Actividad 6 - Dinámica molecular-Gases

Exactas Programa

Verano 2020

Hoy modelaremos un sistema de partículas que simulan el comportamiento de gases confinados.

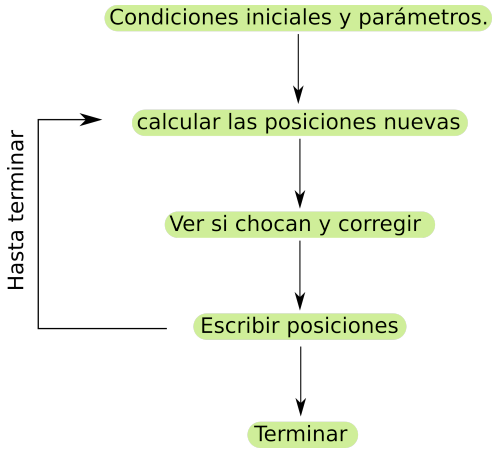
Modelo

El modelo que usaremos es muy sencillo, pero puede representar con gran precisión el comportamiento de muchos gases y constará de un número de partículas en un mundo bidimensional. Vamos a ir construyéndolo en etapas.

1. Partícula única

Para comenzar, simularemos una única partícula contenida en una caja. Para eso tenemos que **integrar** la ecuación de movimiento; en este caso simple no hay aceleración ($F = 0$) por lo que la velocidad sólo cambiará si la partícula choca con la pared. Esta partícula vivirá en una caja bidimensional, un plano XY[1].

El esquema del código es el siguiente:



1.1. Estableciendo las condiciones iniciales

Lo primero que necesitamos, para realizar la simulación, es establecer los parámetros y condiciones iniciales.

1. Defina los valores `min_x`, `max_x`, `min_y`, `max_y` que representen los límites de la caja en la que se moverá las partícula. Para que la visualización sea sencilla, conviene que el tamaño de la caja esté entre 10 y 100 (después podrán probar otros valores).

2. Defina los valores `pos_x` y `pos_y` con la posición inicial de la partícula (Pueden ser cualquier valor dentro de los límites de la caja, incluso pueden ser aleatorios).

3. Defina `pasos_totales` y `dt`, que determinan la cantidad total que pasos que se simularán y el intervalo de tiempo transcurrido entre paso y paso.

[1] Se puede hacer tridimensional también, pero es más *lindo* de ver en dos dimensiones.

4. Defina `vel_x` y `vel_y`, la velocidad inicial de la partícula en el eje X y el eje Y. Procure que la velocidad no sea tan alta como para recorrer la caja en un paso (Es decir, asegúrese de que `vel_x*dt<max_x-min_x`).

1.2. Calculando las posiciones

En cada paso de la simulación debemos determinar la nueva posición y velocidad de la partícula, tanto en el eje X como en el eje Y, cuando pasa una cantidad de tiempo Δt. Recordemos que, como vimos en la clase de Física, esto se hace aplicando la siguiente cuenta:

$$x(t + \Delta t) = x(t) + V_x * \Delta t$$
$$y(t + \Delta t) = y(t) + V_y * \Delta t$$

Es importante considerar qué sucede cuando la partícula se pasa de los bordes de la caja. Por ejemplo, Si el valor de `pos_x` es mayor que `max_x`, significa que la partícula llegó y se pasó de la pared de la derecha. Entonces, debemos corregir la posición y la velocidad haciendo:

```
if pos_x > max_x:
  vel_x = -vel_x
  pos_x = pos_x - 2 * (pos_x - max_x)
```

Con ese código logramos que la partícula *regrese*, lo mismo que ocurriría si le pegamos a la pared con una pelota. Queda para ustedes ver qué es lo que hay que hacer cuando `pos_x < min_x`.

5. Defina la función `rebotar(pos, vel, minimo, maximo)`, que recibe como parámetro la posición y la velocidad de una partícula en un eje y los límites de la caja en ese eje. Si se ha sobrepasado algún límite, efectúa los cambios correspondientes a la posición y la velocidad. Devuelve los nuevos valores de `pos`, y `vel` en una lista, tupla o directamente devolviendo ambas variables simultáneamente (algo así como `return pos,vel`). Vamos a invocar a `rebotar` con las posiciones, velocidades y límites del eje X y las del eje Y.

