Simulación con Dinámica Molecular de un Gas con Potencial de Lennard-Jones

Jesús de la Oliva Iglesias

4 Junio 2024

1 Introducción

La simulación de un gas mediante dinámica molecular utilizando el potencial de Lennard-Jones es una técnica fundamental para estudiar las propiedades de sistemas físicos a nivel microscópico. Utilizaremos el algoritmo de Verlet para resolver las ecuaciones de movimiento de las partículas en el sistema, considerando un gas bidimensional con condiciones de contorno periódicas. Para la realización de esta simulación se ha realizado un código en python usando las librerias numpy y numba, para agilizar los cálculos.

2 Código de Simulación de Lennard Jones

A continuación se muestra un fragmento de código para la simulación del sistema de partículas utilizando unidades del sistema internacional:

Listing 1: Simulación de Lennard Jones en Python

```
import numpy as np
import time
from numba import jit

# Establecemos el tiempo inicial.
t0 = time.time()

# N mero de simulaciones.
simulaciones = 4

# Establecemos los incrementos del tiempo.
```

```
h = 0.001

# N mero de iteraciones.
iteraciones = 30000

# N mero de iteraciones que se saltan
#para guardar los datos.
skip = 1

# Valor sigma
sigma = 3.4

# Pedimos el n mero de part culas.
n = 40

# Tama o de caja
l = 10

# Interespaciado entre las part culas.
s = 1

# Variable para saber si las part culas
#se encuentran en un panal.
panal = True

# Reescalamiento de velocidades en
#tiempos espec ficos.
REESCALAMIENTO = False

# Temperatura cr tica.
Temperatura_critica = False

if Temperatura_critica == True:
    REESCALAMIENTO = False
```

2.1 Explicación del Código

Este código configura una simulación del sistema utilizando unidades del sistema internacional. A continuación se detalla cada sección del código:

- **Importación de bibliotecas**:

```
import numpy as np
import time
from numba import jit
```

Se importan las bibliotecas necesarias: ‘numpy‘ para operaciones matemáticas, ‘time‘ para medir el tiempo de ejecución, y ‘numba‘ para optimizar el rendimiento.

- **Establecimiento del tiempo inicial**:

```
t0 = time.time()
```

Se almacena el tiempo inicial para medir el tiempo de ejecución total de la simulación.

- **Parámetros de la simulación**:

```
# N mero de simulaciones.
simulaciones = 4

# Establecemos los incrementos del tiempo.
h = 0.001

# N mero de iteraciones.
iteraciones = 30000

# N mero de iteraciones que se saltan para
#guardar los datos.
skip = 1
```

Se definen los parámetros de la simulación: número de simulaciones (‘simulaciones‘), incremento del tiempo (‘h‘), número de iteraciones (‘iteraciones‘), y número de iteraciones que se saltan para guardar datos (‘skip‘).

- **Configuración del sistema**:

```
# Valor sigma
sigma = 3.4

# Pedimos el n mero de part culas.
n = 40
```

```
# Tama o de caja
l = 10

# Interespaciado entre las part culas.
s = 1
```

Se configuran los parámetros del sistema: valor de sigma (‘sigma‘), número de partículas (‘n‘), tamaño de la caja (‘l‘), y espaciamiento entre partículas (‘s‘).

- **Condiciones iniciales**:

```
# Variable para saber si las part culas se
#encuentran en un panal.
panal = True

# Reescalamiento de velocidades en tiempos
#espec ficos.
REESCALAMIENTO = False

# Temperatura cr tica.
Temperatura_critica = False

if Temperatura_critica == True:
    REESCALAMIENTO = False
```

Se establecen las condiciones iniciales: si las partículas están en un panal (‘panal‘), si se realizará un reescalamiento de velocidades (‘REESCALAMIENTO‘), y si se considerará la temperatura crítica (‘Temperatura critica‘). Si la temperatura crítica está activada, se desactiva el reescalamiento.

3 Algoritmo de Verlet

El algoritmo de Verlet es un método numérico utilizado para integrar las ecuaciones de movimiento en simulaciones de dinámica molecular. La posición $\mathbf{r}(t)$ y la velocidad $\mathbf{v}(t)$ de cada partícula se actualizan en cada paso temporal Δt mediante las siguientes ecuaciones:

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2, \tag{1}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{1}{2}[\mathbf{a}(t) + \mathbf{a}(t+\Delta t)]\Delta t, \tag{2}$$

donde $\mathbf{a}(t)$ es la aceleración de la partícula en el tiempo t, que se obtiene a partir de las fuerzas calculadas usando el potencial de Lennard-Jones.

- **La implementación para el cálculo del algritmo de Verlet**:

```
#Realizamos el algoritmo de Verlet.
w_i = w_ih(n, velocidades, a_i, w_i, h)
posiciones = p_th(n, posiciones, w_i, h)
posiciones, momento = contorno(posiciones, l,
                               n, velocidades)
a_i, E_p = acel_i_th(n, posiciones, a_i,
                     l, E_p_c, a_c)
velocidades, E_c = velocidad_th(w_i, n,
                     velocidades, a_i, h)
```

4 Definición de funciones clave para la simulación

4.1 Función $w_i(h)$

La función `w_ih` calcula el término intermedio w_i que es parte del algoritmo de integración de Verlet. Esta función utiliza la biblioteca Numba para la compilación Just-In-Time (JIT) y la optimización de matemáticas rápidas.

Listing 2: Función $w_i(h)$

```
@jit(nopython=True, fastmath=True)
def w_ih(n, velocidades, a_i, w_i, h):
    for i in range(n):
        w_i[i] = velocidades[i] + a_i[i]*(h/2)
    return w_i
```

La función `w_ih` toma los siguientes parámetros:

- `n`: Número de partículas.
- `velocidades`: Arreglo de velocidades de las partículas.

- `a_i`: Arreglo de aceleraciones de las partículas.
- `w_i`: Arreglo donde se almacenarán los resultados intermedios.
- `h`: Incremento de tiempo.

