

Informe de Laboratorio N°6

Alumno: Méndez Martín
 Docentes: Dra. Marconi Verónica I.; Dr. Banchio Adolfo
 Universidad Nacional de Córdoba (UNC)
 Facultad de Matemática, Astronomía, Física y Computación (FaMAF)
 Curso de Física Computacional

(*) Para mayor claridad se recomienda ver figuras en la versión digital.

I. INTRODUCCIÓN

I-1. Dinámica Browniana

El movimiento *browniano* (BD) se podría definir como el movimiento de partículas (sólido de tamaño mesoscópico) inmerso en un líquido (solvente de tamaño microscópico), el cual se caracteriza por ser un movimiento azaroso. Ejemplos de estos movimientos podrían ser: el movimiento de partículas de polvo en el aire (haz de luz que emite un proyector), partículas de humo en el aire (bocanada de humo que expulsa un fumador hacia el aire) ó partículas de polen inmersas en agua.

I-2. Fuerza hidrodinámica, fuerza estocástica y escalas de tiempo

Si una partícula mesoscópica se sumerge en un medio compuesto de partículas microscópicas y considerando a este medio microscópico como un fluido laminar, entonces, según la hidrodinámica podemos decir que la partícula mesoscópica experimentará una fuerza de fricción que dependerá de su velocidad (fuerza viscosa). Por otro lado, la partícula mesoscópica experimentará fuerzas debidas al gran número de colisiones que ocurren entre las partículas microscópicas hacia las mesoscópicas (en general del orden de $\mathcal{O}(10^{23})$), estas fuerzas varían de forma azarosa y violenta (en escalas de tiempo del orden de $\tau_s \sim \mathcal{O}(10^{-13})[s]$), pues si hacemos observaciones en las escalas de tiempo mesoscópicas ($t \gg \tau_s$) habrán ocurrido muchas colisiones y observaremos un efecto promedio de estas fuerzas estocásticas, sin embargo, en esta escala de tiempo las fuerzas viscósas comenzarán a experimentarse y variarán muy poco (ver figuras 1, 2). La fuerza estocástica

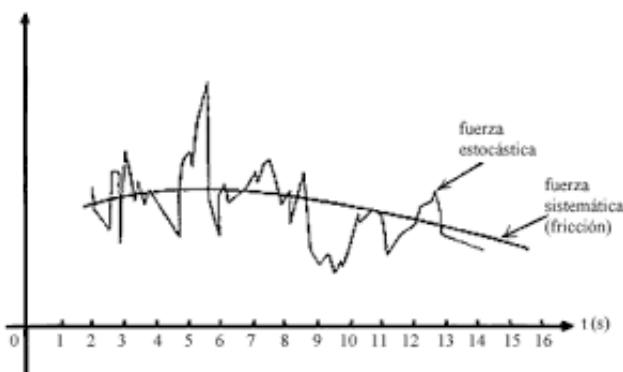


Fig. 1. Variación de fuerza estocástica y viscosa, diferencia de escalas temporales

describe entonces el efecto acumulativo de la gran cantidad de colisiones que producen las moléculas del solvente sobre el sólido. Estas colisiones, se pueden considerar estadísticamente independientes para tiempos largos comparados con τ_s y, de

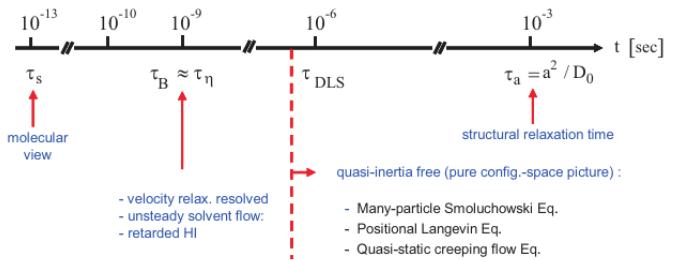


Fig. 2. Escalas temporales: valores típicos para partículas de radio de $100[\text{nm}]$ suspendidas en agua. τ_{DLS} es el tiempo de resolución para la dinámica de difracción de luz

acuerdo al teorema central del límite al sumar de muchas variables aleatorias con segundo momento finito, la distribución de probabilidad converge a una distribución gaussiana, entonces la fuerza estocástica puede ser descrita completamente como una cantidad fluctuante cuyas propiedades estadísticas están completamente determinadas por una distribución gaussiana.

I-3. Ecuación de velocidad de Langevin

Considerando una partícula esférica de masa m (cuyo radio es del orden de $\mathcal{O}(10^{-6}, 10^{-9})[m]$) suspendida en un solvente (en régimen de flujo laminar $Re \ll 1$) compuesto de moléculas (cuyo radio es del orden de $\mathcal{O}(10^{-10})[m]$) entonces, si observamos el movimiento de esta partícula mesoscópica con una resolución temporal que cumple $\Delta t \gg \tau_s$ entonces la ecuación de movimiento para la velocidad traslacional $\vec{v}(t) = \dot{\vec{r}}(t)$ está dada por la ecuación de Langevin según la ecuación 1 donde el primer término del lado derecho corresponde a la fuerza de fricción promedio y el segundo del lado derecho corresponde a la fuerza estocástica fluctuante. Por otro lado, el lado izquierdo corresponde a la fuerza inercial que experimenta la partícula.

$$m \ddot{\vec{r}}(t) = -\zeta_0 \dot{\vec{r}}(t) + \vec{F}^s(t) \quad (1)$$

En la escala temporal donde se realizan las observaciones el solvente se comporta como un medio continuo y como estamos tratando con un flujo en régimen laminar (comportamiento según flujo de Stokes, linealizando la ecuación de Navier-Stokes, a bajo número de Raynolds), por lo tanto, el coeficiente de fricción ζ_0 de una esfera puede aproximarse según la relación $\zeta_0 = 6\pi\eta_0 a$ donde η_0 es la viscosidad dinámica de el fluido y a el radio de la partícula esférica.

I-4. Teorema de fluctuación-disipación

La relación de fluctuación-disipación es la piedra fundamental de la teoría de respuesta lineal. Esta relaciona la relajación de un sistema perturbado débilmente con las fluctuaciones espontáneas en equilibrio térmico. En otras palabras, si un sistema está en el instante inicial en un estado fuera del

equilibrio, el sistema no sabe si fue puesto en tal estado por una fuerza externa o como resultado de una fluctuación al azar, entonces, la evolución subsecuente hacia el equilibrio será la misma en ambos casos. Por lo tanto, el teorema de fluctuación-disipación nos permite encontrar la relación que vincula las fuerzas viscosas (o de resistencia hidrodinámica) con las fuerzas estocásticas pues, el arrastre disipa energía cinética transformándola en calor y la fluctuación correspondiente es el movimiento browniano (movimiento azaroso) y este movimiento convierte energía térmica en energía cinética nuevamente.

I-5. Ecuación de posición de Langevin

Si realizamos observaciones en la escala temporal $t \gg \tau_B$ (donde $\tau_B \gg \tau_s$ es el tiempo de relajación de los momentos) entonces el desplazamiento cuadrático medio resulta lineal, en otras palabras, no resolvemos la relajación del movimiento y consideramos que las velocidades ya cuentan con una distribución de Maxwell-Boltzmann. Además, para observar un cambio configuracional (posicional) las partículas mesoscópicas deberán difundir en el solvente una distancia comparable con su propio tamaño, entonces, se define el tiempo de relajación estructural (el cual es necesario para cambios configuracionales apreciables) como $\tau_a = a^2/D_0$, donde $D_0 = k_B T/\eta_0$ es el coeficiente de fricción. Entonces, para tiempos $\tau_a \gg \tau_B$ los momentos ya habrán relajado completamente y sólo resolveremos la parte configuracional del problema. Una resolución temporal $\Delta t \gg \tau_B$ tiene asociada una resolución configuracional o espacial $\Delta x \gg l_B = \sqrt{D_0 \tau_B}$ y los problemas analizados en esta escala se rigen por la dinámica browniana y se estudian procesos de difusión según la dinámica de Smoluchowski donde el término inercial es despreciable (esto ocurre si las densidades del soluto y el solvente son del mismo orden, y el tiempo de respuesta es muy corto, tenemos un sistema sobreamortiguado. "Free-draining approximation"). Teniendo en cuenta que $l_B \sim \mathcal{O}(10^{-3})[m]$ la difusión de partículas coloidales en la escala temporal de BD se verifican utilizando técnicas de difracción de luz (*dynamic light scattering techniques*).

I-6. Potencial de Lennard-Jones

En el presente trabajo consideramos como potencial de interacción el potencial interatómico de Lennard-Jones (ver figura 3) el cual sirve como modelo de fuerzas de enlace entre pares de moléculas y es utilizado para calcular la fuerza de van der Waals. Este potencial depende básicamente de tres parámetros (ver figura 2) r_{ij} que es la distancia relativa entre centros de masa de un par de partículas, ϵ que es la magnitud del pozo de potencial y σ es la distancia interatómica para la cual el potencial se anula. Además, podemos notar que el primer término del potencial tiene en cuenta qué tan intensa es la repulsión entre el par de partículas a medida que aumenta (o disminuye) la distancia interatómica, y el segundo término tiene en cuenta qué tan intensa es la atracción entre el par de partículas a medida que aumenta (o disminuye) la distancia interatómica.

$$u_{ij} = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 \right] \quad (2)$$

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

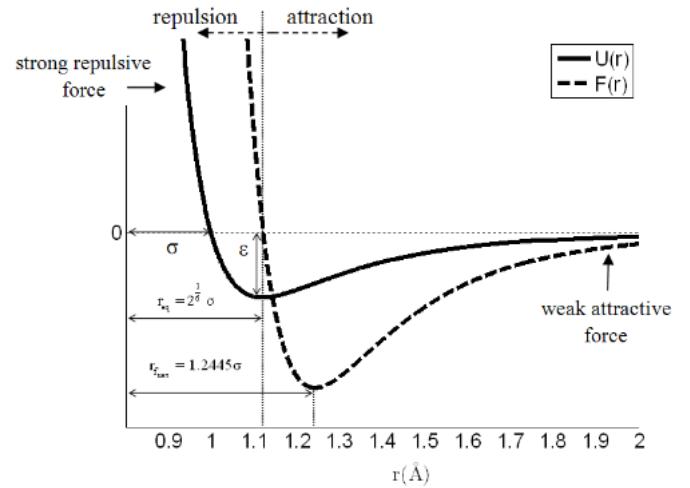


Fig. 3. Lennard-Jones: Potencial y fuerzas de interacción vs distancia interatómica

Ahora bien, para las simulaciones es necesario adimensionalizar todas las magnitudes que tengan dimensiones, entonces el potencial individual quedará de la forma según 3.

$$u_{ij}^* = 4 \left(\frac{1}{r_{ij}^*} \right)^6 \left[\left(\frac{1}{r_{ij}^*} \right)^6 - 1 \right]; u_{ij}^* = \frac{u_{ij}}{\epsilon}; r_{ij}^* = \frac{r_{ij}}{\sigma} \quad (3)$$

Además, cabe mencionar que para evitar una discontinuidad en el potencial se utiliza el potencial truncado y desplazado (Lennard-Jones truncated shifted potential - LJTS) que se define de la siguiente manera;

$$u_{ij}^* = 4 \left(\frac{1}{r_{ij}^*} \right)^6 \left[\left(\frac{1}{r_{ij}^*} \right)^6 - 1 \right] - \dots \quad (4)$$

$$\dots - 4 \left(\frac{1}{r_{cutoff}} \right)^6 \left[\left(\frac{1}{r_{cutoff}} \right)^6 - 1 \right]$$

y en la figura 4 se pueden observar las diferencias entre el método de truncado simple del potencial y el método de truncado y desplazado.

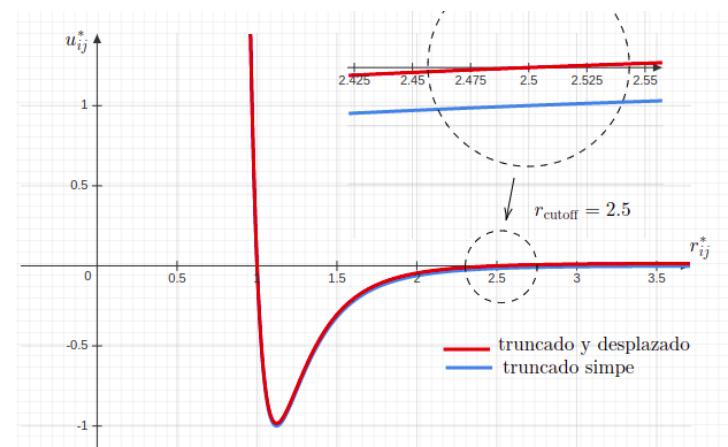


Fig. 4. LJTS vs distancia interatómica

Por otro lado, la fuerza interatómica se define como el opuesto al gradiente del potencial entre pares y en forma adimensional quedará según la ecuación 5. Y en componentes será según 6.

$$f_{r,ij}^* = -\frac{\partial u_{ij}^*}{\partial r_{ij}^*} = 24 \frac{1}{r_{ij}^*} \left(\frac{1}{r_{ij}^*} \right)^6 \left[2 \left(\frac{1}{r_{ij}^*} \right)^6 - 1 \right] \quad (5)$$

$$f_{x_k,ij}^* = \frac{r_{x_k,ij}^*}{r_{ij}^*} f_{r,ij}^*; x_k = \{x, y, z\} \quad (6)$$

Finalmente el potencial total será la suma de los potenciales individuales como muestra la ecuación 7. Notemos que se ha definido una distancia de corte entre pares r_{cutoff} este importante parámetro nos permite decidir hasta qué distancia podemos considerar al potencial de interacción entre pares relevantes, es decir, para distancias interatómicas mayores a este radio el potencial es despreciable y lo consideramos, a efectos prácticos, nulo, en otras palabras, este radio de corte nos permite determinar cuántas partículas vecinas interactúan cuando nos situamos en una dada partícula, la magnitud de este radio estará fuertemente influenciado respecto a si estamos tratando con interacciones de largo o corto alcance, además, deberá ser menor a la mitad de la longitud L de la celda periódica como veremos luego.

$$U_{tot} = \sum_{j=2}^{n_p} \sum_{i=1}^{j-1} u_{ij}^*; \forall r_{ij}^* \leq r_{cutoff} \quad (7)$$

II. RESULTADOS Y DISCUSIONES

Primeramente se consideró un sistema de $n_p = 500$ partículas, densidad $\rho = 0,8$ (partículas por unidad de volumen), temperatura de referencia adimensional $T_{ref} = 1$, paso temporal adimensional de $\Delta t = 0,001$ radio de corte adimensional de $r_{cutoff} = 2,5$, viscosidad dinámica adimensional de $\eta_0 = 2,87$.

