Бильярдные шары

Выполнили: Ю. Ефремов, А. Баринов, Н. Берников, Е. Сергеев

Научный руководитель: А. С. Байгашов,

Аннотация

Данная работа посвящена моделированию и визуализации столкновения бильярдных шаров. Полученные результаты численного моделирования отражают взаимодействие между реальными физическими телами. Смоделированы процессы столкновения шаров, перераспределения их импульсов и дальнейшего движения по физически обусловленной траектории.

Введение

Моделирование и анализ взаимодействий физических тел является одной из постоянно решаемых задач современных исследований и применяется во многих отраслях науки. В рамках настоящей работы рассматривается взаимодействие бильярдных шаров, которые является двумерной моделью физических шаров на бильярдном столе.

В работе рассматривается ситуация, в которой одиночный шар движется и соударяется с группой статичных шаров, выстроенных на столе в форме треугольника. Таким образом одно из моделируемых тел имеет начальную скорость, в то время как остальные неподвижны. Через некоторый промежуток времени тело, имеющие начальную скорость, сталкивается с неподвижными телами. Вследствие этого группа шаров приобретает скорость, отличную от нуля. Каждый шар приобретают свою скорость и направление, напрямую связанные с заданным расположением и описывающиеся решением дифференциального уравнения движения.

Целью работы является моделирование динамики движения шаров после соударения. Для достижения поставленной цели необходимо решить следующие задачи:

• Определить систему дифференциальных уравнений, описывающую динамику взаимодействия (столкновения) и движение шаров.

• Определить начальные условия для решения системы дифференциальных уравнений.

• Написать алгоритм решения поставленной задачи.

Постановка дифференциальной задачи

Ориентируясь на цель нашего исследования, мы составили систему дифференциальных уравнений, которая описывает изменение положения шаров относительно друг друга со временем:

$$\{\frac{dV}{dt} = \ -\ \mu * m * g * V \ \frac{ds}{dt} = V$$

Где:

μ - коэффициент силы трения качения;

$m * g$ – сила тяжести каждого тела;

V – скорость тела в данный момент времени;

Изменяющейся величиной в системе, является **положение** шаров в системе координат относительно начальной точки (0;0), а **переменной – время**.

Начальные условия

Для решения системы дифференциальных уравнений и построения математических моделей, определим начальные параметры:

