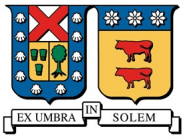


UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Gas Ideal

Tarea N°2

Termodinámica Estadística

IPM-417 - 2023

Martín Achondo Mercado

Rol: 201860005-9

Profesor: Christopher Cooper

11 de Junio de 2023

Resumen

A partir de la mecánica estadística, se pueden obtener relaciones analíticas para el Gas Ideal. Para verificar estas relaciones, se creó un código para simular la evolución en el tiempo de N partículas en un dominio 2D. La simulación realizada considera colisiones elásticas entre las partículas, y contra las paredes.

A partir de los resultados, se notó como a partir de la interacción a escala microscópica entre las partículas, emergen las propiedades macroscópicas. Junto a esto, se pudo obtener que las velocidades de las partículas, cuando el sistema alcanza un estado estacionario, siguen la distribución de *Maxwell-Boltzmann*. Se estimó que el tiempo para alcanzar este estado estacionario crece proporcionalmente al aumentar el número de partículas. Por otra parte, se pudo verificar el cumplimiento de la Ley de Gases Ideales: $PV = Nk_bT$. Esta verificación se realizó para distintos valores de temperatura, volúmenes y cantidad de partículas. De todas formas, se obtuvieron variaciones respecto a la teoría menores al 5 %.

Índice

1. Introducción

El modelo de Gas Ideal es ampliamente usado en la industria y en la investigación debido a la gran simplicidad que se tiene en las expresiones analíticas. Pese a esto, la obtención de estas ecuaciones no es un proceso trivial, ya que necesita un entendimiento de lo que que ocurre a la escala microscópica. Sin embargo, a partir de la mecánica estadística, se pueden derivar estas ecuaciones, y además, encontrar relaciones entre variables macroscópicas y microscópicas.

Es por esto que en este informe se intentará modelar un Gas Ideal a partir de un sistema de partículas evolucionando en el tiempo. De esta forma se podrán evaluar la modelación con los resultados teóricos.

2. Objetivos

2.1. Objetivo general

Modelar satisfactoriamente el Gas Ideal a partir de un sistema de partículas evolucionando en el tiempo.

2.2. Objetivos específicos

- Crear un código que permita modelar la evolución de N partículas dentro de un dominio 2D.
- Comparar los resultados obtenidos con los teóricos dados por la Ley de Gas Ideal. Esto incluye el cálculo de la presión, temperatura y la distribución de velocidades.
- Evaluar el error respecto a la teoría, con variaciones en el volumen, cantidad de partículas y la temperatura en el dominio.

3. Marco teórico

El Gas Ideal es un modelo de la mecánica estadística que simplifica de forma satisfactoria las propiedades de los gases reales, bajo ciertas condiciones. De esta manera, se pueden obtener relaciones termodinámicas entre las variables que definen al gas, y así, encontrar expresiones analíticas para la entropía y energías libres. El modelo de gas ideal se basa en las siguientes premisas:

1. El gas contiene N partículas que siguen un movimiento aleatorio obedeciendo las leyes de Newton.
2. Todas las partículas del gas son idénticas. Tienen la misma forma, masa y dimensión.
3. No existen potenciales de interacción entre las partículas. De esta manera el Hamiltoneano del sistema toma la siguiente forma, en donde $\boldsymbol{p}_i$ y $\boldsymbol{q}_i$ corresponden a la cantidad de movimiento y posición de la partícula i-ésima, y m la masa de cada partícula.

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = \frac{1}{2m} \sum_{i=1}^{N} \|\boldsymbol{p}_i\|^2 \tag{1}$$

Como se dijo anteriormente, bajo estas premisas se pueden encontrar expresiones analíticas para las propiedades termodinámicas. Estas se detallan a continuación:

- Entropía: A partir de la definición de entropía dada por Ludwig Boltzmann [1] y la expresión para el Hamiltoneano dado en la ecuación 1, se puede obtener la siguiente expresión para la entropía de un Gas Ideal:

$$S = k_b N \left[\ln \left(\frac{V}{N} \left(\frac{4\pi m E}{3Nh^2} \right)^{3/2} \right) + \frac{5}{2} \right] \tag{2}$$

  Notar que la entropía queda en función de la cantidad de partículas N, la energía del sistema E y el volumen del sistema V. Las constantes k_b y h corresponden a la constante de Boltzmann y a la constante de Planck respectivamente. A partir de esta expresión analítica, se pueden obtener expresiones para los potenciales termodinámicos en función de N, V, T; siendo T la temperatura del sistema.

