Лабораторная №6 Построение общей модели на основе случайных полей и статической модели блокировки

Цель:

1. Ознакомление с общей моделью на основе случайных полей и статической модели блокировки. Задание. В круге радиуса R, центром которого является целевой приемник, распределены согласно Пуассоновскому равномерному точечному процессу с интенсивностью B интерферирующие устройства. На целевом приемнике расположена всенаправленная антенна, а интерферирующие устройства оборудованы направленными антеннами с углом диаграммы направленности γ. Предполагается, что все антенны сориентированы случайным образом и постоянно излучают сигнал с мощностью P. При этом применяется модель распространения сигнала Cone (аналог модели Free Space Path Loss с поправкой на коэффициент диаграммы направленности). В этом же круге согласно Пуассоновскому равномерному точечному процессу с интенсивностью A распределены блокирующие объекты радиуса D. Для упрощения интерферирующие устройства не считать блокирующими объектами.
2. В случае если сигнал от интерферирующего передатчика до целевого приемника проходит через блокирующий объект, то он не создает помех на целевом приемнике. Аналогично, если целевой приемник не попадает в сектор диаграммы направленности интерферирующего передатчика, то соответствующий интерферирующий сигнал тоже не создает помех.
3. На основе полученной модели построить графики среднего значения интерференции как функции от направленности антенн γ, интенсивности Пуассоновского поля источников интерферирующих устройств B, интенсивности Пуассоновского поля блокирующих объектов A. При построении графиков использовать как минимум 1000 реализаций.

import numpy as np  
import matplotlib.pyplot as plt  
import scipy.stats  
import math  
from numpy.linalg import norm  
from numpy import arccos, dot, pi, cross  
  
def plot\_circle(x,y,r):  
 angles=np.linspace(0,2\*np.pi,50)  
 x\_cir=x+r\*np.cos(angles)  
 y\_cir=y+r\*np.sin(angles)  
 plt.plot(x\_cir,y\_cir,'red')  
  
def plot\_zone(x,y,distance,angle,fixang):  
 x1=x+distance\*np.cos(angle)  
 y1=y+distance\*np.sin(angle)  
 x2=x+distance\*np.cos(angle+fixang)  
 y2=y+distance\*np.sin(angle+fixang)  
 plt.plot([x,x1],[y,y1], '-.b')  
 plt.plot([x,x2],[y,y2], '-.b')  
  
def poisson\_point\_process(lambda0,R,area\_size):  
 N = np.random.poisson(lambda0\*area\_size\*\*2)  
 x1=[]  
 y1=[]  
 while len(x1)!=N:  
 x = np.random.uniform(0,area\_size)  
 y = np.random.uniform(0,area\_size)  
 if ((x-x0)\*\*2+(y-y0)\*\*2<=R\*R):  
 x1.append(x)  
 y1.append(y)  
 return x1,y1  
  
  
  
guard\_size = 10  
guard\_radius=4  
poisson\_lam\_block = 0.1  
poisson\_lam\_int = 0.07  
radius = 0.3  
distance = 1  
fixang = np.pi/4  
list\_point=np.array  
  
x0=guard\_size/2  
y0=guard\_size/2  
  
x\_block, y\_block=poisson\_point\_process(poisson\_lam\_block, guard\_radius, guard\_size)  
x\_int, y\_int=poisson\_point\_process(poisson\_lam\_int, guard\_radius, guard\_size)  
angle1 = np.random.uniform(0,2\*np.pi,size=len(x\_int))  
  
circle\_angels=np.linspace(0, 2\*np.pi, 100)  
circle\_x=x0+guard\_radius\*np.cos(circle\_angels)  
circle\_y=y0+guard\_radius\*np.sin(circle\_angels)  
  
plt.figure(dpi=100, figsize=(8,8), facecolor='olive')  
plt.title('Coverage area')  
plt.plot(x\_block,y\_block,'.', alpha=0.7,label='fist',lw=5,mec='r',mew=3,ms=10)  
for i in range(len(x\_block)):  
 plot\_circle(x\_block[i],y\_block[i], radius)  
plt.plot(x\_int,y\_int,'.', alpha=0.7,label='fist',lw=5,mec='b',mew=2,ms=10)  
for i in range(len(x\_int)):  
 plot\_zone(x\_int[i],y\_int[i],distance,angle1[i],fixang)  
plt.plot(x0,y0,'.', alpha=0.7,label='fist',lw=5,mec='g',mew=2,ms=10)  
plt.plot(circle\_x,circle\_y,'green')  
  