· Ускорение свободного падения – 9.8 м/с2

· Масса шаров – 0,5 г

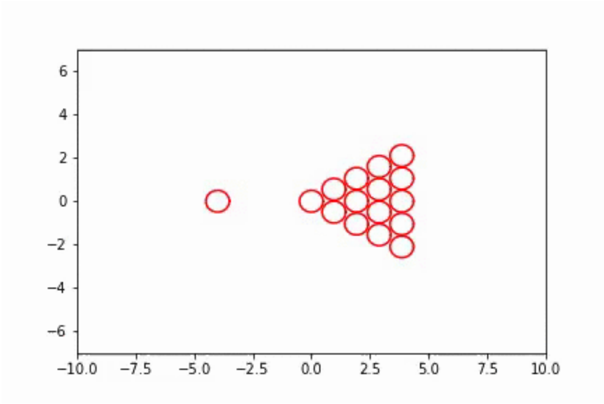
· Радиус шаров – 0,5 дм

· Начальная скорость движущегося шара – 5 м/с

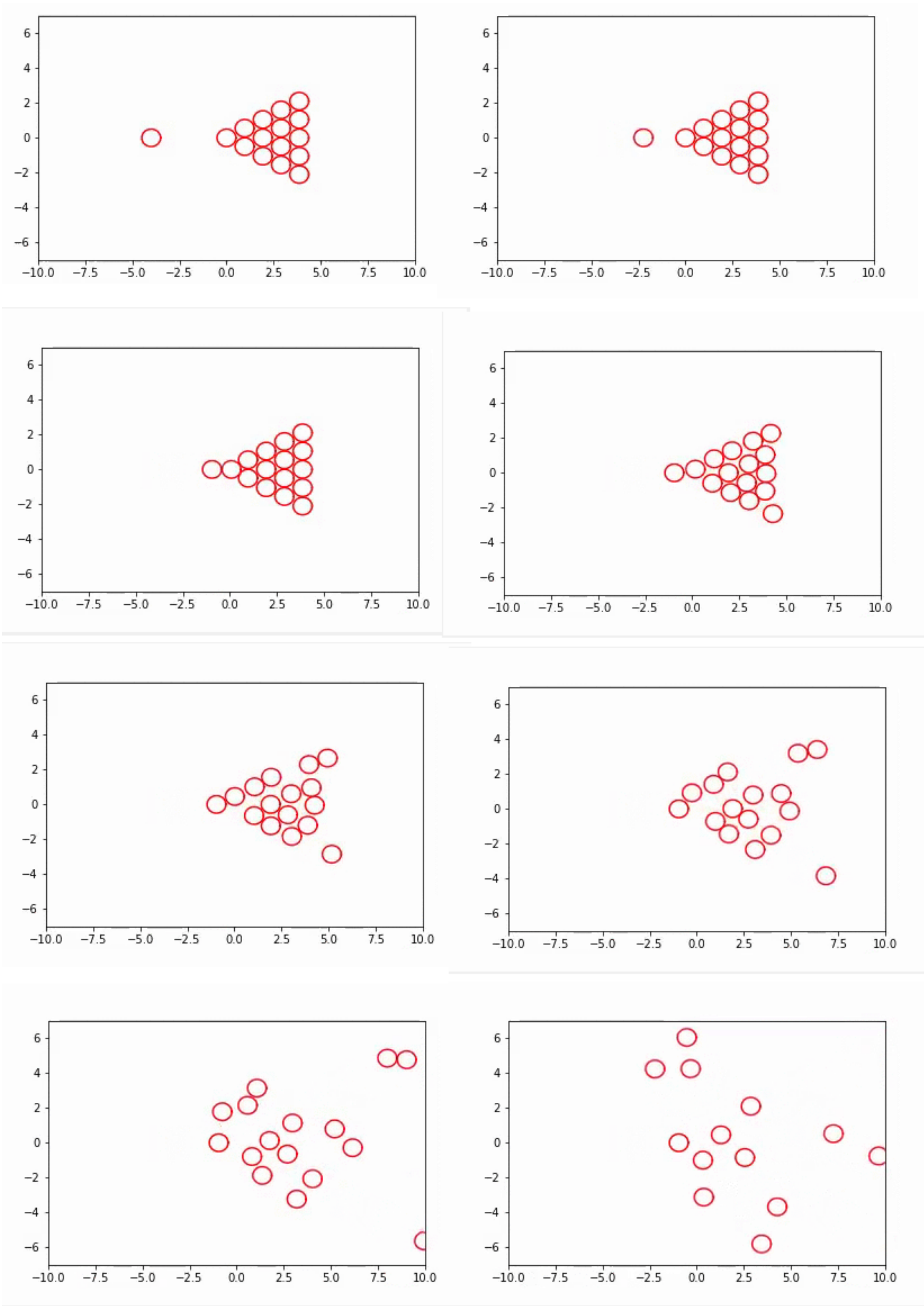
· Количество шаров – 10

Результаты моделирования

Используя среду программирования Python и имеющиеся в ней открытые библиотеки, было проведено моделирование движения шаров после столкновения движущегося шара с неподвижными. Решённые уравнения были визуализированы посредством имеющегося инструментария Python, что позволило получить анимацию движения шаров после столкновения (рис. 1).


6
4
2
0
−2
−4
−6
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

(гиф. Рис.1)


6
4
2
0
−2
−4
−6
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

Заключение

Проведённое исследование наглядно продемонстрировало возможность использования численного моделирования в среде Python для решения различных задач. Несмотря на модельный характер исследования, оно может быть использовано в качестве наглядной иллюстрации работы физических законов. В качестве дальнейшего развития исследования можно рассмотреть моделирование различных более сложных ситуаций – рикошетов шаров от бортиков и друг от друга, столкновения нескольких движущихся шаров и т.д.

