Отчет по дисциплине
“Системы навигации автономных роботов”
hw №4

Тема: “GNSS”

Выполнили студенты:	Григорьев Максим Эдуардович - 34% Максимова Мария Павловна - 33% Биковец Константин Сергеевич - 33%
Преподаватель	Гарцеев Илья Борисович

Москва, 2024

Постановка задачи

Основное задание часть 1:

Даны координаты пяти маяков на плоскости и измерения псевдодальностей от этих маяков до расположенного в той же плоскости приёмника. Эти измерения выражены моделью:

$$\rho_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} + \tau + r_i,$$

где i – номер маяка, p_i – измеренная псевдодальность, (x_i, y_i) – координаты i-го маяка, (x_0, y_0) – координаты приёмника, τ – погрешность часов приёмника, приведённая к единицам длины, r_i – шум измерений. Требуется по данным измерениям оценить координаты приёмника и погрешность его часов с максимально возможной (в смысле минимизации суммы квадратов невязок) точностью.

Основное задание часть 2:

Необходимо понять, как меняется оценка координат и часов приемника при выходе из строя одного и двух маяков. Дизайн исследования свободный. Рассмотреть ситуации с мягкими и жесткими требованиями точности.

Основное задание часть 3:

Создать моделирующее программное обеспечение для автоматического вычисления оценки координат с графическим интерфейсом на основе информации о маяках, приёмнике и точности.

Основная часть 1

Определение местоположения приемника производится путем построения окружностей с центрами в координатах расположения маяков и радиусами, равными полученным расстояниям до них. В идеальной ситуации все окружности должны пересекаться в одной точке, которая и будет являться истинным положением приемника.

Однако на практике из-за неточности часов и помех такого идеального пересечения не происходит. Полученные расстояния не являются истинными и называются псевдорасстояниями, поскольку содержат погрешность, вызванную ошибкой измерения времени.

В связи с этим требуемые координаты местоположения приемника определяются с помощью специального алгоритма, описанного ниже.

