

Индивидуальное задание

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

Выполнил студент 2 курса
учебной группы НММ-02-22
Мулин Иван

Цели

Построить согласно описанному ниже алгоритму имитационную модель, позволяющую вычислять среднее значение интерференции; построить графики зависимости среднего значения интерференции от угла диаграммы направленности, интенсивности Пуассоновского поля источников интерферирующих устройств В, интенсивности Пуассоновского поля блокирующих объектов А.

Задача

В круге радиуса R, центром которого является целевой приемник, распределены согласно Пуассоновскому равномерному точечному процессу с интенсивностью В интерферирующие устройства. На целевом приемнике расположена всенаправленная антенна, а интерферирующие устройства оборудованы направленными антеннами с углом диаграммы направленности θ . Предполагается, что все антенны сориентированы случайным образом и постоянно излучают сигнал с мощностью Р. При этом применяется модель распространения сигнала Соло (аналог модели Free Space Path Loss с поправкой на коэффициент диаграммы направленности). В этом же круге согласно Пуассоновскому равномерному точечному процессу с интенсивностью А распределены блокирующие объекты радиуса D. Для упрощения интерферирующие устройства не создают сигнал блокирующим объектам.

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.

На основе полученной модели построить графики среднего значения интерференции как функции от направленности антенн у, интенсивности Пуассоновского поля источников интерферирующих устройств В, интенсивности Пуассоновского поля блокирующих объектов А. При построении графиков использовать как минимум 1000 реализаций.

Ход работы

Подключаем нужные и ненужные библиотек. После этого вводим константы, данные по условию.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import savgol_filter
from numpy.linalg import norm
from numpy import cross, dot, arcsin, arccos, pi

In [2]: FieldRadius = 6
bRadius = 0.5
power = 1
BlockParameter, InfrParameter = 10, 6
N = 1000
angles = np.linspace(0, 2 * pi, 80)

Описание функций, необходимые для построения модели.

In [3]: # Functions for drawing
def DrawCircle(x0, y0, r, outerhue="a9cfd9", fillhue="aadc1e8", ls="solid"):
    """
    Draws a circle on a plot.
    """
    global angles
    xs = x0 + r*np.cos(angles)
    ys = y0 + r*np.sin(angles)
    plt.plot(xs, ys, color=outerhue, markers=".")
    plt.plot(xs, ys, color=outerhue, linestyle=ls)
    plt.fill(xs, ys, color=fillhue)

def DrawAngle(x0, y0, angle, mu, hush="k3db0a8"):
    """
    Draws angle of vision at (x0, y0) rotated by mu radians.
    """
    x1, y1 = x0 + np.cos(mu), y0 + np.sin(mu)
    x2, y2 = x0 + np.cos(mu - angle), y0 + np.sin(mu - angle)
    plt.plot([x0, x1], [y0, y1], color=hush)
    plt.plot([x0, x2], [y0, y2], color=hush)

In [4]: # Algebraic functions
def sign(x):
    """
    A mathematical sign(x).
    """
    if x == 0:
        return 0
    return x/abs(x)

def Distance(x1, y1, x2, y2):
    """
    A normal euclidean plane metric.
    """
    return np.sqrt((x2-x1)**2 + (y2-y1)**2)

def CalcDistance(L, P):
    """
    Calculates distance between line Ax+By+C=0 and point P=(x0 y0).
    (L=(A B C)).
    """
    Q = L[:1]
    R = np.append( np.transpose(P), 1)
    return abs( np.matmul(L, R) ) / norm(Q)

def GetLine(x1, y1, x2, y2):
    """
    Calculates vector (A B C) for a line Ax+By+C=0.
    """
    det = x1*y2 - x2*y1
    if det != 0: # y!=x, s!=0
        A = (y1-y2) / det
        B = (x2-x1) / det
        return np.array([A, B, 1])
    if x1!=x2 + y1!=y2 == 0 or x1==0 or y1==0: # if x==y, that is not a line!
        return np.array([0, 1, 0])
    # ...then it is y=kx, k!=0
    B = -1
    A = y1/x1
    return np.array([A, B, 0])

def PointInSection(b, c):
    """
    Determines whether point is in section. Angles b, c to be passed.
    """
    return (0 == b) * (b == c)