Приложение

Листинг кода, написанного на python 3.9:

```
62
63  for k in range(n-1):  # Цикл перебора шагов временеи анимации
64      t = [tau[k],tau[k+1]]
65
66      for i in range(N): # Базовая частица
67          s0 = p[i,0], p[i,2], p[i,1], p[i,3]
68          sol = odeint(move_func, s0, t)
69
70          # Перезаписываем положения частиц
71          p[i,0] = sol[1,0]
72          p[i,2] = sol[1,1]
73          p[i,1] = sol[1,2]
74          p[i,3] = sol[1,3]
75
76          # Заноим новые положения в итоговый массив для анимации
77          x[i,k+1], y[i,k+1] = p[i,0], p[i,1]
78
79          x1, y1, vx1, vy1 = p[i,0], p[i,1], p[i,2], p[i,3] # Запись текущих координат базовой частицы
80          x10, y10 = x[i,k], y[i,k] # Запись координат предыдущего шага базовой частицы
81
82          for j in range(0, N): # Запись текущих координат остальных частиц
83              if j == i:
84                  pass
85              else:
86                  x2, y2, vx2, vy2 = p[j,0], p[j,1], p[j,2], p[j,3] # Запись текущих
87                  x20, y20 = x[j,k], y[j,k] # Запись координат предыдущего шага
88
89                  # Проверка условий столкновения
90                  r1 = np.sqrt((x1-x2)**2+(y1-y2)**2)
91                  r0 = np.sqrt((x10-x20)**2+(y10-y20)**2)
92                  if  (r1 <= radius*2 and r0 > 2*radius) or (r1 <= radius*2 - radius * 0.1 and r0 > 2*radius - radius * 0.1):
93                      #Условие столкновение с доп. условием после or (на всякий случай)
94                      res = collision(x1,y1,vx1,vy1,x2,y2,vx2,vy2,radius,mass,mass,K)
95
96                  # Перезаписывание условий, в случае столкновения
97                      p[i,2], p[i,3] = res[0], res[1]
98                      p[j,2], p[j,3] = res[2], res[3]
99
100 # Графический вывод
101 fig = plt.figure()
102
103 bodys = []
104 O = 100 # Функция возращающая сложные объекты (шар) для анимации
105 def ball(x_centre, y_centre, radius = radius, O = O):
106     """Функция анимируемых объектов
107     """
108     x = np.zeros(O)
109     y = np.zeros(O)
110     for i in range(0, O, 1):
111         alpha = np.linspace(0, 2*np.pi, O)
112         x[i] = x_centre + radius*np.cos(alpha[i])
113         y[i] = y_centre + radius*np.sin(alpha[i])
114     return x, y
115
116 for i in range(n): # Добавление объектов для анимации
117     body1, = plt.plot(*ball(x[0, i], y[0, i]), '-', color='r', ms=2)
118     body2, = plt.plot(*ball(x[1, i], y[1, i]), '-', color='r', ms=2)
119     body3, = plt.plot(*ball(x[2, i], y[2, i]), '-', color='r', ms=2)
120     body4, = plt.plot(*ball(x[3, i], y[3, i]), '-', color='r', ms=2)
121     body5, = plt.plot(*ball(x[4, i], y[4, i]), '-', color='r', ms=2)
122     body6, = plt.plot(*ball(x[5, i], y[5, i]), '-', color='r', ms=2)
123     body7, = plt.plot(*ball(x[6, i], y[6, i]), '-', color='r', ms=2)
```

```
62
63 for k in range(n-1):  # Цикл перебора шагов временеи анимации
64     t = [tau[k],tau[k+1]]
65
66     for i in range(N): # Базовая частица
67         s0 = p[i,0], p[i,2], p[i,1], p[i,3]
68         sol = odeint(move_func, s0, t)
69
70         # Перезаписываем положения частиц
71         p[i,0] = sol[1,0]
72         p[i,2] = sol[1,1]
73         p[i,1] = sol[1,2]
74         p[i,3] = sol[1,3]
75
76         # Заноим новые положения в итоговый массив для анимации