- Energía interna:

$$E = \frac{3}{2} N k_b T \tag{3}$$

- Entalpía:

$$H = \frac{5}{2} N k_b T \tag{4}$$

- Energía libre de Gibbs:

$$G = -k_b N T \left[\ln \left(\frac{V}{N} \left(\frac{2\pi k_b T}{h^2} \right)^{3/2} \right) \right] \tag{5}$$

- Energía libre de Helmholtz:

$$A = -k_b N T \left[\ln \left(\frac{V}{N} \left(\frac{2\pi k_b T}{h^2} \right)^{3/2} \right) + 1 \right] \tag{6}$$

Junto a lo anterior, se puede obtener la ecuación de estado de Gas Ideal.

$$PV = N k_b T \tag{7}$$

Por último, es importante mencionar que de la expresión del Hamiltoneano 1 y la mecánica estadística, implica que las cantidades de movimiento de las partículas, y por ende las velocidades, sigan una distribución de probabilidad en específico, llamada *distribución de Maxwell - Boltzmann*. Esta tiene la siguiente forma, considerando $v = \|\boldsymbol{v}\|$.

$$f(v) = 4\pi v^2 \left(\frac{m}{2\pi k_b T} \right)^{3/2} e^{\frac{-mv^2}{2k_b T}} \tag{8}$$

Notar que la curva depende de la temperatura del sistema.

4. Metodología

Para modelar el gas ideal descrito en la sección 3, se considerará lo siguiente. Se tiene un dominio 2D cuadrado de largo L, en donde se encuentran N partículas. En el estado inicial, estas partículas tendrán una posición aleatoria $\boldsymbol{x}_i$ y velocidad con magnitud V_0 en dirección aleatoria. Al ser todas las partículas idénticas, todas tendrán un radio r y una masa m. Estas partículas se dejarán evolucionar manteniendo colisiones elásticas entre ellas y con la pared.

Para la discretización temporal, se tomará un paso de tiempo Δt. El paso temporal se tomará con la restricción: $\Delta t < r/V_0$ para disminuir los errores en la simulación.

4.1. Ecuaciones de movimiento

A medida que se deje evolucionar el sistema, cada partícula se moverá libremente en la dirección aleatoria en que se inicializó. La dirección y magnitud de su velocidad se irá actualizando respecto a las colisiones elásticas con otras partículas y las paredes del dominio. Un detalle se encuentra a continuación:

- Cuando la partícula i-ésima se mueve libremente a través del espacio, se moverá con una velocidad fija con una trayectoria dada por:

$$\boldsymbol{x}_i(t + \Delta t) = \boldsymbol{x}_i + \boldsymbol{v}_i \Delta t \tag{9}$$

- Se considerará que la partícula i-ésima choca con la pared con vector normal unitario $\hat{n}$ en la posición $\boldsymbol{w}$ cuando:

$$\boldsymbol{x}_i - r_i \hat{n} = \boldsymbol{w} \tag{10}$$

  De esta forma, la velocidad se actualizará como:

$$\boldsymbol{v}_i \to \boldsymbol{v}_i - 2(\boldsymbol{v}_i \cdot \hat{n})\hat{n} \tag{11}$$

- Se considerará que la partícula i-ésima con la j-ésima colisionan cuando:

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = r_i + r_j \tag{12}$$

  Durante la colisión, existe una transferencia de cantidad de movimiento $\boldsymbol{I}$ entre ambas partículas. Considerando colisiones elásticas y normales, la transferencia de cantidad de

movimiento se puede calcular como:

$$\boldsymbol{I} = -m\left[(\boldsymbol{v}_i - \boldsymbol{v}_j)\cdot \hat{r}_{ij}\right]\hat{r}_{ij} \tag{13}$$

En donde $\hat{r}_{ij} = (\boldsymbol{x}_i - \boldsymbol{x}_j)/\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ es un vector unitario que apunta en el eje que forman los centros de ambas partículas.

De esta forma, las velocidades pueden ser actualizadas con la siguiente relación:

$$\begin{cases} \boldsymbol{v}_i \to \boldsymbol{v}_i + \boldsymbol{I}/m \\ \boldsymbol{v}_j \to \boldsymbol{v}_j - \boldsymbol{I}/m \end{cases} \tag{14}$$

Con las ecuaciones de movimiento antes mencionadas, se podrá dejar evolucionar el sistema para poder llegar a un estado estacionario.

Por otra parte, en cada paso temporal se calculará la temperatura del sistema. Para esto se calculará la energía total, la cual corresponde a la suma de las energías cinéticas de las partículas, Ec. 1:

$$E = \frac{1}{2}m\sum_{i=1}^{N}\|\boldsymbol{v}_i\|^2 \tag{15}$$

De esta manera, la temperatura se puede estimar como:

$$T = \frac{E}{Nk_b} \tag{16}$$

Notar que esta ecuación difiere de la Ec. 3 dado que se adaptó para el caso 2D.

Un punto importante de mencionar es que el input del código para la energía del sistema es la velocidad inicial de las partículas, pero esto es equivalente a fijar una temperatura de entrada. Producto a las ecuaciones anteriores se puede obtener:

$$T = \frac{mV_0^2}{2Nk_b} \tag{17}$$

Los resultados se mostrarán a base de la temperatura y no la velocidad inicial.

