize=15)
        ax2.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax2.set_title('Track 3', fontsize=15)
        ax2.legend(loc="upper right")
        ax2.set_xlim([0, 1000])

        fig.suptitle("\n".join(wrap('Scenario 1 - Mean Distribution (%s)'%greek_letterz[16]
```

```
+ ' of Motors in Non-Productive Reservoirs after time %.2f'%temp1 + ' secs', 45)),
fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(bottom=-0.5, right=0.8, top=0.7, hspace = 1.2)
        plt.gcf()
        plt.savefig('Graphs1/Scenario 1(0-1000) Leakage motors_'+t+'.png', bbox_inches =
'tight')
        #plt.show()
        plt.close()
        print("Graph 2 Done")

        #-----------
        #Reservoir Graphs (0-500)
        #-----------
        temparray1 = [[0 for j in range(500)] for i in range(rows_count)]
        temparray2 = [[0 for j in range(500)] for i in range(rows_count)]
        temparray3 = [[0 for j in range(500)] for i in range(rows_count)]
        temparray4 = [[0 for j in range(500)] for i in range(rows_count)]

        for i in range(rows_count):
           for j in range(500):

               temparray1[i][j] = productive_a[i][j]/iter
               temparray2[i][j] = productive_b[i][j]/iter
               temparray3[i][j] = non_productive_a[i][j]/iter
               temparray4[i][j] = non_productive_b[i][j]/iter

        greek_letterz=[chr(code) for code in range(945,970)]

        #data = pd.read_csv('Scenario 1 Productive and Non-Productive (0-500).csv')
        plt.figure(1)
        fig, (ax0, ax1, ax2) = plt.subplots(3, 1)

        ax0.plot(cols2, temparray1[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols2, temparray2[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_xlabel('Corresponding Site', fontsize=15)
        ax0.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax0.set_title('Track 1', fontsize=15)
        ax0.legend(loc="upper right")
        ax0.set_xlim([0, 500])

        ax1.plot(cols2, temparray1[1], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols2, temparray2[1], alpha=0.5, color="Red", label="Motor B")
        ax1.set_xlabel('Corresponding Site', fontsize=15)
        ax1.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax1.set_title('Track 2', fontsize=15)
        ax1.legend(loc="upper right")
        ax1.set_xlim([0, 500])

        ax2.plot(cols2, temparray1[2], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols2, temparray2[2], alpha=0.5, color="Red", label="Motor B")
```

```
        ax2.set_xlabel('Corresponding Site', fontsize=15)
        ax2.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax2.set_title('Track 3', fontsize=15)
        ax2.legend(loc="upper right")
        ax2.set_xlim([0, 500])

        fig.suptitle("\n".join(wrap('Scenario 1 - Mean Distribution (%s)'%greek_letterz[16]
+ ' of Motors in Productive Reservoirs in the minus end of track after time %.2f'%temp1
+ ' secs', 45)), fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(bottom=-0.5, right=0.8, top=0.7, hspace = 1.2)
        plt.gcf()
        plt.savefig('Graphs2/Scenario 1(0-500) Waiting motors_'+t+'.png', bbox_inches =
'tight')
        #plt.show()
        plt.close()

        plt.figure(2)
        fig, (ax0, ax1, ax2) = plt.subplots(3, 1)

        ax0.plot(cols2, temparray3[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols2, temparray4[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_xlabel('Corresponding Site', fontsize=15)
        ax0.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax0.set_title('Track 1', fontsize=15)
        ax0.legend(loc="upper right")
        ax0.set_xlim([0, 500])

        ax1.plot(cols2, temparray3[1], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols2, temparray4[1], alpha=0.5, color="Red", label="Motor B")
        ax1.set_xlabel('Corresponding Site', fontsize=15)
        ax1.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax1.set_title('Track 2', fontsize=15)
        ax1.legend(loc="upper right")
        ax1.set_xlim([0, 500])

        ax2.plot(cols2, temparray3[2], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols2, temparray4[2], alpha=0.5, color="Red", label="Motor B")
        ax2.set_xlabel('Corresponding Site', fontsize=15)
        ax2.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax2.set_title('Track 3', fontsize=15)
        ax2.legend(loc="upper right")
        ax2.set_xlim([0, 500])

        fig.suptitle("\n".join(wrap('Scenario 1 - Mean Distribution (%s)'%greek_letterz[16]
+ ' of Motors in Non-Productive Reservoirs in the minus end of track after time
%.2f'%temp1 + ' secs', 45)), fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(bottom=-0.5, right=0.8, top=0.7, hspace = 1.2)
        plt.gcf()
        plt.savefig('Graphs2/Scenario 1(0-500) Leakage motors_'+t+'.png', bbox_inches =
```

```
'tight')
        #plt.show()
        plt.close()
        print("Graph 3 Done")

        #-----------
        #Reservoir Graphs (500-1000)
        #-----------
        temparray1 = [[0 for j in range(500)] for i in range(rows_count)]
        temparray2 = [[0 for j in range(500)] for i in range(rows_count)]
        temparray3 = [[0 for j in range(500)] for i in range(rows_count)]
        temparray4 = [[0 for j in range(500)] for i in range(rows_count)]

        for i in range(rows_count):
            for j in range(500):
                temparray1[i][j] = productive_a[i][j+500]/iter
                temparray2[i][j] = productive_b[i][j+500]/iter
                temparray3[i][j] = non_productive_a[i][j+500]/iter
                temparray4[i][j] = non_productive_b[i][j+500]/iter

        greek_letterz=[chr(code) for code in range(945,970)]

        #data = pd.read_csv('Scenario 1 Productive and Non-Productive (500-1000).csv')
        plt.figure(1)
        fig, (ax0, ax1, ax2) = plt.subplots(3, 1)

        ax0.plot(cols3, temparray1[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols3, temparray2[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_xlabel('Corresponding Site', fontsize=15)
        ax0.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax0.set_title('Track 1', fontsize=15)
        ax0.legend(loc="upper right")
        ax0.set_xlim([500, 1000])

        ax1.plot(cols3, temparray1[1], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols3, temparray2[1], alpha=0.5, color="Red", label="Motor B")
        ax1.set_xlabel('Corresponding Site', fontsize=15)
        ax1.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax1.set_title('Track 2', fontsize=15)
        ax1.legend(loc="upper right")
        ax1.set_xlim([500, 1000])

        ax2.plot(cols3, temparray1[2], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols3, temparray2[2], alpha=0.5, color="Red", label="Motor B")
        ax2.set_xlabel('Corresponding Site', fontsize=15)
        ax2.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax2.set_title('Track 3', fontsize=15)
        ax2.legend(loc="upper right")
        ax2.set_xlim([500, 1000])

        fig.suptitle("\n".join(wrap('Scenario 1 - Mean Distribution (%s)'%greek_letterz[16]
```

```
+ ' of Motors in Productive Reservoirs in the plus end of track after time %.2f'%temp1 + '
secs', 45)), fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(bottom=-0.5, right=0.8, top=0.7, hspace = 1.2)
        plt.gcf()
        plt.savefig('Graphs3/Scenario 1(500-1000) Waiting motors_'+t+'.png', bbox_inches
= 'tight')
        #plt.show()
        plt.close()

        plt.figure(2)
        fig, (ax0, ax1, ax2) = plt.subplots(3, 1)

        ax0.plot(cols3, temparray3[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols3, temparray4[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_xlabel('Corresponding Site', fontsize=15)
        ax0.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax0.set_title('Track 1', fontsize=15)
        ax0.legend(loc="upper right")
        ax0.set_xlim([500, 1000])

        ax1.plot(cols3, temparray3[1], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols3, temparray4[1], alpha=0.5, color="Red", label="Motor B")
        ax1.set_xlabel('Corresponding Site', fontsize=15)
        ax1.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax1.set_title('Track 2', fontsize=15)
        ax1.legend(loc="upper right")
        ax1.set_xlim([500, 1000])

        ax2.plot(cols3, temparray3[2], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols3, temparray4[2], alpha=0.5, color="Red", label="Motor B")
        ax2.set_xlabel('Corresponding Site', fontsize=15)
        ax2.set_ylabel("\n".join(wrap('No of Motors', 22)), fontsize=15)
        ax2.set_title('Track 3', fontsize=15)
        ax2.legend(loc="upper right")
        ax2.set_xlim([500, 1000])

        fig.suptitle("\n".join(wrap('Scenario 1 - Mean Distribution (%s)'%greek_letterz[16]
+ ' of Motors in Non-Productive Reservoirs in the plus end of track after time
%.2f'%temp1 + ' secs', 45)), fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(bottom=-0.5, right=0.8, top=0.7, hspace = 1.2)
        plt.gcf()
        plt.savefig('Graphs3/Scenario 1(500-1000) Leakage motors_'+t+'.png', bbox_inches
= 'tight')
        #plt.show()
        plt.close()
        print("Graph 3 Done")


workbook.close()
```

## *2.9* *HEATMAP GENERATION:*

```
fig, (ax0, ax1, cax) = plt.subplots(ncols=3,figsize=(5.5,3),
                gridspec_kw={"width_ratios":[1,1, 0.05]})

top = max(max(max(heat_map_a_3)),max(max(heat_map_b_3)));

time = (total_runtime-1)*time_stamp
c0 = ax0.pcolor(heat_map_a_3, linewidths = 0.02, cmap='hot_r', vmin=0 , vmax=top)
ax0.set_title("Faster motor", fontsize=12)
ax0.set_xlabel('Sites')
ax0.set_ylabel('Track')

c1 = ax1.pcolor(heat_map_b_3, linewidths = 0.02, cmap='hot_r', vmin=0 , vmax=top)
ax1.set_title("Slower motor", fontsize=12)
ax1.set_xlabel('Sites')

plt.setp(ax0, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])
plt.setp(ax1, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])


fig.suptitle("\n".join(wrap("Distribution heat map of motors (Scenario 1 - With
processivity) after %0.2f"%time + ' secs', 70)), fontsize=12, y=1.1)
fig.tight_layout()
fig.colorbar(c0, cax=cax)
fig.subplots_adjust(wspace=0.3)
plt.draw()
plt.savefig('1 Hot Heatmap.eps', format='eps', dpi=600, bbox_inches = 'tight')
plt.savefig('1 Hot Heatmap.svg', format='svg', dpi=600, bbox_inches = 'tight')
plt.savefig('1 Hot Heatmap.png', format='png', dpi=600, bbox_inches = 'tight')

plt.close()


fig, (ax0, ax1, cax) = plt.subplots(ncols=3,figsize=(5.5,3),
                gridspec_kw={"width_ratios":[1,1, 0.05]})

top = max(max(max(heat_map_a_3)),max(max(heat_map_b_3)));

#time=total_runtime*time_step
c0 = ax0.pcolor(heat_map_a_3, linewidths = 0.02, cmap='viridis_r', vmin=0 , vmax=top)
ax0.set_title("Faster motor", fontsize=12)
ax0.set_xlabel('Sites')
```

```
ax0.set_ylabel('Track')

c1 = ax1.pcolor(heat_map_b_3, linewidths = 0.02, cmap='viridis_r', vmin=0 , vmax=top)
ax1.set_title("Slower motor", fontsize=12)
ax1.set_xlabel('Sites')

plt.setp(ax0, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])
plt.setp(ax1, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])


fig.suptitle("\n".join(wrap("Distribution heat map of motors (Scenario 1 - With
processivity) after %0.2f"%time + ' secs', 70)), fontsize=12, y=1.1)
fig.tight_layout()
fig.colorbar(c0, cax=cax)
fig.subplots_adjust(wspace=0.3)
plt.draw()
plt.savefig('1 Viridis Heatmap.eps', format='eps', dpi=600, bbox_inches = 'tight')
plt.savefig('1 Viridis Heatmap.svg', format='svg', dpi=600, bbox_inches = 'tight')
plt.savefig('1 Viridis Heatmap.png', format='png', dpi=600, bbox_inches = 'tight')
plt.close()

print("Heatmap Graph Done")
```

## *2.10* *GIF CREATION:*

### *2.10.1* *Particle plots:*

```
# filepaths
fp_in = "Images/Particle_*.png"
fp_out = "GIFs/1 Particle.gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=500, loop=0)
```

### *2.10.2* *Productive and Non-Productive Reservoirs (for sites 0-1000):*

```
# filepaths
fp_in = "Graphs1/Scenario 1(0-1000) Waiting motors_*.png"
fp_out = "GIFs/1 Productive Reservoirs(0-1000).gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
```

```
        save_all=True, duration=200, loop=0)

# filepaths
fp_in = "Graphs1/Scenario 1(0-1000) Leakage motors_*.png"
fp_out = "GIFs/1 Non-Productive Reservoirs(0-1000).gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=200, loop=0)
```

### ***2.10.3*** *Productive and Non-Productive Reservoirs (for sites 0-1000):*

```
# filepaths
fp_in = "Graphs2/Scenario 1(0-500) Waiting motors_*.png"
fp_out = "GIFs/1 Productive Reservoirs(0-500).gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=200, loop=0)

# filepaths
fp_in = "Graphs2/Scenario 1(0-500) Leakage motors_*.png"
fp_out = "GIFs/1 Non-Productive Reservoirs(0-500).gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=200, loop=0)
```

### ***2.10.4*** *Productive and Non-Productive Reservoirs (for sites 0-1000):*

```
# filepaths
fp_in = "Graphs3/Scenario 1(500-1000) Waiting motors_*.png"
fp_out = "GIFs/1 Productive Reservoirs(500-1000).gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=200, loop=0)

# filepaths
fp_in = "Graphs3/Scenario 1(500-1000) Leakage motors_*.png"
fp_out = "GIFs/1 Non-Productive Reservoirs(500-1000).gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
```

```
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=200, loop=0)
```

### ***2.10.5*** *Productive Reservoir Dynamics at Regular Intervals:*

```
# filepaths
fp_in = "Reservoir1/Productive_*.png"
fp_out = "GIFs/1 Dynamic Productive.gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=500, loop=0)
```

### ***2.10.6*** *Cumulative Productive Reservoir Dynamics at Regular Intervals:*

```
# filepaths
fp_in = "Reservoir2/Productive_*.png"
fp_out = "GIFs/1 Cumulative Productive.gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
        save_all=True, duration=500, loop=0)
```

# 3. Modifications of codes for different Scenarios:

## *3.1* *Scenario 2 (no provision for lateral movement):*

The transport function is varied only. The motors on crowding are restricted to move into adjacent track and are directly detached into productive reservoirs.

Code:

### *3.1.1* Motor traversing along the same track or movement into lateral track: (TASEP model)

```
def transport(particle, velocity, span, lifetime_cells, rows_count, cols_count, vel_a, vel_b, process_a, process_b, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a, detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b, detach_process_a, detach_process_b, time):
    start=timer()

    for i in range(rows_count):
        gap=0                                          //measuring gap between two motors
        minimum=0
        for j in range(cols_count-1,-1,-1):
            if(particle[i][j] == 0):
                gap += 1
                continue
            else:
                #Output
                if(particle[i][j] == 1):
                    if(gap < vel_a):
                        minimum = 0
                    elif(gap >= vel_a):
                        minimum = vel_a

                #Constant velocity
                elif(particle[i][j] == 2):
                    if(gap < vel_b):
                        minimum = 0
                    elif(gap >= vel_b):
                        minimum = vel_b

                                        //if sufficient gap is present then motor moves with
                                        constant velocity along the microtubular track

                #Movement of motor along the parallel track
                if(minimum > 0):
                    if(particle[i][j] == 1):                    //delivery of motor A
                        if((j + vel_a) >= cols_count-1):
                            throughput_a += 1
```

```
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

        else:                                   //movement of motor A from one site to other
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 1
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            if(span[i][j+minimum] >= process_a):
                #detachment if motor exceeds its processivity
                velocity[i][j+minimum] = 0
                particle[i][j+minimum] = 0
                span[i][j+minimum] = 0
                print('y1')
                if(reservoir_a[i][j].qsize()==reservoir_length):
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1

                reservoir_a[i][j].put(props(time,lifetime_cells[i][j+minimum]))

                lifetime_cells[i][j+minimum]=0
                detachment_a[i][j+minimum] += 1
                detachment_matrix_a[i][j+minimum] = 1
                detach_a +=1
                detach_process_a += 1

    elif(particle[i][j] == 2):                          //delivery of motor B
        if((j + vel_b) >= cols_count-1):
            throughput_b += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

        else:                                   //movement of motor B from one site to other
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 2
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
```

```
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            if(span[i][j+minimum] >= process_b):
               #detachment if motor exceeds its processivity
               velocity[i][j+minimum] = 0
               particle[i][j+minimum] = 0
               span[i][j+minimum] = 0
               print('y2')
               if(reservoir_b[i][j].qsize()==reservoir_length):
                  reservoir_b[i][j].get()
                  leakage_b[i][j] += 1

               reservoir_b[i][j].put(props(time,lifetime_cells[i][j+minimum]))

               lifetime_cells[i][j+minimum]=0
               detachment_b[i][j+minimum] += 1
               detachment_matrix_b[i][j+minimum] = 1
               detach_b +=1
               detach_process_b += 1

