UNIVERSIDAD DE GRANADA

Facultad de Ciencias

Física Computacional

Voluntario 1: Simulación con dinámica molecular de un gas con un potencial de Lennard-Jones

Resumen

En este informe tenemos como objetivo ...

Zhuo Zhuo Liu

Grado en Física

Índice

1. Introducción

2. Planteamiento del problema

Para poder

2.1. Condiciones iniciales

Al introducir las condiciones iniciales del sistema, debemos de tener cuidado de no colocar 2 partículas muy cercas entre ellas inicialmente, ya que puede provocar que las partículas adquieran mucha velocidad.

Para ello consideramos una cuadrícula separada por una distancia $L/5$ en ambos ejes, y permitimos que las partículas se desplace una distancia aleatoria entre 0 y 1 de dicha posición. Y una velocidad con dirección aleatoria, pero con módulo unidad.

```
@jit(nopython=True)
def init_cond():
    r = np.zeros((N, 2))
    v =np.zeros((N, 2))
    for i in range(N):
        r[i] = np.array([i%(L/2)*2+1, i%4*2 +1 ]) + np.random.rand(1, 2)[0]
        theta = np.random.rand()*2*np.pi
        v[i] = v_0 * np.array([np.sin(theta), np.cos(theta)])
    return r, v
```

2.2. Condiciones de contorno

Para introducir la condición de contorno bidimensional periódica empleamos 2 funciones, una para la posición de las partículas, y la otra para la distancia entre partículas. Pero ambos seguirán la misma lógica.

Comenzando para la posición de las partículas, se tiene que si una partícula en su nueva posición tiene una coordenada mayor que L o menor que 0, entonces se le restará o sumará L a dicha coordenada dependiendo del caso.

```
@jit(nopython=True)
def cond_contorno(r):
    if(r[0] > L):
        r[0] = r[0] - L
    if(r[0] < 0):
        r[0] = r[0] + L
    if(r[1] > L):
        r[1] = r[1] - L
    if(r[1] < 0):
```

```
        r[1] = r[1] + L
    return r
```

Mientras que para la distancia entre partículas, se tiene que si la distancia entre partículas en una de las coordenadas es mayor que $L/2$, entonces se le restará L a la distancia. Y aplicando lo mismo al caso contrario, cuando la distancia es menor que $-L/2$.

```
@jit(nopython=True)
def cond_contorno_distancia(r):
    if(r[0] > L/2):
        r[0] = r[0] - L
    elif(r[0] < -L/2):
        r[0] = r[0] + L
    if(r[1] > L/2):
        r[1] = r[1] - L
    elif(r[1] < -L/2):
        r[1] = r[1] + L
    return r
```

Notar que se ha tenido que diferenciar en 2 funciones muy similares, pero con diferentes condiciones, ya que el sistema no está centrado en 0. Si se hubiera centrado en 0, ambas funciones serían iguales.

Una vez tenida estas condiciones, ya podemos calcular la distancia entre partículas, estas distancias será importantes tanto para calcular las fuerzas de interacción, como la energía potencial del sistema.

Guardaremos estas distancias o mejor dicho los vectores que las unen en una matriz de dimensiones $N \times N \times 2$, donde N es el número de partículas, y 2 indica las dimensiones del espacio. El cálculo de estas distancias es sencillo, siendo la resta entre las posiciones de las partículas.

Podemos reducir el número de cálculos al notar que la matriz es antisimétrica, entonces el elemento R_{ij} es igual a $-R_{ji}$, por lo que solo necesitamos calcular menos de la mitad de la matriz, y asignar el valor correspondiente a los otros elementos.

```
@jit(nopython=True)
def compute_distance(r):
    R = np.zeros((N, N, 2))
    for i in range(0, N-1):
        for j in range(i+1, N):
            R[i, j] = r[j]- r[i]
            R[i, j] = cond_contorno_distancia(R[i, j])
            R[j, i] = -R[i, j]
    return R
```

2.3. Potencial Lennard-Jones

Una vez tenido las funciones para imponer la condición de contorno y el cálculo de las distancias entre partículas, podemos calcular la fuerza de interacción entre partículas por el potencial de Lennard-Jones.

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{R}\right)^{12} - \left(\frac{\sigma}{R}\right)^{12} \right] \tag{1}$$

donde se ha usado R en lugar de r, para coincidir en la notación empleada en el código.

Entonces la fuerza de interacción entre las partículas viene dado por:

$$\vec{F}(\vec{R}) = -4\epsilon \left[6\left(\frac{\sigma}{R}\right)^{5} - 12\left(\frac{\sigma}{R}\right)^{11} \right] \tag{2}$$

Para calcular la aceleración de la partícula, se suma la fuerza de interacción con todas las demás las partículas, y se divide por la masa.

Implementando en todo esto en la función que te devuelve la aceleración del sistema en función de la posición de las partículas.

```
@jit(nopython=True)
def lennard_jones(r):
    R = compute_distance(r)
    acc = np.zeros((N, 2))
    for i in range(N):
        for j in range(N):
            if(i!=j):
                norm  = np.linalg.norm(R[i, j])
                if (norm < 3):
                    acc[i] = acc[i] + 4*R[i, j]* epsilon *
                        *(6*np.power((sigma/norm), 5) -
                        - 12*np.power((sigma/norm), 11))/(norm*m)
    return acc, R
```

2.4. Algoritmo de Verlet

Por último solo queda añadir una función que nos permita calcular la nueva posición de las partículas, y la nueva velocidad de las partículas. Para ello emplearemos el algoritmo de Verlet.

A. Tabla de valores

B. Análisis de errores