

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА** - **Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ**

**по дисциплине**

**«Математическое обеспечение систем поддержки принятия решений»**

Студент группы: ИКБО-14-20 Вежновец Ф.Ю.\_\_ *(Ф.И.О.студента)*

Руководитель \_\_Семенов Р.Э.\_\_

*(Ф.И.О. преподавателя)*

Москва 2023

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc132320263)

[ПРАКТИЧЕСКАЯ РАБОТА №1. МЕТОД НУЛЕВОГО ПОРЯДКА: СИМПЛЕКС МЕТОД 5](#_Toc132320264)

[1.1 Постановка задачи 5](#_Toc132320265)

[1.2 Описание симплекс метода 5](#_Toc132320266)

[1.3 Ручной расчёт итераций симплекса 6](#_Toc132320267)

[1.4 Программная реализация 8](#_Toc132320268)

[1.5 Вывод 13](#_Toc132320269)

[ПРАКТИЧЕСКАЯ РАБОТА №2. МЕТОД НУЛЕВОГО ПОРЯДКА: МЕТОД НЕЛДЕРА-МИДА 14](#_Toc132320270)

[2.1 Постановка задачи 14](#_Toc132320271)

[2.2 Описание метода Нелдера-Мида 14](#_Toc132320272)

[2.3 Ручной расчёт итераций Нелдера-Мида 16](#_Toc132320273)

[2.4 Программная реализация 18](#_Toc132320274)

[2.5 Вывод 22](#_Toc132320275)

[ПРАКТИЧЕСКАЯ РАБОТА №3. МЕТОД ПЕРВОГО ПОРЯДКА: МЕТОД ГРАДИЕНТНОГО СПУСКА П ПОСТОЯННЫМ ШАГОМ 23](#_Toc132320276)

[3.1 Постановка задачи 23](#_Toc132320277)

[3.2 Описание метода градиентного спуска с постоянным шагом 23](#_Toc132320278)

[3.3 Ручной расчёт итераций методом градиентного спуска с постоянным шагом 25](#_Toc132320279)

[3.4 Программная реализация 27](#_Toc132320280)

[3.5 Вывод 31](#_Toc132320281)

[ПРАКТИЧЕСКАЯ РАБОТА №4. МЕТОД ПЕРВОГО ПОРЯДКА: МЕТОД НАИСКОРЕЙШЕГО ГРАДИЕНТНОГО СПУСКА 32](#_Toc132320282)

[4.1 Постановка задачи 32](#_Toc132320283)

[4.2 Описание метода наискорейшего градиентного спуска 32](#_Toc132320284)

[4.3 Ручной расчёт итераций методом наискорейшего градиентного спуска 33](#_Toc132320285)

[4.4 Программная реализация 34](#_Toc132320286)

[4.5 Вывод 38](#_Toc132320287)

[ПРАКТИЧЕСКАЯ РАБОТА №5. МЕТОД ВТОРОГО ПОРЯДКА: МЕТОД НЬЮТОНА 39](#_Toc132320288)

[5.1 Постановка задачи 39](#_Toc132320289)

[5.2 Описание метода наискорейшего градиентного спуска 39](#_Toc132320290)

[5.3 Ручной расчёт итераций методом наискорейшего градиентного спуска 40](#_Toc132320291)

[5.4 Программная реализация 42](#_Toc132320292)

[5.5 Вывод 45](#_Toc132320293)

[ЗАКЛЮЧЕНИЕ 46](#_Toc132320294)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47](#_Toc132320295)

[ПРИЛОЖЕНИЯ 48](#_Toc132320296)

ВВЕДЕНИЕ

Любое исследование основывается на применении научного метода, предоставляет научную информацию и теории для объяснения природы, и свойств окружающего мира. При это не существует неприступных границ – и подходы, и методы могут переплетаться в едином исследовании. При этом научные подходы нацеливают на научные разработки и открытия, а методы помогают совершать их.

Только благодаря надежным методам возникают новые плодотворные теории, а наша практическая деятельность становится более эффективной. Хороший специалист, вооруженный научной методологией, не только знает правильные ответы на сложные вчерашние вопросы и умеет самостоятельно находить грамотные ответы на сегодняшние, но и не растеряется перед непростыми завтрашними. Рассмотренные алгоритмы широко применяются в различных прикладных задачах при нахождении минимумов функций. Например, метод наискорейшего спуска лежит в основе обучения искусственных нейронных сетей.

ПРАКТИЧЕСКАЯ РАБОТА №1. МЕТОД НУЛЕВОГО ПОРЯДКА: СИМПЛЕКС МЕТОД

1.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями реализовать симплексный метод.

Найти минимум целевой функции.

Симплексным методом с точностью

Размерность задачи n = 2.

Зададим начальную точку симплекса и длину ребра симплекса m = 0,07.

1.2 Описание симплекс метода

Регулярным симплексом в n-мерном пространстве называется правильный многогранник с n+1вершиной. При n = 2 симплексом является правильный треугольник, при n = 3 – тетраэдр и т.д. Отрезок, соединяющий 2 вершины симплекса, называется ребром симплекса.

Поиск симплексным методом ведется по следующей схеме. Устанавливаются координаты вершин симплексов. Определяется вершина с наибольшим значением целевой функции. Вместо нее сроиться новая вершина отражением старой через центр тяжести остальных вершин симплекса. На рисунке 1.1 представлен процесс построения нового симплекса на плоскости.



Рисунок 1.1 – Построение нового симплекса

Если попытка отражения не приводит к уменьшению целевой функции, то выполняется операция редукции, в результате которой формируется новый симплекс с уменьшенными вдвое сторонами. При операции редукции в качестве базовой точки выбирается вершина старого симплекса, в которой функция принимает минимальное значение.

В результате исключения вершин симплексов с наибольшим значением целевой функции процесс поиска сходится к минимальному значению.

Поиск завершается, когда разности между значениями функции в центре тяжести симплекса и вершинах становится достаточно малым.

1.3 Ручной расчёт итераций симплекса

Отобразим изначальную таблицу (Таблица 1.1).

Таблица 1.1 – Изначальные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |

Вычислим приращения (Формулы 1.2 и 1.3):

Используя и , вычислим координаты двух остальных вершин симплекса (Формулы 1.4 и 1.5):

Итерация k = 0. Вычислим значение целевой функции в вершинах , , :

Отобразим эти значения в таблицe 1.2.

Таблица 1.2 – Новые данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 | ,067 |  |  |

Наибольшее значение целевой функции соответствует вершине , поэтому необходимо отразить ее относительно центра тяжести остальных вершин и . центр тяжести расположен в точке

Используя свойство регулярности, найдем координаты отраженной вершины

В полученной вершине значение целевой функции 𝑓() = 1216,495

Таким образом, получим таблицу 1.3

Таблица 1.3 – Новые данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

Следовательно, наблюдается уменьшение целевой функции < . Новый симплекс образован вершинами , , , которым соответствует значение целевой функции , , .

Проверим условие окончания поиска . Определим центр тяжести симплекса

В полученной точке .

Вычислим , , .

Так как все условия окончания поиска не выполняются, то процесс итераций должен быть продолжен.

1.4 Программная реализация

Реализуем симплекс метод на языке высокого уровня Python.

На рисунках 1.2 – 1.6 представлен результат работы программы. Листинг кода приведен в Приложении А.