#Detachment of motor
elif(minimum == 0):                                    //no lateral movement
   if(particle[i][j] == 1):
      # Lateral association will happen to only cargos lost at initial stages
      channels = list(range(0,rows_count))       # All channels
      channels.remove(i)
      random.shuffle(channels)
      y1 = channels[0]
      channels.remove(channels[0])
      y2 = channels[0]

      if((j + vel_a) >= cols_count-1):
         throughput_a += 1
         velocity[i][j] = 0
         particle[i][j] = 0
         span[i][j] = 0
         lifetime_cells[i][j] = 0

      else:
         velocity[i][j] = 0
         particle[i][j] = 0
         span[i][j] = 0
         print('y3')
         if(reservoir_a[i][j].qsize()  ==  reservoir_length):
            reservoir_a[i][j].get()
            leakage_a[i][j] += 1
```

```
                    reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

                    lifetime_cells[i][j]=0
                    detachment_a[i][j] += 1
                    detachment_matrix_a[i][j] = 1
                    detach_a += 1

                elif(particle[i][j] == 2):
                    channels = list(range(0,rows_count))        # All channels
                    channels.remove(i)
                    random.shuffle(channels)
                    y1 = channels[0]
                    channels.remove(channels[0])
                    y2 = channels[0]

                    if((j + vel_b) >= cols_count-1):
                        throughput_b += 1
                        velocity[i][j] = 0
                        particle[i][j] = 0
                        span[i][j] = 0
                        lifetime_cells[i][j] = 0

                    else:
                        velocity[i][j] = 0
                        particle[i][j] = 0
                        span[i][j] = 0
                        print('y4')
                        if(reservoir_b[i][j].qsize()==reservoir_length):
                            reservoir_b[i][j].get()
                            leakage_b[i][j] += 1

                        reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))

                        lifetime_cells[i][j]=0
                        detachment_b[i][j] += 1
                        detachment_matrix_b[i][j] = 1
                        detach_b += 1

            gap = 0
    end = timer()
    print("y take", end - start)
    #pdb.set_trace()
    return particle, velocity, span, lifetime_cells, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detachment_matrix_a, detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b, detach_process_a, detach_process_b
```

## *3.2* *Scenario 3-5 (staggering at the initial segment of MT track):*

The association is restricted to middle track only. The transport function has few changes too. A part of the adjacent tracks is blocked creating staggered tracks. The motors are restricted to move into single middle track till staggering point is crossed. On crossing this point, the faster motor laterally move to the adjacent track.

Code:

### *3.2.1* Motor-cargo loading on minus end of microtubular track:

```
def initial_association(particle, velocity, span, lifetime_cells, input_res, motor_cargo_a,
motor_cargo_b):

    start = timer()
                                                    //attachment on microtubular track
    c1 = 1                                          //attachment restricted to middle track only
    if(particle[c1][0] == 0):
        particle[c1][0] = input_res.get()
        span[c1][0] = 1
        lifetime_cells[c1][0] = 1

        if(particle[c1][0] == 1):
            motor_cargo_a += 1
            velocity[c1][0] = vel_a

        elif(particle[c1][0] == 2):
            motor_cargo_b += 1
            velocity[c1][0] = vel_b

    end = timer()
    print("z take", end - start)
    #pdb.set_trace()

    return particle, velocity, span, lifetime_cells, input_res, motor_cargo_a,motor_cargo_b
```

### *3.2.2* Detachment and Reattachment dynamics of motor in productive reservoir: (LK model)

```
def lateral_association(particle, velocity, span, lifetime_cells, rows_count, cols_count,
reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
reattach_a_1, reattach_b_1, reattach_a_2, reattach_b_2, reattachment_matrix_a,
reattachment_matrix_b, leakage_a, leakage_b, time):

    start = timer()
    i=0
    j=0
    k=0
    temp = 0
```

```
track = 0
site = 0
flag = 0
flag2 = 0
step = 0
step1 = 0
for i in range(rows_count):
    for j in range(cols_count):
        if(reservoir_a[i][j].qsize() != 0 and reservoir_b[i][j].qsize() == 0):
            flag = 0
        elif(reservoir_a[i][j].qsize() == 0 and reservoir_b[i][j].qsize() != 0):
            flag = 1
        elif(reservoir_a[i][j].qsize() != 0 and reservoir_b[i][j].qsize() != 0):
            flag = random.randint(0,1)
        else:
            flag = 10

        if(flag == 0):
            t1 = reservoir_a[i][j].queue[0].joins
            t2 = reservoir_a[i][j].queue[0].life

                                        //if motors A have reached beyond lifetime, then
                                        they are pushed into non-productive reservoir

            if(time-t1+t2 > lifetime):
                #remove the element as leakage
                #print("Leak A")
                reservoir_a[i][j].get()
                leakage_a[i][j] += 1
            else:                       //motors A can reattach into any one of the nine
                                        neighboring sites

                channels = list(range(0,rows_count))
                channels.remove(i)
                random.shuffle(channels)
                track1 = channels[0]
                track2 = channels[1]
                track = j
                if(j==cols_count-1):
                    if(particle[i][j] == 0): site = j; track = i;
                    elif(particle[i][j-1] == 0): site = j-1; track = i;
                    elif(particle[track1][j] == 0): site = j; track = track1;
                    elif(particle[track1][j-1] == 0): site = j-1; track = track1;
                    elif(particle[track2][j] == 0): site = j; track = track2;
                    elif(particle[track2][j-1] == 0): site = j-1; track = track2;
                    else:  continue;
                else:
                                                    //staggered tracks
                    if(j<=(stagger/100)*cols_count):
                        if(particle[i][j+1] == 0): site = j+1; track = i;
                        elif(particle[i][j] == 0): site = j; track = i;
```

```
                elif(particle[i][j-1] == 0): site = j-1; track = i;
                else: continue
            elif(j>(stagger/100)*cols_count):
                if(particle[i][j+1] == 0): site = j+1; track = i;
                elif(particle[track1][j+1] == 0): site = j+1; track = track1;
                elif(particle[track2][j+1] == 0): site = j+1; track = track2;
                elif(particle[i][j] == 0): site = j; track = i;
                elif(particle[track1][j] == 0): site = j; track = track1;
                elif(particle[track2][j] == 0): site = j; track = track2;
                elif(particle[i][j-1] == 0): site = j-1; track = i;
                elif(particle[track1][j-1] == 0): site = j-1; track = track1;
                elif(particle[track2][j-1] == 0): site = j-1; track = track2;
                else: continue;

            particle[track][site] = 1
            velocity[track][site] = vel_a
            span[track][site] = 1

            t = reservoir_a[i][j].get()
            lifetime_cells[i][site] = time-t.joins+t.life
            #print("Reattach A")
            reattachment_a[track][site] += 1
            reattachment_matrix_a[track][site] = 1
            reattach_a += 1
            if(j<staggered_cells):
                reattach_a_1 += 1
            else:
                reattach_a_2 += 1

    elif(flag == 1):
        t1 = reservoir_b[i][j].queue[0].joins
        t2 = reservoir_b[i][j].queue[0].life

                                        //if motors B have reached beyond lifetime, then
                                        they are pushed into non-productive reservoir

        if(time-t1+t2 > lifetime):
            #remove the element as leakage
            #print("Leak B")
            reservoir_b[i][j].get()
            leakage_b[i][j] += 1
        else:
                                        //motors B can reattach into any one of the nine
                                        neighboring sites

            channels = list(range(0,rows_count))
            channels.remove(i)
            random.shuffle(channels)
            track1 = channels[0]
            track2 = channels[1]
            track = j
```

```
                if(j==cols_count-1):
                    if(particle[i][j] == 0): site = j; track = i;
                    elif(particle[i][j-1] == 0): site = j-1; track = i;
                    elif(particle[track1][j] == 0): site = j; track = track1;
                    elif(particle[track1][j-1] == 0): site = j-1; track = track1;
                    elif(particle[track2][j] == 0): site = j; track = track2;
                    elif(particle[track2][j-1] == 0): site = j-1; track = track2;
                    else:  continue;
                else:
                                                        //staggered tracks
                    if(j<=(stagger/100)*cols_count):
                        if(particle[i][j+1] == 0): site = j+1; track = i;
                        elif(particle[i][j] == 0): site = j; track = i;
                        elif(particle[i][j-1] == 0): site = j-1; track = i;
                        else: continue
                    elif(j>(stagger/100)*cols_count):
                        if(particle[i][j+1] == 0): site = j+1; track = i;
                        elif(particle[track1][j+1] == 0): site = j+1; track = track1;
                        elif(particle[track2][j+1] == 0): site = j+1; track = track2;
                        elif(particle[i][j] == 0): site = j; track = i;
                        elif(particle[track1][j] == 0): site = j; track = track1;
                        elif(particle[track2][j] == 0): site = j; track = track2;
                        elif(particle[i][j-1] == 0): site = j-1; track = i;
                        elif(particle[track1][j-1] == 0): site = j-1; track = track1;
                        elif(particle[track2][j-1] == 0): site = j-1; track = track2;
                        else: continue;

                particle[track][site] = 2
                velocity[track][site] = vel_b
                span[track][site] = 1

                t=reservoir_b[i][j].get()
                lifetime_cells[i][site] = time-t.joins+t.life
                #print("Reattach B")
                reattachment_b[track][site] += 1
                reattachment_matrix_b[track][site] = 1
                reattach_b += 1
                if(j<staggered_cells):
                    reattach_b_1 += 1
                else:
                    reattach_b_2 += 1
            else:
                continue

    end = timer()
    print("x take", end - start)
    #pdb.set_trace()
    return particle, velocity, span, lifetime_cells, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, reattach_a_1, reattach_b_1, reattach_a_2,
reattach_b_2, reattachment_matrix_a, reattachment_matrix_b, leakage_a, leakage_b
```

### *3.2.3* Motor traversing along the same track or movement into lateral track: (TASEP model)

```
def transport (rows_count, cols_count, vel_a, vel_b, process_a, process_b, throughput_a,
throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a, detach_b,
detachment_matrix_a, detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b,
time):

    start=timer()

    for i in range(rows_count):
        gap=0                                                    //check gap between motors
        minimum=0
        for j in range(cols_count-1,-1,-1):
            if(particle[i][j] == 0):
                gap += 1
                continue
            else:
                #Output
                if(particle[i][j] == 1):
                    if(gap < vel_a):
                        minimum = 0
                    elif(gap >= vel_a):
                        minimum = vel_a

                #Constant velocity
                elif(particle[i][j] == 2):
                    if(gap < vel_b):
                        minimum = 0
                    elif(gap >= vel_b):
                        minimum = vel_b

                                        //if sufficient gap is present then motor moves with
                                        constant velocity along the microtubular track

                #Movement
                if(minimum > 0):
                    if(particle[i][j] == 1):
                        if((j + vel_a) >= cols_count-1):          //delivery of motor A
                            throughput_a += 1
                            velocity[i][j] = 0
                            particle[i][j] = 0
                            span[i][j] = 0
                            lifetime_cells[i][j] = 0

                        else:                                //motor A move from one site to another
                            velocity[i][j+minimum] = minimum
                            particle[i][j+minimum] = 1
                            span[i][j+minimum] = span[i][j] + minimum
                            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
                            velocity[i][j] = 0
```

```
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

            if(span[i][j+minimum] >= process_a):
               #detachment if motor exceeds its processivity
               velocity[i][j+minimum] = 0
               particle[i][j+minimum] = 0
               span[i][j+minimum] = 0
               print('y1')
               if(reservoir_a[i][j].qsize()==reservoir_length):
                  reservoir_a[i][j].get()
                  leakage_a[i][j] += 1

               reservoir_a[i][j].put(props(time,lifetime_cells[i][j+minimum]))

               lifetime_cells[i][j+minimum]=0
               detachment_a[i][j+minimum] += 1
               detachment_matrix_a[i][j+minimum] = 1
               detach_a +=1
               if(j<staggered_cells):
                  detach_process_a_1 += 1
               else:
                  detach_process_a_2 += 1

      elif(particle[i][j] == 2):
         if((j + vel_b) >= cols_count-1):                    //delivery of motor B
            throughput_b += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

         else:                                    //motor B move from one site to another
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 2
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

            if(span[i][j+minimum] >= process_b):
               #detachment if motor exceeds its processivity
               velocity[i][j+minimum] = 0
               particle[i][j+minimum] = 0
               span[i][j+minimum] = 0
               print('y2')
               if(reservoir_b[i][j].qsize()==reservoir_length):
                  reservoir_b[i][j].get()
```

```
                leakage_b[i][j] += 1

            reservoir_b[i][j].put(props(time,lifetime_cells[i][j+minimum]))

            lifetime_cells[i][j+minimum]=0
            detachment_b[i][j+minimum] += 1
            detachment_matrix_b[i][j+minimum] = 1
            detach_b +=1
            if(j<staggered_cells):
                detach_process_b_1 += 1
            else:
                detach_process_b_2 += 1

#Detachment or Lateral Movement
elif(minimum == 0):
                                    //the microtubular track beyond the staggering point
                                    allows provision for lateral movement
  if(particle[i][j] == 1):
      if(j>(stagger/100)*cols_count):
          # Lateral association will happen to only cargos lost at initial stages
          channels = list(range(0,rows_count))       # All channels
          channels.remove(i)
          random.shuffle(channels)
          y1 = channels[0]
          channels.remove(channels[0])
          y2 = channels[0]

          if((j + vel_a) >= cols_count-1):
              throughput_a += 1
              velocity[i][j] = 0
              particle[i][j] = 0
              span[i][j] = 0
              lifetime_cells[i][j] = 0
              lateral_a += 1
          else:
              if(particle[y1][j+vel_a] == 0):
                  velocity[y1][j+vel_a] = vel_a
                  particle[y1][j+vel_a] = 1
                  span[y1][j+vel_a] = span[i][j] + vel_a
                  lifetime_cells[y1][j+vel_a] = lifetime_cells[i][j]
                  velocity[i][j] = 0
                  particle[i][j] = 0
                  span[i][j] = 0
                  lifetime_cells[i][j] = 0
                  lateral_a += 1

              elif(particle[y2][j+vel_a] == 0):
                  velocity[y2][j+vel_a] = vel_a
                  particle[y2][j+vel_a] = 1
                  span[y2][j+vel_a] = span[i][j] + vel_a
                  lifetime_cells[y2][j+vel_a] = lifetime_cells[i][j]
```

```
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_a += 1

            else:
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                #print('y3')
                if(reservoir_a[i][j].qsize()==reservoir_length):
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1

                reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

                lifetime_cells[i][j]=0
                detachment_a[i][j] += 1
                detachment_matrix_a[i][j] = 1
                detach_a += 1
                if(j<staggered_cells):
                    detach_a_1 += 1
                else:
                    detach_a_2 += 1

                                        //the initial portion of microtubular track
                                        being blocked restricts lateral movement

        elif(j<=(stagger/100)*cols_count):
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            #print('y3')
            if(reservoir_a[i][j].qsize()==reservoir_length):
                reservoir_a[i][j].get()
                leakage_a[i][j] += 1

            reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

            lifetime_cells[i][j]=0
            detachment_a[i][j] += 1
            detachment_matrix_a[i][j] = 1
            detach_a += 1
            if(j<staggered_cells):
                detach_a_1 += 1
            else:
                detach_a_2 += 1

    elif(particle[i][j] == 2):
                                //the microtubular track beyond the staggering point
```

allows provision for lateral movement

```
if(j>(stagger/100)*cols_count):
    channels = list(range(0,rows_count))        # All channels
    channels.remove(i)
    random.shuffle(channels)
    y1 = channels[0]
    channels.remove(channels[0])
    y2 = channels[0]

    if((j + vel_b) >= cols_count-1):
        throughput_b += 1
        velocity[i][j] = 0
        particle[i][j] = 0
        span[i][j] = 0
        lifetime_cells[i][j] = 0
        lateral_b += 1
    else:
        if(particle[y1][j+vel_b]== 0):
            velocity[y1][j+vel_b] = vel_b
            particle[y1][j+vel_b] = 2
            span[y1][j+vel_b] = span[i][j] + vel_b
            lifetime_cells[y1][j+vel_b] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            lateral_b += 1

        elif(particle[y2][j+vel_b]== 0):
            velocity[y2][j+vel_b] = vel_b
            particle[y2][j+vel_b] = 2
            span[y2][j+vel_b] = span[i][j] + vel_b
            lifetime_cells[y2][j+vel_b] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            lateral_b += 1

        else:
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            #print('y4')
            if(reservoir_b[i][j].qsize()==reservoir_length):
                reservoir_b[i][j].get()
                leakage_b[i][j] += 1

            reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))
```

```
                        lifetime_cells[i][j]=0
                        detachment_b[i][j] += 1
                        detachment_matrix_b[i][j] = 1
                        detach_b += 1
                        if(j<staggered_cells):
                            detach_b_1 += 1
                        else:
                            detach_b_2 += 1

