МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка ПО определения местоположения объекта на основании задержки сигнала от реперных точек

Обучающийся Е.А. Бубнов
(Инициалы Фамилия) (личная подпись)

Руководитель старший преподаватель, С.В. Митин
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант кандидат педагогических наук, доцент, С.А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

В данной бакалаврской работе проводится исследование в области навигационных систем с акцентом на разработку алгоритма определения местоположения объекта на основании задержки сигнала от реперных точек. Рассматриваются два основных метода навигации — время полёта (TOF) и разность времени прибытия (TDOA), их принципы работы, требования и потенциальные области применения.

В работе проводится глубокий анализ этих методов и решается математическая задача определения положения на плоскости по заданной разности сигналов. Работа включает в себя теоретическую и практическую части, а также заключение, содержащее обобщение и выводы.

Алгоритм тестировался на сгенерированных данных, и результаты тестирования показали его высокую точность. Однако было обнаружено, что алгоритм подвержен шуму, что может снижать его точность в реальных условиях.

Работа представляет собой вклад в область навигационных систем и открывает новые возможности для дальнейших исследований в этой области. Работа состоит из 4 глав, содержит 9 формул и 30 рисунков, общий объем работы составляет 62 страницы.

Ключевые слова: НАВИГАЦИОННЫЕ СИСТЕМЫ, TOF, TDOA, АЛГОРИТМ, МЕСТОПОЛОЖЕНИЕ, ЗАДЕРЖКА СИГНАЛА, РЕПЕРНЫЕ ТОЧКИ, ТЕСТИРОВАНИЕ, ШУМ, ВЫСОКАЯ ТОЧНОСТЬ.

Abstract

This bachelor’s thesis investigates the domain of navigation systems, with a focus on developing an algorithm for object localization based on signal delay from reference points. It examines two primary navigation methods: Time of Flight (TOF) and Time Difference of Arrival (TDOA), their operational principles, requirements, and potential applications.

The work presents a comprehensive analysis of these methods and addresses the mathematical problem of determining position on a plane based on given signal differences. The thesis encompasses theoretical and practical parts, as well as a conclusion that summarizes findings and implications.

The algorithm was tested on generated data, and the test results demonstrated its high accuracy. However, it was found to be susceptible to noise, which could reduce its precision in real-world conditions.

The thesis contributes to the field of navigation systems and opens new avenues for further research in this area. It is structured into four chapters, includes 9 formulae and 30 figures, with a total length of 62 pages.

Keywords: NAVIGATION SYSTEMS, TOF, TDOA, ALGORITHM, LOCALIZATION, SIGNAL DELAY, REFERENCE POINTS, TESTING, NOISE, HIGH ACCURACY.

Оглавление

Введение

Значимость задачи определения координат объектов в пространстве и их соотнесения с координатной сеткой бесспорна, особенно при условиях прямой видимости. Это ключевой аспект в различных сферах, в том числе при навигации объектов по заданной траектории и картографировании территорий.

Целью настоящего исследования является создание методики и программного продукта для определения позиций объектов с использованием данных о задержках сигналов от реперных точек. В рамках работы будут рассмотрены существующие методы решения данной задачи, разработана математическая модель и на её основе выполнена программная реализация предложенного метода. Завершающим этапом станет проверка и оценка эффективности созданного программного решения.

В исследовании применяется гиперболический метод определения местоположения, который был выбран за его распространённость в схожих системах и соответствие характеристик гиперболы поставленным задачам. Гипербола представляет собой геометрическое место точек, разность расстояний, до которых от двух фиксированных точек (фокусов) остаётся неизменной.

Разработанный программный комплекс способен функционировать в автономном режиме, не требуя подключения к спутниковым системам.

1 Теоретическая основа исследования

1.1 Введение в навигационные системы

Навигационные системы — неотъемлемый элемент современности, предоставляющий точные данные о расположении объектов. Их применение охватывает широкий спектр сфер, от автонавигации до сложных военных и космических операций. Имеется множество типов навигационных систем, каждая со своими уникальными характеристиками и сферами использования. К ним относятся наземные, морские, аэронавигационные и спутниковые системы, применяющие разнообразные методики определения позиций, включая трилатерацию и триангуляцию.

Триангуляция — это процесс определения местонахождения объекта через измерение углов к нему с известных точек на земле, в отличие от измерения расстояний. Этот метод широко применяется в навигационных системах, где углы измеряются с помощью радиосигналов.

Принцип угла прибытия (AOA) используется для определения направления прихода сигнала, основываясь на угле, под которым сигнал достигает приёмника от передатчика. Этот принцип активно используется в радионавигации для определения источника сигнала [1, 10]. Однако, использование AOA требует наличия антенн с точной ориентацией и возможностью измерения угла прихода сигнала, что может быть затруднительно в определённых условиях, например, при движении антенны или источника сигнала.

Трилатерация же базируется на измерении расстояний. В навигационных системах она используется на основе принципов времени полёта (TOF) или разности времени прибытия (TDOA), определяющих дистанцию до объекта через время, необходимое сигналу для достижения приёмника от передатчика, или разность времён прихода сигнала от нескольких передатчиков к приёмнику. Данный раздел подробно осветит ключевые аспекты этих систем,

их достоинства и ограничения, а также возможные области применения. Это даст более глубокое понимание того, как данные методы могут быть адаптированы для создания собственного программного продукта по определению позиций объектов.

1.1.1 Принцип TOF

Принцип TOF — это методика, согласно которой приёмник вычисляет своё положение, исходя из времени прохождения сигнала. Этот метод базируется на свойствах окружности, где отправная точка служит центром, а радиус определяется как расстояние, вычисляемое на основе времени прихода сигнала к целевой точке. Позиция объекта определяется точкой пересечения нескольких таких окружностей [3, 10, 12, 15, 16].

На рисунке 1 демонстрируются три окружности, каждая из которых соответствует сигналу, исходящему от реперной точки (S1, S2, S3). Радиусы окружностей (r1, r2, r3) представляют собой время прихода сигнала к искомой точке, рассчитанное по принципу TOF. Точка A, находящаяся в месте пересечения всех трёх окружностей, указывает на искомое местоположение. Это иллюстрирует, как метод TOF может быть применён для определения местонахождения объекта на основе времени распространения сигналов от реперных точек.

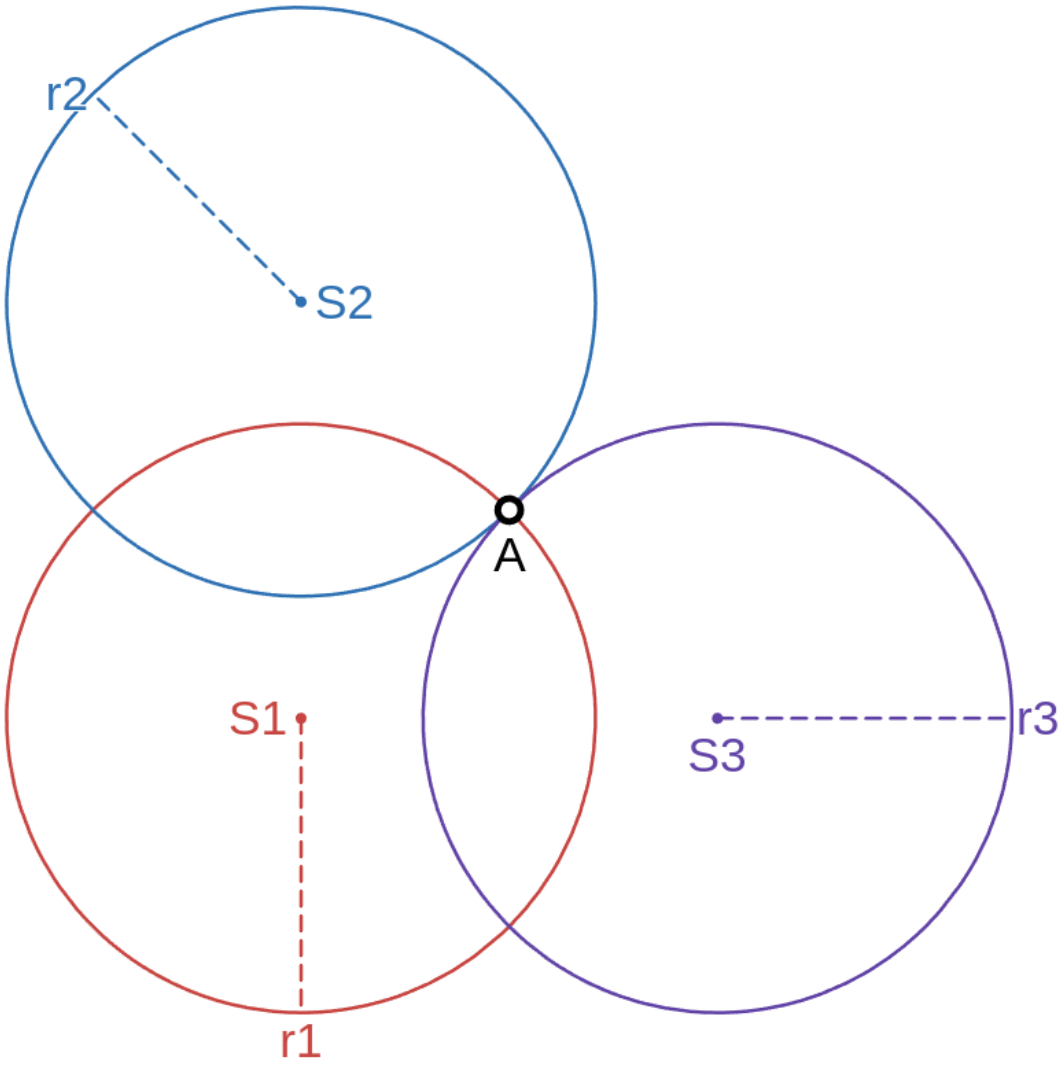


Рисунок 1 - Применение принципа TOF для определения местоположения

Основополагающим элементом метода TOF является строгая синхронизация между источником и приёмником сигнала. Малейшие неточности в измерении времени могут привести к существенным отклонениям в определении местоположения.

Несмотря на высокую эффективность, метод TOF требует сложной инфраструктуры и использования точных часов, таких как атомные часы, что может быть затратно и сложно в реализации.

Метод TOF активно используется в глобальных навигационных спутниковых системах (GNSS), таких, как GPS, ГЛОНАСС, Galileo и BeiDou. В этих системах спутники передают сигналы с точным временем отправки, что

позволяет приёмнику вычислить своё положение, основываясь на времени приёма сигнала.

1.1.2 Принцип TDOA

Принцип TDOA — это методика, позволяющая приёмнику вычислять своё положение на основе временной разницы прихода сигналов от двух или более реперных точек. В отличие от метода TOF, TDOA упрощает требования к синхронизации, делая его более доступным. TDOA представляет собой форму гиперболической навигации, использующую геометрические свойства гипербол для определения местоположения объекта[10, 12, 15, 16].

Гиперболическая навигация — это техника определения позиции, основанная на временной разнице прихода сигналов от нескольких реперных точек. Она базируется на характеристиках гиперболы, где для любой точки на гиперболе разность расстояний до двух фокусов остаётся постоянной.

Реперные точки, или «фокусы», обычно представлены радиостанциями, передающими сигналы на определённой частоте. Объект, чьё местоположение нужно определить, оснащён приёмником, способным измерять время прихода сигналов. Используя разницу во времени прихода сигналов от двух станций, можно вычислить положение объекта на гиперболе. Сигналы от трёх или более станций позволяют точно определить местоположение объекта.

Рисунок 2 демонстрирует использование принципа TDOA для нахождения позиции объекта. На нём изображены три реперные точки: S1, S2 и S3. Искомая точка A находится на пересечении двух гипербол:

- синяя гипербола представляет собой множество точек с одинаковой временной разницей прихода сигналов от точек S1 и S3;
- красная гипербола аналогично представляет точки с одинаковой временной разницей прихода сигналов от точек S1 и S2;
- точка A, расположенная на пересечении этих гипербол, указывает на местоположение объекта.

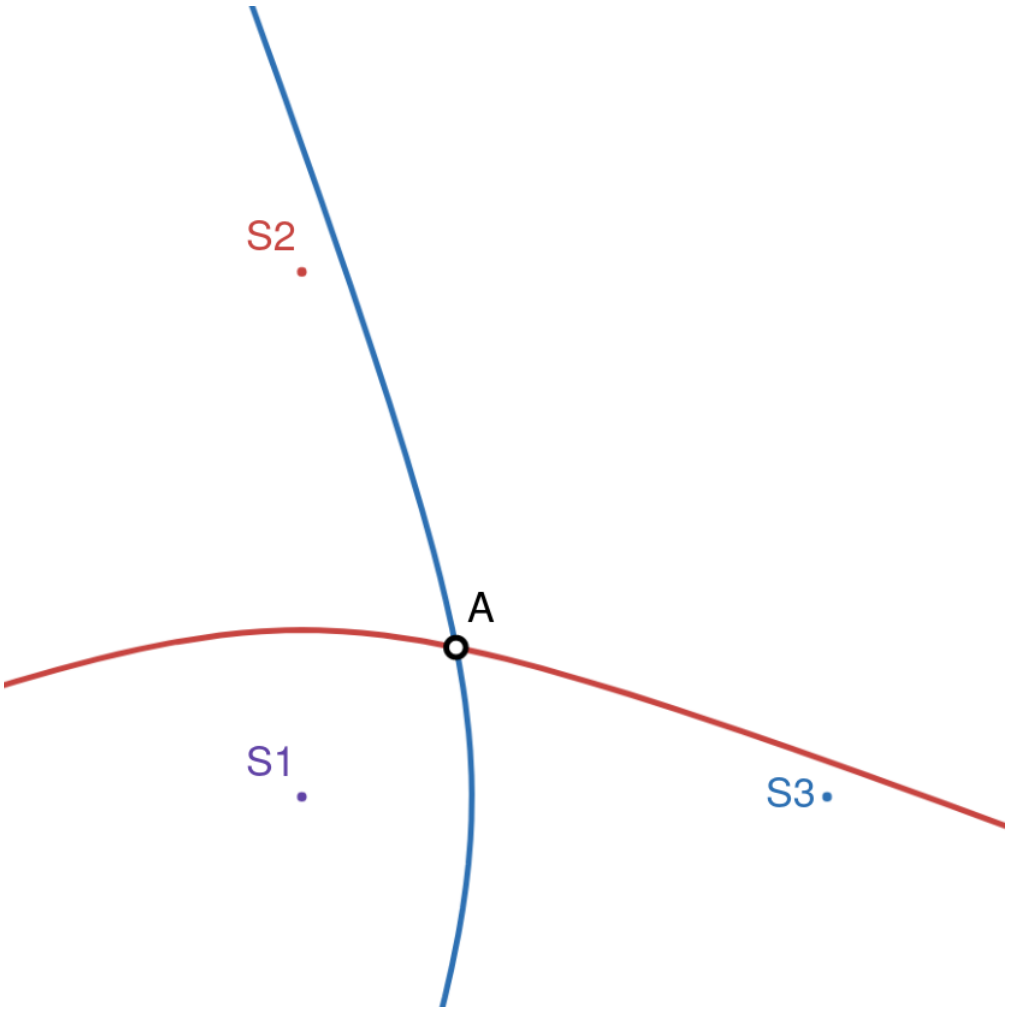


Рисунок 2 - Применение принципа TDOA для определения местоположения объекта

Гиперболическая навигация активно использовалась до появления глобальных навигационных спутниковых систем (GNSS). Примеры таких систем включают Decca, LORAN и Omega [8]. В частности:

- система LORAN (Long Range Navigation), разработанная во время Второй мировой войны, использовала низкочастотные радиосигналы от нескольких передатчиков для определения местоположения по разнице времени прихода сигналов [6, 9];
- система Decca Navigator, популярная в морской навигации середины 20 века, определяла местоположение по разнице в фазе сигналов от трёх или более станций [2].

Современные GNSS предлагают более высокую точность, надёжность и глобальное покрытие, что сделало гиперболическую навигацию менее распространённой, хотя она всё ещё применяется в некоторых специализированных приложениях [5].

1.2 Сравнение TOF и TDOA

Методы TOF и TDOA являются двумя основными методами определения местоположения в системах навигации. Они имеют разные требования к реализации и используются в разных условиях.

Метод TOF обладает рядом преимуществ он может обеспечить очень точное определение местоположения, если доступны сверхточные часы. Этот метод может быть эффективно использован в системах, где передатчик и приёмник могут быть синхронизированы, например, в некоторых военных и космических приложениях.

Однако у метода TOF есть и свои недостатки. Основной недостаток метода TOF - это необходимость в сверхточных часах для синхронизации между передатчиком и приёмником. Это может существенно усложнить реализацию и увеличить стоимость системы. Также метод TOF требует, чтобы приёмник знал время начала передачи сигнала, то есть реперная точка должна уметь передать не просто сигнал, а ещё и информацию в этом сигнале, что ещё сильнее усложняет реализацию.

Метод TDOA упрощает требования к синхронизации, так как основан на измерении временной разницы прихода сигналов от разных передатчиков. Это делает его подходящим для систем, где местоположение определяется с помощью сигналов от нескольких реперных точек.

Но и у метода TDOA есть свои недостатки. Для синхронизации требуется особое расположение реперных точек в зависимости от скорости сигнала. Это может ограничить применение метода TDOA в некоторых сценариях. Так как реперные точки не могут одновременно отправлять сигналы, несмотря на то,

что метод TDOA упрощает требования к синхронизации, он всё равно требует, чтобы одна из реперных точек (мастер) имела часы для синхронизации сигналов. Она в определённое время, по цепочке, передаёт сигнал на соседние точки (вторичные), этот сигнал должен прийти на соседние за определённые промежуток времени, после первая вторичная точка также передаёт сигнал на следующую и так далее. Например, если это радиостанции, которые передают сигнал за одну миллисекунду, то станции должны находиться на расстоянии минимум 300 километров [9].

