Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
**"Национальный исследовательский университет
Высшая школа экономики"**
Департамент прикладной математики, бакалавр

Работу на тему:

Моделирование столкновения шаров внутри замкнутой поверхности, обадающей свойством замкнуости.

Выполнил:
Колодин Матвей Алексеевич

Академический руководитель:
Щур Лев Николаевич

Москва 2022

Содержание

1 Введение

1.1 Постановка задачи

Прежде всего, необходимо четко сформулировать условие задачи, решение которой будет представлено ниже. Прежде всего, важно сказать, что мы будем представлять тор, как его развертку на плоскость - в частном случае квадрат некоторых размеров. Перейдем к условию:

Пусть дано вещественное число L, которое определяет линейные размеры квадрата. Вещественное число R отвечает за радиус шара, а N - является количеством шаров, помещенных в данный квадрат. А также велична $\eta = \frac{N*\Pi*R^2}{L^2}$ которая находится в промежутке $0.5 < \eta < 1$.

Из коробки выбирается случайный шар и начинает свое движение, параллельно основанию коробки. Если на его пути встречается другой шар, то происходит столкновение, в результате чего движущийся шар останавливается, а тот шар, с которым он столкнулся продолжает движение. Если шар достигает границы квадрата, то он начинает "появляться"из начала коробки (в этом и заключается свойство тора, которое применяется в задаче).

Движение заканчивается в тот момент, когда суммарное расстояние, пройденное шарами становится равно l. После чего алгоритм повторяется по оси, направленной параллельно боковой стороне коробки.

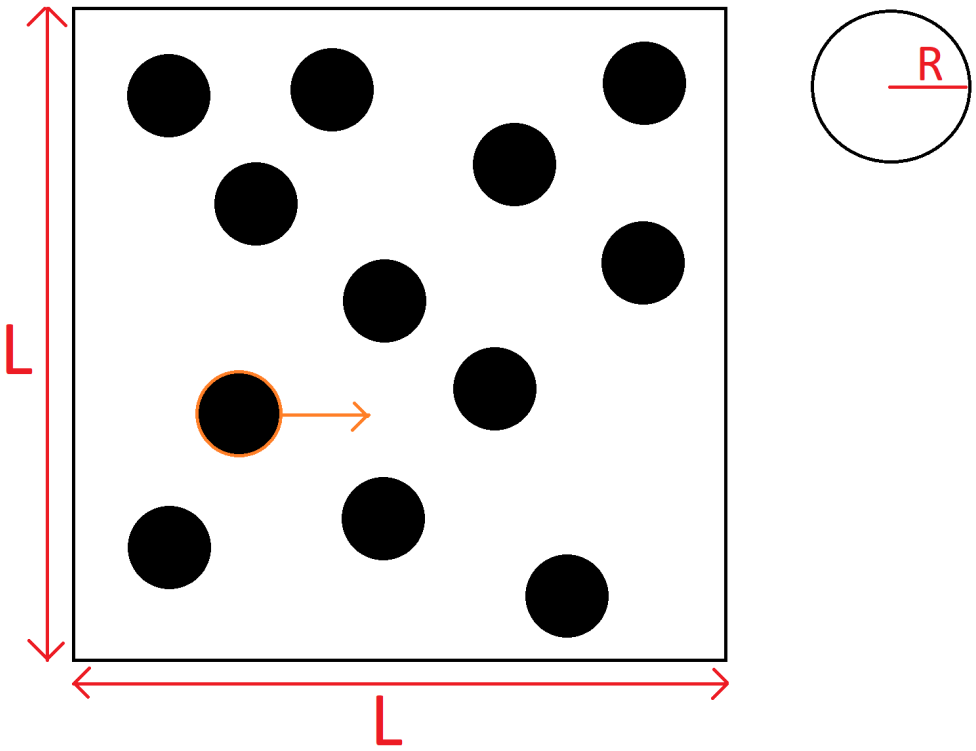

Рис. 1: Пояснительный рисунок к условию задачи.

