# Лабораторная работа №4

## Супонина Анастасия

## группа: НПМмд-01-24

# “Исследование моделей размещения пользователей и распространения сигнала”

### Цель - исследовать сценарий размещения пользователей на плоскости и рассчитать вероятность блокировок прямой видимости.

## Задание.

Построить согласно описанному ниже алгоритму имитационную модель, позволяющую рассчитать вероятность блокировки линии прямой видимости, и сравнить полученную характеристику с результатами математической модели.

1. В квадратной области со стороной А согласно точечному Пуассоновскому процессу с плотностью q разместить объекты (окружности с фиксированным радиусом r), которые могут заблокировать линию прямой видимости между двумя точками, размещенными на расстоянии x друг от друга внутри обозначенной области. Проверить блокирует ли хотя бы один объект линию прямой видимости.
2. Количество блокирующих объектов определяется согласно распределению Пуассона с параметром $ qA^2 $, а объекты вбрасываются независимо друг от друга. При этом центры блокирующих объектов должны быть распределены равномерно внутри рассматриваемой области.

На основе описанного выше алгоритма получить 1000 реализаций факта блокировки линии прямой видимости и рассчитать вероятность блокировки линии прямой видимости в поле блокирующих объектов. Полученный результат сравнить с результатами математической модели.

Для начала работы ввожу необходимые библиотеки:

- numpy # для рандомной генерации чисел с различными распределениями и др.  
- matplotlib.pyplot # для построения графиков

И также отдельно для краткости написания ввожу функции из библиотеки numpy, такие как

- norm # вычисляет норму матрицы или вектора  
- arccos  
- dot # вычисляет скалярное произведение  
- pi  
- cross  
- cos  
- sin

import numpy as np  
import matplotlib.pyplot as plt  
from numpy.linalg import norm  
from numpy import arccos, dot, pi, cross, cos, sin

Задаю основные парметры:

guard\_size = 10 # размер квдратной области  
poisson\_lam\_block = 0.3 # плотность с которой будут возникать блокаторы  
radius\_block = 0.5 # значение радиуса блокатора  
distance\_Tx\_Rx = 3 # расстояние между передатчиком и приемником  
list\_point = np.array # переменная(массив) для записи точек

Рисую круг при помощи формулы окружности через угловые значения

# Рисуем круг  
def plot\_circle(x, y, r):  
 angles = np.linspace(0, 2\*pi, 50)  
 x\_cir = x + r\*cos(angles)  
 y\_cir = y + r\*sin(angles)  
 plt.plot(x\_cir, y\_cir, 'red')

Создаю функцию для точечного Пуассоновского процесса, а именно при помощи random.poisson генерирую случайные значения с Пуассоновским распределением в количестве равном $ qA^2 $ Также задаем координаты центров окружностей с равномерным распределением

# Инициализация Пуассоновского точечного процесса  
def poisson\_point\_process(lambda0, area\_size):  
 Number\_block = np.random.poisson(lambda0 \* area\_size \*\* 2)   
 x = np.random.uniform(0, area\_size, size = Number\_block)   
 y = np.random.uniform(0, area\_size, size = Number\_block)   
 return x,y

Создаю функцию которая при помощи координат двух точек и известного угла, строит прямоугольную область. Для этого мы из заданного угла получаем 3 других.

Из которых основными являеются:

- reverse\_angle # используется для определения положения одной из сторон прямоугольника вокруг отрезка  
- opposite\_angle # помогает определить другую сторону прямоугольника вокруг отрезка

И используя функции sin и cos расситываю четыре угла прямоугольника, окружающего линию между двумя точками.

При помощи которых вычисляю значения x и y, для каждой из вершин.

