**Министерство науки и высшего образования Российской Федерации**

**Московский государственный технический университет имени Н. Э. Баумана**

Факультет: Информатика и системы управления

Кафедра: Информационная безопасность (ИУ8)

**Рубежный контроль №1**

**ПО ДИСЦИПЛИНЕ «ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ»**

«**Бесконечные антагонистические игры**»

**Вариант 1**

**Студент**: Анаян М. С., ИУ8-104

**Преподаватель:** Коннова Н. С.

# Цель и задачи выполнения. Исходные данные

**Цель работы –** изучить постановку бесконечной антагонистической игры поиска на поверхности в соответствии с вариантом и найти стоимость игры.

## Постановка задачи

Разыгрывается игра поиска . Игрок A произвольным образом выбирает точек на кубе. Его вектор стратегий состоит из координат точек . Игрок B случайно выбирает точку на том же кубе. Также задаётся -окрестность точек , попадание в которую приводит к проигрышу игрока A. Таким образом, функция выигрыша для игрока (A) имеет вид:

Иначе говоря, игрок A выигрывает, если точка игрока B попала в --окрестность любой из точек игрока A..

Задача – найти цену игры.

## Исходные данные

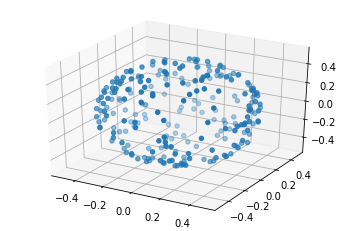
Поверхность – куб.

# Выполнение лабораторной работы

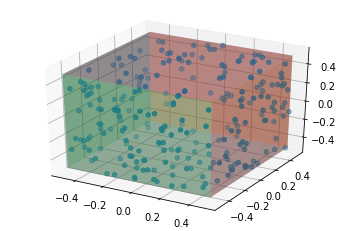
Выполним равномерное распределение точек на поверхности сферы, вписанной в куб, а затем спроецируем точки на поверхность куба при помощи радиус-векторов для полученных точек.

При распределении точек на сфере воспользуемся [методом Монте-Карло](https://projecteuclid.org/download/pdf_1/euclid.aoms/1177692644).

Пример распределения 250 точек на сфере:



Проекция на куб:



Выделим следующие параметры игры:

* – число точек, распределяемых игроком A,
* – радиус поражения точек на поверхности, выбранных игроком A,
* – размер куба.

Выполним модерирование игры с параметрами , , . При 100 розыгрышах получаем стоимость игры .

Выполним модерирование игры с параметрами , , . При 100 розыгрышах получаем стоимость игры , что вполне объяснимо, т. к. увеличив радиус поражения точки, мы увеличиваем площадь покрытия (поражения) на поверхности.

# Выводы

В ходе лабораторной работы получены следующие результаты:

* изучена постановка бесконечной антагонистической игры поиска на поверхности куба,
* выполнено моделирование розыгрышей игр (100 шт.), в результате которого вычислены стоимости исходных игр.

# Приложение

Исходный код на языке Python 3.\* также представлен в репозитории по ссылке: <https://github.com/hms2010/GameTheory/tree/master/src/rk1>