1.2 Уточнения по условию задачи

Давайте обговорим два момента, связанных с условием задачи:

1. Случай столкновения с двумя шарами. Давайте условимся считать, что в случае, когда движущийся шар сталкивается с двумя шарами одновременно, движние продолжает любой из них.

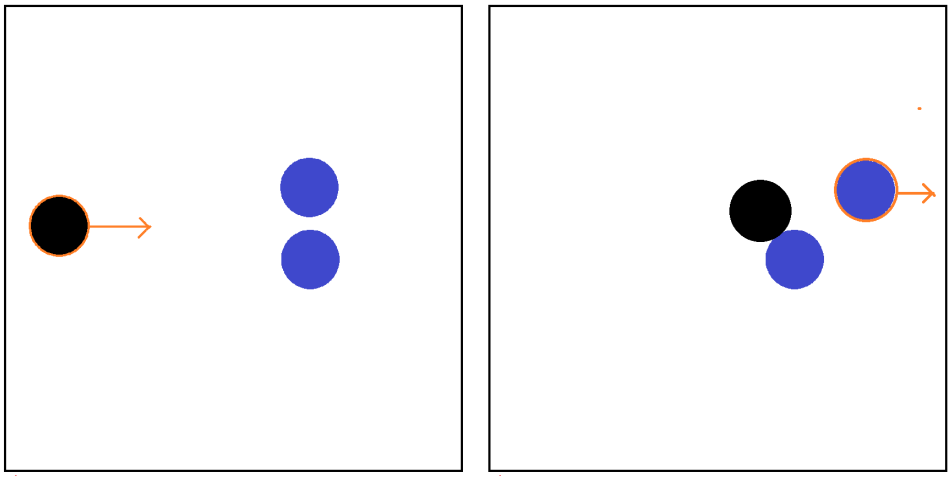

Рис. 2: Случай столкновения с двумя шарами.

2. Случай, если расстояние от центра шара до границы $< R$. В таком случае, будем визуализировать картинку, опираясь на свойства тора, о которых мы сказали выше, таким образом:

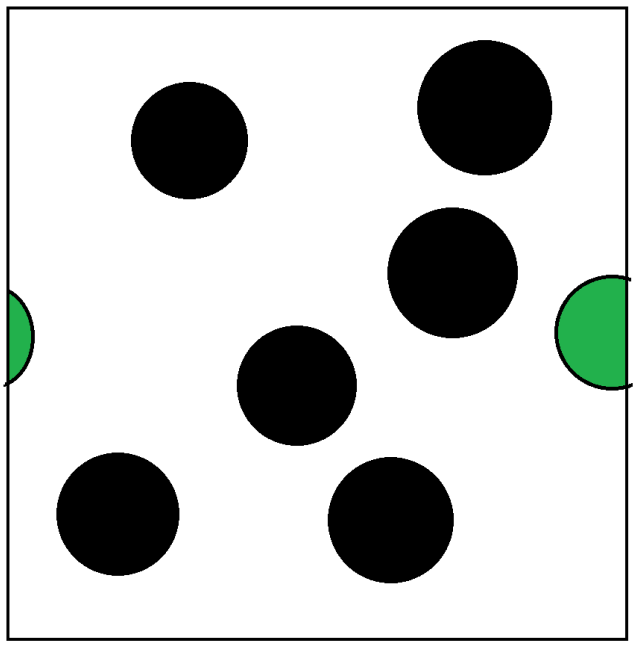

Рис. 3: Случай нахождения шара на границе.

3. Также важным будет следующее замечание: поскольку вещественные числа представлены в компьютере с некоторой точностью, то необходимо аккурат-

но работать с математическими операциями. Для этого введем константу ϵ, применение которой будет показано далее.