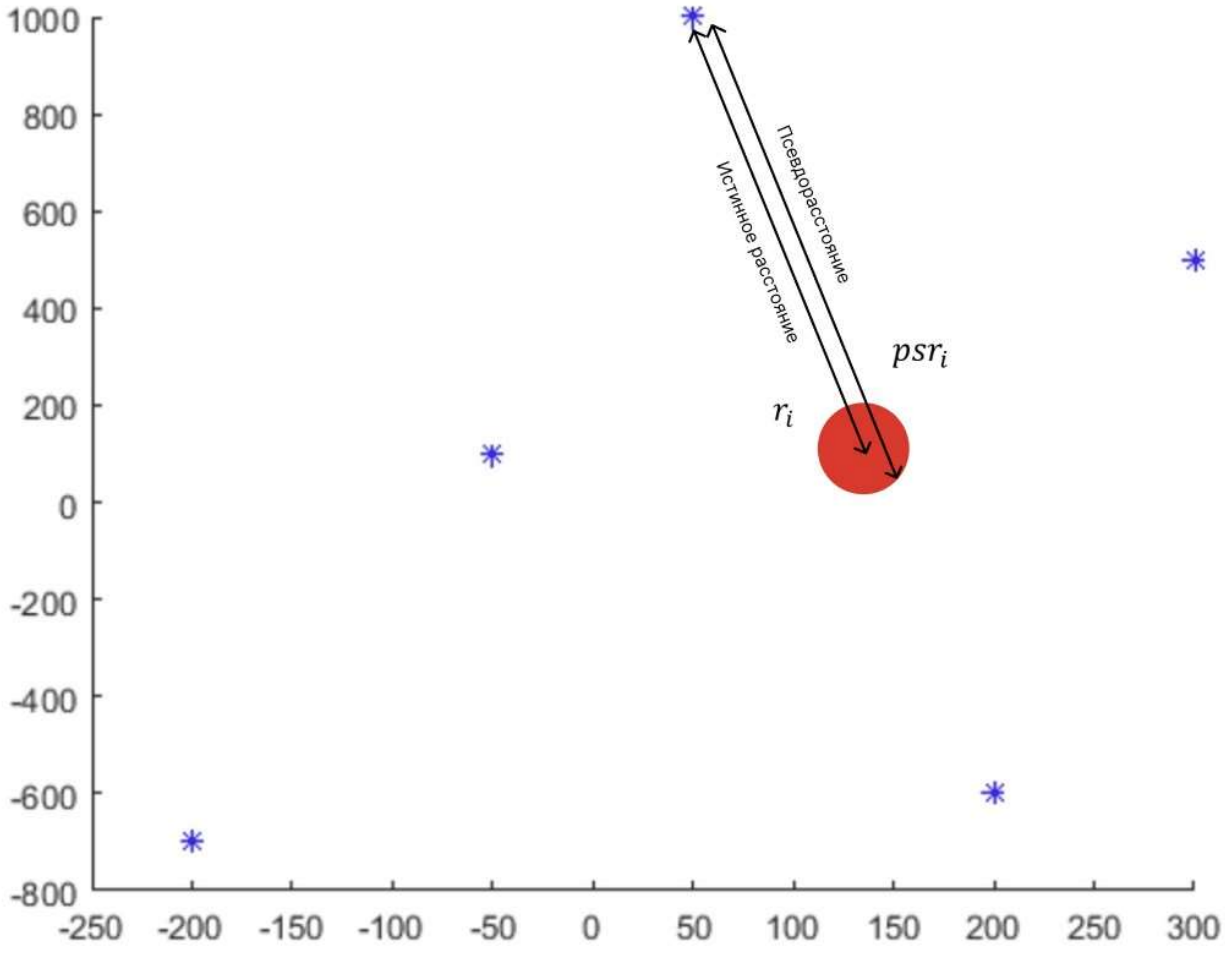

Рис. 1. Карта расположения маяков в общем случае

Нахождение предполагаемой позиции

Для всех псевдодальностных окружностей находятся точки их пересечения путем решения систем, состоящих из двух уравнений. При этом учитывается, что если расстояние между центрами окружностей превышает сумму их радиусов, то пересечения не происходит. Затем вычисляются средние значения координат x и y для полученных точек пересечения. Эти средние значения и будут представлять собой предполагаемое местоположение приемника (показано красной точкой на рис. 2).