La función actualiza w_i para cada partícula i usando la fórmula:

$$w_i[i] = velocidades[i] + a_i[i] \cdot \left(\frac{h}{2}\right)$$

4.2 Función $r(t + h)$

La función `p_th` calcula las nuevas posiciones de las partículas en el tiempo $t + h$.

Listing 3: Función $r(t + h)$

```
@jit(nopython=True, fastmath=True)
def p_th(n, posiciones, w_i, h):
    for i in range(n):
        posiciones[i] = posiciones[i] + w_i[i]*h
    return posiciones
```

La función `p_th` toma los siguientes parámetros:

- `n`: Número de partículas.
- `posiciones`: Arreglo de posiciones de las partículas.
- `w_i`: Arreglo de valores intermedios calculados por `w_ih`.
- `h`: Incremento de tiempo.

La función actualiza las posiciones $r(t + h)$ para cada partícula i usando la fórmula:

$$posiciones[i] = posiciones[i] + w_i[i] \cdot h$$

4.3 Función de velocidad en el tiempo $t + h$

La función `velocidad_th` calcula las nuevas velocidades de las partículas en el tiempo $t + h$ y la energía cinética del sistema.

Listing 4: Función de velocidad en el tiempo $t + h$

```
@jit(nopython=True, fastmath=True)
def velocidad_th(w_i, n, velocidades, a_i_th, h):
    E_c = 0
    for i in range(n):
        velocidades[i] = w_i[i] + a_i_th[i]*(h/2)
        E_c += energia_cinetica(velocidades[i])
    return velocidades, E_c
```

La función `velocidad_th` toma los siguientes parámetros:

- `w_i`: Arreglo de valores intermedios calculados por `w_ih`.
- `n`: Número de partículas.
- `velocidades`: Arreglo de velocidades de las partículas.
- `a_i_th`: Arreglo de aceleraciones en el tiempo $t + h$.
- `h`: Incremento de tiempo.

La función actualiza las velocidades $v(t+h)$ para cada partícula i usando la fórmula:

$$velocidades[i] = w_i[i] + a_{it}h[i] \cdot \left(\frac{h}{2}\right)$$

y calcula la energía cinética total del sistema:

$$E_c = \sum_{i=0}^{n-1} \text{energia\_cinetica}(velocidades[i])$$

donde `energia_cinetica` es una función que calcula la energía cinética de una partícula individual.

5 Potencial de Lennard-Jones

El potencial de Lennard-Jones se define como:

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], \tag{3}$$

donde ϵ y σ son constantes que fijan la escala de energía y distancia de la interacción, respectivamente. En el caso de esta simulación estableceremos

estos dos parámetros como 1. La fuerza entre dos partículas se calcula como la derivada del potencial:

$$\mathbf{F}(r) = -\frac{\partial V}{\partial r} = 24\epsilon \left[2\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \frac{\mathbf{r}}{r}. \tag{4}$$

- **La implementación para el cálculo de la acceleración de la partiícula en cada iteración**:

```
#Definimos la funci n que nos da la acceleracion
#en el tiempo t+h.
@jit(nopython=True, fastmath=True)
def acel_i_th(n, posiciones, a_i, l, E_p_c, a_c):
    E_p = 0
    a_i = np.zeros((n, 2))
    for i in range(n):
        j = i+1
        while j < n:
            distancia, direccion =
            distancia_condiciones(posiciones, i, j, l)
            versor = direccion/distancia
            if distancia <= 3:
                aceleracion = (24/distancia**7)
                (2/distancia**6 - 1) - a_c
                a_i[i] = a_i[i] + aceleracion*versor
                a_i[j] = a_i[j] - aceleracion*versor
                E_p += (4/distancia**6)*
                (1/distancia**6 - 1)- E_p_c +
                (distancia - 3)*a_c
            j = j+1
        return a_i, E_p
```

6 Condiciones de Contorno Periódicas

En una simulación bidimensional, las condiciones de contorno periódicas se implementan para evitar efectos de borde. Cuando una partícula atraviesa una frontera, entra al sistema por el lado opuesto. Por ejemplo, si una partícula pasa por $x = L$, reaparece en $x = 0$.

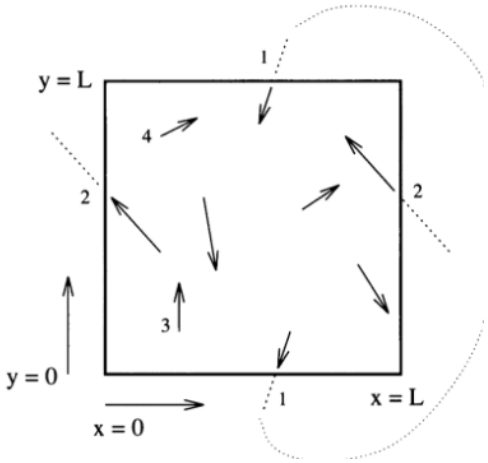

Figure 1: Condiciones de contorno periódicas en una caja $L \times L$.

6.1 Condiciones de Contorno Periódicas

Las condiciones de contorno periódicas se implementan para asegurar que las partículas que se mueven fuera de los límites de la caja de simulación reaparezcan en el lado opuesto. Además, se calcula el momento transferido a la caja cuando una partícula cruza el límite.

Listing 5: Función para aplicar condiciones de contorno periódicas y calcular el momento transferido

```
#Condiciones de contorno peri dicas y c lculo
#del momento transferido a la caja.
@jit(nopython=True, fastmath=True)
def contorno(posiciones, l, n, velocidades):
    momento = np.zeros((n, 2))
    for i in range(n):
        x = posiciones[i][0]
        y = posiciones[i][1]
        if x > l:
            x = x % l
            momento[i][0] = velocidades[i][0]
        if x < 0:
            x = x % l
            momento[i][0] = velocidades[i][0]
        if y > l:
            y = y % l
            momento[i][1] = velocidades[i][1]
```

```
        if y < 0:
            y = y % l
            momento[i][1] = velocidades[i][1]
        posiciones[i][0] = x
        posiciones[i][1] = y
    return posiciones, 2*momento
```

En esta función, se recorren todas las partículas para verificar si sus coordenadas x o y están fuera del rango $[0, l]$. Si una coordenada está fuera de este rango, se ajusta usando la operación módulo ($\%l$), asegurando que la partícula reaparezca en el lado opuesto. Además, se registra el momento transferido a la caja cuando una partícula cruza el límite.