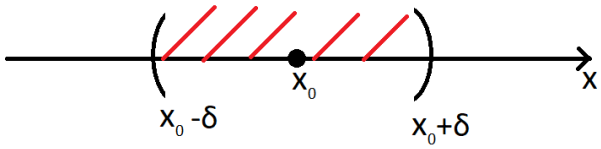

Рис. 4: Проблема с точностью при вычислении одних и тех же выражений. Полученное значение будет лежать в некоторой дельта-окрестности.

2 Алгоритм решения задачи

2.1 Движение шара параллельно основанию квадрата (O_x). Поиск следующего шара в цепочке столкновений

Прежде всего данную подзадачу разобьем на два пункта - поиск шара, с которым произойдет следующее столкновение и вычисление промежуточных положений центров шаров.

Поиск следующего шара в цепочке столкновений. Прежде всего давайте рассмотрим, где может находится шар, с которым возможно столкнуться. Нетрудно понять, что если разность координат центров произволльного шара и выбранного по O_y отличается не более, чем на $2R$, то такие шары столкнуться, в противном случае, при движении параллельно O_x шары столкнуться не могут впринципе.

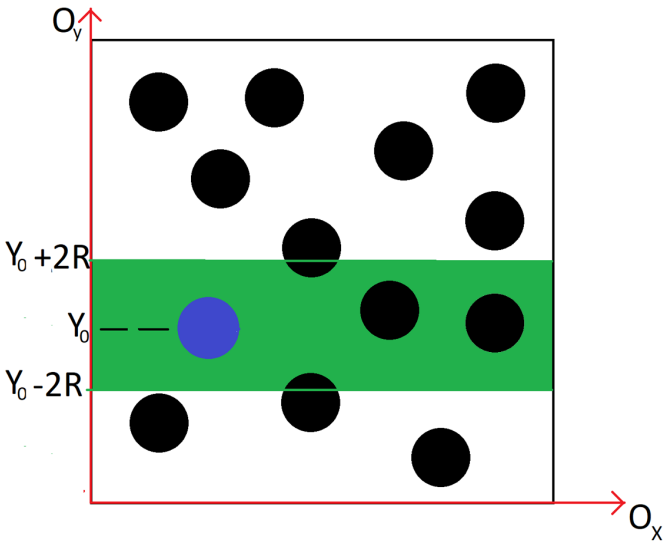

Рис. 5: Шары, чей центр (относительного выбранного синего шара) находится на расстоянии меньше $2R$ подойдут. На данной картинке подойдет всего два шара.

Мы получили список шаров, с которыми мы можем столкнуться, теперь необходимо понять, что произойдет столкновение с тем шаром, который ближе всего находится в синему, относительно O_x, и при этом, лежащему правее него, таким образом:

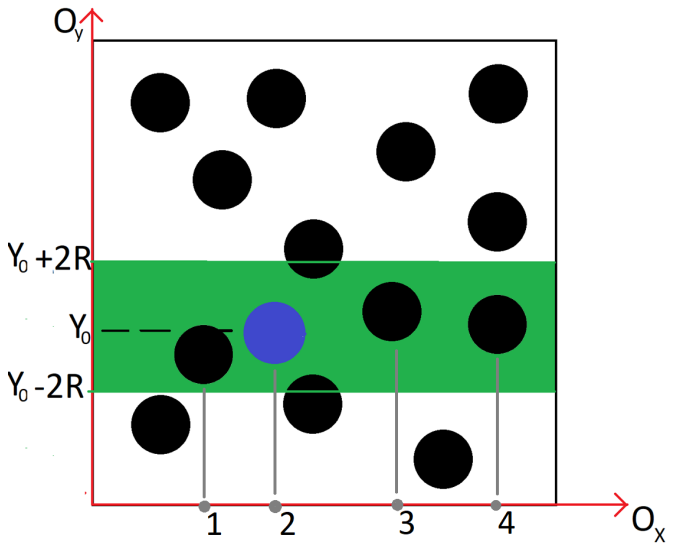

Рис. 6: Можно видеть, что первый шар лежит ближе ко второму, но он находится левее него, таким образом он не подойдет. Остается шар под номером 3.