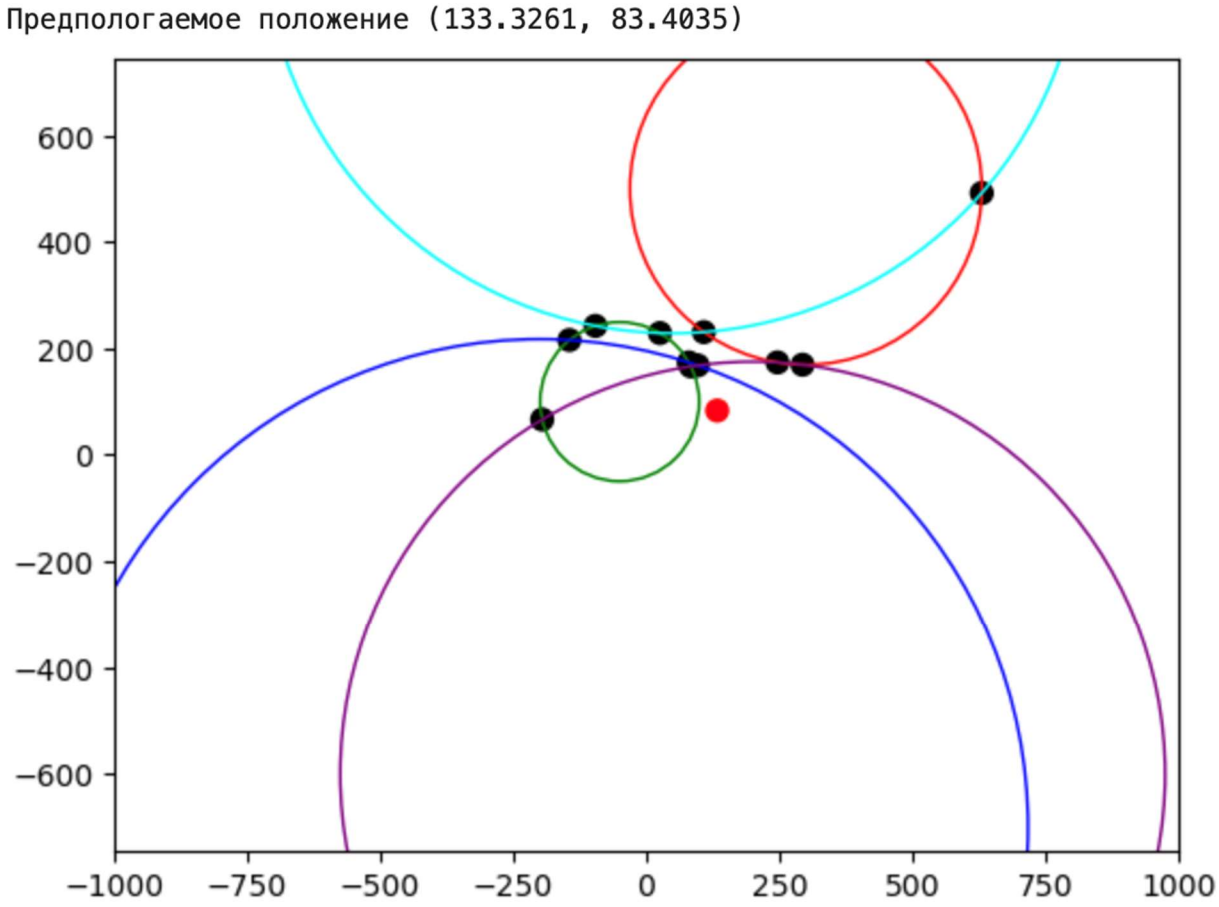

Рис. 2. Предполагаемое положение приёмника

Видно, что одна из точек пересечения окружностей находится на значительном удалении от группы остальных точек, и это существенно искажает вычисленные средние координаты местоположения.

Составление матричного уравнения

Задача состоит в том, чтобы определить расположение объекта с максимальной точностью, то есть минимизировать погрешности Δx и Δy (минимизировать невязки). Начальная оценка местоположения, как было сказано ранее, вычисляется как среднее арифметическое координат всех предполагаемых позиций.

Обозначения:

psr_i - псевдорасстояния

r_i - расстояния до предполагаемой позиции

τ - погрешность часов, переведенная к единицам длины

Δx - погрешность по координате x

Δy - погрешность по координате y

x0, y0 - координаты предполагаемой позиции объекта (среднее по координатам)

Предполагаемая позиция, включающая ошибку (3):

$$x = x_0 + \Delta x$$
$$y = y_0 + \Delta y$$

Расстояние от спутника до объекта вычисляется по формуле (4):

$$r_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}$$

Псевдорасстояние включает в себя погрешность. Связь псевдорасстояния и предполагаемого расстояния (5):

$$psr_i = r_i + \tau$$

Уравнение окружности нелинейно, применяется разложение в ряд Тейлора, берутся первые два слагаемых. Выражение для псевдорасстояний (6):

$$psr_i = r_i + \frac{x_0 - x_i}{r_i}\Delta x + \frac{y_0 - y_i}{r_i}\Delta y + \tau$$

Система уравнений представляется в матричном виде. Вычисление погрешностей координат x, y и часов τ (7):

$$\begin{pmatrix} \Delta x \\ \Delta y \\ \tau \end{pmatrix} = \begin{pmatrix} \frac{x_0 - x_1}{r_1} & \frac{y_0 - y_1}{r_1} & 1 \\ \frac{x_0 - x_2}{r_2} & \frac{y_0 - y_2}{r_2} & 1 \\ \frac{x_0 - x_3}{r_3} & \frac{y_0 - y_3}{r_3} & 1 \\ \frac{x_0 - x_4}{r_4} & \frac{y_0 - y_4}{r_4} & 1 \\ \frac{x_0 - x_5}{r_5} & \frac{y_0 - y_5}{r_5} & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} prs_1 - r_1 \\ prs_2 - r_2 \\ prs_3 - r_3 \\ prs_4 - r_4 \\ prs_5 - r_5 \end{pmatrix}$$

$$C = A^{-1} \cdot B$$

Первая матрица A^{-1} является псевдообратной, поскольку это не квадратная матрица.