6. Defina la función `mover_particula(pos_x, pos_y, vel_x, vel_y, dt, x_min, x_max, y_min, y_max)`, que recibe las posiciones actuales de una partícula, el intervalo de tiempo y las dimensiones de la caja, y la mueve de acuerdo a las velocidades dadas. Esta función invoca a `rebotar` para asegurarse que las nuevas posiciones están dentro de la caja, y, luego, devuelve las posiciones y velocidades actualizadas.

Para simular la evolución de la partícula, vamos a invocar a `mover_particula` tantas veces como `pasos_totales`, actualizando las posiciones y velocidades actuales cada vez. La idea va a ser pisar los valores de `pos_x` y `pos_y`, de manera que siempre contengan la posición actual. La historia de las posiciones anteriores la vamos a ir guardando en un archivo que luego interpretaremos con el programa `VMD`.

1.3. Escribiendo los *frames*

1.3.1. Apertura

Para trabajar con un archivo, necesitamos solicitar, en primer lugar, al Sistema Operativo que lo abra (Si se le pide que abra un archivo que no existe, lo crea). Esto se hace por medio del comando `open()`. Debemos hacer:

```
archivo = open("salida.xyz","w")
```

De esta manera se abre el archivo `salida.xyz` en modo escritura, y lo asigna a la variable `archivo` para que podamos trabajar con él. La apertura del archivo se hace **una sola vez**, antes de iniciar la simulación del movimiento de la partícula.

1.3.2. Escritura

En nuestro archivo tenemos que ir imprimiendo la evolución de la posición de la partícula en el tiempo. Cada "foto" del estado del sistema se llama *frame* y tiene el siguiente formato:

- Una línea con el número de partículas,
- Una línea vacía,
- Una línea por cada partícula con: un número entero menor a 110 (el número atómico) y tres números decimales con las coordenadas X, Y y Z de la partícula.

Las escrituras se hacen por medio de la función `print()`, pero en lugar de mostrarlo en pantalla, lo guardaremos en el archivo. La escritura de un *frame* para una partícula queda:

```
print("1",file=archivo)
print(" ",file=archivo)
print("6", pos_x, pos_y , "0", file=archivo)
```

Nota: El 6 indica que trabajamos con átomos de carbono (si prefieren otro, adelante), pero es solo una cuestión que tendrá efecto a la hora de mostrar el resultado de la simulación. Por otro lado, el 0 en el valor de Z hace que todas las partículas estén en el plano XY (sea bidimensional).

7. Defina la función `escribir_frame(archivo, pos_x, pos_y)`, que tome un archivo y la posición de la partícula, y escriba un *frame* con el formato indicado.

Importante: No queremos escribir TODOS los pasos de una dinámica (pues quedaría un archivo gigante), sino uno cada tanto (digamos, cada 100 pasos). Para ello, podemos usar el práctico % (resto de la división). Podemos hacer:

```
if (paso % 100) == 0:
      escribir_frame(archivo, pos_x, pos_y)
```

1.3.3. Cierre

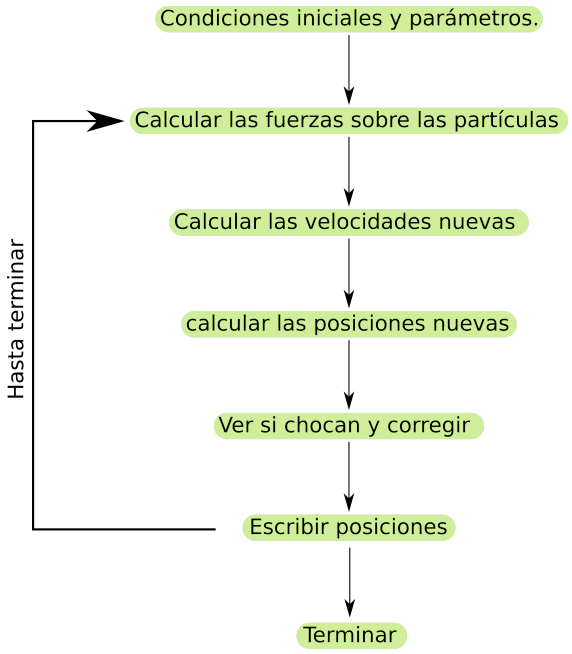
Una vez que terminamos de trabajar con un archivo lo cerramos haciendo:

```
archivo.close()
```

El archivo generado puede ser abierto con un editor de texto (como el block de notas en Windows o el `gedit`) para corroborar que esté imprimiendo lo que queremos. Una vez que estamos seguros, podemos abrirlo desde el `VMD` (Ver la guía de `VMD` para ver cómo).