En la figura I3 se puede observar una animación de la dinámica browniana para los parámetros mencionados y en la sección III se puede consultar el video completo acumulando un total de 100 imágenes para su implementación y usando el paquete de librerías ffmpeg. El código desarrollado corresponde a la subrutina **create_movie** dentro del programa **brownian_dynamic_lennard_jones_01.f90**.

II-A. Configuración de estructura cristalina inicial

La configuración de las posiciones iniciales de las partículas considerada fue la de una estructura cúbica centrada en las caras (FCC), la cual tiene un total de 4 átomos por celda unidad, y cuyos vectores primitivos son $\vec{a}_1 = \frac{a}{2}(\hat{e}_x + \hat{e}_y)$; $\vec{a}_2 = \frac{a}{2}(\hat{e}_y + \hat{e}_z)$; $\vec{a}_3 = \frac{a}{2}(\hat{e}_x + \hat{e}_z)$, entonces, como el total de partículas debe conservarse, la celda unidad deberá repetirse un cierto número de veces de tal forma de respetar este vínculo y además cumplir con la densidad impuesta externamente ρ . El código desarrollado corresponde a la subrutina **initial_lattice_configuration** dentro del módulo **module_bd_lennard_jones.f90** y en la figura 5 se puede observar la disposición de partículas en el estado inicial, la misma fue centrada en el rango $[-L/2; L/2]$.

II-B. Condiciones de contorno periódicas y corrección por imagen mínima

Teniendo en cuenta que quieren estudiar propiedades en el equilibrio termodinámico ($n_p \rightarrow \infty$) y debido a que no nos interesan estudiar efectos de superficie, es decir, excluyendo el comportamiento de las partículas cerca del borde físico del sistema macroscópico, podremos eliminar dicho borde considerando una celda de estudio que se repite infinitamente,

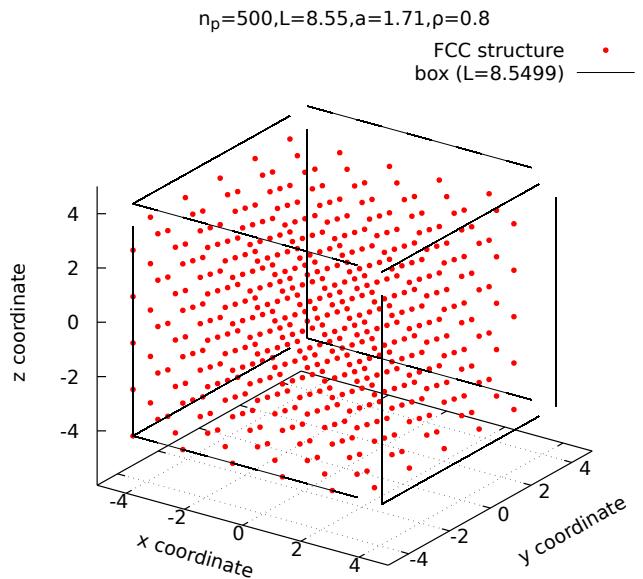


Fig. 5. Posiciones iniciales de las partículas: Estructura FCC

es decir, llenamos el espacio de copias idénticas de estas regiones de simulación. De esta manera una partícula que sale de esta región de estudio por alguna delimitación particular de esta, deberá ser remplazada inmediatamente por otra que tiene el mismo momento lineal desde la delimitación opuesta a la original, de esta forma actuará una condición de contorno periódica (PBC) sobre cada partícula. En la figura 6 se muestra esquemáticamente en qué consisten las PBC en un sistema partículas en 2D, en este caso cada partícula podrá cruzar la región de simulación por cualquiera de los cuatro bordes, sin embargo, en 3D (como es nuestro caso) las partículas podrán cruzar la región por cualquiera de los 6 bordes de la celda tridimensional de estudio. Por el hecho de considerar PBC el

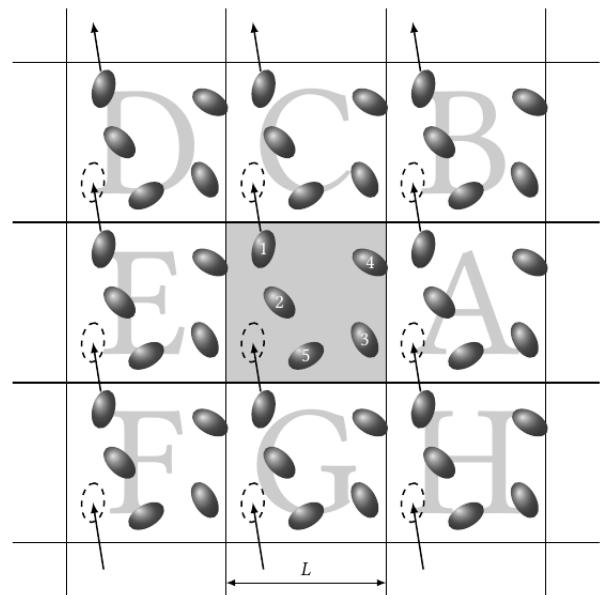


Fig. 6. Condiciones de contorno periódicas en un sistema 2D

cálculo de las fuerzas también requerirán ciertas correcciones, llamadas corrección de desplazamientos por imagen mínima las cuales consisten en que, al momentos de centrarnos en una

determinada partícula para computar la fuerza neta actuando sobre ella debido a la interacción con las otras partículas del entorno, deberemos centrar también la celda de estudio en la propia partícula y considerar las distancias interatómicas dentro de esta nueva celda y, notando que las distancias relativas serán siempre menores o iguales a la mitad de la celda cúbica se deberán corregir todas aquellas distancias que superen la mitad de la celda cúbica de simulación. En la figura 7 se observa esquemáticamente esta convención para corregir el desplazamiento según imagen mínima. El código desarrollado para estas correcciones de posiciones y desplazamientos corresponde a las funciones **pbc_correction**, **rel_pos_correction** y a la subrutina **position_correction** dentro del módulo **module_bd_lennard_jones.f90**.

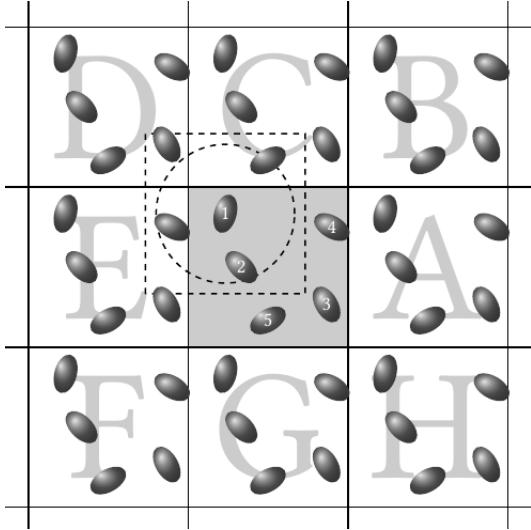


Fig. 7. Correcciones según imagen mínima en un sistema 2D

II-C. Integración de las ecuaciones de movimiento

La dinámica browniana al igual que la dinámica molecular es una dinámica realista y al igual que el método de monte carlo, es una dinámica configuracional. Las ecuaciones de movimiento son las correspondiente a las ecuaciones de Langevin posicional que, integrada de forma conveniente, se puede escribir en la forma:

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \frac{\vec{f}_i(t)}{3\pi\eta\sigma} \Delta t + \vec{r}_i^s(\Delta t) \quad (8)$$

donde $\vec{f}_i(t)$ es la fuerza de interacción que experimenta la partícula i debido a las otras partículas (esta fuerza es la derivada del potencial de interacción de Lennard-Jones) y $\vec{r}_i^s(\Delta t)$ es el desplazamiento browniano, el cual es una variable aleatoria de distribución gaussiana que cumple con:

$$\langle \vec{r}_i^s(\Delta t) \rangle = 0; \langle \vec{r}_i^s(\Delta t) \vec{r}_j^s(\Delta t) \rangle = 2D_0\Delta t\delta_{ij}\delta(\Delta t) \quad (9)$$

El código desarrollado para implementar estas ecuaciones de movimiento corresponde a la función **gaussian_rnd** y a la subrutina **evolution_bd** dentro del módulo **module_bd_lennard_jones.f90**.

II-D. Estado transitorio y estacionario

Para llevar a cabo la equilibración del sistema (estado transitorio) se implementaron $t_{eq} = 15000$ pasos de integración y una vez relajado el sistema, se implementaron $t_{run} = 50000$ pasos de integración. Además, para asegurarnos de obtener mediciones estadísticamente independientes, se consideraron

$t_{ens} = 10$ pasos temporales los cuales nos permiten fabricar el ensamble sobre el cual promediar los observables y obtener resultados físicamente más realistas (nos valemos de la ergodicidad del sistema y de que la dinámica browniana sigue un proceso markoviano).

II-E. Calculo de observables

II-E1. Energía potencial, presión osmótica

Conseguida la equilibración del sistema se graficaron las energías potencial (ver 8) y la presión instantánea del sistema en función del tiempo.

La presión se calculó teniendo en cuenta únicamente el término correspondiente al teorema del virial de la siguiente manera:

$$P = \frac{1}{dV} \left\langle \sum_{\langle ij \rangle} \vec{f}(\vec{r}_{ij}) \cdot \vec{r}_{ij} \right\rangle \quad (10)$$

$$d = 3; V \equiv L^3 = n_p / \rho$$

Entonces, las gráficas asociadas a cálculos de presión corresponden a la presión osmótica total del sistema. Ahora bien, adimensionalmente y trabajando un poco la ecuación tendremos una expresión para la presión adimensional, la cual fue implementada en las simulaciones

$$P^* = \frac{\rho}{3n_p} \left\langle \sum_{\langle ij \rangle} r_{ij}^* f_{r,ij}^* \right\rangle \quad (11)$$

Notemos que $\langle \dots \rangle$ se refiere al promedio temporal, que en nuestro caso corresponde a promediar cada $t_{ens} = 50$ pasos. Además, teniendo en cuenta que para la reducción del tiempo de CPU en las simulaciones se computan los cálculos relacionados a la fuerza de interacción hasta un radio de corte r_{cutoff} sin embargo, el potencial y la fuerza, más allá de este radio no es nula (lo cuál suponemos en los cálculos), entonces, se inducirá un error sistemático respecto a la solución real. En el caso particular de la presión la corrección a la misma se puede calcular de la siguiente manera:

$$\Delta P_{tail} = \frac{16}{3} \pi \rho^2 \epsilon \sigma^3 \left[\frac{2}{3} \left(\frac{\sigma}{r_{cutoff}} \right)^9 - \left(\frac{\sigma}{r_{cutoff}} \right)^3 \right] \quad (12)$$

y de forma adimensional tendremos la siguiente expresión;

$$\Delta P_{tail}^* = \frac{16}{3} \pi \rho^2 \left(\frac{1}{r_{cutoff}^*} \right)^3 \left[\frac{2}{3} \left(\frac{1}{r_{cutoff}^*} \right)^6 - 1 \right] \quad (13)$$

que, en el caso en que, $\rho = 0,8$ y $r_{cutoff}^* = 2,5$ tendremos un tail corrección de $\Delta P_{tail}^* = -0,6844$.

Por otro lado, en para obtener la corrección a la energía potencial podemos usar la siguiente expresión:

$$u_{tail} = \frac{8}{3} \pi \rho \epsilon \sigma^3 \left[\frac{1}{3} \left(\frac{\sigma}{r_{cutoff}} \right)^9 - \left(\frac{\sigma}{r_{cutoff}} \right)^3 \right] \quad (14)$$

y de forma adimensional tendremos la siguiente expresión;

$$u_{tail}^* = \frac{8}{3} \pi \rho \left(\frac{1}{r_{cutoff}^*} \right)^3 \left[\frac{1}{3} \left(\frac{1}{r_{cutoff}^*} \right)^6 - 1 \right] \quad (15)$$

que, en el caso en que, $\rho = 0,8$ y $r_{cutoff}^* = 2,5$ tendremos un tail corrección de $\Delta u_{tail}^* = -0,4198$.

Luego de de termalización del sistema también se calcularon los valores medios (primer momento) y errores de la la energía potencial y de la presión. Los cuales se muestran en líneas punteadas en la figura 8. Para el cálculo de los

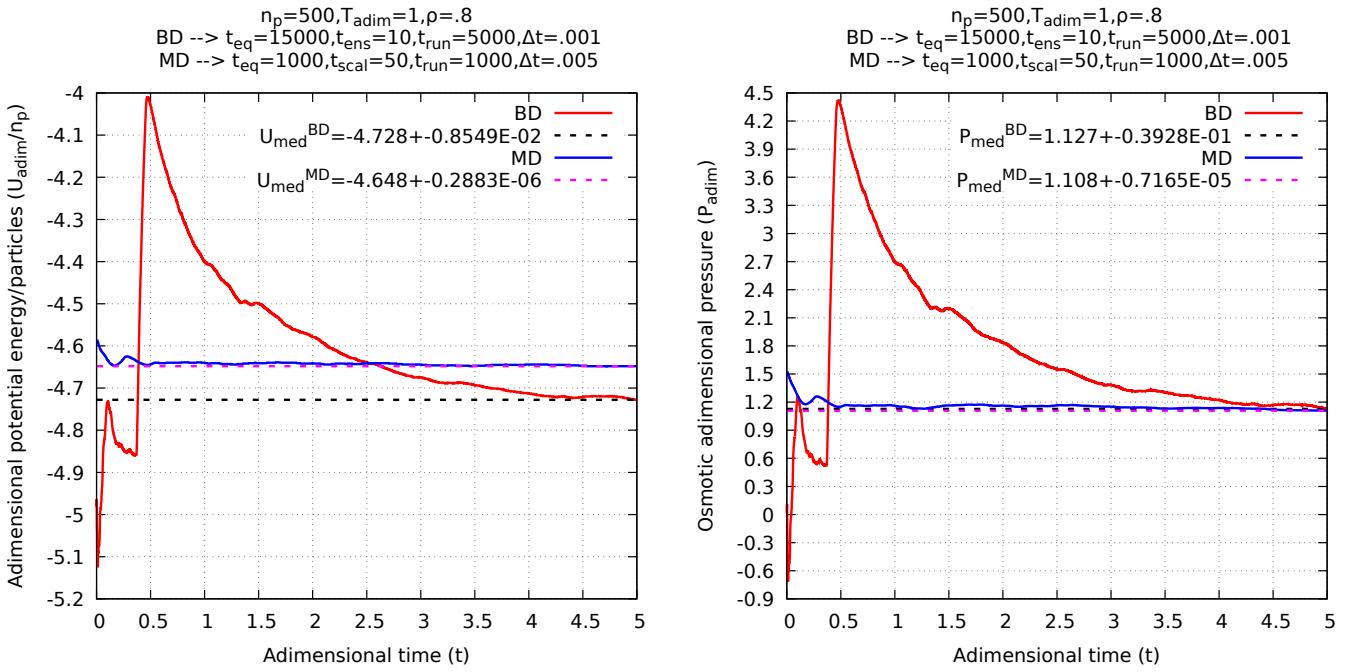


Fig. 8. Energía potencial y presión osmótica vs tiempo adimensional

valores promedios y errores nos valemos de las propiedades de ergodicidad del sistema el cual nos permite calcular valores de expectación de una única corrida (experimento) suficientemente larga (promedio temporal) y no muchos experimentos con configuraciones completamente descorrelacionadas entre si (promedio en el ensamble).

