МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы решения задач оптимизации»

Тема: Решение задач многомерной безусловной оптимизации с использованием производных целевой функции

Студент гр. 9303 _____ Колованов Р.А.

Студент гр. 9303 _____ Птичкин С.А.

Преподаватель _____ Середа А.-В. И.

Санкт-Петербург

2023

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Колованов Р.А.

Студент Птичкин С.А.

Группа 9303

Тема работы: Решение задач многомерной безусловной оптимизации с использованием производных целевой функции

Исходные данные:

Язык программирования Python.

Содержание пояснительной записки:

«Содержание», «Введение», «Постановка задачи и обзор методов ее решения», «Разработка вычислительного алгоритма», «Разработка и тестирование программного модуля», «Исследование и анализ разработанных алгоритмов», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 26.09.2023

Дата сдачи реферата: 12.12.2023

Дата защиты реферата: 12.12.2023

Студент гр. 9303 _____ Колованов Р.А.

Студент гр. 9303 _____ Птичкин С.А.

Преподаватель _____ Середа А.-В. И.

АННОТАЦИЯ

В данной работе рассмотрена задача многомерной безусловной оптимизации. Сформулирована формальная постановка задачи. Проведён обзор методов решения задачи оптимизации: методы градиентного спуска (включая метод наискорейшего градиентного спуска), метод Ньютона, метод Гаусса-Зейделя, методы сопряженных направлений (включая метод Флетчера-Ривса), метод Дэвидона-Флетчера-Пауэлла и метод Марквардта. Было разработано программное средство, включающее в себя реализацию методов градиентного спуска, Ньютона и Флетчера-Ривса. В завершение был проведен экспериментальный анализ разработанных алгоритмов.

SUMMARY

In this paper, the problem of multidimensional unconditional optimization is considered. A formal statement of the problem is formulated. A review of methods for solving the optimization problem is carried out: gradient descent methods (including the steepest gradient descent method), Newton's method, Gauss-Seidel method, conjugate direction methods (including the Fletcher-Reeves method), the Davidon-Fletcher-Powell method and the Marquardt method. A software tool has been developed that includes the implementation of gradient descent, Newton and Fletcher-Reeves methods. Finally, an experimental analysis of the developed algorithms was carried out.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задачи многомерной безусловной оптимизации встречаются во многих областях, таких как инженерия, финансы, информатика и многие другие. При работе со сложными системами часто возникает необходимость одновременной оптимизации нескольких параметров для максимизации или минимизации требуемой функции.

Например, в инженерии оптимизация системы часто предполагает работу с множеством переменных, таких как свойства материала, размеры и нагрузки. При этом целью может быть минимизировать функцию веса, стоимости или максимизировать параметр прочности. В финансах оптимизация бюджета включает в себя поиск оптимального распределения активов для максимизации прибыли при минимизации рисков. В области машинного обучения и искусственного интеллекта оптимизация параметров модели имеет важное значение для достижения наилучшей производительности.

Для решения подобных задач оптимизации применяются различные модификации классических методов оптимизации, таких как градиентный спуск, методы Ньютона, метод сопряженных градиентов. Для каждой задачи применяются разные методы, поскольку они имеют разную эффективность при решении тех или иных классов задач.

Целью работы является решение задачи многомерной безусловной оптимизации с использованием производных целевой функции.

Для достижения поставленной цели были решены следующие задачи:

1. Формулировка задачи;
2. Обзор возможных методов решения задачи;
3. Выбор метода решения задачи;
4. Разработка вычислительного алгоритма и программного модуля;
5. Тестирование вычислительного алгоритма и программного модуля;
6. Проведение численных экспериментов и анализ результатов.

1. ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

1.1. Формулировка задачи

Задача поиска экстремума функции нескольких переменных без ограничений на возможные значения этих переменных может быть сформулирована следующим образом:

Найти вектор $\boldsymbol{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)^{\mathrm{T}}$, такой, что

$$f(\boldsymbol{x}^*) = \min_{x \in E_n} f(\boldsymbol{x}),$$

где $\boldsymbol{x} = \left(x_1, x_2, \ldots, x_n\right)^{\mathrm{T}}$ – вектор варьируемых параметров $x_1, x_2, \ldots, x_n$ задачи (решение задачи), n – размерность вектора $\boldsymbol{x}$, E_n – n-мерное эвклидово пространство (множество допустимых решений), $f(\boldsymbol{x})$ – вещественная скалярная функция векторного аргумента (целевая функция), $\boldsymbol{x}^*$ – оптимальное решение задачи.

1.2. Обзор возможных методов решения задачи

В рамках данной работы рассматриваются методы оптимизации, использующие производные целевой функции, то есть методы оптимизации первого и второго порядка. По результатам поиска возможных методов решения поставленной задачи были отобраны следующие методы:

- Методы градиентного спуска, в число которых входят:
  - Метод градиентного спуска с постоянным шагом;
  - Метод градиентного спуска с дроблением шага;
  - Метод наискорейшего градиентного спуска.
- Метод Ньютона, а также его модификации и аналоги:
  - Метод Ньютона-Рафсона;
  - Метод секущих;
- Метод Гаусса-Зейделя;

- Методы сопряженных направлений:
    - Метод Флетчера-Ривса;
    - Метод Полака-Рибьера;
- Метод Дэвидона-Флетчера-Пауэлла;
- Метод Марквардта.

1.2.1. Методы градиентного спуска

Стратегия метода заключается в построении последовательности точек $\{x_k\}$, таких, что $f(x_{k+1}) < f(x_k),\ k \in \{0\} \cup \mathbb{N}$. Точки последовательности $\{x_k\}$ вычисляются по правилу

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$

где $\nabla f(x_k)$ – градиент функции $f(x)$, α_k – шаг, $\alpha_k > 0$, $k \in \{0\} \cup \mathbb{N}$ – номер итерации.

Как известно, градиент $\nabla f(x_k)$ скалярной функции $f(x)$ в некоторой точке x_k направлен в сторону наискорейшего возрастания функции и ортогонален линии уровня. Вектор, противоположный градиенту $\nabla f(x_k)$, антиградиент, направлен в сторону наискорейшего убывания функции $f(x)$.

Таким образом, методы градиентного спуска используют антиградиент целевой функции, умноженный на некоторый шаг α_k, для определения направления движения к локальному минимуму.

Различные вариации методов градиентного спуска отличаются друг от друга способами выбора шага α_k.

Для сходимости градиентные методы требуют наличие ряда свойств целевой функции $f(x)$:

- Функция ограничена снизу на множестве допустимых решений;
- Функция имеет непрерывные частные производные первого порядка во всех точках множества допустимых решений;
- Градиент функции удовлетворяет условию Липшица.

Соответственно, методы градиентного спуска являются методами первого порядка – они используют частные производные первого порядка (градиент).

Наличие этих свойств гарантирует сходимость к стационарным точкам, которые могут быть локальным или глобальным минимумом. Поэтому для найденной стационарной точки необходимо проводить анализ. Для сильно выпуклых функций гарантируется линейная сходимость к глобальному минимуму.

Метод градиентного спуска с постоянным шагом предполагает, что α_k не меняется от итерации к итерации. Это может приводить к неприемлемо большому количеству итераций или колебаниям вокруг точки минимума. По этой причине были разработаны другие методы градиентного спуска.

Метод градиентного спуска с дроблением шага позволяет решить проблему колебаниями вокруг точки минимума, но вычислительная сложность при этом возрастает.

Метод наискорейшего градиентного спуска позволяет решить проблему с колебаниями вокруг точки минимума, а также проблему с большим количеством итераций, но вычислительная сложность при этом по сравнению с двумя предыдущими методами возрастает.

1.2.2. Метод Ньютона

Стратегия метода Ньютона заключается в построении последовательности точек $\{x_k\}$, таких, что $f(x_{k+1}) < f(x_k),\ k \in \{0\} \cup \mathbb{N}$. Точки последовательности $\{x_k\}$ вычисляются по правилу

$$x_{k+1} = x_k + d_k,$$

где $k \in \{0\} \cup \mathbb{N}$ – номер итерации, d_k – направление спуска, определяющееся по формуле

$$d_k = -H^{-1}(x_k)\nabla f(x_k),$$

где $H(x_k)$ – матрица Гессе функции $f(x)$, $\nabla f(x_k)$ – градиент функции $f(x)$.

Такой выбор d_k гарантирует выполнение требования $f(x_{k+1}) < f(x_k)$ при условии, что $H(x_k) > 0$. Формула для d_k была получена из следующих соображений:

1. Функция $f(x)$ аппроксимируется в каждой точке последовательности квадратичной функцией $F_k = f(x_k) + (\nabla f(x_k), d_k) + \frac{1}{2}(d_k, H(x_k) d_k)$;
2. Направление d_k определяется из необходимого условия экстремума первого порядка: $\frac{\partial F_k}{\partial d_k} = 0$.

Для сходимости градиентные методы требуют наличие ряда свойств целевой функции $f(x)$:

- Функция является сильно выпуклой и ограниченной снизу;
- Функция дважды непрерывно дифференцируема во всех точках множества допустимых решений;
- Функция удовлетворяет условию $\|H(x) - H(y)\| \le L\|x - y\| \quad \forall x, y \in R^n$, $L > 0$;
- Начальная точка x_0 удовлетворяет условию $\|\nabla f(x_0)\| \le \frac{8l^2}{L}$, где l – параметр сильной выпуклости функции $f(x)$.

Наличие этих свойств гарантирует квадратичную сходимость к точке глобального минимума.

Следует отметить, что на практике проверка данных свойств может быть крайне затруднительна, поэтому следует анализировать матрицу Гессе на выполнение условия $H(x_k) > 0$. Чтобы обеспечить выполнение требования $f(x_{k+1}) < f(x_k)$ в случаях, когда для каких-либо значений матрица Гессе $H(x_k)$ не окажется положительно определенной, необходимо для соответствующих значений k находить точку x_{k+1} по методу градиентного спуска.

Метод Ньютона является методом второго порядка – он использует частные производные второго порядка (матрица Гессе), а также частные производные первого порядка (градиент).

Для гарантированной сходимости метод Ньютона требует выбор хорошей начальной точки x_0, достаточно близкой к x^*, что не всегда является тривиальной задачей. Данная проблема решается использованием метода Ньютона-Рафсона. В нем направление d_k дополнительно умножается на регулируемый шаг t_k, чего оказывается достаточно для сходимости метода вне зависимости от выбора начальной точки x_0.

Метод секущих является конечно-разностным аналогом метода Ньютона, который сходится более медленно, но в свою очередь менее трудоемок на каждой итерации.

1.2.3. Метод Гаусса-Зейделя

Стратегия метода Гаусса-Зейделя заключается в построении последовательности точек $\{x_k\}$, таких, что $f(x_{k+1}) < f(x_k),\ k \in \{0\} \cup \mathbb{N}$. Точки последовательности $\{x_k\}$ вычисляются по правилу

$$x_{jk+1} = x_{jk} - t_k \left(\frac{\partial f(x)}{\partial x_{k+1}}\right)_{x=x_{jk}} * e_{k+1},$$

где $j \in \{0\} \cup \mathbb{N}$ – номер цикла вычислений, k = 0, 1, ..., n-1 – номер итерации внутри цикла, e_{k+1} – единичный вектор, k+1 проекция которого равна 1, t_k – размер шага.

Величина шага t_k выбирается из условия

$$\varphi(t_k) = f\left(x_{jk} + t_k \left(\frac{\partial f(x)}{\partial x_{k+1}}\right)_{x=x_{jk}} * e_{k+1}\right) \to \min_{t^k},$$

которое может решаться либо с использованием численных методов, либо исходя из условий $\frac{\partial \varphi}{\partial t_k} = 0$, $\frac{\partial^2 \varphi}{\partial {t_k}^2} > 0$.

Для сходимости метод Гаусса-Зейделя требует наличие ряда свойств целевой функции $f(x)$:

- Функция ограничена снизу;
- Функция имеет непрерывные частные производные первого порядка во всех точках множества допустимых решений;
- Градиент функции удовлетворяет условию Липшица.

Соответственно, метод Гаусса-Зейделя является методом первого порядка так как использует частные производные первого порядка (градиент).

Наличие этих свойств гарантирует сходимость к стационарным точкам, которые могут быть локальным или глобальным минимумом. Поэтому для найденной стационарной точки необходимо проводить анализ.

