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

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.

Empecemos por la posición, para ello al introducir el vector posición debemos de comprobar que en caso de tener alguna coordenada mayor que L, imponer la periodicidad, esto lo haremos usando el operador resto. En Python definiríamos la siguiente función:

```
def cond_contorno(r):
    return r%L
```

donde r es un array de NumPy de dimensiones Nx2 donde se almacena la posición de todas las N partículas.

Por otro lado para la distancia entre partículas, emplearemos la siguiente lógica.

1. Calculamos el vector $\vec{R}_{ij}$ que une 2 puntos (r_j, r_i). Dicho vector formará un ángulo θ con el eje x.
2. A partir de dicho ángulo podemos calcular la distancia H (véase figura), si el módulo de $\vec{R}_{ij}$ supera la distancia H, sabemos que no estamos calculando la distancia más corta entre los dos puntos. s
3. En dicho caso redefinimos el vector $\vec{R}_{ij}$, como:

$$\vec{R}_{ij} = -\vec{R}_{ij}\frac{H - |\vec{R}_{ij}|}{|\vec{R}_{ij}|} \tag{1}$$

Implementando en función tendría la siguiente forma:

```
def compute_distance(r):
    R = np.zeros((N, N, 2))
    for i in range(1, N):
        for j in range(i, N):
```

```
            R[i, j] = r[j]- r[i]
            angle = np.arctan(R[i, j, 1]/R[i, j, 0])
            max_length = abs(L/np.cos(angle%(np.pi/4)))
            norm = np.linalg.norm(R[i, j])
            if(norm > max_length):
                R[i, j] =  -R[i, j] *(max_length - norm) / norm
            R[j, i] = R[i, j]
    return R
```

2.3. Potencial Lennard-Jones

Una vez tenido las funciones para imponer la condición de contorno. Podemos calcular la fuerza ejercida entre partículas por el potencial de Lennard-Jones.

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{R}\right)^{12} - \left(\frac{\sigma}{R}\right)^{12} \right] \tag{2}$$

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:

$$\vec{F}(\vec{R}) = -4\epsilon \left[6\left(\frac{\sigma}{R}\right)^{5} - 12\left(\frac{\sigma}{R}\right)^{11} \right] \tag{3}$$

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.

```
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])
                acc[i] = 4*epsilon(6*(sigma/norm)**5-
                    -12*(sigma/norm)**12)*R[i, j]/(norm*m)
    return acc
```

A. Tabla de valores

B. Análisis de errores