Por otro lado, en la figura 8 también se observa la comparación con simulaciones según dinámica molecular, notando que hay cierta concordancia en cuanto a la presión osmótica, sin embargo, se observan diferencias significativas en cuanto a los valores de energía potencial. Estas diferencias podrían deberse a que la dinámica browniana requiere ciertos valores de parámetros (número de partículas y pasos temporales de evolución) para los cuales sean válidas todas las hipótesis que nos llevan a la ecuación posicional de Langevin, además de una correcta equilibración del sistema (recordemos que al tratarse de una dinámica configuracional las simulaciones de BD son más costosas computacionalmente que la simulaciones de MD). Además, cabe mencionar que la dinámica browniana, al contrario de lo que pasa en dinámica molecular, la energía total no se conserva, pues estamos en presencia de un fenómeno disipativo en el cual la energía térmica se transforma en energía cinética y viceversa (tal como se explicó en la sección I-4), quizás por este fenómeno podrían ocurrir las diferencias entre los valores promedios de energía potencial a los que convergen las simulaciones de BD y MD, aunque, este razonamiento podría no ser del todo correcto pues, si consideramos la relación directa que existe entre la presión osmótica y la energía potencial y teniendo una concordancia muy buena entre BD y MD, esperaríamos que la energía potencial también posea estas similitudes.

Los parámetro utilizados en la simulación de MD son de $t_{eq} = 1000$ pasos de equilibración del sistema, con escaleo de velocidades cada $t_{scal} = 50$ pasos y $t_{run} = 1000$ pasos de corrida para el régimen estacionario sin escaleo de velocidades (promedios en ensamble microcanónico). El integrador utilizado para MD corresponde al velocity-verlet.

El código desarrollado para el cálculo de las energías y pre-

sión corresponden a las funciones **u_ij_total** para la energía potencial total y **osmotic_pressure** para la presión dentro del módulo **module_bd_lennard_jones.f90**. Y el código principal corresponde a **brownian_dynamic_lennard_jones_01**.

II-E2. Función de distribución radial

La función de distribución radial se define de las siguiente manera

$$g(r) = \frac{\langle \langle n_p \rangle \in [r; r + dr] \rangle_{\text{sistema}}}{\langle \langle n_p \rangle \in [r; r + dr] \rangle_{\text{gas ideal}}} = \frac{n_{\text{histogram}}(r)}{n_{\text{gas ideal}}(r)} \quad (16)$$

$$n_{\text{gas ideal}}(r) = \frac{4\pi\rho}{3} [(r + dr)^3 - r^3]$$

entonces la función de correlación de pares nos muestra cómo varía la densidad de partículas en función de la distancia interatómica medida respecto de alguna posición de referencia, es decir, es la probabilidad de encontrar una partícula a una distancia r desde una partícula de referencia, relativa a la misma probabilidad en un gas ideal. La normalización respecto al gas ideal es debido a que, para este los pares de partículas están totalmente descorrelacionados y la correlación aumenta a mediad que el sistema se solidifica, siendo máximo para un sólido e intermedio para un fluido. Cabe mencionar que para fabricar el histograma de distribución de partículas se utilizaron $n_{bins} = 1000$ bins. En la figura 9 podemos observar el resultado obtenido para la función de correlación espacial y su comparación con el resultado obtenido según una simulación de dinámica molecular. Vemos que existe mucha similitud en los resultados obtenidos, aunque hay ligeras variaciones en cuanto al pico máximo de la función de autocorrelación de pares, cuya distancia interatómica nos estaría especificando la distancia aproximada a los primeros vecinos.

El código principal para la obtención de la función de correlación espacial corresponde a **brownian_dynamic_lennard_jones_01** el cual utiliza la subrutina **radial_ditribution_function** dentro del módulo **module_bd_lennard_jones.f90**.

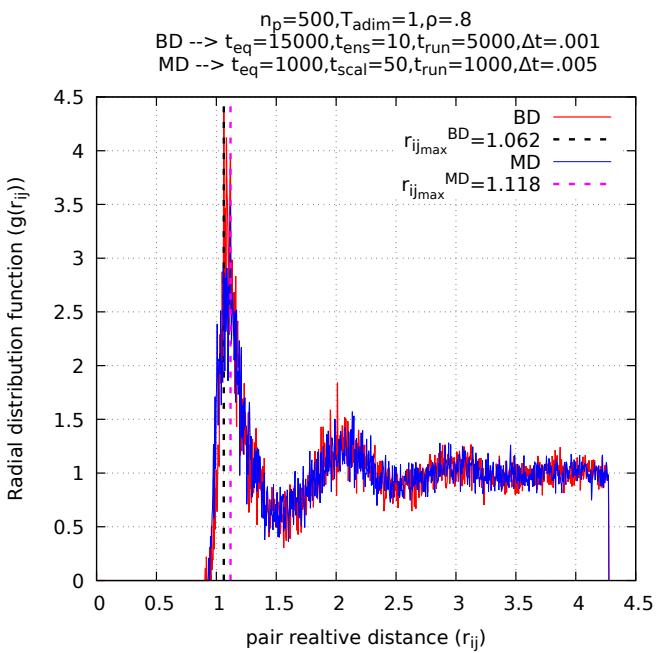


Fig. 9. función de correlación espacial vs distancia interatómica

II-E3. Desplazamiento cuadrático medio y coeficiente de difusión

La definición de desplazamiento cuadrático medio (MSD) y su relación con el coeficiente de difusión es la siguiente:

$$\left\langle |\vec{r}(t) - \vec{r}(0)|^2 \right\rangle = \frac{1}{n_p} \sum_{i=1}^{n_p} [\Delta \vec{r}_i(t)]^2 \cong 6Dt \quad (17)$$

donde D es el coeficiente de difusión, es importante mencionar que para el cálculo de MSD es necesario computar las autocorrelaciones de pares utilizando las posiciones sin corregir según PBC. En la figura 10 se pueden apreciar los resultados obtenidos, donde se puede notar la dependencia del coeficiente de difusión adimensionalizado D/D_0 con el tiempo de correlación adimensionalizado t/t_0 (donde $t_0 = \sigma^2/D_0$) está en acuerdo con la teoría, es decir, en líneas de trazo se puede notar un proceso difusivo para tiempos cortos y proceso subdifusivo para tiempos largo, pendiente menos pronunciada. Para las simulaciones se consideraron $\tau_{corr}^{max} = 100$ pasos máximos de correlación, almacenando $n_{max_{\tau_{corr}}} = 10$ valores distintos, cada vez, para el cálculo del MSD.

El código desarrollado para la obtención del desplazamiento cuadrático medio corresponde a la subrutina **mean_squared_displacement** dentro del código principal **brownian_dynamic_lennard_jones_01.f90**

II-F. Coeficiente de difusión a tiempos largos

Los parámetros elegidos para este caso fueron $n_p = 256$ partículas, un paso temporal de integración de $\Delta t = 0,001$, $t_{eq} = 15000$ pasos de termalización, $t_{run} = 15000$ pasos de corrida en régimen estacionario, $t_{ens} = 10$ pasos para promediar en ensamble, un radio de corte de interacciones de $r_{cutoff} = 2,5$, temperatura adimensional de referencia de $T_{adim} = 1,1$. Además, se varió la densidad entre los valores $\rho_{min} = 0,2$ y $\rho_{max} = 1,05$ (realizando 20 simulaciones con diferentes densidades). Los resultados obtenidos se muestran en la figura 11, donde se pueden observar cambios de curvatura en dos zonas con los siguientes rangos de densidades

$\Delta\rho_1 \sim [0,6921; 0,7816]$ y $\Delta\rho_2 \sim [0,2; 0,3]$ Ahora bien, al observar en escala logarítmica los ordenes de magnitud de estos saltos, podríamos considerar al rango $\Delta\rho_1$ como aquel en el que ocurre una transición de fase de primer orden (teóricamente, tendríamos un salto discontinuo en la primera derivada de la función respuesta) de líquido-sólido, para valores de densidad mayores a ρ_2 tendremos fase sólida y para valores de densidad menores a ρ_1 tendremos fase líquida. Por otro lado, podemos notar que en la zona de transición los errores estadísticos aumentan, pudiendo deberse este aspecto al aumento de las fluctuaciones del sistema.

Finalmente, en la figura 12 se muestra la energía potencial promedio y la presión osmótica promedio en función de la densidad adimensional. Es interesante observar el gráfico de la presión donde se identifica que, a partir de la densidad ρ_1 la presión aumenta, evidenciando una compresión del sistema de partículas, lo cual, como sabemos refleja la compresión debido a que el sistema se está transformando de un líquido a un sólido.

El código principal para el estudio de la transición de fase corresponde a **brownian_dynamic_lennard_jones_02.f90**

REFERENCIAS

- [1] E. Flenner and G. Szamel, Relaxation in a Glassy Binary Mixture: Comparison of the Mode-Coupling Theory to a Brownian Dynamics Simulation, *Phys. Rev. E* 72, 031508 (2005).
- [2] Stokes' Law, in Wikipedia (2022).
- [3] Teorema de fluctuación-disipación, in Wikipedia, la enciclopedia libre (2022).
- [4] G. Nägele, Brownian Dynamics simulations, Institut für Festkörperforschung Forschungszentrum Jülich GmbH
- [5] E. Braun, Un movimiento en zigzag (2011).
- [6] Lennard-Jones Potential, in Wikipedia (2022).

III. CÓDIGOS

Repositorio de GitHub

- <https://github.com/mendzmartin/fiscomp2022.git>

Repositorio GitHub del problema

- <https://github.com/mendzmartin/fiscomp2022/tree/main/lab06/prob01>

Códigos principales y Makefile

- https://github.com/mendzmartin/fiscomp2022/blob/main/lab06/prob01/code/brownian_dynamic_lennard_jones_01.f90

- https://github.com/mendzmartin/fiscomp2022/blob/main/lab06/prob01/code/brownian_dynamic_lennard_jones_02.f90

- <https://github.com/mendzmartin/fiscomp2022/blob/main/lab06/prob01/code/Makefile>

- **Vídeo de dinámica browniana**

- <https://github.com/mendzmartin/fiscomp2022/blob/main/lab06/prob01/plots/movie.mp4>

- **Módulo de funciones y subrutinas**

- https://github.com/mendzmartin/fiscomp2022/blob/main/modules/module_bd_lennard_jones.f90

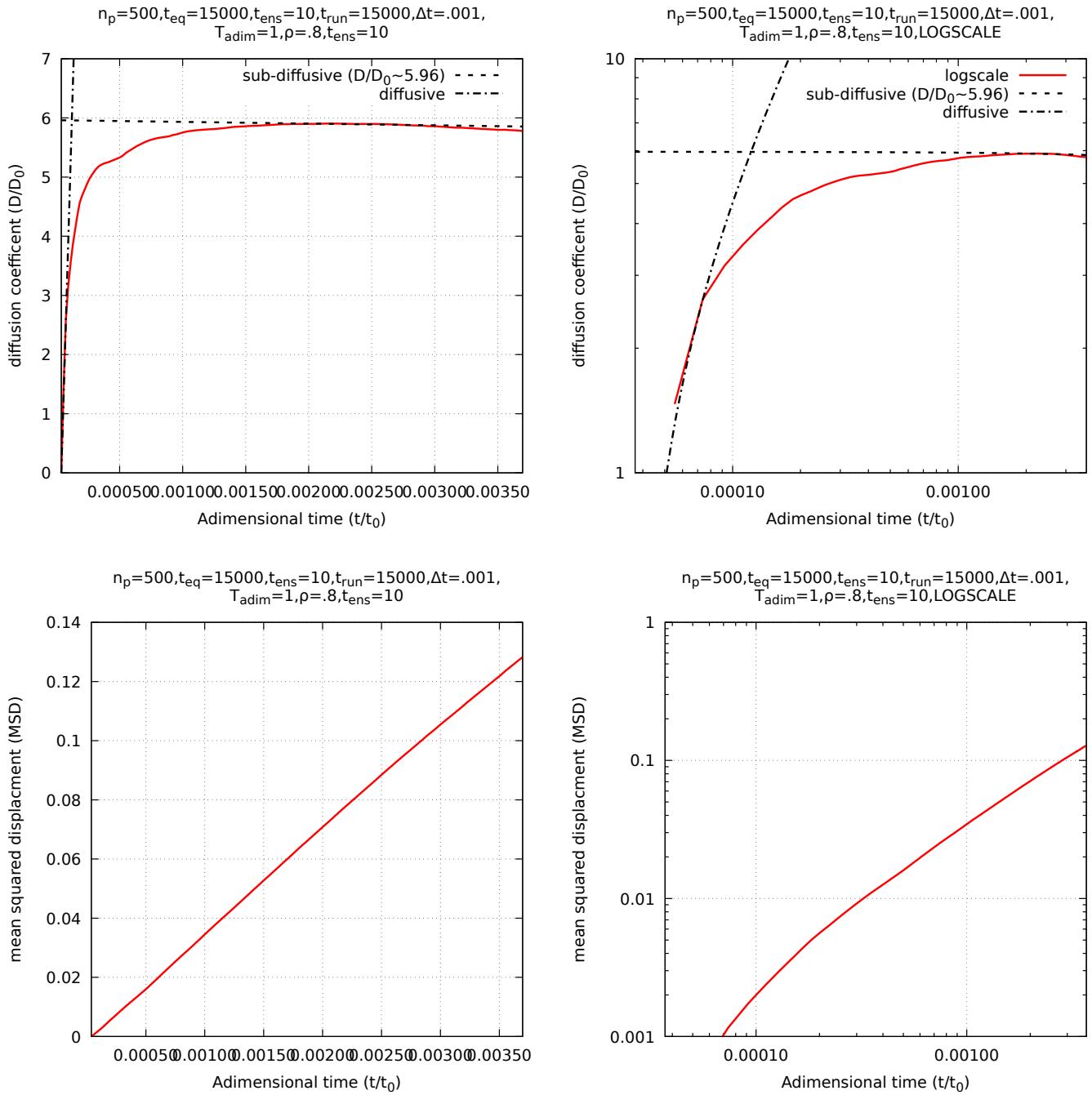


Fig. 10. Coeficiente de difusión y MSD adimensionales vs tiempo de correlación adimensional