In [5]: def PoissonProcessInitializer(lan, R, angle=2*pi, precision=0.0008001, x0=0, y0=0):
    """
    Initializes Poisson process.
    """
    amount = np.random.poisson(lan)
    r = np.random.uniform(0, R, size=amount)
    o = np.random.uniform(0, angle, size=amount)
    x, y = x0 + r * np.cos(o), y0 + r * np.sin(o)
    x = x % precision
    y = y % precision
    return list(zip(x, y))

In [6]: class Diagram:
    """
    Diagram is two sectors of vision.
    """
    def __init__(self, \
        x, x0, y, y0, power, \
        mu, angle):
        """
        # Line of sight stuff
        self.x, self.y = x, y
        self.power = power
        self.distance = norm([x0-x, y0-y])
        if self.distance == 0:
            self.incline = 0
        else:
            self.incline = sign( arcsin((y0-y)/self.distance) ) * arccos((x0-x)/self.distance)
        # Sectors of vision stuff
        self.vAngle, self.mu = angle, mu

    def DrawVisionSector(self, ihue="99748d9", vhue="d9b048"):
        """
        Draws vision sector.
        """
        DrawAngle(self.x, self.y, self.vAngle, self.mu+self.incline, vhue)
        plt.plot(self.x, self.y, markers='x', color=ihue)

In [7]: def BornInterferences(VAngle, lan, R, maxangle=2*pi, precision=0.0008001, x0=0, y0=0):
    global power
    positions = PoissonProcessInitializer(lan, R, maxangle, precision, x0, y0)
    l = len(positions)
    interferences = np.empty(l, dtype=Diagram)
    angles = np.random.uniform(0, maxangle, size=l)
    for i in range(l):
        interferences[i] = Diagram(positions[i][0], x0, positions[i][1], y0, power, angles[i], VAngle)
    return interferences

In [8]: def IsBlocked(I, 0):
    """
    Checks if interference I's signal is blocked.
    """
    global bRadius
    y = I.vAngle
    dist = Distance(I.x, I.y, "0")
    if bRadius == dist:
        return 1
    n = I.mu+I.incline
    ray = np.cos(M), np.sin(M)
    connector = [0,0], I.x, 0, I.y]
    product = dot(ray, connector)/dist
    os = arcsin(bRadius/dist)
    o = sign(cross(ray, connector)) * arccos(product)
    #plt.plot([I.x, I.x+ray[0]], [I.y, I.y+ray[1]], linestyle="dashed", color="9b88d4")
    #plt.plot([I.x, I.x+connector[0]], [I.y, I.y+connector[1]], linestyle="dashed", color="8a6486")
    return ( abs(o) <= y+os ) * ( dist <= I.distance + bRadius)

In [9]: def TotalInterference(blockators, interferences: list[Diagram], x0=0, y0=0):
    """
    Will check interference.
    """
    if len(interferences) == 0:
        return 0
    P = interferences[0].power
    y = interferences[0].vAngle
    I = 0
    for i in interferences:
        if not PointInSection(i.mu, i.vAngle):
            continue
        for b in blockators:
            if IsBlocked(i, b):
                continue
        # signal interferences
        I += I.distance ** (-2)
    o = np.sin(y/4) ** (-2)
    I = (P * O) / (4 * pi)
    return I

In [10]: def MeanInterference(N, VisionAngle, BlockParameter, InterferenceParameter):
    global FieldRadius
    l = 0
    for n in range(N):
        blockators = PoissonProcessInitializer(BlockParameter, FieldRadius)
        interferences = BornInterferences(VisionAngle, InterferenceParameter, FieldRadius)
        l = TotalInterference(blockators, interferences)
    return l / N

Тестовая модель
Участок кода ниже предназначен для проверки работы написанных функций.

