infectiousmontecarlo.py

```python
# -*- coding: utf-8 -*-
"""InfectiousMonteCarlo.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1hVphcNjYUKSSztrBmXcEWYU638DaEE_J
"""

import numpy as np
import random
import matplotlib.pyplot as plt
import numba
import time
import os
from datetime import datetime

def create_lattice(lattice_length, T_num, B_num):
    lattice = np.zeros([lattice_length, lattice_length])

    # Place T's randomly
    t_coords = np.empty((2, 0), dtype=int)  # Initialize as an empty array
    while t_coords.shape[1] < T_num:  # Check the number of columns
        t = np.array([[np.random.randint(lattice_length)], [np.random.randint(lattice_length)]])
        if not np.any(np.all(t == t_coords, axis=0)):
            t_coords = np.hstack((t_coords, t))
            lattice[t[0, 0], t[1, 0]] = 1

    # Place B's randomly
    b_coords = np.empty((2, 0), dtype=int)  # Initialize as an empty array
    while b_coords.shape[1] < B_num:  # Check the number of columns
        b = np.array([[np.random.randint(lattice_length)], [np.random.randint(lattice_length)]])
        if not np.any(np.all(b == b_coords, axis=0)) and not np.any(np.all(b == t_coords, axis=0)):
            b_coords = np.hstack((b_coords, b))
            lattice[b[0, 0], b[1, 0]] = 2

    empty_coords = np.argwhere(lattice == 0).T

    return lattice, t_coords, b_coords, empty_coords

# showing lattices as they evolve

def lattice_plots(lattice_history, selected_indices):

    cmap = plt.cm.colors.ListedColormap(['white', 'blue', 'red'])

    for i in range(len(selected_indices)):

        # Create a plot
        plt.imshow(lattice_history[i], cmap=cmap, extent=[0, size, 0, size])
        plt.colorbar(ticks=[0, 1, 2], label="Legend")
        plt.title("Lattice with T's (Blue) and B's (Red)")


```

```
        #plt.title(f"Lattice at Iteration {selected_indices[i]}")  # Add a title with the iteration number

        #fig_name = f"Plot\Lattice_iter{selected_indices[i]}"
        # plt.savefig(fig_name)
        plt.show()

def position_random(pos): ###Problem
    col = np.random.randint(pos.shape[1]) ###REVIEW
    p = pos[:,col]
    #print("random",col,pos.shape[1]-1 )
    return p, col

def energy(lattice, ID_in, pos_hypo, interaction_matrix):
    s = lattice.shape[0]-1
    i = pos_hypo[0] # x coordinate
    j = pos_hypo[1] # y coordinate

    if i==0:
        up = lattice[s,j]
    else:
        up = lattice[i-1,j]
    up = int(up)

    if i == s:
        down = lattice[0,j]
    else:
        down = lattice[i+1,j]
    down = int(down)

    if j == 0:
        left = lattice[i,s]
    else:
        left = lattice[i,j-1]
    left = int(left)

    if j == s:
        right = lattice[i,0]
    else:
        right = lattice[i,j+1]
    right = int(right)


    E = -(interaction_matrix[ID_in, up] + interaction_matrix[ID_in, down] + interaction_matrix[ID_in, left] +
          interaction_matrix[ID_in, right])
    return E

# total energy of lattice
def lattice_energy(lattice, eps):

    E_total = 0
    rows, cols = lattice.shape

    for i in range(rows):
        for j in range(cols):
            val = int(lattice[i, j])
            E_total += energy(lattice, val, (i, j), eps)

    E_total = E_total / 2
```

```
114
115     return E_total
116
117 def evaluate_particle_addB(lattice, pos2, pos1, pos0, T, E_total, eps):
118     #print("pos1before",pos1.shape)
119     pb, colb = position_random(pos0) #pick a hole to put bacteria
120
121     #ID_in= lattice[pb[0], pb[1]] #change it in the lattice
122     ID_B = 2
123     Efin = energy(lattice, ID_B, pb, eps) #evaluate neighbouring energy
124     #ID_in = lattice[pb[0], pb[1]]
125     ID_empty = 0
126     Ein = energy(lattice,ID_empty, pb, eps) #evaluate neighbouring energy before
127
128     muT, muB = -1, -25
129
130     #print("Efin , Ein", Efin, Ein)
131     Ediff = Efin - Ein
132     #print("Ediff ", Ediff)
133
134     if Ediff < 0 :
135         add = True
136
137     else:
138         probability = np.exp(-(Ediff -muT* pos1.shape[1] - muB* pos2.shape[1])/T)
139         if random.random() < probability: # random.random gives between 0 and 1. Hence
higher prob -> more move
140             add = True
141         else:
142             add = False
143
144     if add:
145         lattice[pb[0], pb[1]] = 2
146         #print("before",pos2, pb)
147         pos0 = np.delete(pos0, colb, axis=1)
148         pos2 = np.hstack((pos2, np.array([pb]).reshape(-1, 1)))
149         #print(pos0)
150         E_total = E_total + Ediff
151         E_total = float(E_total)
152         #print(E_total)
153         #print("Ediff:", Ediff)
154
155     else:
156         lattice[pb[0], pb[1]] = 0
157     #print(E_total, Ediff)
158     return lattice, pos2, pos0, E_total, add
159
160 # check if object moves. pos1 is the coordinates of all objects where one is to be moved.