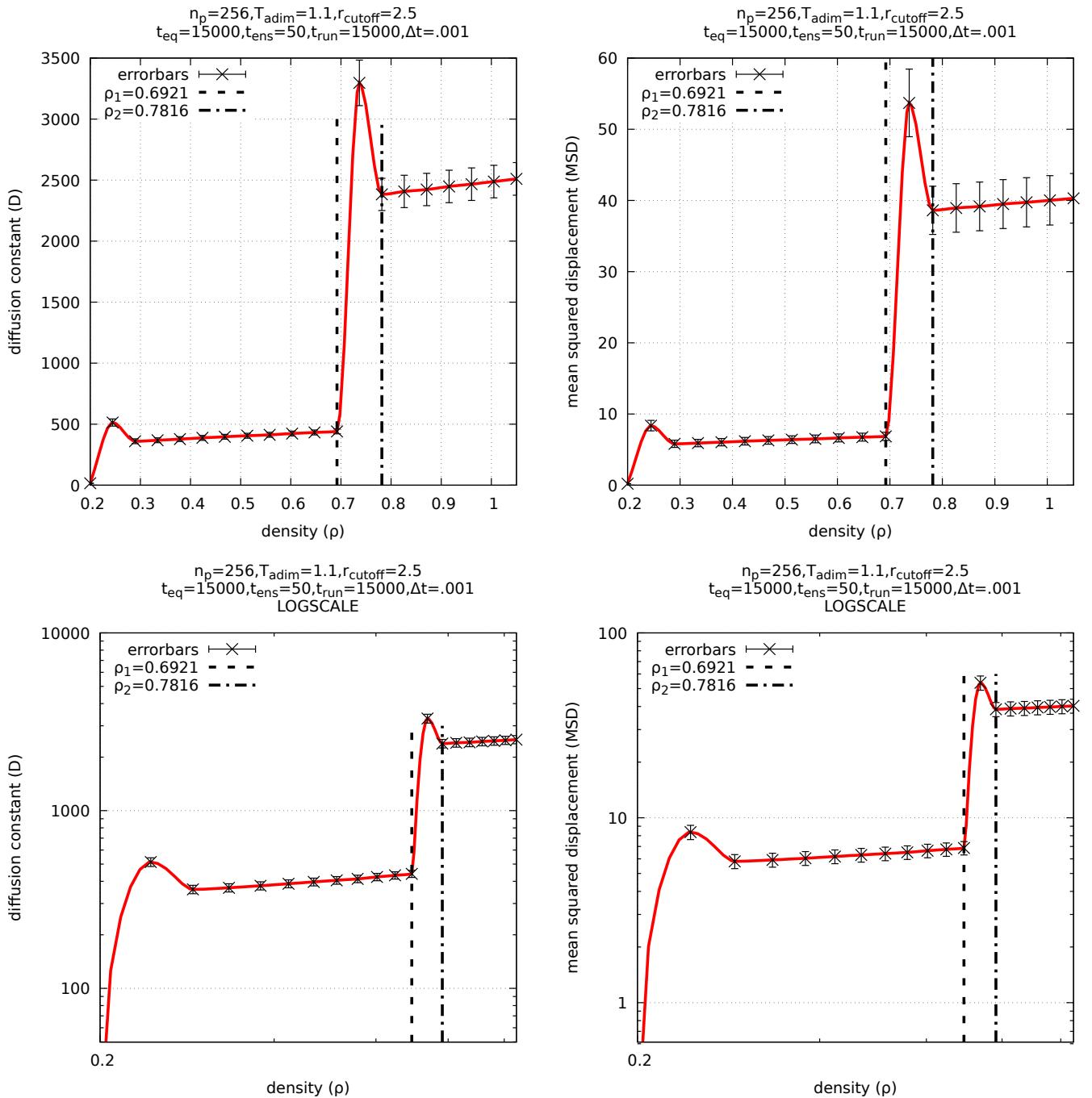


Fig. 11. Coeficiente de difusión y MSD vs densidad

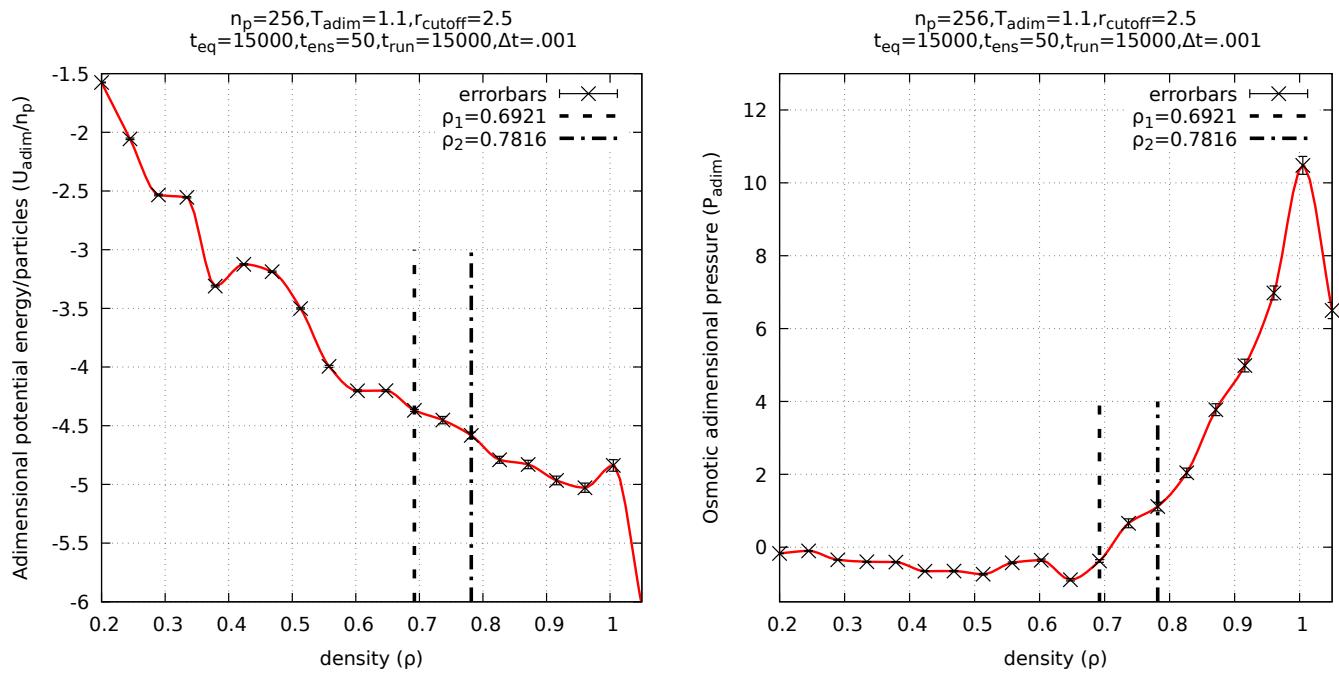


Fig. 12. Energía potencial y presión osmótica vs densidad

Fig. 13. Animación de dinámica browniana

Laboratorio 06 - Códigos

brownian_dynamic_lennard_jones_01.f90

```

1 ! make clean && make brownian_dynamic_lennard_jones_01.o && ./brownian_dynamic_lennard_jones_01.o
2 program brownian_dynamic_lennard_jones_01
3   use module_precision;use module_bd_lennard_jones
4   implicit none
5   ! VARIABLES y PARAMETROS GENERALES
6   integer(sp), parameter :: n_p=500_sp                                ! cantidad de partículas
7   real(dp),    parameter :: delta_time=0.001_dp                         ! paso temporal
8   integer(sp), parameter :: time_eq=15000_sp,&                           ! pasos de equilibración
9     time_run=15000_sp,&                                                 ! pasos de evolución en el estado estacionario
10    ensamble_step=10_sp                                                 ! pasos de evolución para promedio en ensamble
11   real(dp),    parameter :: T_adim_ref=1._dp                            ! temperatura de referencia adimensional
12   real(dp),    parameter :: density=0.8_dp                             ! densidad (partículas/volumen)
13   real(dp),    parameter :: r_cutoff=2.5_dp,mass=1._dp                  ! radio de corte de interacciones y masa
14   real(dp),    parameter :: dinamic_viscosity=2.87_dp
15   real(dp),    parameter :: pi=4._dp*atan(1._dp)
16   real(dp),    parameter :: diffusion_coeff=T_adim_ref*&
17     (1._dp/(3._dp*pi*dinamic_viscosity))
18   real(dp),    allocatable :: x_vector(:),y_vector(:),z_vector(:)      ! componentes de las posiciones/partícula
19   real(dp),    allocatable :: force_x(:,),force_y(:,),force_z(:,)
20   integer(sp)           :: i,j,istat,index                               ! loop index
21   real(dp)             :: time_end,time_start                          ! tiempos de CPU
22   ! VARIABLES LOGICAS PARA DECIDIR QUÉ ESCRIBIR
23   logical, parameter :: movie_switch=.false.,&                         ! escribir película con partículas en la caja
24     fcc_init_switch=.false.,&                                         ! escribir estructura fcc inicial
25     energies_switch=.true.,&                                         ! escribir energías en el estado estacionario
26     msd_switch=.true.,&                                              ! escribir coeficiente de difusión vs densidad
27     gr_switch=.true.                                                 ! escribir distribución de correlación espacial
28   ! VARIABLES PARA COMPUTAR ENERGÍA POTENCIAL
29   real(dp)              :: U_adim,time_press ! observables
30   real(dp)              :: U_med,var_U,err_U
31   real(dp)              :: s1_U,s2_U
32   ! VARIABLES PARA COMPUTAR PRESIÓN OSMÓTICA
33   real(dp)              :: press_med,var_press,err_press
34   real(dp)              :: s1_press,s2_press
35   ! VARIABLES PARA COMPUTAR DESPLAZAMIENTO CUADRÁTICO MEDIO
36   integer(sp), parameter :: tau_max_corr=100_sp                         ! pasos máximos de correlación
37   real(dp),    allocatable :: x_vector_noPBC(:,),y_vector_noPBC(:,),& ! componentes de la posición sin PBC
38     z_vector_noPBC(:)
39   real(dp),    allocatable :: wxx_matrix(:,,:),wyw_matrix(:,,:),&    ! matrices auxiliares para cálculo de msd
40     wzz_matrix(:,,:)
41   real(dp),    allocatable :: sum_wxx_vector(:,),sum_wyy_vector(:,),& ! vectores auxiliares para cálculo de msd
42     sum_wzz_vector(:)
43   integer(sp), allocatable :: counter_data(:)
44   real(dp)              :: msd,msd_med,var_msd,err_msd                ! desplazamiento cuadrático medio
45   real(dp)              :: s1_msd,s2_msd
46   integer(sp)           :: counter
47   ! VARIABLES PARA COMPUTAR AUTOCORRELACIÓN ESPACIAL
48   integer(sp), parameter :: n_bins=100_sp                                ! numero de bins
49   real(dp),    allocatable :: g(:)                                      ! radial distribution vector
50
51   call cpu_time(time_start)
52
53   allocate(x_vector(n_p),y_vector(n_p),z_vector(n_p))
54   x_vector(:)=0._dp;y_vector(:)=0._dp;z_vector(:)=0._dp
55   allocate(x_vector_noPBC(n_p),y_vector_noPBC(n_p),z_vector_noPBC(n_p))
56   x_vector_noPBC(:)=0._dp;y_vector_noPBC(:)=0._dp;z_vector_noPBC(:)=0._dp
57   allocate(force_x(n_p),force_y(n_p),force_z(n_p))
58   force_x(:)=0._dp;force_y(:)=0._dp;force_z(:)=0._dp
59   allocate(wxx_matrix(n_p,tau_max_corr),wyw_matrix(n_p,tau_max_corr),wzz_matrix(n_p,tau_max_corr))
60   allocate(sum_wxx_vector(tau_max_corr),sum_wyy_vector(tau_max_corr),sum_wzz_vector(tau_max_corr))
61   sum_wxx_vector(:)=0._dp;sum_wxx_vector(:)=0._dp;sum_wxx_vector(:)=0._dp
62   allocate(counter_data(tau_max_corr))
63   counter=0_sp;counter_data(:)=0_sp
64   allocate(g(n_bins))
65
66   ! generamos configuración inicial (FCC structure)
67   call initial_lattice_configuration(n_p,density,x_vector,y_vector,z_vector,2)
68   x_vector_noPBC(:)=x_vector(:);y_vector_noPBC(:)=y_vector(:);z_vector_noPBC(:)=z_vector(:)
69   ! computamos fuerzas en el tiempo inicial
70   call f_lj_total(x_vector,y_vector,z_vector,r_cutoff,n_p,density,force_x,force_y,force_z)
71
72   ! ESCRIBIMOS DATOS
73   if (fcc_init_switch.eqv..true.) then
74     open(90,file='../results/fcc.dat',status='replace',action='write',iostat=istat)
75     if (istat/=0) write(*,*) 'ERROR! istat(90file) = ',istat
76     write(90,"(2(A12,x),A12)") 'rx_fcc','ry_fcc','rz_fcc'
77     do i=1,n_p;write(90,"(2(E12.4,x),E12.4)") x_vector(i),y_vector(i),z_vector(i);end do;close(90)
78   end if
79   if (movie_switch.eqv..true.) then
80     index=10;call create_movie(index,x_vector,y_vector,z_vector,n_p)
81   end if