De forma análoga, la presión se estimará a partir de la transferencia de cantidad de movimiento entre las partículas y las paredes. Para esto se calculará la transferencia de momentum total Q como:

$$Q = -2m\sum_{i\in \boldsymbol{w}}\boldsymbol{v}_i\cdot\hat{n} \tag{18}$$

De esta manera, la presión se puede calcular como:

$$P = \frac{Q}{4L\Delta t_n} \tag{19}$$

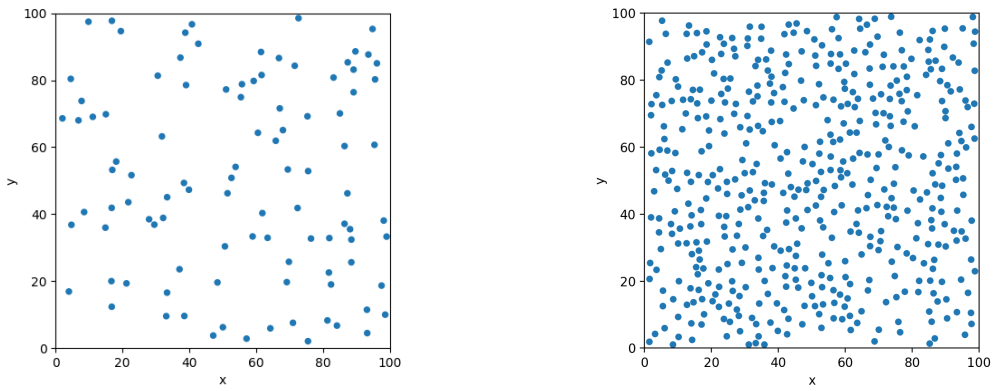
Para obtener un valor de presión estable, se irá calculando la transferencia de momentum acumulada dividido por el tiempo de simulación hasta esa iteración, Δt_n .

En base a lo anterior, se puede calculará el residual R_e de la ecuación de Gas Ideal, Ec. 7 como:

$$R_e = \frac{|PV - Nk_bT|}{N} \tag{20}$$

Notar que dado que el sistema es 2D, el volumen corresponde al área del cuadrado, $V = L^2$.

En las siguientes imágenes se muestra como las partículas se visualizan en las simulaciones.



(a) $N = 100$ (b) N=500

Figura 1: Visualización de las partículas en el dominio para distintos N

4.2. Algoritmo

En base a la metodología planteada, se presenta el algoritmo a seguir para modelar la evolución del sistema de partículas.

Algoritmo 1 Modelación de un gas ideal

Input: $N, V_0, r, m, L, \Delta t, N_{steps}$

Inicializar N partículas con posición y dirección aleatoria
for $n = 1$ to $n = N_{steps}$ **do**
  for $i = 1$ to $i = N$ **do**
    for $j = i + 1$ to $j = N$ **do**
      if $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = r_i + r_j$ **then** ▷ Ec. 12
        $\boldsymbol{v}_i \to \boldsymbol{v}_i + \boldsymbol{I}/m$ ▷ Ec. 14
        $\boldsymbol{v}_j \to \boldsymbol{v}_j - \boldsymbol{I}/m$ ▷ Ec. 14
      end if
    end for
    if $\boldsymbol{x}_i - r_i\hat{n} = \boldsymbol{w}$ **then** ▷ Ec. 10
      $\boldsymbol{v}_i \to \boldsymbol{v}_i - 2(\boldsymbol{v}_i \cdot \hat{n})\hat{n}$ ▷ Ec. 11
    end if
    $\boldsymbol{x}_i \to \boldsymbol{x}_i + \boldsymbol{v}_i\Delta t$ ▷ Ec. 9
  end for
  Estimar temperatura y presión
end for