2. Múltiples partículas

Ahora que tenemos la versión de nuestro sistema con una partícula, convirtámoslo en un esquema general:



2.1. Estableciendo las condiciones iniciales

En este nuevo esquema necesitamos que las posiciones y velocidades sean listas (o `numpy arrays`), donde cada elemento es la posición (o velocidad) de una partícula específica.

8. Defina `nparts`, que establece la cantidad de partículas en el sistema.

9. Modifique los valores de las variables `pos_x` y `pos_y`, que ahora serán listas (o `numpy arrays`), donde el i-ésimo elemento corresponde a la i-ésima partícula (Deberá establecer una posición inicial para cada una. No pueden haber dos partículas con la misma posición inicial).

10. Modifique las variables `vel_x` y `vel_y`, para que también sean listas (o `numpy arrays`) de `nparts` elementos (Y nuevamente, asigne una velocidad inicial a cada partícula).

2.2. Calculando las posiciones

Las partículas de nuestro esquema van a interactuar por medio de fuerzas que van a modificar sus velocidades. Potencialmente podríamos modelar muchos tipos de interacciones (Por ejemplo la interacción gravitacional (¿y por qué no hacerlo?), o la interacción Coulómbica, que mostrará el comportamiento de partículas con carga eléctrica, etc.)

En nuestro caso pondremos un potencial repulsivo de corto alcance (que modela muy bien el comportamiento de gases).

La fuerza en X para la partícula i-ésima se puede calcular según:

$$F_x[i] = 4k \sum^{j \neq i} \frac{(x[i] - x[j])}{|D_{ij}|^6} \tag{1}$$

Recordemos que es más simple si escribimos todo en función de la distancia al cuadrado:

$$F_x[i] = 4k \sum^{j \neq i} \frac{(x[i] - x[j])}{|D2_{ij}|^3} \tag{2}$$

$$(\text{donde } D2_{ij} = D_{ij}^2 = (x[i] - x[j])^2 + (y[i] - y[j])^2)$$

Una vez que hayamos calculado las fuerzas que actúan sobre la partícula, debemos actualizar su velocidad usando las siguientes expresiones:

$$V_x[i](t+\Delta t) = V_x[i](t) + \frac{F_x[i]}{m} * \Delta t$$
$$V_y[i](t+\Delta t) = V_y[i](t) + \frac{F_y[i]}{m} * \Delta t$$

Luego, podemos utilizar la función `mover_particula()` para calcular su nueva posición.

11. Defina la variable `k`, que corresponderá con la constante de la fuerza que estamos utilizando. Inicialmente, se puede poner su valor en 1 (Luego puede experimentar modificándolo).

12. Defina la variable `masas`, una lista que corresponde con las masas de las partículas. Inicialmente puede estar definida como una lista de `nparts` unos (ie. todas las partículas tienen masa 1).

13. Defina la función `dist_cuadrada(x_1,y_1,x_2,y_2)`, que toma la posición en el eje X y el eje Y de dos partículas y calcula su distancia al cuadrado.

14. Defina la función `calcular_fuerzas_x(pos_x,pos_y,k)`, que dadas las listas de posiciones de todas las partículas (y la constante k), determina las fuerzas aplicadas cada una de ellas, y devuelve un par con dos listas de `nparts` elementos: En la primera, la i-ésima posición corresponde con la fuerza a la que es sujeta la i-ésima partícula en el eje X, y en la segunda, la i-ésima posición corresponde con la fuerza a la que es sujeta la i-ésima partícula en el eje Y. Utilice la función `dist_cuadrada()` para facilitar la cuenta. Tenga cuidado de no calcular la interacción de una partícula consigo misma.

15. Defina la función `aplicar_fuerzas(vel_x, vel_y, fzs_x, fzs_y)`, que dadas las listas de velocidades de las partículas y las listas de fuerzas, devuelve un par con las listas de velocidades actualizadas.

16. Defina la función `mover_todas(pos_x,pos_y,vel_x,vel_y,dt)` que toma las listas de posiciones y velocidades y las actualiza aplicándole a cada una `mover_particula()`.