```

```

82 if (energies_switch.eqv..true.) then
83   !open(12,file='../../results/energies_vs_time.dat',status='replace',action='write',iostat=iostat)
84   open(12,file='../../results/improved_energies_vs_time.dat',status='replace',action='write',iostat=iostat)
85   write(*,*) 'istat(12file) = ',istat;write(12,"(2(A12,x),A12)") 'time','pot_ergy','press'
86 end if
87 if (msd_switch.eqv..true.) then
88   !open(13,file='../../results/msd_vs_time.dat',status='replace',action='write',iostat=iostat)
89   open(13,file='../../results/improved_msd_vs_time.dat',status='replace',action='write',iostat=iostat)
90   if (istat/=0) write(*,*) 'ERROR! istat(12file) = ',istat
91   write(13,"(A12,x,A12)") 'time','msd'
92   call mean_squared_displacement(n_p,x_vector,y_vector,z_vector,tau_max_corr,&
93     wxx_matrix,wyw_matrix,wzz_matrix,sum_wxx_vector,sum_wyy_vector,sum_wzz_vector,&
94     counter_data,counter)
95 end if
96
97 ! TRANSITORIO
98 do i=1,time_eq
99   write(*,*) i
100  call evolution_bd(n_p,x_vector,y_vector,z_vector,&
101    x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,&
102    delta_time,mass,r_cutoff,density,force_x,force_y,force_z,&
103    dinamic_viscosity,diffusion_coeff)
104 end do
105
106 ! ESTACIONARIO
107 if (energies_switch.eqv..true.) then
108   U_med=0._dp;s1_U=0._dp;s2_U=0._dp
109   press_med=0._dp;s1_press=0._dp;s2_press=0._dp
110 end if
111
112 ! seteamos variables
113 time=0._dp;i=0_sp
114
115 do j=1,time_run
116   write(*,*) time_eq+j
117
118   call evolution_bd(n_p,x_vector,y_vector,z_vector,&
119     x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,&
120     delta_time,mass,r_cutoff,density,force_x,force_y,force_z,&
121     dinamic_viscosity,diffusion_coeff)
122
123 ! ESCRIBIMOS DATOS
124 if (mod(j,ensamble_step)==0_sp) then
125   i=i+1_sp
126   time=real(i,dp)*delta_time
127   if (energies_switch.eqv..true.) then
128     U_adim=u_lj_total(n_p,x_vector,y_vector,z_vector,r_cutoff,density)
129     U_adim=U_adim*(1._dp/real(n_p,dp))
130     press=osmotic_pressure(n_p,density,mass,r_cutoff,x_vector,y_vector,z_vector)
131
132     s1_U=s1_U+U_adim;s2_U=s2_U+U_adim*U_adim
133     U_med=s1_U*(1._dp/real(i,dp))
134     var_U=(real(i,dp)*s2_U-s1_U*s1_U)*(1._dp/real(i*dp))
135
136     s1_press=s1_press+press;s2_press=s2_press+press*press
137     press_med=s1_press*(1._dp/real(i,dp))
138     var_press=(real(i,dp)*s2_press-s1_press*s1_press)*(1._dp/real(i*dp))
139
140     write(12,"(2(E12.4,x),E12.4)") time,U_med,press_med
141   end if
142   if ((movie_switch.eqv..true.).and.(mod(j,100)==0_sp)) then
143     index=index+1;call create_movie(index,x_vector,y_vector,z_vector,n_p)
144   end if
145   if (msd_switch.eqv..true.) then
146     call mean_squared_displacement(n_p,x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,tau_max_corr,&
147       wxx_matrix,wyw_matrix,wzz_matrix,sum_wxx_vector,sum_wyy_vector,sum_wzz_vector,&
148       counter_data,counter)
149   end if
150 end if
151 end do
152
153 if (energies_switch.eqv..true.) then
154   ! computamos errores en el último paso
155   !err_U=(var_U*0.25_dp)*(1._dp/real(i-1,dp))
156   !err_press=(var_press*0.25_dp)*(1._dp/real(i-1,dp))
157   err_U=sqrt(var_U*(1._dp/real(i-1,dp)))
158   err_press=sqrt(var_press*(1._dp/real(i-1,dp)))
159
160   write(*,'(A12,x,E12.4,x,E12.4)') 'U_med=',U_med,err_U
161   write(*,'(A12,x,E12.4,x,E12.4)') 'press_med=',press_med,err_press
162
163   close(12)
164 end if
165
166 if (msd_switch.eqv..true.) then
167   msd_med=0._dp;s1_msd=0._dp;s2_msd=0._dp
168   do i=1,tau_max_corr

```

```

169      if (counter_data(i)==0_sp) stop
170      time=real(i,dp)*delta_time
171      ! computamos msd
172      msd=(sum_wxx_vector(i)+sum_wyy_vector(i)+sum_wzz_vector(i))*&
173          (1._dp/real(counter_data(i),dp))*(1._dp/real(n_p,dp))
174      ! computamos observables, 1er y 2do momento, valores medios y varianzas
175      s1_msd=s1_msd+msd;s2_msd=s2_msd+msd*msd
176      msd_med=s1_msd*(1._dp/real(i,dp))
177      var_msd=(real(i,dp)*s2_msd-s1_msd*s1_msd)*(1._dp/real(i*i,dp))
178
179      write(13,"(E12.4,x,E12.4)") time,msd_med
180  end do
181      ! computamos errores en el último paso
182      !err_msd=(var_msd*0.25_dp)*(1._dp/real(tau_max_corr-1,dp))
183      err_msd=sqrt(var_msd*(1._dp/real(tau_max_corr-1,dp)))
184      write(*,'(A12,x,E12.4,x,E12.4)') 'msd_med=',msd_med,err_msd
185
186      close(13)
187  end if
188
189  if (gr_switch.eqv..true.) then
190      !call radial_ditribution_function('../results/radial_ditribution_function.dat',n_p,density,&
191      !                                         x_vector,y_vector,z_vector,n_bins,g)
192      call radial_ditribution_function('../results/improved_radial_ditribution_function.dat',n_p,density,&
193                                         x_vector,y_vector,z_vector,n_bins,g)
194  end if
195
196  deallocate(x_vector,y_vector,z_vector)
197  deallocate(x_vector_noPBC,y_vector_noPBC,z_vector_noPBC)
198  deallocate(force_x,force_y,force_z)
199  deallocate(wxx_matrix,wyy_matrix,wzz_matrix)
200  deallocate(sum_wxx_vector,sum_wyy_vector,sum_wzz_vector)
201  deallocate(counter_data)
202  deallocate(g)
203
204  call cpu_time(time_end)
205  write(*,*) 'elapsed time = ',time_end-time_start,['s']
206
207 end program brownian_dynamic_lennard_jones_01
208
209 ! subrutina para crear película de partículas en la caja
210 subroutine create_movie(index,x_vector,y_vector,z_vector,n_p)
211     use module_precision
212     implicit none
213     integer(sp), intent(in) :: index,n_p
214     real(dp),   intent(in) :: x_vector(n_p),y_vector(n_p),z_vector(n_p)
215     character(len=24)      :: file_name
216     character(len=2)       :: index_str
217     integer(sp)            :: i,istat
218     50 format(2(A12,x),A12);51 format(2(E12.4,x),E12.4)
219     write(index_str,'(I2)') index
220     file_name='../results/picture//trim(index_str)'].'dat'
221     open(52,file=file_name,status='replace',action='write',iostat=istat)
222     if (istat/=0) write(*,*) 'ERROR! istat(52file) = ',istat
223     write(52,50) 'rx_fcc','ry_fcc','rz_fcc'
224     do i=1,n_p;write(52,51) x_vector(i),y_vector(i),z_vector(i);end do;close(52)
225 end subroutine create_movie
226
227 ! subrutina para calcular el desplazamiento cuadrático medio
228 subroutine mean_squared_displacement(n_p,x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,tau_max_corr,&
229                                         wxx_matrix,wyy_matrix,wzz_matrix,sum_wxx_vector,sum_wyy_vector,sum_wzz_vector,&
230                                         counter_data,counter)
231     use module_precision
232
233     implicit none
234     integer(sp), intent(in)    :: n_p                                ! numero total de partículas
235     integer(sp), intent(in)    :: tau_max_corr                      ! pasos máximos de autocorrelación
236     real(dp),   intent(in)    :: x_vector_noPBC(n_p),y_vector_noPBC(n_p),&
237                                         z_vector_noPBC(n_p)           ! componentes del vector posición sin PBC
238     real(dp),   intent(inout) :: wxx_matrix(n_p,tau_max_corr),&
239                                         wyy_matrix(n_p,tau_max_corr),&
240                                         wzz_matrix(n_p,tau_max_corr)   ! matrices de acumulación
241     real(dp),   intent(inout) :: sum_wxx_vector(tau_max_corr),&
242                                         sum_wyy_vector(tau_max_corr),&
243                                         sum_wzz_vector(tau_max_corr)  ! vectores de sumas auxiliares
244     integer(sp), intent(inout) :: counter_data(tau_max_corr)        ! contador de datos
245     integer(sp), intent(inout) :: counter                           ! contador de entradas
246
247     integer(sp)             :: i,j
248     integer(sp)             :: tau_corr_0,tau_corr_t ! tiempos de correlación
249     ! NOTA: CONDICIÓN QUE SE DEBE CUMPLIR
250     ! nmax_tau_corr_0 < (time_eq+time_run)/(ensamble_step*tau_max_corr)
251     integer(sp), parameter :: nmax_tau_corr_0=10_sp ! maximo número de tau_corr_0 que almacenamos
252
253     counter=counter+1          ! numero de veces que entro a la subrutina
254     tau_corr_0=mod(counter-1,tau_max_corr)+1 ! tiempo de correlación actual tau_corr_0={1,2,...,tau_max_corr}
255

```

```

256 ! guardamos cíclicamente los últimos tau_max_corr valores
257 ! de las componentes x,y,z de cada partícula
258 do i=1,n_p
259   wxx_matrix(i,tau_corr_0)=x_vector_noPBC(i)
260   wyy_matrix(i,tau_corr_0)=y_vector_noPBC(i)
261   wzz_matrix(i,tau_corr_0)=z_vector_noPBC(i)
262 end do
263
264 if ((mod(counter,nmax_tau_corr_0)==0).and.(counter>tau_max_corr)) then
265   do j=1,tau_max_corr
266     tau_corr_t=mod(counter-j,tau_max_corr)+1
267     do i=1,n_p
268       sum_wxx_vector(j)=sum_wxx_vector(j)+(wxx_matrix(i,tau_corr_0)-wxx_matrix(i,tau_corr_t))*&
269                               (wxz_matrix(i,tau_corr_0)-wxz_matrix(i,tau_corr_t))
270       sum_wyy_vector(j)=sum_wyy_vector(j)+(wyx_matrix(i,tau_corr_0)-wyx_matrix(i,tau_corr_t))*&
271                               (wyx_matrix(i,tau_corr_0)-wyx_matrix(i,tau_corr_t))
272       sum_wzz_vector(j)=sum_wzz_vector(j)+(wzx_matrix(i,tau_corr_0)-wzx_matrix(i,tau_corr_t))*&
273                               (wzx_matrix(i,tau_corr_0)-wzx_matrix(i,tau_corr_t))
274     end do
275     ! actualizamos el contador de datos para cada tiempo de correlación
276     counter_data(j)=counter_data(j)+1_sp
277   end do
278 end if
279 end subroutine mean_squared_displacement

```

brownian_dynamic_lennard_jones_02.f90

```

1 ! make clean && make brownian_dynamic_lennard_jones_02.o && ./brownian_dynamic_lennard_jones_02.o
2 program brownian_dynamic_lennard_jones_02
3   use module_precision;use module_bd_lennard_jones
4   implicit none
5   ! VARIABLES y PARAMETROS GENERALES
6   integer(sp), parameter :: n_p=256_sp                                ! cantidad de partículas
7   real(dp), parameter :: delta_time=0.001_dp                           ! paso temporal
8   integer(sp), parameter :: time_eq=15000_sp,&                         ! pasos de equilibración
9     time_run=15000_sp,&                                                 ! pasos de evolución en el estado estacionario
10    ensamble_step=10_sp                                                 ! pasos de evolución para promedio en ensamble
11   real(dp), parameter :: T_adim_ref=1.1_dp                            ! temperatura de referencia adimensional
12   real(dp), parameter :: r_cutoff=2.5_dp,mass=1._dp                   ! radio de corte de interacciones y masa
13   real(dp), parameter :: dinamic_viscosity=2.87_dp
14   real(dp), parameter :: pi=4._dp*atan(1._dp)
15   real(dp), parameter :: diffusion_coeff=T_adim_ref*&
16     (1._dp/(3._dp*pi*dinamic_viscosity))                           ! componentes de las posiciones/partícula
17   real(dp), allocatable :: x_vector(:),y_vector(:),z_vector(:)        ! componentes de la fuerza/partícula
18   real(dp), allocatable :: force_x(:),force_y(:),force_z(:)
19   integer(sp) :: i,j,k,istat                                         ! loop index
20   real(dp) :: time_end,time_start                                     ! tiempos de CPU
21   ! VARIABLES PARA REALIZAR BARRIDO DE DENSIDADES
22   real(dp) :: density                                                 ! densidad (partículas/volumen)
23   real(dp), parameter :: density_min=0.2_dp,density_max=1.05_dp ! rango de densidades
24   integer(sp), parameter :: n_density=20_sp                           ! cantidad de densidades simuladas
25   real(dp), parameter :: step_density=abs(density_max-density_min)*& ! paso de variación de densidades
26     (1._dp/real(n_density-1,dp))
27   ! VARIABLES LOGICAS PARA DECIDIR QUÉ ESCRIBIR
28   logical, parameter :: energies_switch=.true.,& ! escribir energías en el estado estacionario
29     msd_switch=.true.          ! escribir coeficiente de difusión vs densidad
30   ! VARIABLES PARA COMPUTAR ENERGÍA POTENCIAL
31   real(dp) :: U_adim,time_press ! observables
32   real(dp) :: U_med,var_U,err_U
33   real(dp) :: s1_U,s2_U
34   ! VARIABLES PARA COMPUTAR PRESIÓN OSMÓTICA
35   real(dp) :: press_med,var_press,err_press
36   real(dp) :: s1_press,s2_press
37   ! VARIABLES PARA COMPUTAR DESPLAZAMIENTO CUADRÁTICO MEDIO
38   integer(sp), parameter :: tau_max_corr=100_sp                      ! pasos máximos de correlación
39   real(dp), allocatable :: x_vector_noPBC(:,:),y_vector_noPBC(:,:),& ! componentes de la posición sin PBC
40     z_vector_noPBC(:)
41   real(dp), allocatable :: wxx_matrix(:,:,),wyx_matrix(:,:,),&      ! matrices auxiliares para cálculo de msd
42     wzx_matrix(:,:)
43   real(dp), allocatable :: sum_wxx_vector(:),sum_wyy_vector(:),&      ! vectores auxiliares para cálculo de msd
44     sum_wzz_vector(:)
45   integer(sp), allocatable :: counter_data(:)
46   real(dp) :: msd,msd_med,var_msd,err_msd                          ! desplazamiento cuadrático medio
47   real(dp) :: s1_msd,s2_msd
48   integer(sp) :: counter
49   ! VARIABLES PARA COMPUTAR COEFICIENTE DE DIFUSIÓN
50   real(dp) :: D,D_med,var_D,err_D
51   real(dp) :: s1_D,s2_D
52
53   call cpu_time(time_start)
54
55   allocate(x_vector(n_p),y_vector(n_p),z_vector(n_p))
56   allocate(x_vector_noPBC(n_p),y_vector_noPBC(n_p),z_vector_noPBC(n_p))
57   allocate(force_x(n_p),force_y(n_p),force_z(n_p))
58   allocate(wxx_matrix(n_p,tau_max_corr),wyx_matrix(n_p,tau_max_corr),wzx_matrix(n_p,tau_max_corr))
59   allocate(sum_wxx_vector(tau_max_corr),sum_wyy_vector(tau_max_corr),sum_wzz_vector(tau_max_corr))