6.2 Cálculo de la Distancia Entre Partículas

Para calcular la distancia entre dos partículas, se debe considerar las condiciones de contorno periódicas para asegurar que la distancia calculada sea la más corta posible. Esto se realiza ajustando las distancias cuando las partículas están cerca de los límites de la caja.

Listing 6: Función para calcular la distancia entre dos partículas considerando las condiciones de contorno

```
#Funci n para calcular la distancia entre dos part culas.
@jit(nopython=True, fastmath=True)
def distancia_condiciones(posicion, i, j, l):
    resta = np.zeros(2)
    mitad = l / 2

    resta[0] = posicion[i][0] - posicion[j][0]
    if abs(resta[0]) > mitad:
        resta[0] = -(l - abs(resta[0]))*np.sign(resta[0])

    resta[1] = posicion[i][1] - posicion[j][1]
    if abs(resta[1]) > mitad:
        resta[1] = -(l - abs(resta[1]))*np.sign(resta[1])

    distancia = np.sqrt(resta[0]**2 + resta[1]**2)

    return distancia, resta
```

En esta función, se calcula la distancia en cada dirección (x e y) y se ajusta si esta distancia es mayor que la mitad de la longitud de la caja ($l/2$).

Esto asegura que se utilice la distancia más corta posible considerando las condiciones de contorno periódicas. La distancia total se calcula usando la fórmula de la distancia euclidiana:

$$distancia = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Esta implementación asegura que las simulaciones reflejen correctamente la naturaleza periódica del espacio simulado y que las interacciones entre partículas sean calculadas adecuadamente, independientemente de su posición relativa dentro de la caja.

7 Actividades a Realizar

7.1 Simulación Inicial

1. Simular un gas con 20 átomos de argón en una caja de 10×10 unidades con posiciones iniciales aleatorias y velocidades iniciales de módulo 1 y ángulo aleatorio (distribuido uniformemente entre 0 y 2π). Tomar un paso temporal de $\Delta t = 0.002$.

A. Representar la evolución de las posiciones de las partículas en un gif.

Vídeo de la simulación

B. Almacenar la energía cinética y potencial del sistema. Representar estas energías, junto con la energía total, frente al tiempo para estudiar la relajación al estado de equilibrio.

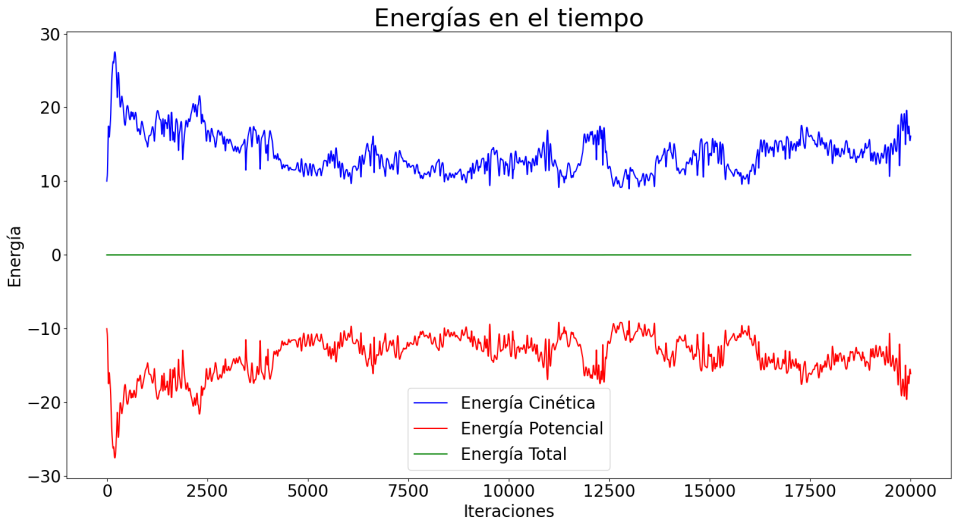

Figure 2: La energía del sistema se conserva. Esta simulación se han realizado 80000 iteraciones con un paso temporal de $\Delta t = 0.002$.

Observamos que para esta configuración inicial pedida el estado de equilibrio se alcanza rápidamente. Es un equilibrio inestable ya que hay fluctuaciones de la energía cinética y potencial a lo largo de la simulación, pero están oscilando en torno a un valor medio de 13.5 ± 0.2 la cinética y -13.5 ± 0.2 la potencial.

C. Calcular la temperatura a partir del teorema de equipartición:

$$k_B T = \frac{m}{2} \left\langle v_x^2 + v_y^2 \right\rangle , \tag{5}$$

donde $k_B = 1$ y $m = 1$.

Para este resultado se han realizado 4 simulaciones y se ha promediado y calculado el error obteniendo que la temperatura sería de $0,664 \pm 0,015$. Es una temperatura muy baja como era de esperar, tenemos muy pocas particulas a baja velocidad.