Псевдообратные матрицы представляют собой обобщение обратных матриц в линейной алгебре. Псевдообращение можно рассматривать как наилучшую аппроксимацию (по методу наименьших квадратов) решения соответствующей системы линейных уравнений. Псевдообращение определено для любых матриц над вещественными и комплексными числами. Псевдообратная матрица может быть вычислена с использованием собственного представления матрицы.

Вычисляется матрица C, содержащая погрешности по координатам и времени. Предполагаемое местоположение пересчитывается по формуле (3). Последние две операции выполняются n раз для минимизации невязок.

После m итераций (при m < n) минимизация достигает своего предела при n от ~100 (определено эмпирическим путем).

Найденное предполагаемое местоположение вместе с исходными и процессионными данными отображается графически.

```
Наивероятностное положение (99.9148, 199.3946)
Погрешности координат и времени:
dx = 8.570752864828289e-14
dy = -4.6858439356077776e-14
dt = 30.53469876137026
```

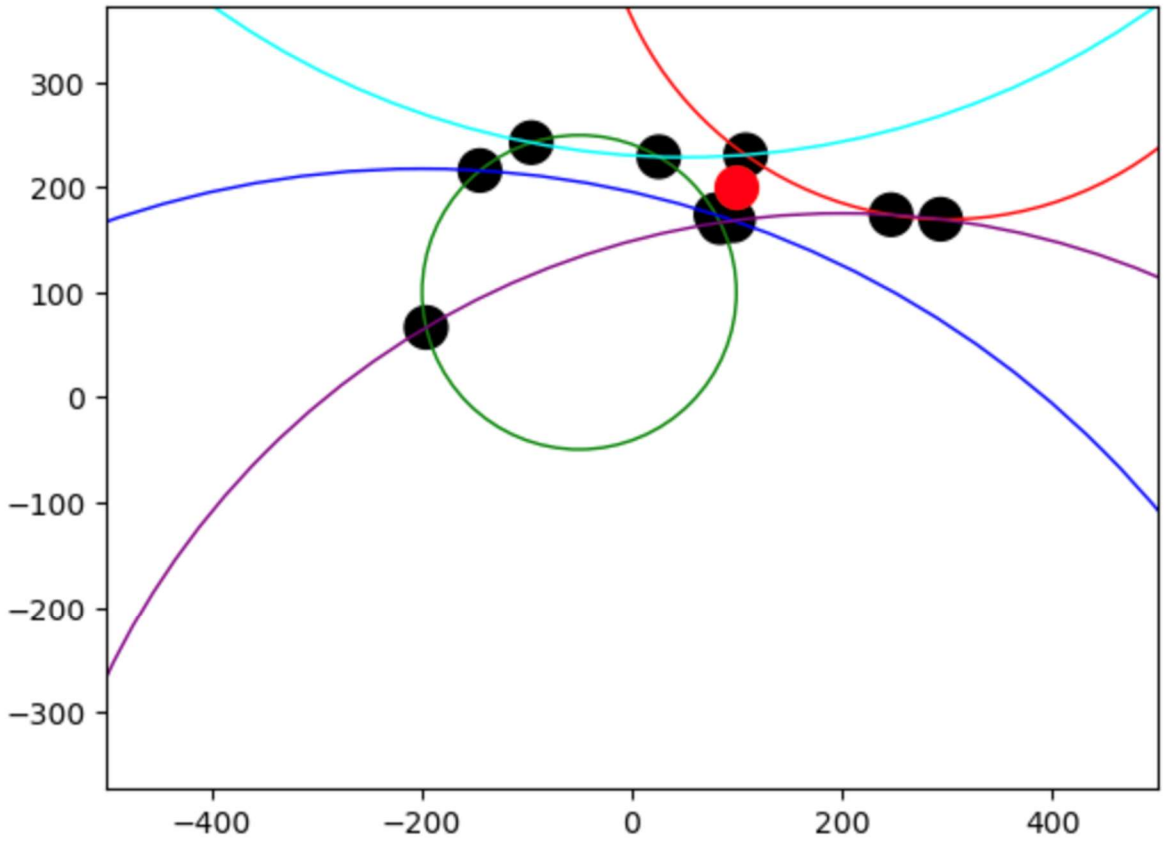

Рис. 8. Окончательно найденное местоположение приёмника при 5 маяках

Основная часть 2

Проводится анализ изменения оценок координат и часов приемника при отказе одного и двух маяков. Данные одного из спутников исключаются из расчетов.

```
Наивероятностное положение (99.9691, 199.3264)
Погрешности координат и времени:
dx = 4.618527782440651e-14
dy = -1.1191048088221578e-13
dt = 30.49500329973851
```

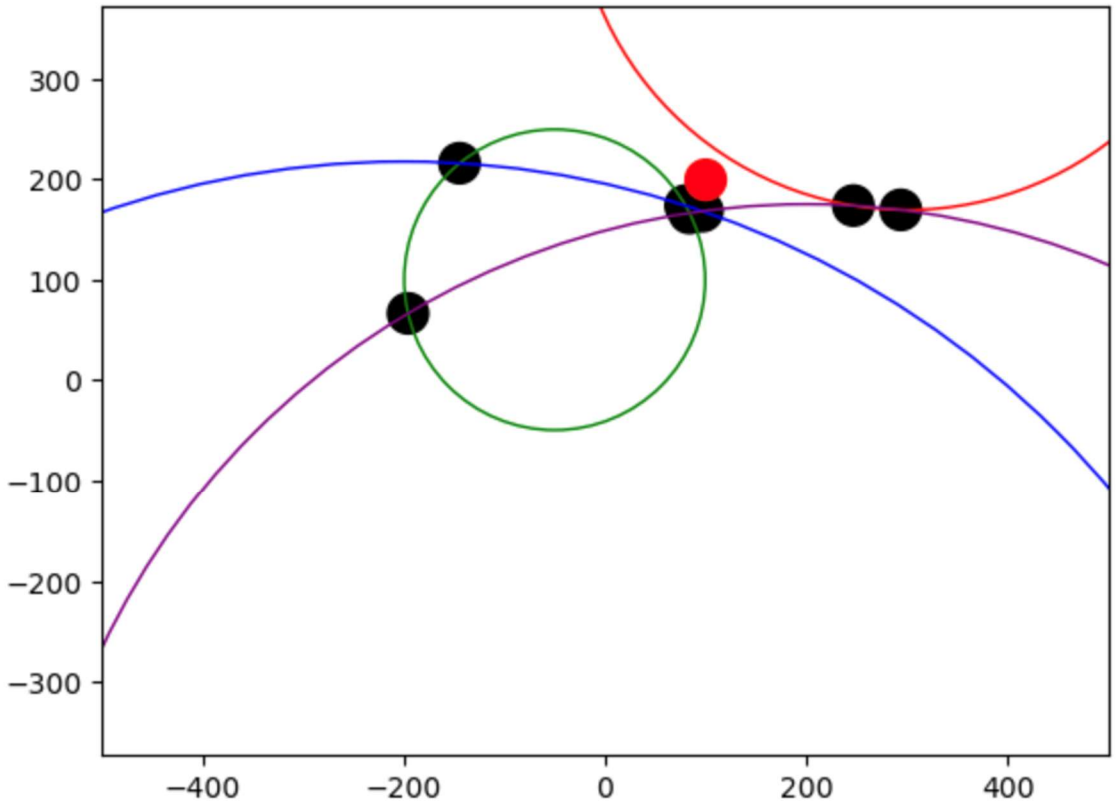

Рис. 9. Окончательно найденное местоположение приёмника при 4 маяках

Теперь исключается спутник, дающий наибольшее количество точек пересечения.

```
Наивероятностное положение (99.7091, 199.4169)
Погрешности координат и времени:
dx = -9.237055564881302e-14
dy = 7.283063041541027e-14
dt = 30.57005380230097
```