После анализа и сравнения, гиперболическая навигация на основе TDOA была выбрана для дальнейшей работы из-за её практичности в реальных условиях и отсутствия необходимости в сверхточных часах для синхронизации, что снижает стоимость и сложность системы, делая её доступной для широкого круга пользователей. Это соответствует цели проекта — создать доступную и эффективную навигационную систему.

В данной главе были освещены ключевые аспекты методов TOF и TDOA, их роль в навигационных системах и потенциал для определения местоположения объектов. Обсуждение включало в себя анализ требований к каждому методу и их применимость в разнообразных сферах.

Были также детально рассмотрены принципы гиперболической навигации и её использование в системах позиционирования, основанное на измерении разности времени прибытия сигналов от реперных точек.

В заключение, методы TOF и TDOA играют значительную роль в обеспечении точности определения местоположения объектов, что критично для многих современных приложений. Выбор между этими методами зависит от специфических требований к точности, зоне покрытия, доступности оборудования и других факторов.

Этот обзор теоретических основ подводит итог первой главы исследования. В последующих главах мы перейдём к практическим аспектам, включая решение математической задачи определения позиции объектов на плоскости и разработку соответствующего алгоритма.

2 Решение математической задачи определения положения на плоскости по заданной разности сигналов

2.1 Математические основы алгоритма

Продолжая тему основных принципов TOF и TDOA, изложенных в предыдущем разделе, данный раздел посвящён практическим аспектам метода TOF и его математическому обоснованию для задач позиционирования объектов на плоскости. Здесь будут изложены ключевые принципы, необходимые для решения практических задач навигации.

В алгоритме термин «дистанция» будет использоваться для обозначения расстояния между объектами. Евклидово расстояние определяется как длина прямой линии между двумя точками в пространстве, где действуют стандартные геометрические законы. Это расстояние вычисляется на основе теоремы Пифагора, и поэтому его иногда называют «пифагоровым расстоянием». Для определения евклидового расстояния необходимы лишь координаты этих двух точек [4, 11].

В математической форме евклидово расстояние между точками $A(x_1, y_1)$ и $B(x_2, y_2)$ в двумерном пространстве может быть выражено как:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

где d — евклидово расстояние между двумя точками или же разница во времени прихода сигналов от реперных точек, преобразованное в расстояние;

x_2, y_2 — координаты второй точки;

x_1, y_1 — координаты первой точки.

Эта формула является фундаментальной для алгоритмов, которые используют геометрические расстояния для определения положения объектов.

Исходя из принципов гиперболической навигации, алгоритм, который использует разницу во времени прихода сигналов от n реперных точек для

определения местоположения объекта, может быть представлен системой уравнений. Эти уравнения основаны на гиперболах, которые формируются в результате разности расстояний до пары реперных точек. Для объекта, находящегося в точке P(x, y), и реперных точек $R_i(x_i, y_i)$, уравнения могут быть выражены как:

$$\sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_i)^2+(y-y_i)^2}=d_i, i \in 1,2,..,n \quad (2)$$

где x, y — координаты точки, которую мы пытаемся найти;

x_0, y_0 — координаты реперной точки «мастер»;

x_i, y_i — координаты реперных точек «вторичные»;

d_i — это разницы во времени прихода сигналов от реперных точек, преобразованные в расстояния;

i — номер реперной точки;

n — количество реперных точек.

Так как d — расстояние, то оно может быть представлено следующим образом:

$$d = c \times \Delta t \quad (3)$$

где c — скорость передачи сигнала;

Δt — разница во времени прихода сигналов от реперных точек.

Эта система уравнений (2), представляет собой n гипербол, и решение этой системы даст точное местоположение объекта.

В алгоритме, который мы рассматриваем, одна из точек, обозначенная как R_0(x_0, y_0), выполняет функцию «главной» или «мастер-точки». Остальные точки, соответственно, являются «второстепенными» или «слейв-точками». «Мастер-точка» служит эталоном для измерения временной разницы прихода

сигналов от других точек, что является стандартной процедурой в гиперболических навигационных системах и способствует синхронизации данных между всеми реперными точками. Кроме того, важно подчеркнуть, что количество реперных точек n должно быть не менее трех, чтобы обеспечить возможность точного определения местоположения объекта в пространстве. Это минимальное количество необходимо для формирования достаточного числа гиперболических уравнений, позволяющих вычислить координаты объекта с учетом его движения и расположения реперных точек.

Для примера, попробуем самостоятельно найти координаты неизвестной точки. В данном примере рассматривается задача определения координат неизвестной точки. Используются четыре реперные точки с координатами A(0, 0) — главная реперная точка, B(0, 10), C(10, 0), D(10, 10) — вторичные реперные точки. Разница во времени прихода сигналов от этих точек, преобразованная в расстояния, составляет 0 для каждой из них.

Сначала составляется система уравнений, основанная на формуле евклидова расстояния:

$$\begin{cases} \sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_1)^2+(y-y_1)^2}=0 \\ \sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_2)^2+(y-y_2)^2}=0 \\ \sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_3)^2+(y-y_3)^2}=0 \end{cases}$$

Подставляем известные переменные, делаем перенос в правую часть:

$$\begin{cases} \sqrt{(x-0)^2+(y-0)^2}=\sqrt{(x-0)^2+(y-10)^2} \\ \sqrt{(x-0)^2+(y-0)^2}=\sqrt{(x-10)^2+(y-0)^2} \\ \sqrt{(x-0)^2+(y-0)^2}=\sqrt{(x-10)^2+(y-10)^2} \end{cases}$$

Затем эти уравнения упрощаются, убирая квадратные корни:

$$\begin{cases} (x-0)^2+(y-0)^2=(x-0)^2+(y-10)^2 \\ (x-0)^2+(y-0)^2=(x-10)^2+(y-0)^2 \\ (x-0)^2+(y-0)^2=(x-10)^2+(y-10)^2 \end{cases}$$

После упрощения уравнений, получаем следующие равенства:

$$\begin{cases} y^2=(y-10)^2 \\ x^2=(x-10)^2 \\ x^2+y^2=(x-10)^2+(y-10)^2 \end{cases}$$

Затем, раскрыв квадраты и упростив выражения, получаем:

$$\begin{cases} y^2=y^2-20y+100 \\ x^2=x^2-20x+100 \end{cases}$$

Далее уравнения упрощаются ещё больше, убирая общие слагаемые и приводя подобные:

$$\begin{cases} 20y=100 \\ 20x=100 \end{cases}$$

И, наконец, решаются эти уравнения, получая координаты искомой точки:

$$\begin{cases} y=5 \\ x=5 \end{cases}$$

Следовательно, гиперболическая навигация эффективно используется для локализации объектов в двумерном пространстве. Для верификации результатов предлагается подставить вычисленные координаты x и y обратно в исходные уравнения. На рисунке 3 демонстрируется решение уравнения, где изображён график, наглядно показывающий механизм действия гиперболической навигации. Изображение показывает, что отсутствие задержки сигналов приводит к формированию прямой линии, что соответствует моменту, когда сигналы от всех реперных точек достигают объекта одновременно, располагая его на равном удалении от них. В случае, когда временная задержка сигналов отличается от нуля, на графике появляются гиперболы, обозначенные пунктирными линиями. Каждая гипербола соответствует набору точек с одинаковой временной разницей прихода сигналов от пары реперных точек. Позиция искомого объекта может быть определена в любой точке на этих гиперболах.

Итак, на рисунке 3 представлена визуализация использования гиперболической навигации для локализации объектов в двухмерном пространстве. Этот метод основывается на анализе различий во времени

прихода сигналов от фиксированных точек и позволяет точно определить положение объекта на карте.

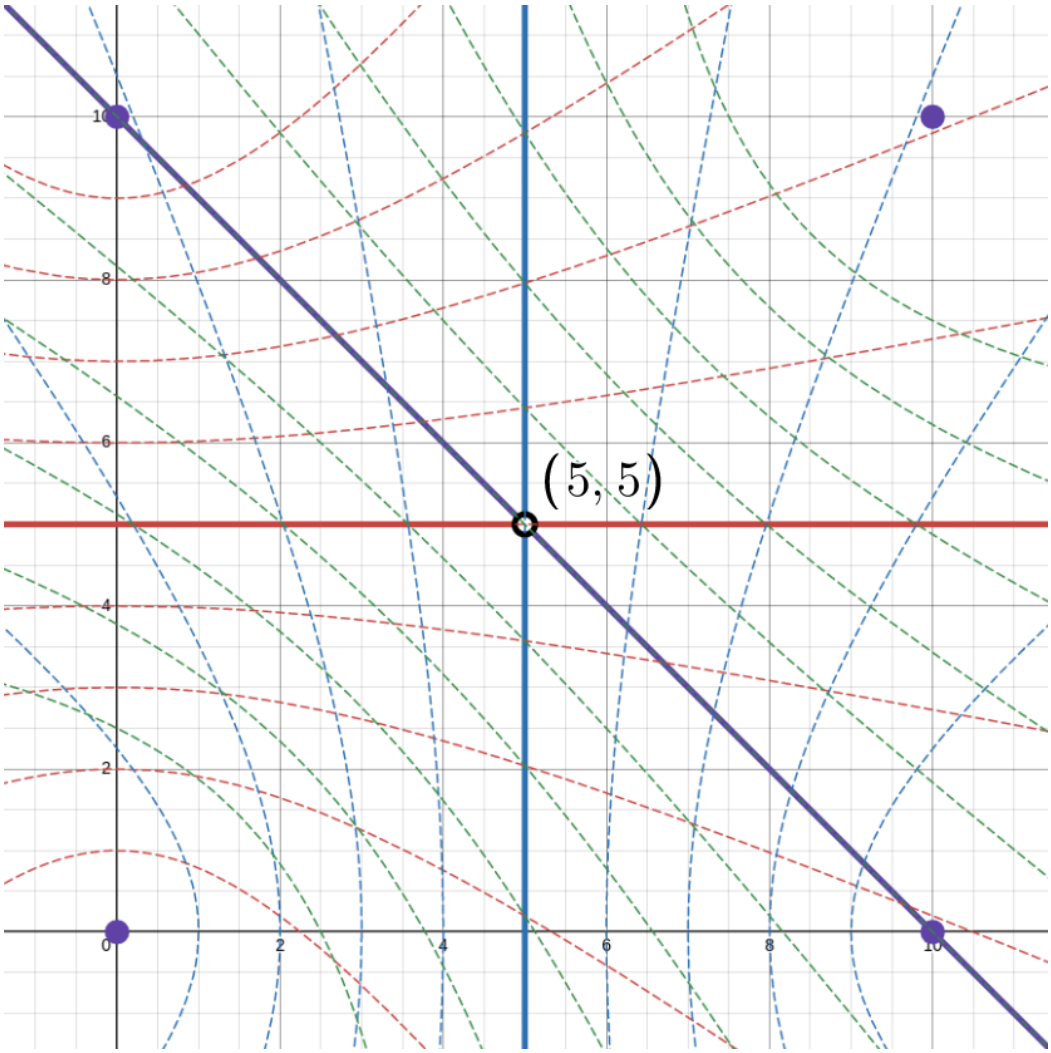


Рисунок 3 - Визуализация решения задачи определения местоположения с использованием метода гиперболической навигации

Также рассмотрим ситуацию, когда, например, разница во времени прихода сигналов от этих точек, преобразованная в расстояния, равна 3, 2, 6.305.

Сначала составляется система уравнений, основанная на формуле евклидова расстояния:

$$\begin{cases} \sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_1)^2+(y-y_1)^2}=3 \\ \sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_2)^2+(y-y_2)^2}=2 \\ \sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_3)^2+(y-y_3)^2}=6.305 \end{cases}$$

Подставляем известные переменные:

$$\begin{cases} \sqrt{(x-0)^2+(y-0)^2}-\sqrt{(x-0)^2+(y-10)^2}=3 \\ \sqrt{(x-0)^2+(y-0)^2}-\sqrt{(x-10)^2+(y-0)^2}=2 \\ \sqrt{(x-0)^2+(y-0)^2}-\sqrt{(x-10)^2+(y-10)^2}=6.305 \end{cases}$$

Убираем лишнее и раскрываем квадраты:

$$\begin{cases} \sqrt{(x^2+y^2)}=\sqrt{(x^2+y^2-20y+100)}+3 \\ \sqrt{(x^2+y^2)}=\sqrt{(x^2+y^2-20x+100)}+2 \\ \sqrt{(x^2+y^2)}=\sqrt{(x^2-20x+y^2-20y+200)}+6.305 \end{cases}$$

Найдём y из первого уравнения:

$$x^2+y^2=9+6\sqrt{(x^2+y^2-20y+100)}+x^2+y^2-20y+100$$

$$20y-109=6\sqrt{(x^2+y^2-20y+100)}$$

$$400y^2-4360y+11881=16y^2-720y+36x^2+3600$$

$$y^2-10y=\frac{(-36x^2+8281)}{364}$$

$$y^2-10y+25=\frac{(-36x^2+8281)}{364}+25$$

$$y-5=\pm\sqrt{\frac{(-36x^2+8281)}{364}+25}$$

$$y_1=\sqrt{\frac{9x^2}{91}+\frac{9}{4}}+5\ y_2=-\sqrt{\frac{9x^2}{91}+\frac{9}{4}}+5$$

Найдём x во 2 уравнении, используя положительный y1:

$$x^2+y^2=4+4\sqrt{y^2+x^2-20x+100}+y^2+x^2-20x+100$$

$$20x=104+4\sqrt{y^2+x^2-20x+100}$$

$$5x=26+\sqrt{y^2+x^2-20x+100}$$

$$5x-26=\sqrt{\frac{100x^2}{91}+10\sqrt{\frac{9x^2}{91}+\frac{9}{4}}-20x+\frac{509}{4}}$$

$$25x^2-260x+676=\frac{100x^2}{91}+10\sqrt{\frac{9x^2}{91}+\frac{9}{4}}-20x+\frac{509}{4}$$

$$\frac{2175x^2}{91}-240x+548.75=10\sqrt{\frac{9x^2}{91}+\frac{9}{4}}$$

$$\frac{4730625x^4}{8281}-\frac{1044000x^3}{91}+\frac{15257325x^2}{182}-263400x+\frac{4818025}{16}=\frac{900x^2}{91}+225$$

$$\frac{4730625\,x^4}{8281}-\frac{1044000\,x^3}{91}+\frac{15255525\,x^2}{182}-263400\,x+\frac{4814425}{16}=0$$

$$\frac{-25\left(348\,x^2-3276\,x+7007\right)\left(8700\,x^2-32820\,x+227591\right)}{132496}=0$$

$$\left(348\,x^2-3276\,x+7007\right)\left(8700\,x^2-32820\,x+227591\right)=0$$

$$\left(x^2-\frac{273\,x}{29}+\frac{7007}{348}\right)=0$$

$$\left(x^2-\frac{273\,x}{29}+\frac{74529}{3364}\right)=\frac{5096}{2525}$$

$$\left(x-273\,\frac{x}{58}\right)^2=\frac{5096}{2525}$$

$$x=\frac{273}{58}\pm\frac{14\sqrt{\frac{26}{3}}}{29}\approx\pm 3.29$$

$$\left(x^2-\frac{1574\,x}{145}+\frac{227591}{8700}\right)=0$$

$$\left(x^2-\frac{1574\,x}{145}+\frac{2393209}{84100}\right)=\frac{144872}{63075}$$

$$\left(x^2-\frac{1547\,x}{290}\right)=\frac{144872}{63075}$$

$$x=\frac{1547}{290}\pm\frac{2\sqrt{108654}}{435}\approx\pm 6.85$$

Так как реперные точки находятся в 1 четверти координатной плоскости, то x не может быть отрицательным. При дальнейшей проверки оставшихся корней x, было найдено, что x равен 6.85.

Далее найдём значение y:

$$y=\sqrt{\frac{9\left(6.85\right)^2}{91}+\frac{9}{4}}+5\approx 7.62$$

Проверим найденные корни для системы уравнений:

$$\begin{cases}\sqrt{\left(6.85\right)^2+\left(7.62\right)^2}-\sqrt{\left(6.85\right)^2+\left(7.62-10\right)^2}=3\\ \sqrt{\left(6.85\right)^2+\left(7.62\right)^2}-\sqrt{\left(6.85-10\right)^2+\left(7.62\right)^2}=2\\ \sqrt{\left(6.85\right)^2+\left(7.62\right)^2}-\sqrt{\left(6.85-10\right)^2+\left(7.62-10\right)^2}=6.305\end{cases}\Rightarrow\begin{cases}2.99463\approx 3\\ 2.00089\approx 2\\ 6.29829\approx 6.305\end{cases}$$

Таким образом можно сделать вывод, что искомая точка имеет примерные координаты (6.85, 7.62). На рисунке 4 также представлена визуализация решённой задачи.