1.2.4. Метод Флетчера-Ривса

Стратегия метода Флетчера-Ривса заключается в построении последовательности точек $\{x_k\}$, таких, что $f(x_{k+1}) < f(x_k),\ k \in \{0\} \cup \mathbb{N}$. Точки последовательности $\{x_k\}$ вычисляются по правилу

$$x_{k+1} = x_k + t_k d_k,$$

где $k \in \{0\} \cup \mathbb{N}$ – номер итерации, t_k – размер шага, d_k – направление спуска, определяющееся по формулам

$$d_k = -\nabla f(x_k) + \beta_{k-1} d_{k-1},$$

$$d_0 = -\nabla f(x_k),$$

$$\beta_{k-1} = \frac{\left\|\nabla f(x_k)\right\|^2}{\left\|\nabla f(x_{k-1})\right\|^2},$$

где $\nabla f(x_k)$ – градиент функции $f(x)$.

Величина шага t_k определяется из условия $\varphi(t_k) = f\left(x_k + t_k d_k\right) \to \min\limits_{t^k}$, которое может решаться либо с использованием численных методов, либо исходя из условий $\frac{\partial \varphi}{\partial t_k} = 0$, $\frac{\partial^2 \varphi}{\partial {t_k}^2} > 0$.

Метод Флетчера-Ривса является методом первого порядка – он использует частные производные первого порядка (градиент).

Если функция $f(x)$ ограничена снизу и ее градиент удовлетворяет условию Липшица, то метод сходится. При минимизации квадратичных функций метод является конечным, иначе – не конечным. При этом, для квадратичных функций с положительно определенной матрицей Гессе метод сходится за число шагов, не превышающее *n*. Если функция $f(x)$ является трижды дифференцируемой и сильно выпуклой, то метод сходится к минимуму с квадратичной скоростью.

Преимуществом данного метода является то, что для сильно выпуклых функций скорость его сходимости является квадратичной, при этом сам метод является методом первого порядка.

У метода Флетчера-Ривса существует модификация – метод Полака-Рибьера, который позволяет использовать его уже для неквадратичных функций.

1.2.5. Метод Дэвидона-Флетчера-Пауэлла

Стратегия метода Дэвидона-Флетчера-Пауэлла заключается в построении последовательности точек $\{x_k\}$, таких, что $f\left(x_{k+1}\right) < f\left(x_k\right),\ k \in \{0\} \cup \mathbb{N}$. Точки последовательности $\{x_k\}$ вычисляются по правилу

$$x_{k+1} = x_k + t_k A_k \nabla f\left(x_k\right),$$

где $k \in \{0\} \cup \mathbb{N}$ – номер итерации, t_k – размер шага, $\nabla f\left(x_k\right)$ – градиент функции $f(x)$, A_k – матрица размера *n* на *n*, которая определяется по формулам

$$A_{k+1} = A_k + A_{kc},$$

$$A_0 = E,$$

$$A_{kc} = \frac{\Delta x_k \left(\Delta x_k\right)^T}{\left(\Delta x_k\right)^T \Delta g_k} - \frac{A_k \Delta g_k \left(\Delta g_k\right)^T A_k}{\left(\Delta g_k\right)^T A_k \Delta g_k},$$

где $\Delta x_k = x_{k+1} - x_k$, $\Delta g_k = \nabla f\left(x_{k+1}\right) - \nabla f\left(x_k\right)$.

Величина шага t_k определяется из условия $\varphi(t_k) = f\left(x_k - t_k A_k \nabla f\left(x_k\right)\right) \to \min_{t^k}$, которое может решаться либо с использованием численных методов, либо исходя из условий $\frac{\partial \varphi}{\partial t_k} = 0$, $\frac{\partial^2 \varphi}{\partial t_k{}^2} > 0$.

Метод Дэвидона-Флетчера-Пауэлла в применении к квадратичной функции вида с положительно определенной матрицей Гессе H обеспечивает отыскание минимума $x^* = H^{-1}b$ не более чем за n шагов.

Если $f(x)$ дважды непрерывно дифференцируема и $H\left(x^*\right) > 0$, то метод Дэвидона-Флетчера-Пауэлла с обновлением сходится к точке x^* со сверхлинейной скоростью. Если дополнительно справедливо, что $\left\|H(x)y\right\| \le k\left\|y\right\| \ \ \forall y \in R^n$ в окрестности точки x^*, то метод сходится к точке x^* с квадратичной скоростью.

1.2.6. Метод Марквардта

Стратегия метода Марквардта заключается в построении последовательности точек $\left\{x_k\right\}$, таких, что $f\left(x_{k+1}\right) < f\left(x_k\right)$, $k \in \{0\} \cup \mathbb{N}$. Точки последовательности $\left\{x_k\right\}$ вычисляются по правилу

$$x_{k+1} = x_k - \left[H\left(x_k\right) + \mu_k E\right]^{-1} \nabla f\left(x_k\right),$$

где $k \in \{0\} \cup \mathbb{N}$ – номер итерации, $H\left(x_k\right)$ – матрица Гессе функции $f(x)$, μ_k – последовательность положительных чисел, таких, что матрица $\left[H\left(x_k\right) + \mu_k E\right]^{-1}$ положительно определена, E – единичная матрица, $\nabla f\left(x_k\right)$ – градиент функции $f(x)$.

За счет выбора достаточно большого μ_k обеспечивается неравенство $f(x_{k+1}) < f(x_k)$. Как правило, значение μ_0 назначается минимум на порядок больше, чем самый большой элемент матрицы Гессе в точке x_0. Далее логика выбора μ_k состоит в следующем:

- Если $f(x_{k+1}) < f(x_k)$, то $\mu_{k+1} = \mu_k / 2$;
- Иначе $\mu_{k+1} = 2\mu_k$.

По своей сути, метод Марквардта является комбинацией метода Ньютона и метода градиентного спуска. Из стратегии метода видно, что в зависимости от величины μ_k метод по своим свойствам приближается либо к методу Ньютона, либо к методу градиентного спуска. Это позволяет использовать положительные свойства обоих методов, а также исправлять некоторые недостатки. Например, в окрестности точки минимума x^* метод Марквардта обладает скоростью сходимости, близкой квадратичной, аналогично методу Ньютона, при этом за счет положительно определенной матрицы $\mu_k E$ решается проблема с положительной определенностью матрицы Гессе.

Использование метода Марквардта определяет некоторые ограничения на функцию $f(x)$:

- Функция ограничена снизу;
- Функция дважды непрерывно дифференцируема во всех точках множества допустимых решений.

Метод Марквардта является методом второго порядка – он использует частные производные второго порядка (матрица Гессе), а также частные производные первого порядка (градиент).

1.3. Выбор метода решения задачи

Список необходимых для реализации методов решения задачи был получен исходя из слов преподавателя о том, что в рамках курсовой работы необходимо реализовать методы, которые были разобраны в ходе лекций.

В качестве методов для решения поставленной задачи были выбраны следующие методы оптимизации:

- Метод наискорейшего градиентного спуска;
- Метод Ньютона;
- Метод Флетчера-Ривса.

2. РАЗРАБОТКА ВЫЧИСЛИТЕЛЬНОГО АЛГОРИТМА

2.1. Вычисление градиента и матрицы Гессе целевой функции

Поскольку во входных данных для алгоритма мы имеем только целевую функцию без функций ее градиента и матрицы Гессе, то необходимо реализовать алгоритмы для приближенного вычисления градиента и матрицы Гессе.

2.1.1. Алгоритм вычисления градиента целевой функции

Градиент целевой функции является набором частных производных целевой функции, поэтому для начала необходимо разработать алгоритм вычисления частной производной целевой функции.

Частная производная целевой функции в точке x рассчитывается по формуле

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x+\varepsilon_i) - f(x-\varepsilon_i)}{2\|\varepsilon_i\|},$$

где $\frac{\partial f}{\partial x_i}$ – частная производная целевой функции по аргументу x_i, f – целевая функция, $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ – вектор аргументов $x_1, x_2, \ldots, x_n$ целевой функции, n – размерность вектора x, $\varepsilon_i = (0,\ldots,0,\varepsilon,0,\ldots,0)^{\mathrm{T}}$ – вектор размера n, у которого i-ая компонента равна ε, а все остальные компоненты равны 0, ε – шаг разностной производной.

Градиент целевой функции в точке x определяется по формуле

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x),\ldots,\frac{\partial f}{\partial x_n}(x)\right)^T.$$

В качестве исходных данных имеется f – целевая функция, x – точка, в которой необходимо вычислить градиент, ε – шаг разностной производной (влияет на точность вычисления частной производной). Алгоритм вычисления градиента целевой функции представляет собой следующую последовательность действий:

0. Инициализируется переменная i = 1 и массив t = [];
1. Вычисляется $\frac{\partial f}{\partial x_i}(x)$ – частная производная целевой функции по i-ой компоненте в точке *x*. Вычисленное значение добавляется в массив t;
2. i увеличивается на 1;
3. Если i <= n, то осуществляется переход на шаг 1, иначе – на шаг 4;
4. Градиент целевой функции в точке *x* найден: массив t.

Исходный код реализованного алгоритма представлен в приложении А.

2.1.2. Алгоритм вычисления матрицы Гессе целевой функции

Матрица Гессе целевой функции является набором вторых частных производных целевой функции, поэтому для начала необходимо разработать алгоритм вычисления второй частной производной целевой функции.

Вторая частная производная целевой функции в точке *x* рассчитывается по формуле

$$\frac{\partial f}{\partial x_i \partial x_j}(x) = \frac{\frac{\partial f}{\partial x_i}(x + \varepsilon_j) - \frac{\partial f}{\partial x_i}(x - \varepsilon_j)}{2\|\varepsilon_j\|},$$

где $\frac{\partial f}{\partial x_i \partial x_j}$ – вторая частная производная целевой функции по аргументам x_i и x_j

Матрица Гессе целевой функции определяется по формуле

$$H(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1 \partial x_1}(x) & \cdots & \frac{\partial f}{\partial x_1 \partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n \partial x_1}(x) & \cdots & \frac{\partial f}{\partial x_n \partial x_n}(x) \end{pmatrix}.$$

В качестве исходных данных имеется f – целевая функция, x – точка, в которой необходимо вычислить матрицу Гессе, ε – шаг разностной производной (влияет на точность вычисления частных производных). Алгоритм вычисления

матрицы Гессе целевой функции представляет собой следующую последовательность действий:

0. Инициализируется переменная i = 1 и двумерный массив m = [];
1. Инициализируется переменная j = 1 и массив t = [];
2. Вычисляется $\frac{\partial f}{\partial x_i \partial x_j}(x)$ – вторая частная производная целевой функции по i-ой и j-ой компоненте в точке *x*. Вычисленное значение добавляется в массив t;
3. j увеличивается на 1;
4. Если j <= n, то осуществляется переход на шаг 2, иначе – на шаг 5;
5. Массив t добавляется в массив m;
6. i увеличивается на 1;
7. Если i <= n, то осуществляется переход на шаг 1, иначе – на шаг 7;
8. Матрица Гессе целевой функции в точке *x* найдена: двумерный массив m.

Исходный код реализованного алгоритма представлен в приложении А.

2.2. Алгоритм для метода наискорейшего градиентного спуска

В качестве исходных данных имеется f – целевая функция, x_0 – начальная точка и ε – точность сравнения.

Алгоритм для метода наискорейшего градиентного спуска представляет собой следующую последовательность действий:

0. Инициализируется переменная $k = 0$;
1. Вычисляется $\nabla f(x_k)$ – градиент функции в точке x_k;
2. Если $\|\nabla f(x_k) < \varepsilon\|$, то осуществляется переход на шаг 7;
3. Находится φ_k из решения задачи минимизации функции одной переменной $\varphi(\alpha_k) = min_{\alpha_k > 0}\{\varphi(\alpha) = f(x_k - \alpha * \nabla f(x_k))\}$;
4. Вычисляется x_{k+1} по формуле $x_{k+1} = x_k - \varphi_k \nabla f(x_k)$;

5. Если $\|x_{k+1} - x_k\| / \|x_k\| \leq \varepsilon$, то осуществляется переход на шаг 7;
6. k увеличивается на 1 и осуществляется переход на шаг 1.
7. Процесс окончен: x_k – искомая точка минимума $f(x)$.

Исходный код реализованного алгоритма представлен в приложении А.

2.3. Алгоритм для метода Ньютона

В качестве исходных данных имеется f – целевая функция, x_0 – начальная точка и ε – точность сравнения.