Таким образом можем получить следующий код (язык - Python + библиотека NumPy):

```
def FNB(x, y, arr, r):
    eps = 1e-8
    minn = 1e+9
    pos = -1
    for i in range(len(arr)):
        if arr[i][1] > y + 2 * r - eps:
            break
        elif arr[i][1] > y - 2 * r + eps:
            if minn - eps > arr[i][0] > x + eps:
                minn = arr[i][0]
                pos = i
    return pos
```

В алгоритме мы в лоб ищем шары, которые будут лежать в полосе (по O_y) от $y - 2 * r$ до $y + 2 * r$

2.2 Движение шара параллельно основанию квадрата. Моделирование движения

По большому счету, моделирование движения шаров сводится к разбору ряда случаев. Но перед тем, как перейти к разбору вариантов, уточним одну вещь: мы будем создавать некоторый буфер (ширины $2R$, а высотой будет высота квадрата). Этот буфер присоединим в конец квадрата. В нем будет отображено начало квадрата (прямоугольник размера $n * 2R$. Данная процедура будет необходима для удобной работы при моделировании движения молекул.

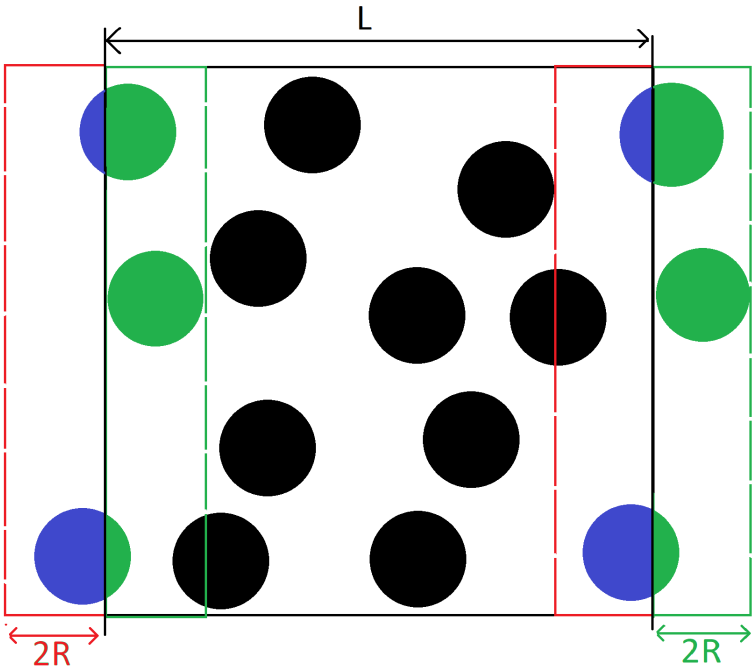

Рис. 7: Визуализация буфера для квадрата.

Для лучшего понимания, будем размечать еще одну зону, симметричную относительно стенки квадрата буферной зоне. Цветом выделены те шары, которые начали процесс перетекания, то есть коснулись стенки квадрата. Такое представление необходимо для того, чтобы точно понимать, когда шар закончил перемещение через границу и оказался в начале квадрата (такое визуальное представление облегчает восприятие. Также часть кода основана на принципе аккуратной работы с буфером.)

В результате чего остается аккуратно разобрать случаи, при движении шара - в частности можно выделить две большие группы ситуаций по столкновению с одним из объектов: с другим шаром или стенкой квадрата.