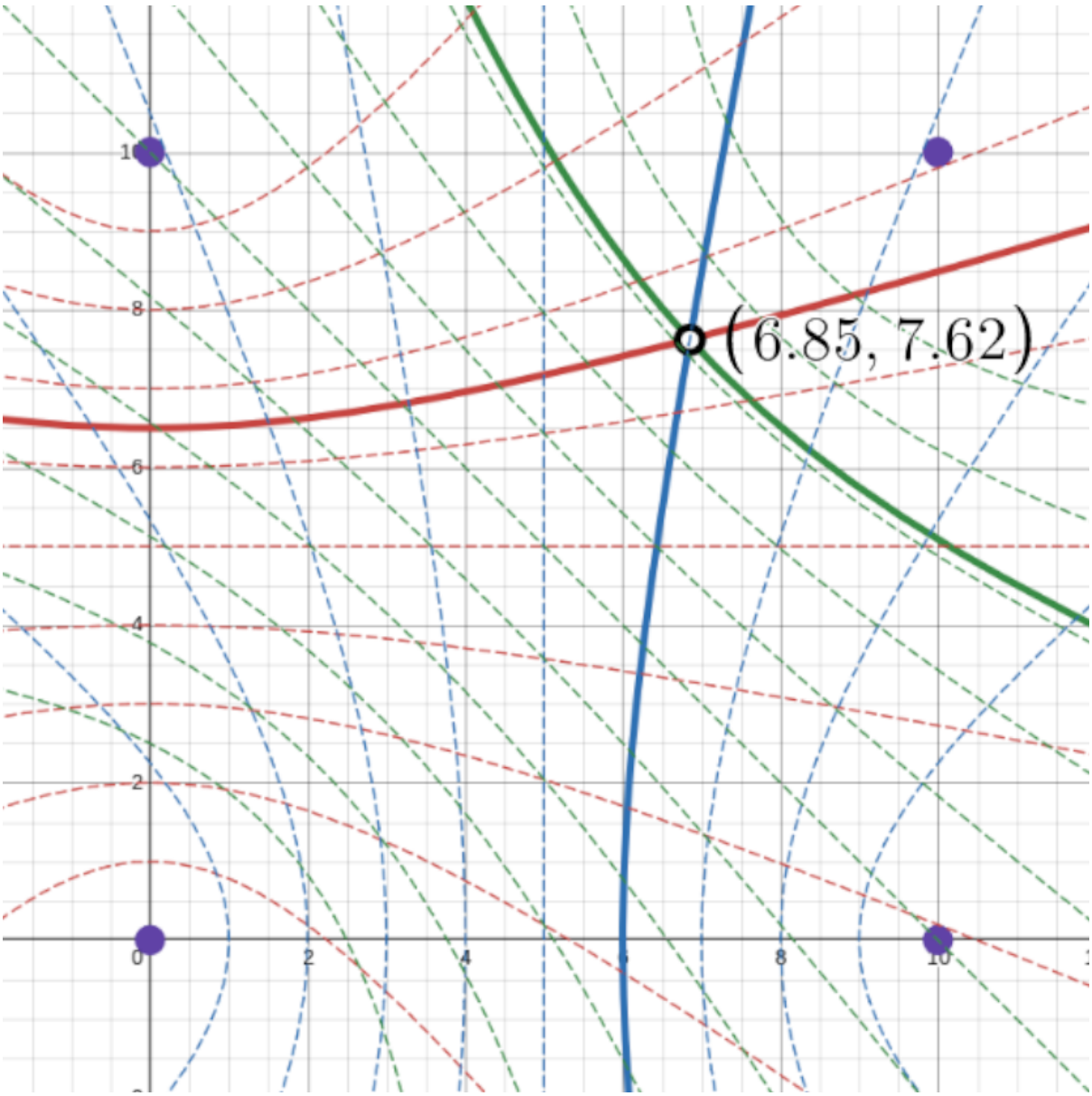


Рисунок 4 - Визуализация решения задачи определения местоположения с использованием метода гиперболической навигации

В этом разделе были рассмотрены математические основы алгоритма определения местоположения на плоскости по заданной разности сигналов. Было показано, как евклидово расстояние используется для представления дистанции между реперными точками и объектом, местоположение которого необходимо определить.

Также было продемонстрировано, как метод гиперболической навигации может быть применён для решения задачи определения местоположения объекта. Были представлены примеры, в которых использовались четыре

реперные точки и разница во времени прихода сигналов от этих точек, преобразованная в расстояния.

С помощью системы уравнений и последовательного упрощения этих уравнений, были найдены координаты искомой точки. Этот процесс был визуализирован на рисунках 3 и 4, которые демонстрируют, как метод гиперболической навигации может быть использован для определения местоположения объекта на плоскости.

Таким образом, в этом разделе были заложены основы для дальнейшего изучения и реализации метода гиперболической навигации. В следующем разделе будет рассмотрен процесс выбора и реализации метода оптимизации, который будет использован для решения системы уравнений и определения местоположения объекта.

2.2 Выбор и реализация метода оптимизации

Применение оптимизационных методов является ключевым для решения систем уравнений, включающих нелинейные функции, такие как евклидово расстояние [3, 7]. Из-за квадратного корня, входящего в выражение евклидова расстояния, прямые методы решения, наподобие метода Гаусса, оказываются неприменимы. В отличие от них оптимизационные методы способны работать с нелинейностями [20].

Оптимизационные методы отличаются повышенной робастностью по сравнению с прямыми методами, что делает их более устойчивыми к изменениям в исходных данных и позволяет эффективно справляться с шумами и отклонениями, которые часто встречаются в реальных условиях.

Эти методы также предоставляют дополнительную гибкость для моделирования сложных систем, позволяя интегрировать различные параметры и условия, что может быть важно для учета множества факторов, влияющих на систему.

Кроме того, оптимизационные методы могут достигать высокой точности, особенно при использовании итеративных подходов, таких как метод Ньютона или градиентный спуск, которые приближаются к оптимальному решению с каждой новой итерацией, улучшая точность итогового результата.

Метод Ньютона, известный также как метод Ньютона-Рафсона, представляет собой итеративный численный метод поиска корней функции или решений систем уравнений. Он использует принцип приближения функции к касательной и определения точки пересечения этой касательной с осью абсцисс в качестве следующего приближения к корню.[19]

Для одного уравнения, метод Ньютона может быть записан следующим образом:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (4)$$

где x_{n+1} — следующее приближение;

x_n — текущее приближение;

$f(x_n)$ — значение функции в точке текущего приближения;

$f'(x_n)$ — значение производной функции в точке текущего приближения.

Метод Ньютона-Рафсона используется для последовательного уточнения решения до достижения заданного уровня точности. Этот метод подходит для нахождения корней системы уравнений. Процесс начинается с выбора начального приближения к положению объекта и последовательно совершенствует это приближение, двигаясь в направлении, которое уменьшает различия между фактическими и расчётными временами прибытия сигналов.

Градиентный спуск представляет собой итерационный метод оптимизации, применяемый для нахождения минимального значения функции. Основная идея метода заключается в перемещении в направлении наибольшего убывания функции, определяемого её градиентом. Этот процесс базируется на предположении, что если функция ошибки аппроксимируется плоскостью, то

уменьшение функции будет наиболее эффективным при движении в направлении противоположному градиенту. Градиентный спуск может быть использован для минимизации функции ошибки, которая вычисляет разницу между измеренными и расчетными расстояниями до реперных точек.

Алгоритм градиентного спуска инициируется с исходного приближения к положению объекта и последовательно улучшает это приближение, двигаясь в направлении, которое уменьшает функцию ошибки. Достижение этого происходит за счет обновления текущего приближения в сторону отрицательного градиента функции ошибки.

Формула градиентного спуска может быть записан следующим образом:

$$x_{n+1} = x_n - \alpha \nabla cost\left(x_n\right) \quad (5)$$

где x_{n+1} — следующее приближение;

x_n — текущее приближение;

α — скорость обучения, которая определяет размер шага на каждой итерации;

$\nabla cost(x_n)$ — градиент функции стоимости в точке текущего приближения.

Градиентный спуск используется итеративно, пока не будет достигнута необходимая точность. Это эффективный и широко используемый метод оптимизации, особенно при работе с большими объёмами данных или сложными функциями. Однако выбор соответствующей скорости обучения может быть проблемой, так как слишком большая скорость обучения может привести к нестабильности, и слишком мало скорость обучения может привести к медленной сходимости [17].

Выбор градиентного спуска как способ оптимизации для этой задачи обусловлен несколькими ключевыми факторами. Прежде всего, градиентный спуск прост и эффективен.

Вторым важным аспектом является способность градиентного спуска эффективно работать с нелинейными функциями, такими как функция стоимости в данной задаче. Это делает его подходящим для использования в

задачах, где функция стоимости имеет сложную форму или не может быть легко решена аналитически.

Третьим фактором является способность градиентного спуска сходиться к глобальному минимуму функции стоимости при правильном выборе скорости обучения и начального приближения. Это особенно важно в задачах, где функция стоимости имеет несколько локальных минимумов.

Четвёртым аспектом является масштабируемость градиентного спуска. Он хорошо масштабируется на большие объёмы данных. Это делает его подходящим для использования в реальных задачах, где количество данных может быть велико.

Пятый фактор — это широкая применимость градиентного спуска в машинном обучении. Он является стандартным методом оптимизации и имеет широкую поддержку в различных библиотеках и фреймворках. Это облегчает его использование и интеграцию в различные системы.

Кроме того, градиентный спуск отличается простотой реализации. Он не требует сложных вычислений или дополнительных структур данных. Это делает его доступным для широкого круга разработчиков и исследователей.

Градиентный спуск также может быть более эффективным, чем некоторые другие методы оптимизации, когда есть большое количество параметров для оптимизации. Это особенно важно в задачах машинного обучения и искусственного интеллекта, где модели могут иметь тысячи, а иногда и миллионы параметров.

Наконец, градиентный спуск позволяет контролировать скорость обучения, что может помочь в настройке процесса обучения. Это позволяет балансировать между скоростью сходимости и вероятностью пропуска минимума.

Все эти причины делают градиентный спуск хорошим выбором для решения данной задачи определения местоположения. Однако, как и любой метод оптимизации, градиентный спуск требует тщательной настройки параметров и проверки его работы на практике.

Для задания градиентного спуска потребуется определить функцию стоимости и градиент функции стоимости. В качестве функции стоимости используем метод наименьших квадратов.

Метод наименьших квадратов — это стандартный выбор для функции стоимости во многих задачах оптимизации, включая задачи, связанные с машинным обучением и определением местоположения.

Этот метод минимизирует сумму квадратов разностей между предсказанными и истинными значениями. Это делает его особенно подходящим для задач, где важно учесть все данные и минимизировать общую ошибку.

Функция стоимости, основанная на методе наименьших квадратов, будет измерять общую разницу между измеренными и теоретическими расстояниями до каждой из реперных точек. Градиент этой функции стоимости затем будет использоваться в алгоритме градиентного спуска для обновления текущего приближения местоположения объекта.

Функция стоимости будет выглядеть следующим образом:

$$cost(x, y) = \sum_{i=1}^{n} \left(\sqrt{(x - x_0)^2 + (y - y_0)^2} - \sqrt{(x - x_i)^2 + (y - y_i)^2} - d_i \right)^2 \quad (6)$$

где x, y — координаты объекта, местоположение которого необходимо определить;

n — количество реперных точек;

x_0, y_0 — координаты главной реперной точки;

x_i, y_i — координаты вторичных реперных точек;

d_i — разница во времени прихода сигналов от реперных точек, преобразованная в расстояние.

Эта функция стоимости измеряет общую разницу между измеряемыми и теоретическими расстояниями к каждому из реперных точек. Затем градиент

этой функции стоимости будет использоваться в алгоритме для обновления текущей приближения.

Частные производные функции стоимости по x и y должны быть определены для определения градиента функции стоимости.

Эти производные показывают, как функция стоимости меняется с небольшими изменениями x и y. Вместе они формируют градиент функции значения, что указывает на направление наибольшего увеличения функции. В градиентном спуске идёт движение в противоположном направлении (отсюда минус в формуле 5) для минимизации функции значения.

Вычисление этих производных может быть проблемой, особенно для сложных задач, поэтому задача была решена через сервис WolframAlpha.

WolframAlpha - это высокоинтеллектуальная вычислительная система, которая отвечает на вопросы, основанные на собственной обширной базе данных и алгоритмов. Он способен обрабатывать запросы естественного языка и предоставлять подробные ответы на широкий спектр вопросов.

Одним из ключевых особенностей WolframAlpha является его способность рассчитать производные, интегралы и другие сложные математические задачи. Это делает его полезным инструментом для студентов, учёных и инженеров, которым нужно решать сложные математические задачи.

Частные производные функции стоимости будут выглядеть следующим образом:

$$\frac{\partial cost(x,y)}{\partial x} = \sum_{i=1}^{n} 2\left(\frac{(x-x_0)}{\sqrt{\left((x-x_0)^2+(y-y_0)^2\right)}} - \frac{(x-x_i)}{\sqrt{\left((x-x_i)^2+(y-y_i)^2\right)}}\right) \times \\ \times\left(-d_i+\sqrt{\left((x-x_0)^2+(y-y_0)^2\right)}-\sqrt{\left((x-x_i)^2+(y-y_i)^2\right)}\right), \quad (7)$$

$$\frac{\partial cost(x,y)}{\partial y} = \sum_{i=1}^{n} 2\left(-d_i+\sqrt{\left((x-x_0)^2+(y-y_0)^2\right)}-\sqrt{\left((x-x_i)^2+(y-y_i)^2\right)}\right) \times \\ \times\left(\frac{(y-y_0)}{\sqrt{\left((x-x_0)^2+(y-y_0)^2\right)}} - \frac{(y-y_i)}{\sqrt{\left((x-x_i)^2+(y-y_i)^2\right)}}\right), \quad (8)$$

где x, y — координаты объекта, местоположение которого необходимо определить;

n — количество реперных точек;

d_i — разница во времени прихода сигналов от реперных точек, преобразованная в расстояние;

x_0, y_0 — координаты главной реперной точки;

x_i, y_i — координаты вторичных реперных точек.

В данном разделе подводятся итоги обсуждения математических основ алгоритма определения местоположения на плоскости по заданной разности сигналов. Рассмотрены принципы работы евклидова расстояния и методов оптимизации, включая метод Ньютона и градиентный спуск.

Описаны ключевые аспекты выбора и реализации метода оптимизации для решения системы уравнений, основанных на нелинейных функциях, таких как евклидово расстояние. Подчёркнута важность выбора подходящей скорости обучения и начального приближения, а также необходимость тщательной настройки параметров и проверки работы методов на практике.

Функция стоимости и градиент функции стоимости были подробно описаны. Указано, что функция стоимости измеряет общую разницу между измеренными и теоретическими расстояниями до каждой из реперных точек, и что градиент этой функции стоимости затем используется в алгоритме градиентного спуска для обновления текущего приближения местоположения объекта.

3 Разработка алгоритма определения местоположения

3.1 Реализация алгоритма

В этом разделе переходят к практической части исследования и начинают разработку алгоритма определения местоположения. Обсуждаются способы применения теоретических знаний, полученных в предыдущих главах, для создания работающего алгоритма. Это включает в себя выбор подходящих инструментов и технологий, а также реализацию и тестирование алгоритма на различных данных.

Выбор языка программирования Python обусловлен его простотой для реализации математических алгоритмов. Кроме того, для Python доступно множество математических библиотек, таких как SciPy, NumPy и другие, а так же библиотекой для визуализации Matplotlib что облегчает процесс разработки.

NumPy — это библиотека Python, используемая для работы с массивами. Она также имеет функции для работы в области линейной алгебры, преобразования Фурье и матриц. NumPy был создан в 2005 году Трэвисом Олифантом. Это проект с открытым исходным кодом, и его можно использовать бесплатно. NumPy предназначен для численных вычислений в Python.

SciPy — это библиотека Python для научных вычислений. SciPy использует NumPy в качестве основы и предоставляет дополнительные функции для оптимизации, статистики и обработки сигналов. SciPy является открытым исходным кодом, поэтому его можно использовать бесплатно. SciPy был создан создателем NumPy, Трэвисом Олифантом.

Matplotlib — это библиотека Python для создания статических, анимированных и интерактивных визуализаций. Matplotlib обеспечивает простоту создания графиков и возможность реализации сложных задач визуализации. Она широко используется для визуализации данных в научных и

инженерных приложениях. Matplotlib была создана Джоном Д. Хантером в 2003 году.

Реализация алгоритма включает в себя несколько ключевых шагов:

- подготовка данных: на этом этапе собираются и обрабатываются данные, которые будут использоваться для определения местоположения. Это может включать в себя измерение времени прихода сигналов от реперных точек, преобразование этих времён в расстояния;
- разработка алгоритма: затем эти данные используются для решения системы уравнений, которая представляет собой гиперболы. Для этого необходимо реализовать градиентный спуск. Также, необходимо добавление шума к данным для имитации реальных условий;
- анализ результатов: после получения решения проводится анализ результатов, чтобы оценить точность и эффективность алгоритма. Это может включать в себя сравнение полученного местоположения с истинным местоположением, вычисление ошибок и анализ влияния шума на точность.
- оптимизация алгоритма: на основе анализа результатов вносятся изменения в алгоритм, чтобы улучшить его точность и эффективность. Это может включать в себя изменение метода решения, увеличение количества реперных точек или улучшение обработки шума.

Перейдём к первому шагу. Сначала мы сгенерируем таблицу, например с 1000 строк, в каждой строке будут храниться координаты реперной точки, истинного положения искомой точки, и разницу в задержке сигналов, преобразованные в расстояния.

В начале процесса необходимо импортировать определенные библиотеки. Рисунок 5 включает в себя pandas для обработки данных, numpy для выполнения математических операций и random для генерации случайных чисел.

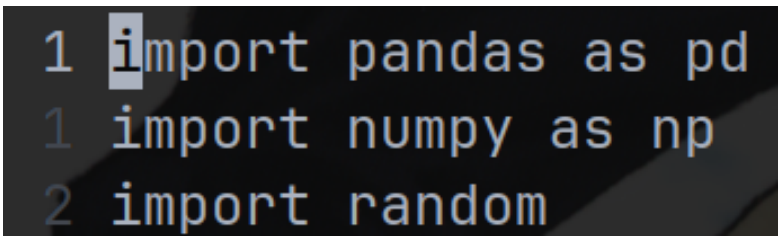


Рисунок 5 - Импорт библиотек

На рисунке 6 происходит установка ключевых параметров, инициализация пустых списков для хранения реперных точек и задержек сигналов, а также генерация случайных координат для реперных точек и точки на поле. Затем вычисляются задержки сигналов на основе расстояний от реперных точек до точки на поле и добавляются в соответствующие списки. Эти операции являются важной частью процесса обработки данных и подготовки их к дальнейшему анализу. Корректное выполнение этих операций обеспечивает точность и надёжность получаемых результатов.

