Tarea 2: Paradojas de simulaciones de discos duros en una caja

Sofía Thiel Pizarro
B36953

26 de febrero de 2021

1. Equiprobabilidad

1.1 A1

- Al añadir un print en el código dentro del if para que imprima $condition_hit$ observamos que imprime un montón de unos, los cuales corresponden a la cantidad de aciertos que hay para cada configuración en cada corrida. Por lo tanto, la variable $condition_hit$ revisa y contabiliza cuando la configuración es de las deseadas y cuando no. Quiere decir que $condition_hit$ va a ser True cuando hay un acierto para alguna configuración deseada y será False cuando no hay acierto o se rechaza.
  El $min(max(...)...)$ lo que propone es si la configuración es deseada o no, el resultado que tira es booleano y como la siguiente línea es el $condition_hit$ vemos que este necesita de la respuesta de $min(max(...)...)$ para revisar la configuración y escribir si es un 1 o un 0.

- Corriendo el programa obtenemos que los aciertos son:

n_runs = 10^4		
Corrida 1	Corrida 2	Corrida 3
a = 0	a = 2	a = 2
b = 0	b = 3	b = 4
c = 2	c = 1	c = 1

n_runs = 10^5		
Corrida 1	Corrida 2	Corrida 3
a = 24	a = 14	a = 10
b = 20	b = 16	b = 9
c = 8	c = 6	c = 17

n_runs = 10^6		
Corrida 1	Corrida 2	Corrida 3
a = 115	a = 110	a = 107
b = 127	b = 120	b = 119
c = 110	c = 119	c = 113

  Sí se observa equiprobabilidad, ya que para un número mayor de n_runs es evidente que aumenta el numero de configuraciones pero siempre se encuentran muy cercanos los valores entre sí. Al igual que para los n_runs más bajos se observa que los valores se encuentran cercanos entre sí.

- Al hacer el cambio sobre σ y del_xy podemos notar que la cantidad de aciertos aumenta radicalmente, esto se debe a que se le da un mayor rango a las configuraciones válidas, quiere decir que hay más maneras de que una configuración sea válida.

- Dicho algoritmo necesita de la equiprobabilidad cada vez que se reordena o se limpia y empieza desde cero después de encontrar un desacierto. En otras palabras, cada vez que la configuración es inválida el sistema se limpia y comienza de nuevo sin sezgar la probabilidad de dicha configuración (porque si siguiera desde la posición antes del overlap no se cumpliría el principio de la equiprobabilidad) y es ahí donde se implementa.

1.2 A2

Los aciertos que se obtienen al correr el programa son:

n_runs = 10^4		
Corrida 1	Corrida 2	Corrida 3
a = 0 b = 0 c = 1	a = 0 b = 1 c = 1	a = 1 b = 2 c = 0
n_runs = 10^5		
Corrida 1	Corrida 2	Corrida 3
a = 27 b = 33 c = 49	a = 22 b = 58 c = 74	a = 9 b = 19 c = 33
n_runs = 10^6		
Corrida 1	Corrida 2	Corrida 3
a = 134 b = 237 c = 356	a = 113 b = 218 c = 335	a = 132 b = 250 c = 353

