Постановка задачи

Написать программу для численного нахождения цены игры поиска на параллелепипеде.

Начальные данные:

- a, b, c — размеры параллелепипеда;
- e — радиус окрестности точек ищущего игрока;
- games_number — число игровых итераций;
- dots_number — число точек ищущего игрока.

Ход работы

Для моделирования игры поиска на параллелепипеде был применен следующий алгоритм:

1. На эллипсоиде, вписанном в параллелепипед, по методу Фибоначчи случайным образом (в каждой итерации игры задается случайное начальное смещение) генерируются точки ищущего игрока. Таким образом, достигается равномерность распределения точек. Точка прячущегося игрока тоже случайным образом генерируется на эллипсоиде.
2. Точки проецируются на стороны параллелепипеда. Координата каждой точки вычисляется решением системы, составленной из уравнений плоскости и прямой, пересекающей эту плоскость.
3. Проверяется, находится ли точка прячущегося игрока в окрестности хотя бы одной точки ищущего игрока. Если да, то побеждает ищущий игрок с результатом 1, иначе — прячущийся игрок с результатом -1.

Алгоритм повторяется для заданного числа игр. Цена игры определяется как усредненное значение всех выигрышей.

Листинг программы представлен в приложении А.

Рассмотрим результат моделирования при следующих начальных данных:

- a = 12, b = 16, c = 24;
- e = 0,5;
- games_number = 1000;
- dots_number = 100.

На рисунке 1 показано расположение точек на эллипсоиде, вписанном в параллелепипед. Здесь и на других рисунках точки ищущего игрока вместе с окрестностями имеют вид

голубых сфер радиуса e. Для наглядности точка прячущегося игрока так же имеет вид сферы радиуса e и окрашена в оранжевый цвет, однако в вычислениях она окрестности не имеет.

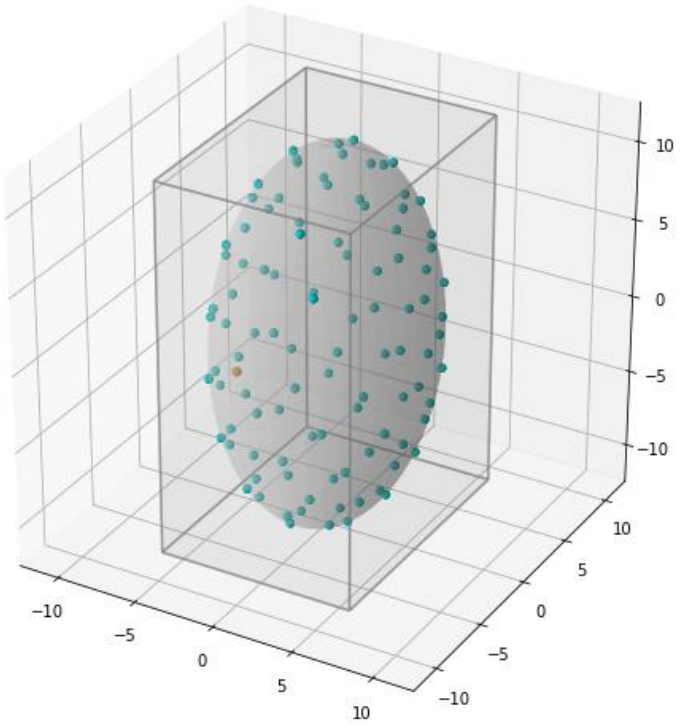

Рисунок 1 — Расположение точек на эллипсоиде при e=0,5

На рисунке 2 показан результат проецирования точек на параллелепипед.