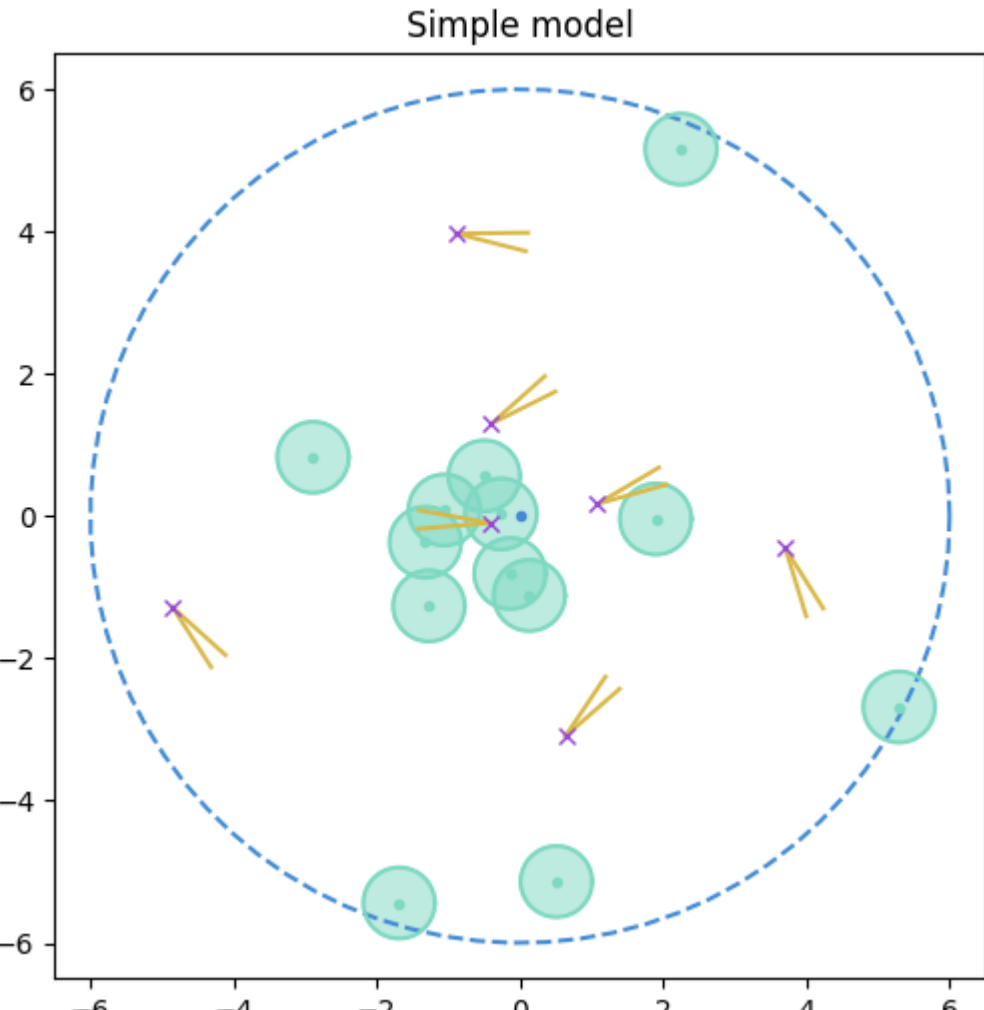
In [11]: VisionAngle = np.random.uniform(0.001, pi/3)
blockators = PoissonProcessInitializer(BlockParameter, FieldRadius)
interferences = BornInterferences(VisionAngle, InterParameter, FieldRadius)
plt.figure(dpi=100, figsize=(6, 6))
plt.title("Simple model")

DrawCircle(0, 0, FieldRadius, "a428b04", "ffffff", "dashed")
for b in blockators:
    DrawCircle(b[0], b[1], bRadius, "a7ed9c2", "a7ed9c2"+"8a")
for i in interferences:
    i.DrawVisionSector()

print( TotalInterference(blockators, interferences) )

u = FieldRadius + bRadius
plt.xlim(-u, u)
plt.ylim(-u, u)
plt.show()

