UNIVERSIDAD DE
COSTA RICA

Física Computacional
Escuela de Física

Dinámica molecular en dos dimensiones: discos sólidos

Autores
Sebastián Alejando Aguilar Rodríguez (C30143)
Isabel Cambronero Jiménez (C31543)
Isaac Gabriel Flores Ureña (C32982)
Jose Daniel Ruiz Zúñiga (C37081)

Profesor(es)
Marlon Brenes
Federico Muños

1. Explicación matemática

1.1. Conservación del momento

El momento lineal, $p = mv$, es una cantidad vectorial que se conserva en interacciones entre partículas en ausencia de fuerzas externas. Antes y después de la colisión, la suma de los momentos debe ser igual. Esta ecuación es válida para cualquier tipo de colisión: elástica o inelástica.

$$m_1 v_1 + m_2 v_2 = m_1 v_1' + m_2 v_2' \tag{1}$$

Esta ecuación describe el principio de conservación del momento lineal en un sistema aislado, donde no actúan fuerzas externas.

- m_1 y m_2 : masas de los objetos.
- v_1 y v_2 : velocidades iniciales de los objetos antes de la colisión.
- v_1' y v_2' : velocidades finales después de la colisión.

1.2. Conservación de la energía cinética

La conservación de la energía cinética, junto con la conservación del momento, permite calcular las velocidades finales en una colisión elástica. Si la colisión es inelástica, parte de la energía cinética inicial se transforma en otras formas de energía, como calor o deformaciones.

$$\frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 v_1'^2 + \frac{1}{2} m_2 v_2'^2 \tag{2}$$

Esta ecuación aplica únicamente a colisiones elásticas, en las que no hay pérdida de energía cinética en el sistema.

- $\frac{1}{2}mv^2$: es la energía cinética de un objeto - La suma de las energías cinéticas antes de la colisión es igual a la suma de las energías cinéticas después de la colisión.

1.3. Velocidades finales después de la colisión

$$v_1' = \frac{v_1(m_1 - m_2) + 2m_2 v_2}{m_1 + m_2} \tag{3}$$

$$v_2' = \frac{v_2(m_2 - m_1) + 2m_1 v_1}{m_1 + m_2} \tag{4}$$

Estas fórmulas son el resultado de resolver conjuntamente las ecuaciones de conservación del momento lineal y conservación de la energía cinética para una colisión elástica en una dimensión.

1. A partir de $m_1 v_1 + m_2 v_2 = m_1 v_1' + m_2 v_2'$ se obtiene una relación entre v_1' y v_2'.
2. Se combina con la ecuación de conservación de la energía cinética: $\frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 v_1'^2 + \frac{1}{2} m_2 v_2'^2$.
3. Se resuelve el sistema para v_1' y v_2'.

Estas expresiones muestran que las velocidades finales dependen tanto de las velocidades iniciales como de las masas de los objetos.

1.4. Ecuación para la posición

Esta ecuación proviene de la cinemática en una dimensión para un movimiento con velocidad constante:

$$x_f = x_i + v_x t \tag{5}$$

Es la ecuación fundamental para describir el movimiento rectilíneo uniforme (MRU). En este caso, la velocidad es constante, por lo que la posición cambia linealmente con el tiempo.

- x_f: posición final del objeto.
- x_i: posición inicial del objeto.
- v_x: velocidad en la dirección x (constante).
- t: tiempo transcurrido.

2. Clases

2.1. Disco

A continuación se muestran las clases que se implementaron para el desarrollo del programa. Para comenzar se muestra la clase `Disco`, la cual contiene todas las características necesarias para tratar cada partícula como si fuera un disco con forma circular. Esta clase toma como atributos en el constructor `elradio` que contiene la información del radio de la partícula, `lamasa` que contiene la información de la masa de cada partícula, `laposicionx` que representa la posición en el eje x de la partícula en un momento dado, `laposiciony` que representa la posición en el eje y de la partícula en un momento dado, `lavelocidadx` representa el componente del vector de la velocidad en el eje x, `lavelocidady` representa el componente de la velocidad en le eje y. A su vez, en el constructor se definen el color y dos arrays que almacenarán los cambios de posición de la partícula.

```
class Disco:
    """
    Representa un disco en el sistema que trabajaremos.

    Attributes:
    radio (float): el radio del disco
    masa (float): la masa del disco
    posicionx (float): la posición del disco en el eje X
    posiciony (float): la posición del disco en el eje Y
    velocidadx (float): la velocidad en el eje X
    velocidady (float): la velocidad en el eje Y
    color (list): el color del disco
    """
    def __init__(self, elradio, lamasa, laposicionx, laposiciony, lavelocidadx, lavelocidady):
        """
        Inicializa un disco con sus propiedades físicas.

        Args:
            elradio (float): radio del disco
```

```
            lamasa (float): masa del disco
            laposicionx (float): posicion inicial en el eje X
            laposiciony (float): posicion inicial en el eje Y
            lavelocidadx (float): velocidad inicial del disco en eje X
            lavelocidady (float): velocidad inicial del disco en eje Y
        """
        self.radio = elradio
        self.masa = lamasa
        self.posicionx = laposicionx
        self.posiciony = laposiciony
        self.velocidad = np.array([lavelocidadx,lavelocidady])
        self.color = [0,0,1]
        #posicion de los discos para crear el histograma
        self.arrayposicionx = np.array([])
        self.arrayposiciony = np.array([])
```