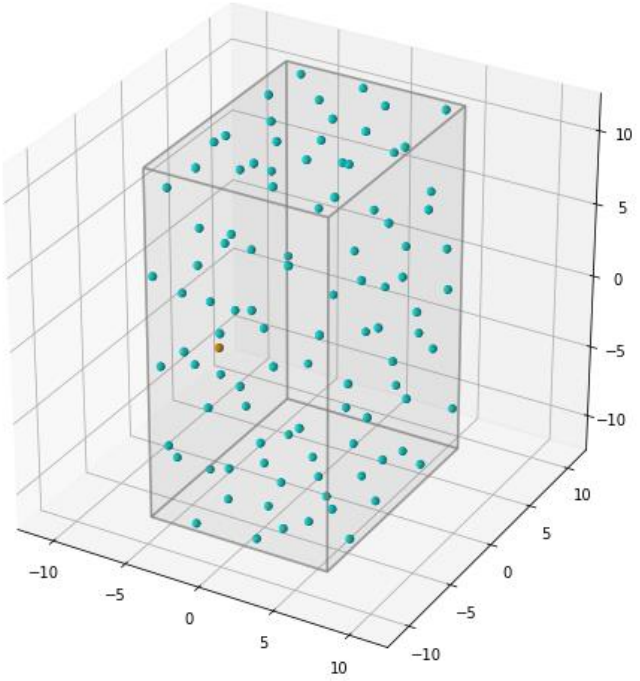

Рисунок 2 — Расположение точек на параллелепипеде при e=0,5

Цена игры после 1000 итераций равна -0,898.

Рассмотрим теперь пример с большим значением e-окрестности, равным 1,5. Цена игры должна увеличиться в пользу ищущего игрока.

На рисунках 3 и 4 показано расположение точек игроков при таком значении окрестности на эллипсоиде и параллелепипеде соответственно.

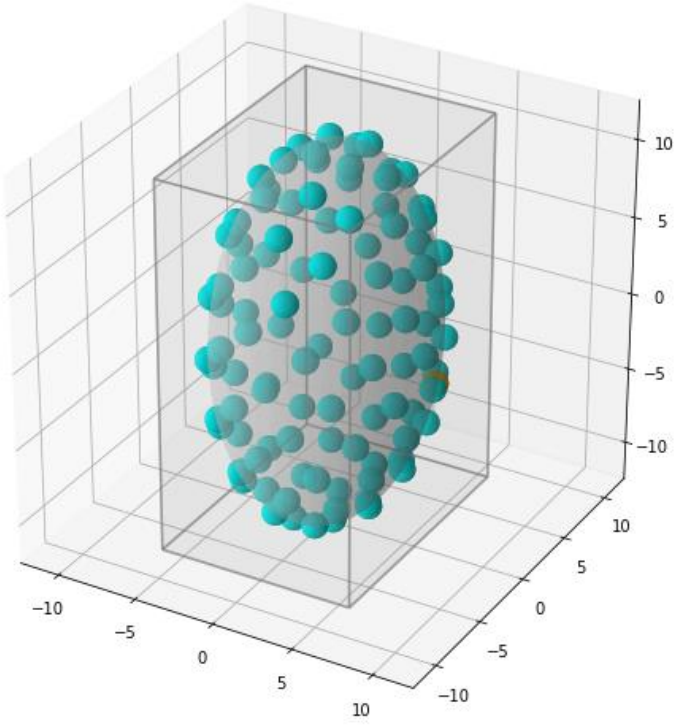

Рисунок 3 — Расположение точек на эллипсоиде при e=1,5

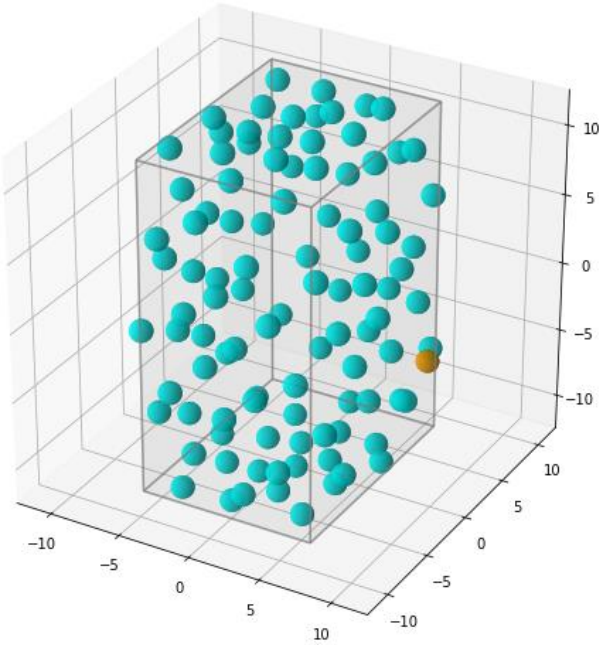

Рисунок 4 — Расположение точек на параллелепипеде при e=1,5

Действительно, цена игры увеличилась и после 1000 итераций стала равна -0,012.