0.8

Simple model


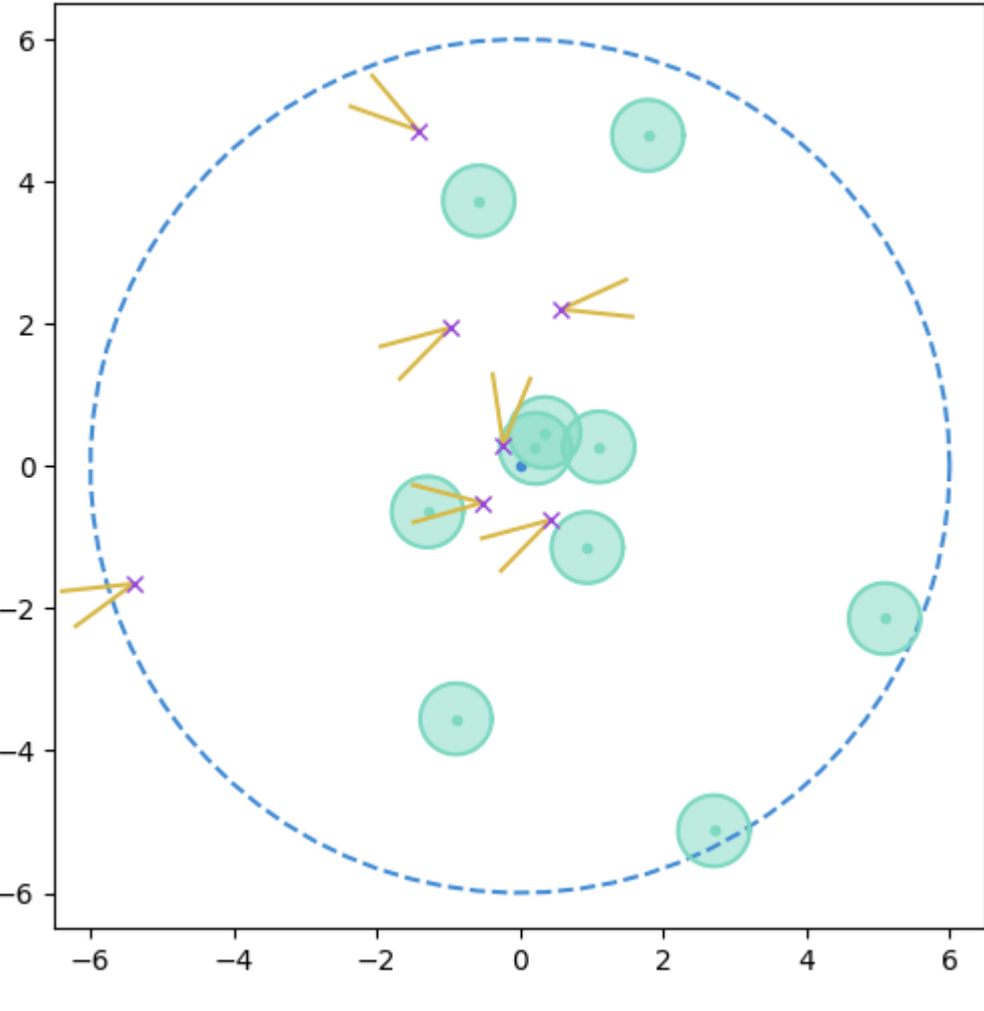
In [12]: VisionAngle = np.random.uniform(0.001, pi/3)
blockators = PoissonProcessInitializer(BlockParameter, FieldRadius)
interferences = BornInterferences(VisionAngle, InfrParameter, FieldRadius)
plt.figure(dpi=100, figsize=(6, 6))
plt.title("Simple model")

DrawCircle(0, 0, FieldRadius, "a428b04", "ffffff", "dashed")
for b in blockators:
    DrawCircle(b[0], b[1], bRadius, "a7ed9c2", "a7ed9c2"+"8a")
for i in interferences:
    i.DrawVisionSector()

print( TotalInterference(blockators, interferences) )

u = FieldRadius + bRadius
plt.xlim(-u, u)
plt.ylim(-u, u)
plt.show()