# Функция для прямоугольника вокруг линии прямой видимости   
def paint\_rectangle(x\_point\_1, y\_point\_1, x\_point\_2, y\_point\_2, angle):   
 difference\_angle = 2 \* pi - angle # угол поворота  
 reverse\_angle = pi/2 - difference\_angle # перпендикулярный угол к углу поворота  
 opposite\_angle = reverse\_angle + pi # противоположный угол для перпендикулярного  
 x\_rectang\_A = x\_point\_1 + radius\_block \* cos(opposite\_angle)   
 y\_rectang\_A = y\_point\_1 + radius\_block \* sin(opposite\_angle)  
 x\_rectang\_B = x\_point\_1 + radius\_block \* cos(reverse\_angle)  
 y\_rectang\_B = y\_point\_1 + radius\_block \* sin(reverse\_angle)  
 x\_rectang\_C = x\_point\_2 + radius\_block \* cos(reverse\_angle)  
 y\_rectang\_C = y\_point\_2 + radius\_block \* sin(reverse\_angle)  
 x\_rectang\_D = x\_point\_2 + radius\_block \* cos(opposite\_angle)   
 y\_rectang\_D = y\_point\_2 + radius\_block \* sin(opposite\_angle)  
   
 return x\_rectang\_A, y\_rectang\_A, x\_rectang\_B, y\_rectang\_B, x\_rectang\_C, y\_rectang\_C, x\_rectang\_D, y\_rectang\_D

Пишу функцию, которая будет находит расстояния от точки до векторов. Находим нормальные вектора и проверяем значения угла между двумя векторами fi, если угол больше 90 градусов или arccos больше 1, то это означает, что точка С находиться ближе к точке А и мы вычисляем и записываем эту дистанцию как наименьшую, аналогично проверяем для В и в ином случае если не одно из предыдущих условий не выполнилось, то это означает, что точка С находится между точками А и В, а значить для получения наимешьшего рассторяния нам нужно нужно вычислить перпендикулярный вектор до прямой АВ.

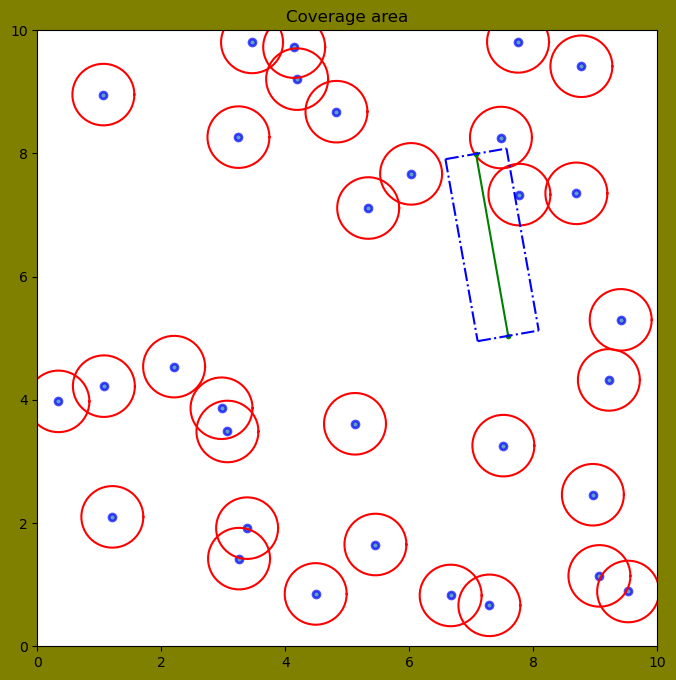
def check\_distance(A, B, C):  
 CA = (C - A) / norm(C - A)   
 BA = (B - A) / norm(B - A)   
 CB = (C - B)/norm(C - B)   
 AB = (A - B) / norm(A - B)   
 if arccos(dot(CA, BA)) > 1:   
 return norm(C - A)   
 if arccos(dot(CB, AB)) > 1:   
 return norm(C - B)   
 return norm(cross(A - B, A - C)) / norm(B - A)

Создаю функцию которая будет определять пересечение блокаторов с линией связи(линией прямой видимости). Она считает длины векторов от точки до нашей прямой и мы сравниваем каждое расстояние с нашим радиусом и если значение меньше, то блокатор пересекается с сигналом, таким образом блокируя его.