D. Representar Histogramas de velocidades en un único plot.

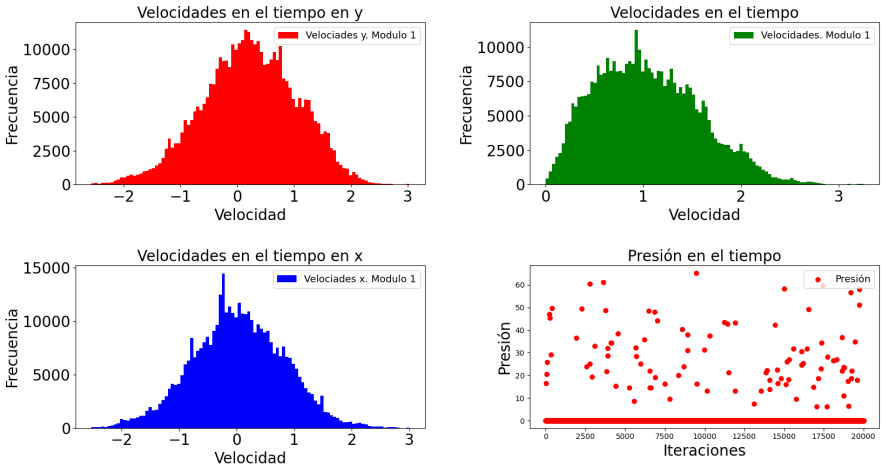

Figure 3: Velocidades del sistema en módulo y para el en eje x e y. Tambíen están las presiones para cada instante temporal.

7.2 Variación de Condiciones Iniciales de Velocidad

1.Estudiar el caso en el que inicialmente los átomos tienen una v_x positiva ($v_x > 0$ y $v_y = 0$), eligiendo, por ejemplo, una v_x aleatoria distribuida uniformemente en el rango $0 - 1$.

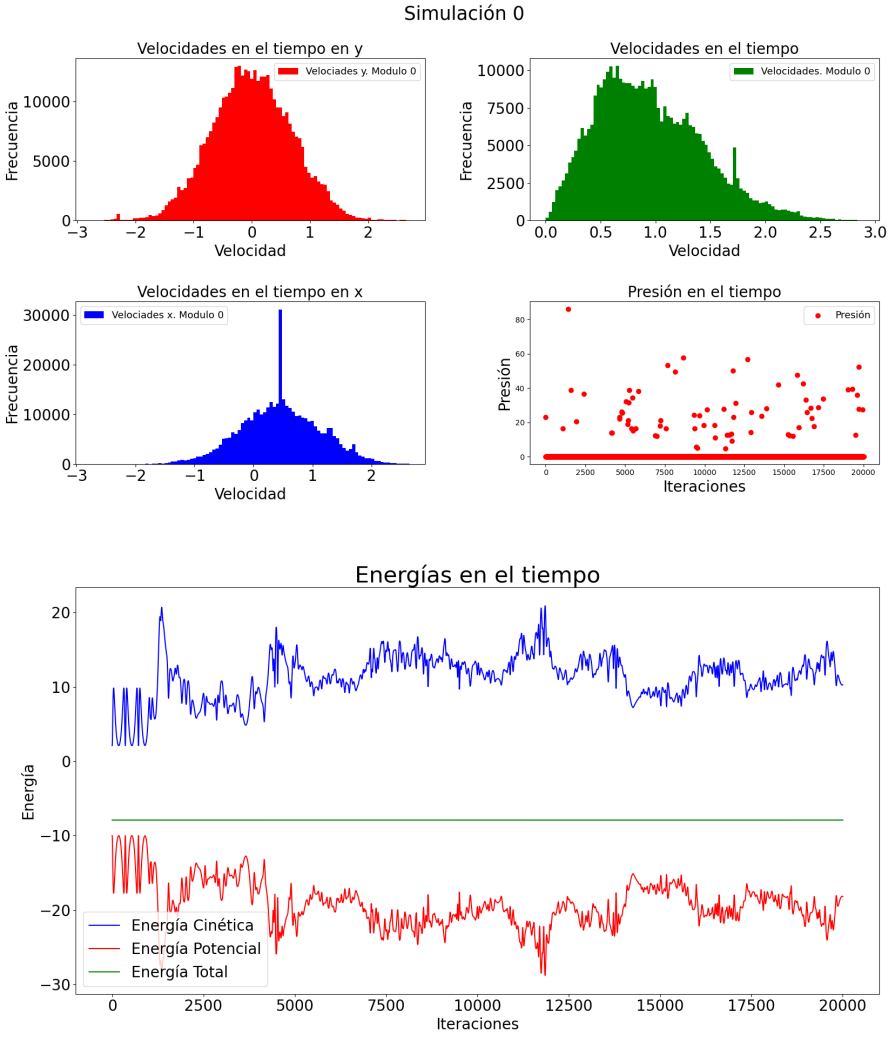

Figure 4: Simulación 0. Velocidades del sistema en módulo y para el en eje x e y. Tambíen están las presiones para cada instante temporal. En este caso las velocidades iniciales para el eje y son cero y para el eje x tienen un valor entre 0 y 1 aleatorio