Алгоритм для метода Ньютона представляет собой следующую последовательность действий:

0. Инициализируется переменная $k = 0$;
1. Вычисляется $\nabla f(x_k)$ – градиент функции в точке x_k;
2. Если $\|\nabla f(x_k) < \varepsilon\|$, то осуществляется переход на шаг 7;
3. Вычисляется $H_k^{-1} = H^{-1}(x_k)$ и формируется вектор $S_k = -H_k^{-1} \nabla f(x_k)$;
4. Вычисляется x_{k+1} по формуле $x_{k+1} = x_k + S_k$
5. Если $\|x_{k+1} - x_k\| / \|x_k\| \leq \varepsilon$, то осуществляется переход на шаг 7;
6. k увеличивается на 1 и осуществляется переход на шаг 1.
7. Процесс окончен: x_k – искомая точка минимума $f(x)$.

Исходный код реализованного алгоритма представлен в приложении А.

2.4. Алгоритм для метода Флетчера-Ривса

В качестве исходных данных имеется f – целевая функция, x_0 – начальная точка и ε – точность сравнения.

Алгоритм для метода Флетчера-Ривса представляет собой следующую последовательность действий:

0. Инициализируется переменная $k = 0$;

1. Вычисляется $\nabla f(x_k)$ – градиент функции в точке x_k. Если $\|\nabla f(x_k) < \varepsilon\|$, перейти на выполнение пункта 7.
2. Если k = 0, то S_k вычисляется по формуле $S_k = -\nabla f(x_k)$ и осуществляется переход на шаг 4.
3. Вычисляется S_k по формуле

$$S_k = -\nabla f(x_k) + \frac{(\nabla f(x_k), \nabla f(x_k))}{(\nabla f(x_{k-1}), \nabla f(x_{k-1}))} S_{k-1}$$

4. Находится φ_k из решения задачи минимизации функции одной переменной $\varphi(\alpha_k) = min_{\alpha_k \epsilon E_1}\{\varphi(\alpha) = f(x_k + \alpha * S_k)\}$;
5. Вычисляется x_{k+1} по формуле $x_{k+1} = x_k + \varphi_k S_k$;
6. k увеличивается на 1. Если $\|x_{k+1} - x_k\| \leq \varepsilon$ или $k = n$, то осуществляется переход на шаг 7. Иначе – на шаг 1.
7. Процесс окончен: x_k – искомая точка минимума $f(x)$.

Исходный код реализованного алгоритма представлен в приложении А.

3. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ

3.1. Архитектура программного модуля

При разработке программного модуля использовались следующие технологии:

0. Язык программирования Python 3;
1. Графическая библиотека Qt 5, представленная в виде модуля pyqt5;
2. Модуль для работы с векторами и матрицами numpy;
3. Модуль для работы с символьными вычислениями sympy;
4. Модуль для выполнения научных и инженерных расчетов scipy;
5. Модуль для работы с регулярными выражениями re;
6. Модуль для работы с путями файловой системы pathlib;
7. Модуль для тестирования unittest.

Класс *OptimizationMethod* является базовым для всех классов, реализующих методы оптимизации. Он содержит функциональность для вычисления градиента и матрицы Гессе для некоторой функции.

Классы *GradientDescentMethod*, *ConjugateGradientsMethod* и *NewtonMethod* являются наследниками класса *OptimizationMethod* и реализуют методы наискорейшего градиентного спуска, сопряженных градиентов и Ньютона соответственно.

Класс *Function* представляет собой функцию, сформированную по строке пользователя.

Класс *MainWindow* представляет собой главное графическое окно приложения, внутри которого располагается пользовательский интерфейс. Обрабатывает действия пользователя и запускает необходимые методы оптимизации.

UML диаграмма классов представлена на рис. 3.1.

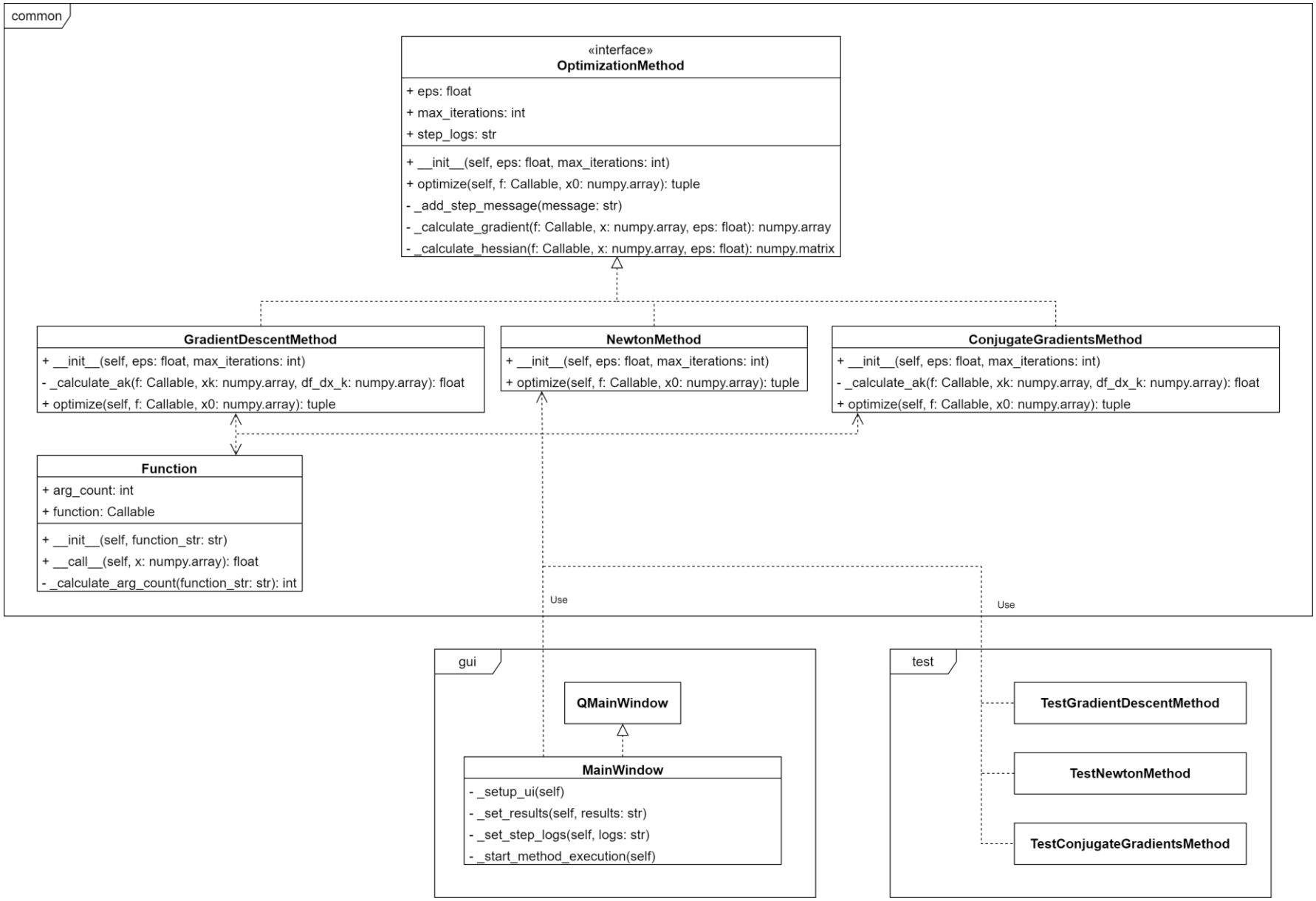

Рисунок 3.1 – UML диаграмма классов.

3.2. Пользовательский интерфейс программного модуля

Пользовательский интерфейс является графическим и состоит из трех секций: секция ввода целевой функции и начальной точки, секция выбора и настройки метода оптимизации целевой функции и секция вывода финальных и промежуточных результатов.

Целевая функция задается в текстовом формате. Допустимо использовать арифметические операции сложения (+), вычитания (-), умножения (*), деления (/) и возведения в степень (**), круглые скобки для изменения порядка выполнения подвыражений, обозначения для аргументов функции (x1, x2, …). Важно отметить, что первый аргумент функции обозначается как «x1», а не как «x0».

Начальная точка задается в текстовом формате в виде последовательного перечисления значений аргументов функции через запятую. Важно отметить, что количество значений в начальной точке должно совпадать с количеством аргументов целевой функции (x1, x2, …).

Метод оптимизации выбирается при помощи ComboBox, доступен выбор из трех методов оптимизации: метод наискорейшего градиентного спуска; метод Ньютона и метод сопряженных градиентов.

Максимальное количество итераций выбирается при помощи SpinBox и задает максимальное количество возможных итераций метода, при достижении которого работы метода прерывается.

Точность выбирается при помощи SpinBox и задает точность сравнения значений с плавающей точкой: $|a - b| < e$, где e – точность. Влияет на близость найденного оптимального значения к истинному оптимальному значению.

Кнопка «Найти оптимальное значение» позволяет запустить поиск оптимального значения при помощи выбранного метода.

В секция вывода финальных и промежуточных результатов имеются две вкладки: «Результаты» и «Шаги алгоритма». Во вкладке «Результаты» отображаются результаты работы метода. Во вкладке «Шаги алгоритма» отображаются все проделанные шаги метода, для каждого шага отображается

подробная информация о значениях различных параметров на момент выполнения шага.

Пользовательский графический интерфейс программного модуля представлен на рис. 3.2.

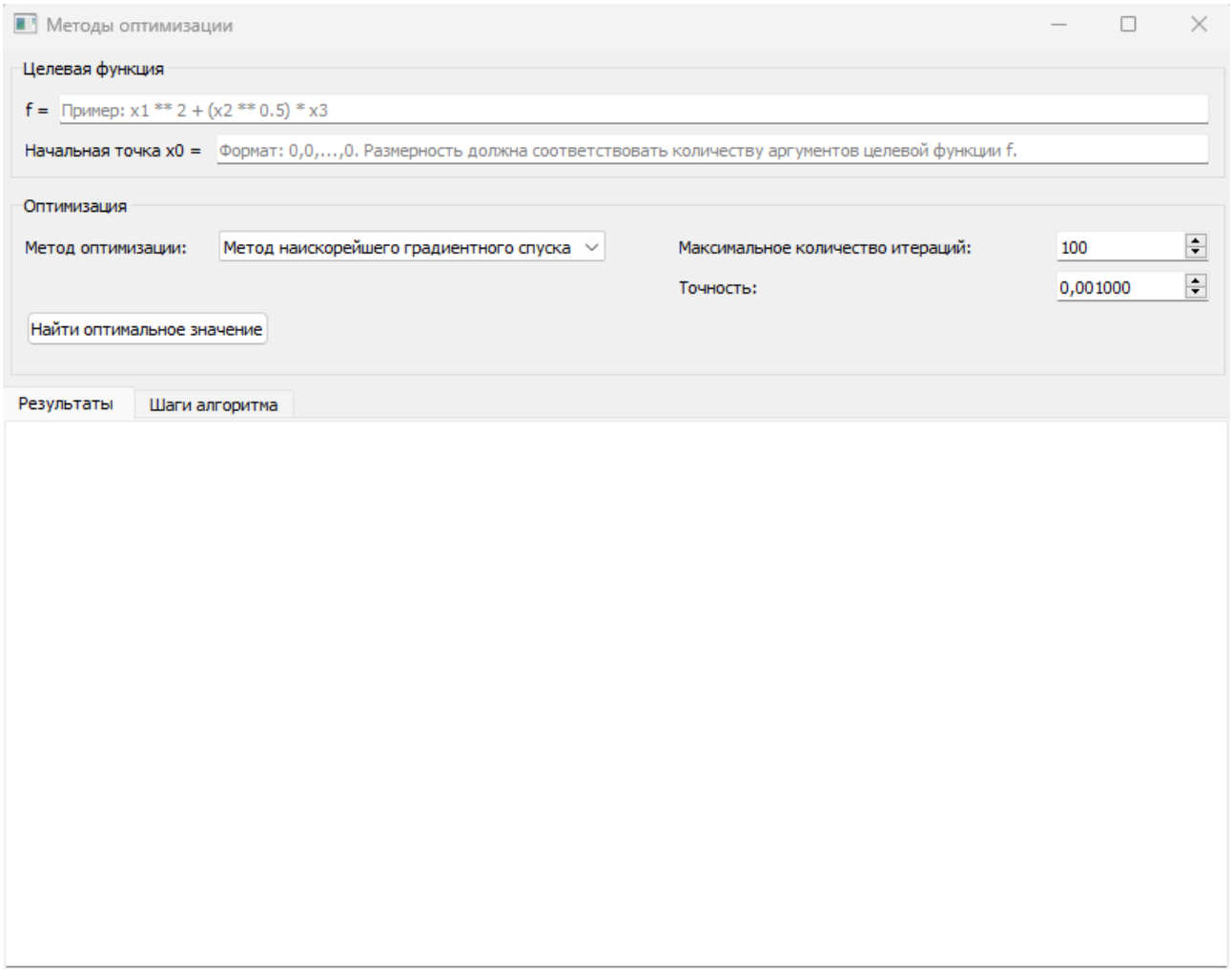

Рисунок 3.2 – Пользовательский интерфейс программного модуля.