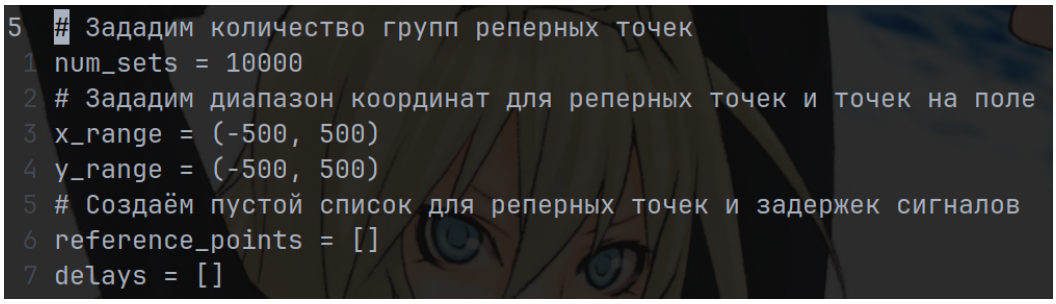


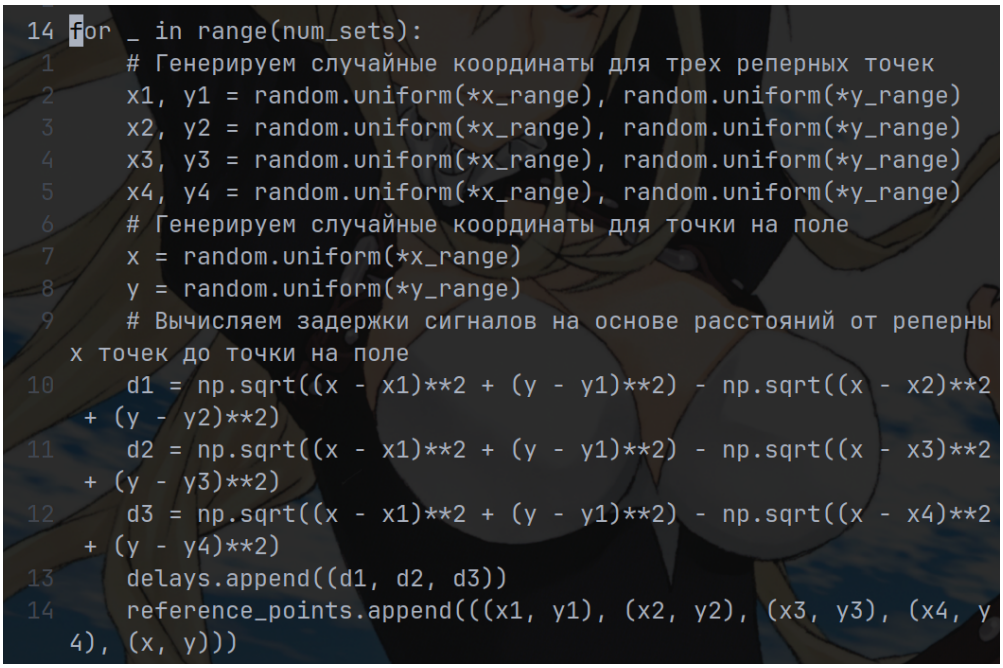
Рисунок 6 - Установка параметров, инициализация списков и генерация данных

Далее, на рисунке 7 происходит генерация случайных координат для реперных точек и точки на поле. Это важный шаг, поскольку эти координаты будут использоваться для вычисления расстояний и задержек сигналов.

Затем на основе этих координат вычисляются задержки сигналов. Это делается путём вычисления расстояний от реперных точек до точки на поле и преобразования этих расстояний во времена задержек сигналов. Эти времена задержек сигналов являются ключевой информацией, которая будет использоваться в дальнейшем для определения местоположения точки на поле.

После вычисления задержек сигналов, они добавляются в соответствующие списки. Эти списки будут использоваться в дальнейшем для анализа данных и вычисления результатов.

Все эти операции являются важной частью процесса обработки данных и подготовки их к дальнейшему анализу. Корректное выполнение этих операций обеспечивает точность и надёжность получаемых результатов.



```
for _ in range(num_sets):
    # Генерируем случайные координаты для трех реперных точек
    x1, y1 = random.uniform(*x_range), random.uniform(*y_range)
    x2, y2 = random.uniform(*x_range), random.uniform(*y_range)
    x3, y3 = random.uniform(*x_range), random.uniform(*y_range)
    x4, y4 = random.uniform(*x_range), random.uniform(*y_range)
    # Генерируем случайные координаты для точки на поле
    x = random.uniform(*x_range)
    y = random.uniform(*y_range)
    # Вычисляем задержки сигналов на основе расстояний от реперны
x точек до точки на поле
    d1 = np.sqrt((x - x1)**2 + (y - y1)**2) - np.sqrt((x - x2)**2
 + (y - y2)**2)
    d2 = np.sqrt((x - x1)**2 + (y - y1)**2) - np.sqrt((x - x3)**2
 + (y - y3)**2)
    d3 = np.sqrt((x - x1)**2 + (y - y1)**2) - np.sqrt((x - x4)**2
 + (y - y4)**2)
    delays.append((d1, d2, d3))
    reference_points.append(((x1, y1), (x2, y2), (x3, y3), (x4, y
4), (x, y)))
```

Рисунок 7 - Генерация данных

На рисунке 8 создаётся DataFrame из списка задержек, к которому добавляются координаты реперных точек и точки на поле. Полученный DataFrame затем сохраняется в CSV-файл.

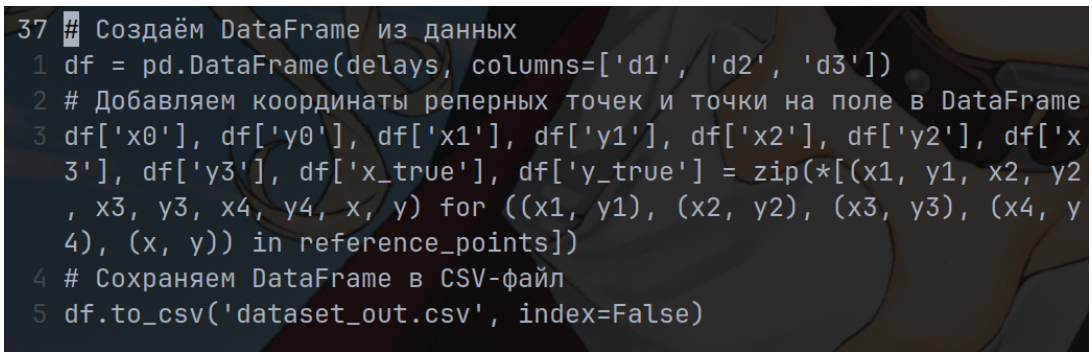


Рисунок 8 - Формирование и сохранение данных

Далее, перейдём к следующему шагу, к реализации программы для нахождения точки. Программа должна уметь прочитать созданную таблицу и вычислить среднеквадратичную ошибку (RMSE) между истинными местоположениями и местоположениями, определёнными алгоритмом. Также нужно визуализировать график остатков, чтобы понять, с какой частотой ошибается алгоритм.

Аналогично рисунку 5, в данном блоке кода (рисунок 9), происходит импорт необходимых библиотек. Однако в этом случае, к уже знакомым numpy, random и pandas, добавляется библиотека matplotlib.pyplot, которая используется для построения графиков. Эти библиотеки позволяют обрабатывать данные, генерировать случайные числа и создавать визуализации, что является ключевым для анализа данных и представления результатов.

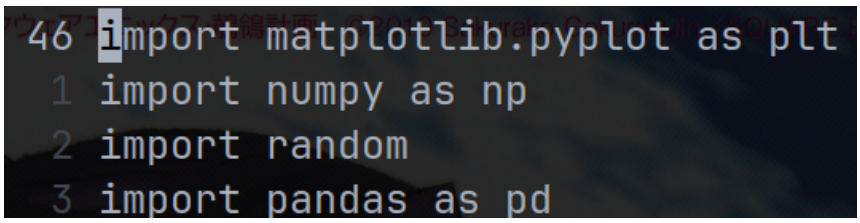


Рисунок 9 - Импорт библиотек для визуализации и обработки данных

Затем, на рисунке 10 происходит загрузка датасета из CSV-файла. Для этого используется функция pd.read_csv(). После загрузки датасета инициализируются пустые списки для хранения результатов, истинных

значений и решений. Эти списки будут использоваться в дальнейшем для анализа данных.

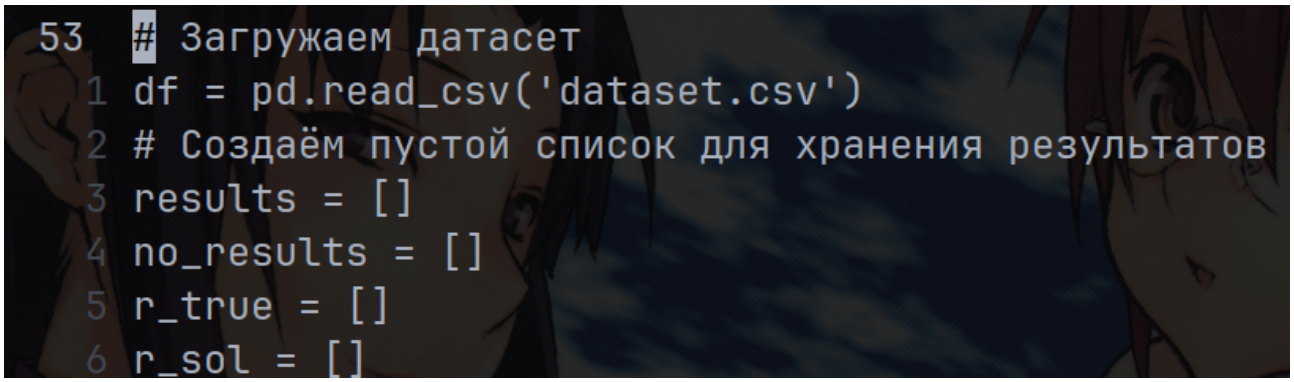


Рисунок 10 - Загрузка датасета и инициализация списков для хранения данных

В следующем блоке кода, на рисунке 11 определяются функции eq1(), eq2() и eq3(), которые вычисляют расстояния между точками. Эти функции являются ключевыми для определения геометрических отношений между точками в пространстве.

Также определяются функция стоимости cost(), которая вычисляет сумму квадратов расстояний, и функция gradient(), которая вычисляет градиент функции стоимости. Эти функции используются для оптимизации расположения точек и минимизации общего расстояния между ними.

Все эти функции играют важную роль в процессе обработки и анализа данных, позволяя извлечь полезную информацию из данных и использовать ее для получения точных и информативных результатов.

```
def eq1(x, y):
    return np.sqrt((x - x0)**2 + (y - y0)**2) - np.sqrt((x - x1)**2 + (y - y1)**2) - d1
def eq2(x, y):
    return np.sqrt((x - x0)**2 + (y - y0)**2) - np.sqrt((x - x2)**2 + (y - y2)**2) - d2
def eq3(x, y):
    return np.sqrt((x - x0)**2 + (y - y0)**2) - np.sqrt((x - x3)**2 + (y - y3)**2) - d3

# Функция стоимости
def cost(params):
    x, y = params
    eq1_val = eq1(x, y)
    eq2_val = eq2(x, y)
    eq3_val = eq3(x, y)
    return eq1_val**2 + eq2_val**2 + eq3_val**2

# Градиент функции стоимости
def gradient(params):
    x, y = params
    dx = 2 * ((-(-x1 + x)/np.sqrt(np.power((-x1 + x),2) + np.power((-y1 + y),2)) + (-x0 + x)/np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) * (-d1 - np.sqrt(np.power((-x1 + x),2) + np.power((-y1 + y),2)) + np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) + (-(-x3 + x)/np.sqrt(np.power((-x3 + x),2) + np.power((-y3 + y),2)) + (-x0 + x)/np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) * (-d3 - np.sqrt(np.power((-x3 + x),2) + np.power((-y3 + y),2)) + np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) + ((-x0 + x)/ np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2)) - (-x2 + x)/ np.sqrt(np.power((-x2 + x),2) + np.power((-y2 + y),2))) * (-d2 + np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2)) - np.sqrt(np.power((-x2 + x),2) + np.power((-y2 + y),2))));
    dy = 2 * ((-(-y1 + y)/np.sqrt(np.power((-x1 + x),2) + np.power((-y1 + y),2)) + (-y0 + y)/np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) * (-d1 - np.sqrt(np.power((-x1 + x),2) + np.power((-y1 + y),2)) + np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) + (-(-y3 + y)/np.sqrt(np.power((-x3 + x),2) + np.power((-y3 + y),2)) + (-y0 + y)/np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) * (-d3 - np.sqrt(np.power((-x3 + x),2) + np.power((-y3 + y),2)) + np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2))) + ((-y0 + y)/ np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2)) - (-y2 + y)/ np.sqrt(np.power((-x2 + x),2) + np.power((-y2 + y),2))) * (-d2 + np.sqrt(np.power((-x0 + x),2) + np.power((-y0 + y),2)) - np.sqrt(np.power((-x2 + x),2) + np.power((-y2 + y),2))));

    return np.array([dx, dy])
```

Рисунок 11 - Определение функций для вычисления расстояний, функции стоимости и градиента функции стоимости

Функция gradient_descent() (рисунок 12), является ключевым компонентом в оптимизации процесса определения местоположения. Она принимает на вход функцию градиента, начальное приближение, скорость обучения, максимальное количество итераций и допустимую погрешность.

В начале функции задаётся вектор, который инициализируется начальным приближением. Затем начинается цикл, который выполняется заданное максимальное количество итераций.

На каждой итерации вычисляется шаг градиентного спуска, который равен произведению скорости обучения и градиента в текущей точке. Знак шага инвертируется, так как мы движемся в направлении наискорейшего убывания функции.

Если все компоненты шага меньше заданной допустимой погрешности, то цикл прерывается. Это означает, что мы достигли минимума с заданной точностью.

В противном случае, к текущему вектору прибавляется шаг, и процесс продолжается.

В конце функции возвращается полученный вектор, который является результатом оптимизации. Этот вектор представляет собой точку минимума функции стоимости, которая может представлять собой расчетное местоположение объекта.

Таким образом, функция gradient_descent() позволяет найти оптимальное местоположение объекта, минимизируя разницу между измеренной и расчётной задержкой сигнала от реперных точек. Это ключевой этап в алгоритме определения местоположения.

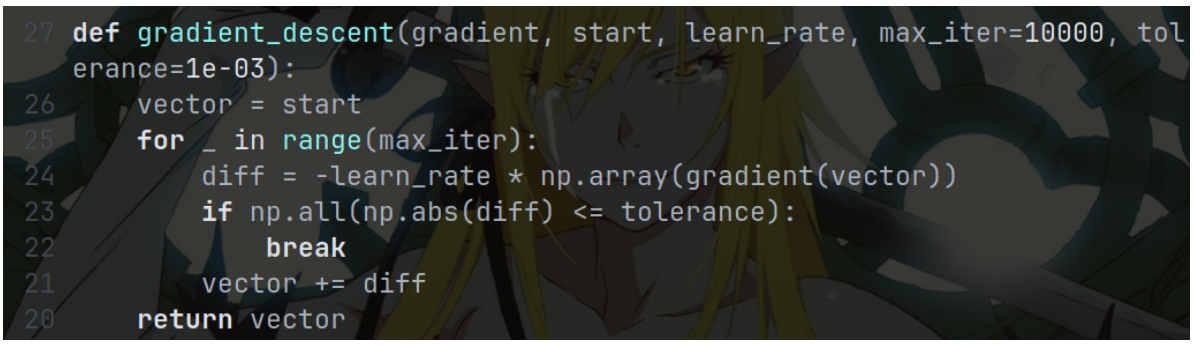


Рисунок 12 - Определение функции градиентного спуска

После, на рисунке 13 происходит обработка каждой строки в датасете для определения местоположения объекта. Для каждой строки извлекаются координаты четырех реперных точек и задержки сигнала от каждой из них. Также извлекаются истинные координаты объекта.

Затем к задержкам сигнала добавляется шум с помощью функции add_noise(). Это делается для имитации реальных условий, где данные могут быть зашумлены из-за различных факторов.

После этого инициализируются переменные best_result и best_cost, которые будут хранить лучший результат и соответствующую ему стоимость. Также определяются границы поиска для координат x и y.

Затем выполняется 10 попыток оптимизации с разными начальными приближениями. На каждой попытке выполняется градиентный спуск и вычисляется значение функции стоимости. Если текущее значение функции стоимости меньше лучшего, то обновляются best_result и best_cost.

Если оптимизация удаётся, то результаты сохраняются в списки results, r_true и r_sol. В противном случае индекс строки добавляется в список no_results.

Таким образом, в этом блоке кода выполняется оптимизация для каждой строки данных, и сохраняются результаты. Это важный этап в процессе определения местоположения объекта на основе задержки сигнала от реперных точек.

```
# Проходим по каждой строке в датасете
for i, row in df.iterrows():
    # Извлекаем данные
    x0, y0 = row['x0'], row['y0']
    x1, y1 = row['x1'], row['y1']
    x2, y2 = row['x2'], row['y2']
    x3, y3 = row['x3'], row['y3']
    d1, d2, d3 = row['d1'], row['d2'], row['d3']
    x_true, y_true = row['x_true'], row['y_true']
    d1, d2, d3 = add_noise([d1,d2,d3], noise)

    best_result = None
    best_cost = float('inf')
    x_min, x_max = min(x1, x2, x3, x0) - 5, max(x1, x2, x3, x0) + 5
    y_min, y_max = min(y1, y2, y3, y0) - 5, max(y1, y2, y3, y0) + 5
    for _ in range(10):
        initial_guess = (random.uniform(x_min, x_max), random.uniform(
y_min, y_max))
        result = gradient_descent(gradient, initial_guess, learn_rate=
0.1)
        cost_value = cost(result)

    if cost_value < best_cost:
        best_result = result
        best_cost = cost_value

    if best_result is not None:
        x, y = best_result.x
        results.append((x, y, x_true, y_true))
        r_true.append((x_true, y_true))
        r_sol.append((x, y))
    else:
        no_results.append(i)
```

Рисунок 13 - Итерация по данным: оптимизация и сохранение результатов определения местоположения

На рисунке 14 происходит вычисление среднеквадратичной ошибки (RMSE) [14], остатков и подсчёт больших остатков определении местоположения объекта на основе задержки сигнала от реперных точек.