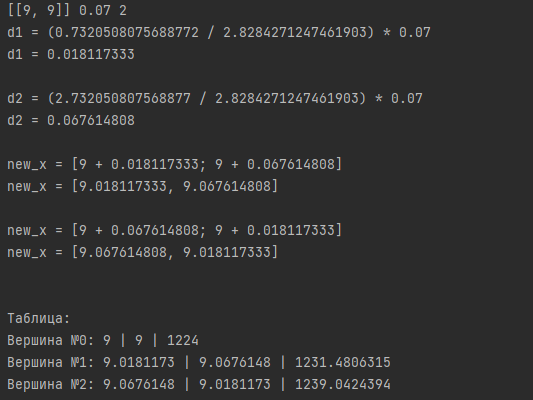


Рисунок 1.2 – Результат работы программы

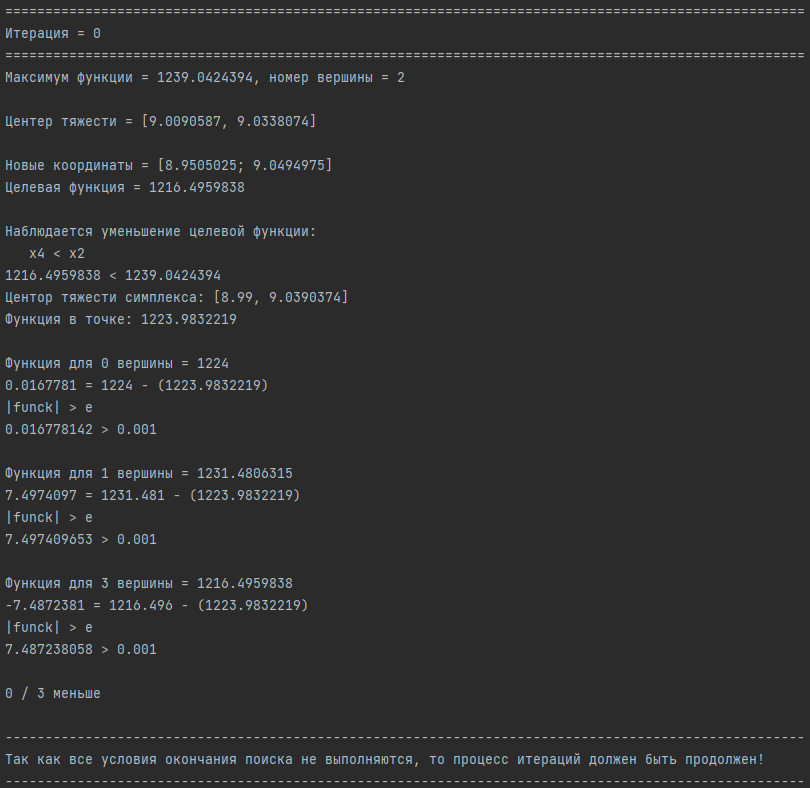


Рисунок 1.3 – Результат работы программы

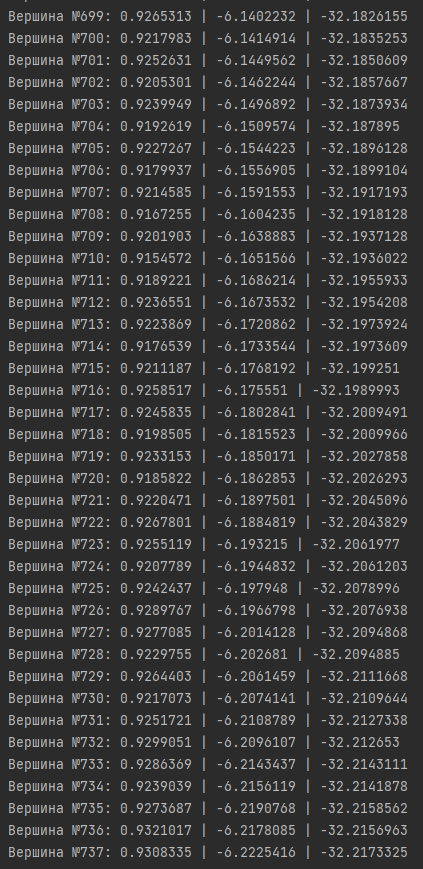


Рисунок 1.4 – Результат работы программы

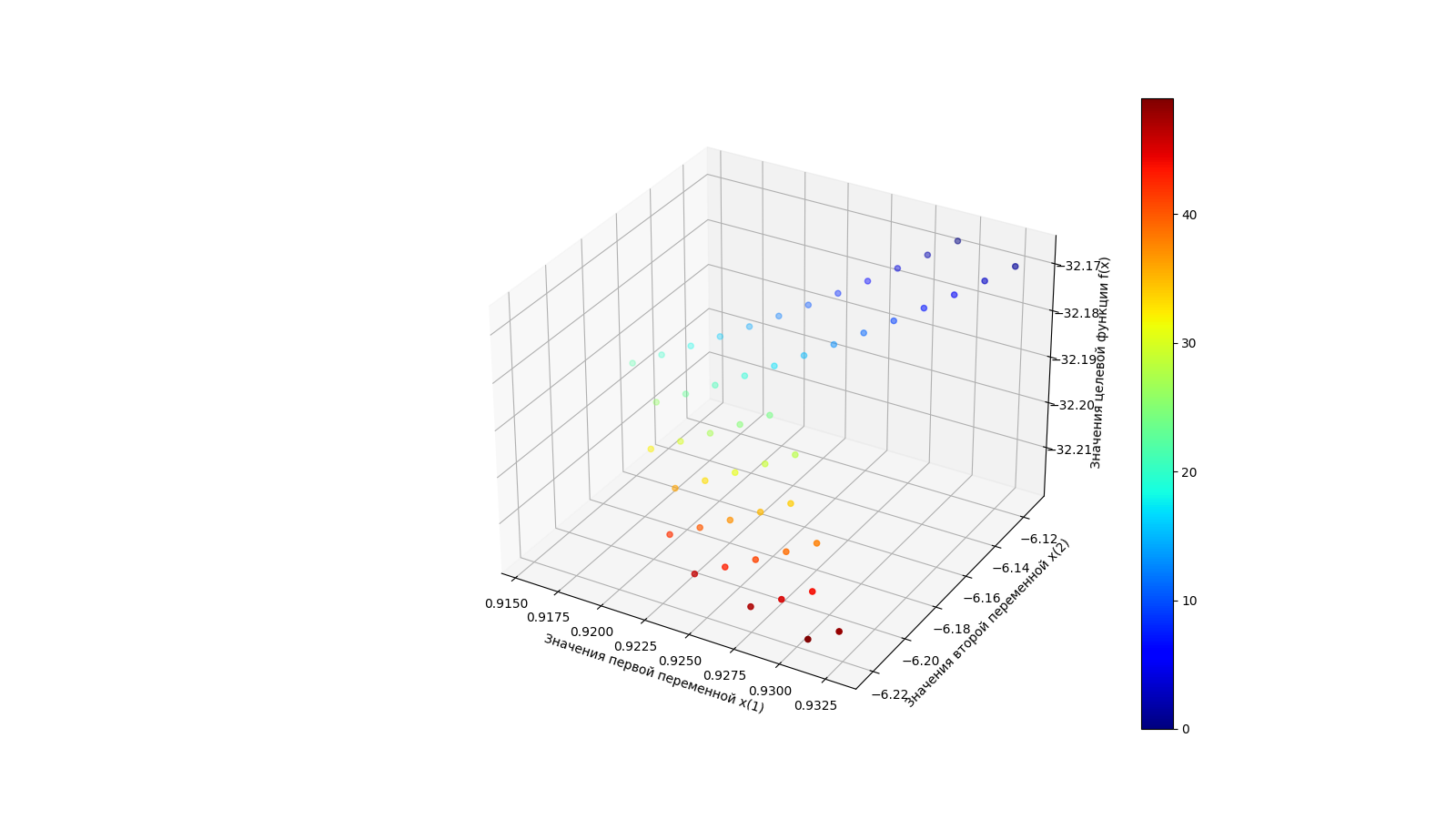


Рисунок 1.5 – Последние 50 элементов таблицы симплекса

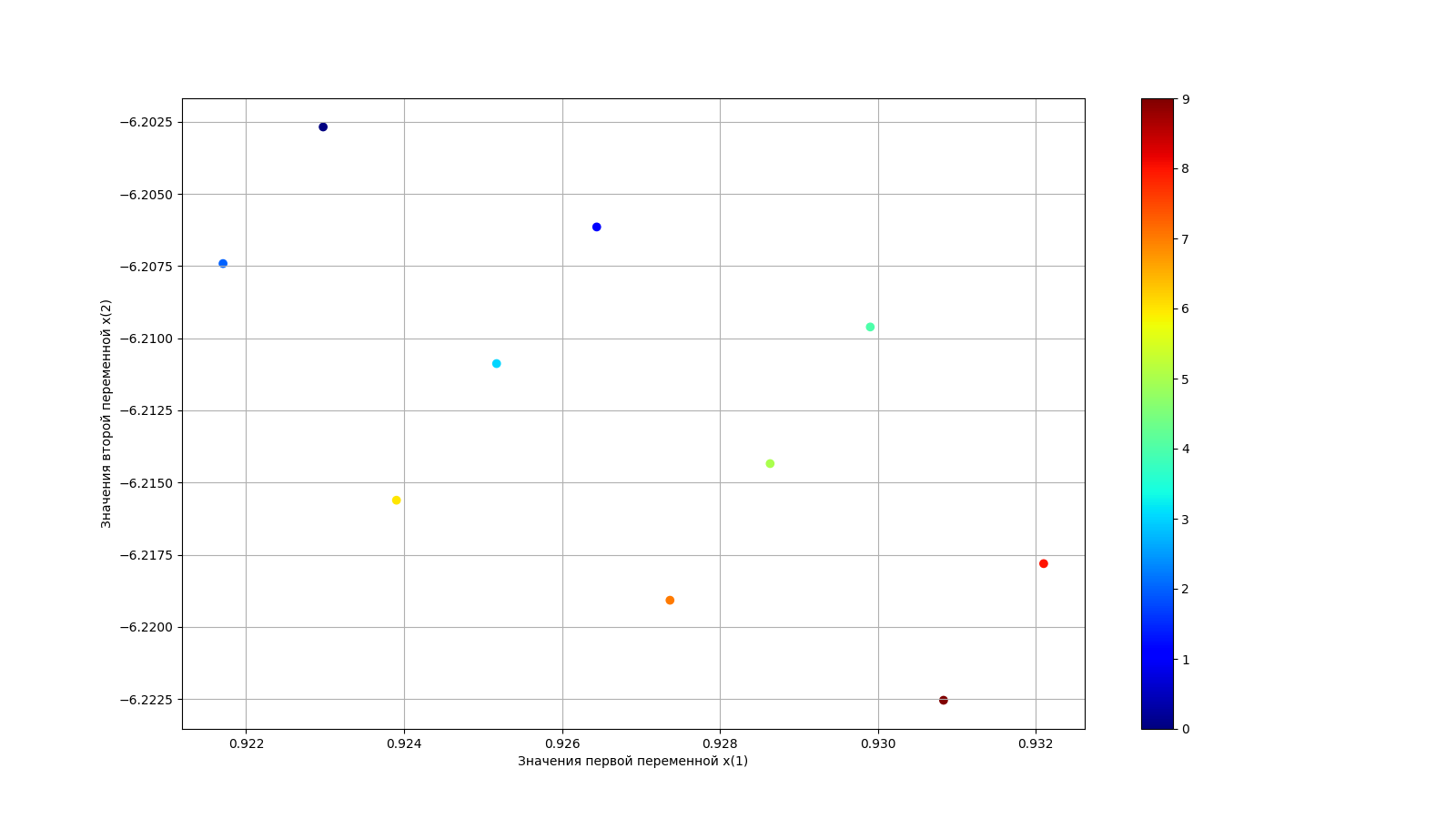


Рисунок 1.6 – Последние 5 элементов таблицы симплекса

Сравним полученные результаты при помощи сайта WolframAlpha (Рисунок 1.7).

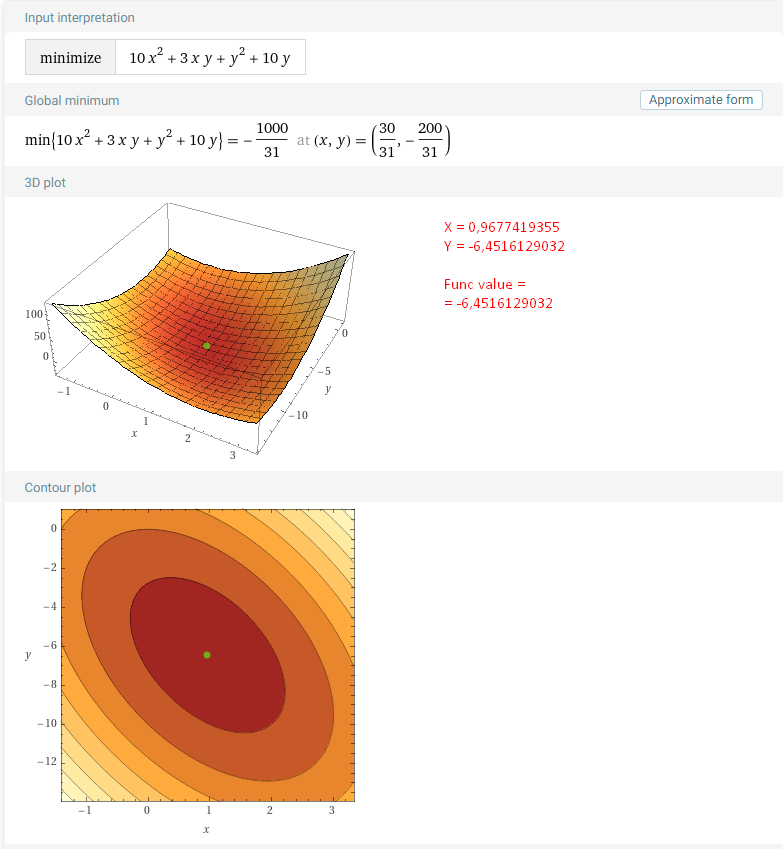


Рисунок 1.7 – Результаты минимизации функции при помощи WolframAlpha

1.5 Вывод

По результатам работы, был реализован симплекс метод (метод нулевого порядка).