3.3. Тестирование программного модуля

Тест-кейсы для метода наискорейшего градиентного спуска, метода Ньютона, метода Флетчера-Ривса представлены в таблице 3.1, таблице 3.2 и таблице 3.3 соответственно.

Таблица 3.1 – Тест-кейсы для метода наискорейшего градиентного спуска.

Случай	Функция	Начальная точка	Точка минимума	Точность	Результат тестирования
Корректность работы метода при различных начальных точках	(x1 - 10) ** 2 + (x2 + 5) ** 2 + x1*x2	(1, 1)	(16.66666666, -13.33333333)	0.00001	Тест пройден
	(x1 - 10) ** 2 + (x2 + 5) ** 2 + x1*x2	(-8, 19)	(16.66666666, -13.33333333)	0.00001	Тест пройден
	(x1 - 10) ** 2 + (x2 + 5) ** 2 + x1*x2	(13, 21)	(16.66666666, -13.33333333)	0.00001	Тест пройден
Корректность работы метода при разных величинах точности	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(15, -10)	(-1.33333333, 1.66666666)	0.01	Тест пройден
	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(15, -10)	(-1.33333333, 1.66666666)	0.001	Тест пройден
	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(15, -10)	(-1.33333333, 1.66666666)	0.0001	Тест пройден
Корректность работы метода при запуске из точки минимума	(x1) ** 2 + (x2) ** 2 + (x3) ** 2	(0, 0, 0)	(0, 0, 0)	0.00001	Тест пройден

Продолжение таблицы 3.1.

Случай	Функция	Начальная точка	Точка минимума	Точность	Результат тестирования
Корректность работы метода при удалённой начальной точке	(x1 + 2) ** 2 + (x2) ** 2 - 2 * x1	(1000, 1000)	(-1, 0)	0.00001	Тест пройден

Таблица 3.2 – Тест-кейсы для метода Ньютона.

Случай	Функция	Начальная точка	Точка минимума	Точность	Результат тестирования
Корректность работы метода при различных начальных точках	(x1 + 2) ** 2 + (x2 - 3) ** 2 + (x2 + 3) ** 2 - 3 * x1	(-5, -10)	(-0.5, 0)	0.00001	Тест пройден
	(x1 + 2) ** 2 + (x2 - 3) ** 2 + (x2 + 3) ** 2 - 3 * x1	(20, -17)	(-0.5, 0)	0.00001	Тест пройден
	(x1 + 2) ** 2 + (x2 - 3) ** 2 + (x2 + 3) ** 2 - 3 * x1	(43, 19)	(-0.5, 0)	0.00001	Тест пройден

Продолжение таблицы 3.2.

Случай	Функция	Начальная точка	Точка минимума	Точность	Результат тестирования
Корректность работы метода при разных величинах точности	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(-30, 24)	(-1.33333333, 1.66666666)	0.01	Тест пройден
	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(-30, 24)	(-1.33333333, 1.66666666)	0.001	Тест пройден
	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(-30, 24)	(-1.33333333, 1.66666666)	0.0001	Тест пройден
Корректность работы метода при запуске из точки минимума	(x1 - 3) ** 2 + (x2 + 7) ** 2 - 3 * x1	(4.5, -7)	(4.5, -7)	0.00001	Тест пройден
Корректность работы метода при удалённой начальной точке	(x1 + 6) ** 2 + (x2 - 1) ** 2 + 5 * x1	(1000, 1000)	(-8.5, 1)	0.00001	Тест пройден

Таблица 3.3 – Тест-кейсы для метода Флетчера-Ривса.

Случай	Функция	Начальная точка	Точка минимума	Точность	Результат тестирования
Корректность работы метода при различных начальных точках	(x1 + 2) ** 2 + (x1 - 3) ** 2 + (x2 + 3) ** 2	(4, -3)	(0.5, -3)	0.00001	Тест пройден
	(x1 + 2) ** 2 + (x1 - 3) ** 2 + (x2 + 3) ** 2	(-9, -5)	(0.5, -3)	0.00001	Тест пройден
	(x1 + 2) ** 2 + (x1 - 3) ** 2 + (x2 + 3) ** 2	(16, 10)	(0.5, -3)	0.00001	Тест пройден
Корректность работы метода при разных величинах точности	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(25, 25)	(-1.33333333, 1.66666666)	0.01	Тест пройден
	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(25, 25)	(-1.33333333, 1.66666666)	0.001	Тест пройден
	(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2	(25, 25)	(-1.33333333, 1.66666666)	0.0001	Тест пройден
Корректность работы метода при запуске из точки минимума	(x1 - 3) ** 2 + (x2 + 7) ** 2 - 3* x1	(4.5, -7)	(4.5, -7)	0.00001	Тест пройден

Продолжение таблицы 3.3.

Случай	Функция	Начальная точка	Точка минимума	Точность	Результат тестирования
Корректность работы метода при удалённой начальной точке	(x1 + x2) ** 2 + (x2 - 6) ** 2 + 9 * x1	(1000, -1000)	(-14.995, 10.5)	0.00001	Тест пройден
Корректность работы метода при положительно определённой матрице Гессе	(x1 + 6) ** 2 + (x2 - 1) ** 2 + 5 * x1	(100000, -100000)	(-8.5, 1)	0.00001	Тест пройден

4. ИССЛЕДОВАНИЕ И АНАЛИЗ РАЗРАБОТАННЫХ АЛГОРИТМОВ

4.1. Стратегия исследования алгоритмов

Стратегия исследования рассматриваемых алгоритмов заключается в следующей последовательности действий:

1. Постановка задачи для эксперимента: выбор целевой функции, зависящей от некоторого параметра $a > 0$, и выбор точности;
2. Аналитическое решение поставленной задачи;
3. Формирование перечня вариантов запуска алгоритмов: различные начальные точки и различные значения параметра a целевой функции;
4. Получение итогового количества шагов и количества вызовов целевой функции для всего перечня вариантов запуска каждого рассматриваемого алгоритма.

4.1.1. Постановка конкретной задачи

Минимизировать функцию $f(x_1, x_2, a) = (x_2 - x_1^2)^2 + a \cdot (x_1 - 1)^2$ с точностью до 10^{-5} $\left(\left| f(x_{1k}, x_{2k}, a) - f(x_1^*, x_2^*, a) \right| < 10^{-3} \right)$ рассматриваемыми методами оптимизации, то есть найти вектор $\boldsymbol{x}^* = \left(x_1^*,\ x_2^* \right)^{\mathrm{T}}$, такой, что

$$f(\boldsymbol{x}^*) = \min_{x \in E_n} f(\boldsymbol{x}).$$

4.1.2. Аналитическое решение задачи

Рассмотрим функцию $f(x_1, x_2, a) = (x_2 - x_1^2)^2 + a \cdot (x_1 - 1)^2$. Заметим, что значение выражений $(x_2 - x_1^2)^2$ и $(x_1 - 1)^2$ всегда положительны или равны 0, а параметр a по условию больше 0. Отсюда можно сделать вывод, что функция $F(x_1, x_2, a)$ не может принимать отрицательные значения. Предположим, что существуют такие x_1, x_2, что $F(x_1, x_2, a) = 0$. Отсюда:

$$\begin{cases} a \cdot (x_1 - 1)^2 = 0 \ (a > 0) \\ (x_2 - x_1{}^2)^2 = 0 \end{cases} \Rightarrow \begin{cases} x_1 = 1 \\ x_2 = 1 \end{cases}$$

$x^* = (1, 1)$ - глобальный минимум функции F

4.1.3. Перечень вариантов запуска алгоритмов

В качестве рассматриваемых начальных точек были выбраны три точки:

- Одна точка, расположенная относительно близко к точке глобального минимума – точка (0, 0);
- Две точки, расположенные относительно далеко от точки глобального минимума на каком-нибудь склоне – точка (2, 4) и точка (-4, 6).

В качестве значений параметра a были взяты два значение: относительно маленькое (0.1), и большое (10). Значения выбирались исходя из графиков целевой функции, представленные на рисунке 4.1.

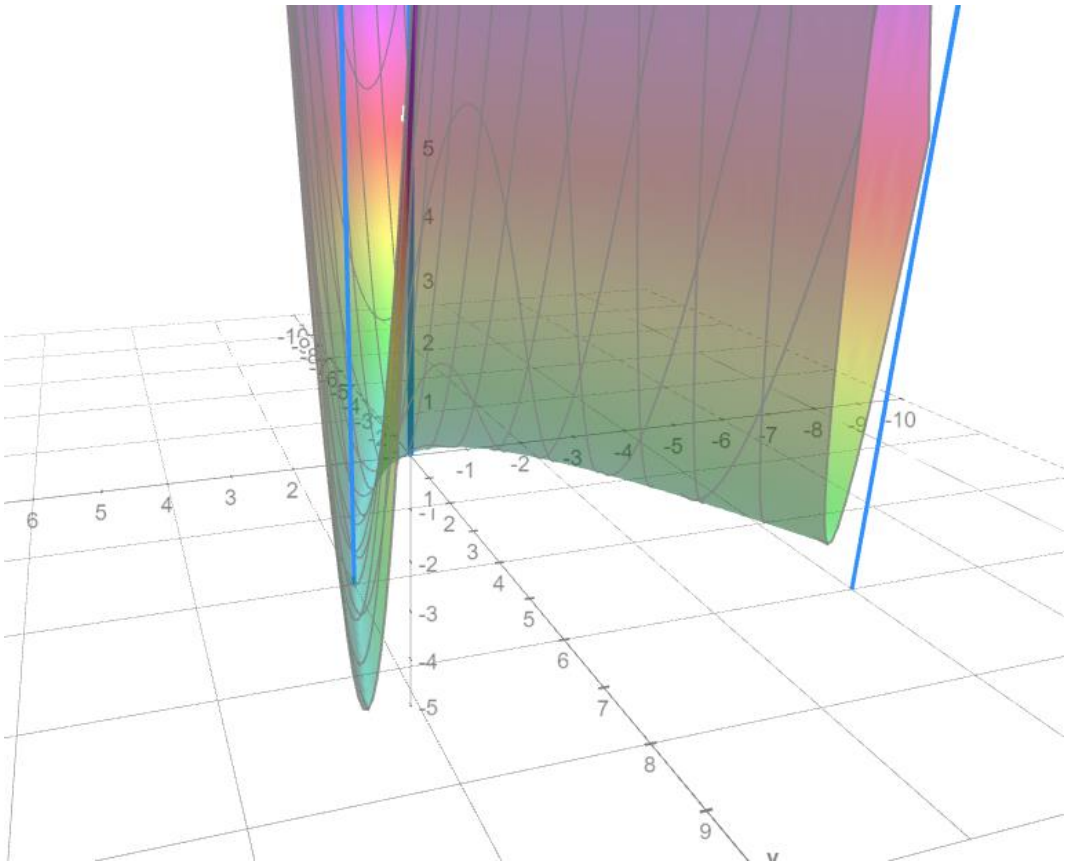

Рисунок 4.1 – Целевая функция при a = 0.1.