Output: Posiciones y velocidades de cada partícula

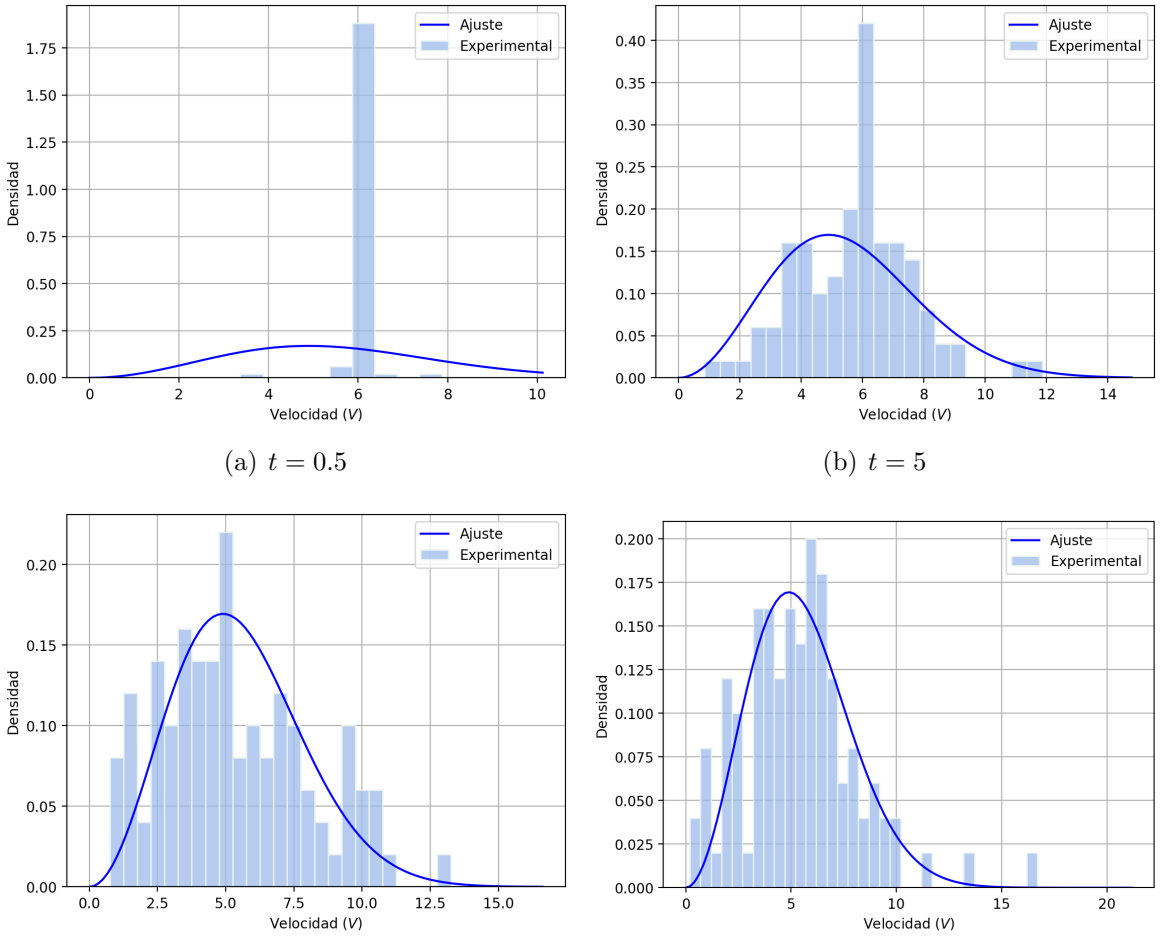
5. Resultados

Se presentan los resultados obtenidos. Para normalizar los valores, se le asignó el siguiente valor a la constante de Boltzmann, $k_b = 0.1$. Es por esto que los resultados se presentan sin unidad.

Para todas las simulaciones se utilizarán partículas de radio $r = 1$, masa $m = 1$ y un paso de tiempo de $dt = 0.005$.

5.1. Evolución de la distribución de velocidad

Se incluyen las distribuciones de velocidades para distintos tiempos finales con $N = 100$. Se mantiene constante la temperatura en $T = 180$ y el volumen en $L = 100^2$.



(a) $t = 0.5$

(b) $t = 5$

(c) $t = 15$

(d) $t = 50$

Figura 2: Distribución de velocidades a distintos tiempos

Para encontrar la dependencia existente entre el número de colisiones por unidad de tiempo y partículas, y el número de partículas en el dominio, se grafican estas variables y se estima la curva de ajuste. Estas simulaciones se realizaron para $T = 180$ y $V = 100^2$.

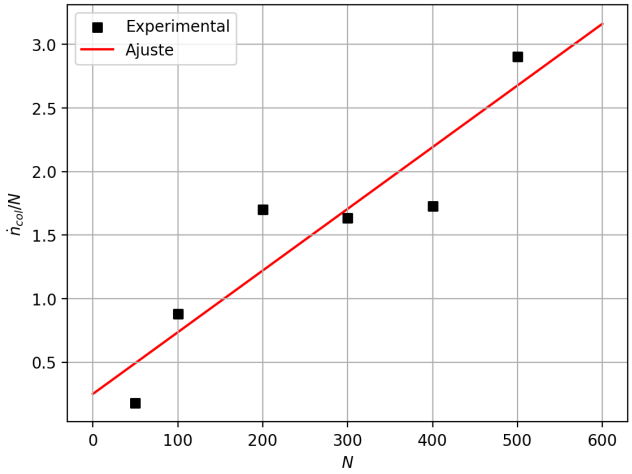


Figura 3: Número de colisiones por unidad de tiempo y partículas vs el número de partículas

