Индивидуальный проект

# Моделирование взаимной диффузии двух газов

Михеев Алексей
МФТИ
2025

# Основные положения

## Цели

- Смоделировать процесс дифузии двух газов в двух сосудах, соединенных трубкой;
- Сделать анимацию, показывающую поведение молекул газа во времени, а также изменяющиеся в реальном времени графики, показывающие глубину протекания диффузии и распределение частиц по скоростям.

## Используемые технологии

- Python 3.12.3;
- Numpy, Matplotlib, Scipy;
- Ноутбук DELL Latitude 5530 Intel core i5;
- Ubuntu 24.04;
- Visual Studio Code;
- Мозг и руки.

# Постановка задачи

## Описание системы

- Имеются два цилиндрических сосуда, соединенных трубкой;
- Имеются два одноатомных идеальных газа, из которых один изначально находится в первом сосуде, а второй - во втором;
- Температура одинакова во всех точках системы и постоянна;
- Начальные положения частиц внутри сосудов и начальные скорости случайны;
- Скорости нормируются в соответствии с температурой;
- Столкновения частиц со стенками и с другими частицами абсолютно упругие.

# Постановка задачи 2

## Описание графиков

- На первом графике строятся гистограммы распределения для каждого газа, а также теоретические кривые в соответствии с распределением Максвелла.
- На втором графике строятся две кривые - зависимости доли частиц первого газа в первом сосуде и доли частиц второго газа во втором сосуде от времени.

# Параметры системы

- Количества частиц для каждого газа;
- Радиусы сосудов, радиус трубки;
- Длина сосудов, длина трубки;
- Радиусы частиц для каждого газа;
- Массы частиц для каждого газа;
- Температура;
- Константа Больцмана
- Интервал времени обновления системы.

# Инициализация сосудов и соединительной трубки

```python
def plot_cylinder(ax, radius, length, x_offset, color, alpha=0.1):
    theta = np.linspace(0, 2*np.pi, 50)
    z = np.linspace(0, length, 2)
    theta_grid, z_grid = np.meshgrid(theta, z)
    x_grid = x_offset + z_grid
    y_grid = radius * np.cos(theta_grid)
    z_grid = radius * np.sin(theta_grid)
    ax.plot_surface(x_grid, y_grid, z_grid, color=color, alpha=alpha, edgecolor='none')

    theta = np.linspace(0, 2*np.pi, 50)
    r = np.linspace(0, radius, 10)
    theta_grid, r_grid = np.meshgrid(theta, r)
    x = x_offset * np.ones_like(theta_grid)
    y = r_grid * np.cos(theta_grid)
    z = r_grid * np.sin(theta_grid)
    ax.plot_surface(x, y, z, color=color, alpha=alpha)

    x = (x_offset + length) * np.ones_like(theta_grid)
    ax.plot_surface(x, y, z, color=color, alpha=alpha)

def plot_tube(ax, radius, length, x_offset, color, alpha=0.1):
    theta = np.linspace(0, 2*np.pi, 50)
    z = np.linspace(0, length, 2)
    theta_grid, z_grid = np.meshgrid(theta, z)
    x_grid = x_offset + z_grid
    y_grid = radius * np.cos(theta_grid)
    z_grid = radius * np.sin(theta_grid)
    ax.plot_surface(x_grid, y_grid, z_grid, color=color, alpha=alpha, edgecolor='none')

fig = plt.figure(figsize=(12, 8))
ax1 = fig.add_subplot(1,5,(1,3), projection='3d')
ax2 = fig.add_subplot(2,5,(4,5))
ax3 = fig.add_subplot(2,5,(9,10))

plot_cylinder(ax1, box1_radius, box_length, 0, 'gray')
plot_cylinder(ax1, box2_radius, box_length, box_length + tube_length, 'gray')
plot_tube(ax1, tube_radius, tube_length, box_length, 'gray')

total_length = 2*box_length + tube_length
ax1.set_xlim(0, total_length)
ax1.set_ylim(-max(box1_radius, box2_radius, tube_radius), max(box1_radius, box2_radius, tube_radius))
ax1.set_zlim(-max(box1_radius, box2_radius, tube_radius), max(box1_radius, box2_radius, tube_radius))
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')
```