Наконец, рассмотрим пример со значением e-окрестности, равным 10, при котором обеспечивается полное покрытие параллелепипеда. Точка прячущегося игрока гарантированно должна быть найдена.

На рисунках 5 и 6 показано расположение точек игроков при таком значении окрестности на эллипсоиде и параллелепипеде соответственно.

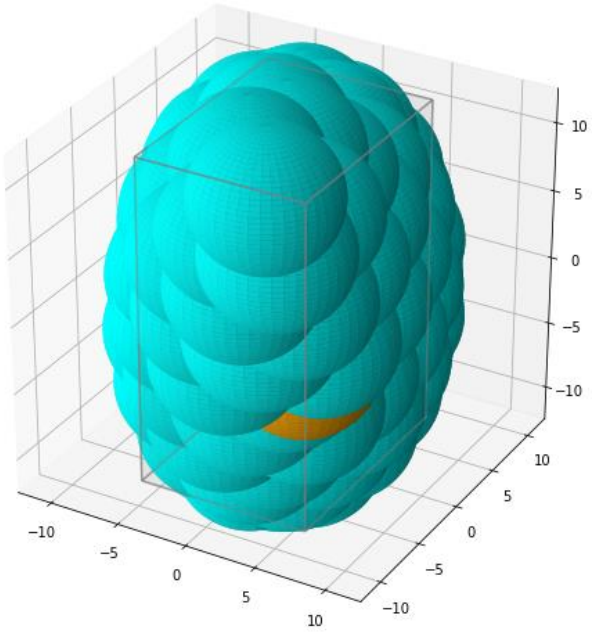

Рисунок 5 — Расположение точек на эллипсоиде при e=10

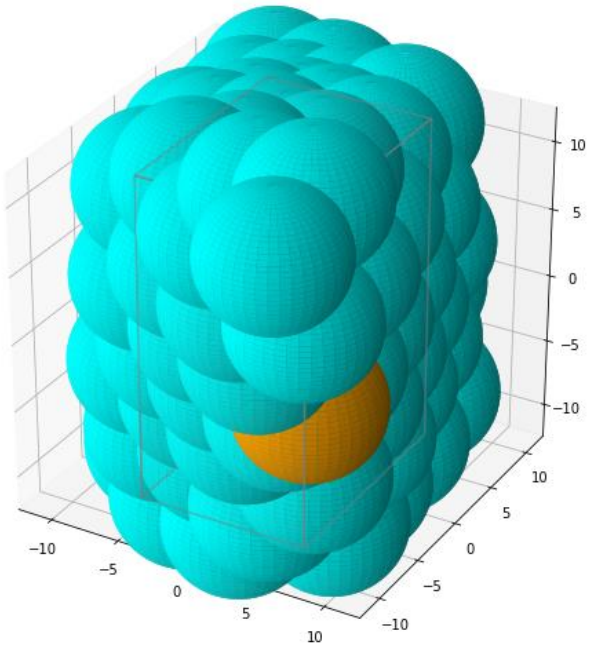

Рисунок 6 — Расположение точек на параллелепипеде при e=10

Цена игры после 1000 итераций равна 1, что соответствует ожидаемому результату.

Вывод

В ходе выполнения рубежного контроля была разработана программа для моделирования игр поиска на параллелепипеде. Равномерное случайное распределение точек на параллелепипеде обеспечивается проекцией этих точек с эллипсоида, на котором они генерируются методом Фибоначчи.

Было проведено моделирование игры при различных значениях окрестности точек ищущего игрока. С увеличением значения окрестности цена игры так же увеличивается в пользу ищущего игрока. Таким образом, результаты моделирования совпали с теоретически ожидаемыми.