ПРАКТИЧЕСКАЯ РАБОТА №2. МЕТОД НУЛЕВОГО ПОРЯДКА: МЕТОД НЕЛДЕРА-МИДА

2.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями.

Найти минимум целевой функции.

методом Нелдера-Мида с точностью

Размерность задачи .

Длину ребра симплекса

Параметр растяжения

Параметр сжатия

Зададим начальную точку симплекса .

2.2 Описание метода Нелдера-Мида

В 1964 году Нелдер и Мид предложили модификацию, в которой симплекс может изменять свою форму (растягиваясь и сжимаясь) в зависимости от свойств поверхности целевой функции. Так как в этом случае симплекс не будет уже регулярным, метод назвали поиском по деформируемому многограннику.

И так модифицируем рассмотренный на предыдущей лекции алгоритм минимизации целевой функции по регулярному симплексу, добавив к процедуре отражения при построении нового симплекса процедуры сжатия и растяжения. Геометрическая иллюстрация этих процедур для случая 𝑛=2 представлена на рис. 1, 2 и 3, где введены следующие обозначения:

– наибольшее значение целевой функции;

– следующее по величине за наибольшим значение целевой функции;

– наименьшее значение целевой функции;

– текущие значения целевой функции

1. Операция отображения (Рисунок 2.1)



Рисунок 2.1 – Операция отображения

1. Если то выполняется операция растяжения , где 𝛽 – параметр растяжения (Рисунок 2.2).



Рисунок 2.2 – Операция растяжения

1. Если , то выполняется операция сжатия , где – параметр сжатия (Рисунок 2.3).



Рисунок 2.3 – Операция сжатия

При решении практически задач минимизации параметры растяжения и сжатия 𝛾 Нелдер и Мид рекомендует брать , , но Правильнее – выбирать эти параметры из интервалов и .

2.3 Ручной расчёт итераций Нелдера-Мида

Отобразим изначальную таблицу (Таблица 2.1).

Таблица 2.1 – Изначальные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |

Вычислим приращения (Формулы 1.2 и 1.3):

Используя и , вычислим координаты двух остальных вершин симплекса (Формулы 1.4 и 1.5):

Итерация k = 0. Вычислим значение целевой функции в вершинах , , и обозначим наибольшее значение функции , следующее за наибольшим значением , наименьшее значение функции :

, , ;

Отобразим эти значения в таблицe 2.2.

Таблица 2.2 – Новые данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 | ,067 |  |  |

Наибольшее значение целевой функции соответствует вершине , поэтому необходимо отразить ее относительно центра тяжести остальных вершин и . центр тяжести расположен в точке

Используя свойство регулярности, найдем координаты отраженной вершины

В полученной вершине значение целевой функции ) = 1216,495

Таким образом, получим таблицу 2.3

Таблица 2.3 – Новые данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

Следовательно, наблюдается уменьшение целевой функции < .

Так как не выполняется условие то выполним операцию растяжения симплекса.

В полученной вершине значение целевой функции .

Условие растяжения выполнено

Проверим условие окончания поиска. Определим координаты центра тяжести симплекса

В полученной точке .

Вычислим 𝜎 (сигма)



Так как условие окончания поиска не выполняется, то процесс итерации должен быть продолжен.

2.4 Программная реализация

Реализуем симплекс метод на языке высокого уровня Python.

На рисунках 2.2 – 2.7 представлен результат работы программы. Листинг кода приведен в Приложении Б.

