Miniproyecto # 1. Simulación 2D de un sistema de discos rígidos

Johans Restrepo Cárdenas

Instituto de Física. Universidad de Antioquia.

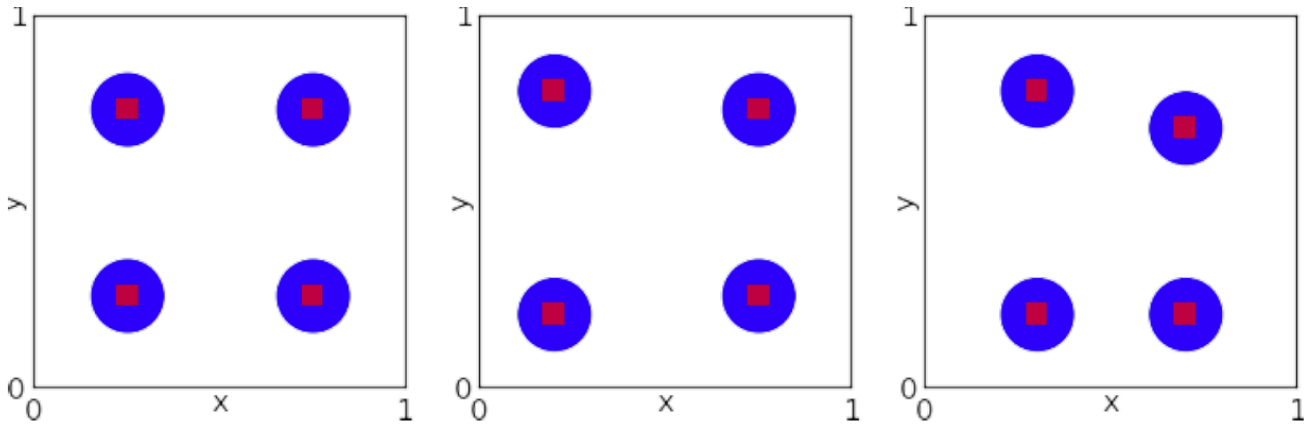
23 de marzo de 2021

Equiprobabilidad - Muestreo Directo

En el principio de equiprobabilidad todas las configuraciones legales y posibles en un sistema aislado son igualmente probables. En esta primera parte debe determinar si dicho principio se cumple para un sistema 2D de bolas rígidas ($N = 4$) que se mueven en una caja 2D de lado unidad. Para ello considere solo las tres siguientes configuraciones o microestados (a, b, c) de todas las configuraciones posibles y que están definidas por las coordenadas (x, y) de los centros de masa de los discos:

```
a = ((0.30, 0.30), (0.30, 0.70), (0.70, 0.30), (0.70,0.70))
b = ((0.20, 0.20), (0.20, 0.80), (0.75, 0.25), (0.75,0.75))
c = ((0.30, 0.20), (0.30, 0.80), (0.70, 0.20), (0.70,0.70))
```

La identificación computacional de configuraciones necesita el uso de pequeñas cajas como las mostradas en rojo en la siguiente figura. Por qué?



Equiprobabilidad - Muestreo Directo

Muestre que dentro de cierta precisión numérica dichas configuraciones (a, b, c) son igualmente probables utilizando el siguiente programa:

```python
import random, math
def direct_disks_box(N, sigma):
    condition = False
    while condition == False:
        L = [(random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))]
        for k in range(1, N):
            a = (random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))
            min_dist = min(math.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) for b in L)
            if min_dist < 2.0 * sigma:
                condition = False
                break
            else:
                L.append(a)
                condition = True
    return L


sigma = 0.15
del_xy = 0.05
n_runs = 10000
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}
#print direct_disks_box(4, sigma)
for run in range(n_runs):
    x_vec = direct_disks_box(4, 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

for conf in configurations:
    print conf, hits[conf]
```

Equiprobabilidad - Muestreo Directo

1) Estudie el programa y responda las siguientes preguntas:

1. Cuál es el significado de **sigma** y de la definición **direct_disks_box**?
2. Bajo qué condiciones la variable lógica **condition_hit** es verdadera.
3. A qué se refiere la **condicion_b** que aparece en la línea 32 y el condicional que aparece en las líneas 34 y 35.

2) Corra el programa para las siguientes situaciones:

1. Tres veces para **n_runs**$= 10^4$
2. Tres veces para **n_runs**$= 10^5$
3. Tres veces para **n_runs**$= 10^6$

Escriba en cada caso el número de **hits** para cada configuración a, b, c cada vez. Qué puede decir acerca de la *equiprobabilidad*? De qué manera estimaría las probabilidades acosciadas a cada configuración?

3) Corra su programa de nuevo para **sigma**=0.15 y para **del_xy**=0.10 usando diferentes valores de **n_runs**. Explíque qué sucede. Qué pasaría si los tamaños de las cajas rojas son demasiado grandes?