0.8

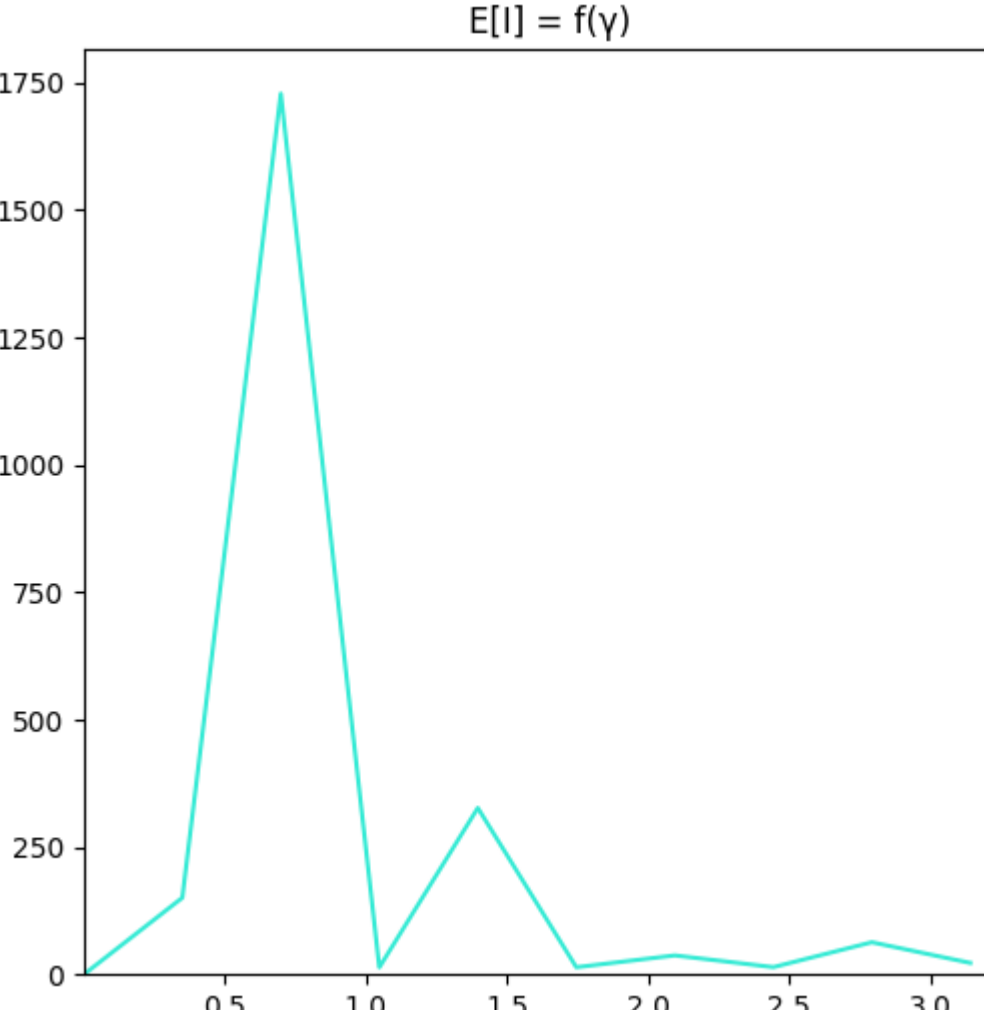
Simple model


In [13]: Y = np.linspace(0.001, pi, 10)
EIs = np.empty(len(Y))
for y in range(len(Y)):
    EIs[y] = MeanInterference(N, Y[y], BlockParameter, InfrParameter)

plt.figure(dpi=100, figsize=(6, 6))
plt.title("E[I] = f(y)")

plt.plot(Y, EIs, color="k34eb05")

plt.xlim(min(Y)*0.95, max(Y)*1.05)
plt.ylim(0, max(EIs)*1.05)
plt.show()

E[I] = f(y)