def crossing():   
 circle\_point = [];   
 point\_1 = [];   
 point\_2 = [];   
 point\_1.extend([x1, y1])   
 point\_2.extend([x2, y2])   
 for i in range(len(x)):   
 circle\_point.append([x[i], y[1]])   
 for i in range(len(x)):   
 if (np.round(check\_distance(list\_point(point\_1), list\_point(point\_2), list\_point(circle\_point[i])), 1) <= radius\_block):   
 return True

Переходим к отображению на графике, вычисляя все значения при помощи ранее созданных функций. Задаем случайную начальную точки и угол, и также задаем значение при помощи формулы второй точки.

x, y = poisson\_point\_process(poisson\_lam\_block,guard\_size)   
x1 = np.random.uniform (0,guard\_size)   
y1 = np.random.uniform (0,guard\_size)   
angle = np.random.uniform (0,2\*pi)   
x2 = x1 + distance\_Tx\_Rx \* cos(angle)   
y2 = y1 + distance\_Tx\_Rx \* sin(angle)   
x\_rectang\_A, y\_rectang\_A, x\_rectang\_B, y\_rectang\_B, x\_rectang\_C, y\_rectang\_C, x\_rectang\_D, y\_rectang\_D = paint\_rectangle(x1, y1, x2, y2, angle)  
  
plt.figure(dpi=100, figsize=(8, 8), facecolor='olive')  
plt.title('Coverage area')  
plt.plot(x, y, '.', alpha=0.7, label="first", lw=5, mec = 'b', mew=2, ms=10)   
for i in range(len(x)):   
 plot\_circle(x[i], y[i], radius\_block) # функция рисующая круги  
plt.plot([x1,x2], [y1,y2], '.-g')   
  
# функция рисующая зону блокировки вокруг линии прямой видимости  
plt.plot([x\_rectang\_A, x\_rectang\_B], [y\_rectang\_A, y\_rectang\_B], '-.b')   
plt.plot([x\_rectang\_A, x\_rectang\_D], [y\_rectang\_A, y\_rectang\_D], '-.b')   
plt.plot([x\_rectang\_B, x\_rectang\_C], [y\_rectang\_B, y\_rectang\_C], '-.b')   
plt.plot([x\_rectang\_D, x\_rectang\_C], [y\_rectang\_D, y\_rectang\_C], '-.b')   
plt.xlim(0,guard\_size)   
plt.ylim(0,guard\_size)   
plt.show()



png

# События блокировки  
if (crossing()):  
 print('LoS blocked')  
else:  
 print('LoS in not blocked')

LoS blocked

На основе описанного выше алгоритма получаю 1000 реализаций факта блокировки линии прямой видимости и рассчитываю вероятность блокировки линии прямой видимости в поле блокирующих объектов.

# Имитационное моделирование   
N = 1000 #param (type: "integer"}   
summa = 0  
for i in range(N):   
 x, y = poisson\_point\_process(poisson\_lam\_block, guard\_size)   
 x1 = np.random.uniform (0, guard\_size)   
 y1 = np.random.uniform (0, guard\_size)   
 angle = np.random.uniform (0, 2 \* pi)   
 x2 = x1 + distance\_Tx\_Rx \* cos(angle)   
 y2 = y1 + distance\_Tx\_Rx \* sin(angle)   
 if (crossing()):   
 summa += 1   
print('Beроятность блокировки:', summa/N)

Beроятность блокировки: 0.259

Получаю то значение для математической модели

# Вероятность блокировки по аналитической формуле  
S = 2 \* radius\_block \* distance\_Tx\_Rx   
lamb = poisson\_lam\_block \* S   
probability = 1-np.exp(-lamb)   
print('Beроятность блокировки-2:', probability)

Beроятность блокировки-2: 0.5934303402594008

# Вывод

В процессе выполнения лабораточной работы была изучена и реализована модель расчета вероятности блокировки линии прямой видимости между двумя точками в заданной области, были получены значения вероятности блокировки линии прямой видимости для заданных параметров плотности препятствий, радиуса и расстояния между точками.

Сравнивая оба результата мы можем прийти к выводу, что написанная модель работает корректно, демонстрируя результаты в пределах допустимых отклонений.