A continuación, se muestran las funciones miembro de la clase llamadas `right` , `left` , `top` y `bottom` ; las cuales tienen definen los limites de cada partícula por medio de realizar operaciones que involucran el radio y la posición de esta. Por último, esta la función `colores` , la cual asigna los colores a cada partícula de forma aleatoria.

```
  def right(self):
        return self.posicionx + self.radio

    #función miembro para obtener el límite izquierdo
    def left(self):
        return self.posicionx - self.radio

    #funcion miembro para obtener el límite superior
    def top(self):
        return self.posiciony + self.radio

    #funcion miembro para obtener el límite inferior
    def bottom(self):
        return self.posiciony - self.radio

    def colores(self):
      colors = [[1,0,0],[0,1,0],[0,0,1],[1,1,0],[1,0,1],[0,1,1],[0,0,0]]
      new_color_number = ri(0,len(colors)-1)
      new_color = colors[new_color_number]
      return new_color
```

2.2. Box

A continuación se muestra la clase `Box` la cual crea una caja que contendrá la grilla donde se encuentran las partículas. Toma como atributos el largo de la caja, `lalongitudX` ; y el ancho de esta, `lalongitudY` .

```
class Box:
    def __init__(self,lalongitudX,lalongitudY):
      self.longitudx = lalongitudX
      self.longitudy = lalongitudY
```

2.3. Grilla

Por último, a continuación se muestra la clase `Grilla`, la cual se encarga de general la grilla donde se localizan las partículas. Esta clase toma como atributos `xmax`, como el valor máximo de x; `ymax`, como el valor máximo de y; `radio` que toma el valor del radio más grande de todas las partículas. De igual forma se inicaliza `subdivisionesx` y `subdivisionesy` las cuales contiene la cantidad de subdivisiones de la grilla en cada eje. Además, hay una función miembro dentro del constructor llamada `creacion_grilla` que se encarga de generarl unos linspace para cada eje. Además las variables `dist_entre_separX` y `dist_entre_separY` contienen los valores que representan la diferencia que hay entre cada elemento del linspace.

```
class Grilla:
  def __init__(self,xmax,ymax,radio):
    self.xmax = xmax
    self.ymax = ymax
    self.subdivisionesx = int(xmax/(2*radio)) #Cantidad de subdivisiones en el eje x
    self.subdivisionesy = int(ymax/(2*radio)) #Cantidad de subdivisiones en el eje y

    def creacion_grilla(self,xmax,ldiscos):
      divisionX = np.linspace(0,xmax,self.subdivisionesx)
      divisionY = np.linspace(0,ymax,self.subdivisionesy)
      dist_entre_separX = divisionX[1]
      dist_entre_separY = divisionY[1]
      return divisionX, divisionY, dist_entre_separX, dist_entre_separY

    self.divisionX, self.divisionY, self.dist_entre_separX, self.dist_entre_separY =
    ↪  creacion_grilla(self,xmax,ymax)
```

3. Funciones

Función `colision_proxima`

Esta función se encarga de gestionar las colisiones entre discos. Los pasos que realiza son los siguientes:

- Verifica las posiciones de los discos en la grilla, considerando sus discos vecinos más cercanos en el eje x.
- Calcula el tiempo hasta la próxima colisión entre discos y con las paredes.
- Si se detecta una colisión, ajusta las velocidades de los discos involucrados.
- Si la colisión es con las paredes, se gestionan los cambios en la velocidad y la posición de los discos de acuerdo a la física de los impactos.
- El proceso se repite hasta que se resuelven todas las colisiones posibles.