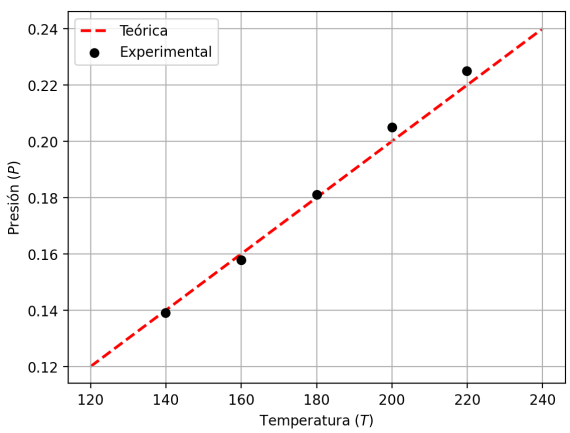
5.2. Verificación de la Ley de Gases Ideales

Se incluyen las comparaciones para la presión obtenida experimentalmente y la presión teórica. Para los gráficos adjuntos se utilizaron las siguientes variables:

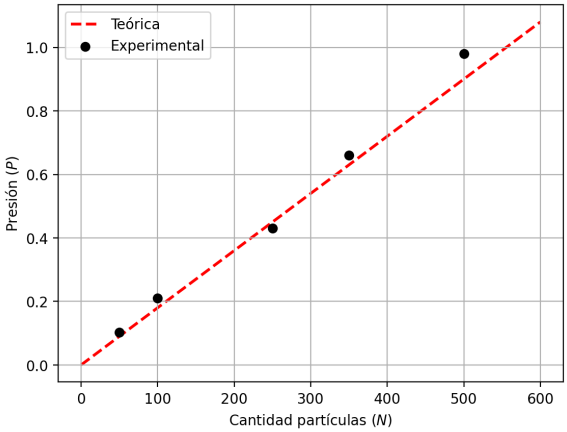
a) P vs T: Se mantuvo $N = 100$ y $V = 100^2$.

b) P vs N: Se mantuvo $T = 180$ y $V = 100^2$.

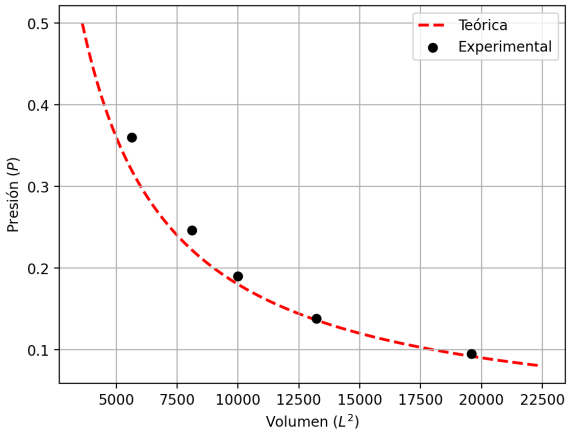
c) P vs V: Se mantuvo $N = 100$ y $T = 180$.



(a) P vs T



(b) P vs N



(c) P vs V

Figura 4: Comparación entre la presión estimada y la curva teórica dada por la Ley de Gases Ideales

Además, para verificar el cumplimiento de la ecuación de Gas Ideal 7, se grafica el residual obtenido en cada iteración n.

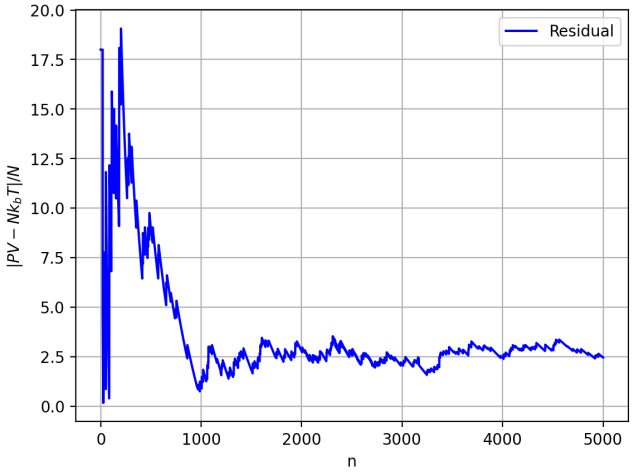


Figura 5: Residual en cada iteración

6. Análisis

6.1. Evolución de la distribución de velocidad

A partir de los resultados presentados, se nota como la distribución de velocidades de las partículas evoluciona con el tiempo de simulación. Esta distribución varía hasta alcanzar el estado estacionario dado por la distribución de Maxwell-Boltzmann dado en la Ec. 8. Los resultados se muestran en la figura 2. Es importante notar que esta distribución es alcanzada producto de las colisiones elásticas que existen entre las partículas del dominio, debido a que en estas colisiones existe una transferencia de cantidad de movimiento entre las partículas. Para las condiciones fijadas; $N = 100$, $T = 180$ y $V = 100^2$, el tiempo necesario para alcanzar esta distribución corresponde aproximadamente a $t_{ST} = 50$. Por otra parte, es válido inferir que mientras se tengan más partículas, más continua será la distribución obtenida.