                                                    //the initial portion of microtubular track
                                                    being blocked restricts lateral movement

                    elif(j<=(stagger/100)*cols_count):
                        velocity[i][j] = 0
                        particle[i][j] = 0
                        span[i][j] = 0
                        #print('y4')
                        if(reservoir_b[i][j].qsize()==reservoir_length):
                            reservoir_b[i][j].get()
                            leakage_b[i][j] += 1

                        reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))

                        lifetime_cells[i][j]=0
                        detachment_b[i][j] += 1
                        detachment_matrix_b[i][j] = 1
                        detach_b += 1
                        if(j<staggered_cells):
                            detach_b_1 += 1
                        else:
                            detach_b_2 += 1

                gap = 0

        end = timer()
        print("y take", end - start)
        #pdb.set_trace()
        return particle, velocity, span, lifetime_cells, throughput_a, throughput_b, reservoir_a,
reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detach_a_1, detach_b_1,
detach_a_2, detach_b_2, detachment_matrix_a, detachment_matrix_b, leakage_a,
leakage_b, lateral_a, lateral_b, detach_process_a, detach_process_b, detach_process_a_1,
detach_process_b_1, detach_process_a_2, detach_process_b_2
```

## *3.3* *Scenario 6-8 (staggering at the latter/distil segment of MT track):*

Even though the association occurs equally in three tracks, in the transport function beyond a certain staggering point, the adjacent tracks are blocked and the movement is channelized into one single middle track.

Code:

### *3.3.1* Detachment and Reattachment dynamics of motor in productive reservoir: (LK model)

```
def lateral_association(particle, velocity, span, lifetime_cells, rows_count, cols_count,
reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
reattach_a_1, reattach_b_1, reattach_a_2, reattach_b_2, reattachment_matrix_a,
reattachment_matrix_b, leakage_a, leakage_b, time):

    start = timer()
    i=0
    j=0
    k=0
    temp = 0
    track = 0
    site = 0
    flag = 0
    flag2 = 0
    step = 0
    step1 = 0
    for i in range(rows_count):
        for j in range(cols_count):
            if(reservoir_a[i][j].qsize() != 0 and reservoir_b[i][j].qsize() == 0):
                flag = 0
            elif(reservoir_a[i][j].qsize() == 0 and reservoir_b[i][j].qsize() != 0):
                flag = 1
            elif(reservoir_a[i][j].qsize() != 0 and reservoir_b[i][j].qsize() != 0):
                flag = random.randint(0,1)
            else:
                flag = 10

            if(flag == 0):
                t1 = reservoir_a[i][j].queue[0].joins
                t2 = reservoir_a[i][j].queue[0].life

                                        //if motors A have reached beyond lifetime, then
                                        they are pushed into non-productive reservoir

                if(time-t1+t2 > lifetime):
                    #remove the element as leakage
                    #print("Leak A")
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1
                else:                   //motors A can reattach into any one of the nine
```

neighboring sites

```
channels = list(range(0,rows_count))
channels.remove(i)
random.shuffle(channels)
track1 = channels[0]
track2 = channels[1]
track = j
if(i==1):
    if(j==cols_count-1):
        if(particle[i][j] == 0): site = j; track = i;
        elif(particle[i][j-1] == 0): site = j-1; track = i;
        else:  continue;
    else:
        if(particle[i][j+1] == 0): site = j+1; track = i;
        elif(particle[i][j] == 0): site = j; track = i;
        elif(particle[i][j-1] == 0): site = j-1; track = i;
        else: continue
else:
                                                    //staggered tracks
    if(j<=(stagger/100)*cols_count):
        if(particle[i][j+1] == 0): site = j+1; track = i;
        elif(particle[i][j] == 0): site = j; track = i;
        elif(particle[i][j-1] == 0): site = j-1; track = i;
        else: continue
    elif(j>(stagger/100)*cols_count):
        if(particle[i][j+1] == 0): site = j+1; track = i;
        elif(particle[track1][j+1] == 0): site = j+1; track = track1;
        elif(particle[track2][j+1] == 0): site = j+1; track = track2;
        elif(particle[i][j] == 0): site = j; track = i;
        elif(particle[track1][j] == 0): site = j; track = track1;
        elif(particle[track2][j] == 0): site = j; track = track2;
        elif(particle[i][j-1] == 0): site = j-1; track = i;
        elif(particle[track1][j-1] == 0): site = j-1; track = track1;
        elif(particle[track2][j-1] == 0): site = j-1; track = track2;
        else: continue;

particle[track][site] = 1
velocity[track][site] = vel_a
span[track][site] = 1

t = reservoir_a[i][j].get()
lifetime_cells[i][site] = time-t.joins+t.life
#print("Reattach A")
reattachment_a[track][site] += 1
reattachment_matrix_a[track][site] = 1
reattach_a += 1
if(j<unstaggered_cells):
    reattach_a_1 += 1
else:
    reattach_a_2 += 1
```

```
elif(flag == 1):
    t1 = reservoir_b[i][j].queue[0].joins
    t2 = reservoir_b[i][j].queue[0].life

                                        //if motors B have reached beyond lifetime, then
                                        they are pushed into non-productive reservoir

    if(time-t1+t2 > lifetime):
        #remove the element as leakage
        #print("Leak B")
        reservoir_b[i][j].get()
        leakage_b[i][j] += 1
    else:
                                        //motors B can reattach into any one of the nine
                                        neighboring sites

        channels = list(range(0,rows_count))
        channels.remove(i)
        random.shuffle(channels)
        track1 = channels[0]
        track2 = channels[1]
        track = j
        if(i==1):
            if(j==cols_count-1):
                if(particle[i][j] == 0): site = j; track = i;
                elif(particle[i][j-1] == 0): site = j-1; track = i;
                else:  continue;
            else:
                if(particle[i][j+1] == 0): site = j+1; track = i;
                elif(particle[i][j] == 0): site = j; track = i;
                elif(particle[i][j-1] == 0): site = j-1; track = i;
                else: continue

        else:
                                                    //staggered tracks
            if(j>=(cols_count-(stagger/100)*cols_count)):
                if(particle[1][j+1] == 0): site = j+1; track = 1;
                elif(particle[1][j] == 0): site = j; track = 1;
                elif(particle[1][j-1] == 0): site = j-1; track = 1;
                else: continue
            elif(j<(cols_count-(stagger/100)*cols_count)):
                if(particle[i][j+1] == 0): site = j+1; track = i;
                elif(particle[track1][j+1] == 0): site = j+1; track = track1;
                elif(particle[track2][j+1] == 0): site = j+1; track = track2;
                elif(particle[i][j] == 0): site = j; track = i;
                elif(particle[track1][j] == 0): site = j; track = track1;
                elif(particle[track2][j] == 0): site = j; track = track2;
                elif(particle[i][j-1] == 0): site = j-1; track = i;
                elif(particle[track1][j-1] == 0): site = j-1; track = track1;
                elif(particle[track2][j-1] == 0): site = j-1; track = track2;
```

```
                    else: continue;

                particle[track][site] = 2
                velocity[track][site] = vel_b
                span[track][site] = 1

                t=reservoir_b[i][j].get()
                lifetime_cells[i][site] = time-t.joins+t.life
                #print("Reattach B")
                reattachment_b[track][site] += 1
                reattachment_matrix_b[track][site] = 1
                reattach_b += 1
                if(j<unstaggered_cells):
                    reattach_b_1 += 1
                else:
                    reattach_b_2 += 1
            else:
                continue


    end = timer()
    print("x take", end - start)
    #pdb.set_trace()
    return particle, velocity, span, lifetime_cells, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, reattach_a_1, reattach_b_1, reattach_a_2,
reattach_b_2, reattachment_matrix_a, reattachment_matrix_b, leakage_a, leakage_b
```

### *3.3.2* Motor traversing along the same track or movement into lateral track: (TASEP model)

```
def transport (rows_count, cols_count, vel_a, vel_b, process_a, process_b, throughput_a,
throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a, detach_b,
detachment_matrix_a, detachment_matrix_b, leakage_a, leakage_b, lateral_a, lateral_b,
time):

    start=timer()

    for i in range(rows_count):
        gap=0
        minimum=0
        for j in range(cols_count-1,-1,-1):
            if(particle[i][j] == 0):
                gap += 1
                continue
            else:
                #Output
                if(particle[i][j] == 1):
                    if(i==1):
                        if(gap < vel_a):
                            minimum = 0
```

```
        elif(gap >= vel_a):
            minimum = vel_a
    else:                           //movement restriction beyond staggering point
        if(j>(cols_count-(stagger/100)*cols_count) or gap < vel_a):
            minimum = 0
        elif(gap >= vel_a):
            minimum = vel_a

#Constant velocity
elif(particle[i][j] == 2):
    if(i==1):
        if(gap < vel_b):
            minimum = 0
        elif(gap >= vel_b):
            minimum = vel_b
    else:                           //movement restriction beyond staggering point
        if(j>(cols_count-(stagger/100)*cols_count) or gap < vel_b):
            minimum = 0
        elif(gap >= vel_b):
            minimum = vel_b

                                    //if sufficient gap is present then motor moves with
                                    constant velocity along the microtubular track

#Movement
if(minimum > 0):
    if(particle[i][j] == 1):
        if((j + vel_a) >= cols_count-1):                //delivery of motor A
            throughput_a += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

        else:                               //motor A move from one site to another
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 1
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

            if(span[i][j+minimum] >= process_a):
                #detachment if motor exceeds its processivity
                velocity[i][j+minimum] = 0
                particle[i][j+minimum] = 0
                span[i][j+minimum] = 0
                print('y1')
                if(reservoir_a[i][j].qsize()==reservoir_length):
```

```
                reservoir_a[i][j].get()
                leakage_a[i][j] += 1

            reservoir_a[i][j].put(props(time,lifetime_cells[i][j+minimum]))

            lifetime_cells[i][j+minimum]=0
            detachment_a[i][j+minimum] += 1
            detachment_matrix_a[i][j+minimum] = 1
            detach_a +=1
            detach_process_a += 1
            if(j<unstaggered_cells):
                detach_process_a_1 += 1
            else:
                detach_process_a_2 += 1

    elif(particle[i][j] == 2):
        if((j + vel_b) >= cols_count-1):                    //delivery of motor B
            throughput_b += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

        else:                                   //motor B move from one site to another
            velocity[i][j+minimum] = minimum
            particle[i][j+minimum] = 2
            span[i][j+minimum] = span[i][j] + minimum
            lifetime_cells[i][j+minimum] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0

            if(span[i][j+minimum] >= process_b):
                #detachment if motor exceeds its processivity
                velocity[i][j+minimum] = 0
                particle[i][j+minimum] = 0
                span[i][j+minimum] = 0
                print('y2')
                if(reservoir_b[i][j].qsize()==reservoir_length):
                    reservoir_b[i][j].get()
                    leakage_b[i][j] += 1

                reservoir_b[i][j].put(props(time,lifetime_cells[i][j+minimum]))

                lifetime_cells[i][j+minimum]=0
                detachment_b[i][j+minimum] += 1
                detachment_matrix_b[i][j+minimum] = 1
                detach_b +=1
                detach_process_a += 1
                if(j<unstaggered_cells):
```

```
                detach_process_a_1 += 1
            else:
                detach_process_a_2 += 1

#Detachment or Lateral Movement
elif(minimum == 0):
    if(particle[i][j] == 1):
                                    //the microtubular track before the staggering point
                                    has provision for lateral movement for both motors
        if(j<((cols_count-(stagger/100)*cols_count)-vel_a)):
            # Lateral association will happen to only cargos lost at initial stages
            channels = list(range(0,rows_count))        # All channels
            channels.remove(i)
            random.shuffle(channels)
            y1 = channels[0]
            channels.remove(channels[0])
            y2 = channels[0]

            if((j + vel_a) >= cols_count-1):
                throughput_a += 1
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_a += 1
            else:
                if(particle[y1][j+vel_a] == 0):
                    velocity[y1][j+vel_a] = vel_a
                    particle[y1][j+vel_a] = 1
                    span[y1][j+vel_a] = span[i][j] + vel_a
                    lifetime_cells[y1][j+vel_a] = lifetime_cells[i][j]
                    velocity[i][j] = 0
                    particle[i][j] = 0
                    span[i][j] = 0
                    lifetime_cells[i][j] = 0
                    lateral_a += 1

                elif(particle[y2][j+vel_a] == 0):
                    velocity[y2][j+vel_a] = vel_a
                    particle[y2][j+vel_a] = 1
                    span[y2][j+vel_a] = span[i][j] + vel_a
                    lifetime_cells[y2][j+vel_a] = lifetime_cells[i][j]
                    velocity[i][j] = 0
                    particle[i][j] = 0
                    span[i][j] = 0
                    lifetime_cells[i][j] = 0
                    lateral_a += 1

                else:
                    velocity[i][j] = 0
                    particle[i][j] = 0
```

```
            span[i][j] = 0
            #print('y3')
            if(reservoir_a[i][j].qsize()==reservoir_length):
                reservoir_a[i][j].get()
                leakage_a[i][j] += 1

            reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

            lifetime_cells[i][j]=0
            detachment_a[i][j] += 1
            detachment_matrix_a[i][j] = 1
            detach_a += 1
            if(j<unstaggered_cells):
                detach_a_1 += 1
            else:
                detach_a_2 += 1

                                        //the latter portion of microtubular track
                                        being blocked restricts lateral movement

    elif(j>=((cols_count-(stagger/100)*cols_count)-vel_a)):
        if(i==1):
            if((j + vel_a) >= cols_count-1):
                throughput_a += 1
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0

            else:
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                #print('y3')
                if(reservoir_a[i][j].qsize()==reservoir_length):
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1

                reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

                lifetime_cells[i][j]=0
                detachment_a[i][j] += 1
                detachment_matrix_a[i][j] = 1
                detach_a += 1
                if(j<staggered_cells):
                    detach_a_1 += 1
                else:
                    detach_a_2 += 1

        else:
            if(particle[1][j+vel_a] == 0):
```

```
                velocity[1][j+vel_a] = vel_a
                particle[1][j+vel_a] = 1
                span[1][j+vel_a] = span[i][j] + vel_a
                lifetime_cells[1][j+vel_a] = lifetime_cells[i][j]
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_a += 1

            else:
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                #print('y3')
                if(reservoir_a[i][j].qsize()==reservoir_length):
                    reservoir_a[i][j].get()
                    leakage_a[i][j] += 1

                reservoir_a[i][j].put(props(time,lifetime_cells[i][j]))

                lifetime_cells[i][j]=0
                detachment_a[i][j] += 1
                detachment_matrix_a[i][j] = 1
                detach_a += 1
                if(j<unstaggered_cells):
                    detach_a_1 += 1
                else:
                    detach_a_2 += 1

elif(particle[i][j] == 2):
                    //the microtubular track before the staggering point
                    has provision for lateral movement for both motors

    if(j<((cols_count-(stagger/100)*cols_count)-vel_b)):
        channels = list(range(0,rows_count))        # All channels
        channels.remove(i)
        random.shuffle(channels)
        y1 = channels[0]
        channels.remove(channels[0])
        y2 = channels[0]

        if((j + vel_b) >= cols_count-1):
            throughput_b += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            lateral_b += 1
        else:
            if(particle[y1][j+vel_b]== 0):
```

```
            velocity[y1][j+vel_b] = vel_b
            particle[y1][j+vel_b] = 2
            span[y1][j+vel_b] = span[i][j] + vel_b
            lifetime_cells[y1][j+vel_b] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            lateral_b += 1

        elif(particle[y2][j+vel_b]== 0):
            velocity[y2][j+vel_b] = vel_b
            particle[y2][j+vel_b] = 2
            span[y2][j+vel_b] = span[i][j] + vel_b
            lifetime_cells[y2][j+vel_b] = lifetime_cells[i][j]
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
            lateral_b += 1

        else:
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            #print('y4')
            if(reservoir_b[i][j].qsize()==reservoir_length):
                reservoir_b[i][j].get()
                leakage_b[i][j] += 1

            reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))

            lifetime_cells[i][j]=0
            detachment_b[i][j] += 1
            detachment_matrix_b[i][j] = 1
            detach_b += 1
            if(j<unstaggered_cells):
                detach_b_1 += 1
            else:
                detach_b_2 += 1
                                    //the latter portion of microtubular track
                                    being blocked restricts lateral movement

    elif(j>=((cols_count-(stagger/100)*cols_count)-vel_b)):
     if(i == 1):
        if((j + vel_b) >= cols_count-1):
            throughput_b += 1
            velocity[i][j] = 0
            particle[i][j] = 0
            span[i][j] = 0
            lifetime_cells[i][j] = 0
```