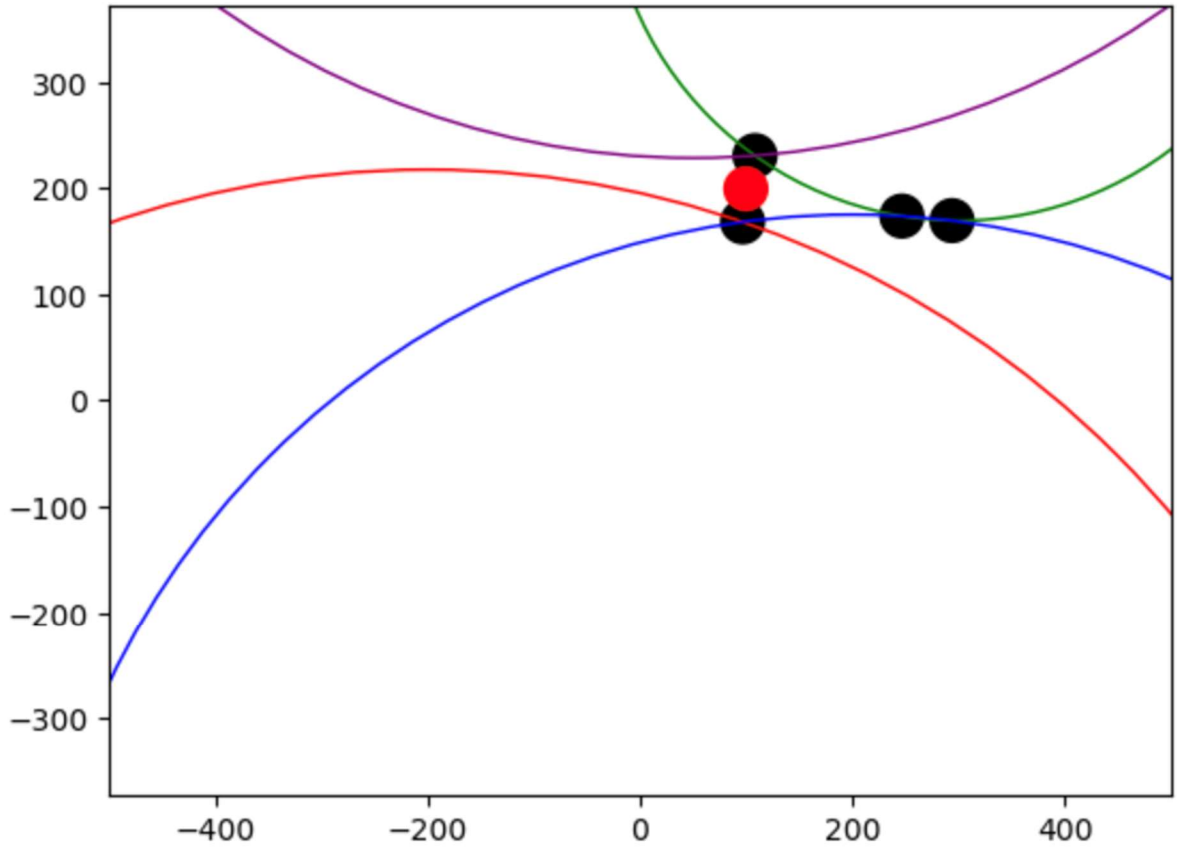

Рис. 10. Окончательно найденное местоположение приёмника при 4 маяках (2 случай)

Также проверяется работа программы при использовании данных только от трех спутников.

```
Наивероятностное положение (99.7137, 199.4148)
Погрешности координат и времени:
dx = 2.520442433990584e-13
dy = -1.4568032349397234e-13
dt = 30.56848368391586
```