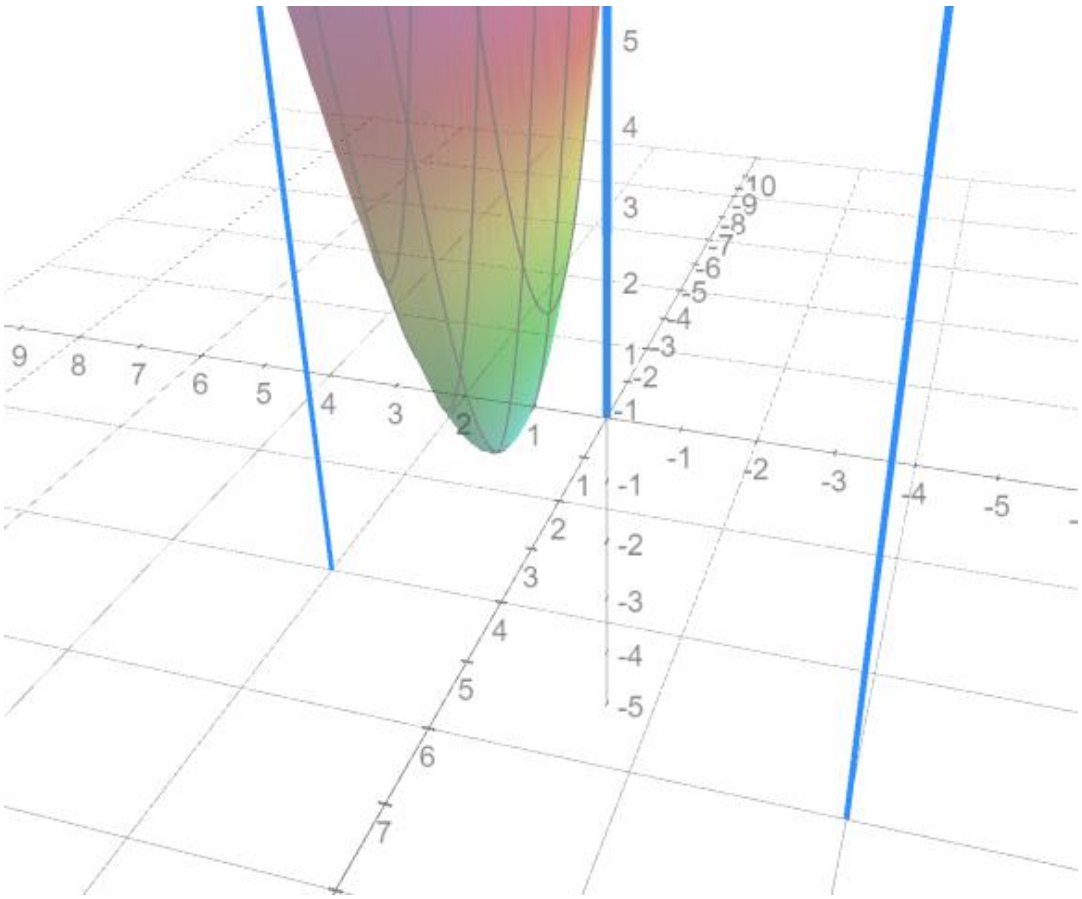

Рисунок 4.1 – Целевая функция при a = 10.

Перечень вариантов запуска алгоритмов представлен в таблице 4.1.

Таблица 4.1 – Перечень вариантов запуска алгоритмов.

№	Начальная точка	Параметр *a*
1	(0, 0)	0.1
2	(0, 0)	10
3	(-4, 6)	0.1
4	(-4, 6)	10
5	(2, 4)	0.1
6	(2, 4)	10

4.2. Исследование алгоритма наискорейшего градиентного спуска

4.2.1 Вычислительные эксперименты

Результаты работы алгоритма наискорейшего градиентного спуска при начальной точке (0, 0) и значении параметра a = 0.1 представлены в таблице 4.2.1.

Таблица 4.2.1 – Результаты работы алгоритма при x_0 = (0, 0) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	0.3234125	0.0	0.056717	56
2	0.32341617	0.10459565	0.045776	106
3	0.41755222	0.10459895	0.038789	158
…				
131	0.95032655	0.90051021	0.000253	6142
132	0.95033259	0.9031434	0.000246	6192
133	0.95167009	0.90314035	0.000240	6234
134	0.95167702	0.90570227	0.000233	6284
Найденная точка x*: (0.95168, 0.9057) Значение целевой функции в точке x*: 0.00023 Количество шагов: 134 Количество вызовов целевой функции: 6326				

Результаты работы алгоритма наискорейшего градиентного спуска при начальной точке (0, 0) и значении параметра a = 10 представлены в таблице 4.2.2.

Таблица 4.2.2 – Результаты работы алгоритма при x_0 = (0, 0) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	0.86875	0.0	0.74187	36
2	0.86991885	0.75878322	0.16922	86
3	0.96611346	0.75863392	0.04202	120
…				
7	0.9977564	0.98433439	0.00017	288
8	0.99788076	0.99599733	4.49e-05	338
9	0.99942361	0.99598085	1.15e-05	372
10	0.99945899	0.99898372	2.93e-06	422
Найденная точка x*: (0.99946, 0.99898)				

Значение целевой функции в точке x*: 2.9312e-06 Количество шагов: 10 Количество вызовов целевой функции: 456

Результаты работы алгоритма наискорейшего градиентного спуска при начальной точке (-4, 6) и значении параметра a = 0.1 представлены в таблице 4.2.3.

Таблица 4.2.3 – Результаты работы алгоритма при x_0 = (-4, 6) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	-2.47301563	6.1896875	1.21164	30
2	-2.47693575	6.17370145	1.21038	70
3	-2.46939901	6.17185389	1.20913	102
…				
437	0.96256837	0.92478674	0.00014	17818
438	0.96366812	0.92996326	0.00013	17874
439	0.96504084	0.92967163	0.00012	17914
440	0.96606156	0.93449149	0.00011	17970
Найденная точка x*: (0.96606, 0.93449) Значение целевой функции в точке x*: 0.00012 Количество шагов: 440 Количество вызовов целевой функции: 18010				

Результаты работы алгоритма наискорейшего градиентного спуска при начальной точке (-4, 6) и значении параметра a = 10 представлены в таблице 4.2.4.

Таблица 4.2.4 – Результаты работы алгоритма при x_0 = (-4, 6) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.9840625	6.4603125	16.053	38
2	2.0886036	4.94539645	12.190	84
3	1.68510774	4.91755192	9.0116	116
…				
19	1.0030388	1.02186816	0.00034	770
20	1.00394326	1.01063383	0.00016	818

21	1.00144812	1.01043305	7.77e-05	852
22	1.00188174	1.00507775	3.71e-05	900
Найденная точка x*: (1.00188, 1.00508) Значение целевой функции в точке x*: 3.7127e-05 Количество шагов: 22 Количество вызовов целевой функции: 934				

Результаты работы алгоритма наискорейшего градиентного спуска при начальной точке (2, 4) и значении параметра a = 0.1 представлены в таблице 4.2.5.

Таблица 4.2.5 – Результаты работы алгоритма при x_0 = (2, 4) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.9840625	4.0	0.09937	34
2	1.99371927	3.97474638	0.09874	84
3	1.98744111	3.97475706	0.09812	118
…				
409	1.07582841	1.16094325	0.00058	17882
410	1.07585405	1.15751772	0.00057	17932
411	1.07426112	1.15750581	0.00056	17972
412	1.07428705	1.15414926	0.00055	18022
Найденная точка x*: (1.07429, 1.15415) Значение целевой функции в точке x*: 0.00055 Количество шагов: 412 Количество вызовов целевой функции: 18062				

Результаты работы алгоритма наискорейшего градиентного спуска при начальной точке (2, 4) и значении параметра a = 10 представлены в таблице 4.2.6.

Таблица 4.2.6 – Результаты работы алгоритма при x_0 = (2, 4) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.51625	4.0	5.55849	32
2	1.51192489	2.27307403	2.62083	82
3	1.20050558	2.27384941	1.09530	116
…				

9	1.0045975	1.03207414	0.00073	368
10	1.00454821	1.009019	0.00020	418
11	1.00129357	1.00902599	5.81e-05	452
12	1.00124768	1.00240937	1.55e-05	502
Найденная точка x*: (1.00125, 1.00241) Значение целевой функции в точке x*: 1.5575e-05 Количество шагов: 12 Количество вызовов целевой функции: 536				

4.3. Исследование алгоритма Ньютона

4.3.1 Вычислительные эксперименты

Результаты работы алгоритма Ньютона при начальной точке (0, 0) и значении параметра a = 0.1 представлены в таблице 4.3.1.

Таблица 4.3.1 – Результаты работы алгоритма при x_0 = (0, 0) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.00000026	0.0	1.000001	20
2	1.00000046	1.00000098	2.45e-14	40
Найденная точка x*: (1.00000046, 1.00000098) Значение целевой функции в точке x*: 2.45272e-14 Количество шагов: 2 Количество вызовов целевой функции: 44				

Результаты работы алгоритма Ньютона при начальной точке (0, 0) и значении параметра a = 10 представлены в таблице 4.3.2.

Таблица 4.3.2 – Результаты работы алгоритма при x_0 = (0, 0) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.00000014	0.0	1.00000	20
2	1.00000003	0.99999999	1.88e-14	40
Найденная точка x*: (1.00000003, 0.99999999) Значение целевой функции в точке x*: 1.88721e-14 Количество шагов: 2 Количество вызовов целевой функции: 44				

Результаты работы алгоритма Ньютона при начальной точке (-4, 6) и значении параметра a = 0.1 представлены в таблице 4.3.3.

Таблица 4.3.3 – Результаты работы алгоритма при x_0 = (-4, 6) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	-3.97519356	15.80172175	2.47525	20
2	0.95639458	-23.4058699	591.4898	40
3	0.95653387	0.91452301	0.00018	60
4	0.99962593	0.99739508	3.46e-06	80
5	0.9999866	0.99997308	1.79e-11	100
Найденная точка x*: (0.99999, 0.99997) Значение целевой функции в точке x*: 1.79643e-11 Количество шагов: 5 Количество вызовов целевой функции: 104				

Результаты работы алгоритма Ньютона при начальной точке (-4, 6) и значении параметра a = 10 представлены в таблице 4.3.4.

Таблица 4.3.4 – Результаты работы алгоритма при x_0 = (-4, 6) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	-2.33331176	2.66643401	118. 8264	20
2	-0.19053461	-4.55507876	35.25451	40
3	0.43009342	-0.20019597	3.39629	60
4	0.9592372	0.64014284	0.09501	80
5	0.997838	0.994191	4.89e-05	100
6	0.99999936	0.99999404	2.59e-11	120
Найденная точка x*: (0.99999, 0.99999) Значение целевой функции в точке x*: 2.59545e-11 Количество шагов: 6 Количество вызовов целевой функции: 124				

Результаты работы алгоритма Ньютона при начальной точке (2, 4) и значении параметра a = 0.1 представлены в таблице 4.3.5.

Таблица 4.3.5 – Результаты работы алгоритма при $x_0 = (2, 4)$ и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.00000077	3.12517e-06	0.99999	20
2	1.00000086	1.00000177	7.69e-14	40
Найденная точка x*: (1.00000, 1.00000) Значение целевой функции в точке x*: 7.69289e-14 Количество шагов: 2 Количество вызовов целевой функции: 44				

Результаты работы алгоритма Ньютона при начальной точке (2, 4) и значении параметра a = 10 представлены в таблице 4.3.6.

Таблица 4.3.6 – Результаты работы алгоритма при $x_0 = (2, 4)$ и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.00000008	3.30775e-07	0.99999	20
2	1.00000003	0.99999999	1.88-14	40
Найденная точка x*: (1.00000003, 0.99999999) Значение целевой функции в точке x*: 1.88723e-14 Количество шагов: 2 Количество вызовов целевой функции: 44				

4.4. Исследование алгоритма Флетчера-Ривса

Поскольку данный алгоритм гарантированно сходится только для квадратичных функций, экспериментируемая функция была заменена на квадратичную:

$$f(x_1, x_2, a) = (x_2 - x_1)^2 + a \cdot (x_1 - 1)^2$$

Данная функция имеет глобальный минимум в точке (1, 1).

4.4.1 Вычислительные эксперименты

Результаты работы алгоритма Флетчера-Ривса при начальной точке (0, 0) и значении параметра a = 0.1 представлены в таблице 4.4.1.

Таблица 4.4.1 – Результаты работы алгоритма при x_0 = (0, 0) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	0.0909125	0.0	0.09090	50
2	0.64322039	0.30574615	1.64e-11	114
Найденная точка x*: (1.00000, 1.00000) Значение целевой функции в точке x*: 1.64992e-11 Количество шагов: 2 Количество вызовов целевой функции: 114				

Результаты работы алгоритма Флетчера-Ривса при начальной точке (0, 0) и значении параметра a = 10 представлены в таблице 4.4.2.

Таблица 4.4.2 – Результаты работы алгоритма при x_0 = (0, 0) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	0.90875	0.0	0.90909	36
2	1.00374	0.99985	0.00015	86
Найденная точка x*: (1.00374, 0.99985) Значение целевой функции в точке x*: 0.00015 Количество шагов: 2 Количество вызовов целевой функции: 86				

Результаты работы алгоритма Флетчера-Ривса при начальной точке (-4, 6) и значении параметра a = 0.1 представлены в таблице 4.4.3.