El código utilizado es el siguiente:

```
import random, math
def markov_disk_box(L, sigma):
    delta=0.1
    a=random.choice(L)
    b=[a[0]+random.uniform(-delta, delta), a[1]+random.uniform(-delta,delta)]
    min_dist= min((b[0]-c[0])**2 + (b[1]-c[1])**2 for c in L if c!= a)
    box_cond = min(b[0], b[1])< sigma or max(b[0], b[1])> 1.0 - sigma
    if not (box_cond or min_dist < 4.0 * sigma**2):
        a[:] = b
    return L

sigma = 0.15
del_xy = 0.05
n_steps = 10000
n_runs= 3
conf_a = ((0.30, 0.30), (0.30, 0.70), (0.70, 0.30), (0.70, 0.70))
conf_b = ((0.20, 0.20), (0.20, 0.80), (0.75, 0.25), (0.75, 0.75))
conf_c = ((0.30, 0.20), (0.30, 0.80), (0.70, 0.20), (0.70, 0.70))
L = [[0.25,0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]

configurations = [conf_a, conf_b, conf_c]
hits = {conf_a: 0, conf_b: 0, conf_c: 0}
for run_i in range(n_runs):
    L = [[0.25,0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
    for steps in range (n_steps):
```

```
        x_vec = markov_disk_box(L, sigma)
        for conf in configurations:
            condition_hit = True
            for b in conf:
                condition_b = min(max(abs(a[0]-b[0]),abs(a[1]-b[1]))for a in x_vec) < del_xy
                condition_hit *= condition_b
            if condition_hit:
                hits[conf] += 1

    print (conf, hits[conf])
```

Se observa que entre los valores obtenidos en las tablas hay deltas más grandes comparado con la sección anterior. El código no presenta buena equiprobabilidad cuando se topa con una configuración rechazada, por ende el error aumenta como con las cadenas de Markov de la tarea 1. Es por esta razón que los valores se dispersan más. Aún así conforme aumenta el n_runs aumenta la cantidad de aciertos.

1.3 A3

- En las secciones anteriores (A1 y A2) se podía ver que eran algoritmos estadísticos y aleatorios, pero en este caso tenemos un comportamiento caótico completamente, quiere decir que depende de las condiciones iniciales y estas de manera determinista evolucionan el sistema. Este caso no va a seguir una distribución de probabilidades, debido a esto no utiliza el principio de equiprobabilidad.

n_runs = $5,010^6$
Corrida 1
a = 620 b = 615 c = 653

- Si se espera la equiprobabilidad de los resultados, ya que si bien se dijo en el punto anterior que el método no utiliza el principio de equiprobabilidad los resultados si son equiprobables. La diferencia entre los resultados es muy pequeña.
- El tiempo totalde corrida al final de la simulación es de: 612112.1236616528

B1

la paradoja que da un histograma no constante para una distribución de probabilidad equiprobable se puede explicar observando la figura 1 obtenida al correr el programa. En donde gran parte de las coliciones suceden en el centro y por ende los discos tienden a dispersarse hacia los lados o extremos de la caja, por lo cual la probabilidad de encontrarlo en los extremos es más alta. Como nos dice el enunciado de la tarea 2 si pensamos como si fuera un gas ideal se observa mayor frecuencia en los bordes, ya que colisionan con los bordes, quiere decir que las partículas pasan dos veces por dichas coordenadas y la frecuencia aumenta.