# Инициализация частиц

```
positions1 = np.random.rand(num_part1, 3)
positions1[:, 0] = positions1[:, 0] * (box_length - 2*part1_radius) + part1_radius
theta1 = 2 * np.pi * np.random.rand(num_part1)
r1 = (box1_radius - part1_radius) * np.sqrt(np.random.rand(num_part1))
positions1[:, 1] = r1 * np.cos(theta1)
positions1[:, 2] = r1 * np.sin(theta1)

positions2 = np.random.rand(num_part2, 3)
positions2[:, 0] = positions2[:, 0] * (box_length - 2*part2_radius) + box_length + tube_length + part2_radius
theta2 = 2 * np.pi * np.random.rand(num_part2)
r2 = (box2_radius - part2_radius) * np.sqrt(np.random.rand(num_part2))
positions2[:, 1] = r2 * np.cos(theta2)
positions2[:, 2] = r2 * np.sin(theta2)

positions = np.vstack((positions1, positions2))

velocities1 = (np.random.rand(num_part1, 3)-0.5)
velocities1 *= np.sqrt((num_part1)*v2_ave1/(sum(sum(velocities1**2))))
velocities2 = (np.random.rand(num_part2, 3)-0.5)
velocities2 *= np.sqrt((num_part2)*v2_ave2/(sum(sum(velocities2**2))))
velocities = np.vstack((velocities1,velocities2))

colors = np.array(['red']*num_part1 + ['blue']*num_part2)

scatter = ax1.scatter(
    positions[:, 0], positions[:, 1], positions[:, 2],
    s=200*(np.array([part1_radius**2 for i in range(num_part1)]+[part2_radius**2 for i in range(num_part2)])),
    c=colors, alpha=0.7
)
```

# Распределение скоростей

```
ax2.set_xlim(0, vmax)
ax2.set_ylim(0, (num_part2+num_part1)/4)
ax2.set_xlabel('Скорость')
ax2.set_ylabel('Количество частиц')
ax2.set_title('Распределение скоростей частиц')
ax2.grid(True, linestyle='--', alpha=0.6)

dv = vmax/(num_bins+1)
X = np.arange(dv/2, vmax, dv)
Fv1 = 4*np.pi*(mass1/(2*np.pi*Kb*T))**1.5 * X**2 * np.exp(-mass1*X**2/(2*Kb*T))
Fv2 = 4*np.pi*(mass2/(2*np.pi*Kb*T))**1.5 * X**2 * np.exp(-mass2*X**2/(2*Kb*T))
Y1 = num_part1*Fv1*dv
Y2 = num_part2*Fv2*dv
ax2.plot(X, Y1, 'red',label='Теор. распределение 1')
ax2.plot(X, Y2, 'blue',label='Теор. распределение 2')

bin_edges = np.linspace(0, vmax, num_bins + 1)
bar_container1 = ax2.bar(
    bin_edges[:-1], np.zeros(num_bins),
    width=bin_edges[1]-bin_edges[0],
    color='red', edgecolor='darkred', alpha=0.5, label='Газ 1'
)
bar_container2 = ax2.bar(
    bin_edges[:-1], np.zeros(num_bins),
    width=bin_edges[1]-bin_edges[0],
    color='blue', edgecolor='darkblue', alpha=0.5, label='Газ 2'
)
ax2.legend()
```

# График диффузии

```
ax3.set_xlabel('Время')
ax3.set_ylabel('Доля частиц')
ax3.set_title('График зависимостей долей частиц в сосудах от времени')
ax3.set_ylim(0, 1)
ax3.grid(True, linestyle='--', alpha=0.6)

line_red, = ax3.plot([], [], 'r-', label='Доля красных в левом цилиндре')
line_blue, = ax3.plot([], [], 'b-', label='Доля синих в правом цилиндре')
ax3.legend()
```

# Столкновения со стенками

Функция handle_wall_collisions отвечает за обработку столкновений частиц со стенками:

- Для каждой частицы проверяет, соприкасается ли она в текущий момент времени со стенкой;
- Если да, то меняет направление нормальной к стенке составляющей скорости на противоположное;
- Во избежание "залипаний" отодвигает частицу от стенки на безопасное расстояние.

```python
def handle_wall_collisions(pos, vel, k, rad1, rad2):
    for i in range(len(pos)):
        if i < k:
            rad = rad1
        else:
            rad = rad2
        x, y, z = pos[i]
        r = np.sqrt(y**2 + z**2)

        if x < box_length:
            current_radius = box1_radius
            if r >= tube_radius:
                max_x = box_length - rad
            else:
                max_x = box_length+1
            min_x = rad
        elif x > box_length + tube_length:
            current_radius = box2_radius
            if r >= tube_radius:
                min_x = box_length + tube_length + rad
            else:
                min_x = box_length + tube_length
            max_x = 2*box_length + tube_length - rad
        else:
            current_radius = tube_radius
            min_x = box_length
            max_x = box_length + tube_length

        if x < min_x:
            vel[i, 0] = abs(vel[i, 0])
            pos[i, 0] = 1.01*min_x
        elif x > max_x:
            vel[i, 0] = -abs(vel[i, 0])
            pos[i, 0] = 0.99*max_x

        if r > current_radius - rad:
            normal = np.array([0, y/r, z/r])
            delta = (current_radius - rad) / r
            pos[i, 1] *= delta * 0.99
            pos[i, 2] *= delta * 0.99
            vel_normal = np.dot(vel[i], normal) * normal
            vel_tangent = vel[i] - vel_normal
            vel[i] = vel_tangent - vel_normal
```