```

```

60 allocate(counter_data(tau_max_corr))
61
62 ! ESCRIBIMOS DATOS
63 if (energies_switch.eqv..true.) then
64   open(12,file='../../results/energies_vs_density.dat',status='replace',action='write',iostat=istat)
65   write(*,*) 'istat(12file) = ',istat;write(12,"(4(A12,x),A12)") 'density','U_med','err_U','press_med','err_press'
66 end if
67 if (msd_switch.eqv..true.) then
68   open(13,file='../../results/msd_vs_density.dat',status='replace',action='write',iostat=istat)
69   if (istat/=0) write(*,*) 'ERROR! istat(12file) = ',istat
70   write(13,"(4(A12,x),A12)") 'density','msd_med','err_msd','D/D0_med','err_D/D0'
71 end if
72
73 do k=1,n_density
74   print*, k
75
76   ! seteo de variables y parámetros
77   x_vector(:)=0._dp;y_vector(:)=0._dp;z_vector(:)=0._dp
78   x_vector_noPBC(:)=0._dp;y_vector_noPBC(:)=0._dp;z_vector_noPBC(:)=0._dp
79   force_x(:)=0._dp;force_y(:)=0._dp;force_z(:)=0._dp
80   sum_wxx_vector(:)=0._dp;sum_wyy_vector(:)=0._dp;sum_wzz_vector(:)=0._dp
81   counter=0_sp;counter_data(:)=0_sp
82
83   ! definimos denisty en el rango [density_min:density_max]
84   density=density_min+step_density*real(k-1,dp)
85
86   ! generamos configuración inicial (FCC structure)
87   call initial_lattice_configuration(n_p,density,x_vector,y_vector,z_vector,2)
88   x_vector_noPBC(:)=x_vector(:);y_vector_noPBC(:)=y_vector(:);z_vector_noPBC(:)=z_vector(:)
89   ! computamos fuerzas en el tiempo inicial
90   call f_lj_total(x_vector,y_vector,z_vector,r_cutoff,n_p,density,force_x,force_y,force_z)
91
92   if (msd_switch.eqv..true.) then
93     call mean_squared_displacement(n_p,x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,tau_max_corr,&
94       wxx_matrix,wyy_matrix,wzz_matrix,sum_wxx_vector,sum_wyy_vector,sum_wzz_vector,&
95       counter_data,counter)
96 end if
97
98   ! TRANSITORIO
99   do i=1,time_eq
100     call evolution_bd(n_p,x_vector,y_vector,z_vector,&
101       x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,&
102       delta_time,mass,r_cutoff,density,force_x,force_y,force_z,&
103       dinamic_viscosity,diffusion_coeff)
104   end do
105
106   ! ESTACIONARIO
107   if (energies_switch.eqv..true.) then
108     U_med=0._dp;s1_U=0._dp;s2_U=0._dp
109     press_med=0._dp;s1_press=0._dp;s2_press=0._dp
110   end if
111
112   ! seteamos variables
113   time=0._dp;i=0_sp
114
115   do j=1,time_run
116     call evolution_bd(n_p,x_vector,y_vector,z_vector,&
117       x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,&
118       delta_time,mass,r_cutoff,density,force_x,force_y,force_z,&
119       dinamic_viscosity,diffusion_coeff)
120
121   ! ESCRIBIMOS DATOS
122   if (mod(j,ensamble_step)==0_sp) then
123     i=i+1_sp
124     time=real(i,dp)*delta_time
125     if (energies_switch.eqv..true.) then
126       U_adim=u_lj_total(n_p,x_vector,y_vector,z_vector,r_cutoff,density)
127       U_adim=U_adim*(1._dp/real(n_p,dp))
128       press=osmotic_pressure(n_p,density,mass,r_cutoff,x_vector,y_vector,z_vector)
129
130       s1_U=s1_U+U_adim;s2_U=s2_U+U_adim*U_adim
131       U_med=s1_U*(1._dp/real(i,dp))
132       var_U=(real(i,dp)*s2_U-s1_U*s1_U)*(1._dp/real(i*i,dp))
133
134       s1_press=s1_press+press;s2_press=s2_press+press*press
135       press_med=s1_press*(1._dp/real(i,dp))
136       var_press=(real(i,dp)*s2_press-s1_press*s1_press)*(1._dp/real(i*i,dp))
137     end if
138     if (msd_switch.eqv..true.) then
139       call mean_squared_displacement(n_p,x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,tau_max_corr,&
140         wxx_matrix,wyy_matrix,wzz_matrix,sum_wxx_vector,sum_wyy_vector,sum_wzz_vector,&
141         counter_data,counter)
142     end if
143   end if
144 end do
145
146 if (energies_switch.eqv..true.) then

```

```

147      ! computamos errores en el último paso
148      !err_U=(var_U*0.25_dp)*(1._dp/real(i-1,dp))
149      !err_press=(var_press*0.25_dp)*(1._dp/real(i-1,dp))
150      err_U=sqrt(var_U*(1._dp/real(i-1,dp)))
151      err_press=sqrt(var_press*(1._dp/real(i-1,dp)))
152
153      write(12,"(4(E12.4,x),E12.4)") density,U_med,err_U,press_med,err_press
154  end if
155
156  if (msd_switch.eqv..true.) then
157    msd_med=0._dp;s1_msd=0._dp;s2_msd=0._dp
158    D_med=0._dp;s1_D=0._dp;s2_D=0._dp
159    do i=1,tau_max_corr
160      if (counter_data(i)==0_sp) stop
161      time=real(i,dp)*delta_time
162      ! computamos msd
163      msd=(sum_wxx_vector(i)+sum_wyy_vector(i)+sum_wzz_vector(i))*&
164        (1._dp/real(counter_data(i),dp))*(1._dp/real(n_p,dp))
165      D=msd*(1._dp/(6._dp*time))*(1._dp/diffusion_coeff)
166      ! computamos observables, 1er y 2do momento, valores medios y varianzas
167      s1_msd=s1_msd+msd;s2_msd=s2_msd+msd*msd
168      msd_med=s1_msd*(1._dp/real(i,dp))
169      var_msd=(real(i,dp)*s2_msd-s1_msd*s1_msd)*(1._dp/real(i*i,dp))
170
171      s1_D=s1_D+D;s2_D=s2_D+D*D
172      D_med=s1_D*(1._dp/real(i,dp))
173      var_D=(real(i,dp)*s2_D-s1_D*s1_D)*(1._dp/real(i*i,dp))
174    end do
175    ! computamos errores en el último paso
176    !err_msd=(var_msd*0.25_dp)*(1._dp/real(tau_max_corr-1,dp))
177    err_msd=sqrt(var_msd*(1._dp/real(tau_max_corr-1,dp)))
178    err_D=sqrt(var_D*(1._dp/real(tau_max_corr-1,dp)))
179    write(13,"(4(E12.4,x),E12.4)") density,msd_med,err_msd,D_med,err_D
180  end if
181 end do
182
183 if (energies_switch.eqv..true.) close(12)
184 if (msd_switch.eqv..true.) close(13)
185
186 deallocate(x_vector,y_vector,z_vector)
187 deallocate(x_vector_noPBC,y_vector_noPBC,z_vector_noPBC)
188 deallocate(force_x,force_y,force_z)
189 deallocate(wxx_matrix,wyy_matrix,wzz_matrix)
190 deallocate(sum_wxx_vector,sum_wyy_vector,sum_wzz_vector)
191 deallocate(counter_data)
192
193 call cpu_time(time_end)
194 write(*,*) 'elapsed time = ',time_end-time_start,'[s]'
195
196 end program brownian_dynamic_lennard_jones_02
197
198 ! subrutina para calcular el desplazamiento cuadrático medio
199 subroutine mean_squared_displacement(n_p,x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,tau_max_corr,&
200   wxx_matrix,wyy_matrix,wzz_matrix,sum_wxx_vector,sum_wyy_vector,sum_wzz_vector,&
201   counter_data,counter)
202   use module_precision
203
204   implicit none
205   integer(sp), intent(in) :: n_p                               ! numero total de partículas
206   integer(sp), intent(in) :: tau_max_corr                      ! pasos máximos de autocorrelación
207   real(dp),   intent(in) :: x_vector_noPBC(n_p),y_vector_noPBC(n_p),&
208                           z_vector_noPBC(n_p)                                ! componentes del vector posición sin PBC
209   real(dp),   intent(inout) :: wxx_matrix(n_p,tau_max_corr),&
210                           wyy_matrix(n_p,tau_max_corr),&
211                           wzz_matrix(n_p,tau_max_corr)                         ! matrices de acumulación
212   real(dp),   intent(inout) :: sum_wxx_vector(tau_max_corr),&
213                           sum_wyy_vector(tau_max_corr),&
214                           sum_wzz_vector(tau_max_corr)                          ! vectores de sumas auxiliares
215   integer(sp), intent(inout) :: counter_data(tau_max_corr)       ! contador de datos
216   integer(sp), intent(inout) :: counter                         ! contador de entradas
217
218   integer(sp)          :: i,j
219   integer(sp)          :: tau_corr_0,tau_corr_t ! tiempos de correlación
220   ! NOTA: CONDICIÓN QUE SE DEBE CUMPLIR
221   ! nmax_tau_corr_0 < (time_eq+time_run)/(ensamble_step*tau_max_corr)
222   integer(sp), parameter :: nmax_tau_corr_0=5_sp ! maximo número de tau_corr_0 que almacenamos
223
224   counter=counter+1           ! numero de veces que entro a la subrutina
225   tau_corr_0=mod(counter-1,tau_max_corr)+1 ! tiempo de correlación actual tau_corr_0={1,2,...,tau_max_corr}
226
227   ! guardamos cíclicamente los últimos tau_max_corr valores
228   ! de las componentes x,y,z de cada partícula
229   do i=1,n_p
230     wxx_matrix(i,tau_corr_0)=x_vector_noPBC(i)
231     wyy_matrix(i,tau_corr_0)=y_vector_noPBC(i)
232     wzz_matrix(i,tau_corr_0)=z_vector_noPBC(i)
233   end do

```

```

234
235 if ((mod(counter,nmax_tau_corr_0)==0).and.(counter>tau_max_corr)) then
236   do j=1,tau_max_corr
237     tau_corr_t=mod(counter-j,tau_max_corr)+1
238     do i=1,n_p
239       sum_wxx_vector(j)=sum_wxx_vector(j)+(wxx_matrix(i,tau_corr_0)-wxx_matrix(i,tau_corr_t))*&
240                               (wxz_matrix(i,tau_corr_0)-wxz_matrix(i,tau_corr_t))
241       sum_wyy_vector(j)=sum_wyy_vector(j)+(wyx_matrix(i,tau_corr_0)-wyx_matrix(i,tau_corr_t))*&
242                               (wyx_matrix(i,tau_corr_0)-wyx_matrix(i,tau_corr_t))
243       sum_wzz_vector(j)=sum_wzz_vector(j)+(wzx_matrix(i,tau_corr_0)-wzx_matrix(i,tau_corr_t))*&
244                               (wzx_matrix(i,tau_corr_0)-wzx_matrix(i,tau_corr_t))
245     end do
246     ! actualizamos el contador de datos para cada tiempo de correlación
247     counter_data(j)=counter_data(j)+1_sp
248   end do
249 end if
250 end subroutine mean_squared_displacement

```

molecular_dynamic_lennard_jones_02.f90

```

1  ! make clean && make molecular_dynamic_lennard_jones_02.o && ./molecular_dynamic_lennard_jones_02.o
2  program molecular_dynamic_lennard_jones_02
3    use module_precision;use module_md_lennard_jones
4    implicit none
5    integer(sp), parameter :: n_p=500_sp                                ! cantidad de partículas
6    real(dp), parameter :: delta_time=0.005_dp                         ! paso temporal
7    integer(sp), parameter :: time_eq=1000_sp,time_scal=50_sp,&          ! pasos de equilibración y de escaleo de veloc.
8                  time_run=1000_sp                                         ! pasos de evolución en el estado estacionario
9    real(dp), parameter :: T_adim_ref=1._dp                            ! temperatura de referencia adimensional
10   real(dp), parameter :: density=0.8_dp                             ! densidad (partículas/volumen)
11   real(dp), parameter :: r_cutoff=2.5_dp,mass=1._dp                 ! radio de corte de interacciones y masa
12   real(dp), allocatable :: x_vector(:,y_vector(:,z_vector(:))        ! componentes de las posiciones/partícula
13   real(dp), allocatable :: vx_vector(:,vy_vector(:,vz_vector(:))      ! componentes de la velocidad/partícula
14   real(dp), allocatable :: force_x(:,force_y(:,force_z(:))           ! componentes de la fuerza/partícula
15   integer(sp) :: i,j,istat,index                                      ! loop index
16   real(dp) :: U_adim,U_med,var_U,err_U                             ! energías
17   real(dp) :: Ec_adim,Ec_med,var_Ec,err_Ec                         ! energías totales
18   real(dp) :: Etot_adim,Etot_med,var_Etot,err_Etot                  ! energías totales
19   real(dp) :: press,press_med,var_press,err_press                   ! presiones
20   real(dp) :: T_adim,T_med,var_T,err_T                             ! temperaturas
21   real(dp) :: s1_U,s2_U                                           ! momentos lineales
22   real(dp) :: s1_Ec,s2_Ec                                         ! energías cinéticas
23   real(dp) :: s1_Etot,s2_Etot                                       ! energías totales
24   real(dp) :: s1_press,s2_press                                     ! presiones
25   real(dp) :: s1_T,s2_T                                           ! temperaturas
26   real(dp) :: vx_mc,vy_mc,vz_mc                                     ! componentes de la velocidad del centro de masas
27   real(dp) :: time_end,time_start                                    ! tiempos de CPU
28
29 call cpu_time(time_start)
30
31 allocate(x_vector(n_p),y_vector(n_p),z_vector(n_p))
32 x_vector(:)=0._dp;y_vector(:)=0._dp;z_vector(:)=0._dp
33
34 ! generamos configuración inicial (FCC structure)
35 call initial_lattice_configuration(n_p,density,x_vector,y_vector,z_vector,2)
36
37 allocate(vx_vector(n_p),vy_vector(n_p),vz_vector(n_p))
38 vx_vector(:)=0._dp;vy_vector(:)=0._dp;vz_vector(:)=0._dp
39 call md_initial_parameters(n_p,x_vector,y_vector,z_vector,&
40 vx_vector,vy_vector,vz_vector,T_adim_ref,delta_time,density,mass)
41
42 ! computamos fuerzas en el tiempo inicial
43 allocate(force_x(n_p),force_y(n_p),force_z(n_p))
44 force_x(:)=0._dp;force_y(:)=0._dp;force_z(:)=0._dp
45 call f_lj_total(x_vector,y_vector,z_vector,r_cutoff,n_p,density,force_x,force_y,force_z)
46
47
48 index=0
49 ! TRANSITORIO
50 do i=1,time_eq
51   index=index+1
52   write(*,*) 'paso temporal =',index,' de',time_eq+time_run
53   if (mod(i,time_scal)==0_sp) call rescaling_velocities(n_p,vx_vector,vy_vector,vz_vector,T_adim_ref,mass)
54   call velocity_verlet(n_p,x_vector,y_vector,z_vector,&
55                         vx_vector,vy_vector,vz_vector,delta_time,mass,r_cutoff,density,force_x,force_y,force_z)
56   ! velocity center of mass to zero
57   vx_mc=sum(vx_vector(:)*(1._dp/real(n_p,dp));vx_vector(:)=(vx_vector(:)-vx_mc)
58   vy_mc=sum(vy_vector(:)*(1._dp/real(n_p,dp));vy_vector(:)=(vy_vector(:)-vy_mc)
59   vz_mc=sum(vz_vector(:)*(1._dp/real(n_p,dp));vz_vector(:)=(vz_vector(:)-vz_mc)
60   T_adim=temperature(n_p,mass,vx_vector,vy_vector,vz_vector)
61 end do
62
63 write(*,*) 'termino el transitorio'
64
65 ! ESTACIONARIO
66 open(10,file='..../results/md_fluctuations.dat',status='replace',action='write',iostat=istat)