ПРИЛОЖЕНИЕ А

Листинг программы на языке Python

```
from mpl_toolkits.mplot3d import Axes3D
from scipy.spatial import distance
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random


def generate_dots(dots_number, a, b, c):
    """Равномерно сгененировать точки на эллипсоиде"""

    bias = random.uniform(0, 1)
    indices = np.arange(0, dots_number, dtype=float) + bias
    phi = np.arccos(1 - 2*indices/dots_number)
    theta = np.pi * (1 + 5**0.5) * indices

    # перевод в декартову систему координат:
    x = a * np.cos(theta) * np.sin(phi)
    y = b * np.sin(theta) * np.sin(phi)
    z = c * np.cos(phi)

    dots = [(x[i],y[i],z[i]) for i in range(dots_number)]
    return dots


def draw_ellipsoid(a, b, c, ax, color, alpha=1):
    """Нарисовать эллипсоид"""

    phi = np.linspace(0,2*np.pi, 256).reshape(256, 1)
    theta = np.linspace(0, np.pi, 256).reshape(-1, 256)

    # перевод в декартову систему координат:
    x = a * np.sin(theta) * np.cos(phi)
    y = b * np.sin(theta) * np.sin(phi)
    z = c * np.cos(theta)

    ax.plot_surface(x, y, z, color=color, alpha=alpha)


def draw_sphere(dot, e, ax, color, alpha):
    """Нарисовать сферу"""

    phi = np.linspace(0,2*np.pi, 256).reshape(256, 1)
```

```
    theta = np.linspace(0, np.pi, 256).reshape(-1, 256)

    # перевод в декартову систему координат:
    x = e/2 * np.sin(theta) * np.cos(phi) + dot[0]
    y = e/2 * np.sin(theta) * np.sin(phi) + dot[1]
    z = e/2 * np.cos(theta) + dot[2]

    ax.plot_surface(x, y, z, color=color, alpha=alpha)


def draw_dots(dots, e, ax, color, alpha=1):
    """Нарисовать точки"""

    for dot in dots:
        draw_sphere(dot, e, ax, color, alpha)


def draw_parallelepiped(a, b, c, ax, color, wireframe_alpha=1, sides_alpha=1):
    """Нарисовать параллелепипед"""

    # генерация массивов точек для последующей отрисовки
    x = np.linspace(-a/2, a/2, 2)
    y = np.linspace(-b/2, b/2, 2)
    z = np.linspace(-c/2, c/2, 2)

    xy_grid, yx_grid = np.meshgrid(x, y)
    xz_grid, zx_grid = np.meshgrid(x, z)
    yz_grid, zy_grid = np.meshgrid(y, z)

    x = np.full(xy_grid.shape, a/2)
    y = np.full(xy_grid.shape, b/2)
    z = np.full(xy_grid.shape, c/2)

    # отрисовка ребер
    ax.plot_wireframe(x, yz_grid, zy_grid, color=color, alpha=wireframe_alpha)
    ax.plot_wireframe(-x, yz_grid, zy_grid, color=color, alpha=wireframe_alpha)
    ax.plot_wireframe(xz_grid, y, zx_grid, color=color, alpha=wireframe_alpha)
    ax.plot_wireframe(xz_grid, -y, zx_grid, color=color, alpha=wireframe_alpha)
    ax.plot_wireframe(xy_grid, yx_grid, z, color=color, alpha=wireframe_alpha)
    ax.plot_wireframe(xy_grid, yx_grid, -z, color=color, alpha=wireframe_alpha)

    # отрисовка граней
    ax.plot_surface(x, yz_grid, zy_grid, color=color, alpha=sides_alpha)
    ax.plot_surface(-x, yz_grid, zy_grid, color=color, alpha=sides_alpha)
    ax.plot_surface(xz_grid, y, zx_grid, color=color, alpha=sides_alpha)
```

```
    ax.plot_surface(xz_grid, -y, zx_grid, color=color, alpha=sides_alpha)
    ax.plot_surface(xy_grid, yx_grid, z, color=color, alpha=sides_alpha)
    ax.plot_surface(xy_grid, yx_grid, -z, color=color, alpha=sides_alpha)


def project_dots_to_parallelepiped(dots, subspaces):
    """Спроецировать точки на параллелепипед"""

    projections = [find_projection(dot, subspaces) for dot in dots]
    return projections


def find_projection(dot, subspaces):
      """Спроецировать точку на параллелепипед"""

      dot_x, dot_y, dot_z = dot
      projections = []

      # проецирование точки на каждую из плоскостей, полученных
      # продолжением сторон параллелепипеда
      for subspace in subspaces:
          A, B, C, D = subspace
          t = -D/(A*dot_x + B*dot_y + C*dot_z)
          projections.append([dot_x*t, dot_y*t, dot_z*t])