plt.xlim(0,guard\_size)  
plt.ylim(0,guard\_size)  
plt.show()  
  
def distance(x1, y1, x2, y2):  
 return np.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)  
  
  
def check\_pol(line\_d1, line\_d2, angle, fixang):  
 x1, y1 = line\_d1  
 x2, y2 = line\_d2  
 atan1 = math.atan2((y2 - y1), (x2 - x1))  
 if atan1 < 0:  
 atan1 = 2 \* np.pi + atan1  
 d1 = atan1 > angle and atan1 < (angle + fixang)  
 if d1 == True:  
 return True  
 else:  
 return False  
  
  
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)  
  
  
def crossing(x1, y1, x2, y2, x, y, radius):  
 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[i]])  
  
 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):  
 return True  
  
  
def interf(x0, y0, x\_int, y\_int, P, fixang):  
 rastoyanie = distance(x0, y0, x\_int, y\_int)  
 G = 2 / (1 - np.cos(fixang / 2))  
 prx = P \* G / (4 \* np.pi \* rastoyanie \*\* 2)  
 return prx  
  
P=2  
num\_experiments = 1000  
  
def immit(num\_experiments,P,fixang,poisson\_lam\_block,poisson\_lam\_int,guard\_radius,guard\_size):  
 Interf=[]  
 for j in range (num\_experiments):  
 Inter1=0  
 x\_block, y\_block=poisson\_point\_process(poisson\_lam\_block, guard\_radius, guard\_size)  
 x\_int, y\_int=poisson\_point\_process(poisson\_lam\_int, guard\_radius, guard\_size)  
 angle1 = np.random.uniform(0,2\*np.pi,size=len(x\_int))  
 for i in range((len(x\_int))):  
 if check\_pol([x\_int[i],y\_int[i]],[x0,y0],angle1[i],fixang)==True and crossing(x0,y0,x\_int[i],y\_int[i],x\_block,y\_block,radius) == None:  
 Inter1=Inter1+interf(x0,y0,x\_int[i],y\_int[i],P,fixang)  
 Interf.append(Inter1)  
 return np.mean(Interf)  
  
num\_experiments = 1000  
Interf\_fi=[]  
fixang=0  
fixang\_int=[]  
shag=10  
for l in range(shag):  
 fixang=fixang+np.pi/shag  
 fixang\_int.append(fixang)  
 Interf\_fi.append(immit(num\_experiments,P,fixang,poisson\_lam\_block,poisson\_lam\_int,guard\_radius,guard\_size))  
  
plt.figure(dpi=100)  
plt.title('Interferencia by angle')  
plt.plot(fixang\_int,Interf\_fi)  
plt.xlabel('Угол')  
plt.ylabel('Значение интерференции')  
plt.show()  
  
Interf\_fi=[]  
fixang=np.pi/4  
poisson\_lam\_int = 0  
poisson\_lam\_int\_op = []  
#2PI/(PI/6)  
for l in range(shag):  
 poisson\_lam\_int=poisson\_lam\_int+0.01  
 poisson\_lam\_int\_op.append(poisson\_lam\_int)  
 Interf\_fi.append(immit(num\_experiments,P,fixang,poisson\_lam\_block,poisson\_lam\_int,guard\_radius,guard\_size))  
  
plt.figure(dpi=100)  
plt.title('Interferencia by B')  
plt.plot(poisson\_lam\_int\_op,Interf\_fi)  
plt.xlabel('Интенсивность распределения интерферирующих устройств')  
plt.ylabel('Значение интерференции')  
  
Interf\_fi=[]  
fixang=np.pi/4  
poisson\_lam\_int = 0.07  
poisson\_lam\_block = 0  
poisson\_lam\_block\_op = []  
for l in range(shag):  
 poisson\_lam\_block=poisson\_lam\_block+0.01  
 poisson\_lam\_block\_op.append(poisson\_lam\_block)  
 Interf\_fi.append(immit(num\_experiments,P,fixang,poisson\_lam\_block,poisson\_lam\_int,guard\_radius,guard\_size))  
  
plt.figure(dpi=100)  
plt.title('Interferencia by A')  
plt.plot(poisson\_lam\_block\_op,Interf\_fi)  
plt.xlabel('Интенсивность распределения блокирующих устройств')  
plt.ylabel('Значение интерференции')  
plt.show()