```
            else:
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                #print('y4')
                if(reservoir_b[i][j].qsize()==reservoir_length):
                    reservoir_b[i][j].get()
                    leakage_b[i][j] += 1

                reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))

                lifetime_cells[i][j]=0
                detachment_b[i][j] += 1
                detachment_matrix_b[i][j] = 1
                detach_b += 1
                if(j<unstaggered_cells):
                    detach_b_1 += 1
                else:
                    detach_b_2 += 1

        else:
            if(particle[1][j+vel_b]== 0):
                velocity[1][j+vel_b] = vel_b
                particle[1][j+vel_b] = 2
                span[1][j+vel_b] = span[i][j] + vel_b
                lifetime_cells[1][j+vel_b] = lifetime_cells[i][j]
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                lifetime_cells[i][j] = 0
                lateral_b += 1

            else:
                velocity[i][j] = 0
                particle[i][j] = 0
                span[i][j] = 0
                #print('y4')
                if(reservoir_b[i][j].qsize()==reservoir_length):
                    reservoir_b[i][j].get()
                    leakage_b[i][j] += 1

                reservoir_b[i][j].put(props(time,lifetime_cells[i][j]))

                lifetime_cells[i][j]=0
                detachment_b[i][j] += 1
                detachment_matrix_b[i][j] = 1
                detach_b += 1
                if(j<unstaggered_cells):
                    detach_b_1 += 1
                else:
                    detach_b_2 += 1
```

```
                gap = 0

    end = timer()
    print("y take", end - start)
    #pdb.set_trace()
return particle, velocity, span, lifetime_cells, throughput_a, throughput_b, reservoir_a,
reservoir_b, detachment_a, detachment_b, detach_a, detach_b, detach_a_1, detach_b_1,
detach_a_2, detach_b_2, detachment_matrix_a, detachment_matrix_b, leakage_a,
leakage_b, lateral_a, lateral_b, detach_process_a, detach_process_b, detach_process_a_1,
detach_process_b_1, detach_process_a_2, detach_process_b_2
```

# 4. Multisegmentation:

The following code is defined for multi-segment microtubular track system. Here we have considered 3 segment system where motors delivered from one system is collected into a temp junction reservoir which acts as input for the subsequent segment.

Each segment has a different format of simulation restrictions like:
i) Scenario Combination 1-3P-6D, 1-4P-7D and 1-5P-8D (Segment 1,2 and 3 has conditions of Scenario 1, 3/4/5P and 6/7/8D respectively)
ii) Scenario Combination 1-6D-3P, 1-7D-4P and 1-8D-5P (Segment 1,2 and 3 has conditions of Scenario 1, 6/7/8D and 3/4/5P respectively)

Code for Scenario 1-4P-7D is shown below:

## *4.1* Header Files:

```
import xlsxwriter
import random
import matplotlib.pyplot as plt
import math
import numpy as np
import seaborn as sns
import csv
from timeit import default_timer as timer
import pdb
import os

from queue import Queue
from collections import deque
import queue
from numpy.random import permutation

from matplotlib.colors import LogNorm
from PIL import Image, ImageDraw, ImageFont
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
from textwrap import wrap
import glob
import pandas as pd



from matplotlib.colors import LinearSegmentedColormap
cmap_name = 'my_list'
colors = [(1, 1, 1), (0, 0, 1), (1, 0, 0)]  # White -> Blue -> Red
cm = LinearSegmentedColormap.from_list(cmap_name, colors, N=3)
```

## *4.2* Output Directory Folders:

```
script_dir = os.getcwd()
graphs_dir = os.path.join(script_dir, 'Reservoir1/')

if not os.path.isdir(graphs_dir):
    os.makedirs(graphs_dir)

graphs_dir = os.path.join(script_dir, 'Reservoir2/')

if not os.path.isdir(graphs_dir):
    os.makedirs(graphs_dir)

graphs_dir = os.path.join(script_dir, 'Reservoir_Stagger/')

if not os.path.isdir(graphs_dir):
    os.makedirs(graphs_dir)

graphs_dir = os.path.join(script_dir, 'Reservoir_NonStagger/')

if not os.path.isdir(graphs_dir):
    os.makedirs(graphs_dir)

images_dir = os.path.join(script_dir, 'Images/')

if not os.path.isdir(images_dir):
    os.makedirs(images_dir)

images_dir = os.path.join(script_dir, 'GIFs/')

if not os.path.isdir(images_dir):
    os.makedirs(images_dir)
```

## *4.3* Data Parameters:

```
association_rate = 20

rows_count = 3
cols_count = 1000
segments = 3

cols1 = range(0,1000)


# Velocity steps:

# Velocity ratio : Va/Vb (Varying it)
vel_a = 5  #1000nm/s
vel_b = 3  #800nm/s
```

```
#Keeping processivity of A and B same ie Pa/Pb = 1
process_a = 1005
process_b = 150

#Lifetime
lifetime = 1500 #1 min lifetime

gap = 0
minimum = 0
randomize_traffic = 0.4
reservoir_length = 20

total_runtime = 18001

print("Multisegment Sceanrio 1-4P-7D")
stagger1 = float(input("Enter the staggering percentage for Scenario 4P: "))
stagger2 = float(input("Enter the staggering percentage for Scenario 7D: "))
```

## *4.4* **Output Files:**

```
workbook = xlsxwriter.Workbook('Scenario 147 Outputs.xlsx')

worksheet9 = workbook.add_worksheet()            #Outputs

worksheet9.write(0, 0, "Time")
worksheet9.write(0, 1, "Input a")
worksheet9.write(1, 1, "Segment 1")
worksheet9.write(1, 2, "Segment 2")
worksheet9.write(1, 3, "Segment 3")
worksheet9.write(0, 4, "Input b")
worksheet9.write(1, 4, "Segment 1")
worksheet9.write(1, 5, "Segment 2")
worksheet9.write(1, 6, "Segment 3")

worksheet9.write(0, 7, "Output a")
worksheet9.write(1, 7, "Segment 1")
worksheet9.write(1, 8, "Segment 2")
worksheet9.write(1, 9, "Segment 3")
worksheet9.write(0, 10, "Output b")
worksheet9.write(1, 10, "Segment 1")
worksheet9.write(1, 11, "Segment 2")
worksheet9.write(1, 12, "Segment 3")

worksheet9.write(0, 13, 'Lateral a')
worksheet9.write(1, 13, "Segment 1")
worksheet9.write(1, 14, "Segment 2")
worksheet9.write(1, 15, "Segment 3")
worksheet9.write(0, 16, 'Lateral b')
worksheet9.write(1, 16, "Segment 1")
```

```
worksheet9.write(1, 17, "Segment 2")
worksheet9.write(1, 18, "Segment 3")

worksheet9.write(0, 19, 'Detach a')
worksheet9.write(1, 19, "Segment 1")
worksheet9.write(1, 20, "Segment 2")
worksheet9.write(1, 21, "Segment 3")

worksheet9.write(0, 22, 'Detach b')
worksheet9.write(1, 22, "Segment 1")
worksheet9.write(1, 23, "Segment 2")
worksheet9.write(1, 24, "Segment 3")

worksheet9.write(0, 25, 'Reattach a')
worksheet9.write(1, 25, "Segment 1")
worksheet9.write(1, 26, "Segment 2")
worksheet9.write(1, 27, "Segment 3")

worksheet9.write(0, 28, 'Reattach b')
worksheet9.write(1, 28, "Segment 1")
worksheet9.write(1, 29, "Segment 2")
worksheet9.write(1, 30, "Segment 3")
```

## *4.5* **Data Metrics:**

```
input_res = [Queue(maxsize = total_runtime) for l in range(segments+1)]

#Association
motor_cargo_a = [0 for l in range(segments)]    #no of a type motor cargos
motor_cargo_b = [0 for l in range(segments)]    #no of b type motor cargos

#Detachment at the end
throughput_a = [0 for l in range(segments)] #cargo_a output
throughput_b = [0 for l in range(segments)] #cargo_b output

lateral_a = [0 for l in range(segments)]
lateral_b = [0 for l in range(segments)]

#Motors lost due to detachment
detachment_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
detachment_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]

#Motors used due to reattachment
reattachment_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
reattachment_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
```

```
heat_map_a = [[[0 for j in range(200)] for i in range(rows_count)] for l in range(segments)]
heat_map_b = [[[0 for j in range(200)] for i in range(rows_count)] for l in range(segments)]

particle = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in range(segments)]
span = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in range(segments)]
lifetime_cells = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]

waiting_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
waiting_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]

leakage_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
leakage_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]

productive_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
productive_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]

non_productive_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
non_productive_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]


wait_a = [0 for l in range(segments)]
wait_b = [0 for l in range(segments)]
leak_a = [0 for l in range(segments)]
leak_b = [0 for l in range(segments)]
detach_a = [0 for l in range(segments)]
detach_b = [0 for l in range(segments)]
reattach_a = [0 for l in range(segments)]
reattach_b = [0 for l in range(segments)]


//Productive Reservoir
class props(object): #detachment time and lifetime
    def __init__(self, joins, life):
        self.joins = joins
        self.life = life
        return

reservoir_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
reservoir_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
```

```
for t in range(segments):
    for i in range(rows_count):
        for j in range(cols_count):
            reservoir_a[t][i][j] = Queue(maxsize = reservoir_length)
            reservoir_b[t][i][j] = Queue(maxsize = reservoir_length)
```

## *4.6* Initial Setup for Generation of Motor Cargo Pair:

```
res_in_a_len = int((association_rate*total_runtime)/100)
res_in_b_len = int((association_rate*total_runtime)/100)
res_in_len = total_runtime-(res_in_a_len + res_in_b_len)

res_in_l = []
res_in_l = deque()


for i in range(res_in_len):
    res_in_l.append(0)

for i in range(res_in_a_len):
    res_in_l.append(1)

for i in range(res_in_b_len):
    res_in_l.append(2)

random.shuffle(res_in_l)

for i in range(total_runtime):
    input_res[0].put(res_in_l[i])
```

## *4.7* Motor Association on Loading Site:

```
//Association for Scenario 1
def initial_association1(input_res, motor_cargo_a, motor_cargo_b, segment):

    start = timer()

    channels = list(range(0,rows_count))        # All channels
    random.shuffle(channels)
    c1 = channels[0]
    channels.remove(channels[0])
    c2 = channels[0]
    channels.remove(channels[0])
    c3 = channels[0]
```

```
    if(particle[segment][c1][0] == 0):
        particle[segment][c1][0] = input_res[segment].get()
        span[segment][c1][0] = 1
        lifetime_cells[segment][c1][0] = 1

        if(particle[segment][c1][0] == 1):
            motor_cargo_a[segment] += 1

        elif(particle[segment][c1][0] == 2):
            motor_cargo_b[segment] += 1


    elif(particle[segment][c2][0] == 0):
        particle[segment][c2][0] = input_res[segment].get()
        span[segment][c2][0] = 1
        lifetime_cells[segment][c2][0] = 1

        if(particle[segment][c2][0] == 1):
            motor_cargo_a[segment] += 1

        elif(particle[segment][c2][0] == 2):
            motor_cargo_b[segment] += 1

    elif(particle[segment][c3][0] == 0):
        particle[segment][c3][0] = input_res[segment].get()
        span[segment][c3][0] = 1
        lifetime_cells[segment][c3][0] = 1

        if(particle[segment][c3][0] == 1):
            motor_cargo_a[segment] += 1

        elif(particle[segment][c3][0] == 2):
            motor_cargo_b[segment] += 1

    end = timer()
    print("Assocaition of 1 takes", end - start)
    #pdb.set_trace()
    #print(1)
    return input_res, motor_cargo_a, motor_cargo_b
```

**//Association for Scenario 4**

```
def initial_association4(input_res, motor_cargo_a, motor_cargo_b, segment):

    start = timer()

    c1 = 1

    if(particle[segment][c1][0] == 0):
        particle[segment][c1][0] = input_res[segment].get()
        span[segment][c1][0] = 1
        lifetime_cells[segment][c1][0] = 1
```

```
        if(particle[segment][c1][0] == 1):
            motor_cargo_a[segment] += 1

        elif(particle[segment][c1][0] == 2):
            motor_cargo_b[segment] += 1

    end = timer()
    print("Association of 4 take", end - start)
    #pdb.set_trace()
    #print(2)
    return input_res, motor_cargo_a, motor_cargo_b


//Association for Scenario 7
def initial_association7(input_res, motor_cargo_a, motor_cargo_b, segment):

    start = timer()

    channels = list(range(0,rows_count))        # All channels
    random.shuffle(channels)
    c1 = channels[0]
    channels.remove(channels[0])
    c2 = channels[0]
    channels.remove(channels[0])
    c3 = channels[0]

    if(particle[segment][c1][0] == 0):
        particle[segment][c1][0] = input_res[segment].get()
        span[segment][c1][0] = 1
        lifetime_cells[segment][c1][0] = 1

        if(particle[segment][c1][0] == 1):
            motor_cargo_a[segment] += 1

        elif(particle[segment][c1][0] == 2):
            motor_cargo_b[segment] += 1


    elif(particle[segment][c2][0] == 0):
        particle[segment][c2][0] = input_res[segment].get()
        span[segment][c2][0] = 1
        lifetime_cells[segment][c2][0] = 1

        if(particle[segment][c2][0] == 1):
            motor_cargo_a[segment] += 1

        elif(particle[segment][c2][0] == 2):
            motor_cargo_b[segment] += 1

    elif(particle[segment][c3][0] == 0):
```

```
    particle[segment][c3][0] = input_res[segment].get()
    span[segment][c3][0] = 1
    lifetime_cells[segment][c3][0] = 1

    if(particle[segment][c3][0] == 1):
        motor_cargo_a[segment] += 1

    elif(particle[segment][c3][0] == 2):
        motor_cargo_b[segment] += 1

end = timer()
print("Assocaition of 7 takes", end - start)
#pdb.set_trace()
#print(1)
return input_res, motor_cargo_a, motor_cargo_b
```

## *4.8* Reattachment and Detachment of Productive Reservoir:

```
// Productive Reservoir Scenario 1
def lateral_association1(rows_count, cols_count, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, leakage_a, leakage_b, segment, time):

    start = timer()
    i=0
    j=0
    k=0
    temp = 0
    track = 0
    site = 0
    flag = 0
    flag2 = 0
    for i in range(rows_count):
        for j in range(cols_count):
            if(reservoir_a[segment][i][j].qsize() != 0 and reservoir_b[segment][i][j].qsize() == 0):
                flag = 0
            elif(reservoir_a[segment][i][j].qsize() == 0 and reservoir_b[segment][i][j].qsize() != 0):
                flag = 1
            elif(reservoir_a[segment][i][j].qsize() != 0 and reservoir_b[segment][i][j].qsize() != 0):
                flag = random.randint(0,1)
            else:
                flag = 10

            if(flag == 0):
                t1 = reservoir_a[segment][i][j].queue[0].joins
                t2 = reservoir_a[segment][i][j].queue[0].life
                if(time-t1+t2 > lifetime):
                    #remove the element as leakage
                    #print("Leak A")
                    reservoir_a[segment][i][j].get()
```

```
            leakage_a[segment][i][j] += 1
        else:
            channels = list(range(0,rows_count))
            channels.remove(i)
            random.shuffle(channels)
            track1 = channels[0]
            track2 = channels[1]
            track = i
            site = j
            if(j == cols_count-1):
                if(particle[segment][i][j] == 0): site = j; track = i;
                elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                elif(particle[segment][track1][j] == 0): site = j; track = track1;
                elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                elif(particle[segment][track2][j] == 0): site = j; track = track2;
                elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                else:  continue;
            else:
                if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                elif(particle[segment][track1][j+1] == 0): site = j+1; track = track1;
                elif(particle[segment][track2][j+1] == 0): site = j+1; track = track2;
                elif(particle[segment][i][j] == 0): site = j; track = i;
                elif(particle[segment][track1][j] == 0): site = j; track = track1;
                elif(particle[segment][track2][j] == 0): site = j; track = track2;
                elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                else:  continue;


            particle[segment][track][site] = 1
            span[segment][track][site] = 1

            t = reservoir_a[segment][i][j].get()
            lifetime_cells[segment][track][site] = time-t.joins+t.life
            #print("Reattach A")
            reattachment_a[segment][track][site] += 1
            reattach_a[segment] += 1

    elif(flag == 1):
        t1 = reservoir_b[segment][i][j].queue[0].joins
        t2 = reservoir_b[segment][i][j].queue[0].life
        if(time-t1+t2 > lifetime):
            #remove the element as leakage
            #print("Leak B")
            reservoir_b[segment][i][j].get()
            leakage_b[segment][i][j] += 1
        else:
            channels = list(range(0,rows_count))
            channels.remove(i)
            random.shuffle(channels)
```

```
                track1 = channels[0]
                track2 = channels[1]
                track = i
                site = j
                if(j == cols_count-1):
                    if(particle[segment][i][j] == 0): site = j; track = i;
                    elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                    elif(particle[segment][track1][j] == 0): site = j; track = track1;
                    elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                    elif(particle[segment][track2][j] == 0): site = j; track = track2;
                    elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                    else:  continue;
                else:
                    if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                    elif(particle[segment][track1][j+1] == 0): site = j+1; track = track1;
                    elif(particle[segment][track2][j+1] == 0): site = j+1; track = track2;
                    elif(particle[segment][i][j] == 0): site = j; track = i;
                    elif(particle[segment][track1][j] == 0): site = j; track = track1;
                    elif(particle[segment][track2][j] == 0): site = j; track = track2;
                    elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                    elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                    elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                    else:  continue;

                particle[segment][track][site] = 2
                span[segment][track][site] = 1

                t = reservoir_b[segment][i][j].get()
                lifetime_cells[segment][track][site] = time-t.joins+t.life
                #print("Reattach B")
                reattachment_b[segment][track][site] += 1
                reattach_b[segment] += 1
            else:
                continue


    end = timer()
    print("Lateral 1 take", end - start)
    #pdb.set_trace()
    #print(4)
    return reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
leakage_a, leakage_b