# Столкновения между частицами

Функция
handle_particle_collisions
отвечает
за обработку столкновений
частиц между собой:

- Находит соприкасающиеся пары частиц;
- Для
  каждой такой пары меняет скорости составляющих
  в соответствии
  с законами сохранения;
- Во избежание "залипаний" раздвигает частицы.

```
def handle_particle_collisions(pos, vel, k, rad1, rad2, mass1, mass2):
    dist = squareform(pdist(pos))
    x1, y1 = np.where(dist[:k,:k] < 2*rad1)
    x2, y2 = np.where(dist[:k,k:] < rad1+rad2)
    x3, y3 = np.where(dist[k:,k:] < 2*rad2)
    y2 += k
    x3 += k
    y3 += k

    for x,y,rad in ((x1,y1,rad1),(x3,y3,rad2)):
        for z in range(len(x)):
            i = x[z]
            j = y[z]
            if i >= j:
                continue
            r_ij = pos[i] - pos[j]
            d = np.linalg.norm(r_ij)
            if d > 0:
                r_ij = r_ij/d
            v_ij = vel[i] - vel[j]
            v_along = np.dot(v_ij, r_ij)
            if v_along < 0:
                vel[i] -= v_along*r_ij
                vel[j] += v_along*r_ij
                overlap = 2*rad - d
                pos[i] += overlap*r_ij/2
                pos[j] -= overlap*r_ij/2

    for x in range(len(x2)):
        i = x2[x]
        j = y2[x]
        r_ij = pos[i] - pos[j]
        d = np.linalg.norm(r_ij)
        if d > 0:
            r_ij = r_ij/d
        v_ij = vel[i] - vel[j]
        v_along = np.dot(v_ij, r_ij)
        if v_along < 0:
            vel[i] -= v_along*r_ij*2*mass2/(mass1+mass2)
            vel[j] += v_along*r_ij*2*mass1/(mass1+mass2)
            overlap = rad1+rad2 - d
            pos[i] += overlap*r_ij/2
            pos[j] -= overlap*r_ij/2
```

# Подсчет количества частиц в сосудах

Функция $\mathrm{count\_particles}$ рассчитывает доли частиц первого газа в первом сосуде и частиц второго газа во втором сосуде.

```
def count_particles(positions):
    red_in_left = np.sum((positions[:num_part1, 0] < box_length) &
                         (np.sqrt(positions[:num_part1, 1]**2 + positions[:num_part1, 2]**2) < box1_radius))
    blue_in_left = np.sum((positions[num_part1:, 0] < box_length) &
                          (np.sqrt(positions[num_part1:, 1]**2 + positions[num_part1:, 2]**2) < box1_radius))

    blue_in_right = np.sum((positions[num_part1:, 0] > box_length + tube_length) &
                           (np.sqrt(positions[num_part1:, 1]**2 + positions[num_part1:, 2]**2) < box2_radius))
    red_in_right = np.sum((positions[:num_part1, 0] > box_length + tube_length) &
                          (np.sqrt(positions[:num_part1, 1]**2 + positions[:num_part1, 2]**2) < box2_radius))

    return red_in_left/(red_in_left+blue_in_left), blue_in_right/(blue_in_right+red_in_right)
```

# Обновление системы

```
def update_system(dt):
    global positions, velocities
    positions += velocities * dt
    handle_wall_collisions(positions, velocities, num_part1, part1_radius, part2_radius)
    handle_particle_collisions(positions, velocities, num_part1, part1_radius, part2_radius, mass1, mass2)
```

# Функция анимации

```
time_points = []
blue_in_right = []
red_in_left = []
cnt = 0
def animate(i):
    global cnt
    cnt += 1
    update_system(dt)

    scatter._offsets3d = (positions[:, 0], positions[:, 1], positions[:, 2])

    speeds = np.linalg.norm(velocities, axis=1)
    speeds1 = speeds[:num_part1]
    speeds2 = speeds[num_part1:]

    hist1, _ = np.histogram(speeds1, bins=bin_edges)
    hist2, _ = np.histogram(speeds2, bins=bin_edges)

    for rect, h in zip(bar_container1.patches, hist1):
        rect.set_height(h)
    for rect, h in zip(bar_container2.patches, hist2):
        rect.set_height(h)

    time_points.append(cnt*dt)
    red_frac, blue_frac = count_particles(positions)
    red_in_left.append(red_frac)
    blue_in_right.append(blue_frac)

    line_red.set_data(time_points, red_in_left)
    line_blue.set_data(time_points, blue_in_right)

    ax3.set_xlim(0, max(10*dt, len(time_points)*dt)+dt)

    return scatter, *bar_container1.patches, *bar_container2.patches, line_red, line_blue

ani = FuncAnimation(fig, animate, frames=100, interval=10, repeat=True)
```

# Итоги

Полученная модель корректна с точки зрения физических законов и кореллирует с теоретическими сведениями о распределении скоростей и взаимной дифузии двух газов.

Благодарю за внимание!