Таблица 4.4.3 – Результаты работы алгоритма при x_0 = (-4, 6) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.00708	1.125	0.00144	46
2	1.00708374	1.00078163	4.47e-05	112
Найденная точка x*: (1.00708, 1.00078)				

Значение целевой функции в точке x*: 4.47345e-05 Количество шагов: 2 Количество вызовов целевой функции: 112

Результаты работы алгоритма Флетчера-Ривса при начальной точке (-4, 6) и значении параметра a = 10 представлены в таблице 4.4.4.

Таблица 4.4.4 – Результаты работы алгоритма при x_0 = (-4, 6) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	1.42625	5.095625	15.28120	38
2	1.01823209	0.99785064	0.00373	88
Найденная точка x*: (1.018232, 0.99785) Значение целевой функции в точке x*: 0.00374 Количество шагов: 2 Количество вызовов целевой функции: 88				

Результаты работы алгоритма Флетчера-Ривса при начальной точке (2, 4) и значении параметра a = 0.1 представлены в таблице 4.4.5.

Таблица 4.4.5 – Результаты работы алгоритма при x_0 = (2, 4) и a = 0.1.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции
1	2.928625	3.0225	0.38077	46
2	0.99843472	1.00176811	1.13e-05	112
Найденная точка x*: (0.99843, 1.00177) Значение целевой функции в точке x*: 1.13564e-05 Количество шагов: 2 Количество вызовов целевой функции: 112				

Результаты работы алгоритма Флетчера-Ривса при начальной точке (2, 4) и значении параметра a = 10 представлены в таблице 4.2.6.

Таблица 4.2.6 – Результаты работы алгоритма при x_0 = (2, 4) и a = 10.

№ шага	x_1	x_2	$f(x_1, x_2)$	Количество вызовов целевой функции

1	1.195	3.79875	7.15976	36
2	1.0020923	1.0006305	4.59e-05	86
Найденная точка x*: (1.00209, 1.00063) Значение целевой функции в точке x*: 4.59141e-05 Количество шагов: 2 Количество вызовов целевой функции: 86				

4.5. Анализ полученных результатов

Исходя из результатов экспериментов можно сделать вывод, что рассматриваемые методы оптимизации работают корректно – для всех вариантов каждый метод нашел глобальный минимум, который совпадает истинным глобальным минимумом, полученным аналитически, с точностью 10^{-3}.

Метод наискорейшего градиентного спуска по сравнению с другими рассматриваемыми методами работает относительно долго (имеет большое количество шагов). Это наблюдение сходится с теоретическими данными о том, что метод имеет линейную скорость сходимости (метод Ньютона имеет квадратичную скорость сходимости, а метод Флетчера-Ривса – сходится за n шагов, где n – размерность вектора-аргумента целевой функции). Судя по тому, что при a = 0.1 метод работает значительно дольше, чем при a = 10, «сильная овражность» целевой функции способствует быстрой работе метода наискорейшего градиентного спуска. Выбор начальной точки также влияет на скорость работы метода – чем «ближе» начальная точка находится к точке глобального минимум, тем быстрее работает метод наискорейшего градиентного спуска.

Важно отметить, что одним из преимуществ метода градиентного спуска является низкие вычислительные затраты (низкое количество вычислений целевой функции), но в данном случае это преимущество не подтверждается, поскольку в ходе использования метода на каждой итерации необходимо решать задачу одномерной минимизации, которая решается с использованием вычислительного алгоритма (который в свою очередь требует дополнительных вызовов целевой функции).

Метод Ньютона для всех случаев показал относительно быструю скорость сходимости – не более 6 шагов, а также наименьшее количество вычислений целевой функции – около 20 на каждую итерацию. Это объясняется тем, что в методе Ньютона хоть и считается Гессиан, который увеличивает количество вычислений целевой функции, но при этом не надо решать задачу одномерной оптимизации на каждой итерации. Соответственно, вычисления целевой функции происходят только при вычислении градиента и Гессиана. Судя по тому, что при a = 0.1 метод работает примерно столько же, сколько при a = 10, «овражность» целевой функции не влияет на скорость работы метода. Выбор начальной точки же влияет на скорость работы метода – чем «ближе» начальная точка находится к точке глобального минимум, тем быстрее работает метод Ньютона.

Для метода Флетчера-Ривса использовалась другая целевая функция, а именно квадратичная, поскольку данный метод гарантирует сходимость только для квадратичных функций. В общем случае, данный метод будет работать быстрее, чем методы наискорейшего градиентного спуска и Ньютона – количество шагов не превышает размерности вектора-аргумента. Из экспериментальных данных видно, что метод сходится за не более чем 2 шага, что сходится с теоретическими данными (количество шагов не превышает размерности вектора-аргумента – 2). Выбор разных начальных точек на текущих исходных данных, не повлиял на скорость сходимости метода. Это может объяснятся тем, что целевая функция имеет аргумент небольшой размерности – 2, из-за чего изменение скорости сходимости не так заметно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была достигнута поставленная цель, а также были выполнены все поставленные задачи.

Для начала была рассмотрена задача многомерной безусловной оптимизации, после чего для была сформулирована формальная постановка задачи.

Далее был проведен обзор возможных методов решения задачи оптимизации. Было рассмотрено 6 методов оптимизации (включая некоторые модификации), а именно: методы градиентного спуска (включая метод наискорейшего градиентного спуска), метод Ньютона, метод Гаусса-Зейделя, методы сопряженных направлений (включая метод Флетчера-Ривса), метод Дэвидона-Флетчера-Пауэлла и метод Марквардта.

Далее из рассмотренных методов были выбраны три метода, которые интересны в рамках данной курсовой работы: метод наискорейшего градиентного спуска, метод Ньютона и метод Флетчера-Ривса.

Далее для выбранных методов были разработаны вычислительные алгоритмы. Для удобного использования разработанных алгоритмов решения поставленной задачи было разработано программное средство.

Далее для проверки корректности работы алгоритмов и программного средства были разработаны юнит-тесты, покрывающие основную функциональность. Все юнит-тесты были успешно пройдены.

В завершение были проведены вычислительные эксперименты для исследования реализованных методов, на основании которых был проведен некоторый анализ. Анализ показал, что метод Ньютона и метод Флетчера-Ривса обладают преимуществом в скорости перед методом наискорейшего градиентного спуска, но и обладают недостатками – на целевую функцию накладываются более серьезные ограничения, что ограничивает класс решаемых задач.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пантелеев А. В., Летова Т. А. Методы оптимизации в примерах и задачах. М.: Высшая Школа, 2005. 544 с.

2. Моисеев Н. Н., Иванилов Ю. П., Столярова Е. М. Методы оптимизации. М.: Главная редакция физико-математической литературы, 1978. 352 с.

3. Северин В. П. Методы многомерной безусловной оптимизации. Х.: НТУ «ХПИ», 2013. 160 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Листинг 1. Файл __main__.py

```
import sys
from PyQt5.QtWidgets import QApplication
from optimization_methods.gui import MainWindow


if __name__ == "__main__":
    try:
        app = QApplication(sys.argv)
        window = MainWindow()
        window.show()
        sys.exit(app.exec())
    except Exception as error:
        print(f"Ошибка: {error}")
```

Листинг 2. Файл common/methods.py

```
import numpy as np
from numpy.linalg import norm
from scipy.optimize import minimize
from typing import Callable


class OptimizationMethod:
    def __init__(self, eps: float, max_iterations: int):
        self.eps = eps
        self.max_iterations = max_iterations
        self.step_logs = ""

    def _add_step_message(self, message: str):
        self.step_logs += message

    @staticmethod
    def _calculate_gradient(f: Callable, x: np.array, eps: float = 0.00001) ->
np.array:
        def df_dxi(i: int) -> float:
            d = np.zeros(x.shape)
            d[i] = eps
            return (f(x + d) - f(x - d)) / (2 * eps)

        return np.array([df_dxi(i) for i in range(len(x))])

    @staticmethod
    def _calculate_inv_hessian(f: Callable, x: np.array, eps: float = 0.00001)
-> np.array:
        def df_dxi(x: np.array, i: int) -> float:
            d = np.zeros(x.shape)
            d[i] = eps
            return (f(x + d) - f(x - d)) / (2 * eps)

        def df_dxixj(x: np.array, i: int = 0, j: int = 0) -> float:
            d = np.zeros(x.shape)
            d[j] = eps
            return (df_dxi(x + d, i) - df_dxi(x - d, i)) / (2 * eps)
```

```
        n = len(x)
        matrix_arrays = np.array([[df_dxixj(x, i, j) for j in range(n)] for i
in range(n)])
        return np.linalg.inv(np.array(matrix_arrays))

    def optimize(self, f: Callable, x0: np.array) -> tuple:
        pass


class GradientDescentMethod(OptimizationMethod):
    def __init__(self, eps: float, max_iterations: int):
        super().__init__(eps, max_iterations)

    @staticmethod
    def _calculate_ak(f: Callable, xk: np.array, df_dx_k: np.array) -> float:
        alpha0 = np.array([0])
        return minimize(lambda alpha: f(xk - alpha * df_dx_k), alpha0,
method='nelder-mead').x

    def optimize(self, f: Callable, x0: np.array) -> (np.array, int):
        self._add_step_message(f"Начало работы метода. Начальная точка x0 =
{x0}.\n\n")

        k = 0
        x = [x0]
        a = []
        df_dx = []

        while k < self.max_iterations:
            df_dx.append(self._calculate_gradient(f, x[k]))
            if norm(df_dx[k]) < self.eps:
                break

            a.append(self._calculate_ak(f, x[k], df_dx[k]))
            x.append(x[k] - a[k] * df_dx[k])

            if np.sum(x[k]) != 0 and norm(x[k + 1] - x[k]) / norm(x[k]) <
self.eps:
                break

            self._add_step_message(f"[Шаг k={k + 1}]\n"
                                   f"    f(x_k) = {f(x[k + 1])}\n"
                                   f"    f(x_k) call count = {f.call_count -
(k + 1)}\n"
                                   f"    x_k = {x[k + 1]}\n"
                                   f"    a_k = {a[k]}\n"
                                   f"    grad(x_k) = {df_dx[k]}\n\n")

            k += 1

        self._add_step_message(f"Завершение работы метода. ")
        if k >= self.max_iterations:
            self._add_step_message(f"Превышено максимальное количество
итераций. Последняя найденная точка x_k = {x[k]}. Значение f(x_k) =
{f(x[k])}.\n\n")
        else:
            self._add_step_message(f"Оптимальная точка найдена. x* = {x[k]}.
Значение f(x*) = {f(x[k])}.\n\n")

        return x[k], k
```

```
class ConjugateGradientsMethod(OptimizationMethod):
    def __init__(self, eps: float, max_iterations: int):
        super().__init__(eps, max_iterations)

    @staticmethod
    def _calculate_ak(f: Callable, xk: np.array, sk: np.array):
        alpha0 = np.array([0])
        return minimize(lambda alpha: f(xk - alpha * sk), alpha0,
method='nelder-mead').x

    def optimize(self, f: Callable, x0: np.array) -> (np.array, int):
        self._add_step_message(f"Начало работы метода. Начальная точка x0 =
{x0}.\n\n")

        k = 0
        n = len(x0)
        x = [x0]
        a = []
        s = []
        df_dx = []

        while k < self.max_iterations:
            df_dx.append(self._calculate_gradient(f, x[k]))
            if norm(df_dx[k]) < self.eps:
                break

            if k == 0:
                s.append(-df_dx[k])
            else:
                s.append(-df_dx[k] + s[k - 1] * (np.dot(df_dx[k], df_dx[k]) /
np.dot(df_dx[k - 1], df_dx[k - 1])))

            a.append(self._calculate_ak(f, x[k], s[k]))
            x.append(x[k] - a[k] * s[k])

            self._add_step_message(f"[Шаг k={k+1}]\n"
                                   f"    x_k = {x[k+1]}\n"
                                   f"    f(x_k) = {f(x[k+1])}\n"
                                   f"    f(x_k) call count = {f.call_count -
(k+1)}\n"
                                   f"    a_k = {a[k]}\n"
                                   f"    s_k = {s[k]}\n"
                                   f"    grad(x_k) = {df_dx[k]}\n\n")

            k += 1

            if norm(x[k] - x[k - 1]) < self.eps or k == n:
                break

        self._add_step_message(f"Завершение работы метода. ")
        if k >= self.max_iterations:
            self._add_step_message(
                f"Превышено максимальное количество итераций. Последняя
найденная точка x_k = {x[k]}. Значение f(x_k) = {f(x[k])}.\n\n")
        else:
            self._add_step_message(f"Оптимальная точка найдена. x* = {x[k]}.
Значение f(x*) = {f(x[k])}.\n\n")

        return x[k], k


class NewtonMethod(OptimizationMethod):
    def __init__(self, eps: float, max_iterations: int):
```

```
        super().__init__(eps, max_iterations)

    def optimize(self, f: Callable, x0: np.array) -> (np.array, int):
        self._add_step_message(f"Начало работы метода. Начальная точка x0 =
{x0}.\n\n")

        k = 0
        x = [x0]
        h = []
        s = []
        df_dx = []

        while k < self.max_iterations:
            df_dx.append(self._calculate_gradient(f, x[k]))
            if norm(df_dx[k]) < self.eps:
                break

            h.append(self._calculate_inv_hessian(f, x[k]))
            s.append(np.array(np.transpose(np.negative(np.matmul(h[k],
df_dx[k])))).ravel())
            x.append(x[k] + s[k])

            if np.sum(x[k]) != 0 and norm(x[k + 1] - x[k]) / norm(x[k]) <
self.eps:
                break

            t = str(h[k]).replace("\n", "\n                    ")
            self._add_step_message(f"[Шаг k={k + 1}]\n"
                                   f"    x_k = {x[k + 1]}\n"
                                   f"    f(x_k) = {f(x[k + 1])}\n"
                                   f"    f(x_k) call count = {f.call_count -
(k + 1)}\n"
                                   f"    s_k = {s[k]}\n"
                                   f"    H(x_k) = {t}\n"
                                   f"    grad(x_k) = {df_dx[k]}\n\n")
            k += 1

        self._add_step_message(f"Завершение работы метода. ")
        if k >= self.max_iterations:
            self._add_step_message(
                f"Превышено максимальное количество итераций. Последняя
найденная точка x_k = {x[k]}. Значение f(x_k) = {f(x[k])}.\n\n")
        else:
            self._add_step_message(f"Оптимальная точка найдена. x* = {x[k]}.
Значение f(x*) = {f(x[k])}.\n\n")

        return x[k], k
```