```

```

67 if (istat/=0) write(*,*) 'ERROR! istat(10file) = ',istat
68 11 format(9(E12.4,x),E12.4);12 format(9(A12,x),A12)
69 write(10,12) 'U_med','err_U','Ec_med','err_Ec','Etot_med','err_Etot','press_med','err_press','T_med','err_T'
70
71 open(20,file='..../results/md_fluctuations_vs_time.dat',status='replace',action='write',iostat=istat)
72 if (istat/=0) write(*,*) 'ERROR! istat(11file) = ',istat
73 21 format(10(E12.4,x),E12.4);22 format(10(A12,x),A12)
74 write(20,22) 'time','U_med','var_U','Ec_med','var_Ec','Etot_med','var_Etot','press_med','var_press','T_med','var_T'
75
76 U_med=0._dp
77 Ec_med=0._dp
78 Etot_med=0._dp
79 press_med=0._dp
80 T_med=0._dp
81
82 s1_U=0._dp;s2_U=0._dp
83 s1_Ec=0._dp;s2_Ec=0._dp
84 s1_Etot=0._dp;s2_Etot=0._dp
85 s1_press=0._dp;s2_press=0._dp
86 s1_T=0._dp;s2_T=0._dp
87
88 do i=1,time_run
89   index=index+1
90   write(*,*) 'paso temporal =',index,' de',time_eq+time_run
91   call velocity_verlet(n_p,x_vector,y_vector,z_vector,&
92     vx_vector,vy_vector,vz_vector,delta_time,mass,r_cutoff,density,force_x,force_y,force_z)
93
94   U_adim=u_lj_total(n_p,x_vector,y_vector,z_vector,r_cutoff,density)
95   Ec_adim=kinetic_ergy_total(n_p,vx_vector,vy_vector,vz_vector,mass)
96   Etot_adim=(U_adim+Ec_adim)*(1._dp/real(n_p,dp))
97   press=pressure(n_p,density,mass,r_cutoff,x_vector,y_vector,z_vector,&
98     vx_vector,vy_vector,vz_vector)
99   T_adim=temperature(n_p,mass,vx_vector,vy_vector,vz_vector)
100
101 ! computamos 1er y 2do momento
102 s1_U=s1_U+U_adim*(1._dp/real(n_p,dp));s2_U=s2_U+U_adim*U_adim*(1._dp/real(n_p*n_p,dp))
103 s1_Ec=s1_Ec+Ec_adim*(1._dp/real(n_p,dp));s2_Ec=s2_Ec+Ec_adim*Ec_adim*(1._dp/real(n_p*n_p,dp))
104 s1_Etot=s1_Etot+Etot_adim;s2_Etot=s2_Etot+Etot_adim*Etot_adim
105 s1_press=s1_press+press;s2_press=s2_press+press*press
106 s1_T=s1_T+T_adim;s2_T=s2_T+T_adim*T_adim
107
108 ! computamos valores medios (mejor a mayor paso evolucionado)
109 U_med=s1_U*(1._dp/real(i,dp))
110 Ec_med=s1_Ec*(1._dp/real(i,dp))
111 Etot_med=s1_Etot*(1._dp/real(i,dp))
112 press_med=s1_press*(1._dp/real(i,dp))
113 T_med=s1_T*(1._dp/real(i,dp))
114
115 ! computamos varianzas (mejor a mayor paso evolucionado)
116 var_U=(real(i,dp)*s2_U-s1_U*s1_U)*(1._dp/real(i*i,dp))
117 var_Ec=(real(i,dp)*s2_Ec-s1_Ec*s1_Ec)*(1._dp/real(i*i,dp))
118 var_Etot=(real(i,dp)*s2_Etot-s1_Etot*s1_Etot)*(1._dp/real(i*i,dp))
119 var_press=(real(i,dp)*s2_press-s1_press*s1_press)*(1._dp/real(i*i,dp))
120 var_T=(real(i,dp)*s2_T-s1_T*s1_T)*(1._dp/real(i*i,dp))
121
122 write(20,21) delta_time*real(i,dp),U_med,var_U,Ec_med,var_Ec,Etot_med,var_Etot,press_med,var_press,T_med,var_T
123 end do
124 close(20)
125
126 write(*,*) 'termino el estacionario'
127
128 ! computamos errores en el último paso
129 err_U=(var_U*0.25_dp)*(1._dp/real(time_eq-1,dp))
130 err_Ec=(var_Ec*0.25_dp)*(1._dp/real(time_eq-1,dp))
131 err_Etot=(var_Etot*0.25_dp)*(1._dp/real(time_eq-1,dp))
132 err_press=(var_press*0.25_dp)*(1._dp/real(time_eq-1,dp))
133 err_T=(var_T*0.25_dp)*(1._dp/real(time_eq-1,dp))
134
135 write(10,11) U_med,err_U,Ec_med,err_Ec,Etot_med,err_Etot,press_med,err_press,T_med,err_T
136 close(10)
137
138 deallocate(x_vector,y_vector,z_vector)
139 deallocate(vx_vector,vy_vector,vz_vector)
140 deallocate(force_x,force_y,force_z)
141
142 call cpu_time(time_end)
143 write(*,*) 'elapsed time = ',time_end-time_start,['s']
144 end program molecular_dynamic_lennard_jones_02

```

Laboratorio 06 - Módulo principal

module_bd_lennard_jones.f90

```

1 ! module of borwnian dynamic to lennard jones potential
2 ! gfortran -c module_precision.f90 module_mt19937.f90 module_bd_lennard_jones.f90

```

```

3  module module_bd_lennard_jones
4      use module_precision;use module_mt19937, only: sgrnd, grnd
5      implicit none
6      contains
7
8      ! FUNCIONES
9
10     ! función para obtener numero random con distribución gaussiana
11     function gaussian_rnd(sigma,media)
12         real(dp), intent(in) :: sigma,media
13         integer(sp)          :: iset=0_sp,seed,seed_val(8)
14         real(dp), parameter :: pi=4._dp*atan(1._dp)
15         real(dp)              :: gaussian_rnd2,gaussian_rnd,nrand_01,nrand_02
16         save iset,gaussian_rnd2
17
18         if (iset==0_sp) then
19             call date_and_time(values=seed_val)
20             seed=seed_val(8)*seed_val(7)*seed_val(6)+seed_val(5);call sgrnd(seed)
21             nrand_01=real(grnd(),dp);nrand_02=real(grnd(),dp)
22             gaussian_rnd=sqrt(-2._dp*log(nrand_01))*sin(2._dp*pi*nrand_02)
23             gaussian_rnd=sigma*gaussian_rnd+media
24             gaussian_rnd2=sqrt(-2._dp*log(nrand_01))*cos(2._dp*pi*nrand_02)
25             iset=1_sp
26         else
27             gaussian_rnd=gaussian_rnd2
28             gaussian_rnd=sigma*gaussian_rnd+media
29             iset=0_sp
30         end if
31     end function gaussian_rnd
32
33     ! FUNCION PARA CALCULAR EL FACTOR DE ESTRUCTURA ESTÁTICO (PARAMETRO DE ORDEN CRYSTALINO)
34     function static_structure_factor(n_p,density,x_vector,y_vector,z_vector)
35         integer(sp), intent(in) :: n_p                                ! numero de partículas
36         real(dp),   intent(in) :: density                            ! densidad de partículas
37         real(dp),   intent(in) :: x_vector(n_p),y_vector(n_p),z_vector(n_p) ! coordenadas del vector posición
38         real(dp)              :: static_structure_factor           ! función de estructura
39         real(dp),   parameter :: pi=4._dp*atan(1._dp)
40         real(dp)              :: a                                  ! parámetro de red
41         real(dp)              :: points_unitcells                ! número de partículas por celda unidad
42         real(dp)              :: factor1,factor2                 ! componentes del vector de onda
43         real(dp)              :: kx,ky,kz
44         integer(sp)           :: i
45
46         ! valido únicamente para una red FCC
47         points_unitcells=4._dp
48         a=(points_unitcells*(1._dp/density))**((1._dp/3._dp))
49         factor1=0._dp;factor2=0._dp
50         do i=1,n_p
51             ! cambiar en caso de tener otro vector de onda
52             kx=-2._dp*pi*(1._dp/a)*x_vector(i)
53             ky=2._dp*pi*(1._dp/a)*y_vector(i)
54             kz=-2._dp*pi*(1._dp/a)*z_vector(i)
55             factor1=factor1+cos(kx+ky+kz)
56             factor2=factor2+sin(kx+ky+kz)
57         end do
58         factor1=factor1*factor1;factor2=factor2*factor2
59         static_structure_factor=(1._dp/real(n_p*n_p,dp))*(factor1+factor2)
60     end function static_structure_factor
61
62     function osmotic_pressure(n_p,density,mass,r_cutoff,x_vector,y_vector,z_vector)
63         integer(sp), intent(in) :: n_p
64         real(dp),   intent(in) :: r_cutoff,density,mass
65         real(dp),   intent(in) :: x_vector(n_p),y_vector(n_p),z_vector(n_p)
66         real(dp)              :: osmotic_pressure,rij_pow02,result,force_indiv
67         integer(sp)           :: i,j
68         result=0._dp
69         do j=2,n_p
70             do i=1,j-1
71                 rij_pow02=rel_pos_correction(x_vector(i),y_vector(i),z_vector(i),&
72                     x_vector(j),y_vector(j),z_vector(j),n_p,density)
73                 if (rij_pow02<=r_cutoff*r_cutoff) then
74                     force_indiv=f_lj_individual(rij_pow02)
75                 else; force_indiv=0._dp; end if
76                 result=result+force_indiv*rij_pow02
77             end do
78         end do
79         osmotic_pressure=density*(1._dp/(3._dp*real(n_p,dp)))*result
80     end function osmotic_pressure
81
82     ! compute individual lennard jones potential (simple truncation)
83     function u_lj_individual(r12_pow02)
84         real(dp), intent(in) :: r12_pow02 ! distancia adimensional entre pares de partículas
85         real(dp)              :: r12_pow06 ! potencias de la distancia relativa
86         real(dp)              :: u_lj_individual ! adimensional lennard jones potential
87         integer(sp)           :: i
88         ! calculamos distancia relativa corregida según PBC
89         r12_pow06=1._dp