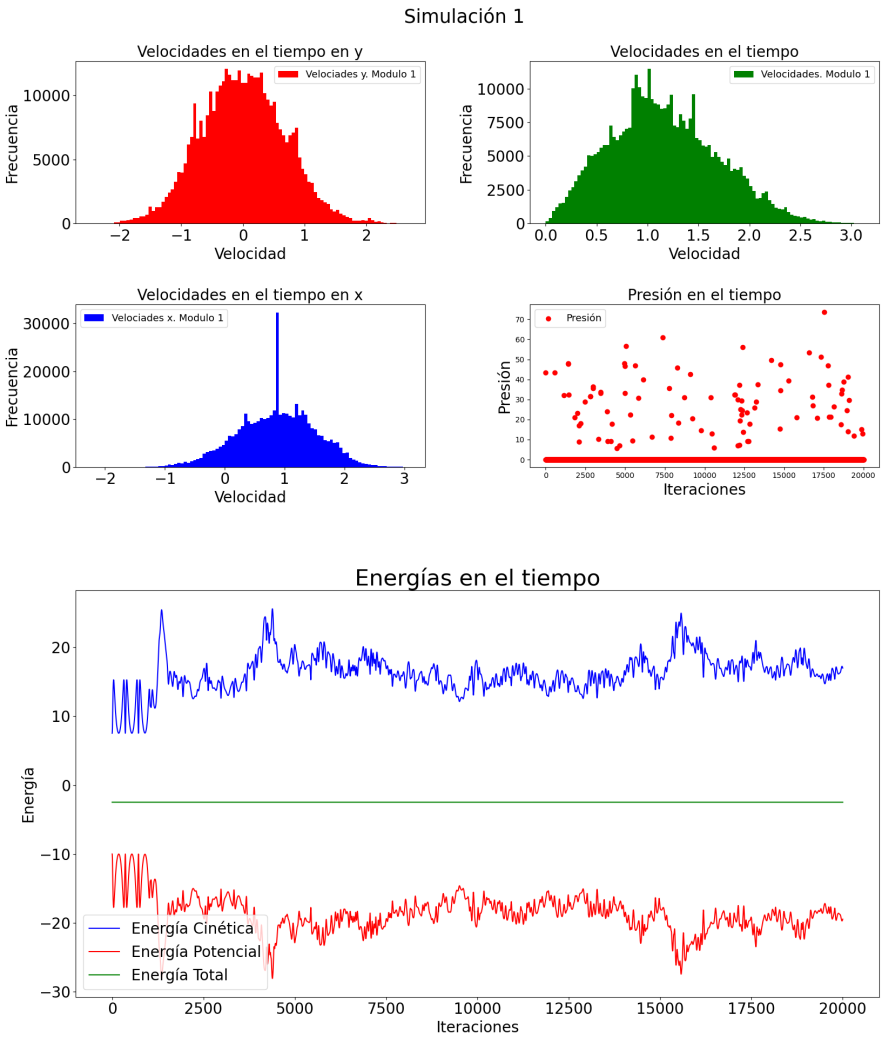

Figure 5: Simulación 1. Velocidades del sistema en módulo y para el en eje x e y. Tambíen están las presiones para cada instante temporal. En este caso las velocidades iniciales para el eje y son cero y para el eje x tienen un valor entre 0 y 1 aleatorio

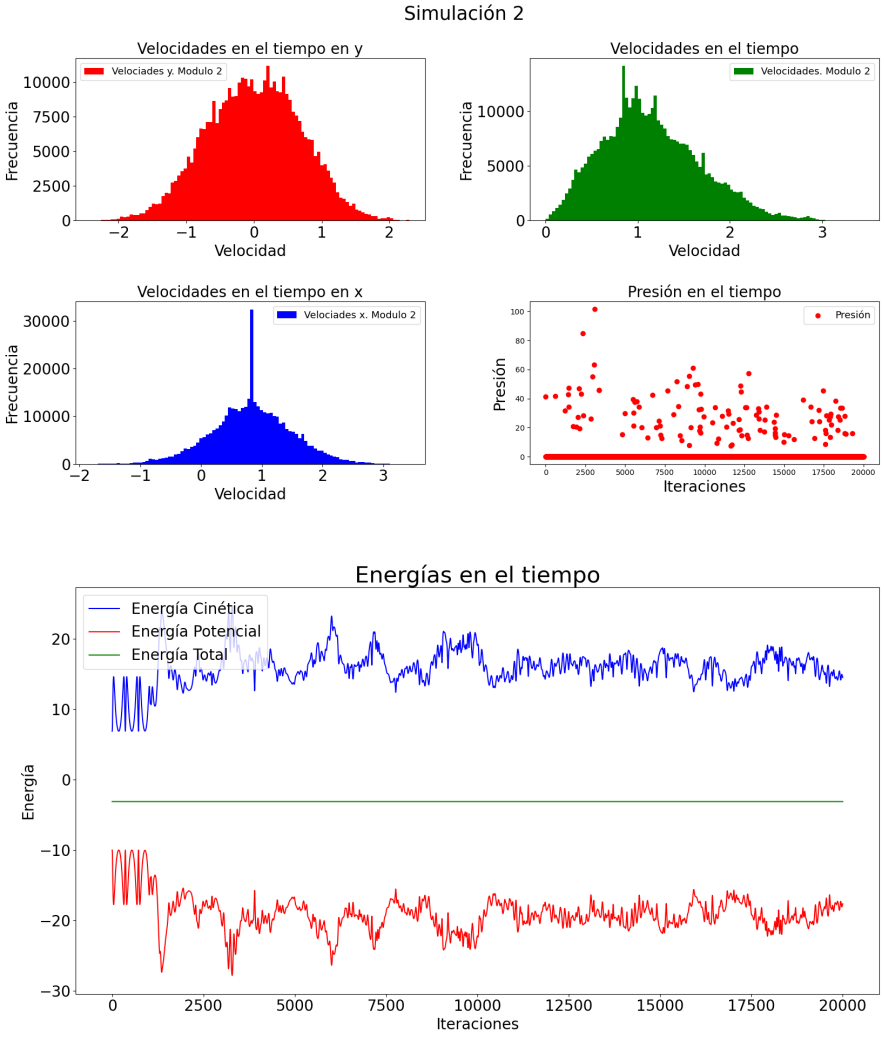

Figure 6: Simulación 2. Velocidades del sistema en módulo y para el en eje x e y. Tambíen están las presiones para cada instante temporal. En este caso las velocidades iniciales para el eje y son cero y para el eje x tienen un valor entre 0 y 1 aleatorio

Se han realizado tres simulaciones para estudiar el comportamiebto del sistema en estas condiciones. Vemos como ahora el sistema ya no tiene una distribución de velocidades tipo Boltszman tan bien marcada como antes. Las energías por otro lado resultan ser más estables.

7.3 Ecuación de Estado del Sistema

1. Estudiar la ecuación de estado midiendo la presión en función de la temperatura para 20 partículas en una caja 10×10 con $\Delta t = 0.002$. 2. La presión se calcula a partir del cambio de momento que las partículas ejercen al 'atravesar' las fronteras del sistema. Mostrar que la presión varía linealmente con la temperatura y explicar por qué.