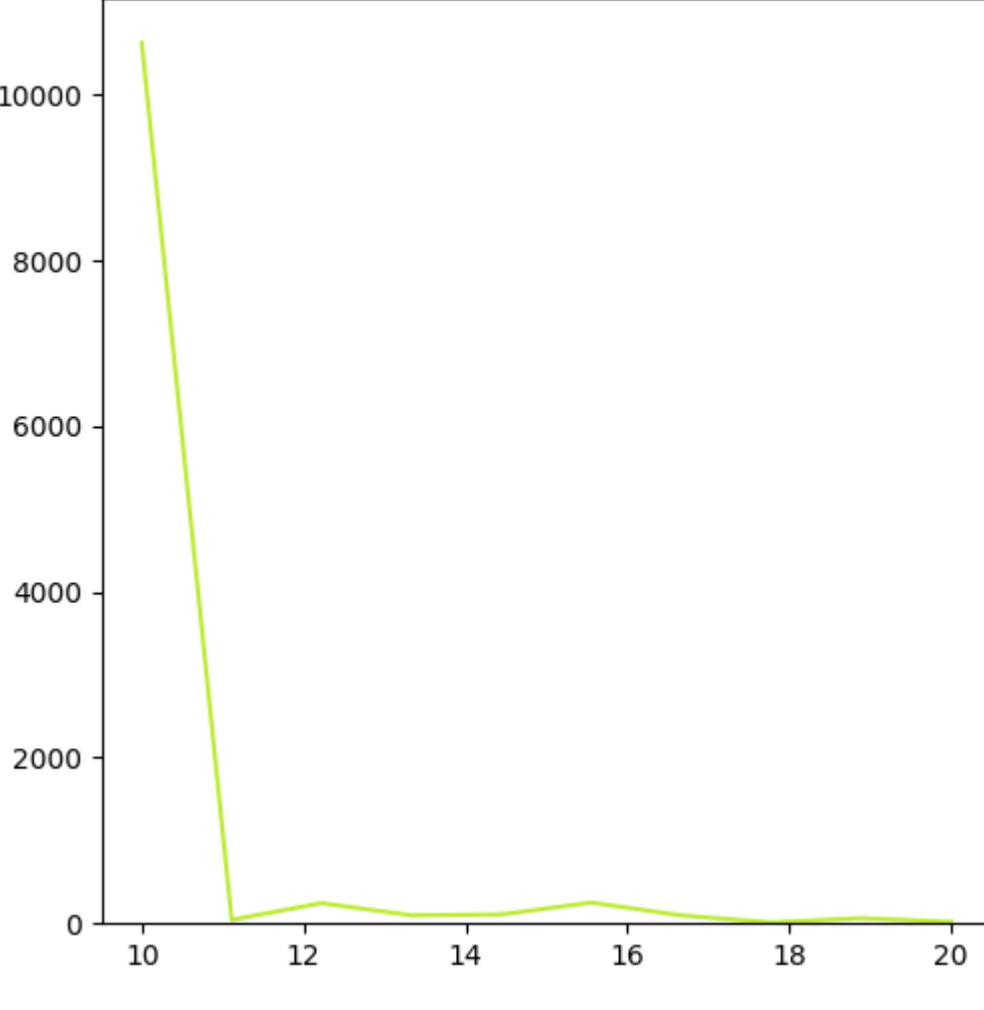
In [14]: BlockParameters = np.linspace(10, 20, 10)
EIs = np.empty(len(BlockParameters))
for b in range(len(BlockParameters)):
    EIs[b] = MeanInterference(N, VisionAngle, BlockParameters[b], InfrParameter)

#EIs = savgol_filter(EIs, len(EIs), 3) # smooth the graph if you wish

plt.figure(dpi=100, figsize=(6, 6))
plt.title("E[I] = f(B)")

plt.plot(BlockParameters, EIs, color="9b8b34")

plt.xlim(min(BlockParameters)*0.95, max(BlockParameters)*1.05)
plt.ylim(0, max(EIs)*1.05)
plt.show()