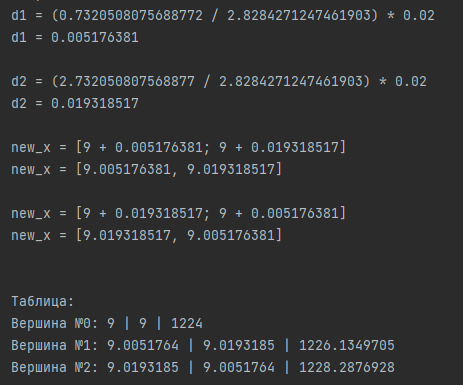


Рисунок 2.2 – Результат работы программы

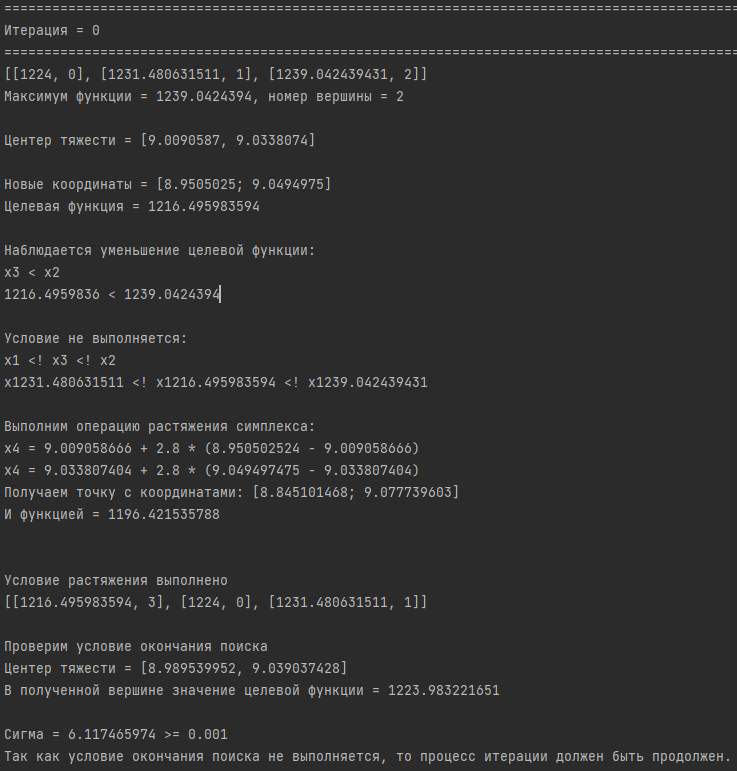


Рисунок 2.3 – Результат работы программы

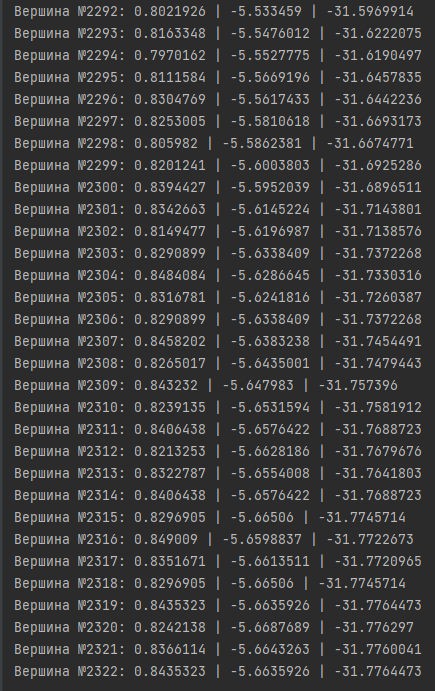


Рисунок 2.4 – Результат работы программы

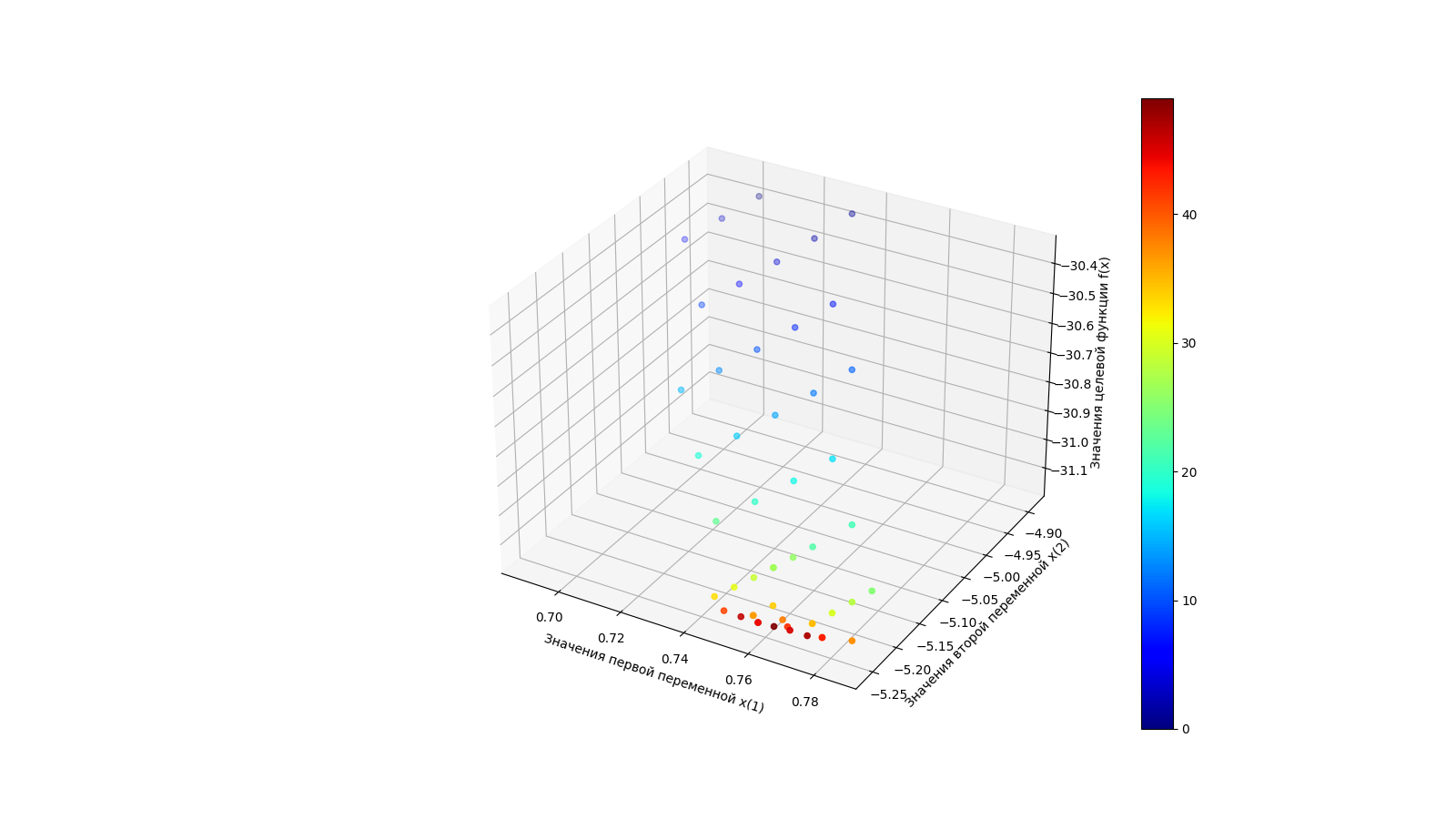


Рисунок 2.5 – Последние 50 элементов таблицы Нелдера Мида

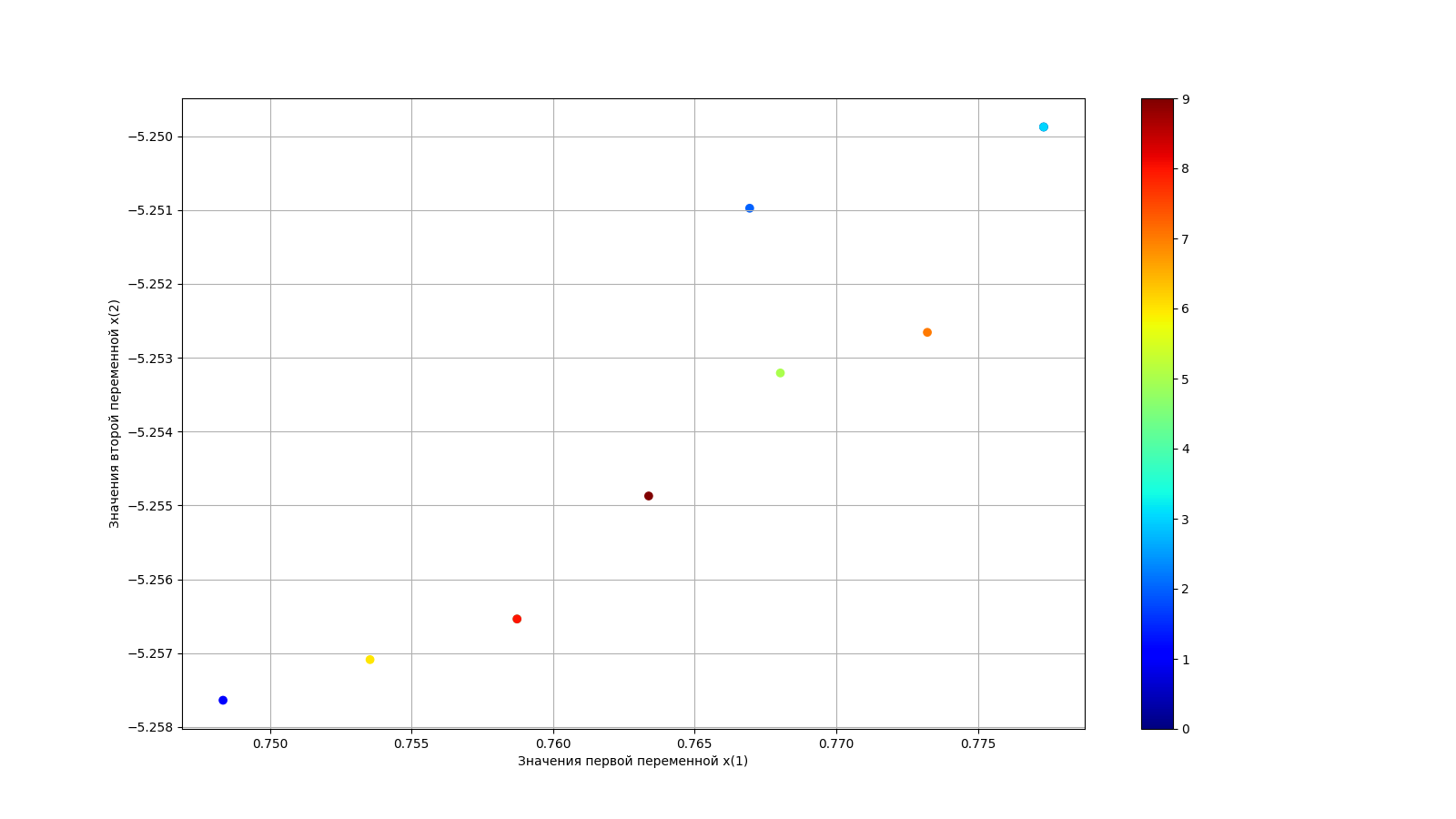


Рисунок 2.6 – Последние 5 элементов таблицы Нелдера Мида

Сравним полученные результаты при помощи сайта WolframAlpha.



Рисунок 2.7 – Результаты минимизации функции при помощи WolframAlpha

2.5 Вывод

По результатам работы, был реализован метод Нелдера Мида (метод нулевого порядка).

ПРАКТИЧЕСКАЯ РАБОТА №3. МЕТОД ПЕРВОГО ПОРЯДКА: МЕТОД ГРАДИЕНТНОГО СПУСКА П ПОСТОЯННЫМ ШАГОМ

3.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями.

Найти минимум целевой функции

методом градиентного спуска с постоянным шагом с точностью

Зададим начальную точку .

Начальную величину шага .

3.2 Описание метода градиентного спуска с постоянным шагом

Сущность метода градиентного спуска с постоянным шагом заключается в следующем. Выбирается начальная точка из области определения функции . Координаты новой точки вычисляются по формуле

где *k* − номер итерации *k* = 0, 1, …,

− величина шага,

– градиент функции в точке ,

Начальная величина шага задаётся пользователем. В каждом новой точке поиска проверяется условие убывания функции . Если условие нарушается, то постепенно уменьшается величина шага , т.е. точка приближается к точке до тех пор, пока условие не выполнится. В полученной точке определяется новое направление градиента и осуществляется новый спуск. Процесс продолжается пока не будет выполнено условие окончания поиска. В качестве условия окончания поиска используется близость к нулю нормы градиента .

Геометрическая иллюстрация поиска минимума целевой функции методом градиентного спуска с постоянным шагом для случая *n* = 2 представлена на рисунке 3.1.



Рисунок 3.1 - Графическая иллюстрация поиска точки минимума методом градиентного спуска с постоянным шагом

3.3 Ручной расчёт итераций методом градиентного спуска с постоянным шагом

Отобразим изначальную таблицу (Таблица 3.1).

Таблица 3.1 – Изначальные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |

Найдем градиент функции в произвольной точке

***Итерация k = 0.*** Вычислим значение целевой функции 𝑓(𝑥 (0)) и градиент

в начальной точке 𝑥 (0): 𝑓(𝑥 (0)) = 1224; ∇𝑓(𝑥 (0)) = (207, 55) 𝑇.

Определим координаты точки

и значение целевой функции в этой точке

Сравним и . Поскольку , то условие убывания функции не выполнено. Уменьшим величину шага и повторим вычисления координат точки .

Определим координаты точки

и значение целевой функции в этой точке

Сравним и . Поскольку , то условие убывания функции не выполнено. Уменьшим величину шага и повторим вычисления координат точки .

Определим координаты точки

и значение целевой функции в этой точке

Сравним и . Поскольку , то условие убывания функции не выполнено. Уменьшим величину шага и повторим вычисления координат точки .

Определим координаты точки

и значение целевой функции в этой точке

Сравним и . Поскольку , то условие убывания функции выполнено. Составим таблицу 3.2.

Таблица 3.2 – Новая таблица данных

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |
| 1 |  |  |  |

Проверим условие окончания поиска. Для этого вычислим вектор градиента в точке :

Найдем норму вектора градиента

Так как условие окончания поиска не выполняется, то процесс итерации должен быть продолжен.

3.4 Программная реализация

Реализуем симплекс метод на языке высокого уровня Python.

На рисунках 3.2 – 3.10 представлен результат работы программы. Листинг кода приведен в Приложении В.