Сначала вычисляется RMSE для предсказаний местоположения объекта. Затем вычисляются остатки для каждой координаты. Наконец, находятся остатки, которые больше 10, и подсчитывается их количество. Это позволяет оценить, насколько велики ошибки в предсказаниях.

График остатков - это важный инструмент в статистическом анализе и машинном обучении, который помогает оценить качество модели. Остатки представляют собой разницу между наблюдаемыми (истинными) и предсказанными значениями [13].

В определении местоположения объекта, остатки - это разница между истинными координатами объекта и координатами, предсказанными моделью.

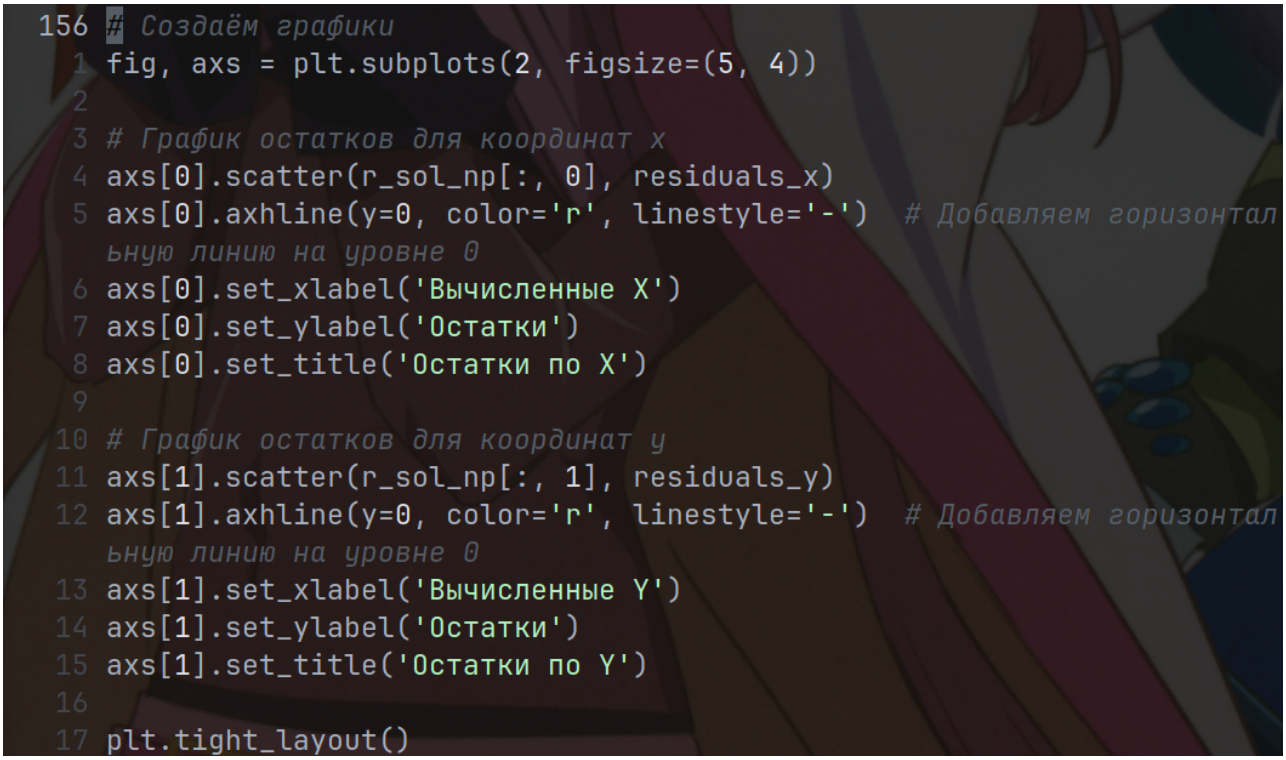
График остатков для координат X и Y показывает, насколько точно модель предсказывает каждую из координат. На горизонтальной оси графика откладываются вычисленные координаты, а на вертикальной - соответствующие остатки.

Если модель работает идеально, все остатки должны быть равны нулю, и все точки должны лежать на горизонтальной линии на уровне нуля. Если же точки расположены хаотично и далеко от нулевой линии, это может указывать на проблемы в модели.

```
134 # Вычисляем RMSE
  1 errors = [(x - x_true)**2 + (y - y_true)**2 for x, y, x_true, y_true
    in results]
  2 rmse = np.sqrt(sum(errors) / len(errors))
  3 print("RMSE = ", rmse)
  4 r_true_np = np.array(r_true)
  5 r_sol_np = np.array(r_sol)
  6 # Вычисляем остатки для каждой координаты
  7 residuals_x = r_true_np[:, 0] - r_sol_np[:, 0]
  8 residuals_y = r_true_np[:, 1] - r_sol_np[:, 1]
  9 # Поиск остатков, которые больше единицы
 10 large_residuals_x = residuals_x[abs(residuals_x) > 10]
 11 large_residuals_y = residuals_y[abs(residuals_y) > 10]
 12
 13 # Вычисляем сумму этих остатков
 14 count_large_residuals_x = len(large_residuals_x)
 15 count_large_residuals_y = len(large_residuals_y)
 16
 17 print("Количество остатков по X, которые больше 10:", count_large_res
    iduals_x)
 18 print("Количество остатков по Y, которые больше 10:", count_large_res
    iduals_y)
```

Рисунок 14 - Вычисление среднеквадратичной ошибки, остатков и подсчет больших остатков

Наконец, на рисунке 15 создаются графики остатков для координат X, и Y. Остатки представляют собой разницу между истинными и вычисленными координатами объекта. Визуализация остатков помогает оценить, насколько точно модель определяет местоположение объекта.



```
# Создаём графики
fig, axs = plt.subplots(2, figsize=(5, 4))

# График остатков для координат x
axs[0].scatter(r_sol_np[:, 0], residuals_x)
axs[0].axhline(y=0, color='r', linestyle='-')  # Добавляем горизонтальную линию на уровне 0
axs[0].set_xlabel('Вычисленные X')
axs[0].set_ylabel('Остатки')
axs[0].set_title('Остатки по X')

# График остатков для координат y
axs[1].scatter(r_sol_np[:, 1], residuals_y)
axs[1].axhline(y=0, color='r', linestyle='-')  # Добавляем горизонтальную линию на уровне 0
axs[1].set_xlabel('Вычисленные Y')
axs[1].set_ylabel('Остатки')
axs[1].set_title('Остатки по Y')

plt.tight_layout()
```

Рисунок 15 - Создание графиков остатков

3.2 Тестирование и оценка алгоритма

В данном алгоритме используются четыре точки: одна мастер-точка и три вторичные точки. Сначала определим, сколько реперных точек необходимо использовать.

Если использовать только две реперные точки, то получим только одну гиперболу. В то время как три реперные точки дают две гиперболы, и их пересечение определяет местоположение искомой точки. Однако предварительное тестирование показало, что при использовании только трех станций могут быть случаи, когда две гиперболы пересекаются в двух точках.

Обычно это происходит, когда приёмник находится вне ограниченного диапазона.

На рисунке 16 представлена визуализация решения задачи определения местоположения с использованием метода гиперболической навигации. На графике изображены три реперные точки и искомая точка, а также две гиперболы, образованные на основе задержек сигналов от реперных точек. Пересечение этих гипербол определяет местоположение искомой точки. Но на данном рисунке, два пересечения, отчего можно сделать вывод, что точка может находиться в двух разных местах.

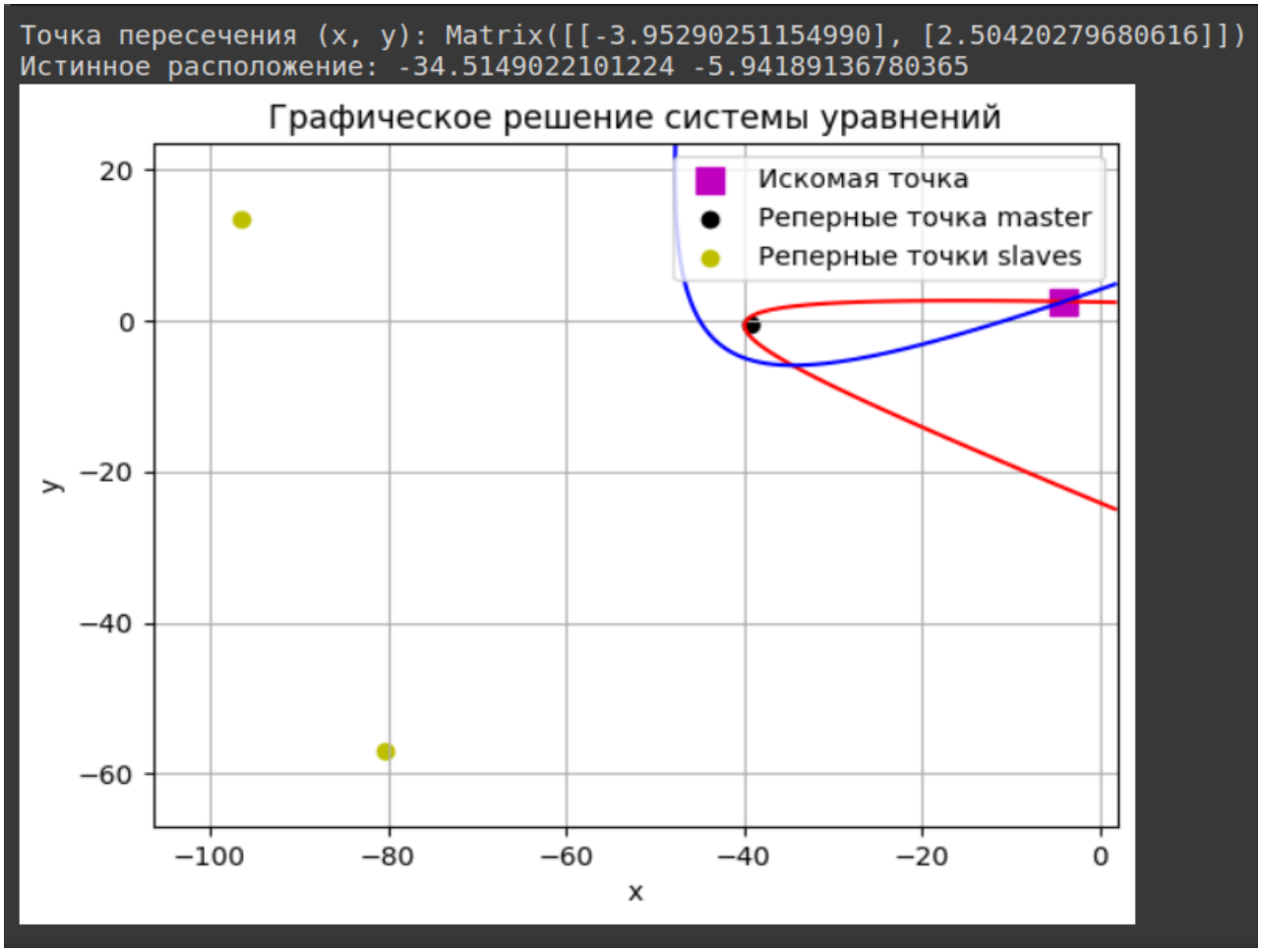


Рисунок 16 - Визуализация решения задачи определения с использованием метода гиперболической навигации

С другой, стороны, тестирование показало, что добавление четвертой реперной точки может улучшить точность определения местоположения

объекта. Даже если объект находится вне ограниченного диапазона, его местоположение можно определить на пересечении трёх гипербол.

На Рисунке 17 представлена визуализация решения задачи определения местоположения с использованием четырех реперных точек. На графике изображены четыре реперные точки и искомая точка, а также три гиперболы, образованные на основе задержек сигналов от реперных точек. Пересечение этих гипербол определяет местоположение искомой точки. Это демонстрирует, как добавление четвертой реперной точки может улучшить точность определения местоположения объекта.

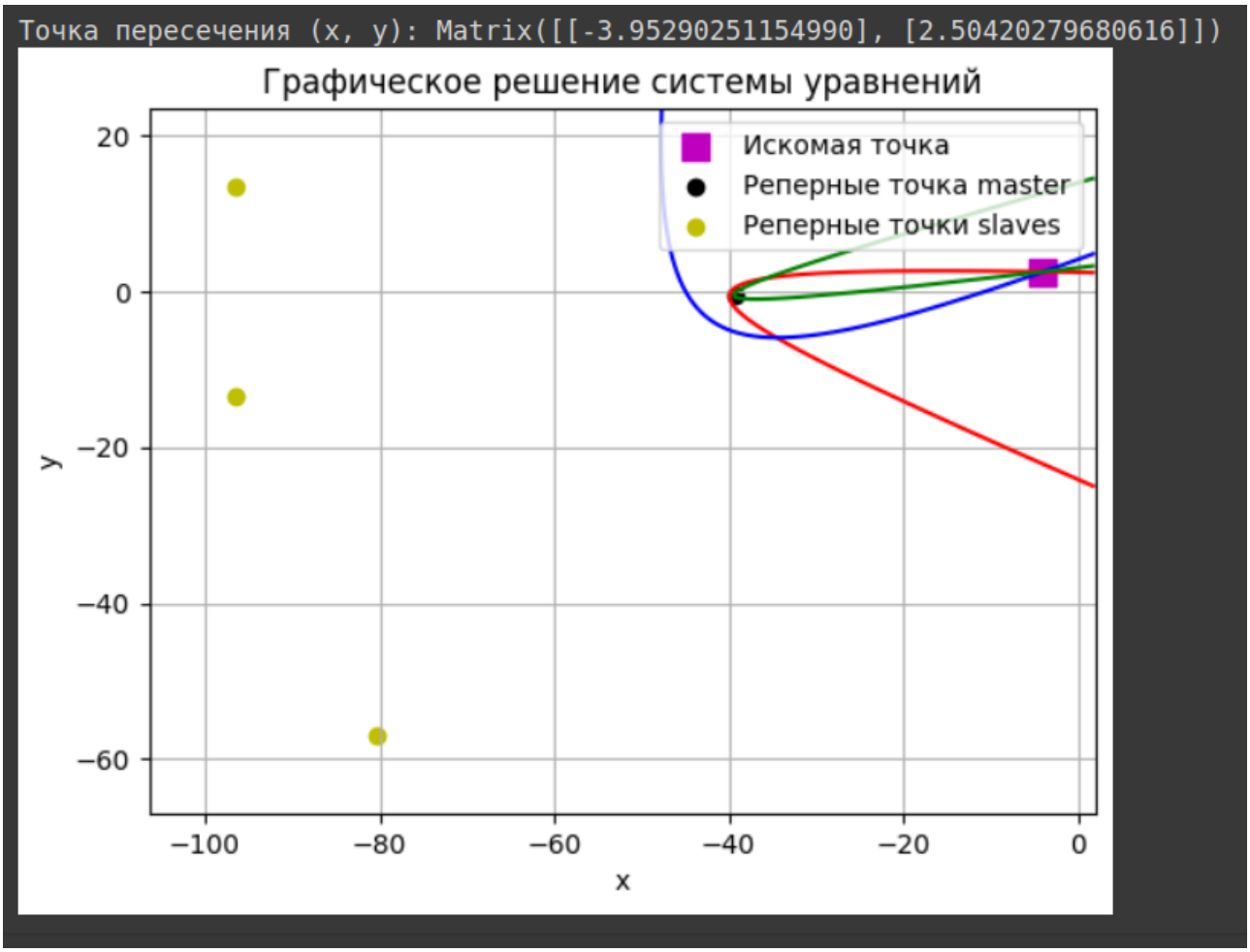


Рисунок 17 - Визуализация решения задачи определения местоположения с использованием четырех реперных точек

Также был проведён тест поведения модели для множества точек заданных по траектории. Тест показал стабильную работу без каких-либо

значительных отклонений. На рисунке 18 видно один из результатов тестирования.