Equiprobabilidad - Muestreo basado en Markov

4) En general explique cómo funciona el programa dado y de qué manera está implementada la equiprobabilidad?
5) Estudie y modifique ahora el siguiente programa, el cual hace uso del concepto de muestreo basado en cadenas de Markov - **Markov-chain sampling**- (en qué consisten?), para chequear la equiprobabilidad de las configuraciones a, b, c dadas anteriormente. Para ello incorpore y modifique en este nuevo programa líneas del programa anterior.

```
1  import random
2
3  L = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
4  sigma = 0.15
5  sigma_sq = sigma ** 2
6  delta = 0.1
7  n_steps = 1000
8  for steps in range(n_steps):
9      a = random.choice(L)
10     b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta, delta)]
11     min_dist = min((b[0] - c[0]) ** 2 + (b[1] - c[1]) ** 2 for c in L if c != a)
12     box_cond = min(b[0], b[1]) < sigma or max(b[0], b[1]) > 1.0 - sigma
13     if not (box_cond or min_dist < 4.0 * sigma ** 2):
14         a[:] = b
15 print L
16
```

Equiprobabilidad - Muestreo basado en Markov

6) Estudie el programa de la figura anterior y explique las líneas 9-14.
7) Una vez modificado dicho nuevo programa con base en lo solicitado en 5), córralo para las siguientes situaciones:

1. Tres veces para **n_runs**$= 10^4$
2. Tres veces para **n_runs**$= 10^5$
3. Tres veces para **n_runs**$= 10^6$
4. Tres veces para **n_runs**$= 10^7 (opcional)$

Escriba en cada caso el número de **hits** para cada configuración a, b, c cada vez. Qué puede decir acerca de la *equiprobabilidad* en este caso? De qué manera estimaría las probabilidades asociadas a cada configuración? Compare sus resultados con los obtenidos inicialmente mediante el método de muestreo directo. Qué puede decir acerca de los fluctuaciones de los resultados.

Equiprobabilidad - Muestreo basado en eventos

7) Incorpore ahora las siguientes dos definiciones de eventos de colisión en el programa de la página siguiente:

```
3   def wall_time(pos_a, vel_a, sigma):
4       if vel_a > 0.0:
5           del_t = (1.0 - sigma - pos_a) / vel_a
6       elif vel_a < 0.0:
7           del_t = (pos_a - sigma) / abs(vel_a)
8       else:
9           del_t = float('inf')
10      return del_t
11
12  def pair_time(pos_a, vel_a, pos_b, vel_b, sigma):
13      del_x = [pos_b[0] - pos_a[0], pos_b[1] - pos_a[1]]
14      del_x_sq = del_x[0] ** 2 + del_x[1] ** 2
15      del_v = [vel_b[0] - vel_a[0], vel_b[1] - vel_a[1]]
16      del_v_sq = del_v[0] ** 2 + del_v[1] ** 2
17      scal = del_v[0] * del_x[0] + del_v[1] * del_x[1]
18      Upsilon = scal ** 2 - del_v_sq * ( del_x_sq - 4.0 * sigma **2)
19      if Upsilon > 0.0 and scal < 0.0:
20          del_t = - (scal + math.sqrt(Upsilon)) / del_v_sq
21      else:
22          del_t = float('inf')
23      return del_t
24
```

Dicho programa realiza un muestreo basados en eventos (de colisión) usando dinámica molecular. Investique en qué consiste la modelación usando dicha dinámica y en particular acerca del algoritmo *event-driven Molecular dynamics* por Alder and Wainwright (1957). Qué es un evento? Explique cada línea de código.

Equiprobabilidad - Muestreo basado en eventos

```
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 = 0.10
t = 0.0
n_events = 5000000
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
        print event, t
        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
            if condition_hit:
                hits[conf] += 1
    t += next_event
    del_t = t - t_previous
#    print t, del_t
    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

for conf in configurations:
    print conf, hits[conf]
```

Dicho programa realiza un muestreo basados en eventos (de colisión) usando dinámica molecular.

Equiprobabilidad - Muestreo basado en eventos

8) Investique en qué consiste la modelación usando dicha dinámica y en particular acerca del algoritmo *event-driven Molecular dynamics* por Alder and Wainwright (1957). Qué es un evento?

9) Note que dicho programa arroja como salida las posiciones asociadas a configuraciones de eventos. Son las posiciones asociadas a eventos igualmente probables? Justifique su respuesta.