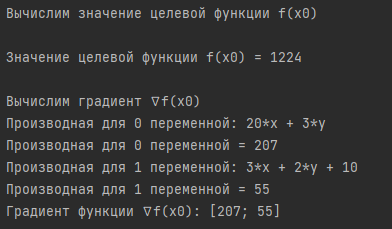


Рисунок 3.2 – Результат работы программы

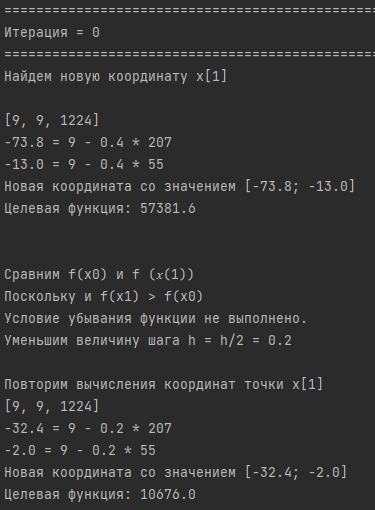


Рисунок 3.3 – Результат работы программы

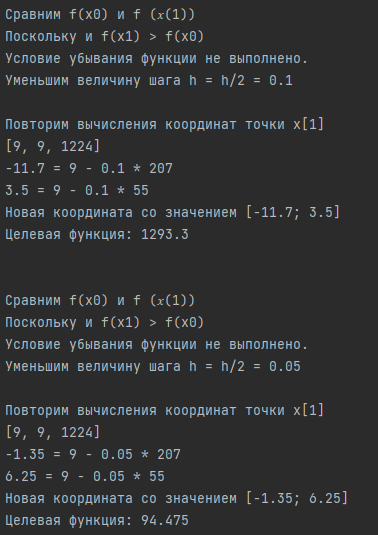


Рисунок 3.4 – Результат работы программы

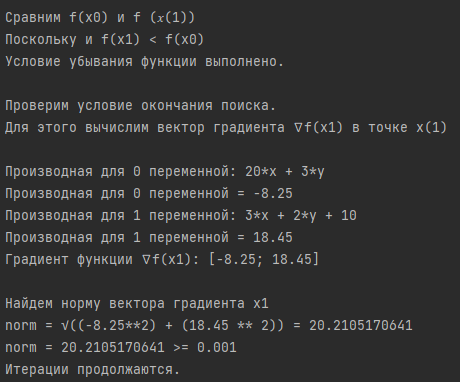


Рисунок 3.5 – Результат работы программы



Рисунок 3.6 – Результат работы программы

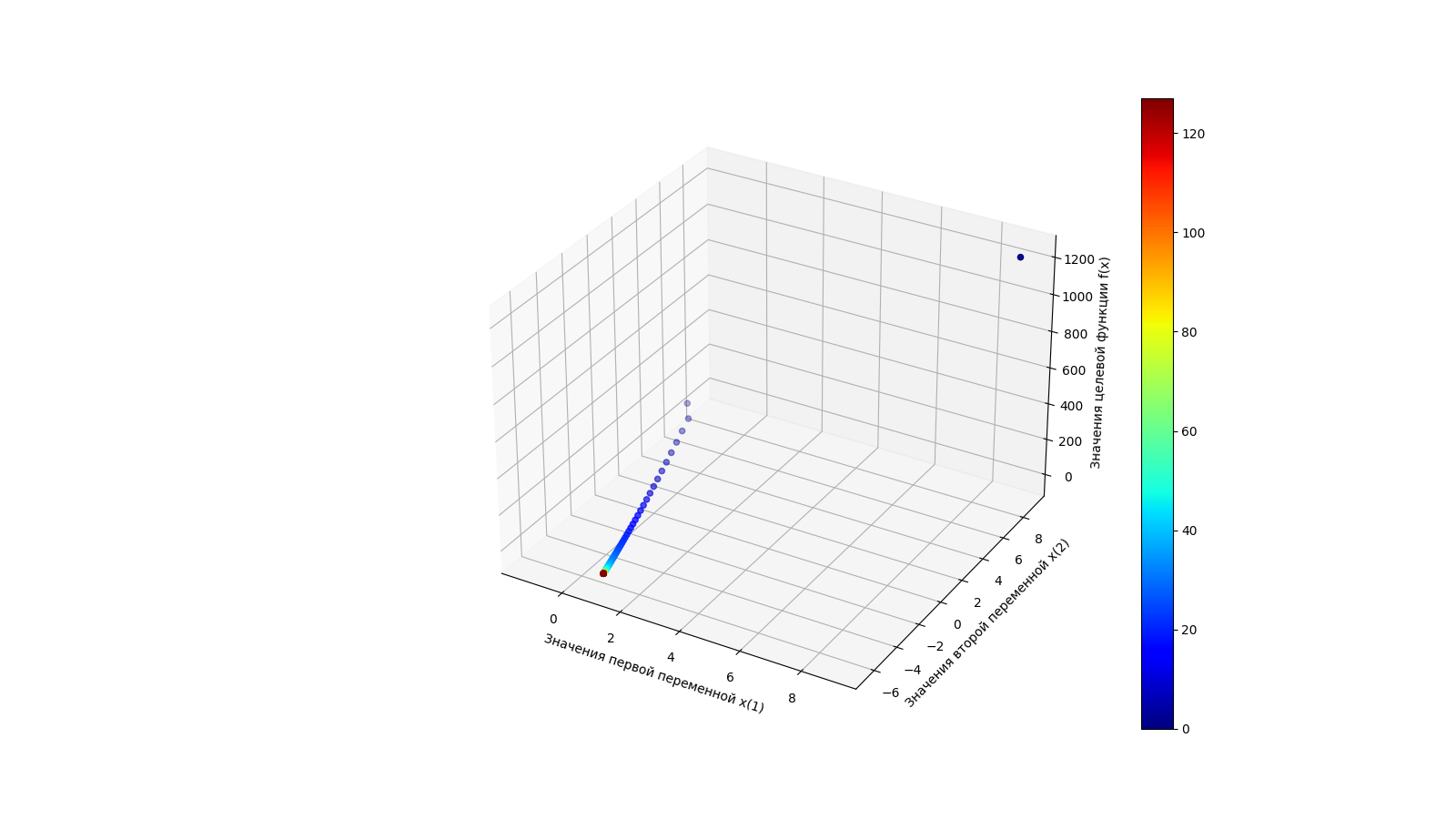


Рисунок 3.7 – Все элементов таблицы методом градиентного спуска с постоянным шагом

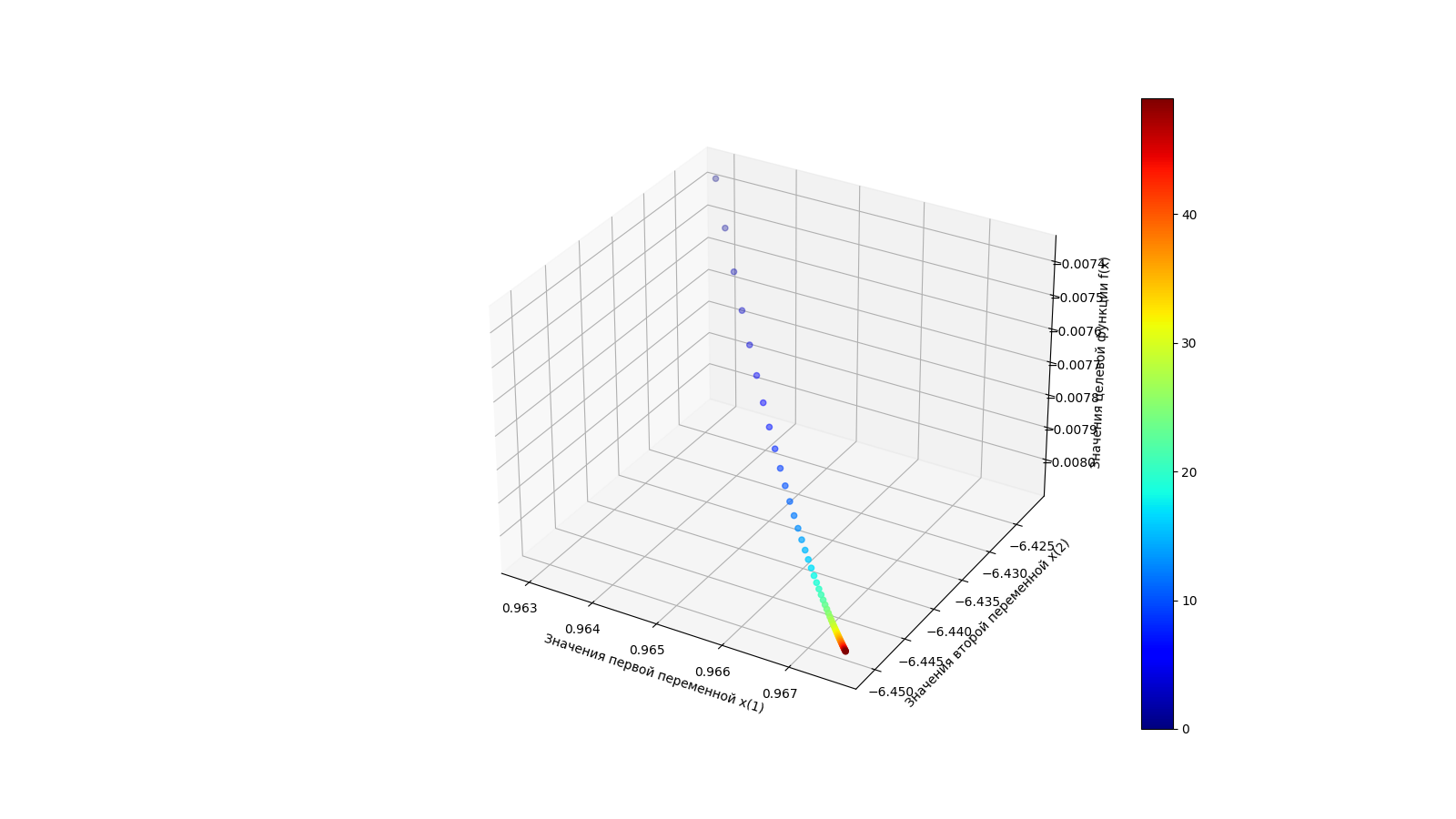


Рисунок 3.8 – Последние 50 элементов таблицы методом градиентного спуска с постоянным шагом