Мы умеем находить ближайший шар, теперь нужно научиться моделировать его движение, до столкновения с другим шаром. Таким образом стоит задача о перемещении шара, до момента касания с другим шаром (шаром столкновения). В решении данной проблемы нам поможет курс школьной геометрии и алгебры:

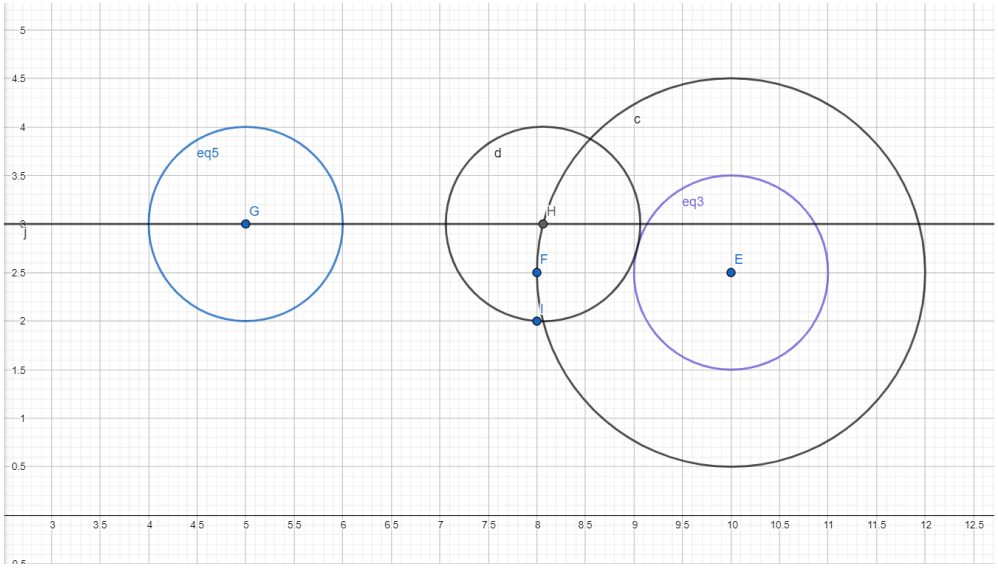

Рис. 8: Моделирование столкновения двух шаров.

Пусть есть два шара с центрами G (имеет координаты (x_1, y_1)) и E (имеет координаты (x_2, y_2)) соответственно. Очевидно, что центр шара G движется по прямой $y = y_1$. Встает вопрос - в какой момент столкнуться шары? Ответ крайне прост - когда расстояние между их центрами станет равно $2R$. Таким образом, чтобы шары столкнулись, необходимо, чтобы центр шара G находился на окружности с центром в точке E и радиусом $2R$. Пусть при движении в точку столкновения центр шара G изменил значения своих координат с (x_1, y_1) на (x_{new}, y_{new}). Тогда можно сказать следующее:

$$\begin{cases} y_{new} = y_1 \\ (x - x_2)^2 + (y - y_2)^2 = (2R)^2 \end{cases}$$

Поскольку значение y_{new} мы знаем, то чтобы получить x_{new} остается решить квадратное уравнение. Тогда получаем: $x = \pm((4R^2 - (y_1 - y_2)^2)^{\frac{1}{2}}) + x_2$. А по скольку движение идет слева от шара E, то из полученых корней необходимо брать меньшее значение. Таким образом: $x_{new} = -((4R^2 - (y_1 - y_2)^2)^{\frac{1}{2}}) + x_2$. А значит новые координаты центра шара G были получены.

Объвим функцию CCB, которая и будет обрабатывать процесс движения шаров:

```
def CCB(x1, y1, x2, y2, r):
    x_new = -(((2 * r) ** 2 - (y1 - y2) ** 2) ** (1 / 2)) + x2
    l_pr = x_new - x1
    return [x_new, l_pr]
```

Она будет возрващать новую координату по O_x и расстояние, которое шар пройдет до столкновения.