Es importante destacar que este tiempo, t_{ST}, depende de N, V, T, dado que con estas variables se varía la frecuencia de colisiones por unidad de tiempo y partícula, $\dot{n}_{\text{col}}/N$. Se asume que esta relación es proporcional.

$$t_{ST} \propto \frac{\dot{n}_{\text{col}}}{N} \tag{21}$$

Dado que no es trivial obtener el tiempo al alcanzar el estado estacionario, se estimará la relación existente entre la frecuencia de colisión por partícula y la cantidad de partículas. Los resultados de este experimento se presentan en la figura 3. A partir del ajuste obtenido, se tiene que ambas variables se relacionan como:

$$\frac{\dot{n}_{\text{col}}}{N} \propto N \tag{22}$$

Esto implica que la frecuencia de colisiones es proporcional al número de partículas en el dominio. Esto lleva a inferir que el tiempo para alcanzar el estado estacionario también varía de manera proporcional con el número de partículas, a temperatura y volumen constante.

6.2. Verificación de la Ley de Gases Ideales

Respecto a los resultados obtenidos, se nota que de manera general el modelo creado cumple con las ecuaciones que describen a un Gas Ideal. El primer análisis realizado se basa en probar si el modelo logra replicar el valor de presión a partir de las colisiones de las partículas con las paredes. En la figura 4 se notan los resultados respecto a variaciones en el volumen, la temperatura y la cantidad de partículas. En base a esto, se nota como el modelo es capaz

de predecir satisfactoriamente la presión, manteniendo las tendencias entre las variables. Las tendencias mencionadas corresponden a la proporcionalidad respecto a la temperatura y la cantidad de partículas, y la proporcionalidad inversa con el volumen. De todas formas, se obtiene variaciones no mayores al 5 %.

Es válido mencionar que las mayores variaciones obtenidas ocurren cuando la frecuencia de colisiones aumenta. Esto se puede deber a deficiencias en la simulación, las cuales pueden ser corregidas disminuyendo el paso temporal.

De la misma forma, para evaluar la ecuación de Gas Ideal, se calcula el residual de esta en cada iteración. En la figura 5 se muestra como el residual evoluciona con el número de iteraciones. Notar que en las primeras iteraciones este tiende a crecer, y luego se mantiene constante en un valor cercano a 2.5. Este comportamiento se puede deber principalmente a que en las primeras iteraciones se tiene un valor de presión obtenido bastante bajo dado que todavía no se tienen suficientes colisiones con la pared, las cuales se estabilizan con las iteraciones.

Entonces, de forma general, es válido destacar que para ambas simulaciones realizadas, los resultados obtenidos coinciden con las predicciones teóricas. Además, se obtuvieron los mismos resultados que obtuvo Chang, J. en su publicación [3], verificando que la mecánica estadística es un marco confiable para modelar grandes cantidades de partículas.

7. Conclusiones

Para finalizar, se nota como a partir del código elaborado, se logró modelar satisfactoriamente el Gas Ideal. De los resultados, se nota como se pudo replicar la distribución de *Maxwell-Boltzmann* para las velocidades, la estimación de la presión respecto a las colisiones de las partículas con la pared y la verificación de la relación entre la temperatura y la presión dada por la ecuación de Gas Ideal. Lo anterior refleja como las interacciones a las escalas microscópicas reflejan propiedades a escalas macroscópicas.

Pese a lo anterior, las simulaciones se realizaron para bajos números de partículas. Esto es debido al gran costo computacional que implica. Esto lleva a demostrar la gran complejidad que tiene realizar simulaciones moleculares a grandes escalas.

Por último, es valido mencionar que al código elaborado se le pueden añadir potenciales de interacción entre las partículas, como por ejemplo el potencial de Lennard-Jones. Sería interesante llevar a cabo el mismo análisis pero incluyendo este potencial, para ver si se pueden obtener las relaciones de Gas de Van der Waals.

8. Referencias

[1] Reif, F. (1965). *Fundamentals of Statistical and Thermal Physics*. McGraw-Hill.

[2] Landau, L.D., Lifshitz, E.M. (1976). *Mechanics, Course of Theoretical Physics*. Pergamon.

[3] Chang, J. (2018). *Simulating an Ideal Gas to Verify Statistical Mechanics*.

[4] Siti Nurul Khotimaha, dan Sparisoma Viridia. (2011). *Partition function of 1–, 2–, and 3–D monatomic ideal gas: A simple and comprehensive review*.

[5] Cooper, C. (2023). *Apuntes de Termodinámica Estadística*. UTFSM.

9. Anexo

Se incluye el código implementado para la modelación de Gas Ideal.