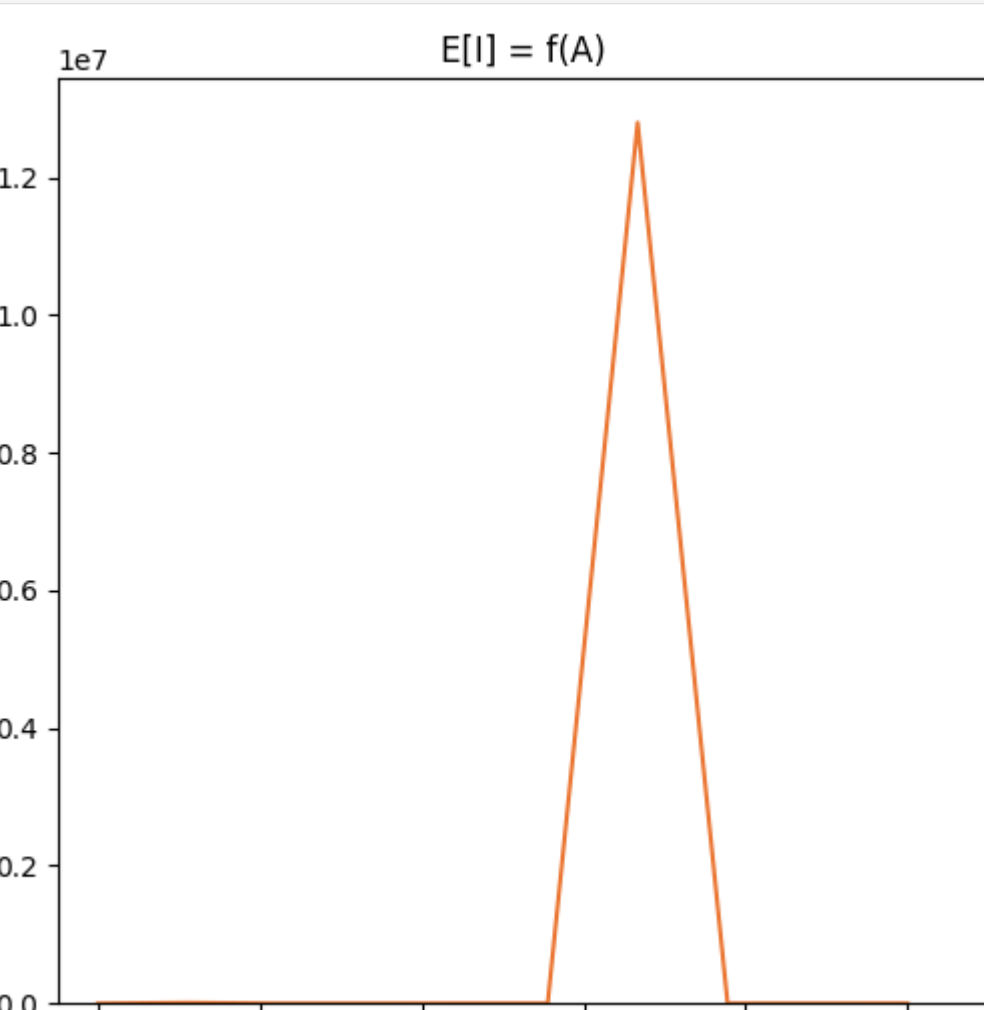
E[I] = f(B)


In [15]: InfrParameters = np.linspace(10, 20, 10)
EIs = np.empty(len(InfrParameters))
for i in range(len(InfrParameters)):
    EIs[i] = MeanInterference(N, VisionAngle, BlockParameter, InfrParameters[i])
#EIs = savgol_filter(EIs, len(EIs), 3) # smooth the graph if you wish

plt.figure(dpi=100, figsize=(6, 6))
plt.title("E[I] = f(A)")

plt.plot(InfrParameters, EIs, color="eb7734")

plt.xlim(min(InfrParameters)*0.95, max(InfrParameters)*1.05)
plt.ylim(0, max(EIs)*1.05)
plt.show()

E[I] = f(A)

```

Вывод