Листинг 3. Файл common/function.py

```
import re
import sympy as sp
import numpy as np


class Function:
    def __init__(self, function_str: str):
        self.arg_count = self._calculate_arg_count(function_str)
        self.function = sp.sympify(function_str)
        self.call_count = 0
```

```
    @staticmethod
    def _calculate_arg_count(function_str: str) -> int:
        pattern = re.compile("x\\d+")
        results = re.findall(pattern, function_str)
        indexes = set([int(r[1:]) for r in results])
        return max(indexes)

    def __call__(self, x: np.array) -> float:
        self.call_count += 1
        args = {}
        for i in range(0, len(x)):
            args[f"x{i+1}"] = x[i]
        return float(self.function.subs(args))
```

Листинг 4. Файл gui/MainWindow.py

```
import numpy as np
from pathlib import Path
from PyQt5 import uic
from PyQt5.QtWidgets import QMainWindow
from optimization_methods.common import Function
from optimization_methods.common import GradientDescentMethod, NewtonMethod,
ConjugateGradientsMethod


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self._setup_ui()

    def _setup_ui(self):
        ui_filepath =
str(Path(__file__).parent.absolute().joinpath(f"{self.__class__.__name__}.ui")
)
        uic.loadUi(ui_filepath, self)

        self.startButton.pressed.connect(self._start_method_execution)

    def _set_results(self, results: str):
        self.resultsTextBrowser.setText(results)

    def _set_step_logs(self, logs: str):
        self.stepsTextBrowser.setText(logs)

    def _start_method_execution(self):
        self._set_results("")
        self._set_step_logs("")

        function_str = self.functionEdit.text()
        try:
            function = Function(function_str)
        except:
            self._set_results(f"Ошибка. Не удалось распознать целевую
функцию.\n"
                              f"Проверьте корректность ввода целевой
функции.\n")
            return

        x0_str = self.xEdit.text()
        try:
            x0 = np.array(np.fromstring(x0_str, dtype=float, sep=","))
```

```
        except:
            self._set_results(f"Ошибка. Не удалось распознать начальную
точку.\n"
                              f"Проверьте корректность ввода начальной
точки.\n")
            return

        if function.arg_count != len(x0):
            self._set_results(f"Ошибка. Количество аргументов целевой функции
f не совпадает с размерностью "
                              f"начальной точки x0.\nПроверьте корректность
ввода исходных данных.")
            return

        max_iterations = int(self.iterationSpinBox.value())
        eps = float(self.epsilonSpinBox.value())

        if self.methodComboBox.currentIndex() == 0:
            method_class = GradientDescentMethod
        elif self.methodComboBox.currentIndex() == 1:
            method_class = NewtonMethod
        elif self.methodComboBox.currentIndex() == 2:
            method_class = ConjugateGradientsMethod
        else:
            self._set_results(f"Выбран неизвестный метод.\n")
            return

        method = method_class(eps, max_iterations)

        try:
            x, steps = method.optimize(function, x0)
        except Exception as error:
            self._set_results(f"При работе метода возникла ошибка:
{error}\nПроверьте корректность ввода исходных данных.\n")
            return

        self._set_step_logs(method.step_logs)

        if steps >= method.max_iterations:
            self._set_results(f"Достигнуто максимальное количество итераций.
Последняя точка x_k = {x}. Значение f(x_k) = {function(x)}.\n")
        else:
            self._set_results(f"Оптимальное значение найдено. x* = {x}\n"
                              f"Значение f(x*) = {function(x)}\n"
                              f"Количество итераций: {steps}\n")
```

Листинг 5. Файл gui/MainWindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="minimumSize">
```

```
  <size>
   <width>800</width>
   <height>600</height>
  </size>
 </property>
 <property name="windowTitle">
  <string>Методы оптимизации</string>
 </property>
 <widget class="QWidget" name="centralwidget">
  <layout class="QVBoxLayout" name="verticalLayout">
   <property name="spacing">
    <number>0</number>
   </property>
   <property name="leftMargin">
    <number>0</number>
   </property>
   <property name="topMargin">
    <number>0</number>
   </property>
   <property name="rightMargin">
    <number>0</number>
   </property>
   <property name="bottomMargin">
    <number>0</number>
   </property>
   <item>
    <layout class="QVBoxLayout" name="topLayout">
     <property name="leftMargin">
      <number>6</number>
     </property>
     <property name="topMargin">
      <number>6</number>
     </property>
     <property name="rightMargin">
      <number>6</number>
     </property>
     <property name="bottomMargin">
      <number>6</number>
     </property>
     <item>
      <widget class="QGroupBox" name="functionGroupBox">
       <property name="title">
        <string>Целевая функция</string>
       </property>
       <layout class="QVBoxLayout" name="verticalLayout_2">
        <item>
         <layout class="QHBoxLayout" name="horizontalLayout">
          <item>
           <widget class="QLabel" name="functionLabel">
            <property name="text">
             <string>f =</string>
            </property>
           </widget>
          </item>
          <item>
           <widget class="QLineEdit" name="functionEdit">
            <property name="text">
             <string/>
            </property>
            <property name="placeholderText">
             <string>Пример: x1 ** 2 + (x2 ** 0.5) * x3</string>
            </property>
           </widget>
```

```
           </item>
          </layout>
         </item>
         <item>
          <layout class="QHBoxLayout" name="horizontalLayout_2">
           <property name="leftMargin">
            <number>0</number>
           </property>
           <item>
            <widget class="QLabel" name="xLabel">
             <property name="text">
              <string>Начальная точка x0 = </string>
             </property>
            </widget>
           </item>
           <item>
            <widget class="QLineEdit" name="xEdit">
             <property name="text">
              <string/>
             </property>
             <property name="placeholderText">
              <string>Формат: 0,0,...,0. Размерность должна соответствовать
количеству аргументов целевой функции f.</string>
             </property>
            </widget>
           </item>
          </layout>
         </item>
        </layout>
       </widget>
      </item>
      <item>
       <spacer name="verticalSpacer">
        <property name="orientation">
         <enum>Qt::Vertical</enum>
        </property>
        <property name="sizeType">
         <enum>QSizePolicy::Fixed</enum>
        </property>
        <property name="sizeHint" stdset="0">
         <size>
          <width>0</width>
          <height>10</height>
         </size>
        </property>
       </spacer>
      </item>
      <item>
       <widget class="QGroupBox" name="methodGroupBox">
        <property name="title">
         <string>Оптимизация</string>
        </property>
        <layout class="QVBoxLayout" name="verticalLayout_3">
         <item>
          <layout class="QGridLayout" name="gridLayout_2">
           <property name="horizontalSpacing">
            <number>48</number>
           </property>
           <item row="1" column="0">
            <layout class="QHBoxLayout" name="horizontalLayout_3">
             <item>
              <widget class="QLabel" name="label_3">
               <property name="text">
```

```
                <string>Метод оптимизации:</string>
               </property>
              </widget>
             </item>
             <item>
              <spacer name="horizontalSpacer_4">
               <property name="orientation">
                <enum>Qt::Horizontal</enum>
               </property>
               <property name="sizeType">
                <enum>QSizePolicy::Minimum</enum>
               </property>
               <property name="sizeHint" stdset="0">
                <size>
                 <width>20</width>
                 <height>0</height>
                </size>
               </property>
              </spacer>
             </item>
             <item>
              <widget class="QComboBox" name="methodComboBox">
               <item>
                <property name="text">
                 <string>Метод наискорейшего градиентного спуска</string>
                </property>
               </item>
               <item>
                <property name="text">
                 <string>Метод Ньютона</string>
                </property>
               </item>
               <item>
                <property name="text">
                 <string>Метод сопряженных градиентов</string>
                </property>
               </item>
              </widget>
             </item>
            </layout>
           </item>
           <item row="2" column="1">
            <layout class="QHBoxLayout" name="horizontalLayout_6">
             <item>
              <widget class="QLabel" name="label_5">
               <property name="text">
                <string>Точность:</string>
               </property>
              </widget>
             </item>
             <item>
              <spacer name="horizontalSpacer_3">
               <property name="orientation">
                <enum>Qt::Horizontal</enum>
               </property>
               <property name="sizeType">
                <enum>QSizePolicy::Minimum</enum>
               </property>
               <property name="sizeHint" stdset="0">
                <size>
                 <width>20</width>
                 <height>0</height>
                </size>
```

```
          </property>
         </spacer>
        </item>
        <item>
         <widget class="QDoubleSpinBox" name="epsilonSpinBox">
          <property name="minimumSize">
           <size>
            <width>100</width>
            <height>0</height>
           </size>
          </property>
          <property name="maximumSize">
           <size>
            <width>100</width>
            <height>16777215</height>
           </size>
          </property>
          <property name="decimals">
           <number>6</number>
          </property>
          <property name="maximum">
           <double>100.000000000000000</double>
          </property>
          <property name="singleStep">
           <double>0.001000000000000</double>
          </property>
          <property name="stepType">
           <enum>QAbstractSpinBox::AdaptiveDecimalStepType</enum>
          </property>
          <property name="value">
           <double>0.001000000000000</double>
          </property>
         </widget>
        </item>
       </layout>
      </item>
      <item row="1" column="1">
       <layout class="QHBoxLayout" name="horizontalLayout_4">
        <item>
         <widget class="QLabel" name="label_4">
          <property name="text">
           <string>Максимальное количество итераций:</string>
          </property>
         </widget>
        </item>
        <item>
         <spacer name="horizontalSpacer_2">
          <property name="orientation">
           <enum>Qt::Horizontal</enum>
          </property>
          <property name="sizeType">
           <enum>QSizePolicy::Minimum</enum>
          </property>
          <property name="sizeHint" stdset="0">
           <size>
            <width>20</width>
            <height>0</height>
           </size>
          </property>
         </spacer>
        </item>
        <item>
         <widget class="QSpinBox" name="iterationSpinBox">
```