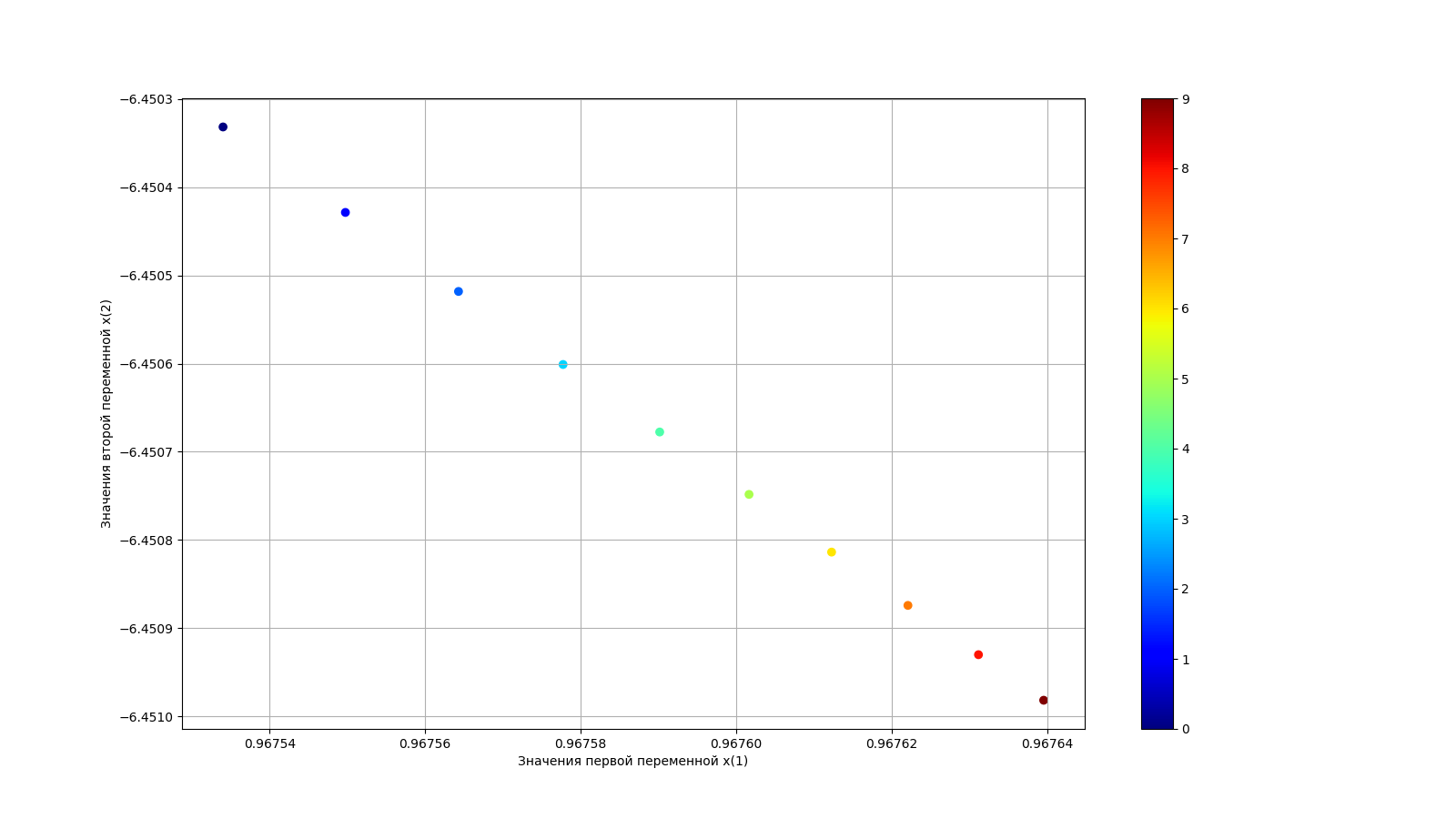


Рисунок 3.9 – Последние 5 элементов таблицы методом градиентного спуска с постоянным шагом

Сравним полученные результаты при помощи сайта WolframAlpha.



Рисунок 3.10 – Результаты минимизации функции при помощи WolframAlpha

3.5 Вывод

По результатам работы, был реализован методом градиентного спуска с постоянным шагом (метод первого порядка).

ПРАКТИЧЕСКАЯ РАБОТА №4. МЕТОД ПЕРВОГО ПОРЯДКА: МЕТОД НАИСКОРЕЙШЕГО ГРАДИЕНТНОГО СПУСКА

4.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями.

Найти минимум целевой функции

Методом наискорейшего градиентного спуска с точностью

Зададим начальную точку .

4.2 Описание метода наискорейшего градиентного спуска

Метод наискорейшего спуска отличается от метода градиентного спуска способом определения величины шага . Величина шага задается не произвольно, а выбирается так, чтобы на каждой итерации достигалось максимально возможное уменьшение целевой функции вдоль направления ее антиградиента , вычисленного в точке . Величина шага определяется из решения вспомогательной одномерной задачи минимизации

которая может быть решена аналитически или численно. При квадратичной интерполяции целевой функции величину шага можно определить по формуле

где – матрица Гессе, вычисленная в точке .

На рисунке 4.1 представлена траектория приближения к точке минимума методом наискорейшего спуска для случая n = 2. Здесь каждая последующая точка находится как точка касания антиградиента целевой функции и линии уровня.



Рисунок 4.1 - Траектория движения к точке минимума в методе наискорейшего спуска

4.3 Ручной расчёт итераций методом наискорейшего градиентного спуска

Отобразим изначальную таблицу (Таблица 4.1).

Таблица 4.1 – Изначальные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |

Найдем градиент функции в произвольной точке

***Итерация k = 0.*** Вычислим значение целевой функции 𝑓 (𝑥 (0)) и градиент

в начальной точке 𝑥 (0): 𝑓 (𝑥 (0)) = 1224; ∇𝑓 (𝑥 (0)) = (207, 55) 𝑇.

Определим координаты точки по формуле

Найдем значение из условия

Численным методом определим значения шага .

и значение целевой функции в этой точке

Проверим условие окончания процесса поиска. Для этого вычислим градиент целевой функции в точке .

Так как норма вектора градиента

то переходим к следующей итерации.

4.4 Программная реализация

Реализуем симплекс метод на языке высокого уровня Python.

На рисунках 4.2 – 4.7 представлен результат работы программы. Листинг кода приведен в Приложении Г.