      # выбираем ближайшую проекцию
      nearest_projection = projections[0]
      for projection in projections:
          if distance.euclidean(projection, dot) < distance.euclidean(near-
est_projection, dot):
              nearest_projection = projection

      return nearest_projection


def find_game_cost(a, b, c, e, subspaces, dots_number, games_number):
    """Найти среднюю цену игры"""

    # проводим games_number игр, записывая исходы. Затем вычисляем среднее зна-
чение
    wins_history = []
    for _ in range(games_number):
        player_1_dots = generate_dots(dots_number, a/2, b/2, c/2)
        player_1_dots_projections = project_dots_to_parallelepi-
ped(player_1_dots,
```

```
                                                                    subspaces)
        player_2_dot = generate_dots(1, a/2, b/2, c/2)
        player_2_dot_projection = project_dots_to_parallelepiped(player_2_dot,
                                                                 subspaces)
        wins_history.append(find_winner(player_1_dots_projections,
                                        player_2_dot_projection, e))
    return np.mean(wins_history)


def find_winner(player_1_dots, player_2_dot, e):
    """Найти победителя игры"""

    if any([distance.euclidean(player_1_dot, player_2_dot[0]) <= e
            for player_1_dot in player_1_dots]):
      return 1
    else:
      return -1


def get_subspaces(a, b, c):
    """Получить коэффициенты уравнений плоскостей, образующих параллелепипед"""

    return [[1,0,0,a/2],
            [1,0,0,-a/2],
            [0,1,0,b/2],
            [0,1,0,-b/2],
            [0,0,1,c/2],
            [0,0,1,-c/2]]


def main():
    # стороны прямоугольника
    a = 12
    b = 16
    c = 24

    e = 0.5 # окрестность точки
    dots_number = 100 # кол-во точек
    games_number = 1000 # кол-во игр

    # настройки графического отображения
    fig = plt.figure(figsize=(20,20))
    ax_1 = fig.add_subplot(221, projection='3d')
    ax_2 = fig.add_subplot(222, projection='3d')
    ax_1.grid(True, which='both')
```

```
    ax_2.grid(True, which='both')
    lim = max(a, b, c)/2
    ax_1.set_xlim(-lim, lim)
    ax_1.set_ylim(-lim, lim)
    ax_1.set_zlim(-lim, lim)
    ax_2.set_xlim(-lim, lim)
    ax_2.set_ylim(-lim, lim)
    ax_2.set_zlim(-lim, lim)

    subspaces = get_subspaces(a, b, c)

    # нарисовать эллипсоид с точками внутри параллелепипеда
    player_1_dots = generate_dots(dots_number, a/2, b/2, c/2)
    draw_dots(player_1_dots, e, ax_1, color='cyan', alpha=1)
    draw_ellipsoid(a/2, b/2, c/2, ax_1, color='lightgrey', alpha=0.3)
    draw_parallelepiped(a, b, c, ax_1, color='grey',
                        wireframe_alpha=0.5, sides_alpha=0.05)

    # точка прячущегося игрока
    player_2_dot = generate_dots(1, a/2, b/2, c/2)
    draw_dots(player_2_dot, e, ax_1, color='orange', alpha=1)

    # рисуем проекции точек на параллелепипед
    player_2_dot_projection = project_dots_to_parallelepiped(player_2_dot,
                                                             subspaces)
    player_1_dots_projections = project_dots_to_parallelepiped(player_1_dots,
                                                               subspaces)
    draw_dots(player_1_dots_projections, e, ax_2, color='cyan', alpha=1)
    draw_dots(player_2_dot_projection, e, ax_2, color='orange', alpha=1)
    draw_parallelepiped(a, b, c, ax_2, color='grey',
                        wireframe_alpha=0.5, sides_alpha=0.05)

    avg_game_cost = find_game_cost(a, b, c, e, subspaces,
                                   dots_number, games_number)
    print("Цена игры после {0} итераций: {1}".format(games_number,
                                                     round(avg_game_cost,3)))

main()
```