77         x[i,k+1], y[i,k+1] = p[i,0], p[i,1]
78
79         x1, y1, vx1, vy1 = p[i,0], p[i,1], p[i,2], p[i,3] # Запись текущих координат базовой частицы
80         x10, y10 = x[i,k], y[i,k] # Запись координат предыдущего шага базовой частицы
81
82         for j in range(0, N): # Запись текущих координат остальных частиц
83             if j == i:
84                 pass
85             else:
86                 x2, y2, vx2, vy2 = p[j,0], p[j,1], p[j,2], p[j,3] # Запись текущих
87                 x20, y20 = x[j,k], y[j,k] # Запись координат предыдущего шага
88
89                 # Проверка условий столкновения
90                 r1 = np.sqrt((x1-x2)**2+(y1-y2)**2)
91                 r0 = np.sqrt((x10-x20)**2+(y10-y20)**2)
92                 if  (r1 <= radius*2 and r0 > 2*radius) or (r1 <= radius*2 - radius * 0.1 and r0 > 2*radius - radius * 0.1):
93                     #Условие столкновение с доп. условием после or (на всякий случай)
94                     res = collision(x1,y1,vx1,vy1,x2,y2,vx2,vy2,radius,mass,mass,K)
95
96                 # Перезаписывание условий, в случае столкновения
97                     p[i,2], p[i,3] = res[0], res[1]
98                     p[j,2], p[j,3] = res[2], res[3]
99
100 # Графический вывод
101 fig = plt.figure()
102
103 bodys = []
104 O = 100 # Функция возращающая сложные объекты (шар) для анимации
105 def ball(x_centre, y_centre, radius = radius, O = O):
106     """Функция анимируемых объектов
107     """
108     x = np.zeros(O)
109     y = np.zeros(O)
110     for i in range(0, O, 1):
111         alpha = np.linspace(0, 2*np.pi, O)
112         x[i] = x_centre + radius*np.cos(alpha[i])
113         y[i] = y_centre + radius*np.sin(alpha[i])
114     return x, y
115
116 for i in range(n): # Добавление объектов для анимации
117     body1, = plt.plot(*ball(x[0, i], y[0, i]), '-', color='r', ms=2)
118     body2, = plt.plot(*ball(x[1, i], y[1, i]), '-', color='r', ms=2)
119     body3, = plt.plot(*ball(x[2, i], y[2, i]), '-', color='r', ms=2)
120     body4, = plt.plot(*ball(x[3, i], y[3, i]), '-', color='r', ms=2)
121     body5, = plt.plot(*ball(x[4, i], y[4, i]), '-', color='r', ms=2)
122     body6, = plt.plot(*ball(x[5, i], y[5, i]), '-', color='r', ms=2)
123     body7, = plt.plot(*ball(x[6, i], y[6, i]), '-', color='r', ms=2)
```

```
124     body8, = plt.plot(*ball(x[7, i], y[7, i]), '-', color='r', ms=2)
125     body9, = plt.plot(*ball(x[8, i], y[8, i]), '-', color='r', ms=2)
126     body10, = plt.plot(*ball(x[9, i], y[9, i]), '-', color='r', ms=2)
127     body11, = plt.plot(*ball(x[10, i], y[10, i]), '-', color='r', ms=2)
128     body12, = plt.plot(*ball(x[11, i], y[11, i]), '-', color='r', ms=2)
129     body13, = plt.plot(*ball(x[12, i], y[12, i]), '-', color='r', ms=2)
130     body14, = plt.plot(*ball(x[13, i], y[13, i]), '-', color='r', ms=2)
131     body15, = plt.plot(*ball(x[14, i], y[14, i]), '-', color='r', ms=2)
132     body16, = plt.plot(*ball(x[15, i], y[15, i]), '-', color='r', ms=2)
133     bodys.append([body1,body2,body3,body4,body5,body6,body7,body8,body9,body10,body11,body12,body13,body14,body15,body16])
134
135 ani = ArtistAnimation(fig, bodys, interval=25)
136
137 plt.xlim(-10, 10)
138 plt.ylim(-7, 7)
139 ani.save("billiard.gif")
```

billiardProject.py