Código 1: Código en Python para la modelación

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.collections import PatchCollection as plt_circles
from PIL import Image, ImageDraw
from tqdm import tqdm as log_progress
import os
from scipy.stats import maxwell
import pandas as pd


class Particles():

    particle_list = list()

    def __init__(self,r,m):
        Particles.particle_list.append(self)
        self.r = r
        self.A = np.pi*self.r**2
        self.s = self.A*4/np.pi

        self.m = m

        x = np.random.uniform(0,1)*self.L
        y = np.random.uniform(0,1)*self.L
        self.X = np.array([x,y])

        theta = np.random.uniform(0,2*np.pi)
        Vx = Particles.V0*np.cos(theta)
        Vy = Particles.V0*np.sin(theta)
        self.V = np.array([Vx,Vy])

        self.dp = 0

    @property
    def E(self):
        E = 0.5*self.m*np.linalg.norm(self.V)**2
        return E

    def step(self):
        self.update_wall()
        self.update_position()
```

```
43
44      def update_position(self):
45          self.X += self.V*Particles.dt
46
47      def update_collision(self,particle2):
48          r = (self.X-particle2.X)/np.linalg.norm(self.X-particle2.X)
49          I = -2*(self.m**2/(2*self.m))*(np.dot((self.V-particle2.V),r)*r
               )
50          self.V += I/self.m
51          particle2.V -= I/particle2.m
52
53      def update_wall(self):
54          flagx,flagy = self.check_wall()
55          if flagx:
56              self.V[0] *= -1
57              self.dp += 2*self.m*np.abs(self.V[0])
58          if flagy:
59              self.V[1] *= -1
60              self.dp += 2*self.m*np.abs(self.V[1])
61
62      def check_wall(self):
63          x,y = self.X
64          ret = np.array([0,0])
65          if (x+self.r >= self.L) or (x-self.r <= 0):
66              ret[0] = 1
67          if (y+self.r >=  self.L) or (y-self.r <= 0):
68              ret[1] = 1
69          return ret
70
71      def check_collision(self,particle2):
72          if np.linalg.norm(self.X - particle2.X) <= self.r + particle2.r
               :
73              return True
74          else:
75              return False
76
77
78  class Simulation():
79
80      def __init__(self, Particles,
81                   N=100,
82                   L = 1.0,
83                   V0 = 0.05,
84                   r=0.02,
85                   m=1,
86                   dt=0.01):
87
88          self.Particles = Particles
```

```
89          self.N_particles = N
90          self.Particles.L = L
91          self.Particles.V0 = V0
92          self.Particles.r = r
93          self.Particles.m = m
94
95          self.dt = dt
96          self.Particles.dt = self.dt
97          self.L = L
98
99          self.total_dp = 0
100         self.P = 0
101         self.T = 0
102         self.n = 0
103
104         self.kb = 0.1
105         self.collision_number = 0
106         self.L_residual = list()
107
108         self.dir_path = 'results'
109
110
111     def create_particles(self):
112         print('Creating particles')
113         for i in range(self.N_particles):
114             flag = True
115             while flag:
116                 flag = False
117                 particle = self.Particles(r=self.Particles.r, m=self.
                        Particles.m)
118                 wall = particle.check_wall()
119                 s = wall.sum()
120                 if s>0:
121                     flag = True
122                     self.Particles.particle_list.remove(particle)
123                     del particle
124                 else:
125                     for particle2 in Particles.particle_list:
126                         if particle is particle2:
127                             continue
128                         elif particle.check_collision(particle2):
129                             flag = True
130                             self.Particles.particle_list.remove(
                                    particle)
131                             del particle
132                             break
133         print('particles created')
134
```

```
135
136
137     def simulation_step(self):
138
139         L_particles = list(self.Particles.particle_list)
140         N_total = len(L_particles)
141
142         for i in range(N_total):
143             for j in range(i+1,N_total):
144                 particle1 = L_particles[i]
145                 particle2 = L_particles[j]
146                 if particle1.check_collision(particle2):
147                     particle1.update_collision(particle2)
148                     self.collision_number += 1
149             particle1.step()
150
151             self.total_dp += particle1.dp
152             self.E += particle1.E
153
154
155     def update_variables(self):
156         N = self.N_particles
157         P = self.total_dp/(self.Particles.dt*4*self.Particles.L*self.n)
158         T = (2/2)*(1/(N*self.kb))*self.E
159
160         self.P = P
161         self.T = T
162         self.residual = np.abs(self.P*self.L**2 - N*self.kb*self.T)/N
163         self.L_residual.append(self.residual)
164
165
166     def run_simulation(self, N_steps=10, plot=False):
167
168         self.N_steps = N_steps
169         self.plot = plot
170         self.create_particles()
171
172         frames = list()
173         pbar = log_progress(range(self.N_steps))
174         pbar.set_description("Residual: %s " % 1000)
175         for n in pbar:
176             self.n = n + 1
177             if self.plot:
178                 if n%5==0:
179                     frame = self.create_frame_image()
180                     frames.append(frame)
181             self.time_step(n)
182             if n % 1 == 0:
```