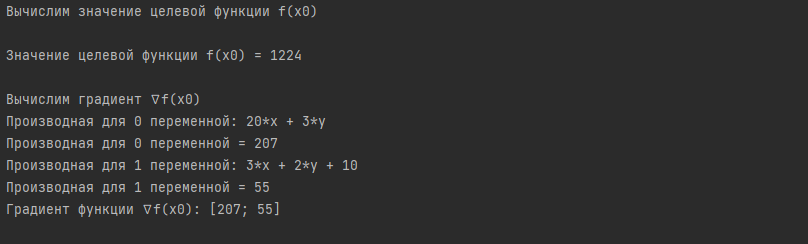


Рисунок 4.2 – Результат работы программы

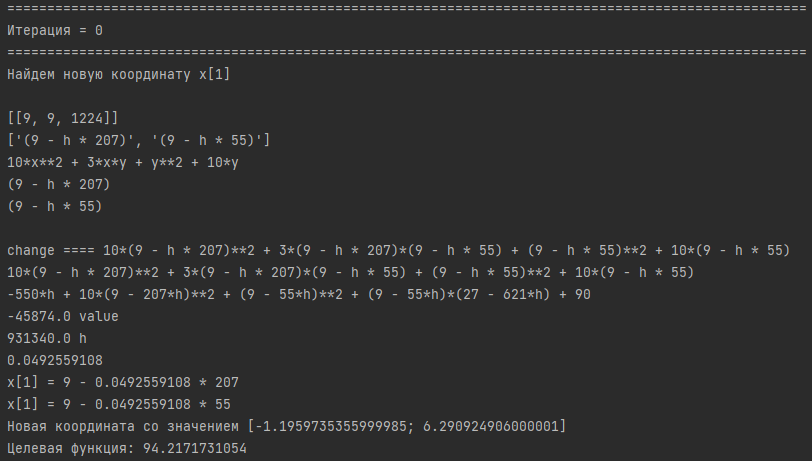


Рисунок 4.3 – Результат работы программы

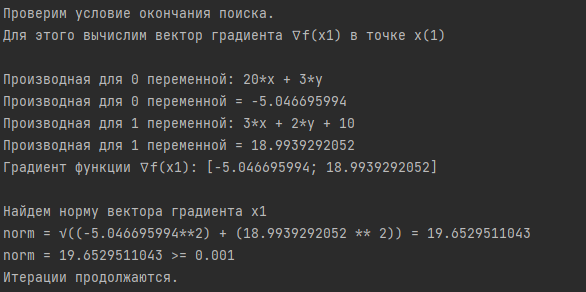


Рисунок 4.4 – Результат работы программы

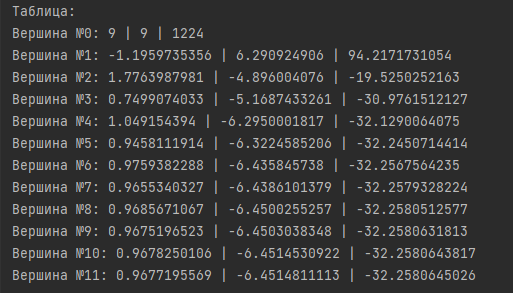


Рисунок 4.5 – Результат работы программы

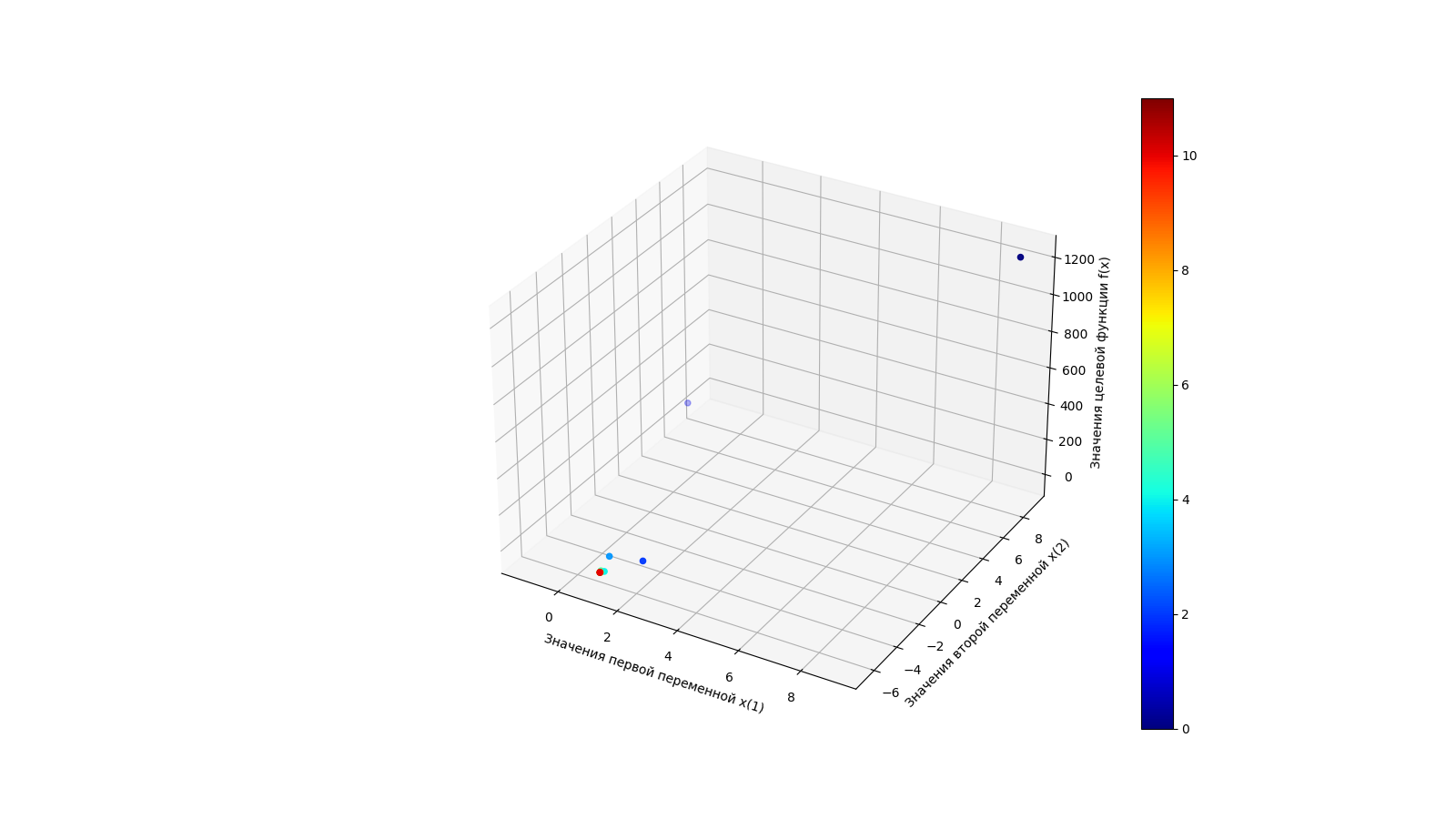


Рисунок 4.5 – Все элементов таблицы методом наискорейшего градиентного спуска

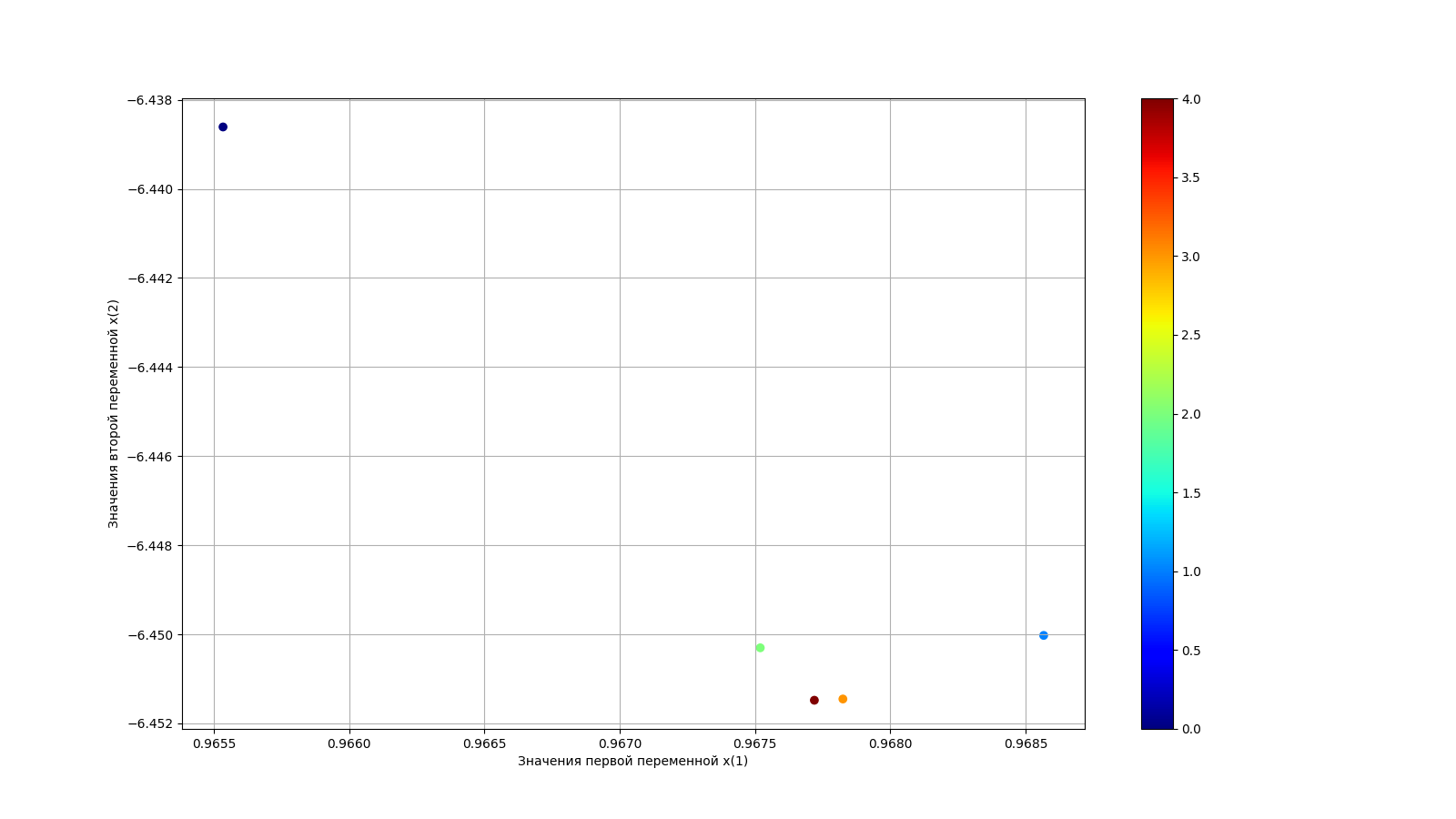


Рисунок 4.6 – Последние 5 элементов таблицы методом наискорейшего градиентного спуска

Сравним полученные результаты при помощи сайта WolframAlpha.



Рисунок 4.7 – Результаты минимизации функции при помощи WolframAlpha

4.5 Вывод

По результатам работы, был реализован методом наискорейшего градиентного спуска (метод первого порядка).

ПРАКТИЧЕСКАЯ РАБОТА №5. МЕТОД ВТОРОГО ПОРЯДКА: МЕТОД НЬЮТОНА

5.1 Постановка задачи

Пользуясь лекционными материалами и методическими указаниями.

Найти минимум целевой функции

методом ньютона с точностью

Зададим начальную точку .

5.2 Описание метода наискорейшего градиентного спуска

В методах второго порядка при поиске минимума функции многих переменных используют информацию о частных производных целевой функции первого и второго порядка. К этой группе относят метод Ньютона и его модификации.

В основе метода Ньютона лежит квадратичная’ аппроксимация целевой функции. Последовательность итераций строится таким образом, чтобы во вновь получаемой точке градиент аппроксимирующей функции обращался в нуль.

Последовательность приближений строится в соответствии с формулой

где k — номер итерации (k = 0, 1, …),

— начальное приближение,

— вектор направления спуска.

Здесь — матрица Гессе.

Направление спуска ведет к убыванию целевой функции только при положительной определенности матрицы Гессе . В тех итерациях, в которых матрица Гессе отрицательно определена , последовательность приближений к точке минимума строится по методу наискорейшего градиентного спуска. С этой целью проводится замена вектора направления спуска на антиградиентное

5.3 Ручной расчёт итераций методом наискорейшего градиентного спуска

Отобразим изначальную таблицу (Таблица 5.1).

Таблица 5.1 – Изначальные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |

Найдем градиент функции и матрицу Гессе в произвольной точке

***Итерация k = 0.*** Определим в формуле

направление спуска . Для этого проведем анализ матрицы Гессе в точке

. Вычислим угловые миноры матрицы Гессе:

Так как знаки угловых миноров строго положительны, то согласно критерию Сильвестра, матрица Гессе положительна определена.

Критерием Сильвестра и состоит в следующем:

1) для того чтобы квадратичная форма была положительно определённой необходимо и достаточно, чтобы все главные диагональные миноры матрицы этой квадратичной формы были положительны.

2) для того чтобы квадратичная форма была отрицательно определённой, необходимо и достаточно, чтобы знаки главных диагональных миноров матрицы этой квадратичной формы чередовались, начиная со знака «–» для.

Следовательно, направление спуска определяем по формуле Ньютона

Вычислим составляющие вектора

Вычислим координаты точки

и значение целевой функции в этой точке

Составим таблицу с новыми значениями

Таблица 4.1 – Изначальные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Номер вершины | Координата 1 | Координата 2 | Значение функции |
| 0 |  |  |  |
| 1 |  |  |  |

Проверим условие окончания процесса поиска для этого вычислим градиент целевой функции в точке :

Так как норма вектора градиента

Условия остановки выполнены, итерации заканчиваются.

5.4 Программная реализация

Реализуем методом наискорейшего градиентного спуска на языке высокого уровня Python.

На рисунках 5.2 – 5.5 представлен результат работы программы. Листинг кода приведен в Приложении Д.