// Productive Reservoir Scenario 4
def lateral_association4(rows_count, cols_count, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, leakage_a, leakage_b, segment, time):

    start = timer()
    i=0
    j=0
    k=0
```

```
temp = 0
track = 0
site = 0
flag = 0
flag2 = 0
for i in range(rows_count):
   for j in range(cols_count):
      if(reservoir_a[segment][i][j].qsize() != 0 and reservoir_b[segment][i][j].qsize() == 0):
         flag = 0
      elif(reservoir_a[segment][i][j].qsize() == 0 and reservoir_b[segment][i][j].qsize() != 0):
         flag = 1
      elif(reservoir_a[segment][i][j].qsize() != 0 and reservoir_b[segment][i][j].qsize() != 0):
         flag = random.randint(0,1)
      else:
         flag = 10

      if(flag == 0):
         t1 = reservoir_a[segment][i][j].queue[0].joins
         t2 = reservoir_a[segment][i][j].queue[0].life
         if(time-t1+t2 > lifetime):
            #remove the element as leakage
            #print("Leak A")
            reservoir_a[segment][i][j].get()
            leakage_a[segment][i][j] += 1
         else:
            channels = list(range(0,rows_count))
            channels.remove(i)
            random.shuffle(channels)
            track1 = channels[0]
            track2 = channels[1]
            track = i
            site = j
            if(j == cols_count-1):
               if(particle[segment][i][j] == 0): site = j; track = i;
               elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
               elif(particle[segment][track1][j] == 0): site = j; track = track1;
               elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
               elif(particle[segment][track2][j] == 0): site = j; track = track2;
               elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
               else:  continue;
            else:
               if(i == 1 and j == 1):
                  if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                  elif(particle[segment][i][j] == 0): site = j; track = i;

               elif(i == 1 and j <= (stagger1/100)*cols_count):
                  if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                  elif(particle[segment][i][j] == 0): site = j; track = i;
                  elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                  else: continue
```

```
            elif(j > (stagger1/100)*cols_count):
                if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                elif(particle[segment][track1][j+1] == 0): site = j+1; track = track1;
                elif(particle[segment][track2][j+1] == 0): site = j+1; track = track2;
                elif(particle[segment][i][j] == 0): site = j; track = i;
                elif(particle[segment][track1][j] == 0): site = j; track = track1;
                elif(particle[segment][track2][j] == 0): site = j; track = track2;
                elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                else: continue;


        particle[segment][track][site] = 1
        span[segment][track][site] = 1

        t = reservoir_a[segment][i][j].get()
        lifetime_cells[segment][track][site] = time-t.joins+t.life
        #print("Reattach A")
        reattachment_a[segment][track][site] += 1
        reattach_a[segment] += 1

elif(flag == 1):
    t1 = reservoir_b[segment][i][j].queue[0].joins
    t2 = reservoir_b[segment][i][j].queue[0].life
    if(time-t1+t2 > lifetime):
        #remove the element as leakage
        #print("Leak B")
        reservoir_b[segment][i][j].get()
        leakage_b[segment][i][j] += 1
    else:
        channels = list(range(0,rows_count))
        channels.remove(i)
        random.shuffle(channels)
        track1 = channels[0]
        track2 = channels[1]
        track = i
        site = j
        if(j == cols_count-1):
            if(particle[segment][i][j] == 0): site = j; track = i;
            elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
            elif(particle[segment][track1][j] == 0): site = j; track = track1;
            elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
            elif(particle[segment][track2][j] == 0): site = j; track = track2;
            elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
            else:  continue;
        else:
            if(i == 1 and j == 1):
                if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                elif(particle[segment][i][j] == 0): site = j; track = i;
```

```
                elif(i == 1 and j <= (stagger1/100)*cols_count):
                    if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                    elif(particle[segment][i][j] == 0): site = j; track = i;
                    elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                    else: continue

                elif(j > (stagger1/100)*cols_count):
                    if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                    elif(particle[segment][track1][j+1] == 0): site = j+1; track = track1;
                    elif(particle[segment][track2][j+1] == 0): site = j+1; track = track2;
                    elif(particle[segment][i][j] == 0): site = j; track = i;
                    elif(particle[segment][track1][j] == 0): site = j; track = track1;
                    elif(particle[segment][track2][j] == 0): site = j; track = track2;
                    elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                    elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                    elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                    else: continue;

                particle[segment][track][site] = 2
                span[segment][track][site] = 1

                t = reservoir_b[segment][i][j].get()
                lifetime_cells[segment][track][site] = time-t.joins+t.life
                #print("Reattach B")
                reattachment_b[segment][track][site] += 1
                reattach_b[segment] += 1
            else:
                continue


    end = timer()
    print("Lateral 4 take", end - start)
    #pdb.set_trace()
    #print(5)
    return reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
leakage_a, leakage_b
```

**// Productive Reservoir Scenario 7**

```
def lateral_association7(rows_count, cols_count, reservoir_a, reservoir_b, reattachment_a,
reattachment_b, reattach_a, reattach_b, leakage_a, leakage_b, segment, time):

    start = timer()
    i=0
    j=0
    k=0
    temp = 0
    track = 0
    site = 0
    flag = 0
```

```
flag2 = 0
for i in range(rows_count):
    for j in range(cols_count):
        if(reservoir_a[segment][i][j].qsize() != 0 and reservoir_b[segment][i][j].qsize() == 0):
            flag = 0
        elif(reservoir_a[segment][i][j].qsize() == 0 and reservoir_b[segment][i][j].qsize() != 0):
            flag = 1
        elif(reservoir_a[segment][i][j].qsize() != 0 and reservoir_b[segment][i][j].qsize() != 0):
            flag = random.randint(0,1)
        else:
            flag = 10

        if(flag == 0):
            t1 = reservoir_a[segment][i][j].queue[0].joins
            t2 = reservoir_a[segment][i][j].queue[0].life
            if(time-t1+t2 > lifetime):
                #remove the element as leakage
                #print("Leak A")
                reservoir_a[segment][i][j].get()
                leakage_a[segment][i][j] += 1
            else:
                channels = list(range(0,rows_count))
                channels.remove(i)
                random.shuffle(channels)
                track1 = channels[0]
                track2 = channels[1]
                track = i
                site = j
                if(i==1):
                    if(j==cols_count-1):
                        if(particle[segment][i][j] == 0): site = j; track = i;
                        elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                        else:  continue;
                    else:
                        if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                        elif(particle[segment][i][j] == 0): site = j; track = i;
                        elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                        else: continue

                else:
                    if(j>=(cols_count-(stagger2/100)*cols_count)):
                        if(particle[segment][1][j+1] == 0): site = j+1; track = 1;
                        elif(particle[segment][1][j] == 0): site = j; track = 1;
                        elif(particle[segment][1][j-1] == 0): site = j-1; track = 1;
                        else: continue
                    elif(j<(cols_count-(stagger2/100)*cols_count)):
                        if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                        elif(particle[segment][track1][j+1] == 0): site = j+1; track = track1;
                        elif(particle[segment][track2][j+1] == 0): site = j+1; track = track2;
                        elif(particle[segment][i][j] == 0): site = j; track = i;
                        elif(particle[segment][track1][j] == 0): site = j; track = track1;
```

```
                elif(particle[segment][track2][j] == 0): site = j; track = track2;
                elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                else: continue;

        particle[segment][track][site] = 1
        span[segment][track][site] = 1

        t = reservoir_a[segment][i][j].get()
        lifetime_cells[segment][track][site] = time-t.joins+t.life
        #print("Reattach A")
        reattachment_a[segment][track][site] += 1
        reattach_a[segment] += 1

elif(flag == 1):
    t1 = reservoir_b[segment][i][j].queue[0].joins
    t2 = reservoir_b[segment][i][j].queue[0].life
    if(time-t1+t2 > lifetime):
        #remove the element as leakage
        #print("Leak B")
        reservoir_b[segment][i][j].get()
        leakage_b[segment][i][j] += 1
    else:
        channels = list(range(0,rows_count))
        channels.remove(i)
        random.shuffle(channels)
        track1 = channels[0]
        track2 = channels[1]
        track = j
        if(i==1):
            if(j==cols_count-1):
                if(particle[segment][i][j] == 0): site = j; track = i;
                elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                else:  continue;
            else:
                if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                elif(particle[segment][i][j] == 0): site = j; track = i;
                elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                else: continue

        else:
            if(j>=(cols_count-(stagger2/100)*cols_count)):
                if(particle[segment][1][j+1] == 0): site = j+1; track = 1;
                elif(particle[segment][1][j] == 0): site = j; track = 1;
                elif(particle[segment][1][j-1] == 0): site = j-1; track = 1;
                else: continue
            elif(j<(cols_count-(stagger2/100)*cols_count)):
                if(particle[segment][i][j+1] == 0): site = j+1; track = i;
                elif(particle[segment][track1][j+1] == 0): site = j+1; track = track1;
                elif(particle[segment][track2][j+1] == 0): site = j+1; track = track2;
```

```
                    elif(particle[segment][i][j] == 0): site = j; track = i;
                    elif(particle[segment][track1][j] == 0): site = j; track = track1;
                    elif(particle[segment][track2][j] == 0): site = j; track = track2;
                    elif(particle[segment][i][j-1] == 0): site = j-1; track = i;
                    elif(particle[segment][track1][j-1] == 0): site = j-1; track = track1;
                    elif(particle[segment][track2][j-1] == 0): site = j-1; track = track2;
                    else: continue;


                particle[segment][track][site] = 2
                span[segment][track][site] = 1

                t=reservoir_b[segment][i][j].get()
                lifetime_cells[segment][track][site] = time-t.joins+t.life
                #print("Reattach B")
                reattachment_b[segment][track][site] += 1
                reattach_b[segment] += 1
            else:
                continue


    end = timer()
    print("Lateral 7 take", end - start)
    #pdb.set_trace()
    #print(6)
    return reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
leakage_a, leakage_b
```

## *4.9* Transport of Motor Cargo Pair using TASEP principles:

```
//Transport Scenario 1
def transport1(input_res, rows_count, cols_count, vel_a, vel_b, process_a, process_b,
throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a,
detach_b, leakage_a, leakage_b, lateral_a, lateral_b, segment, time):

    start=timer()
    flag = 0
    for i in range(rows_count):
        gap=0
        minimum=0
        for j in range(cols_count-1,-1,-1):
            if(particle[segment][i][j] == 0):
                gap += 1
                continue
            else:
                #Output
                if(particle[segment][i][j] == 1):
                    if(gap < vel_a):
                        minimum = 0
```

```
    elif(gap >= vel_a):
        minimum = vel_a

#Constant velocity
elif(particle[segment][i][j] == 2):
    if(gap < vel_b):
        minimum = 0
    elif(gap >= vel_b):
        minimum = vel_b


#Movement
if(minimum > 0):
    if(particle[segment][i][j] == 1):
        if((j + vel_a) >= cols_count-1):
            throughput_a[segment] += 1
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            flag = 1
            input_res[segment+1].put(1)

        else:
            particle[segment][i][j+minimum] = 1
            span[segment][i][j+minimum] = span[segment][i][j] + minimum
            lifetime_cells[segment][i][j+minimum] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            if(span[segment][i][j+minimum] >= process_a):
                #detachment if motor exceeds its processivity
                particle[segment][i][j+minimum] = 0
                span[segment][i][j+minimum] = 0
                #print('y1')
                if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                    reservoir_a[segment][i][j].get()
                    leakage_a[segment][i][j] += 1


             reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j+mini
             mum]))

              lifetime_cells[segment][i][j+minimum]=0
              detachment_a[segment][i][j+minimum] += 1
              detach_a[segment] += 1

    elif(particle[segment][i][j] == 2):
        if((j + vel_b) >= cols_count-1):
            throughput_b[segment] += 1
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
```

```
            lifetime_cells[segment][i][j] = 0
            flag = 1
            input_res[segment+1].put(2)

        else:
            particle[segment][i][j+minimum] = 2
            span[segment][i][j+minimum] = span[segment][i][j] + minimum
            lifetime_cells[segment][i][j+minimum] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            if(span[segment][i][j+minimum] >= process_b):
                #detachment if motor exceeds its processivity
                particle[segment][i][j+minimum] = 0
                span[segment][i][j+minimum] = 0
                #print('y2')
                if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                    reservoir_b[segment][i][j].get()
                    leakage_b[segment][i][j] += 1


             reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j+mini
             mum]))

              lifetime_cells[segment][i][j+minimum]=0
              detachment_b[segment][i][j+minimum] += 1
              detach_b[segment] += 1

#Detachment
elif(minimum == 0):
    if(particle[segment][i][j] == 1):
        # Lateral association will happen to only cargos lost at initial stages
        channels = list(range(0,rows_count))        # All channels
        channels.remove(i)
        random.shuffle(channels)
        y1 = channels[0]
        channels.remove(channels[0])
        y2 = channels[0]

        if((j + vel_a) >= cols_count-1):
            throughput_a[segment] += 1
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1
            flag = 1
            input_res[segment+1].put(1)

        else:
            if(particle[segment][y1][j+vel_a] == 0):
                particle[segment][y1][j+vel_a] = 1
```

```
            span[segment][y1][j+vel_a] = span[segment][i][j] + vel_a
            lifetime_cells[segment][y1][j+vel_a] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1

        elif(particle[segment][y2][j+vel_a] == 0):
            particle[segment][y2][j+vel_a] = 1
            span[segment][y2][j+vel_a] = span[segment][i][j] + vel_a
            lifetime_cells[segment][y2][j+vel_a] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1

        else:
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            #print('y3')
            if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                reservoir_a[segment][i][j].get()
                leakage_a[segment][i][j] += 1

            reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

            lifetime_cells[segment][i][j]=0
            detachment_a[segment][i][j] += 1
            detach_a[segment] += 1

elif(particle[segment][i][j] == 2):
    channels = list(range(0,rows_count))        # All channels
    channels.remove(i)
    random.shuffle(channels)
    y1 = channels[0]
    channels.remove(channels[0])
    y2 = channels[0]

    if((j + vel_b) >= cols_count-1):
        throughput_b[segment] += 1
        particle[segment][i][j] = 0
        span[segment][i][j] = 0
        lifetime_cells[segment][i][j] = 0
        lateral_b[segment] += 1
        flag = 1
        input_res[segment+1].put(2)

    else:
        if(particle[segment][y1][j+vel_b]== 0):
            particle[segment][y1][j+vel_b] = 2
            span[segment][y1][j+vel_b] = span[segment][i][j] + vel_b
```

```
                    lifetime_cells[segment][y1][j+vel_b] = lifetime_cells[segment][i][j]
                    particle[segment][i][j] = 0
                    span[segment][i][j] = 0
                    lifetime_cells[segment][i][j] = 0
                    lateral_b[segment] += 1

                elif(particle[segment][y2][j+vel_b]== 0):
                    particle[segment][y2][j+vel_b] = 2
                    span[segment][y2][j+vel_b] = span[segment][i][j] + vel_b
                    lifetime_cells[segment][y2][j+vel_b] = lifetime_cells[segment][i][j]
                    particle[segment][i][j] = 0
                    span[segment][i][j] = 0
                    lifetime_cells[segment][i][j] = 0
                    lateral_b[segment] += 1

                else:
                    particle[segment][i][j] = 0
                    span[segment][i][j] = 0
                    #print('y4')
                    if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                        reservoir_b[segment][i][j].get()
                        leakage_b[segment][i][j] += 1

                    reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                    lifetime_cells[segment][i][j]=0
                    detachment_b[segment][i][j] += 1
                    detach_b[segment] += 1

            gap = 0

    if(flag == 0):
        input_res[segment+1].put(0)

    end = timer()
    print("Transport 1 take", end - start)
    #pdb.set_trace()
    #print(7)
    return input_res, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a,
detachment_b, detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b
```

**//Transport Scenario 4**

```
def transport4(input_res, rows_count, cols_count, vel_a, vel_b, process_a, process_b,
throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a,
detach_b, leakage_a, leakage_b, lateral_a, lateral_b, segment, time):

    start=timer()
    flag = 0
```

```
for i in range(rows_count):
    gap=0
    minimum=0
    for j in range(cols_count-1,-1,-1):
        if(particle[segment][i][j] == 0):
            gap += 1
            continue
        else:
            #Output
            if(particle[segment][i][j] == 1):
                if(gap < vel_a):
                    minimum = 0
                elif(gap >= vel_a):
                    minimum = vel_a