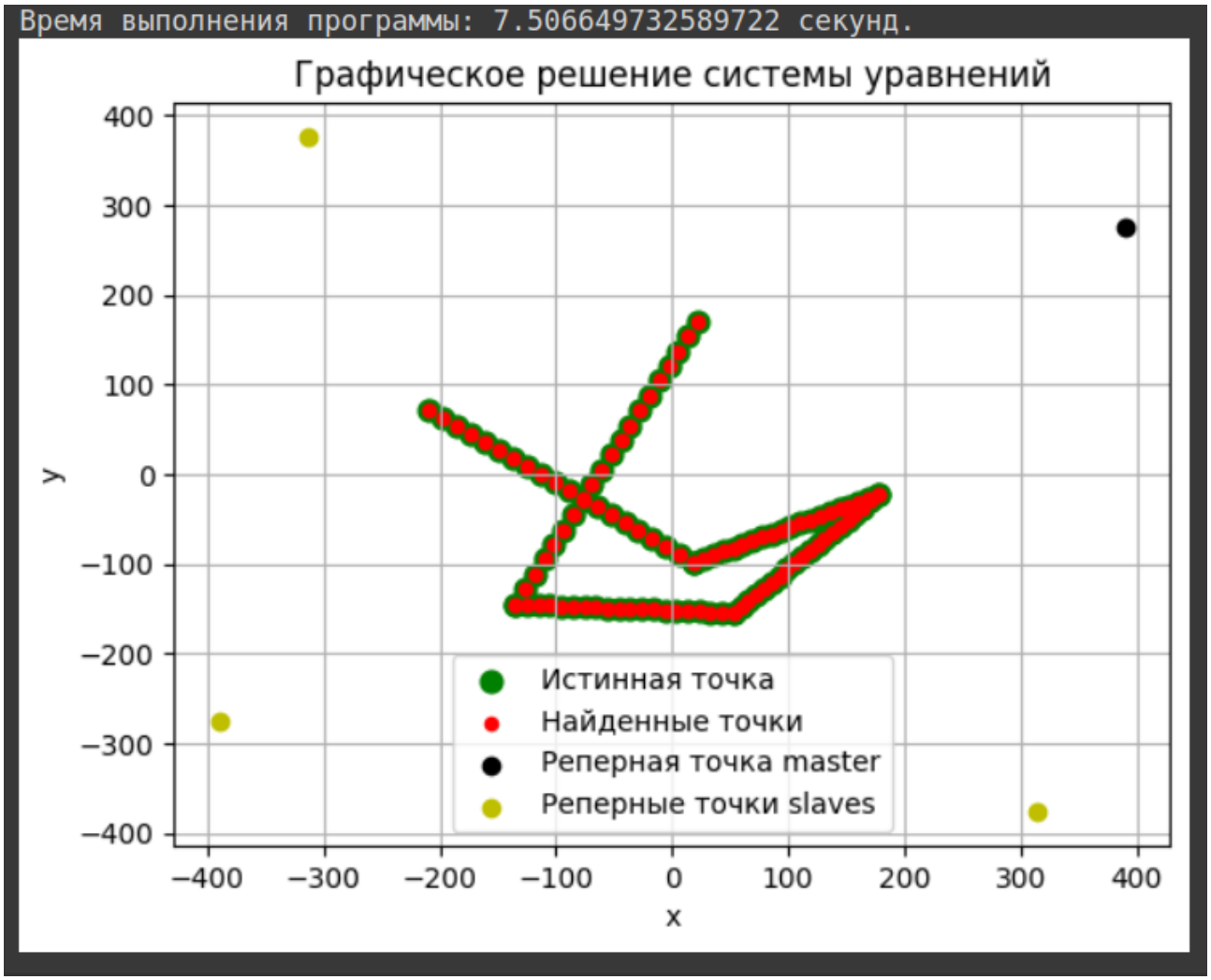


Рисунок 18 - Визуализация тестирования на множестве точек заданных по траектории

Было проведено самое важное тестирование алгоритма на 1000 данных, сгенерированных случайным образом. На рисунке 19, по полученным данным видно, что процент ошибок равен 4.6%. Это достаточно низкий уровень ошибок, что говорит о высокой точности алгоритма. Однако значение RMSE (среднеквадратичной ошибки) получилось достаточно большим. Это может быть обусловлено неадекватным расположением точек.

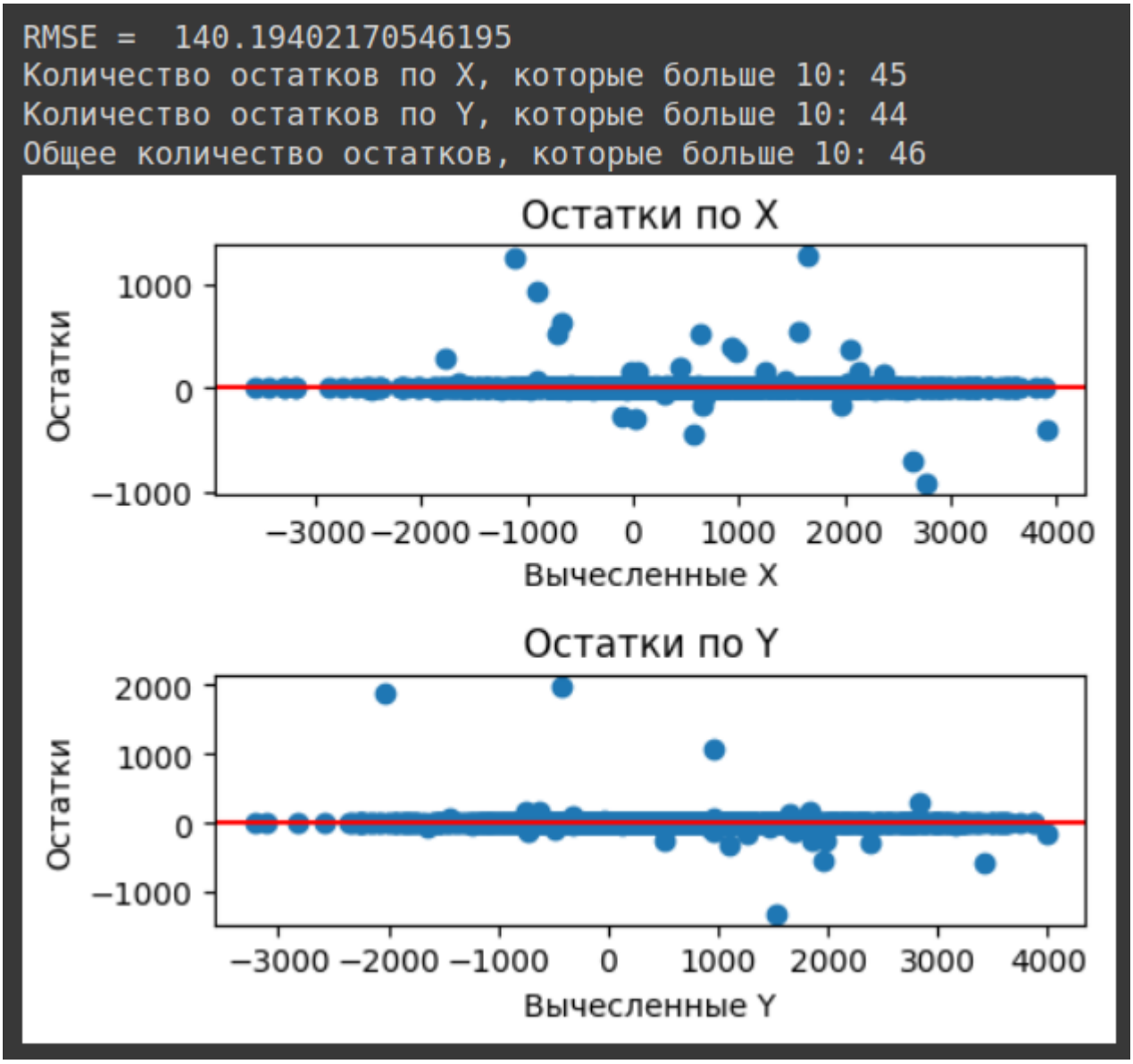


Рисунок 19 - Графическое представление результатов тестирования алгоритма определения местоположения

3.3 Оценка точности алгоритма с помощью введённого шума

Одним из важных аспектов при оценке алгоритма определения местоположения является его устойчивость к шуму. В реальном мире сигналы, которые мы получаем от реперных точек, могут быть подвержены различным видам помех, таким как многолучевое распространение, затухание сигнала, шум от электронных устройств и т.д. Эти помехи могут искажать сигналы и вносить ошибки в измерения времени прихода, что в свою очередь может снижать точность определения местоположения.

Для оценки устойчивости алгоритма к шуму, можно ввести искусственный шум в наши данные и оценить, как это влияет на результаты. Это может быть сделано путём добавления случайного шума к измеренным временам прихода сигналов или к координатам реперных точек.

Была реализованна простая функция добавления шума (рисунок 20)

```
177 def add_noise(data, max_noise_strenght=100):
  1     index = random.randint(0, len(data) - 1)
  2     noise_level = max_noise_strenght / (np.log2(np.abs(data[index]) +
     1))
  3     data[index] = (data[index] + np.random.uniform(-noise_level, nois
    e_level))
  4     return data
```

Рисунок 20 - Функция добавления шума

На рисунке 21, можно заметить, как шум искажает результат до 187%.

Таким образом, оценка устойчивости алгоритма к шуму является важной частью процесса тестирования. Это помогает понять, насколько надёжно алгоритм может работать в условиях реального мира, где данные часто содержат шум и другие втиды помех.

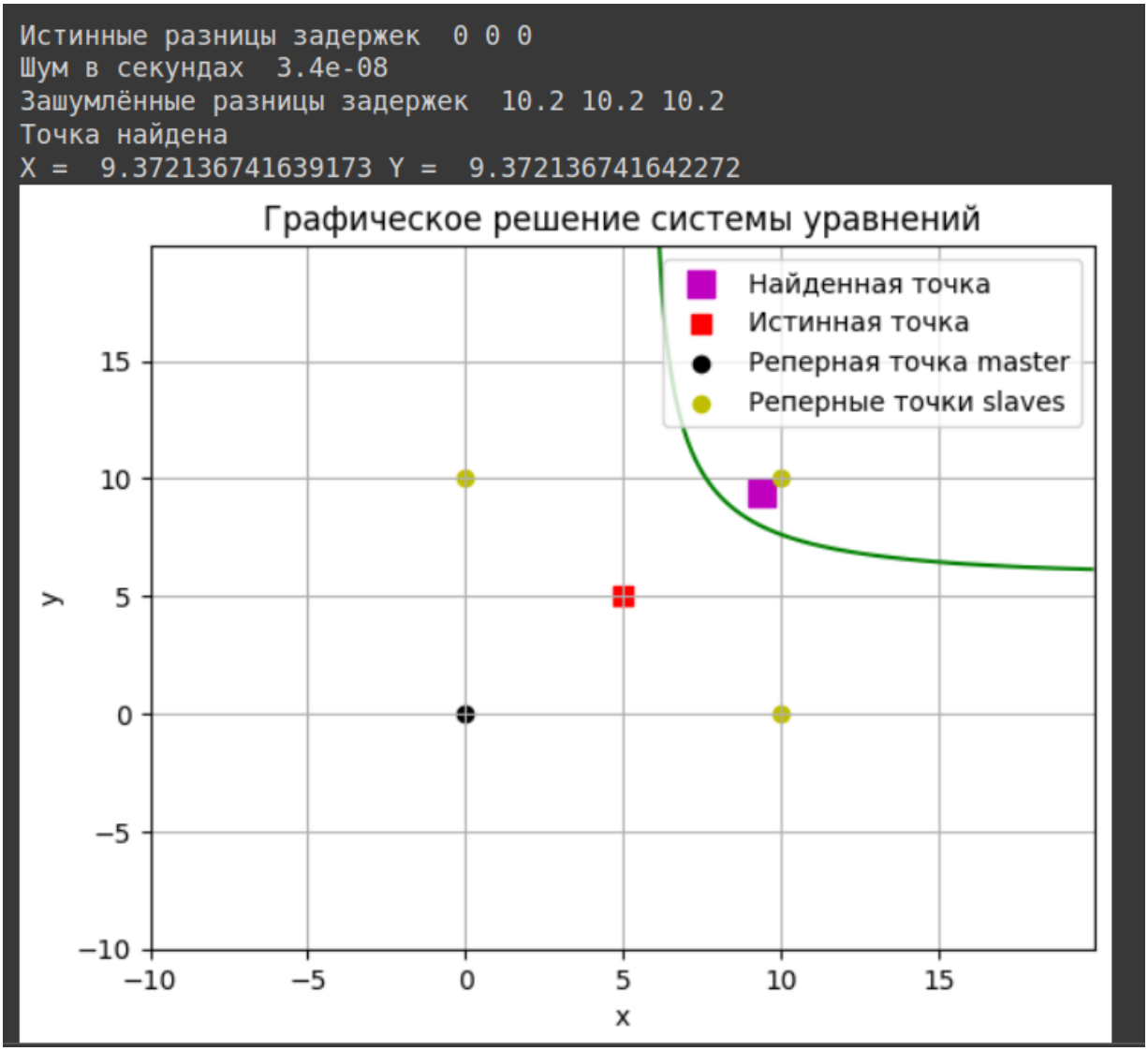


Рисунок 21 - Визуализация влияния шума на определение местоположения

На рисунке 22 представлена визуализация решения задачи определения местоположения с учётом введенного шума, с тем же набором данных из 1000 строк. График демонстрирует, как шум может влиять на точность определения местоположения объекта.

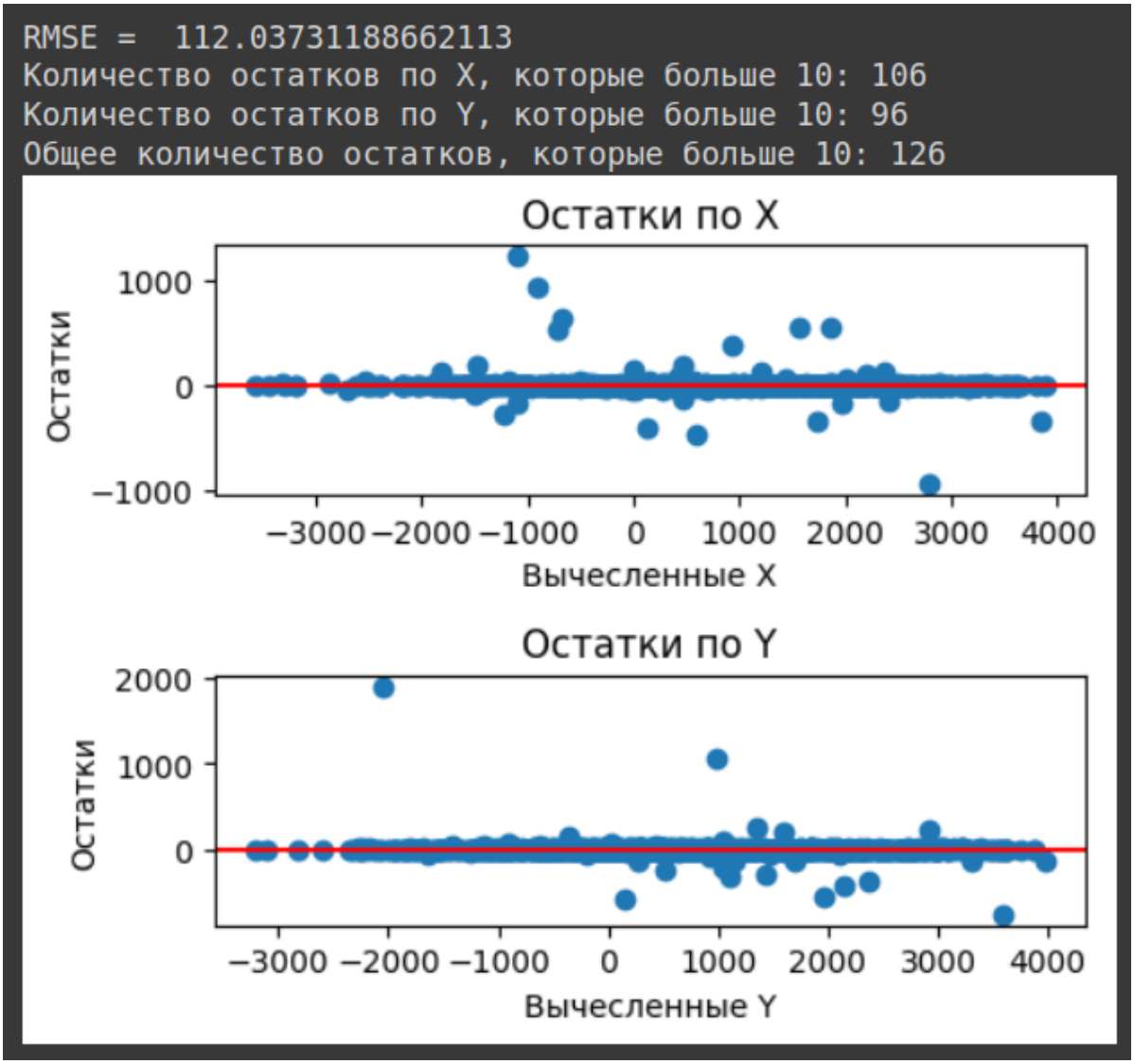


Рисунок 22 - Влияние шума на точность определения местоположения

Анализ результатов показывает, что введение шума серьезно влияет на точность алгоритма. Несмотря на то, что значение среднеквадратичной ошибки даже уменьшилось, количество остатков, которые больше 10, выросло до 12.6 процентов. То есть вероятность ошибочных результатов увеличилось в 3 раза, что указывает на значительное снижение точности алгоритма в условиях шума.

Таким образом, можно сделать вывод, что данный метод оптимизации для алгоритма подвержен шуму и требует дальнейшей оптимизации для повышения его устойчивости к шуму.

3.4 Оптимизация алгоритма

Чтобы увеличить устойчивость алгоритма к шуму, есть несколько способов:

- увеличение числа итераций: увеличение числа итераций в алгоритме оптимизации может помочь алгоритму лучше справиться с шумом, поскольку он получает больше возможностей для корректировки своих предсказаний;
- усреднение результатов: можно выполнить оптимизацию несколько раз с разными начальными точками и затем усреднить результаты, что позволит алгоритму выбрать более оптимальное решение;
- регуляризация: регуляризация добавляет штраф к функции стоимости за сложность модели. Это может помочь предотвратить переобучение на шумных данных [21];
- увеличение числа градиентных спусков: можно задействовать ещё два градиентных спуска для 1-2, 2-3 и 1-3 уравнений, после найти медиану.

В первом способе было увеличено число итераций с 1000 до 2000, что увеличивает время поиска точки, по рисунку 23 видно, что шум уменьшился на уровне погрешности.