161 # most likely a Tcell
162 # pos0 are coordinates of holes in the lattice
163
164 def evaluate_particle_moveT(lattice, pos1, pos0, T, E_total, eps):
165     #print("pos1before",pos1.shape)
166     p1, col1 = position_random(pos1)
167
168     ID_in = lattice[p1[0], p1[1]]
169     ID_in = int(ID_in)
170     Ein = energy(lattice,ID_in, p1, eps)
171
172     p0, col0 = position_random(pos0)
```

```
    # seeing what energy would be for particle if it moved the the chosen empty location
    lattice[p1[0], p1[1]] = 0 # temporarily moving object so not to be seen as neighbor by itself
    Efin = energy(lattice, ID_in, p0, eps)
    #print("Efin , Ein", Efin, Ein)
    Ediff = Efin - Ein


    if Ediff < 0 :
        move = True

    else:
        probability = np.exp(-Ediff/T)
        if random.random() < probability: # random.random gives between 0 and 1. Hence higher prob -> more move
            move = True
        else:
            move = False

    if move:
        lattice[p0[0], p0[1]] = 1
        # update arrays containing coordinates of 0's and 1's
        pos1[:,col1] = [p0[0], p0[1]]
        pos0[:,col0] = [p1[0], p1[1]]
        #print(pos1)
        E_total = E_total + Ediff
        E_total = float(E_total)
        #print("Ediff:", Ediff)

    else:
        lattice[p0[0], p0[1]] = 0
        lattice[p1[0], p1[1]] = 1
    #print(E_total, Ediff)
    return lattice, pos1, pos0, E_total, move

def gridprint(lattice,lattice_length):
    cmap = plt.cm.colors.ListedColormap(['white', 'blue', 'red'])
    plt.imshow(lattice, cmap=cmap, extent=[0, lattice_length, 0, lattice_length])
    plt.colorbar(ticks=[0, 1, 2], label="Legend")
    plt.title("Lattice with T's (Blue) and B's (Red)")
    #plt.grid(True, linewidth=0.5, color='black')
    plt.show()

    return 0

#B = np.zeros(len(T))
#B

def monte_carlo(Temp, eps, lattice_length, T_num_in, B_num_in, num_runs, num_lattices_to_store=None):


    #gridprint(lattice,lattice_length)
    #print('p0:',pos0,'T:',pos1,'B:',pos2)

    E_history = {}
    #pos0_hist=[]
    #pos1_hist=[]
    #pos2_hist=[]
    Tcell = []
    B_num = np.zeros(len(Temp))
```

```
    pos2t=[]
    for ind, t in enumerate(Temp):
        #lattice_history = []
        E_history_for_Temp = []
        #pos0t=[]
        #pos1t=[]
        pos2_each_T = []
        lattice, pos1, pos2, pos0 = create_lattice(lattice_length, T_num_in, B_num_in)
        E_lattice = lattice_energy(lattice, eps)

        #for temperature in T:
        for i in range(0,num_runs): # change to from one and append initial E and lattice to outisde
            E_history_for_Temp.append(E_lattice)

            pos2_each_T.append(pos2.shape)

            if any(pos0[1]):
                lattice, pos1, pos0, E_lattice, move = evaluate_particle_moveT(lattice, pos1, pos0, t, E_lattice, eps)

                lattice, pos2, pos0, E_lattice, add = evaluate_particle_addB(lattice, pos2, pos1, pos0, t, E_lattice, eps)
                #pos2T_shape = pos2[1].size
                #gridprint(lattice,lattice_length)
                #print(add, move)
                #print(pos1)