Теперь же перейдем к функции, которая и будет просчитывать движение всех шаров:

```
def MOB_x(ind_sh, arr, r, l_ost, l_box):

# Прежде всего, необходимо понять, находится ли шар в буферной зоне
или нет.
eps = 1e-8
x = arr[ind_sh][0]
y = arr[ind_sh][1]

if x < l_box - r - eps:
    # Мы находимся внутри коробки.
    pos = FNB(x, y, arr, r)

    if pos == -1:
        # Значит шаров, с которыми можно столкнуться внутри
        коробки нет.
        if l_ost > (l_box - r) - x + eps:
            l_ost -= (l_box - r) - x
            arr[ind_sh][0] = l_box - r
            return MOB_x(ind_sh, arr, r, l_ost, l_box)
        else:
            arr[ind_sh][0] += l_ost
            return arr

    else:
        # Значит есть шар, с которым можно столкнуться.
        x_new, l_pr = CCB(x, y, arr[pos][0], arr[pos][1], r)
        if l_ost > l_pr + eps:
            l_ost -= l_pr
            arr[ind_sh][0] += l_pr
            return MOB_x(pos, arr, r, l_ost, l_box)
        else:
            arr[ind_sh][0] += l_ost
            return arr

else:
    # Мы находимся внутри буферной зоны или на ее границе.
    # Важным уточнением является то, что мы ввели буферную зону, но
    не заполняли ее. Поэтому важна отдельная обработка поиска шаров
    для начала коробки, так как в начальный момент, например,
    в буфере ничего нет, но вот в начале шар может быть, это
    необходимо учитывать.
    pos_buf = FNB(x, y, arr, r)
    pos_strt = FNB(x - l_box, y, arr, r)
```

```
# Для начала проверим, есть ли шары, с которыми мы можем
столкнуться в буферной зоне.
if pos_buf != -1:
    # Таким образом, есть шар, с которым мы столкнемся в
    буферной зоне.
    pos = pos_buf
    x_new, l_pr = CCB(x, y, arr[pos][0], arr[pos][1], r)
    if l_ost > l_pr + eps:
        l_ost -= l_pr
        arr[ind_sh][0] += l_pr
        return MOB_x(pos, arr, r, l_ost, l_box)
    else:
        arr[ind_sh][0] += l_ost
        return arr

else:
    # То есть буферная полоса пустая. По крайней мере, если
    "смотреть" с конца, давайте рассмотрим ситуацию
    от левого края коробки (левой буферной зоны).
    if pos_strt != -1 and
    np.array_equal(arr[pos_strt], np.array(x, y)) == False:
        # Значит мы нашли шар, с которым мы столкнемся, причем
        он лежит внутри коробки, иначе мы бы нашли его при
        просмотре буферной зоны.
        # Здесь необходимо работать аккуратно и, в случае
        если шар прошел буферную зону перевести его в
        обычную зону.
        pos = pos_strt
        x_new, l_pr = CCB(x - l_box, y, arr[pos][0], arr[pos][1], r)
        if l_ost > l_pr + eps:
            l_ost -= l_pr
            arr[ind_sh][0] += l_pr
            if arr[ind_sh][0] > (l_box + r) - eps:
                arr[ind_sh][0] -= (l_box + r)
            return MOB_x(pos, arr, r, l_ost, l_box)
        else:
            arr[ind_sh][0] += l_ost
            if arr[ind_sh][0] > (l_box + r) - eps:
                arr[ind_sh][0] -= (l_box + r)
            return arr

    else:
        # Значит шаров для столкновения, тогда двигаем настолько,
        насколько можем.
```

```
        arr[ind_sh][0] += l_ost
        if arr[ind_sh][0] >= (l_box + r) - eps:
            arr[ind_sh][0] -= (l_box + r)
        return arr
```

P.S. В конце отчета будет приведен итоговый код без комментариев, для удобства.

Таким образом, была получена функция, которая моделирует движение шаров, параллельно O_x.

2.3 Движение шара параллельно высоте квадрата (O_y)

В действительности же, моделирование, при движении параллельно высоте квадрата ничем не отличается от движения параллельно его основанию. Таким образом, код для данного случая уже получен (остается лишь в полученные функции передать "перевернутый массив"(и отсортированный), то есть заменить координаты: x на y. И после того, как мы прогоним массив, еще раз его перевернуть, тем самым мы вернемся в систему O_x O_y). Теперь перейдем к финальной части!