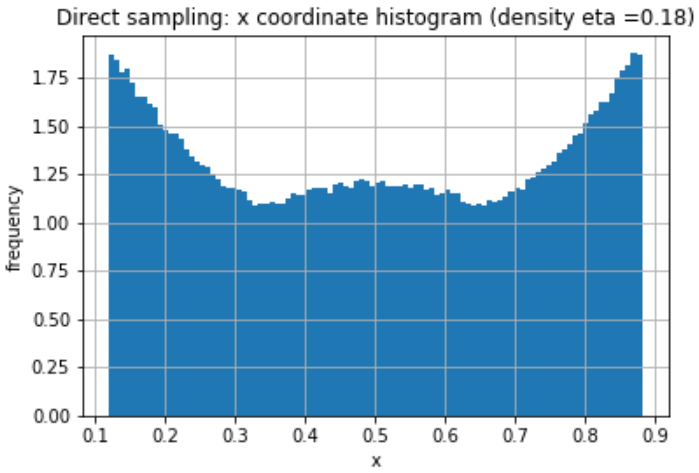

Figura 1: Histograma de la coordenada x para $\eta = 0{,}18$

B2

El código utilizado es:

```
import random, pylab

delta = 0.1

def markov_disks_box(N,sigma):
    a = random.choice(L)#RANDOM CHOICE PARA UN DISCO
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta, delta)]#se modifica su p
    min_dist = min((b[0] - c[0]) ** 2 + (b[1] - c[1]) ** 2 for c in L if c != a)
    box_cond = min(b[0], b[1]) < sigma or max(b[0], b[1]) > 1.0 - sigma
    if not (box_cond or min_dist < 4.0 * sigma ** 2): #si hay overlaps se queda en a, si no pasa a b
        a[:] = b

    return L

L = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
N = 4
sigma = 0.1197
n_runs = 2000000
histo_data = []
for run in range(n_runs):
    pos = markov_disks_box(N, sigma)
    for k in range (N):
        histo_data.append(pos[k][0])

pylab.hist(histo_data, bins = 100, normed = True)
pylab.xlabel('x')
pylab.ylabel ('frequency')
pylab.title('Histograma para el algoritmo "Markov_diks_box" (density eta =0.18)')
pylab.grid()
pylab.savefig('markov_disks_histo.png')
pylab.show()
```