|  |
| --- |
| #!/usr/bin/env python  # coding: utf-8  # # Рубежный контроль 1. Бесконечные антагонистические игры  # In[1]:  import matplotlib.pyplot as plot  import mpl\_toolkits.mplot3d  import numpy as np  # ## Вариант 1. Равномерное распределение точек на кубе  # Сделаем равномерное распределение точек на поверхности сферы, вписанной в куб,а затем спроецируем точки на поверхность куба при помощи радиус-векторов для полученных точек. Точки на сфере распределим при помощи [метода Монте-Карло](https://projecteuclid.org/download/pdf\_1/euclid.aoms/1177692644)).  # ### 1. Равномерное распределение точек на сфере  # In[2]:  def generate\_sphere(n):      coords = np.array([[],[]])      s = np.array([])      while coords.shape[1] != n:          assert coords.shape[1] == s.shape[0]          cc = np.random.uniform(-1, 1, size = (2, n - coords.shape[1]))          ss = np.square(cc[0]) + np.square(cc[1])          ok = ss < 1          coords = np.concatenate([coords, np.compress(ok, cc, axis = 1)], axis = 1)          s = np.concatenate([s, ss[ok]])          z = 1 - 2 \* s      m = np.sqrt((1 - np.square(z)) / s)      x = coords[0] \* m / 2      y = coords[1] \* m / 2      z = z / 2      return np.stack([x, y, z])  # ### 2. Распределение точек на кубе  # In[3]:  from itertools import product  def create\_cube(a):      return [Plane(\*tmp) for tmp in product(range(3), [-abs(a), abs(a)])]  # In[4]:  from numpy import dot  from numpy import reshape  from numpy import append  from numpy.linalg import solve  from numpy import array, full, meshgrid  class Plane:      def \_\_init\_\_(self, axis, const):          self.axis = axis          self.const = const      def generate\_grid(self):          res = meshgrid(\*[[-self.const, self.const]]\*2)          assert res[0].shape == res[1].shape          res.insert(self.axis, full(res[0].shape, self.const))          return array(res)      def get\_coefs(self):          dots = reshape(self.generate\_grid().T, (4, 3))[:3]          return append(solve(dots, [-1] \* 3), 1)        def get\_intersection(self, point):          e = self.get\_coefs()          t = -e[-1] / dot(e[:3], point.T)          return point \* t, t        def is\_on\_surface(self, point):          if point[self.axis] != self.const:              return False          for i in range(3):              if i != self.axis:                  if (point[i] < -abs(self.const)) or (point[i] > abs(self.const)):                      return False          return True  # In[5]:  def project\_to\_surface(surface, points):      coords, param = np.apply\_along\_axis(surface.get\_intersection, 0, np.array(\*[points], dtype=np.longdouble))      coords = np.stack(coords)      on\_surf = np.apply\_along\_axis(surface.is\_on\_surface, 0, coords.T)        return coords[on\_surf & (param > 0)]  # In[6]:  def project\_to\_cube(surfaces, points):      return np.concatenate([project\_to\_surface(surface, points) for surface in surfaces])  # In[7]:  def generate\_cube(n, cube):      cube\_points = project\_to\_cube(cube, generate\_sphere(n))      while cube\_points.shape[0] != n:          cube\_points = np.concatenate([cube\_points, project\_to\_cube(cube, generate\_sphere(n - cube\_points.shape[0]))])      return cube\_points  # ### 3.1. Пример распределения точек на сфере  # In[8]:  dots\_num = 250  fig = plot.figure()  ax = plot.axes(projection='3d')  ax.scatter(\*generate\_sphere(dots\_num))  plot.show()  # ### 3.2. Пример распределения точек на кубе  # In[9]:  fig = plot.figure()  size =  0.5  ax = plot.axes(projection='3d')  ax.scatter(\*generate\_cube(dots\_num, create\_cube(size)).T.astype('f'))  for surf in create\_cube(size):      ax.plot\_surface(\*surf.generate\_grid(), alpha = 0.35)  plot.show()  # ### 4. Моделирование игры  # In[10]:  class Game:      def \_\_init\_\_(self, dots\_num, size, eps):          self.dots\_num = dots\_num          self.eps = eps          self.size = size          self.cube = create\_cube(self.size)      def is\_point\_on\_sphere(self, O, r, point):          return ((O - point) \*\* 2).sum() <= r \*\* 2      def get\_rand\_point(self):          surface = np.random.choice(self.cube)          c = (np.random.random(size = 2) \* abs(surface.const) \* 2 - abs(surface.const)).tolist()          c.insert(surface.axis, surface.const)          return np.array(c)      def model\_game(self):          cube\_pts = generate\_cube(self.dots\_num, self.cube)          p = self.get\_rand\_point()          return np.apply\_along\_axis(lambda o: self.is\_point\_on\_sphere(o, self.eps, p), 0, cube\_pts.T).any()      def model\_game\_iters(self, iters\_num):          successfull\_attempts = 0          for i in range(iters\_num):              if self.model\_game():                  successfull\_attempts += 1          return successfull\_attempts / iters\_num      def \_\_str\_\_(self):          return "Game params:\n    dots\_num: {:d},\n    size: {:.4f},\n    eps: {:.4f}".format(self.dots\_num, self.size, self.eps)    # In[11]:  game = Game(dots\_num = 300, size = 0.5, eps = 0.01)  print(game)  print("Game cost:", game.model\_game\_iters(100))  # In[12]:  game = Game(dots\_num = 300, size = 0.5, eps = 0.1)  print(game)  print("Game cost:", game.model\_game\_iters(100))  # In[ ]: |