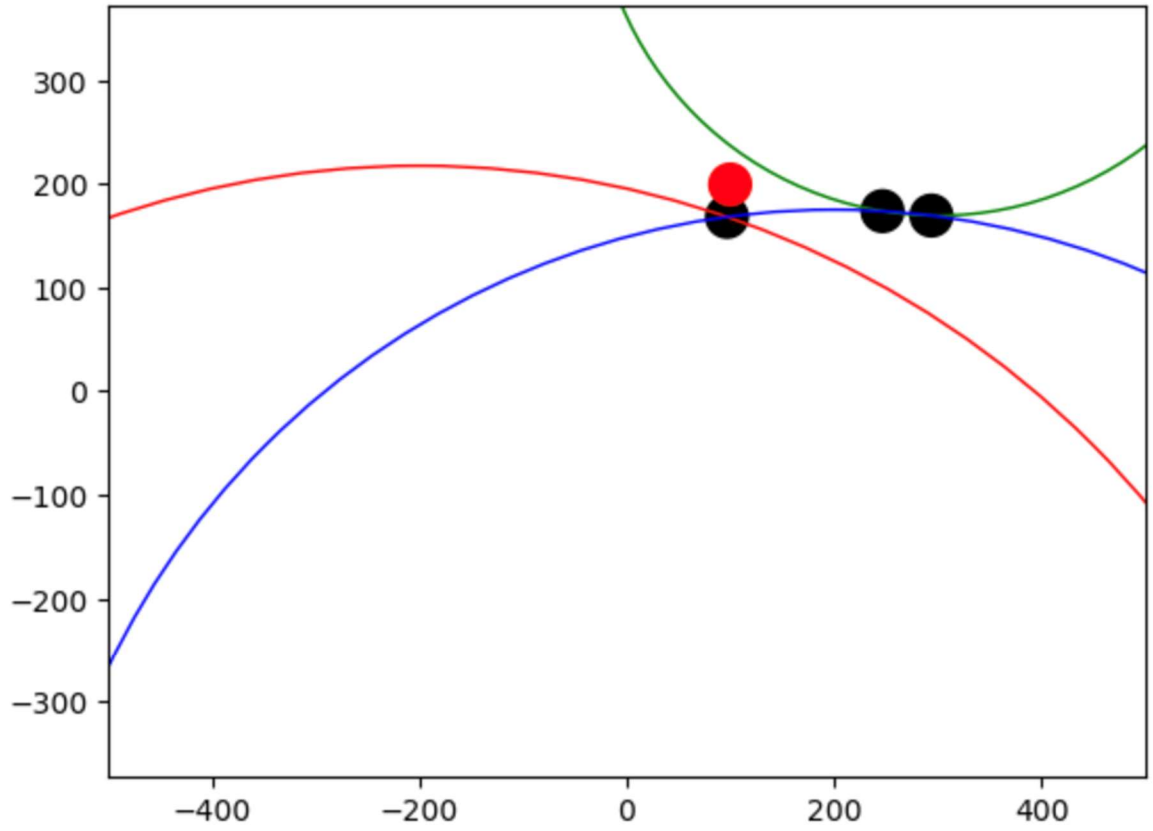

Рис. 11. Окончательно найденное местоположение приёмника при 3 маяках

По вычисленным координатам и погрешностям часов, отображенным над графиками, видно, что при меньшем объеме входных данных полученный результат несколько отличается.

Основная часть 3

Было разработано моделирующее программное обеспечение, которое автоматически вычисляет предполагаемое местоположение объекта на основе введенных данных через интерфейс Python и визуализирует их графически.

Помимо использования предоставленных данных, существует возможность самостоятельно задать входные параметры.

```
Укажите количество спутников:
 4
Укажите координаты x, y 1 спутника
 300, 500
Укажите расстояние до 1 спутника
 330
Укажите координаты x, y 2 спутника
 200, 700
Укажите расстояние до 2 спутника
 300
Укажите координаты x, y 3 спутника
 -200, -700
Укажите расстояние до 3 спутника
 950
Укажите координаты x, y 4 спутника
 50, 50
Укажите расстояние до 4 спутника
 100
[(300.0, 500.0), (200.0, 700.0), (-200.0, -700.0), (50.0, 50.0)] [330.0, 300.0, 950.0, 100.0]
```