            #Constant velocity
            elif(particle[segment][i][j] == 2):
                if(gap < vel_b):
                    minimum = 0
                elif(gap >= vel_b):
                    minimum = vel_b


            #Movement
            if(minimum > 0):
                if(particle[segment][i][j] == 1):
                    if((j + vel_a) >= cols_count-1):
                        throughput_a[segment] += 1
                        particle[segment][i][j] = 0
                        span[segment][i][j] = 0
                        lifetime_cells[segment][i][j] = 0
                        flag = 1
                        input_res[segment+1].put(1)

                    else:
                        particle[segment][i][j+minimum] = 1
                        span[segment][i][j+minimum] = span[segment][i][j] + minimum
                        lifetime_cells[segment][i][j+minimum] = lifetime_cells[segment][i][j]
                        particle[segment][i][j] = 0
                        span[segment][i][j] = 0
                        lifetime_cells[segment][i][j] = 0
                        if(span[segment][i][j+minimum] >= process_a):
                            #detachment if motor exceeds its processivity
                            particle[segment][i][j+minimum] = 0
                            span[segment][i][j+minimum] = 0
                            #print('y1')
                            if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                                reservoir_a[segment][i][j].get()
                                leakage_a[segment][i][j] += 1
```

```
                reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j+mini
                mum]))

                  lifetime_cells[segment][i][j+minimum]=0
                  detachment_a[segment][i][j+minimum] += 1
                  detach_a[segment] += 1

        elif(particle[segment][i][j] == 2):
            if((j + vel_b) >= cols_count-1):
                throughput_b[segment] += 1
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                flag = 1
                input_res[segment+1].put(2)

            else:
                particle[segment][i][j+minimum] = 2
                span[segment][i][j+minimum] = span[segment][i][j] + minimum
                lifetime_cells[segment][i][j+minimum] = lifetime_cells[segment][i][j]
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                if(span[segment][i][j+minimum] >= process_b):
                    #detachment if motor exceeds its processivity
                    particle[segment][i][j+minimum] = 0
                    span[segment][i][j+minimum] = 0
                    #print('y2')
                    if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                        reservoir_b[segment][i][j].get()
                        leakage_b[segment][i][j] += 1

                  reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j+mini
                  mum]))

                  lifetime_cells[segment][i][j+minimum]=0
                  detachment_b[segment][i][j+minimum] += 1
                  detach_b[segment] += 1

#Detachment
elif(minimum == 0):
    if(particle[segment][i][j] == 1):
        if(j>(stagger1/100)*cols_count):
            # Lateral association will happen to only cargos lost at initial stages
            channels = list(range(0,rows_count))        # All channels
            channels.remove(i)
            random.shuffle(channels)
            y1 = channels[0]
            channels.remove(channels[0])
            y2 = channels[0]
```

```
    if((j + vel_a) >= cols_count-1):
        throughput_a[segment] += 1
        particle[segment][i][j] = 0
        span[segment][i][j] = 0
        lifetime_cells[segment][i][j] = 0
        lateral_a[segment] += 1
        flag = 1
        input_res[segment+1].put(1)

    else:
        if(particle[segment][y1][j+vel_a] == 0):
            particle[segment][y1][j+vel_a] = 1
            span[segment][y1][j+vel_a] = span[segment][i][j] + vel_a
            lifetime_cells[segment][y1][j+vel_a] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1

        elif(particle[segment][y2][j+vel_a] == 0):
            particle[segment][y2][j+vel_a] = 1
            span[segment][y2][j+vel_a] = span[segment][i][j] + vel_a
            lifetime_cells[segment][y2][j+vel_a] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1

        else:
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            #print('y3')
            if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                reservoir_a[segment][i][j].get()
                leakage_a[segment][i][j] += 1


      reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

            lifetime_cells[segment][i][j]=0
            detachment_a[segment][i][j] += 1
            detach_a[segment] += 1

elif(j<=(stagger1/100)*cols_count):
    particle[segment][i][j] = 0
    span[segment][i][j] = 0
    #print('y3')
    if(reservoir_a[segment][i][j].qsize()==reservoir_length):
        reservoir_a[segment][i][j].get()
        leakage_a[segment][i][j] += 1
```

```
            reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

            lifetime_cells[segment][i][j]=0
            detachment_a[segment][i][j] += 1
            detach_a[segment] += 1

    elif(particle[segment][i][j] == 2):
        if(j>(stagger1/100)*cols_count):
            channels = list(range(0,rows_count))        # All channels
            channels.remove(i)
            random.shuffle(channels)
            y1 = channels[0]
            channels.remove(channels[0])
            y2 = channels[0]

            if((j + vel_b) >= cols_count-1):
                throughput_b[segment] += 1
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                lateral_b[segment] += 1
                flag = 1
                input_res[segment+1].put(2)

            else:
                if(particle[segment][y1][j+vel_b]== 0):
                    particle[segment][y1][j+vel_b] = 2
                    span[segment][y1][j+vel_b] = span[segment][i][j] + vel_b
                    lifetime_cells[segment][y1][j+vel_b] = lifetime_cells[segment][i][j]
                    particle[segment][i][j] = 0
                    span[segment][i][j] = 0
                    lifetime_cells[segment][i][j] = 0
                    lateral_b[segment] += 1

                elif(particle[segment][y2][j+vel_b]== 0):
                    particle[segment][y2][j+vel_b] = 2
                    span[segment][y2][j+vel_b] = span[segment][i][j] + vel_b
                    lifetime_cells[segment][y2][j+vel_b] = lifetime_cells[segment][i][j]
                    particle[segment][i][j] = 0
                    span[segment][i][j] = 0
                    lifetime_cells[segment][i][j] = 0
                    lateral_b[segment] += 1

                else:
                    particle[segment][i][j] = 0
                    span[segment][i][j] = 0
                    #print('y4')
                    if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                        reservoir_b[segment][i][j].get()
                        leakage_b[segment][i][j] += 1
```

```
                        reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                            lifetime_cells[segment][i][j]=0
                            detachment_b[segment][i][j] += 1
                            detach_b[segment] += 1

                    elif(j<=(stagger1/100)*cols_count):
                        particle[segment][i][j] = 0
                        span[segment][i][j] = 0
                        #print('y4')
                        if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                            reservoir_b[segment][i][j].get()
                            leakage_b[segment][i][j] += 1

                        reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                        lifetime_cells[segment][i][j]=0
                        detachment_b[segment][i][j] += 1
                        detach_b[segment] += 1

                gap = 0

        if(flag == 0):
            input_res[segment+1].put(0)

        end = timer()
        print("Transport 4 take", end - start)
        #pdb.set_trace()
        #print(8)

        return input_res, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a,
detachment_b, detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b
```

**//Transport Scenario 7**

```
def transport7(input_res, rows_count, cols_count, vel_a, vel_b, process_a, process_b,
throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b, detach_a,
detach_b, leakage_a, leakage_b, lateral_a, lateral_b, segment, time):

    start=timer()
    flag = 0

    for i in range(rows_count):
        gap=0
        minimum=0
        for j in range(cols_count-1,-1,-1):
            if(particle[segment][i][j] == 0):
                gap += 1
                continue
            else:
                #Output
```

```
if(particle[segment][i][j] == 1):
    if(i==1):
        if(gap < vel_a):
            minimum = 0
        elif(gap >= vel_a):
            minimum = vel_a
    else:
        if(j>(cols_count-(stagger2/100)*cols_count) or gap < vel_a):
            minimum = 0
        elif(gap >= vel_a):
            minimum = vel_a

#Constant velocity
elif(particle[segment][i][j] == 2):
    if(i==1):
        if(gap < vel_b):
            minimum = 0
        elif(gap >= vel_b):
            minimum = vel_b
    else:
        if(j>(cols_count-(stagger2/100)*cols_count) or gap < vel_b):
            minimum = 0
        elif(gap >= vel_b):
            minimum = vel_b


#Movement
if(minimum > 0):
    if(particle[segment][i][j] == 1):
        if((j + vel_a) >= cols_count-1):
            throughput_a[segment] += 1
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            flag = 1
            input_res[segment+1].put(1)

        else:
            particle[segment][i][j+minimum] = 1
            span[segment][i][j+minimum] = span[segment][i][j] + minimum
            lifetime_cells[segment][i][j+minimum] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            if(span[segment][i][j+minimum] >= process_a):
                #detachment if motor exceeds its processivity
                particle[segment][i][j+minimum] = 0
                span[segment][i][j+minimum] = 0
                #print('y1')
                if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                    reservoir_a[segment][i][j].get()
```

```
                    leakage_a[segment][i][j] += 1


                reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j+mini
                mum]))

                  lifetime_cells[segment][i][j+minimum]=0
                  detachment_a[segment][i][j+minimum] += 1
                  detach_a[segment] += 1

        elif(particle[segment][i][j] == 2):
            if((j + vel_b) >= cols_count-1):
                throughput_b[segment] += 1
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                flag = 1
                input_res[segment+1].put(2)

            else:
                particle[segment][i][j+minimum] = 2
                span[segment][i][j+minimum] = span[segment][i][j] + minimum
                lifetime_cells[segment][i][j+minimum] = lifetime_cells[segment][i][j]
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                if(span[segment][i][j+minimum] >= process_b):
                    #detachment if motor exceeds its processivity
                    particle[segment][i][j+minimum] = 0
                    span[segment][i][j+minimum] = 0
                    #print('y2')
                    if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                        reservoir_b[segment][i][j].get()
                        leakage_b[segment][i][j] += 1


                reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j+mini
                mum]))

                  lifetime_cells[segment][i][j+minimum]=0
                  detachment_b[segment][i][j+minimum] += 1
                  detach_b[segment] += 1

    #Detachment
    elif(minimum == 0):
        if(particle[segment][i][j] == 1):
            if(j<((cols_count-(stagger2/100)*cols_count)-vel_a)):
                # Lateral association will happen to only cargos lost at initial stages
                channels = list(range(0,rows_count))        # All channels
                channels.remove(i)
                random.shuffle(channels)
```

```
    y1 = channels[0]
    channels.remove(channels[0])
    y2 = channels[0]

    if((j + vel_a) >= cols_count-1):
        throughput_a[segment] += 1
        particle[segment][i][j] = 0
        span[segment][i][j] = 0
        lifetime_cells[segment][i][j] = 0
        lateral_a[segment] += 1
        flag = 1
        input_res[segment+1].put(1)
    else:
        if(particle[segment][y1][j+vel_a] == 0):
            particle[segment][y1][j+vel_a] = 1
            span[segment][y1][j+vel_a] = span[segment][i][j] + vel_a
            lifetime_cells[segment][y1][j+vel_a] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1

        elif(particle[segment][y2][j+vel_a] == 0):
            particle[segment][y2][j+vel_a] = 1
            span[segment][y2][j+vel_a] = span[segment][i][j] + vel_a
            lifetime_cells[segment][y2][j+vel_a] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_a[segment] += 1

        else:
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            #print('y3')
            if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                reservoir_a[segment][i][j].get()
                leakage_a[segment][i][j] += 1


      reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

            lifetime_cells[segment][i][j]=0
            detachment_a[segment][i][j] += 1
            detach_a[segment] += 1

elif(j >= ((cols_count-(stagger2/100)*cols_count)-vel_a)):
    if(i == 1):
        if((j + vel_a) >= cols_count-1):
            throughput_a[segment] += 1
            particle[segment][i][j] = 0
```

```
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                flag = 1
                input_res[segment+1].put(1)

            else:
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                #print('y3')
                if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                    reservoir_a[segment][i][j].get()
                    leakage_a[segment][i][j] += 1


          reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                lifetime_cells[segment][i][j]=0
                detachment_a[segment][i][j] += 1
                detach_a[segment] += 1

        else:
            if(particle[segment][1][j+vel_a] == 0):
                particle[segment][1][j+vel_a] = 1
                span[segment][1][j+vel_a] = span[segment][i][j] + vel_a
                lifetime_cells[segment][1][j+vel_a] = lifetime_cells[segment][i][j]
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                lifetime_cells[segment][i][j] = 0
                lateral_a[segment] += 1

            else:
                particle[segment][i][j] = 0
                span[segment][i][j] = 0
                #print('y3')
                if(reservoir_a[segment][i][j].qsize()==reservoir_length):
                    reservoir_a[segment][i][j].get()
                    leakage_a[segment][i][j] += 1


          reservoir_a[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                lifetime_cells[segment][i][j]=0
                detachment_a[segment][i][j] += 1
                detach_a[segment] += 1

elif(particle[segment][i][j] == 2):
    if(j < ((cols_count-(stagger2/100)*cols_count)-vel_b)):
        channels = list(range(0,rows_count))        # All channels
        channels.remove(i)
        random.shuffle(channels)
        y1 = channels[0]
```

```
    channels.remove(channels[0])
    y2 = channels[0]

    if((j + vel_b) >= cols_count-1):
        throughput_b[segment] += 1
        particle[segment][i][j] = 0
        span[segment][i][j] = 0
        lifetime_cells[segment][i][j] = 0
        lateral_b[segment] += 1
        flag = 1
        input_res[segment+1].put(2)
    else:
        if(particle[segment][y1][j+vel_b]== 0):
            particle[segment][y1][j+vel_b] = 2
            span[segment][y1][j+vel_b] = span[segment][i][j] + vel_b
            lifetime_cells[segment][y1][j+vel_b] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_b[segment] += 1

        elif(particle[segment][y2][j+vel_b]== 0):
            particle[segment][y2][j+vel_b] = 2
            span[segment][y2][j+vel_b] = span[segment][i][j] + vel_b
            lifetime_cells[segment][y2][j+vel_b] = lifetime_cells[segment][i][j]
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            lifetime_cells[segment][i][j] = 0
            lateral_b[segment] += 1

        else:
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
            #print('y4')
            if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                reservoir_b[segment][i][j].get()
                leakage_b[segment][i][j] += 1


      reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

            lifetime_cells[segment][i][j]=0
            detachment_b[segment][i][j] += 1
            detach_b[segment] += 1

elif(j >= ((cols_count-(stagger2/100)*cols_count)-vel_b)):
    if(i == 1):
        if((j + vel_b) >= cols_count-1):
            throughput_b[segment] += 1
            particle[segment][i][j] = 0
            span[segment][i][j] = 0
```

```
                        lifetime_cells[segment][i][j] = 0
                        flag = 1
                        input_res[segment+1].put(2)
                    else:
                        particle[segment][i][j] = 0
                        span[segment][i][j] = 0
                        #print('y4')
                        if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                            reservoir_b[segment][i][j].get()
                            leakage_b[segment][i][j] += 1


                    reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                        lifetime_cells[segment][i][j]=0
                        detachment_b[segment][i][j] += 1
                        detach_b[segment] += 1

                else:
                    if(particle[segment][1][j+vel_b]== 0):
                        particle[segment][1][j+vel_b] = 2
                        span[segment][1][j+vel_b] = span[segment][i][j] + vel_b
                        lifetime_cells[segment][1][j+vel_b] = lifetime_cells[segment][i][j]
                        particle[segment][i][j] = 0
                        span[segment][i][j] = 0
                        lifetime_cells[segment][i][j] = 0
                        lateral_b[segment] += 1

                    else:
                        particle[segment][i][j] = 0
                        span[segment][i][j] = 0
                        #print('y4')
                        if(reservoir_b[segment][i][j].qsize()==reservoir_length):
                            reservoir_b[segment][i][j].get()
                            leakage_b[segment][i][j] += 1


                    reservoir_b[segment][i][j].put(props(time,lifetime_cells[segment][i][j]))

                        lifetime_cells[segment][i][j]=0
                        detachment_b[segment][i][j] += 1
                        detach_b[segment] += 1


            gap = 0

    if(flag == 0):
        input_res[segment+1].put(0)

    end = timer()
    print("Transport 7 take", end - start)
```

```
    #pdb.set_trace()
    #print(9)

    return input_res, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a,
detachment_b, detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b
```

## *4.10* **Main Function:**

```
print("Here we modelled a multisegment with " + str(scenario)+ " scenario formation")

column =
"Site,Waiting_A_Track_1,Waiting_B_Track_1,Waiting_A_Track_2,Waiting_B_Track_2,Wait
ing_A_Track_3,Waiting_B_Track_3,Leakage_A_Track_1,Leakage_B_Track_1,Leakage_A_T
rack_2,Leakage_B_Track_2,Leakage_A_Track_3,Leakage_B_Track_3\n"

dir_1 = "Scenario 147 Segment 1 Productive and Non-Productive.csv"
csv1 = open(dir_1, "a", newline='')
writer1 = csv.writer(csv1, dialect='excel')
csv1.write(column)

dir_4 = "Scenario 147 Segment 2 Productive and Non-Productive.csv"
csv4 = open(dir_4, "a", newline='')
writer4 = csv.writer(csv4, dialect='excel')
csv4.write(column)

dir_7 = "Scenario 147 Segment 3 Productive and Non-Productive.csv"
csv7 = open(dir_7, "a", newline='')
writer7 = csv.writer(csv7, dialect='excel')
csv7.write(column)


lines = [' ', 'Multisegment' + str(scenario) + 'at' + str(association_rate/2) + 'motors_sec']

with open(script_dir + '/'+ str(scenario) + '_at_' + str(association_rate/2) + '_motors_sec.txt',
'w') as f:
    f.write('\n'.join(lines))

lines = [' ', 'Multisegment' + str(scenario) + 'at' + str(association_rate/2) + 'motors_sec']

with open(script_dir + '/'+ str(scenario) + '_at_' + str(association_rate/2) +
'_motors_sec(per_sec).txt', 'w') as f:
    f.write('\n'.join(lines))

for iter in range(total_runtime):

    start = timer()
    for segment in range(segments):
        if(segment == 0):
```