```
            <property name="minimumSize">
             <size>
              <width>100</width>
              <height>0</height>
             </size>
            </property>
            <property name="maximumSize">
             <size>
              <width>100</width>
              <height>16777215</height>
             </size>
            </property>
            <property name="maximum">
             <number>999999999</number>
            </property>
            <property name="singleStep">
             <number>10</number>
            </property>
            <property name="stepType">
             <enum>QAbstractSpinBox::AdaptiveDecimalStepType</enum>
            </property>
            <property name="value">
             <number>100</number>
            </property>
           </widget>
          </item>
         </layout>
        </item>
       </layout>
      </item>
      <item>
       <layout class="QHBoxLayout" name="horizontalLayout_5">
        <item>
         <widget class="QPushButton" name="startButton">
          <property name="maximumSize">
           <size>
            <width>200</width>
            <height>16777215</height>
           </size>
          </property>
          <property name="text">
           <string>Найти оптимальное значение</string>
          </property>
         </widget>
        </item>
        <item>
         <spacer name="horizontalSpacer">
          <property name="orientation">
           <enum>Qt::Horizontal</enum>
          </property>
          <property name="sizeHint" stdset="0">
           <size>
            <width>40</width>
            <height>20</height>
           </size>
          </property>
         </spacer>
        </item>
       </layout>
      </item>
      <item>
       <spacer name="verticalSpacer_2">
        <property name="orientation">
```

```
            <enum>Qt::Vertical</enum>
           </property>
           <property name="sizeType">
            <enum>QSizePolicy::Fixed</enum>
           </property>
           <property name="sizeHint" stdset="0">
            <size>
             <width>0</width>
             <height>10</height>
            </size>
           </property>
          </spacer>
         </item>
        </layout>
       </widget>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QVBoxLayout" name="bottomLayout">
      <property name="spacing">
       <number>0</number>
      </property>
      <item>
       <widget class="QTabWidget" name="tabWidget">
        <property name="currentIndex">
         <number>0</number>
        </property>
        <widget class="QWidget" name="resultsTab">
         <attribute name="title">
          <string>Результаты</string>
         </attribute>
         <layout class="QVBoxLayout" name="verticalLayout_4">
          <property name="spacing">
           <number>0</number>
          </property>
          <property name="leftMargin">
           <number>0</number>
          </property>
          <property name="topMargin">
           <number>0</number>
          </property>
          <property name="rightMargin">
           <number>0</number>
          </property>
          <property name="bottomMargin">
           <number>0</number>
          </property>
          <item>
           <widget class="QTextBrowser" name="resultsTextBrowser"/>
          </item>
         </layout>
        </widget>
        <widget class="QWidget" name="stepsTab">
         <attribute name="title">
          <string>Шаги алгоритма</string>
         </attribute>
         <layout class="QVBoxLayout" name="verticalLayout_5">
          <property name="spacing">
           <number>0</number>
          </property>
          <property name="leftMargin">
           <number>0</number>
```

```
          </property>
          <property name="topMargin">
           <number>0</number>
          </property>
          <property name="rightMargin">
           <number>0</number>
          </property>
          <property name="bottomMargin">
           <number>0</number>
          </property>
          <item>
           <widget class="QTextBrowser" name="stepsTextBrowser"/>
          </item>
         </layout>
        </widget>
       </widget>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Листинг 6. Файл tests/TestConjugateGradientsMethod.py

```
import unittest
import numpy as np
from numpy.linalg import norm
from optimization_methods.common.methods import ConjugateGradientsMethod
from optimization_methods.common.function import Function


class TestConjugateGradientsMethod(unittest.TestCase):
    def test_simple_1(self):
        f = Function('(x1 + 2) ** 2 + (x1 - 3) ** 2 + (x2 + 3) ** 2')
        cgm = ConjugateGradientsMethod(0.00001, 10000)
        x0 = np.array([4, -3])
        x, k = cgm.optimize(f, x0)
        eps = 0.0002
        test = np.array([0.5, -3])
        self.assertTrue((norm(x - test) < eps).all())

    def test_simple_2(self):
        f = Function("(x1 + 2) ** 2 + (x1 - 3) ** 2 + (x2 + 3) ** 2")
        cgm = ConjugateGradientsMethod(0.00001, 10000)
        x0 = np.array([-9, -5])
        x, k = cgm.optimize(f, x0)
        eps = 0.002
        test = np.array([0.5, -3])
        self.assertTrue((norm(x - test) < eps).all())

    def test_simple_3(self):
        f = Function("(x1 + 2) ** 2 + (x1 - 3) ** 2 + (x2 + 3) ** 2")
        cgm = ConjugateGradientsMethod(0.00001, 10000)
        x0 = np.array([16, 10])
        x, k = cgm.optimize(f, x0)
        eps = 0.003
        test = np.array([0.5, -3])
```

```
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_1(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        cgm = ConjugateGradientsMethod(0.01, 10000)
        x0 = np.array([25, 25])
        x, k = cgm.optimize(f, x0)
        eps = 0.01
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_2(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        cgm = ConjugateGradientsMethod(0.001, 10000)
        x0 = np.array([25, 25])
        x, k = cgm.optimize(f, x0)
        eps = 0.003
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_3(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        cgm = ConjugateGradientsMethod(0.0001, 10000)
        x0 = np.array([25, 25])
        x, k = cgm.optimize(f, x0)
        eps = 0.003
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_hessian_positive(self):
        f = Function('(x1 + 6) ** 2 + (x2 - 1) ** 2 + 5 * x1')
        cgm = ConjugateGradientsMethod(0.00001, 10000)
        x0 = np.array([100000, -100000])
        x, k = cgm.optimize(f, x0)
        eps = 0.00001
        test = np.array([-8.5, 1])
        self.assertTrue((norm(x - test) < eps).all() and k <= len(x0))

    def test_already_min(self):
        f = Function('(x1 - 3) ** 2 + (x2 + 7) ** 2 - 3* x1')
        gdm = ConjugateGradientsMethod(0.00001, 10000)
        x0 = np.array([4.5, -7])
        x, k = gdm.optimize(f, x0)
        eps = 0.00001
        test = np.array([4.5, -7])
        self.assertTrue((norm(x - test) < eps).all())

    def test_far_min(self):
        f = Function('(x1 + x2) ** 2 + (x2 - 6) ** 2 + 9 * x1')
        cgm = ConjugateGradientsMethod(0.00001, 10000)
        x0 = np.array([1000, -1000])
        x, k = cgm.optimize(f, x0)
        eps = 0.001
        test = np.array([-14.995, 10.5])
        self.assertTrue((norm(x - test) < eps).all())
```

Листинг 7. Файл tests/TestGradientDescentMethod.py

```
import unittest
import numpy as np
from numpy.linalg import norm
```

```
from optimization_methods.common.methods import GradientDescentMethod
from optimization_methods.common.function import Function


class TestGradientDescentMethod(unittest.TestCase):
    def test_simple_1(self):
        f = Function('(x1 - 10) ** 2 + (x2 + 5) ** 2 + x1*x2')
        gdm = GradientDescentMethod(0.00001, 10000)
        x0 = np.array([1, 1])
        x, k = gdm.optimize(f, x0)
        eps = 0.0001
        test = np.array([16.66666666, -13.33333333])
        self.assertTrue((norm(x - test) < eps).all())

    def test_simple_2(self):
        f = Function('(x1 - 10) ** 2 + (x2 + 5) ** 2 + x1*x2')
        gdm = GradientDescentMethod(0.00001, 10000)
        x0 = np.array([-8, 19])
        x, k = gdm.optimize(f, x0)
        eps = 0.0001
        test = np.array([16.66666666, -13.33333333])
        self.assertTrue((norm(x - test) < eps).all())

    def test_simple_3(self):
        f = Function('(x1 - 10) ** 2 + (x2 + 5) ** 2 + x1*x2')
        gdm = GradientDescentMethod(0.00001, 10000)
        x0 = np.array([13, 21])
        x, k = gdm.optimize(f, x0)
        eps = 0.001
        test = np.array([16.66666666, -13.33333333])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_1(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        gdm = GradientDescentMethod(0.01, 10000)
        x0 = np.array([15, -10])
        x, k = gdm.optimize(f, x0)
        eps = 0.02
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_2(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        gdm = GradientDescentMethod(0.001, 10000)
        x0 = np.array([15, -10])
        x, k = gdm.optimize(f, x0)
        eps = 0.002
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_3(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        gdm = GradientDescentMethod(0.0001, 10000)
        x0 = np.array([15, -10])
        x, k = gdm.optimize(f, x0)
        eps = 0.001
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_already_min(self):
        f = Function('(x1) ** 2 + (x2) ** 2 + (x3) ** 2')
        gdm = GradientDescentMethod(0.00001, 10000)
        x0 = np.array([0, 0, 0])
```

```
        x, k = gdm.optimize(f, x0)
        eps = 0.00001
        test = np.array([0, 0, 0])
        self.assertTrue((norm(x - test) < eps).all())

    def test_far_min(self):
        f = Function('(x1 + 2) ** 2 + (x2) ** 2 - 2 * x1')
        gdm = GradientDescentMethod(0.00001, 10000)
        x0 = np.array([1000, 1000])
        x, k = gdm.optimize(f, x0)
        eps = 0.00001
        test = np.array([-1, 0])
        self.assertTrue((norm(x - test) < eps).all())

    def test_local_min(self):
        f = lambda x: x[0] * x[1] * np.sin(x[1])
        gdm = GradientDescentMethod(0.00001, 10000)
        x0 = np.array([0, 0])
        x, k = gdm.optimize(f, x0)
        eps = 0.00001
        test = np.array([0, 0])
        self.assertTrue((norm(x - test) < eps).all())
```

Листинг 8. Файл tests/TestNewtonMethod.py

```
import unittest
import numpy as np
from numpy.linalg import norm
from optimization_methods.common.methods import NewtonMethod
from optimization_methods.common.function import Function


class TestNewtonMethod(unittest.TestCase):
    def test_simple_1(self):
        f = Function("(x1 + 2) ** 2 + (x2 - 3) ** 2 + (x2 + 3) ** 2 - 3 * x1")
        nm = NewtonMethod(0.00001, 10000)
        x0 = np.array([-5, -10])
        x, k = nm.optimize(f, x0)
        eps = 0.00001
        test = np.array([-0.5, 0])
        self.assertTrue((norm(x - test) < eps).all())

    def test_simple_2(self):
        f = Function("(x1 + 2) ** 2 + (x2 - 3) ** 2 + (x2 + 3) ** 2 - 3 * x1")
        nm = NewtonMethod(0.00001, 10000)
        x0 = np.array([20, -17])
        x, k = nm.optimize(f, x0)
        eps = 0.00001
        test = np.array([-0.5, 0])
        self.assertTrue((norm(x - test) < eps).all())

    def test_simple_3(self):
        f = Function("(x1 + 2) ** 2 + (x2 - 3) ** 2 + (x2 + 3) ** 2 - 3 * x1")
        nm = NewtonMethod(0.00001, 10000)
        x0 = np.array([43, 19])
        x, k = nm.optimize(f, x0)
        eps = 0.00001
        test = np.array([-0.5, 0])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_1(self):
```

```
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        nm = NewtonMethod(0.01, 10000)
        x0 = np.array([-30, 24])
        x, k = nm.optimize(f, x0)
        eps = 0.02
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_2(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        nm = NewtonMethod(0.001, 10000)
        x0 = np.array([-30, 24])
        x, k = nm.optimize(f, x0)
        eps = 0.001
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_different_eps_3(self):
        f = Function("(x1 + x2) ** 2 + (x1 + 1) ** 2 + (x2 - 2) ** 2")
        nm = NewtonMethod(0.0001, 10000)
        x0 = np.array([-30, 24])
        x, k = nm.optimize(f, x0)
        eps = 0.0001
        test = np.array([-1.33333333, 1.66666666])
        self.assertTrue((norm(x - test) < eps).all())

    def test_already_min(self):
        f = Function('(x1 - 3) ** 2 + (x2 + 7) ** 2 - 3 * x1')
        nm = NewtonMethod(0.00001, 10000)
        x0 = np.array([4.5, -7])
        x, k = nm.optimize(f, x0)
        eps = 0.00001
        test = np.array([4.5, -7])
        self.assertTrue((norm(x - test) < eps).all())

    def test_far_min(self):
        f = Function('(x1 + 6) ** 2 + (x2 - 1) ** 2 + 5 * x1')
        nm = NewtonMethod(0.00001, 10000)
        x0 = np.array([1000, 1000])
        x, k = nm.optimize(f, x0)
        eps = 0.00001
        test = np.array([-8.5, 1])
        self.assertTrue((norm(x - test) < eps).all())
```

Листинг 9. Файл tests/__main__.py

```
import unittest
from TestGradientDescentMethod import TestGradientDescentMethod
from TestNewtonMethod import TestNewtonMethod
from TestConjugateGradientsMethod import TestConjugateGradientsMethod


if __name__ == "__main__":
    unittest.main()
```