```

```

90      do i=1,3;r12_pow06=r12_pow06*r12_pow02;end do ! (r12)^6
91      u_lj_individual=4._dp*(1._dp/r12_pow06)*((1._dp/r12_pow06)-1._dp)
92  end function u_lj_individual
93 ! compute total lennard jones potential (TRUNCADO Y DESPLAZADO)
94 function u_lj_total(n_p,x_vector,y_vector,z_vector,r_cutoff,density)
95     integer(sp), intent(in) :: n_p
96     real(dp), intent(in) :: x_vector(n_p),y_vector(n_p),z_vector(n_p)
97     real(dp), intent(in) :: r_cutoff,density
98     real(dp)             :: u_lj_total,u_indiv,rij_pow02
99     integer(sp)           :: i,j
100    u_lj_total=0._dp
101    do j=2,n_p
102        do i=1,j-1
103            ! calculamos distancia relativa corregida según PBC
104            rij_pow02=rel_pos_correction(x_vector(i),y_vector(i),z_vector(i),&
105                                         x_vector(j),y_vector(j),z_vector(j),n_p,density)
106            if (rij_pow02<=r_cutoff*r_cutoff) then
107                u_indiv=u_lj_individual(rij_pow02)-u_lj_individual(r_cutoff*r_cutoff)
108            else; u_indiv=0._dp; end if
109            u_lj_total=u_lj_total+u_indiv
110        end do
111    end do
112 end function u_lj_total
113
114 ! calculo de la fuerza individual (par de partículas)
115 function f_lj_individual(r12_pow02)
116     real(dp), intent(in) :: r12_pow02    ! distancia adimensional entre pares de partículas
117     real(dp)              :: r12_pow06    ! factores potencia
118     real(dp)              :: f_lj_individual    ! adimensional individual lennard jones force
119     integer(sp)           :: i
120     r12_pow06=1._dp
121     do i=1,3;r12_pow06=r12_pow06*r12_pow02;end do ! (r12)^6
122     f_lj_individual=24._dp*(1._dp/(r12_pow02*r12_pow06))*(2._dp*(1._dp/r12_pow06)-1._dp)
123 end function f_lj_individual
124 ! calculo de la componente xi de la fuerza total (TRUNCADO Y DESPLAZADO)
125 subroutine f_lj_total(x_vector,y_vector,z_vector,r_cutoff,n_p,density,&
126                      fx_lj_total_vector,fy_lj_total_vector,fz_lj_total_vector)
127     integer(sp), intent(in)   :: n_p
128     real(dp),  intent(in)    :: x_vector(n_p),y_vector(n_p),z_vector(n_p)
129     real(dp),  intent(in)    :: r_cutoff,density
130     real(dp),  intent(inout) :: fx_lj_total_vector(n_p),fy_lj_total_vector(n_p),&
131                               fz_lj_total_vector(n_p) ! vector de fuerzas netas
132     real(dp)                :: rij_pow02
133     real(dp)                :: force_indiv ! fuerza neta acuando en una determinada partícula
134     integer(sp)              :: i,j
135     real(dp)                :: dx,dy,dz,L
136     fx_lj_total_vector(:)=0._dp;fy_lj_total_vector(:)=0._dp;fz_lj_total_vector(:)=0._dp
137     L=(real(n_p,dp)*(1._dp/density))**2*(1._dp/3._dp)
138     do j=2,n_p
139         do i=1,j-1
140             ! calculamos distancia relativa corregida según PBC
141             rij_pow02=rel_pos_correction(x_vector(i),y_vector(i),z_vector(i),&
142                                         x_vector(j),y_vector(j),z_vector(j),n_p,density)
143
144             if (rij_pow02<=r_cutoff*r_cutoff) then
145                 force_indiv=f_lj_individual(rij_pow02)
146             else; force_indiv=0._dp;end if
147
148             dx=pbc_correction((x_vector(i)-x_vector(j)),n_p,density)
149             dy=pbc_correction((y_vector(i)-y_vector(j)),n_p,density)
150             dz=pbc_correction((z_vector(i)-z_vector(j)),n_p,density)
151             ! COMPONENTES DE LA FUERZA
152             fx_lj_total_vector(i)=fx_lj_total_vector(i)+force_indiv*dx
153             fx_lj_total_vector(j)=fx_lj_total_vector(j)-force_indiv*dx
154             fy_lj_total_vector(i)=fy_lj_total_vector(i)+force_indiv*dy
155             fy_lj_total_vector(j)=fy_lj_total_vector(j)-force_indiv*dy
156             fz_lj_total_vector(i)=fz_lj_total_vector(i)+force_indiv*dz
157             fz_lj_total_vector(j)=fz_lj_total_vector(j)-force_indiv*dz
158         end do
159     end do
160 end subroutine f_lj_total
161
162 ! corrección de las posiciones relativas (PBC)
163 function rel_pos_correction(x1,y1,z1,x2,y2,z2,n_p,density)
164     integer(sp), intent(in) :: n_p          ! numero total de partículas
165     real(dp),  intent(in)  :: density       ! densidad de partículas
166     real(dp),  intent(in)  :: x1,y1,z1,x2,y2,z2 ! coordenadas del par de partículas
167     real(dp)              :: rel_pos_correction ! posición relativa corregida (PBC) al cuadrado
168     real(dp)              :: dx,dy,dz          ! diferencia de posiciones segun coordenadas
169     dx=pbc_correction((x1-x2),n_p,density)
170     dy=pbc_correction((y1-y2),n_p,density)
171     dz=pbc_correction((z1-z2),n_p,density)
172     rel_pos_correction=dx*dx+dy*dy+dz*dz
173 end function rel_pos_correction
174
175 function pbc_correction(x,n_p,density)
176     integer(sp), intent(in) :: n_p          ! numero total de partículas

```

```

177     real(dp), intent(in) :: density ! densidad de partículas
178     real(dp), intent(in) :: x         ! variable a corregir
179     real(dp)             :: pbc_correction
180     L=(real(n_p,dp)*(1._dp/density))**(1._dp/3._dp)
181     pbc_correction=(x-L*anint(x*(1._dp/L),dp))
182 end function pbc_correction
183
184 ! SUBRUTINAS
185
186 ! corrección de las posiciones (PBC)
187 subroutine position_correction(n_p,density,x,y,z)
188     integer(sp), intent(in) :: n_p      ! numero total de partículas
189     real(dp), intent(in) :: density    ! densidad de partículas
190     real(dp), intent(out) :: x,y,z     ! posiciones
191     x=pbc_correction(x,n_p,density)
192     y=pbc_correction(y,n_p,density)
193     z=pbc_correction(z,n_p,density)
194 end subroutine position_correction
195
196 ! Subroutine to set up sc and fcc lattice
197 subroutine initial_lattice_configuration(n_p,density,x_vector,y_vector,z_vector,type_structure)
198     integer(sp), intent(in) :: n_p      ! numero total de partículas
199     integer(sp), intent(in) :: type_structure ! 1 by sc lattice/ 2 by fcc lattice
200     real(dp), intent(in) :: density    ! densidad de partículas
201     real(dp), intent(out) :: x_vector(n_p),&
202                           y_vector(n_p),&
203                           z_vector(n_p)
204     integer(sp)           :: i,j,k,index2,index ! loop index
205     real(dp)             :: L          ! longitud macroscópica por dimensión
206     real(dp)             :: a          ! parámetro de red
207     real(dp)             :: n_unitcells ! numero de celdas unidad por dimensión
208     real(dp)             :: points_unitcells ! número de partículas por celda unidad
209     real(dp), allocatable :: aux_matrix(:,:)
210     L=(real(n_p,dp)*(1._dp/density))**(1._dp/3._dp)
211     select case (type_structure)
212         case(1) ! sc laticce
213             points_unitcells=1._dp
214             n_unitcells=anint((real(n_p,dp)*(1._dp/points_unitcells))**(1._dp/3._dp),dp)
215             a=L*(1._dp/(n_unitcells-1._dp))!a=(4*(1._dp/density))**(1._dp/3._dp)
216             ! cargamos vectores de coordenadas
217             index=0
218             do i=1,int(n_unitcells,sp)
219                 do j=1,int(n_unitcells,sp)
220                     do k=1,int(n_unitcells,sp)
221                         index=index+1
222                         ! CENTRAMOS LA CELDA EN EL RANGO [-L/2:L/2]
223                         x_vector(index)=real(i-1,dp)*a-0.5_dp*L
224                         y_vector(index)=real(j-1,dp)*a-0.5_dp*L
225                         z_vector(index)=real(k-1,dp)*a-0.5_dp*L
226                     end do
227                 end do
228             end do
229         case(2) ! fcc laticce
230             points_unitcells=4._dp
231             n_unitcells=anint((real(n_p,dp)*(1._dp/points_unitcells))**(1._dp/3._dp),dp)
232             a=L*(1._dp/(n_unitcells))!a=(4*(1._dp/density))**(1._dp/3._dp)
233             ! cargamos matriz con vectores primitivos (specific for FCC structure)
234             allocate(aux_matrix(4,3));aux_matrix(:,:)=a*0.5_dp
235             aux_matrix(1,:)=0.0_dp;aux_matrix(2,:)=0.0_dp
236             aux_matrix(3,:)=0.0_dp;aux_matrix(4,:)=0.0_dp
237             ! cargamos vectores de coordenadas
238             index=0
239             do i=1,int(n_unitcells,sp)
240                 do j=1,int(n_unitcells,sp)
241                     do k=1,int(n_unitcells,sp)
242                         index2=i,4
243                         index=index+1
244                         ! CENTRAMOS LA CELDA EN EL RANGO [-L/2:L/2]
245                         x_vector(index)=(aux_matrix(index2,1)+real(i-1,dp)*a)-0.5_dp*L
246                         y_vector(index)=(aux_matrix(index2,2)+real(j-1,dp)*a)-0.5_dp*L
247                         z_vector(index)=(aux_matrix(index2,3)+real(k-1,dp)*a)-0.5_dp*L
248                     end do
249                 end do
250             end do
251         case(3) ! fcc laticce (other method)
252             points_unitcells=4._dp
253             n_unitcells=anint((real(n_p,dp)*(1._dp/points_unitcells))**(1._dp/3._dp),dp)
254             a=(points_unitcells*(1._dp/density))**(1._dp/3._dp)
255             index=0
256             do i=0,int(n_unitcells,sp)-1
257                 do j=0,int(n_unitcells,sp)-1
258                     do k=0,int(n_unitcells,sp)-1
259                         index=index+1
260                         x_vector(index)=a*real(i,dp)-L*0.5_dp
261                         y_vector(index)=a*real(j,dp)-L*0.5_dp
262
263

```

```

264         z_vector(index)=a*real(k,dp)-L*0.5_dp
265         index=index+1
266         x_vector(index)=a*(real(i,dp)+0.5_dp)-L*0.5_dp
267         y_vector(index)=a*(real(j,dp)+0.5_dp)-L*0.5_dp
268         z_vector(index)=a*real(k,dp)-L*0.5_dp
269         index=index+1
270         x_vector(index)=a*real(i,dp)-L*0.5_dp
271         y_vector(index)=a*(real(j,dp)+0.5_dp)-L*0.5_dp
272         z_vector(index)=a*(real(k,dp)+0.5_dp)-L*0.5_dp
273         index=index+1
274         x_vector(index)=a*(real(i,dp)+0.5_dp)-L*0.5_dp
275         y_vector(index)=a*real(j,dp)-L*0.5_dp
276         z_vector(index)=a*(real(k,dp)+0.5_dp)-L*0.5_dp
277     enddo
278   enddo
279 end select
280 end subroutine initial_lattice_configuration
281
283 ! SUBRUTINA DE INTEGRACIÓN DE ECUACIONES DE MOVIMIENTO
284 subroutine evolution_bd(n_p,x_vector,y_vector,z_vector,&
285   x_vector_noPBC,y_vector_noPBC,z_vector_noPBC,&
286   delta_time,mass,r_cutoff,density,force_x,force_y,force_z,&
287   dinamic_viscosity,diffusion_coeff)
288
289   integer(sp), intent(in) :: n_p
290   real(dp), intent(in) :: delta_time,mass,r_cutoff,density,dinamic_viscosity,diffusion_coeff
291   real(dp), intent(out) :: x_vector(n_p),y_vector(n_p),z_vector(n_p)
292   real(dp), intent(out) :: x_vector_noPBC(n_p),y_vector_noPBC(n_p),&
293   z_vector_noPBC(n_p) ! componentes del vector posición sin PBC
294   real(dp), intent(out) :: force_x(n_p),force_y(n_p),force_z(n_p)
295   integer(sp) :: i
296   real(dp) :: factor,brownian_position
297   real(dp), parameter :: pi=4._dp*atan(1._dp)
298
299   factor=3._dp*pi*dinamic_viscosity
300
301 ! COMPONENTES DE LA POSICION A TIEMPO EVOLUCIONADO (con y sin pBC)
302 do i=1,n_p
303   brownian_position=gaussian_rnd(sqrt(2._dp*diffusion_coeff*delta_time),0._dp)
304   x_vector(i)=x_vector(i)+force_x(i)*(1._dp/factor)*delta_time+brownian_position
305   x_vector_noPBC(i)=x_vector_noPBC(i)+force_x(i)*(1._dp/factor)*delta_time+brownian_position
306
307   brownian_position=gaussian_rnd(sqrt(2._dp*diffusion_coeff*delta_time),0._dp)
308   y_vector(i)=y_vector(i)+force_y(i)*(1._dp/factor)*delta_time+brownian_position
309   y_vector_noPBC(i)=y_vector_noPBC(i)+force_x(i)*(1._dp/factor)*delta_time+brownian_position
310
311   brownian_position=gaussian_rnd(sqrt(2._dp*diffusion_coeff*delta_time),0._dp)
312   z_vector(i)=z_vector(i)+force_z(i)*(1._dp/factor)*delta_time+brownian_position
313   z_vector_noPBC(i)=z_vector_noPBC(i)+force_x(i)*(1._dp/factor)*delta_time+brownian_position
314
315   call position_correction(n_p,density,x_vector(i),y_vector(i),z_vector(i))
316 end do
317
318 ! ACTUALIZAMOS COMPONENTES DE LA FUERZA A TIEMPO EVOLUCIONADO
319 call f_lj_total(x_vector,y_vector,z_vector,r_cutoff,n_p,density,force_x,force_y,force_z)
320 end subroutine evolution_bd
321
322 ! SUBRUTINA PARA CALCULAR LA FUNCION DE DISTRIBUCIÓN RADIAL (FUNCION DE CORRELACIÓN)
323 subroutine radial_distribution_function(file_name,n_p,density,x_vector,y_vector,z_vector,n_bins,g)
324   character(len=*), intent(in) :: file_name ! nombre del archivo de datos
325   integer(sp), intent(in) :: n_p,n_bins ! numero de partículas y numero total de
326   bins
326   real(dp), intent(in) :: x_vector(n_p),y_vector(n_p),z_vector(n_p) ! coordenadas del vector posición
327   real(dp), intent(in) :: density ! densidad de partículas
328   real(dp), intent(out) :: g(n_bins) ! funcion distribución
329
330   integer(sp) :: ngr ! contador de partículas
331   real(dp) :: rij_pow02,rij ! distancia relativa entre par de partículas
332   real(dp) :: L ! longitud macroscópica
333   real(dp) :: step_bins ! tamaño de bins
334   real(dp) :: volume ! volumen de partículas
335   real(dp) :: nid ! numero de partículas del gas ideal en volumen
336   integer(sp) :: i,j,index,istat ! loop variables
337
338   ! initialization
339   L=(real(n_p,dp)*(1._dp/density))**1._dp/3._dp
340   step_bins=L/(2*n_bins)
341   g(:)=0._dp
342
343   ! sample
344   ngr=0_sp;ngr=ngr+1_sp
345   do j=2,n_p;do i=1,j-1
346     ! relative distances (pow 2) with pbc correction
347     rij_pow02=rel_pos_correction(x_vector(i),y_vector(i),z_vector(i),&
348     x_vector(j),y_vector(j),z_vector(j),n_p,density)
349     ! relative distances

```

```
350      rij=sqrt(rij_pow02)
351      if (rij<=L/2) then ! only within half the box length
352          index=int(rij/step_bins)
353          g(index)=g(index)+2._dp ! contribution for particle i and j
354      end if
355  end do,end do
356
357 ! result initialization (determine g(rij))
358 open(100,file=file_name,status='replace',action='write',iostat=istat)
359   if (istat/=0) write(*,*) 'ERROR! istat(11file) = ',istat
360   101 format(E12.4,x,E12.4);102 format(A12,x,A12)
361   write(100,102) 'rij','g(rij)'
362   do i=1,n_bins
363       rij=step_bins*(i+0.5_dp)
364       ! volumen between bin i+1 and bin i
365       volume=(real(i+1,dp)**3-real(i,dp)**3)*(step_bins**3)
366       ! number of ideal gas particles in volume
367       nid=(1._dp/3._dp)*16._dp*atan(1._dp)*volume*density
368       ! normalize g(rij)
369       g(i)=g(i)/(real(ngr*n_p,dp)*nid)
370       write(100,101) rij,g(i)
371   end do
372   close(100)
373 end subroutine radial_ditribution_function
374
375 end module module_bd_lennard_jones
```