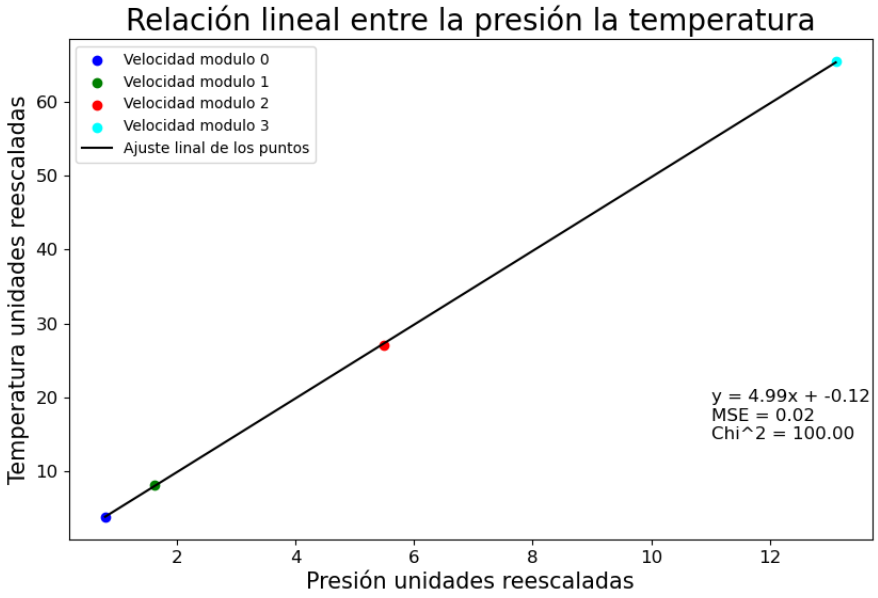

Figure 7: Ajuste lineal de la temperatura del sistema frnte a la presión. Se ha representado en el gráfico el coeficiente de ajuste Chi cuadrado y la suma de los residuos del ajuste como comprobantes del error.

Dada la ecuación 5 ya sabíamos de ante mano que la temperatura depende diréctamente de la velocidad de las partículas del sistema. La presión por otra parte también depende de la velocidad de las partículas del sistema, ya que al chocar con las paredes, éstas les transfieren su momento lineal a las paredes del recipiente, además salen rebotadas con la misma velocidad pero sentido contrario a la normal de la pared con la colisionaron. Por tanto:

$$\Delta p = 2mv * \hat{n} \tag{6}$$

Además la Fuerza se calcula como:

$$F = \frac{\Delta p}{\Delta t} \tag{7}$$

Y la presión es $P = F/A$ siendo A el área de contacto de las partículas. Por tanto si sustituímos en la expresión anterior tenemos que:

$$P = \frac{\Delta p}{\Delta t * A} \tag{8}$$

7.4 Transición de Fase Sólido-Líquido

1. Simular un sistema con 16 partículas en una caja 4×4, comenzando con las partículas dispuestas en una red cuadrada y en reposo. 2. Simular el sistema hasta que se alcance el equilibrio. Observar cómo las partículas se disponen en una estructura triangular, característica del estado sólido.

Vídeos de la simulación para diferentes condiciones iniciales

7.5 Estudio de Diferentes Configuraciones Iniciales

1. Demostrar que, independientemente de la disposición inicial (por ejemplo, desordenada o en forma de panal), el sistema evoluciona hacia la estructura triangular característica del estado sólido.

Vídeos de la simulación para diferentes condiciones iniciales

7.6 Variación de la Temperatura

1. Rescalar las velocidades por un factor 1.5 en los tiempos $t = 20$, $t = 30$, $t = 35$ y $t = 45$. 2. Observar la pérdida de la estructura sólida y la transición a un estado líquido. 3. Estudiar la evolución de las fluctuaciones de la posición de una partícula cualquiera:

$$\left\langle (\mathbf{r}(t) - \mathbf{r}(0))^2 \right\rangle, \tag{9}$$

y observar cómo estas fluctuaciones se incrementan con la temperatura. Las siguientes mediciones se han realizado en una caja de 6×6 con 15 partículas.