2.2.1. Escritura

Si bien la apertura y el cierre del archivo no cambian, ahora la escritura de un frame debe contemplar a todas las partículas del sistema, adoptando la siguiente forma:

```
print(nparts, file=salida)
print(" ", file=salida)
for i in range(npart):
  print("6", pos_x[i], pos_y[i], "0", file=salida)
```

17. Modifique la función `escribir_frame(archivo, pos_x, pos_y)`, para que ahora tome en `pos_x` y `pos_y` las listas de posiciones y escriba todas en el archivo de salida.

Una vez hechas las funciones se puede correr una simulación, para comenzar es conveniente que se usen los siguientes parámetros que están probados y funcionan.

`nparts=30`, `dt=0.001`, `x_min=0`, `x_max=20`, `y_min=0`, `y_max=20`.

3. Cálculo de propiedades

Un requisito mínimo que debe cumplir un modelo es que la energía total se conserve. Para verificar esto, debemos calcular la energía cinética total según:

$$E_c = \sum_{npart} \frac{1}{2} m[i]|V[i]|^2 \tag{3}$$

(donde $|V[i]|^2 = Vx[i]^2 + Vy[i]^2$)

La energía potencial se calcula según la ecuación 4 (conviene calcularla en simultáneo con las fuerzas).

$$E = \sum_{i=1}^{npart} \sum_{j=i+1}^{npart} \frac{k}{|D2_{ij}|^2} \tag{4}$$

Para visualizar mejor la conservación de la energía se puede graficar usando las herramienta que ya vimos de `Python` o escribir los tres valores en un archivo:

```
print(paso,epot,ecin,epot+ecin,file=ener)
```

Luego podremos abrir el archivo así escrito mediante algún programa graficador (en Linux tenemos el `xmgrace` por ejemplo, si tienen otros sistemas operativos pueden solucionarlo instalando Linux).

La ventaja de escribir a un archivo es que puedo analizar los resultados con herramientas más complejas e interesantes.

4. Si sobra tiempo (o para seguir en casa)

Como ya son grandes programadores es posible que les sobre tiempo y quieran hacer más cosas, aquí algunas ideas:

4.1. Atracciones

Se pueden poner atracciones que actúen a distancias no tan cortas. A la fuerza calculada anteriormente le agregamos un término atractivo:

$$F_x[i] = \sum^{j} 4k\frac{(x[i]-x[j])}{|D2_{ij}|^4} - 2k_2\frac{(x[i]-x[j])}{|D2_{ij}|^2} \tag{5}$$

El primer término de la sumatoria es el repulsivo de antes y el segundo es el atractivo (pueden probar graficar en función de la distancia... ¡es re-cool!).

Manejando los valores de k, k_2 las velocidades iniciales y el número de partículas por unidad de área (o volumen en 3D) podemos ver gotitas de líquido o incluso sólido.

4.2. Diferentes partículas

Se pueden poner en la caja partículas distintas. Pueden tener distinta masa y/o distintas constantes de interacción (las constantes pueden ser distintas para partículas del mismo tipo y otra para interacciones entre las de distinto tipo).

Incluso se puede poner un potencial atractivo para partículas distintas y repulsivo para las igual (o a la inversa) y ver qué pasa.

4.3. Temperatura constante

Es muy común que uno quiera que la temperatura (energía cinética) sea constante (en vez de la energía total). Esto es particularmente útil cuando hay atracciones que pueden calentar a mi sistema demasiado (y porque en general vivimos a temperatura constante).

Esto se logra re-escalando las velocidades según $Vx[i] = Vx[i] \times factor$ (lo mismo para Vy) y

$$factor = \sqrt{Ecin^0/Ecin} \tag{6}$$

donde $Ecin$ es la energía cinética que tiene mi sistema y $Ecin^0$ es la energía cinética que quiero que tenga.

4.4. Moléculas diatómicas

Los átomos pueden estar agrupado en pares formando moléculas diatómicas, para ello tenemos que agregar una interacción entre átomos de la misma molécula cómo:

$$E_{enlace} = \frac{1}{2}k(Dab - D^0)^2 \tag{7}$$

Ojo que acá los a y b son sólo los átomos de una misma molécula y D^0 es la distancia de equilibrio (un parámetro que hay que definir). La fuerza quedaría cómo:

$$Fx[a] = -k(D_{a-b} - D^0)\frac{(x_a - x_b)}{D_{a-b}} \tag{8}$$

Ojo que para cada partícula hay una sola con la que tiene enlace, al menos en diatómicas. ¡¡¡Con esto puedo simular gases diatómicos con un excelente acuerdo con los experimentos!!!