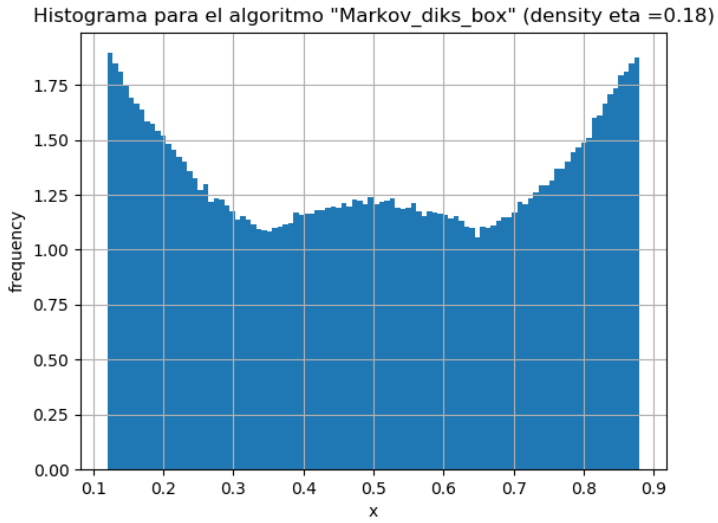

Figura 2: Histograma del algoritmo "$Markov\_disks\_box$" para $\eta = 0{,}18$

B3

Código utilizado en esta sección es:

```
import math

def wall_time(pos_a, vel_a, sigma):
    if vel_a > 0.0:
        del_t = (1.0 - sigma - pos_a) / vel_a
    elif vel_a < 0.0:
        del_t = (pos_a - sigma) / abs(vel_a)
    else:
        del_t = float('inf')
    return del_t

def pair_time(pos_a, vel_a, pos_b, vel_b, sigma):
    del_x = [pos_b[0] - pos_a[0], pos_b[1] - pos_a[1]]
    del_x_sq = del_x[0] ** 2 + del_x[1] ** 2
    del_v = [vel_b[0] - vel_a[0], vel_b[1] - vel_a[1]]
    del_v_sq = del_v[0] ** 2 + del_v[1] ** 2
    scal = del_v[0] * del_x[0] + del_v[1] * del_x[1]
    Upsilon = scal ** 2 - del_v_sq * ( del_x_sq - 4.0 * sigma **2)
    if Upsilon > 0.0 and scal < 0.0:
        del_t = - (scal + math.sqrt(Upsilon)) / del_v_sq
    else:
        del_t = float('inf')
    return del_t
conf_a = ((0.30, 0.30), (0.30, 0.70), (0.70, 0.30), (0.70, 0.70))
conf_b = ((0.20, 0.20), (0.20, 0.80), (0.75, 0.25), (0.75, 0.75))
conf_c = ((0.30, 0.20), (0.30, 0.80), (0.70, 0.20), (0.70, 0.70))
configurations = [conf_a, conf_b, conf_c]
hits = {conf_a: 0, conf_b: 0, conf_c: 0}
```

```
del_xy =0.10
pos = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
vel = [[0.21, 0.12], [0.71, 0.18], [-0.23, -0.79], [0.78, 0.1177]]
singles = [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1), (3, 0), (3, 1)]
pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
sigma = 0.1197
t = 0.0
n_events = 5000000
histo_data = []
for event in range(n_events):
    if event %100000 == 0:
        print (event)
    wall_times = [wall_time(pos[k][l], vel[k][l], sigma) for k, l  in singles]
    pair_times = [pair_time(pos[k], vel[k], pos[l], vel[l], sigma) for k, l in pairs]
    next_event = min(wall_times + pair_times)
    t_previous = t
    for inter_times in range (int(t + 1), int(t + next_event + 1)):
        del_t = inter_times - t_previous

        for k, l in singles:
            pos[k][l] += vel[k][l] * del_t
        t_previous = inter_times
        for conf in configurations:
            condition_hit = True
            for b in conf:
                condition_b = min(max(abs(a[0]-b[0]),abs(a[1]-b[1]))for a in pos) < del_xy
                condition_hit *= condition_b
            histo_data.append(pos[k][0])

    t += next_event
    del_t = t - t_previous
    for k, l in singles:
        pos[k][l] += vel[k][l] * del_t
    if min(wall_times) < min(pair_times):
        collision_disk, direction = singles[wall_times.index(next_event)]
        vel[collision_disk][direction] *= -1.0
    else:
        a, b = pairs[pair_times.index(next_event)]
        del_x = [pos[b][0] - pos[a][0], pos[b][1] - pos[a][1]]
        abs_x = math.sqrt(del_x[0] ** 2 + del_x[1] ** 2)
        e_perp = [c / abs_x for c in del_x]
        del_v = [vel[b][0] - vel[a][0], vel[b][1] - vel[a][1]]
        scal = del_v[0] * e_perp[0] + del_v[1] * e_perp[1]
        for k in range(2):
            vel[a][k] += e_perp[k] * scal
            vel[b][k] -= e_perp[k] * scal

pylab.hist(histo_data, bins = 100, normed = True)
pylab.xlabel('x')
pylab.ylabel ('frequency')
pylab.title('Direct sampling: x coordinate histogram (density eta =0.18)')
pylab.grid()
pylab.savefig('event_disks_histo.png')
pylab.show()
```

Se puede notar que lo único que se cambió en el código fue el condition_hit por el histo_data.append.

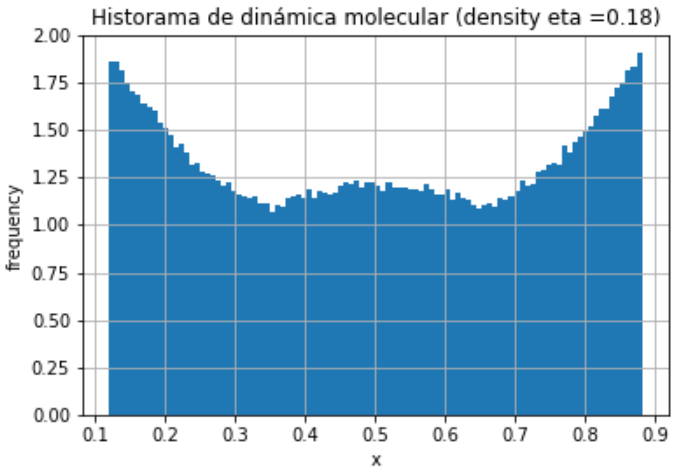

Figura 3: Histograma de danámica molecular para $\eta = 0{,}18$

Se puede notar que los tres histogramas presentados en la tarea son bastante similares, lo cual concuerda con lo esperado porque fisicamente está sucediendo lo mismo, ya que las posiciones de los discos tienden a ser más probables y frecuentes en los extremos de la caja, también se puede notar que la frecuencia en los tres es constante en el centro.