```
            reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
            leakage_a, leakage_b  = lateral_association1(rows_count, cols_count,reservoir_a,
            reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b, leakage_a,
            leakage_b, segment, iter)

            input_res, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a,
            detachment_b, detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b,
            = transport1(input_res, rows_count, cols_count, vel_a, vel_b, process_a, process_b,
            throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b,
            detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b, segment, iter)

            input_res, motor_cargo_a, motor_cargo_b = initial_association1(input_res,
            motor_cargo_a, motor_cargo_b, segment)

        elif(segment == 1):

            reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
            leakage_a, leakage_b  = lateral_association4(rows_count, cols_count,reservoir_a,
            reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b, leakage_a,
            leakage_b, segment, iter)

            input_res, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a,
            detachment_b, detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b,
            = transport4(input_res, rows_count, cols_count, vel_a, vel_b, process_a, process_b,
            throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b,
            detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b, segment, iter)

            input_res, motor_cargo_a, motor_cargo_b = initial_association4(input_res,
            motor_cargo_a, motor_cargo_b, segment)

        elif(segment == 2):

            reservoir_a, reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b,
            leakage_a, leakage_b  = lateral_association7(rows_count, cols_count,reservoir_a,
            reservoir_b, reattachment_a, reattachment_b, reattach_a, reattach_b, leakage_a,
            leakage_b, segment, iter)

            input_res, throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a,
            detachment_b, detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b,
            = transport7(input_res, rows_count, cols_count, vel_a, vel_b, process_a, process_b,
            throughput_a, throughput_b, reservoir_a, reservoir_b, detachment_a, detachment_b,
            detach_a, detach_b, leakage_a, leakage_b, lateral_a, lateral_b, segment, iter)

            input_res, motor_cargo_a, motor_cargo_b = initial_association7(input_res,
            motor_cargo_a, motor_cargo_b, segment)


    worksheet9.write(iter+2, 0, iter+1)
    worksheet9.write(iter+2, 1, motor_cargo_a[0])
    worksheet9.write(iter+2, 2, motor_cargo_a[1])
    worksheet9.write(iter+2, 3, motor_cargo_a[2])
```

```
    worksheet9.write(iter+2, 4, motor_cargo_b[0])
    worksheet9.write(iter+2, 5, motor_cargo_b[1])
    worksheet9.write(iter+2, 6, motor_cargo_b[2])
    worksheet9.write(iter+2, 7, throughput_a[0])
    worksheet9.write(iter+2, 8, throughput_a[1])
    worksheet9.write(iter+2, 9, throughput_a[2])
    worksheet9.write(iter+2, 10, throughput_b[0])
    worksheet9.write(iter+2, 11, throughput_b[1])
    worksheet9.write(iter+2, 12, throughput_b[2])
    worksheet9.write(iter+2, 13, lateral_a[0])
    worksheet9.write(iter+2, 14, lateral_a[1])
    worksheet9.write(iter+2, 15, lateral_a[2])
    worksheet9.write(iter+2, 16, lateral_b[0])
    worksheet9.write(iter+2, 17, lateral_b[1])
    worksheet9.write(iter+2, 18, lateral_b[2])
    worksheet9.write(iter+2, 19, detach_a[0])
    worksheet9.write(iter+2, 20, detach_a[1])
    worksheet9.write(iter+2, 21, detach_a[2])
    worksheet9.write(iter+2, 22, detach_b[0])
    worksheet9.write(iter+2, 23, detach_b[1])
    worksheet9.write(iter+2, 24, detach_b[2])
    worksheet9.write(iter+2, 25, reattach_a[0])
    worksheet9.write(iter+2, 26, reattach_a[1])
    worksheet9.write(iter+2, 27, reattach_a[2])
    worksheet9.write(iter+2, 28, reattach_b[0])
    worksheet9.write(iter+2, 29, reattach_b[1])
    worksheet9.write(iter+2, 30, reattach_b[2])


    waiting_a = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
    waiting_b = [[[0 for j in range(cols_count)] for i in range(rows_count)] for l in
range(segments)]
    wait_a = [0 for l in range(segments)]
    wait_b = [0 for l in range(segments)]
    leak_a = [0 for l in range(segments)]
    leak_b = [0 for l in range(segments)]

    for l in range(segments):
        for i in range(rows_count):
            for j in range(cols_count):
                waiting_a[l][i][j] = reservoir_a[l][i][j].qsize()
                waiting_b[l][i][j] = reservoir_b[l][i][j].qsize()
                productive_a[l][i][j] += waiting_a[l][i][j]
                productive_b[l][i][j] += waiting_b[l][i][j]
                non_productive_a[l][i][j] += leakage_a[l][i][j]
                non_productive_b[l][i][j] += leakage_b[l][i][j]
                wait_a[l] += waiting_a[l][i][j]
                wait_b[l] += waiting_b[l][i][j]
                leak_a[l] += leakage_a[l][i][j]
                leak_b[l] += leakage_b[l][i][j]
```

```
    for l in range(segments):
        for i in range(rows_count):
            for j in range(cols_count):
                temp = int(j/5)
                if(particle[l][i][j] == 1):
                    heat_map_a[l][i][temp] += particle[l][i][j]
                elif(particle[l][i][j] == 2):
                    heat_map_b[l][i][temp] += particle[l][i][j]
    print("Done")
    #column9 = "Site, Waiting A Track 1, Waiting B Track 1, Waiting A Track 2, Waiting B
Track 2, Leakage A Track 1, Leakage B Track 1, Leakage A Track 2, Leakage B Track 2"

    d_a = [[0 for j in range(cols_count)] for l in range(segments)]
    d_b = [[0 for j in range(cols_count)] for l in range(segments)]
    r_a = [[0 for j in range(cols_count)] for l in range(segments)]
    r_b = [[0 for j in range(cols_count)] for l in range(segments)]
    w_a = [[0 for j in range(cols_count)] for l in range(segments)]
    w_b = [[0 for j in range(cols_count)] for l in range(segments)]
    l_a = [[0 for j in range(cols_count)] for l in range(segments)]
    l_b = [[0 for j in range(cols_count)] for l in range(segments)]

    for l in range(segments):
        for j in range(cols_count):
            for i in range(rows_count):
                d_a[l][j] += detachment_a[l][i][j]
                d_b[l][j] += detachment_b[l][i][j]
                r_a[l][j] += reattachment_a[l][i][j]
                r_b[l][j] += reattachment_b[l][i][j]
                w_a[l][j] += reservoir_a[l][i][j].qsize()
                w_b[l][j] += reservoir_b[l][i][j].qsize()
                l_a[l][j] += leakage_a[l][i][j]
                l_b[l][j] += leakage_b[l][i][j]

        for j in range(cols_count):
            d_a[l][j] = d_a[l][j]/total_runtime
            d_b[l][j] = d_b[l][j]/total_runtime
            r_a[l][j] = r_a[l][j]/total_runtime
            r_b[l][j] = r_b[l][j]/total_runtime

    end = timer()
    print("\n Run: "+ str(iter+1), (end - start))

    if(iter==0):
        continue

    temp1a = output_a[0][0][iter] + output_a[0][1][iter] + output_a[0][2][iter]
    temp1b = output_b[0][0][iter] + output_b[0][1][iter] + output_b[0][2][iter]
    temp2a = output_a[1][0][iter] + output_a[1][1][iter] + output_a[1][2][iter]
    temp2b = output_b[1][0][iter] + output_b[1][1][iter] + output_b[1][2][iter]
    temp3a = output_a[2][0][iter] + output_a[2][1][iter] + output_a[2][2][iter]
```

```
    temp3b = output_b[2][0][iter] + output_b[2][1][iter] + output_b[2][2][iter]


    line1 = [' ','Timestep:' + str((iter+1)*40) + 'msecs']
    line2 = [' ','Motor a: Seg 1:' + str(temp1a) + ', Seg 2:' + str(temp2a) + ', Seg 3:' + str(temp3a)]
    line3 = [' ','Motor b: Seg 1:' + str(temp1b) + ', Seg 2:' + str(temp2b) + ', Seg 3:' +
str(temp3b)]
    line4 = [' ','Throughput a: Seg 1:' + str(throughput_a[0]) + ', Seg 2:' + str(throughput_a[1]) +
', Seg 3:' + str(throughput_a[2])]
    line5 = [' ','Throughput b: Seg 1:' + str(throughput_b[0]) + ', Seg 2:' + str(throughput_b[1]) +
', Seg 3:' + str(throughput_b[2])]

    with open(script_dir + '/'+ str(scenario) + '_at_' + str(association_rate/2) + '_motors_sec.txt',
'a') as f:
        f.write('\n'.join(line1))
        f.write('\n'.join(line2))
        f.write('\n'.join(line3))
        f.write('\n'.join(line4))
        f.write('\n'.join(line5))


    if(count!=25):
        tmp1a += temp1a
        tmp1b += temp1b
        tmp2a += temp2a
        tmp2b += temp2b
        tmp3a += temp3a
        tmp3b += temp3b
        count += 1

    if(iter%25 == 0):

        line1 = [' ','Timestep:' + str(int(iter/25)) + 'sec']
        line2 = [' ','Motor a: Seg 1:' + str(tmp1a) + ', Seg 2:' + str(tmp2a) + ', Seg 3:' + str(tmp3a)]
        line3 = [' ','Motor b: Seg 1:' + str(tmp1b) + ', Seg 2:' + str(tmp2b) + ', Seg 3:' + str(tmp3b)]
        line4 = [' ','Throughput a: Seg 1:' + str(throughput_a[0]) + ', Seg 2:' + str(throughput_a[1])
+ ', Seg 3:' + str(throughput_a[2])]
        line5 = [' ','Throughput b: Seg 1:' + str(throughput_b[0]) + ', Seg 2:' + str(throughput_b[1])
+ ', Seg 3:' + str(throughput_b[2])]


        tmp1a = 0
        tmp1b = 0
        tmp2a = 0
        tmp2b = 0
        tmp3a = 0
        tmp3b = 0
        count = 0

        with open(script_dir + '/'+ str(scenario) + '_at_' + str(association_rate/2) +
'_motors_sec(per_sec).txt', 'a') as f:
```

```
            f.write('\n'.join(line1))
            f.write('\n'.join(line2))
            f.write('\n'.join(line3))
            f.write('\n'.join(line4))
            f.write('\n'.join(line5))


    if(iter>0 and iter%100 == 0):
        csv1 = open(dir_1, "a", newline='')
        writer1 = csv.writer(csv1, dialect='excel')
        for i in range(cols_count):
            strings = list()
            strings.append(str((i+1)))
            strings.append(str(productive_a[0][0][i]/iter))
            strings.append(str(productive_b[0][0][i]/iter))
            strings.append(str(productive_a[0][1][i]/iter))
            strings.append(str(productive_b[0][1][i]/iter))
            strings.append(str(productive_a[0][2][i]/iter))
            strings.append(str(productive_b[0][2][i]/iter))
            strings.append(str(non_productive_a[0][0][i]/iter))
            strings.append(str(non_productive_b[0][0][i]/iter))
            strings.append(str(non_productive_a[0][1][i]/iter))
            strings.append(str(non_productive_b[0][1][i]/iter))
            strings.append(str(non_productive_a[0][2][i]/iter))
            strings.append(str(non_productive_b[0][2][i]/iter))
            strings.append("\n")
            writer1.writerow(strings)
        #pdb.set_trace()
        csv1.close()

        csv4 = open(dir_4, "a", newline='')
        writer4 = csv.writer(csv4, dialect='excel')
        for i in range(cols_count):
            strings = list()
            strings.append(str((i+1)))
            strings.append(str(productive_a[1][0][i]/iter))
            strings.append(str(productive_b[1][0][i]/iter))
            strings.append(str(productive_a[1][1][i]/iter))
            strings.append(str(productive_b[1][1][i]/iter))
            strings.append(str(productive_a[1][2][i]/iter))
            strings.append(str(productive_b[1][2][i]/iter))
            strings.append(str(non_productive_a[1][0][i]/iter))
            strings.append(str(non_productive_b[1][0][i]/iter))
            strings.append(str(non_productive_a[1][1][i]/iter))
            strings.append(str(non_productive_b[1][1][i]/iter))
            strings.append(str(non_productive_a[1][2][i]/iter))
            strings.append(str(non_productive_b[1][2][i]/iter))
            strings.append("\n")
            writer4.writerow(strings)
        #pdb.set_trace()
        csv4.close()
```

```
csv7 = open(dir_7, "a", newline='')
writer7 = csv.writer(csv7, dialect='excel')
for i in range(cols_count):
    strings = list()
    strings.append(str((i+1)))
    strings.append(str(productive_a[2][0][i]/iter))
    strings.append(str(productive_b[2][0][i]/iter))
    strings.append(str(productive_a[2][1][i]/iter))
    strings.append(str(productive_b[2][1][i]/iter))
    strings.append(str(productive_a[2][2][i]/iter))
    strings.append(str(productive_b[2][2][i]/iter))
    strings.append(str(non_productive_a[2][0][i]/iter))
    strings.append(str(non_productive_b[2][0][i]/iter))
    strings.append(str(non_productive_a[2][1][i]/iter))
    strings.append(str(non_productive_b[2][1][i]/iter))
    strings.append(str(non_productive_a[2][2][i]/iter))
    strings.append(str(non_productive_b[2][2][i]/iter))
    strings.append("\n")
    writer7.writerow(strings)
#pdb.set_trace()
csv7.close()


t = str(iter+1).zfill(5)
#-----------
#Particle Graphs
#-----------
time = (iter)*0.04
fig, (ax0, ax1, ax2) = plt.subplots(ncols=1, nrows = 3, figsize=(8,6),
            gridspec_kw={"height_ratios":[1,1,1]})

temp=particle[0][0][0]
particle[0][0][0] = 2
c0 = ax0.pcolor(particle[0], linewidths = 0.2, cmap=cm)
ax0.set_title("Segment 1", fontsize=12)
ax0.set_xlabel('Lattice Sites')
ax0.set_ylabel('Track')
ax0.set_yticks((0.5, 1.5, 2.5),('1', '2', '3'))
particle[0][0][0] = temp

temp=particle[1][0][0]
particle[1][0][0] = 2
c1 = ax1.pcolor(particle[1], linewidths = 0.2, cmap=cm)
ax1.set_title("Segment 2", fontsize=12)
ax1.set_xlabel('Lattice Sites')
ax1.set_ylabel('Track')
custom_lines = [Patch(facecolor='white', edgecolor='k',label='Empty Space'),
            Patch(facecolor='blue', edgecolor='k',label='Motor A (faster)'),
            Patch(facecolor='red', edgecolor='k',label='Motor B (slower)')]
ax1.legend(handles=custom_lines,loc='center left', bbox_to_anchor=(1, 0.5))
```