```
183                 pbar.set_description("Residual: {:6.4e}, Collisions: {}
                        ".format(self.residual,self.collision_number))
184
185         if self.plot:
186             self.save_animation(frames)
187
188
189     def time_step(self,n):
190         self.total_dp = 0
191         self.E = 0
192         self.simulation_step()
193         self.update_variables()
194
195
196     def postprocessing(self):
197         print('')
198         print(f'Number of particles: {self.N_particles}')
199         print(f'Volume = {self.L}^2')
200         print(f'V0 = {self.Particles.V0}')
201         print(f'Temperature = {self.T}')
202         print(f'Pressure Theoretical = {self.T*self.kb*self.N_particles
                /self.L**2}')
203         print(f'Pressure Simulated = {self.P}')
204         print(f'Energy / N = {self.E/self.N_particles}' )
205         print(f'Total steps: {self.N_steps}')
206         print(f'Residual = {self.residual}')
207         print(f'Total collisions: {self.collision_number}')
208         print(f'Time step dt = {self.dt}')
209         print(f'Final time {self.dt*self.N_steps} [s]')
210         print('')
211
212         self.plot_residual()
213         self.plot_velocity()
214         self.plot_particles()
215
216     def plot_residual(self):
217         fig, ax = plt.subplots()
218         ax.plot(self.L_residual, label='Residual ', c='b')
219         ax.set_xlabel('n')
220         ax.set_ylabel(r'$|PV-Nk_{b}T|/N$')
221         ax.grid()
222         ax.legend()
223         ax.set_title('Ideal Gas Law Residual')
224         name = 'Residual.png'
225         fig.savefig(os.path.join(self.dir_path,name))
226
227     def plot_velocity(self):
228         fig, ax = plt.subplots()
```

```
229         L_V = list()
230         L_Vx = list()
231         L_Vy = list()
232         for particle in self.Particles.particle_list:
233             V = np.sqrt(particle.V[0]**2 + particle.V[1]**2)
234             L_V.append(V)
235             L_Vx.append(particle.V[0]**2)
236             L_Vy.append(particle.V[1]**2)
237         L_V = np.array(L_V)
238 
239         sns.histplot(L_V, stat='density', binwidth=0.6, label='
                Simulation', color='#9dbbeb', edgecolor='#e9f2f1')
240         params = maxwell.fit(L_V, floc=0)
241         x = np.linspace(0, np.max(L_V)*1.5, 100)
242         y = maxwell.pdf(x, *params)
243         sns.lineplot(x=x, y=y, color='b', label='Fitted Distribution',
                ax=ax)
244 
245         ax.set_title('Velocity Distribution')
246         ax.grid(True)
247         ax.set_xlabel('Velocity')
248         ax.set_ylabel('Density')
249         ax.legend()
250 
251         name = 'Velocity_distribution.png'
252         fig.savefig(os.path.join(self.dir_path,name))
253 
254 
255     def plot_particles(self):
256 
257         fig, ax = plt.subplots()
258 
259         self.circles = list()
260         self.x_particles = list()
261         self.y_particles = list()
262 
263         for particle in self.Particles.particle_list:
264             self.x_particles.append(particle.X[0])
265             self.y_particles.append(particle.X[1])
266             self.circles.append(plt.Circle((particle.X[0],particle.X
                    [1]),linewidth=0 ,radius=particle.r, color='k'))
267 
268         c = plt_circles(self.circles)
269         ax.add_collection(c)
270         ax.set_box_aspect(1)
271         ax.set_xlim([0,Particles.L])
272         ax.set_ylim([0,Particles.L])
273         ax.set_xlabel('x')
```

```
274         ax.set_ylabel('y')
275 
276         name = 'Particles.png'
277         fig.savefig(os.path.join(self.dir_path,name))
278 
279 
280     def create_frame_image(self):
281         scale_factor = 10
282         image_width = int(self.L * scale_factor)
283         image_height = int(self.L * scale_factor)
284         image = Image.new('RGB', (image_width, image_height), 'white')
285         draw = ImageDraw.Draw(image)
286         for particle in self.Particles.particle_list:
287             x1 = int((particle.X[0] - particle.r) * scale_factor)
288             y1 = int((particle.X[1] - particle.r) * scale_factor)
289             x2 = int((particle.X[0] + particle.r) * scale_factor)
290             y2 = int((particle.X[1] + particle.r) * scale_factor)
291             draw.ellipse([(x1, y1), (x2, y2)], fill='blue')
292         return image
293 
294     def save_animation(self, frames):
295         name = 'gas_simulation.gif'
296         frames[0].save(os.path.join(self.dir_path,name), save_all=True,
                append_images=frames[1:], optimize=False, duration=100,
               loop=0)
297 
298 
299 
300 if __name__=='__main__':
301 
302     parameters = {
303         'm': 1,
304         'r': 1,
305         'V0': 6,
306         'L': 100,
307         'dt': 0.005
308     }
309 
310     IdealGas = Simulation(Particles, N=100, **parameters)
311     IdealGas.run_simulation(N_steps=2000, plot=False)
312     IdealGas.postprocessing()
```