            #pos0t.append(pos0)
            #pos1t.append(pos1)
        pos2t.append(pos2_each_T)

        B_num[ind] = pos2[1].size
        Tcell.append(pos1.shape[1])


        E_history[t] = E_history_for_Temp.copy()
        #gridprint(lattice,lattice_length)

        #pos0_hist.append(pos0t)
        #pos1_hist.append(pos1t)
        #pos2_hist.append(pos2t)

    #SAVE ON FILE
    current_datetime = datetime.now()
    datetime_str = current_datetime.strftime('%Y%m%d-%H-%M')
    run_name = f'{datetime_str}'

    return lattice, E_history, B_num,pos2t #, pos0_hist, pos1_hist, pos2_hist
    #return lattice

# the interaction matrix can be used to decide how many bacteria are able to multiply.
# if surrounded by T cells -> no division
# the body is modelled by an N by N lattice

num_runs = 10_000
#Temp = 0.2
T = np.arange(3,0.01,-0.1)
#T = np.arange(.1,.01,-0.1) ##Test
size = 40

T_num_in = int(size**2/2)    # number of initial T-cells
```

```
289 B_num_in = 1
290
291 BB_int = 1      # interaction energy between bacterias
292 TT_int = -1      # interaction energy between T-cells
293 BT_int = 4     # interaction energy between bacteria and T-cells
294 interaction_matrix = np.array([
295     [0, 0, 0],
296     [0, TT_int, BT_int],
297     [0, BT_int, BB_int]
298 ])
299
300 lattice, E_history, B_num, pos2t = monte_carlo(T, interaction_matrix, size, T_num_in,
    B_num_in, num_runs, num_lattices_to_store=None)
301 #print(len(pos2t[0]))
302
303 plt.figure()
304 plt.plot(T,B_num)
305 plt.xlabel('T')
306 plt.ylabel('B_num')
307 plt.title(f'Size: {size} ,Runs: {num_runs}')
308 plt.show()
309
310 def mean_energy(T, E_history, ind_equilibrium):
311
312     E = E_history
313     E_mean = np.zeros([len(T)])
314     E_variance = np.zeros([len(T)])
315
316     for ind,t in enumerate(T):
317         E_mean[ind] = np.mean(E[t][ind_equilibrium:-1])
318         E_variance[ind] = np.var(E[t][ind_equilibrium:-1])
319
320     return E_mean, E_variance
321
322 #ind_equi = int((2/3)*num_runs)
323 ind_equi = int((3/8)*num_runs) # index where equilibrium is assumed.
324 E_mean, E_var = mean_energy(T, E_history, ind_equi)
325
326 plt.figure()
327 plt.plot(T,E_mean)
328 plt.xlabel('T')
329 plt.ylabel('U')
330 plt.title(f'Size: {size} ,Runs: {num_runs}')
331 plt.show()
332 #lattice_plots(lattice, np.arange(0,100,5))
333
334 # SAVE DATA
335
336 current_dir = os.getcwd()
337
338 # directory of Data folder
339 new_dir = f'{current_dir}/Data/'
340
341 # Save data there
342 file_spec = 'runs10000'
343 file_name = f'{run_name}_{file_spec}.npz'
344 file_dir = f'{new_dir}{file_name}'
345
346 np.savez(file_dir,
347          T = Temp,
```

```
348          eps = interaction_matrix,
349          Tcell_num = T_num_in,
350          B_num = B_num,
351          size = size,
352          E_mean = E_mean,
353          E_variance = E_var,
354          num_runs = num_runs,
355         )
356 #np.savez(file_dir, T = T, num_runs = num_runs, size = size, eps=eps, T_num_in , B_num =
    B_num, E_mean = E_mean, E_variance=E_var, execution_time=execution_time )
357
358
```