Рис. 12. Тестирование с ручным вводом данных

Приложение

```
from matplotlib import pyplot as plt
import numpy as np
import math
import sympy as sym


def equations(x, y, r):
    return ((x - sym.Symbol('x')) ** 2 + (y - sym.Symbol('y')) ** 2 - r ** 2)


def distance(x1, y1, x2, y2):
    return (math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2))


def collision(eq1, eq2):
    x, y = sym.symbols('x, y')
    roots = sym.solve([sym.Eq(eq1, 0), sym.Eq(eq2, 0)], [x, y])
    return roots


def drawPoints(x, y, r, color):
    circle = plt.Circle((x, y), r, color=color)
    plt.gca().add_artist(circle)


def drawCircle(center, r, color):
    circle = plt.Circle(center, r, color=color, fill=False)
    plt.gca().add_artist(circle)


def finder_collision(points, radius):
    point_col = []
    ps = []
    for i in range(len(radius) - 1):
        for j in range(i + 1, len(radius)):

            d = distance(points[i][0], points[i][1], points[j][0],
points[j][1])

            if d != 0 and not (d > (radius[j] + radius[i]) or d <
(abs(radius[j] - radius[i]))):
                eq1 = equations(points[i][0], points[i][1], radius[i])
                eq2 = equations(points[j][0], points[j][1], radius[j])
                ps = collision(eq1, eq2)

                for k in range(len(ps)):
                    point_col.append(ps[k])
    return point_col


def find_pos(point_col):
    x = 0
    y = 0
    for xi, yi in point_col:
```

```
        x += xi
        y += yi

    return float(x / len(point_col)), float(y / len(point_col))


def find_real_pos(x, y, points, radius):
    dx = 0
    dy = 0

    for i in range(20):
        x += dx
        y += dy

        avg = []
        matr = []

        for i in range(len(radius)):
            r = math.sqrt((points[i][0] - x) ** 2 + (points[i][1] - y) ** 2)
            avg.append([radius[i] - r])
            matr.append([(x - points[i][0]) / r, (y - points[i][1]) / r, 1])

        [[dx], [dy], [dt]] = np.matmul(np.linalg.pinv(np.array(matr)),
np.array(avg))

    return x, y, dx, dy, dt

def draw(points, radius, col_points, size):
    COLOR = ['green', 'red', 'blue', 'purple', 'cyan', 'maroon']

    plt.axis([-1 * size, size, -1 * size, size])
    plt.axis("equal")

    for xi, yi in col_points:
        drawPoints(xi, yi, 20, 'black')

    for i in range(len(radius)):
        drawCircle(points[i], radius[i], color=COLOR[i])

def predict(points, radius):
    mypoints = finder_collision(points, radius)

    draw(points, radius, mypoints, 1000)

    x, y = find_pos(mypoints)
    print(f'Предпологаемое положение ({x:.4f}, {y:.4f})')
    drawPoints(x, y, 20, color='red')

def exect(points, radius):
    mypoints = finder_collision(points, radius)
    x, y = find_pos(mypoints)

    x, y, dx, dy, dt = find_real_pos(x, y, points, radius)

    print(f'Наивероятностное положение ({x:.4f}, {y:.4f})')
    print(f'Погрешности координат и времени:')
    print(f'dx = {dx}')
```

```
    print(f'dy = {dy}')
    print(f'dt = {abs(dt)}')


    draw(points, radius, mypoints, 500)

    drawPoints(x, y, 20, 'red')


def get_data():
    points = []
    radius = []

    print("Укажите количество спутников:")
    n = int(input())

    for i in range(n):
        print(f'Укажите координаты x, y {i + 1} спутника')
        x, y = map(float, input().split(', '))
        points.append((x, y))

        print(f'Укажите расстояние до {i + 1} спутника')
        radius.append(float(input()))

    return points, radius


points = [(-50, 100), (300, 500), (-200, -700), (200, -600), (50, 1000)]
radius = [149.4139, 330.6324, 917.4691, 775.1122, 771.5541]

predict(points, radius)
exect(points, radius)

points, radius = get_data()
print(points, radius)

predict(points, radius)
exect(points, radius)
```