Equiprobabilidad - Histograma de posiciones

Una manera de cambiar la densidad en el sistema es cambiando el radio de los discos manteniendo $N = 4$. Considere una densidad del 18 % lo que equivale a usar discos de radio **sigma**=0.1197. En lugar de configuraciones, considere ahora un observable:la coordenada x del centro del disco, para obtener un histograma de las posiciones x usando el siguiente programa con muestreo directo:

```python
import random, pylab

def direct_disks_box(N, sigma):
    overlap = True
    while overlap == True:
        L = [(random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))]
        for k in range(1, N):
            a = (random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))
            min_dist_sq = min(((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) for b in L)
            if min_dist_sq < 4.0 * sigma ** 2:
                overlap = True
                break
            else:
                overlap = False
                L.append(a)
    return L


N = 4
sigma = 0.1197
n_runs = 1000000
histo_data = []
for run in range(n_runs):
    pos = direct_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('Direct sampling: x coordinate histogram (density eta=0.18)')
pylab.grid()
pylab.savefig('direct_disks_histo.png')
pylab.show()
```

Explique por qué no se obtiene un histograma constante o plano que es lo que en principio se esperaría para una distribución equiprobable, y qué sería necesario implementar para corregir dicho problema.

Equiprobabilidad - Histograma de posiciones

Haga lo mismo para las dinámicas de muestreo basadas en cadenas de Markov (con corridas largas, al menos **n_steps**=2000000) y en eventos modificando cuidadosamente los programas correspondientes. Compare los tres histogramas y discuta sus resultados.

Animación

Finalmente estudie el siguiente programa para hacer una película o animación que permita observar cómo se mueven las partículas en la caja cuando se usa dinámica molecular basada en eventos. Para ello básese en el siguiente programa:

- *1/2*:

```python
import os, math, pylab

output_dir = "event_disks_box_movie"
colors = ['r', 'b', 'g', 'orange']

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

def min_arg(l):
    return min(zip(l, range(len(l))))

def compute_next_event(pos, vel):
    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]
    return min_arg(wall_times + pair_times)

def compute_new_velocities(pos, vel, next_event_arg):
    if next_event_arg < len(singles):
        collision_disk, direction = singles[next_event_arg]
        vel[collision_disk][direction] *= -1.0
    else:
        a, b = pairs[next_event_arg - len(singles)]
        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

```

- *2/2*:

```python

pylab.subplots_adjust(left=0.10, right=0.90, top=0.90, bottom=0.10)
pylab.gcf().set_size_inches(6, 6)
img = 0
if not os.path.exists(output_dir): os.makedirs(output_dir)
def snapshot(t, pos, vel, colors, arrow_scale=.2):
    global img
    pylab.cla()
    pylab.axis([0, 1, 0, 1])
    pylab.setp(pylab.gca(), xticks=[0, 1], yticks=[0, 1])
    for (x, y), (dx, dy), c in zip(pos, vel, colors):
        dx *= arrow_scale
        dy *= arrow_scale
        circle = pylab.Circle((x, y), radius=sigma, fc=c)
        pylab.gca().add_patch(circle)
        pylab.arrow( x, y, dx, dy, fc="k", ec="k", head_width=0.05, head_length=0.05 )
    pylab.text(.5, 1.03, 't = %.2f' % t, ha='center')
    pylab.savefig(os.path.join(output_dir, '%04i.png' % img))
    img += 1

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 = 0.15
t = 0.0
dt = 0.02      # dt=0 corresponds to event-to-event animation
n_steps = 100
next_event, next_event_arg = compute_next_event(pos, vel)
snapshot(t, pos, vel, colors)
for step in range(n_steps):
    if dt:
        next_t = t + dt
    else:
        next_t = t + next_event
    while t + next_event <= next_t:
        t += next_event
        for k, l in singles: pos[k][l] += vel[k][l] * next_event
        compute_new_velocities(pos, vel, next_event_arg)
        next_event, next_event_arg = compute_next_event(pos, vel)
    remain_t = next_t - t
    for k, l in singles: pos[k][l] += vel[k][l] * remain_t
    t += remain_t
    next_event -= remain_t
    snapshot(t, pos, vel, colors)
    print 'time',t

print('Producing animation.gif using ImageMagick...')
os.system("convert -delay 1 -dispose Background +page " + str(output_dir)
          + "/*.png -loop 0 " + str(output_dir) + "/animation.gif")

```

Sobre el informe.

El informe en forma de artículo debe contener:

- Título
- Nombre autor, afiliación.
- Resumen y palabras claves.
- Introducción.
- Marco teórico (haga alusión a los aspectos teóricos del problema planteado: *direct sampling Monte Carlo*, *Markov chain Monte Carlo* y *dinámica molecular por eventos*.)
- Resultados y discusión (incluya las gráficas solicitadas).
- Conclusiones
- Bibliografía
- Agradecimientos

El informe en forma de artículo debe tener como anexo los códigos desarrollados con sus respectivos comentarios. Enviar archivo comprimido con todos los archivos adjuntos a johans.restrepo@udea.edu.co.