2.4 Нормализация координат и вывод результатов

Итоговый алгоритм будет выглядеть следующим образом:

```
a = np.array([[1,1], [4,1], [10, 2.5], [3,3], [5,3], [1,4], [3,5]])
a = a[np.argsort(a[:, 1])]
shx = 0
shy = 1

a_res_x = MOB_x(shx, a, 1, 7, 11)
copy_x = a_res_x.copy()
for i in range(len(a_res_x)):
    a_res_x[i] = np.array([a_res_x[i][1], a_res_x[i][0]])
a_res_x = a_res_x[np.argsort(a_res_x[:, 1])]



a_res_xy = MOB_x(shy, a_res_x, 1, 5, 11)
for i in range(len(a_res_xy)):
    a_res_xy[i] = np.array([a_res_xy[i][1], a_res_xy[i][0]])

a_res_x = copy_x
```

В итоге мы получим два массива: $a_r es_x$ - который содержит новые координаты центра шаров после движения параллельно O_x, а также $a_r es_x y$ - который содержит новые координаты центра шаров после движения параллельно O_x и O_y.

Для вывода начального, промежуточного и конечного положения шаров я буду пользоваться библиотекой Matplotlib:

```
import matplotlib.pyplot as plt
%matplotlib inline
l_box = 11
r = 1
a = np.array([[1,1], [4,1], [10, 2.5], [3,3], [5,3], [1,4], [3,5]])
a = a[np.argsort(a[:, 1])]

fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, figsize=(15,15))

ax1.set(xlim=(0, l_box), ylim=(0, l_box))
for i in range(len(a)):
    circle = plt.Circle((a[i][0] % l_box, a[i][1] % l_box), r,
    color='b')
    ax1.add_patch(circle)
ax1.set_aspect('equal')

ax2.set(xlim=(0, l_box), ylim=(0, l_box))
for i in range(len(a_res_x)):
    circle = plt.Circle((a_res_x[i][0] % l_box, a_res_x[i][1] % l_box)
    r, color='b')
    ax2.add_patch(circle)
ax2.set_aspect('equal')

ax3.set(xlim=(0, l_box), ylim=(0, l_box))
for i in range(len(a_res_xy)):
    circle = plt.Circle((a_res_xy[i][0] % l_box, a_res_xy[i][1] %
    l_box), r, color='b')
    ax3.add_patch(circle)
ax3.set_aspect('equal')
```

При отрисовке шаров координаты берутся по модулю l_{box}, поскольку при работе алгоритма мы позволяли центрам шаров будто оказываться за границей, но на деле же, их центр всегда пренадлежит квадрату.

3 Результаты

В качестве результатов рассмотрим один пример движения шаров:

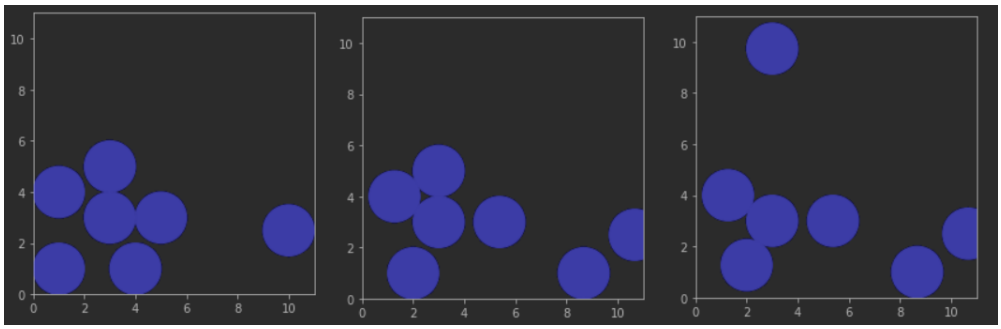

Рис. 9: Пример работы алгоритма

3.1 Приложение

Ссылка на проект: https://github.com/HpPpL/Collision-of-balls.git.