Función `manejo_de_colisiones_pares`

Esta función maneja el cambio de velocidades de dos discos debido a una colisión. Sus pasos son los siguientes:

- Cambia las velocidades de los discos usando la función `cambio_velocidad`.
- Realiza una simulación de pasos temporales muy pequeños (Δt) para evitar que los discos sigan colisionando.
- Ajusta las posiciones de los discos hasta que la distancia entre ellos sea mayor que la suma de sus radios, asegurando que dejen de colisionar.
- Retorna las nuevas posiciones y velocidades de los discos después de la colisión.

Función `inicializacion_discos`

La función `inicializacion_discos` crea discos con características específicas. Los pasos son los siguientes:

- Crea una matriz de posiciones para distribuir los discos en una grilla.
- Si la inicialización es aleatoria (sin colores aleatorios), asigna una posición aleatoria a cada disco sin que se solapen en la matriz.
- Si se permite la aleatorización de colores, asigna un color aleatorio a cada disco.
- Si la inicialización no es aleatoria, organiza los discos de forma ordenada en la grilla, rellenando las filas de abajo hacia arriba.
- Los discos se colocan en posiciones específicas dentro de la grilla, respetando las restricciones de espacio entre ellos.

Función `graf_discos`

La función `graf_discos` genera una visualización de los discos en un gráfico 2D. Los pasos que sigue son:

- Para cada disco, dibuja un círculo con el radio correspondiente en la posición x e y especificadas.
- Configura los límites de los ejes x e y según el tamaño de la caja que contiene los discos.
- Añade los discos al gráfico y guarda cada fotograma generado como un archivo PNG en el directorio de trabajo.
- Después de cada fotograma, cierra la figura generada para evitar que se acumulen en memoria.

Función `histo_discos`

La función `histo_discos` crea un histograma de las posiciones de los discos en el eje x. Los pasos son:

- Divide el rango de posiciones en el eje x en un número especificado de subdivisiones.
- Cuenta cuántos discos caen en cada subdivisión del eje x.
- Muestra el histograma con las posiciones de los discos y guarda la imagen generada en formato PNG.

Función `crear_video`

La función `crear_video` genera un video a partir de los fotogramas generados. Los pasos son:

- Toma todos los fotogramas guardados como archivos PNG y los organiza en el orden correcto.
- Combina los fotogramas en un video utilizando la librería `moviepy`, especificando los fotogramas por segundo (fps).
- Guarda el video en formato MP4 en el directorio de trabajo.
- Elimina todos los fotogramas PNG después de generar el video para liberar espacio.

4. Main

A continuación se realizará una muestra de como implementrar las clases y funciones previamente expuestas. Para ello se van a realizar dos muestras de la implementación, una con pocos discos pero mostrando un video de como se comportan, se omitirá la salida de código de esta muestra debido a que no es posible agregar dicho video a este reporte. Por otro lado, se realizara otra muestra con una mayor cantidad de discos para generara un histograma de la posición de estos.

Para simplificar la explicación se va a mostrar todas las librerías que se deben importar para los dos casos juntas, las cuales son las que se muestran a continuación:

```
#!/usr/bin/env python3
from clases import Disco, Box, Grilla
from funciones import *
import numpy as np
import matplotlib.pyplot as plt
from random import uniform as uf
from moviepy.editor import *
import glob
import os
```

4.1. Gráfico de discos

A continuación se muestran los datos necesarios para inicializar los discos y las dimensiones de la caja y el tiempo que dura el video.

```
#dimensiones de la caja
ladohorizontal = 1
ladovertical = 1

#parametros de los discos
numero_discos = 6
masa = 1
radio = 0.1
velomax = 0.1

#tiempo de simulacion en segundos
tmax = 60
```

Después se crean los elementos del sistema, como los discos, la caja y la grilla

```
############################### Creacion de los elementos del sistema
↪  ###################################
discos = []

caja = Box(ladohorizontal, ladovertical)

grilla = Grilla(caja.longitudx,caja.longitudy,radio)

discos = inicializacion_discos(radio, masa, -velomax, velomax, grilla, numero_discos, True, True)
```

Por último se define la cantidad de fotogramas y se hacen dos ciclos for, uno para que haga la rutina en cada fotograma, y otro para que la realice en cada disco. Por último se crea un video.

```
######################################### Evolucion del sistema
↪  ##########################################

FPS = 60
newt = 1/FPS

fotograma = 0

for n in range(0,tmax*FPS):
    #actualizacion de las posiciones de los discos
    for i in range(numero_discos):
        discos[i] = nueva_posicion(discos[i],newt)

    #Verificacion de colisiones
    discos = colision_proxima(grilla, discos,cambio_velocidad_colision_pares, newt,
    ↪  manejo_de_colisiones_pares, tiempo_colision_pared,deteccion_colision_pared_con_manejo,
    ↪  caja.longitudx,caja.longitudy)

    graf_discos(discos,caja,fotograma,grilla)
    fotograma += 1

crear_video(FPS)
```