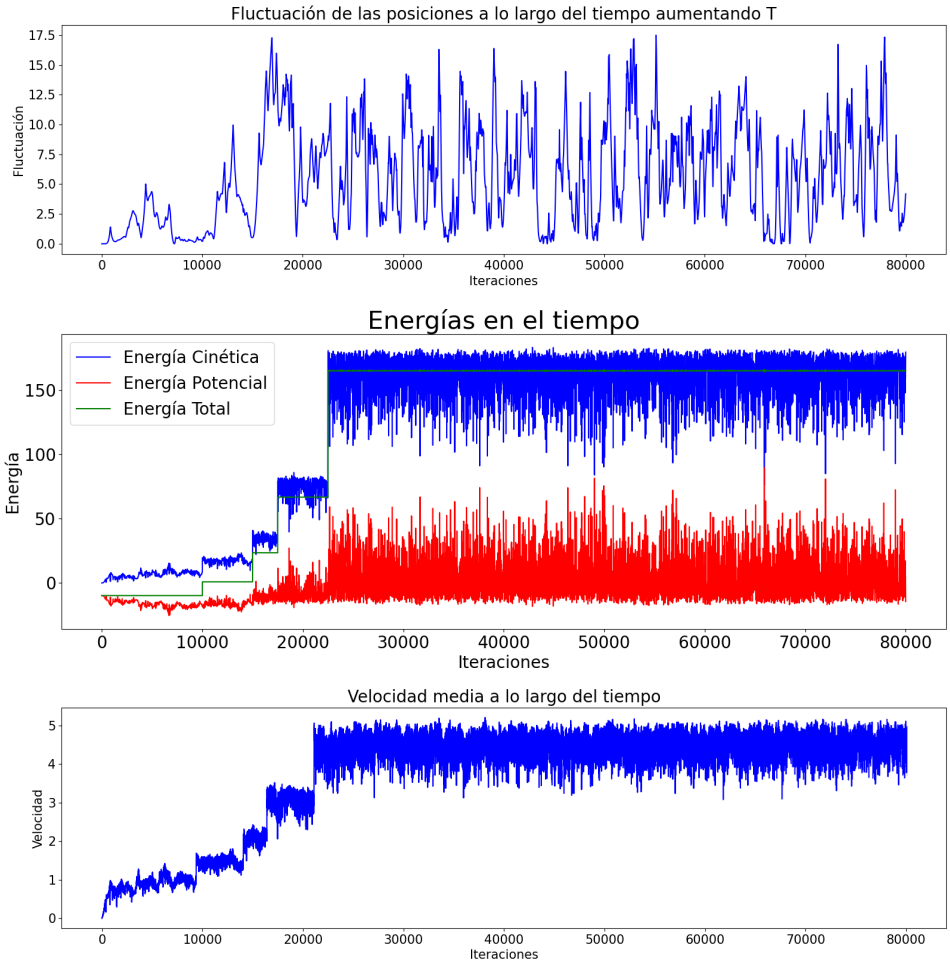

Figure 8: En el gráfico superior vemos las fluctuaciones de posición relativa. En el central podemos ver las energías del sistema. Y el inferior mustra el módulo de la media de velocidades.

Podemos ver la relación entre los incrementos de velocidad de las partículas del sistema y por ende de las temperaturas con los picos de las fluctuaciones de la posición. Sin embargo no parece quedar muy bien definido el punto en el que hay un cambio de fase, es decir el momento en el que hay un incremento significativo de las fluctuaciones. Podríamos considerar que este incremento ocurre cuando la energía cinética está entre 50 y 100. Cosa que se respalda si observamos que la propia energía cinética fluctúa en mayor medida a partir de este punto. La temperatura crítica sería entonces 4 ± 2 unidades reescaladas.

7.7 Determinación de la Temperatura Crítica

1. Estudiar la separación cuadrática media de un par de átomos:

$$\left\langle (\mathbf{r}_i(t) - \mathbf{r}_j(t))^2 \right\rangle, \tag{10}$$

como función de la temperatura. 2. Calentar el sistema lentamente rescalando las velocidades de las partículas por un factor de 1.1 en los tiempos $t = 60, 120, 180$, etc. 3. Determinar la temperatura crítica alrededor de la cual los átomos experimentan un alejamiento considerable y abrupto de sus posiciones. Se ha vuelto a usar una caja de 6×6 con 15 partículas.

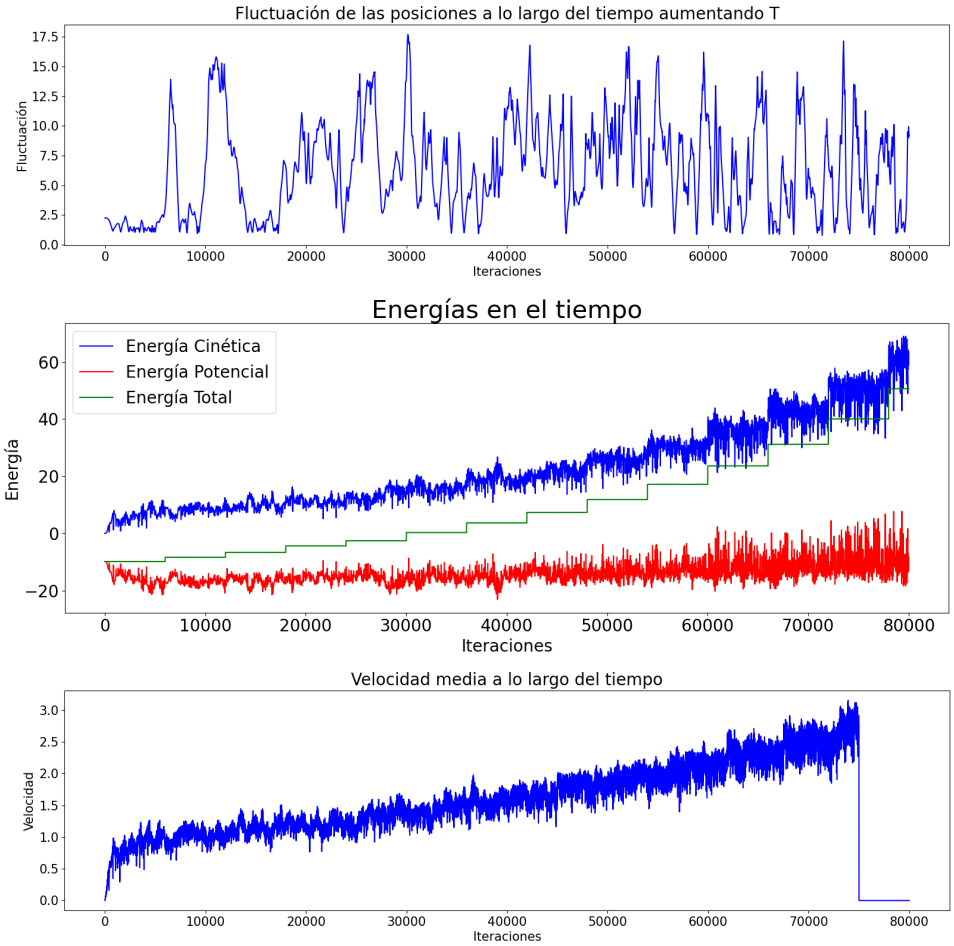

Figure 9: En el gráfico superior vemos las fluctuaciones de posición relativa. En el central podemos ver las energías del sistema. Y el inferior mustra el módulo de la media de velocidades.

En este caso parece precisarse un poco más la energía cinética ya que

las fluctuaciones ocurren cuando tiene un valor de entre 40 y 60. Por tanto ese sería el valor de la temperatura crítica del sistema, sin poder precisarlo más con este método. Esto nos daría una T crítica de 3.2 ± 0.72 unidades reescaladas.

8 Apartado de optimización y alto rendimiento

En primer lugar se ha realizado la comparación de tiempo de ejecución entre el programa con numba en todas las funciones permitidas y sólo en las funciones principales del programa. La optimización del programa con numba queda evidenciada en este gráfico.