```
        ax1.set_yticks((0.5, 1.5, 2.5),('1', '2', '3'))
        particle[1][0][0] = temp

        temp=particle[2][0][0]
        particle[2][0][0] = 2
        c2 = ax2.pcolor(particle[2], linewidths = 0.2, cmap=cm)
        ax2.set_title("Segment 3", fontsize=12)
        ax2.set_xlabel('Sites')
        ax2.set_ylabel('Track')
        ax1.set_yticks((0.5, 1.5, 2.5),('1', '2', '3'))
        particle[2][0][0] = temp

        fig.suptitle("\n".join(wrap("Movement of Kinesin motors Multisegmented tracks
(Scenario 147) After time %0.2f"%round(time,2) + ' secs', 90)), fontsize=13)
        fig.tight_layout()
        plt.subplots_adjust(top=0.88, hspace = 0.9)

        plt.gcf().text(.1, -0.01, "Data Values:", fontsize=12)
        plt.gcf().text(.1, -0.05, "Association rate: 10 motors/sec", fontsize=12)
        plt.gcf().text(.1, -0.08, "Processivity::   Motor A: 8000 nm       Motor B:
"+str(process_b*8)+" nm", fontsize=10)
        plt.gcf().text(.1, -0.11, "Input:: Motor A:     Segment 1: " +str(motor_cargo_a[0]) + "
Segment 2: " +str(motor_cargo_a[1]) + "     Segment 3: " +str(motor_cargo_a[2]), fontsize=10)
        plt.gcf().text(.1, -0.14, "Input:: Motor B:     Segment 1: " +str(motor_cargo_b[0]) + "
Segment 2: " +str(motor_cargo_b[1]) + "     Segment 3: " +str(motor_cargo_b[2]), fontsize=10)
        plt.gcf().text(.1, -0.17, "Output:: Motor A:  Segment 1: " +str(throughput_a[0]) + "
Segment 2: " +str(throughput_a[1]) + "     Segment 3: " +str(throughput_a[2]), fontsize=10)
        plt.gcf().text(.1, -0.20, "Output:: Motor B:  Segment 1: " +str(throughput_b[0]) + "
Segment 2: " +str(throughput_b[1]) + "     Segment 3: " +str(throughput_b[2]), fontsize=10)
        plt.gcf().text(.1, -0.23, "Productive Reservoir:: Motor A:         Segment 1: "
+str(wait_a[0]) + "     Segment 2: " +str(wait_a[1]) + "     Segment 3: " +str(wait_a[2]),
fontsize=10)
        plt.gcf().text(.1, -0.26, "Productive Reservoir:: Motor B:         Segment 1: "
+str(wait_b[0]) + "     Segment 2: " +str(wait_b[1]) + "     Segment 3: " +str(wait_b[2]),
fontsize=10)
        plt.gcf().text(.1, -0.29, "Non-productive Reservoir:: Motor A:  Segment 1: "
+str(leak_a[0]) + "     Segment 2: " +str(leak_a[1]) + "     Segment 3: " +str(leak_a[2]),
fontsize=10)
        plt.gcf().text(.1, -0.32, "Non-productive Reservoir:: Motor B:  Segment 1: "
+str(leak_b[0]) + "     Segment 2: " +str(leak_b[1]) + "     Segment 3: " +str(leak_b[2]),
fontsize=10)
        #plt.tight_layout()
        plt.savefig('Images/147 Particle_'+t+'.png',  dpi=100, bbox_inches = 'tight')
        plt.draw()
        plt.close()
        print("Graph 1 Done")


        t = str(iter).zfill(5)
        plt.figure(1)
        fig, (ax0, ax1, ax2, ax3, ax4, ax5) = plt.subplots(ncols=1, nrows = 6, figsize=(8,11),
```

```
                gridspec_kw={"height_ratios":[1,1,1,1,1,1]})

        ax0.plot(cols1, w_a[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols1, w_b[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_ylim(bottom=0)
        ax0.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax0.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax0.set_title("\n".join(wrap('No of motors waiting in the productive reservoir of Segment
1', 70)), fontsize=14)
        #ax0.legend(loc="upper right")

        ax1.plot(cols1, l_a[0], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols1, l_b[0], alpha=0.5, color="Red", label="Motor B")
        ax1.set_ylim(bottom=0)
        ax1.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax1.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax1.set_title("\n".join(wrap('No of motors leaked from the productive reservoir of
Segment 1', 70)), fontsize=14)
        #ax1.legend(loc="upper right")

        ax2.plot(cols1, w_a[1], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols1, w_b[1], alpha=0.5, color="Red", label="Motor B")
        ax2.set_ylim(bottom=0)
        ax2.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax2.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax2.set_title("\n".join(wrap('No of motors waiting in the productive reservoir of Segment
2', 70)), fontsize=14)
        ax2.legend(loc="upper right")

        ax3.plot(cols1, l_a[1], alpha=0.6, color="Blue", label="Motor A")
        ax3.plot(cols1, l_b[1], alpha=0.5, color="Red", label="Motor B")
        ax3.set_ylim(bottom=0)
        ax3.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax3.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax3.set_title("\n".join(wrap('No of motors leaked from the productive reservoir of
Segment 2', 70)), fontsize=14)
        ax3.legend(loc="upper right")

        ax4.plot(cols1, w_a[2], alpha=0.6, color="Blue", label="Motor A")
        ax4.plot(cols1, w_b[2], alpha=0.5, color="Red", label="Motor B")
        ax4.set_ylim(bottom=0)
        ax4.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax4.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax4.set_title("\n".join(wrap('No of motors waiting in the productive reservoir of Segment
3', 70)), fontsize=14)
        #ax4.legend(loc="upper right")

        ax5.plot(cols1, l_a[2], alpha=0.6, color="Blue", label="Motor A")
        ax5.plot(cols1, l_b[2], alpha=0.5, color="Red", label="Motor B")
        ax5.set_ylim(bottom=0)
        ax5.set_xlabel('Corresponding Lattice Site', fontsize=13)
```

```
        ax5.set_ylabel("\n".join(wrap('No of motors', 22)), fontsize=13)
        ax5.set_title("\n".join(wrap('No of motors leaked from the productive reservoir of
Segment 3', 70)), fontsize=14)
        #ax5.legend(loc="upper right")

        fig.suptitle("\n".join(wrap('Dynamic state of Productive Reservoir Metrics in
Multisegmented tracks (Scenario 147) after time %0.2f'%round(time,2) + ' secs', 70)),
fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(top=0.88, hspace = 0.95)
        plt.gcf()
        plt.savefig('Reservoir1/147 Productive_'+t+'.png',  dpi=100, bbox_inches = 'tight')
        #plt.show()
        plt.close()
        print("Graph 2 Done")
        plt.figure(2)
        fig, (ax0, ax1, ax2, ax3, ax4, ax5) = plt.subplots(ncols=1, nrows = 6, figsize=(8,11),
                gridspec_kw={"height_ratios":[1,1,1,1,1,1]})

        ax0.plot(cols1, d_a[0], alpha=0.6, color="Blue", label="Motor A")
        ax0.plot(cols1, d_b[0], alpha=0.5, color="Red", label="Motor B")
        ax0.set_ylim(bottom=0)
        ax0.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax0.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=12)
        ax0.set_title("\n".join(wrap('No of times motors detached into reservoir from lattice site of
Segment 1', 80)), fontsize=14)
        #ax0.legend(loc="upper right")

        ax1.plot(cols1, r_a[0], alpha=0.6, color="Blue", label="Motor A")
        ax1.plot(cols1, r_b[0], alpha=0.5, color="Red", label="Motor B")
        ax1.set_ylim(bottom=0)
        ax1.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax1.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=12)
        ax1.set_title("\n".join(wrap('No of times motors reattached from reservoir into lattice site
of Segment 1', 80)), fontsize=14)
        #ax1.legend(loc="upper right")

        ax2.plot(cols1, d_a[1], alpha=0.6, color="Blue", label="Motor A")
        ax2.plot(cols1, d_b[1], alpha=0.5, color="Red", label="Motor B")
        ax2.set_ylim(bottom=0)
        ax2.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax2.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=12)
        ax2.set_title("\n".join(wrap('No of times motors detached into reservoir from lattice site of
Segment 2', 80)), fontsize=14)
        ax2.legend(loc="upper right")

        ax3.plot(cols1, r_a[1], alpha=0.6, color="Blue", label="Motor A")
        ax3.plot(cols1, r_b[1], alpha=0.5, color="Red", label="Motor B")
        ax3.set_ylim(bottom=0)
        ax3.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax3.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=12)
```

```
        ax3.set_title("\n".join(wrap('No of times motors reattached from reservoir into lattice site
of Segment 2', 80)), fontsize=14)
        ax3.legend(loc="upper right")

        ax4.plot(cols1, d_a[2], alpha=0.6, color="Blue", label="Motor A")
        ax4.plot(cols1, d_b[2], alpha=0.5, color="Red", label="Motor B")
        ax4.set_ylim(bottom=0)
        ax4.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax4.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=12)
        ax4.set_title("\n".join(wrap('No of times motors detached into reservoir from lattice site of
Segment 3', 80)), fontsize=14)
        #ax4.legend(loc="upper right")

        ax5.plot(cols1, r_a[2], alpha=0.6, color="Blue", label="Motor A")
        ax5.plot(cols1, r_b[2], alpha=0.5, color="Red", label="Motor B")
        ax5.set_ylim(bottom=0)
        ax5.set_xlabel('Corresponding Lattice Site', fontsize=13)
        ax5.set_ylabel("\n".join(wrap('Normalized No of Motors', 12)), fontsize=12)
        ax5.set_title("\n".join(wrap('No of times motors reattached from reservoir into lattice site
of Segment 3', 80)), fontsize=14)
        #ax5.legend(loc="upper right")

        fig.suptitle("\n".join(wrap('Cumulative state of Productive Reservoir Metrics in
Multisegmented tracks (Scenario 147) after time %0.2f'%round(time,2) + ' secs', 70)),
fontsize=16)
        fig.tight_layout()
        plt.subplots_adjust(top=0.88, hspace = 0.95)
        plt.gcf()
        plt.savefig('Reservoir2/147 Productive_'+t+'.png',  dpi=100, bbox_inches = 'tight')
        #plt.show()
        plt.close()
        print("Graph 3 Done")

        if(iter == 15000):
            fig5 = plt.figure()
            widths = [1, 1, 1, 0.05]
            heights = [1, 1]
            spec5 = fig5.add_gridspec(ncols=4, nrows=2, width_ratios=widths,
height_ratios=heights)
            for row in range(2):
                for col in range(4):
                    ax = fig5.add_subplot(spec5[row, col])
                    top1 = max(max(max(heat_map_a)))
                    top2 = max(max(max(heat_map_b)))
            #         top1 = 500
            #         top2 = 2000
                    if(row==0 and col==0):
                        time=iter*0.04
                        c0 = ax.pcolor(heat_map_a[0], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top1)
                        ax.set_title("Segment 1: Faster", fontsize=10)
```

```
                ax.set_xlabel('Sites', fontsize=11)
                ax.set_ylabel('Track', fontsize=11)
                ax.set_xlim([0, 200])
                plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

            elif(row==0 and col==1):
                time=iter*0.04
                c1 = ax.pcolor(heat_map_a[1], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top1)
                ax.set_title("Segment 2: Faster", fontsize=10)
                ax.set_xlabel('Sites', fontsize=11)
                #ax.set_ylabel('Track')
                ax.set_xlim([0, 200])
                plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

            elif(row==0 and col==2):
                time=iter*0.04
                c2 = ax.pcolor(heat_map_a[2], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top1)
                ax.set_title("Segment 3: Faster", fontsize=10)
                ax.set_xlabel('Sites', fontsize=11)
                #ax.set_ylabel('Track')
                ax.set_xlim([0, 200])
                plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

            elif(row==0 and col==3):
                fig5.colorbar(c0, cax=ax)

            elif(row==1 and col==0):
                time=iter*0.04
                c3 = ax.pcolor(heat_map_b[0], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top2)
                ax.set_title("Segment 1: Slower", fontsize=10)
                ax.set_xlabel('Sites', fontsize=11)
                ax.set_ylabel('Track', fontsize=11)
                ax.set_xlim([0, 200])
                plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

            elif(row==1 and col==1):
                time=iter*0.04
                c4 = ax.pcolor(heat_map_b[1], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top2)
                ax.set_title("Segment 2: Slower", fontsize=10)
                ax.set_xlabel('Sites', fontsize=11)
                #ax.set_ylabel('Track')
                ax.set_xlim([0, 200])
                plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
```

```
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

            elif(row==1 and col==2):
                time=iter*0.04
                c5 = ax.pcolor(heat_map_b[2], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top2)
                ax.set_title("Segment 3: Slower", fontsize=10)
                ax.set_xlabel('Sites', fontsize=11)
                #ax.set_ylabel('Track')
                ax.set_xlim([0, 200])
                plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

            elif(row==1 and col==3):
                fig5.colorbar(c3, cax=ax)


        fig5.suptitle("\n".join(wrap("Distribution heat map of motors in Multisegmented Track
(Scenario 147) after time 600" + ' secs', 60)), fontsize=12, y=1.05)
        fig5.tight_layout()
        plt.subplots_adjust(wspace = 0.3, hspace = 0.6)
        plt.draw()
        plt.savefig('147 Heatmap.eps', format='eps', dpi=1200, bbox_inches = 'tight')
        plt.savefig('147 Heatmap.svg', format='svg', dpi=1200, bbox_inches = 'tight')
        plt.savefig('./147 Heatmap.svg', format='svg', dpi=1200, bbox_inches = 'tight')
        plt.savefig('147 Heatmap.png', format='png', dpi=1200, bbox_inches = 'tight')
        plt.savefig('./147 Heatmap.png', format='png', dpi=1200, bbox_inches = 'tight')
        plt.close()

        print("Final Graph Done")

workbook.close()
```

## *4.11* **Heatmap Generation:**

```
fig5 = plt.figure()
widths = [1, 1, 1, 0.05]
heights = [1, 1]
spec5 = fig5.add_gridspec(ncols=4, nrows=2, width_ratios=widths, height_ratios=heights)
for row in range(2):
    for col in range(4):
        ax = fig5.add_subplot(spec5[row, col])
        top1 = max(max(max(heat_map_a)))
        top2 = max(max(max(heat_map_b)))
#         top1 = 500
#         top2 = 2000
        if(row==0 and col==0):
            time=iter*0.04
            c0 = ax.pcolor(heat_map_a[0], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top1)
```

```
            ax.set_title("Segment 1: Faster", fontsize=10)
            ax.set_xlabel('Sites', fontsize=11)
            ax.set_ylabel('Track', fontsize=11)
            ax.set_xlim([0, 200])
            plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

        elif(row==0 and col==1):
            time=iter*0.04
            c1 = ax.pcolor(heat_map_a[1], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top1)
            ax.set_title("Segment 2: Faster", fontsize=10)
            ax.set_xlabel('Sites', fontsize=11)
            #ax.set_ylabel('Track')
            ax.set_xlim([0, 200])
            plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

        elif(row==0 and col==2):
            time=iter*0.04
            c2 = ax.pcolor(heat_map_a[2], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top1)
            ax.set_title("Segment 3: Faster", fontsize=10)
            ax.set_xlabel('Sites', fontsize=11)
            #ax.set_ylabel('Track')
            ax.set_xlim([0, 200])
            plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

        elif(row==0 and col==3):
            fig5.colorbar(c0, cax=ax)

        elif(row==1 and col==0):
            time=iter*0.04
            c3 = ax.pcolor(heat_map_b[0], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top2)
            ax.set_title("Segment 1: Slower", fontsize=10)
            ax.set_xlabel('Sites', fontsize=11)
            ax.set_ylabel('Track', fontsize=11)
            ax.set_xlim([0, 200])
            plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

        elif(row==1 and col==1):
            time=iter*0.04
            c4 = ax.pcolor(heat_map_b[1], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top2)
            ax.set_title("Segment 2: Slower", fontsize=10)
            ax.set_xlabel('Sites', fontsize=11)
            #ax.set_ylabel('Track')
            ax.set_xlim([0, 200])
```

```
        plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

    elif(row==1 and col==2):
        time=iter*0.04
        c5 = ax.pcolor(heat_map_b[2], linewidths = 0.02, cmap='viridis_r', vmin=0 ,
vmax=top2)
        ax.set_title("Segment 3: Slower", fontsize=10)
        ax.set_xlabel('Sites', fontsize=11)
        #ax.set_ylabel('Track')
        ax.set_xlim([0, 200])
        plt.setp(ax, xticks=[0,50,100,150,200], xticklabels=['0','250','500','750','1000'],
yticks=[0.5, 1.5, 2.5], yticklabels=['1', '2', '3'])

    elif(row==1 and col==3):
        fig5.colorbar(c3, cax=ax)


fig5.suptitle("\n".join(wrap("Distribution heat map of motors in Multisegmented Track
(Scenario 147) after time %0.2f"%round(time,2) + ' secs', 60)), fontsize=12, y=1.05)
fig5.tight_layout()
plt.subplots_adjust(wspace = 0.3, hspace = 0.6)
plt.draw()
plt.savefig('147 Heatmap.eps', format='eps', dpi=1200, bbox_inches = 'tight')
plt.savefig('147 Heatmap.svg', format='svg', dpi=1200, bbox_inches = 'tight')
plt.savefig('./147 Heatmap.svg', format='svg', dpi=1200, bbox_inches = 'tight')
plt.savefig('147 Heatmap.png', format='png', dpi=1200, bbox_inches = 'tight')
plt.savefig('./147 Heatmap.png', format='png', dpi=1200, bbox_inches = 'tight')
plt.close()

print("Final Graph Done")
```

## *4.12* Heatmap Generation:

### *4.12.1* Particle GIF:

```
# filepaths
fp_in = "Images/147 Particle_*.png"
fp_out = "GIFs/147 Particle.gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
         save_all=True, duration=200, loop=0)
```

### *4.12.2* Productive Reservoir Dynamics at Dynamic Time Stamp:

```
# filepaths
fp_in = "Reservoir1/147 Productive_*.png"
fp_out = "GIFs/147 Dynamic Productive.gif"
```

```
# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
         save_all=True, duration=200, loop=0)
```

### *4.12.3* **Productive Reservoir Dynamics at Cumulative Time Stamp: __**

```
# filepaths
fp_in = "Reservoir2/147 Productive_*.png"
fp_out = "GIFs/147 Cumulative Productive.gif"

# https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#gif
img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in))]
img.save(fp=fp_out, format='GIF', append_images=imgs,
         save_all=True, duration=200, loop=0)
```