4.2. Creación de histograma

A continuación se muestran los datos necesarios para inicializar los discos y las dimensiones de la caja y el tiempo en el que se realiza el histograma.

```
#dimensiones de la caja
ladohorizontal = 1
ladovertical = 1

#parametros de los discos
numero_discos = 200
masa = 1
radio = 0.01
velomax = 0.1

#tiempo de simulacion en segundos
tmax = 60
```

Después se crean los elementos del sistema, como los discos, la caja y la grilla

```
################################ Creacion de los elementos del sistema
↪  ##################################
discos = []

caja = Box(ladohorizontal, ladovertical)

grilla = Grilla(caja.longitudx,caja.longitudy,radio)

discos = inicializacion_discos(radio, masa, -velomax, velomax, grilla, numero_discos, True, True)
```

Por último se define la cantidad de fotogramas y se hacen dos ciclos for, uno para que haga la rutina en cada fotograma, y otro para que la realice en cada disco. Por para concluir generando un histograma de las posiciones de las partículas.

```
######################################## Evolucion del sistema
↪  ########################################

FPS = 60
newt = 1/FPS

fotograma = 0

for n in range(0,tmax*FPS):
    #actualizacion de las posiciones de los discos
    for i in range(numero_discos):
        discos[i] = nueva_posicion(discos[i],newt)

    #Verificacion de colisiones
    discos =
    ↪  colision_proxima(grilla,discos,cambio_velocidad_colision_pares,newt,manejo_de_colisiones_pares,tiempo_colision_p

#Creacion del histograma
histo_discos(discos,caja.longitudx,50)
crear_video(FPS)
```

A continuación se adjunta una muestra del histograma con 200 partículas

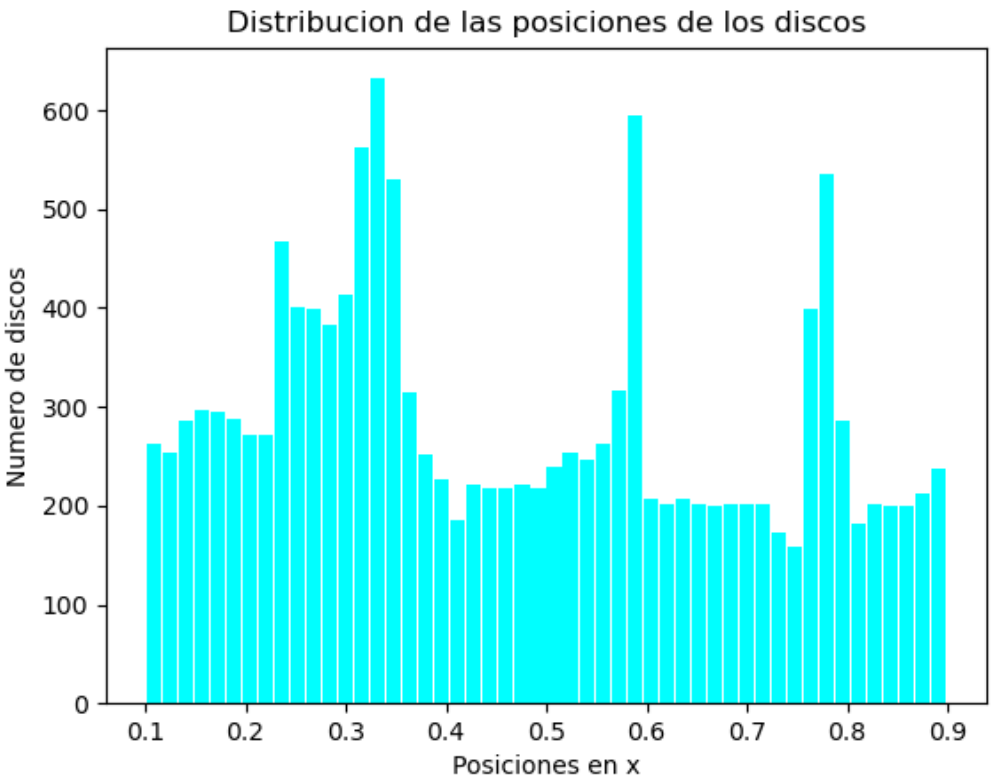

Figura 1: Histograma con 200 partículas

Referencias

[1] Reducible. Building Collision Simulations: An Introduction to Computer Graphics. You Tube; 2021.

[2] Berchek C. 2-Dimensional Elastic Collisions without Trigonometry. Vobarion; 2009.