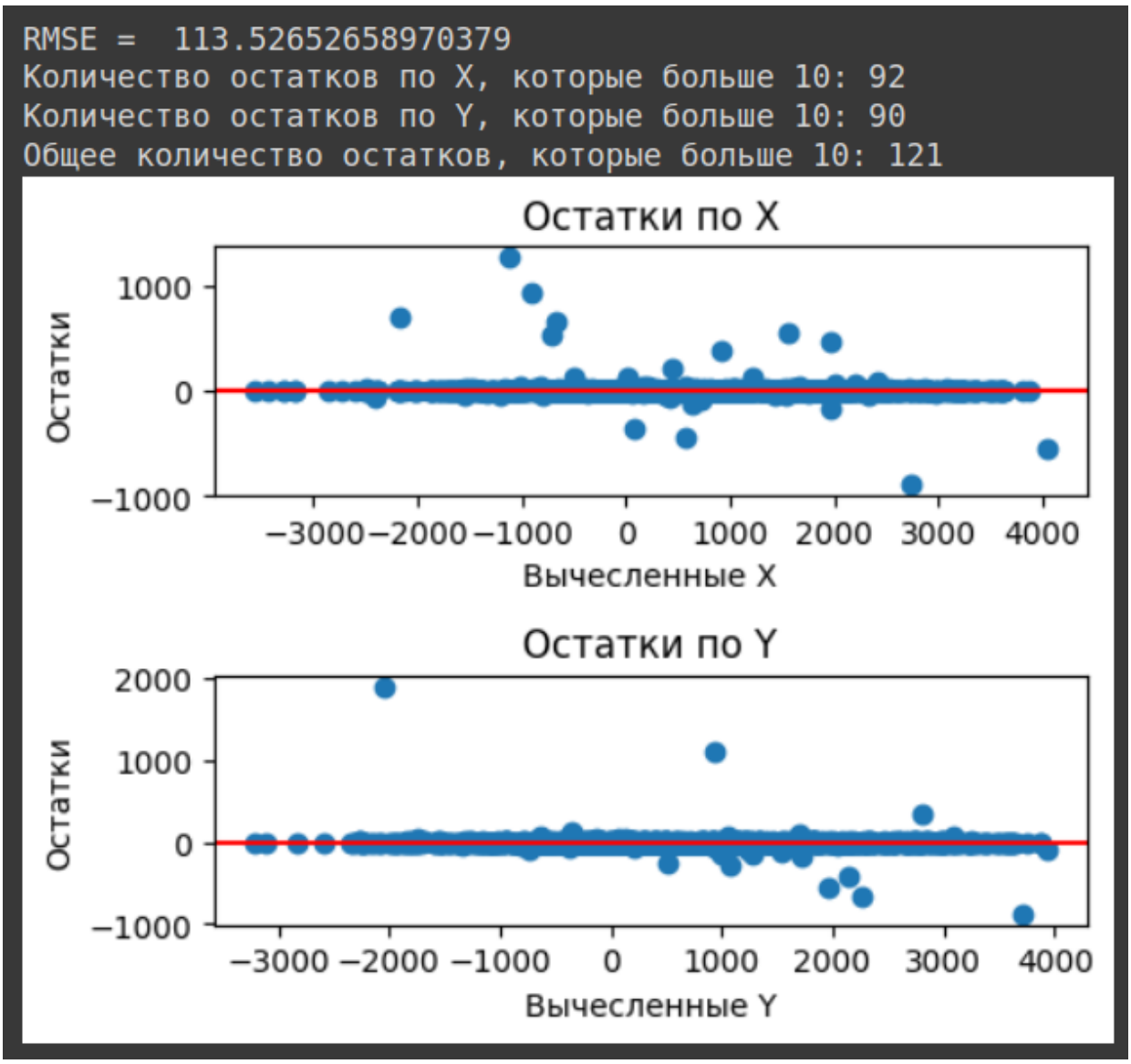


Рисунок 23 - Попытка увеличения устойчивости шума путём увеличения количества итераций

Во втором способе было увеличение числа начальных точек для поиска наилучшей точки. Число итераций было увеличено с 10, до 20. По рисунку 24 видно, что алгоритм сработал даже хуже, чем раньше, хоть и разница тоже в районе погрешности.

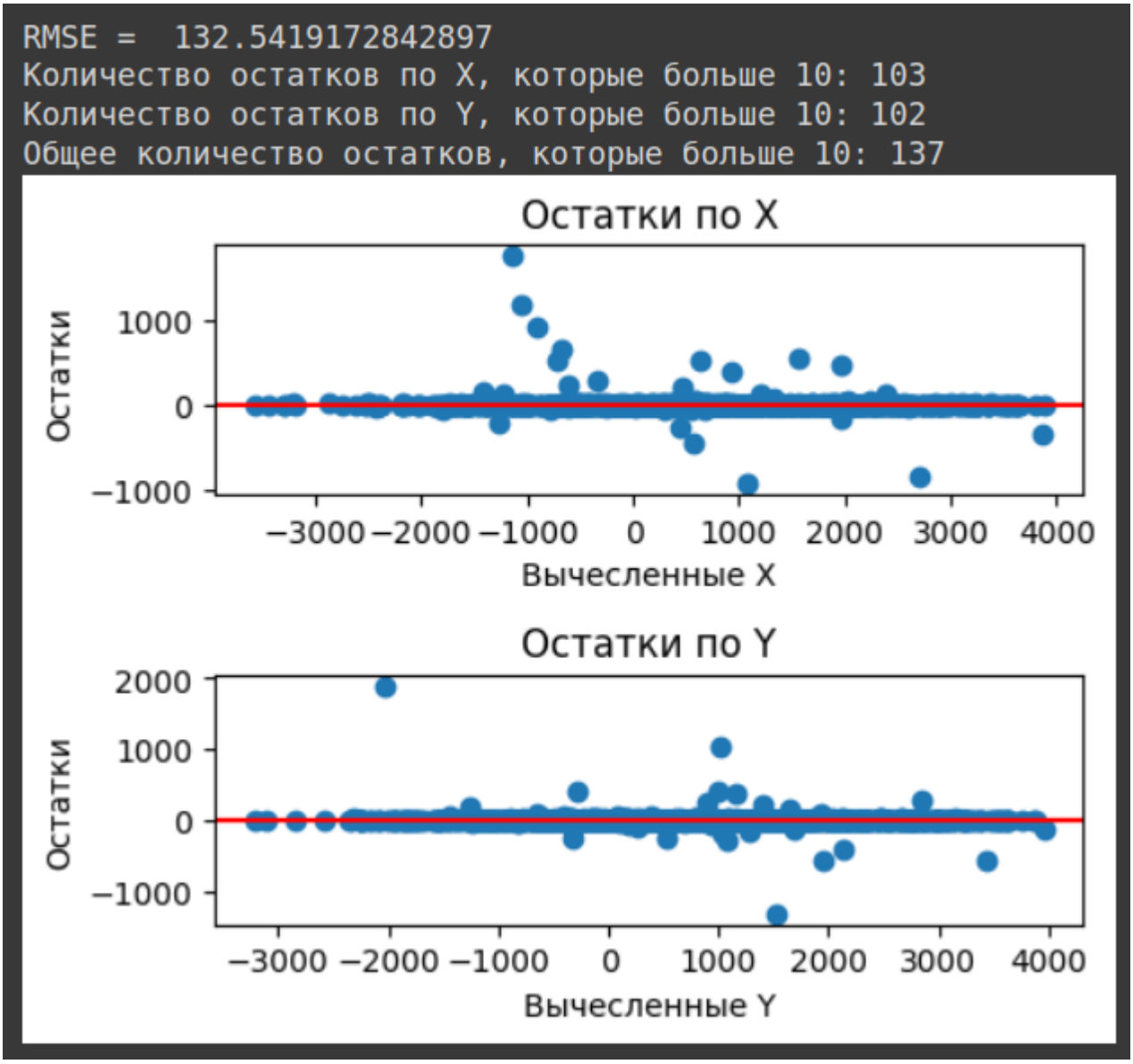


Рисунок 24 - Попытка увеличения устойчивости шума путём усреднения результатов

В третьем способе был применен метод регуляризации Тихонова [18], в градиентном спуске, регуляризация добавляется к функции стоимости следующим образом:

$$cost(x,y)=\sum_{i=1}^{n}\left(\sqrt{(x-x_0)^2+(y-y_0)^2}-\sqrt{(x-x_0)^2+(y-y_i)^2}-d_i\right)^2+\lambda\left(x^2+y^2\right) \quad (9)$$

где λ — это параметр регуляризации, который контролирует степень штрафа.

На рисунке 25 видно, что регуляризация уже существеннее ухудшает ситуацию, так что этот способ тоже не походит.

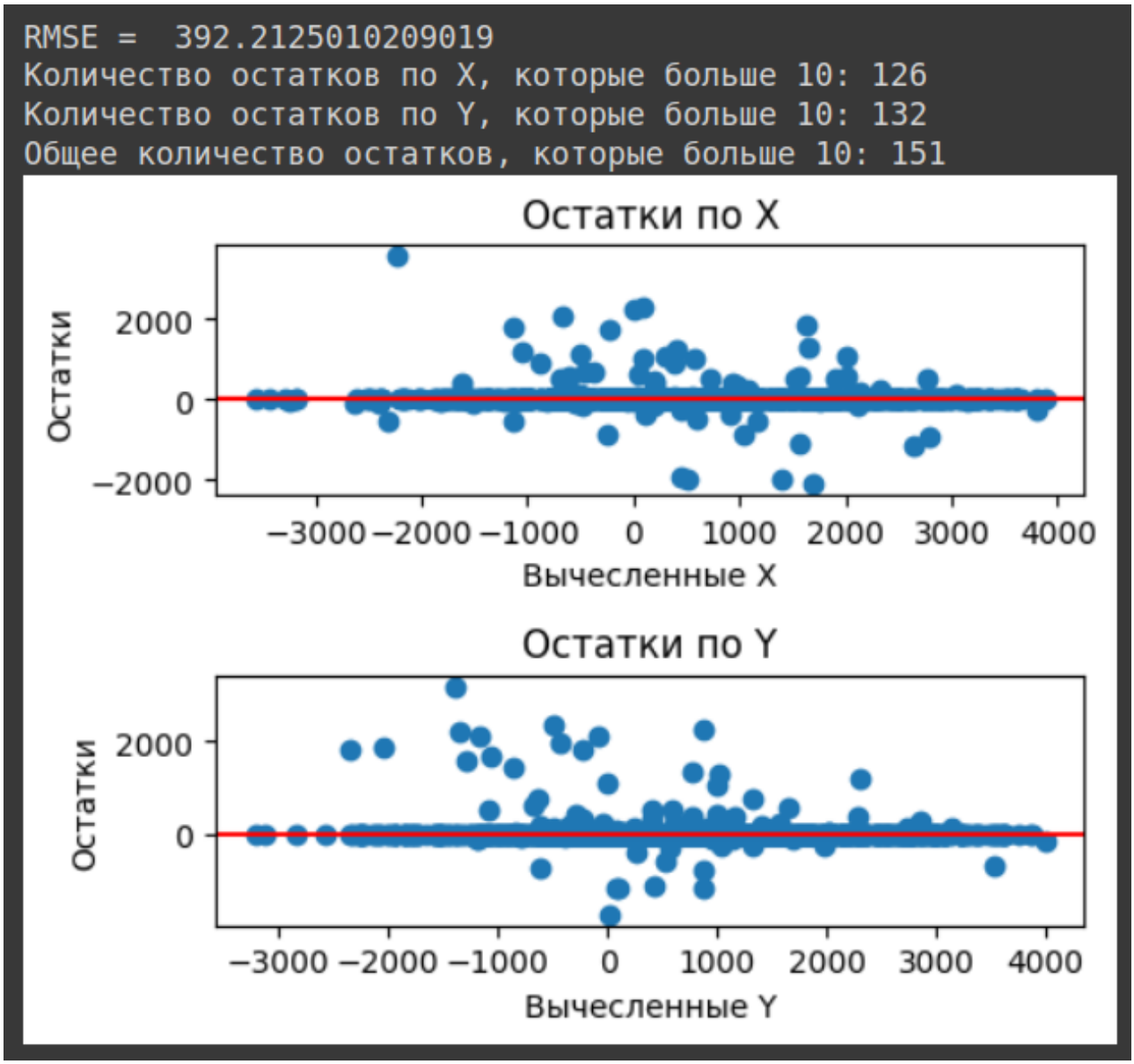


Рисунок 25 - Попытка увеличения устойчивости шума путём введения регуляризации

Осталась надежда на последний способ. Были добавлены ещё 2 функции стоимости и градиента стоимости, их можно посмотреть на рисунке 26. Градиенты стоимости были также решены с помощью WorframAlpha.

```
def cost_12(params):
    x, y = params
    eq1_val = eq1(x, y)
    eq2_val = eq2(x, y)
    return eq1_val**2 + eq2_val**2

def cost_23(params):
    x, y = params
    eq2_val = eq2(x, y)
    eq3_val = eq3(x, y)
    return eq2_val**2 + eq3_val**2

def cost_13(params):
    x, y = params
    eq1_val = eq1(x, y)
    eq3_val = eq3(x, y)
    return eq1_val**2 + eq3_val**2

def gradient_12(params):
    x, y = params
    dx = 2 * (x - x0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2) - (x - x1)/np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) * (-d1 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) + 2 * ((x - x0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (x - x2)/np.sqrt(np.power((x - x2),2) + np.power((y - y2),2)))*(-d2 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x2),2) + np.power((y - y2),2)))
    dy = 2 * (-d1 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x1),2) + np.power((y - y1),2)))*((y - y0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (y - y1)/np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) + 2 * (-d2 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x2),2) + np.power((y - y2),2))) * ((y - y0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (y - y2)/np.sqrt(np.power((x - x2),2) + np.power((y - y2),2)))
    return np.array([dx, dy])
Продолжение листинга 13.
def gradient_23(params):
    x, y = params
    dx = 2* ((x - x0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (x - x2)/np.sqrt(np.power((x - x2),2) + np.power((y - y2),2)))*(-d2 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x2),2) + np.power((y - y2),2))) + 2*((x - x0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (x - x3)/np.sqrt(np.power((x - x3),2) + np.power((y - y3),2)))*(-d3 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x3),2) + np.power((y - y3),2)))
    dy = 2* (-d2 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x2),2) + np.power((y - y2),2)))*((y - y0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (y - y2)/np.sqrt(np.power((x - x2),2) + np.power((y - y2),2))) + 2*(-d3 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x3),2) + np.power((y - y3),2)))*((y - y0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (y - y3)/np.sqrt(np.power((x - x3),2) + np.power((y - y3),2)))
    return np.array([dx, dy])
def gradient_13(params):
    x, y = params
    dx = 2*((x - x0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (x - x1)/np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) * (-d1 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) + 2*((x - x0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (x - x3)/np.sqrt(np.power((x - x3),2) + np.power((y - y3),2))) * (-d3 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x3),2) + np.power((y - y3),2)))
    dy = 2*(-d1 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) * ((y - y0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (y - y1)/np.sqrt(np.power((x - x1),2) + np.power((y - y1),2))) + 2*(-d3 + np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - np.sqrt(np.power((x - x3),2) + np.power((y - y3),2))) * ((y - y0)/np.sqrt(np.power((x - x0),2) + np.power((y - y0),2)) - (y - y3)/np.sqrt(np.power((x - x3),2) + np.power((y - y3),2)))
    return np.array([dx, dy])
```

Рисунок 26 - Добавленные функций для подсчёта медианы.

На рисунке 27 видно, что этот способ тоже ухудшает работу алгоритма, плюс, существенно ухудшается производительность, оптимизация происходит четыре раза.

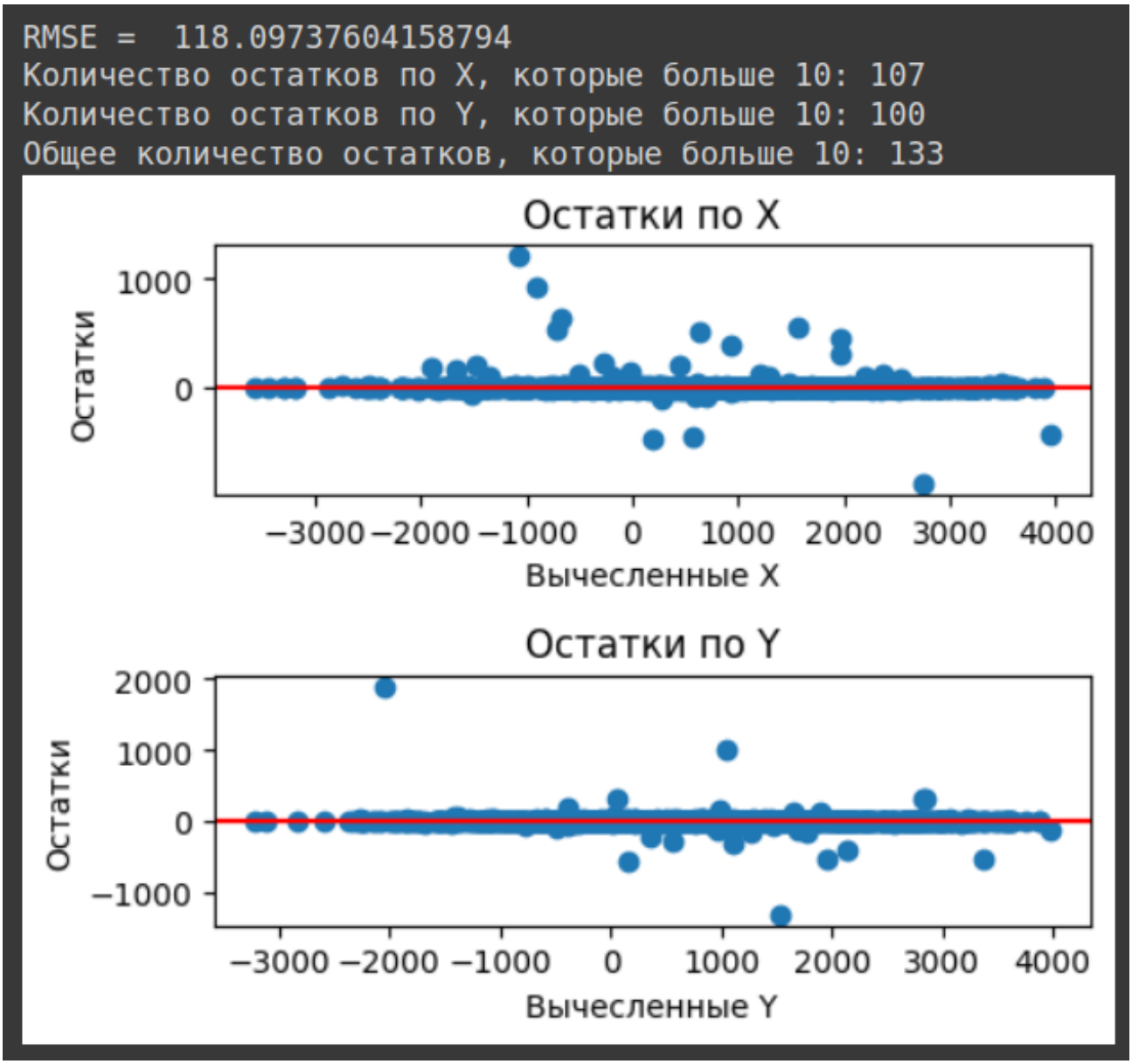


Рисунок 27 - Попытка увеличения устойчивости шума путём увеличения числа градиентных спусков

К сожалению, можно сделать вывод, что оптимизация оказалась провальной на данном этапе разработки алгоритма.