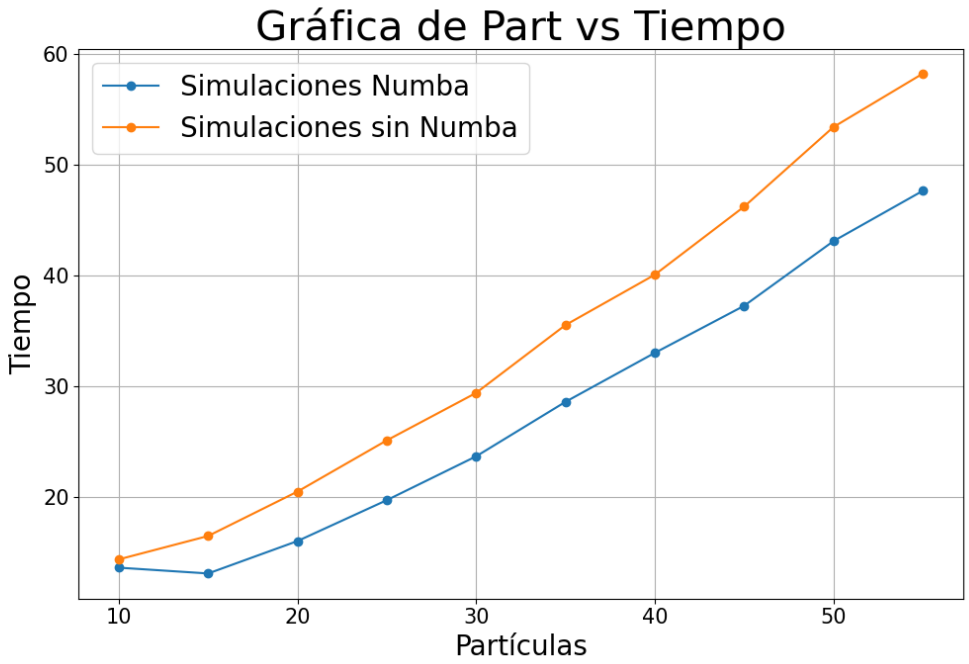

Figure 10: En el gráfico superior vemos los tiempos de ejecución de las simulaciones en función de las partículas del sistema, para ambos casos.

A continuación se ha realizado el mismo experimento para comparar el programa completamente optimizado ejecutándose en Joel y en mi ordenador personal el cuál tiene un procesador 12th Gen Intel(R) Core(TM) i5-12400F con seis núcleas de frecuencia base de 2.50 GHz y de máxima de 4.4 GHz.

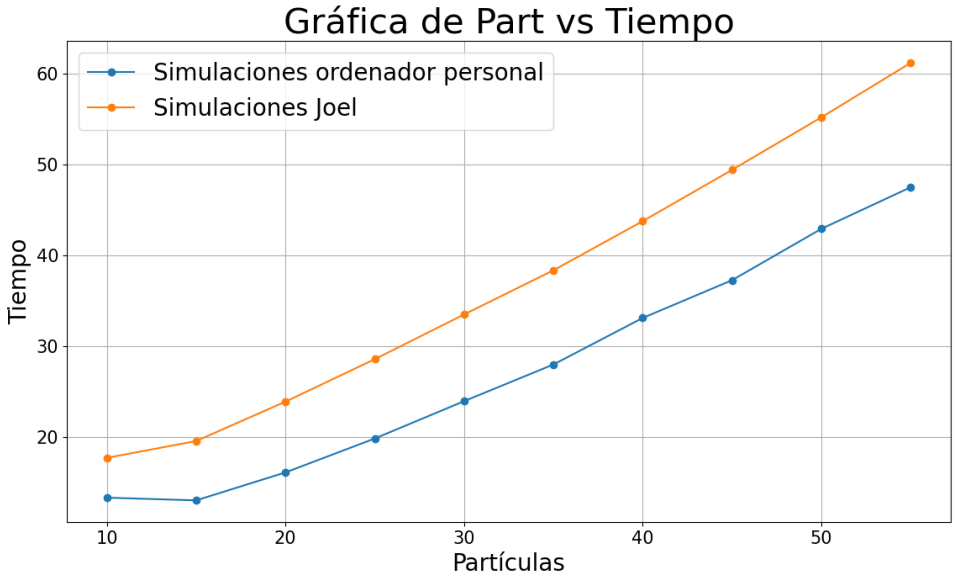

Figure 11: En el gráfico superior vemos los tiempos de ejecución de las simulaciones en función de las partículas del sistema, para ambos casos.

Claramente el ordenador personal es más rápido que Joel para un núcleo, además al tener menos núcleos disponibles también sería más lento en multinúcleo, que el procesador del ordenador personal.

Al no ser un problema paralelizable, y tener funciones muy concretas en mi caso el paralelizar el programa sólo hace que la simulación tarde mucho más. Esto se debe a que los núcleos se tienen que comunicar entre sí para gestionar la información que calcule cada uno, y en este proceso de comunicación entr núcleos se pierde mucho más tiempo que realizando todo en cadena en un sólo núcleo. Si el problema fuera paralelizable no sería necesario que los núcleos se comunicaran contínuamente entre sí, y sólo en algunos momentos específicos, entonces el rendimiento si mejoraría.

Para conseguir esto hice que se realizaran simulaciones independientes en cada núcleo simultáneamente lo que permite realizar en caso de mi ordenador personal 6 simulaciones simultáneas lo cual claramente reduce mucho el tiempo de espera si se necesitan realizar muchas simulaciones con distintos parámetros. Los tiempos de ejecución en este caso serían los mismos que para la simulación que tarde menos en mononúcleo. Si se quisieran realizar más simulaciones de seis lo óptimo sería realizar tandas de seis en seis siempre intentando que sean del mismo orden de tiempo de ejecución para no desperdiciar tiempo de cómputo.