В данной работе был представлен алгоритм определения местоположения объекта на основе задержки сигнала от реперных точек. Алгоритм был реализован с использованием метода гиперболической навигации и оптимизирован с помощью метода градиентного спуска.

В ходе работы было проведено тестирование алгоритма на сгенерированных данных. Результаты тестирования показали, что алгоритм обладает достаточно высокой точностью при определении местоположения объекта. Однако было также обнаружено, что алгоритм подвержен шуму, что может снижать его точность в реальных условиях. Также, не сработала ни одна из предложенных оптимизаций.

В целом, результаты данной работы показывают, что предложенный алгоритм является перспективным для применения в системах навигации.

4 Применение алгоритма для разных задач

4.1 Портирование на язык C

Алгоритм был портирован на язык C с целью использования в микрокомпьютерах для аппаратно-программной навигации. Было также произведено тестирование алгоритма на тех же 1000 данных. Тестирование дало результат отклонения в 20 процентов. Что делает алгоритм не особо применимым для точных вычислений, тем не менее, производительность, и потребление памяти готовой программы оказались приемлемы для выполнения на микрокомпьютерах: 40 секунд без оптимизации и 1.5 секунды с оптимизацией O1, O2, O3. Правда, тестирование было выполнено с использованием процессора Intel i7-3940XM, что явно производительней микрокомпьютера, что ставит вопрос о скорости выполнения программы открытым. Размер использованного стека был не более 9 килобайт, а размер программы составил 20328 байта без оптимизации и 16192 байта с оптимизацией. На рисунке 28 показан результат тестирования программы.

Рисунок 28 - Результат тестирования программы портированной на C

4.2 Разработка клиент-сервера

Также, из-за сомнений в производительности кода на языке C, был предложен другой подход, который заключается в том, чтобы выполнять все

вычисления на сервере, куда будут поступать данные с клиента, то бишь приёмника. Простую схему можно посмотреть на рисунке 29.

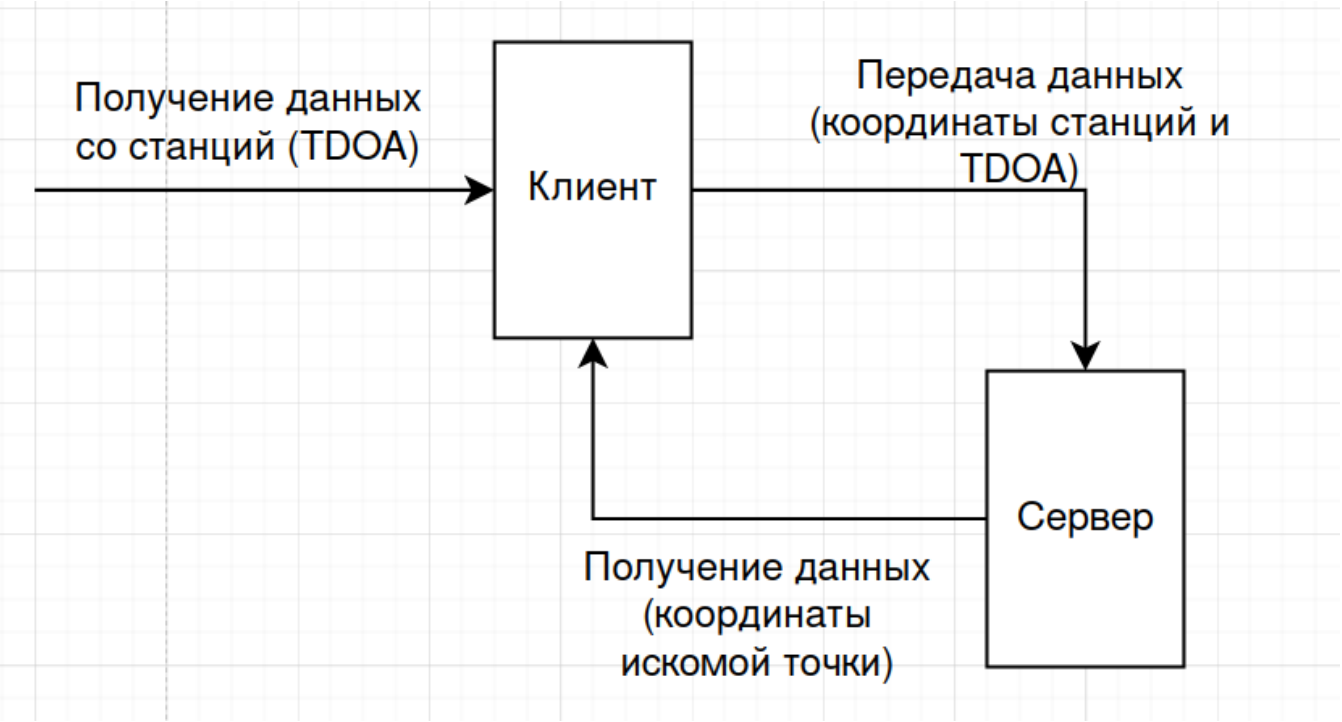


Рисунок 29 - Схема архитектуры клиент-сервер

В текущей имплементации сервер работает на языке Go по протоколу http, принимает в себя POST запрос в формате JSON, который содержит координаты реперных точек. Далее запускает 2 питоновских скрипта, первый генерирует случайную точку и её TDOA, второй вычисляет эту искомую точку и все сгенерированные и вычисленные данные. Сервер тестировался на виртуальной машине под debian 12.

Клиент был написан для Android устройств с использованием UI фреймфорка Jetpack Compose и дизайн-системой Material You, для работы с http протоколом была использована библиотека Retrofit (рисунок 30).

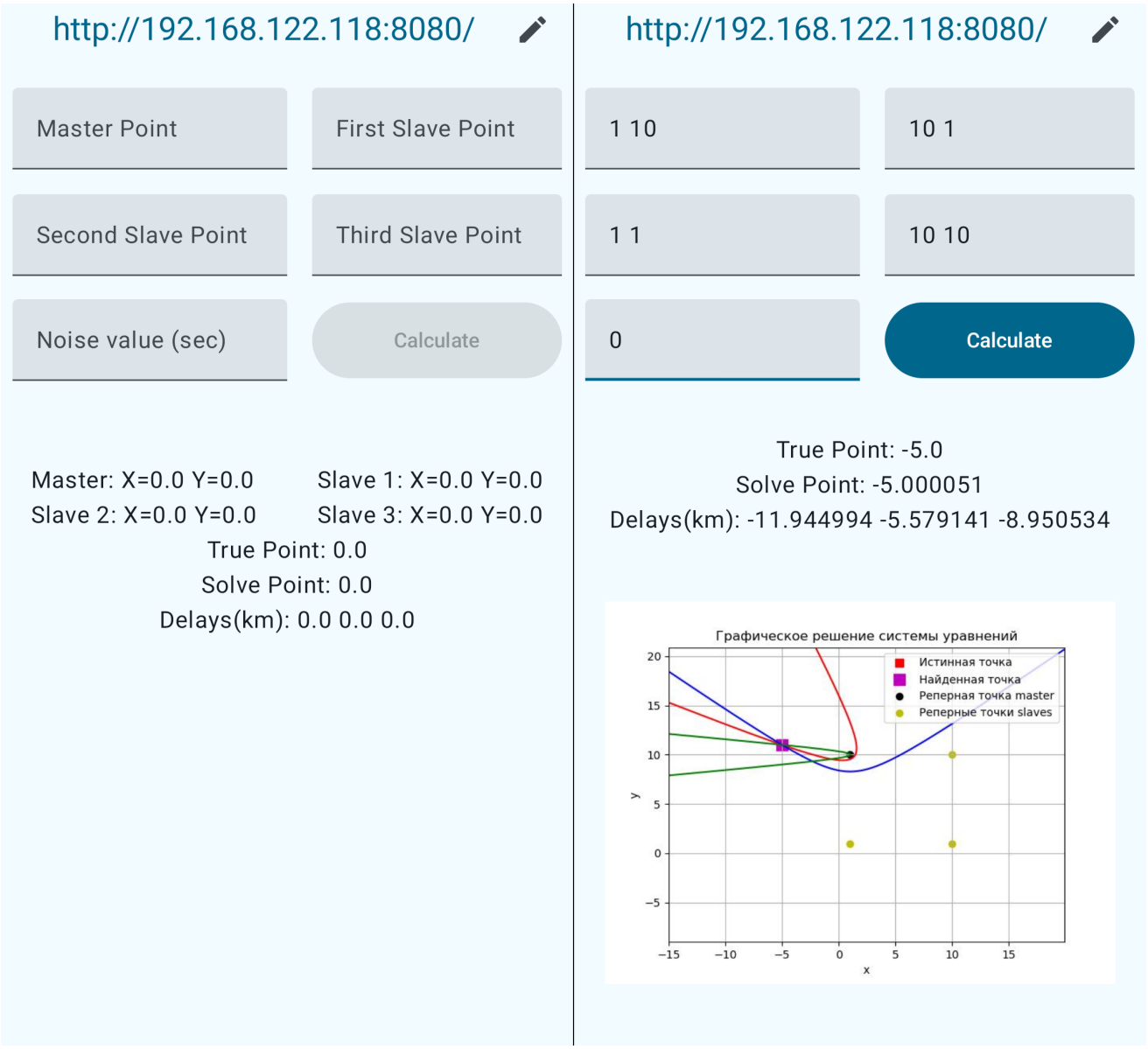


Рисунок 30 - Android клиент для работы с сервером

В данной главе были рассмотрены два подхода к применению алгоритма:

- портирование на язык C;
- разработка клиент-сервера.

Портирование на язык C позволило использовать алгоритм на микрокомпьютерах. Однако тестирование показало отклонение в 20%, что делает его не совсем подходящим для точных вычислений. Несмотря на это, производительность и потребление памяти оказались приемлемыми для работы на микрокомпьютерах. Вопрос о скорости выполнения программы остается

открытым, так как тестирование проводилось на процессоре, который значительно превосходит по производительности микрокомпьютер.

Разработка клиент-сервера была предложена в связи с сомнениями в производительности кода на языке C. Сервер, работающий на языке Go, принимает POST запросы в формате JSON и запускает два скрипта на Python для генерации и вычисления точек. Клиентская часть была разработана для Android устройств с использованием UI фреймворка Jetpack Compose с дизайн-системой Material You и библиотеки Retrofit для работы с http протоколом.

В целом, оба подхода имеют свои преимущества и недостатки, и выбор между ними будет зависеть от конкретных условий и требований. Таким образом, в данной главе были представлены различные способы применения алгоритма, что позволяет выбрать наиболее подходящий вариант в зависимости от конкретных условий и требований.

Заключение

В данной работе было проведено исследование в области навигационных систем, с основным фокусом на разработке алгоритма определения местоположения объекта на основе задержки сигнала от реперных точек.

В ходе работы были рассмотрены два основных метода навигации - TOF и TDOA, их основные принципы работы, требования и потенциальные области применения. Было проведено сравнение этих методов, что позволило выявить их преимущества и недостатки.

Далее была решена математическая задача определения положения на плоскости по заданной разности сигналов. Были рассмотрены принципы работы евклидова расстояния и методов оптимизации, включая метод Ньютона и градиентный спуск. Был проведен анализ функции стоимости и градиента функции стоимости, которые играют ключевую роль в алгоритме градиентного спуска.

На основе полученных знаний был разработан алгоритм определения местоположения объекта. Алгоритм был протестирован на сгенерированных данных, результаты тестирования показали его высокую точность. Однако было обнаружено, что алгоритм подвержен шуму, что может снижать его точность в реальных условиях.

Также была попытка портирования алгоритма на язык C, для выполнения программы на микрокомпьютере. Тестирование показало малое потребление ресурсов и увеличение производительности относительно реализации на Python, тем не менее точность снизилась до 20%.

В целом, результаты данной работы показывают, что предложенный алгоритм является перспективным для применения в системах навигации. Однако для повышения его устойчивости к шуму и улучшения точности определения местоположения требуется дальнейшая оптимизация алгоритма. Это исследование служит важным шагом в развитии навигационных систем и открывает новые возможности для дальнейших исследований в этой области.

Список используемых источников

1. Angle of Arrival (AoA): How It Works, How to Deploy, and Use Cases [Электронный ресурс] : Dusun. URL: https://www.dusuniot.com/resources/technical-brief/the-benefits-of-ble-aoa-and-how-you-can-use-it-in-life/ (дата обращения: 17.05.2024).

2. Decca Navigator Introduction Page [Электронный ресурс] : QSL. URL: https://www.qsl.net/g4ftc/decca/home.htm (дата обращения: 17.05.2024).

3. Elaboration of Methods and Algorithms for Passive Aircraft and Vehicle Detection over Time of Signal Arrival Differences [Электронный ресурс] : Graz University of Technology. URL: https://diglib.tugraz.at/download.php?id=576a75430e75f&location=browse (дата обращения: 17.05.2024).

4. Global Positioning System [Электронный ресурс] : Wikipedia. URL: https://en.wikipedia.org/wiki/Global_Positioning_System#Navigation_equations (дата обращения: 17.05.2024).

5. GNSS and Geodesy Concepts [Электронный ресурс] : North Group LTD. URL: https://northsurveying.com/index.php/soporte/gnss-and-geodesy-concepts (дата обращения: 17.05.2024).

6. Heffley, G. The Development Of Loran-C Navigation And Timing / G.Heffley — Washington : U.S. Dept. of Commerce, National Bureau of Standards, 1972. — 159p.

7. Hyperbolic position location estimator with TDOAs from four stations [Электронный ресурс] : New Jersey Institute of Technology. URL: https://archives.njit.edu/vol01/etd/2000s/2002/njit-etd2002-037/njit-etd2002-037.pdf (дата обращения: 17.05.2024).

8. Hyperbolic Radionavigation Systems [Электронный ресурс] : The Web Pages Of Jerry Proc. URL: https://jproc.ca/hyperbolic/ (дата обращения: 17.05.2024).

9. LORAN [Электронный ресурс] : Wikipedia. URL: https://en.wikipedia.org/wiki/LORAN (дата обращения: 17.05.2024).

10. Malik, A. RTLS For Dummies / A. Malik. — Hoboken, N.J. : Wiley, 2009. —336 p.

11. Multilateration [Электронный ресурс] : ScienceDirect. URL: https://www.sciencedirect.com/topics/engineering/multilateration (дата обращения: 17.05.2024).

12. Pseudo-range multilateration [Электронный ресурс] : Wikipedia. URL: https://en.wikipedia.org/wiki/Pseudo-range_multilateration (дата обращения: 17.05.2024).

13. Residuals vs. Fits Plot [Электронный ресурс] : Eberly College of Science. URL: https://online.stat.psu.edu/stat462/node/117/ (дата обращения: 17.05.2024).

14. Root Mean Square Error (RMSE) [Электронный ресурс] : Kalman filter for professionals. URL: https://kalman-filter.com/root-mean-square-error/ (дата обращения: 17.05.2024).

15. Time of arrival [Электронный ресурс] : Wikipedia. URL: https://en.wikipedia.org/wiki/Time_of_arrival (дата обращения: 17.05.2024).

16. True-range multilateration [Электронный ресурс] : Wikipedia. URL: https://en.wikipedia.org/wiki/True-range_multilateration (дата обращения: 17.05.2024).

17. Градиентный спуск [Электронный ресурс] : Википедия. URL: https://ru.wikipedia.org/wiki/Градиентный_спуск (дата обращения: 17.05.2024).

18. О некорректных задачах линейной алгебры и устойчивом методе их решения [Электронный ресурс] : Доклады Академии наук. URL: https://www.mathnet.ru/rus/dan31374 (дата обращения: 17.05.2024).

19. Обзор методов численной оптимизации. Безусловная оптимизация: метод линий [Электронный ресурс] : Хабр. URL: https://habr.com/ru/articles/561128/ (дата обращения: 17.05.2024).

20. Оптимизация (математика) [Электронный ресурс] : Википедия. URL: https://ru.wikipedia.org/wiki/Оптимизация_(математика) (дата обращения: 17.05.2024).

21. Тезисы докладов международного научного семинара по обратным и некорректно поставленным задачам [Электронный ресурс] : Кафедра математики физического факультета МГУ. URL: http://math.phys.msu.ru/data/248/semabstracts.pdf (дата обращения: 17.05.2024).