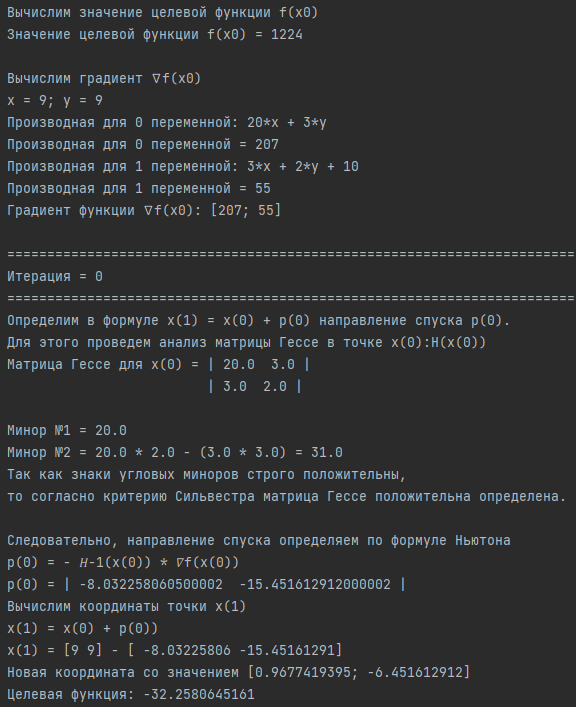


Рисунок 5.2 – Результат работы программы

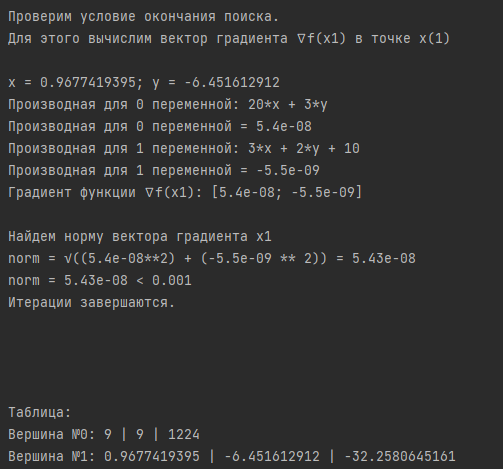


Рисунок 5.3 – Результат работы программы

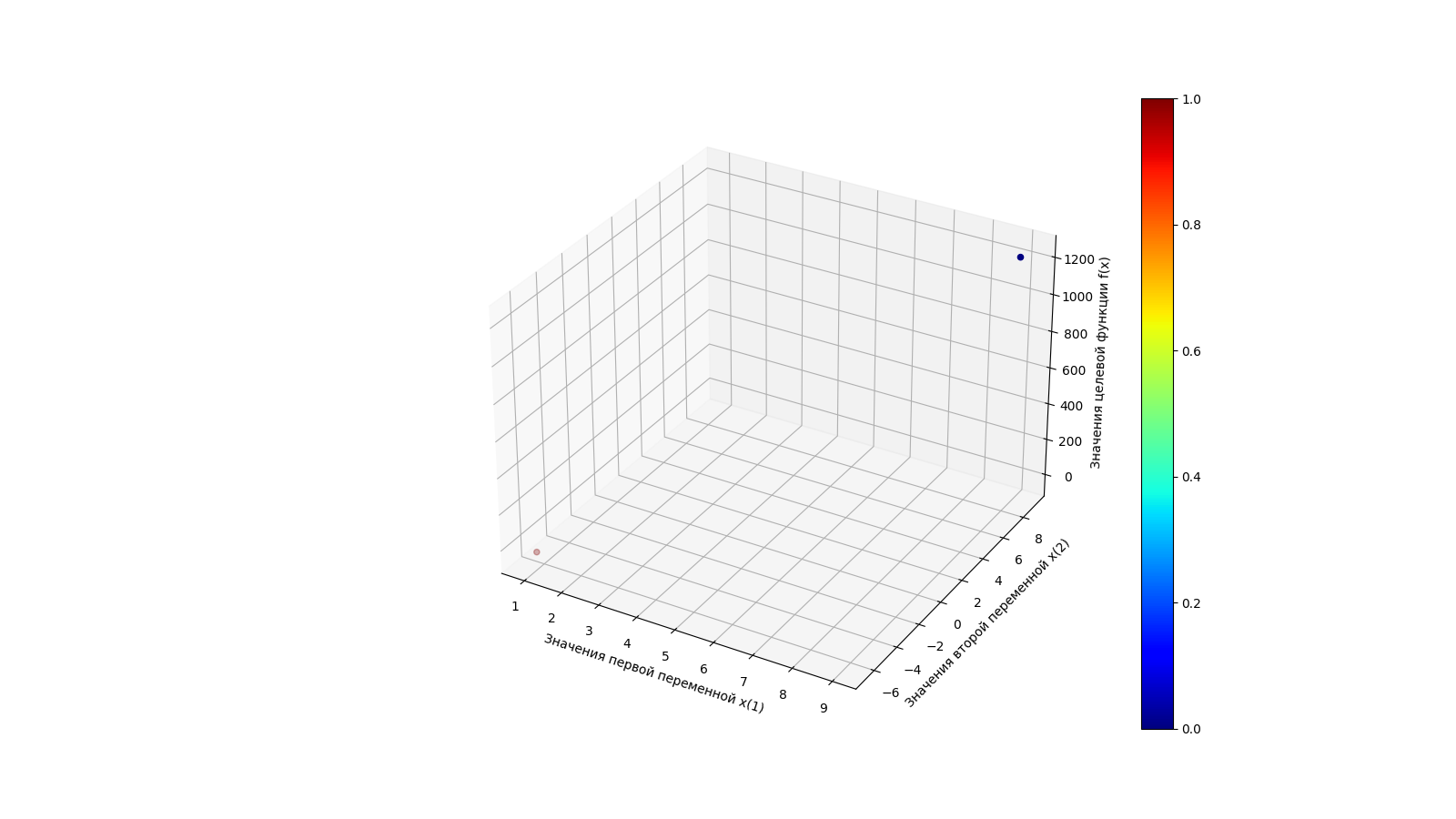


Рисунок 5.4 – Все элементов таблицы методом наискорейшего градиентного спуска

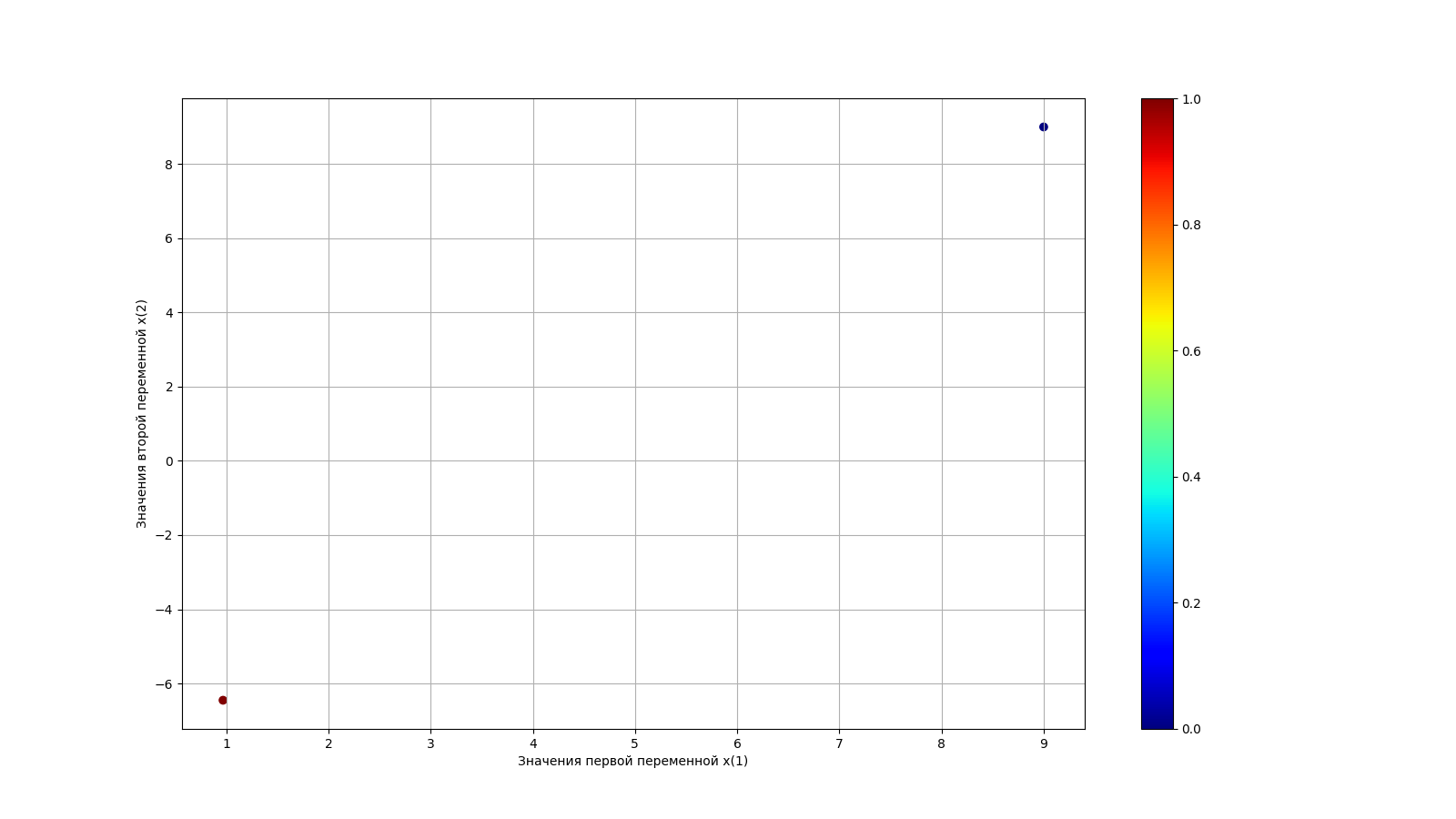


Рисунок 5.5 – Последние 2 элементов таблицы методом наискорейшего градиентного спуска

Сравним полученные результаты при помощи сайта WolframAlpha.



Рисунок 5.6 – Результаты минимизации функции при помощи WolframAlpha

5.5 Вывод

По результатам работы, был реализован метод ньютона (метод второго порядка).

ЗАКЛЮЧЕНИЕ

В ходе выполнения практических работ, были изучены и реализованы 5 методов оптимизации.

Из них методы нулевого порядка:

* Симплекс метод;
* Метод Нелдера-Мида.

Методы первого порядка:

* Метод градиентного спуска с постоянным шагом;
* Метод наискорейшего градиентного спуска.

Метод второго порядка:

* Метод Ньютона.

Все методы были реализованы на языке высокого уровня Python.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гончаров, В.А. Методы оптимизации: Учебное пособие для ВУЗов / В.А.Гончаров. - Люберцы: Юрайт, 2016. - 191 c.

2. Горелик, В.А. Исследование операций и методы оптимизации: Учебник / В.А. Горелик. - М.: Academia, 2018. - 384 c.

3. Келлер, И.Э. Методы оптимизации в примерах и задачах: Учебное пособие / И.Э. Келлер. - СПб.: Лань, 2015. - 512 c.

4. Ширяев, В.И. Исследование операций и численные методы оптимизации: Учебное пособие / В.И. Ширяев. - М.: Ленанд, 2015. - 216 c.

5. Зайцев, М.Г. Методы оптимизации управления и принятия решений: примеры, задачи, кейсы / М.Г. Зайцев, С.Е. Варюхин. - М.: Дело АНХ, 2015. - 640 c.

6. Кочегурова, Е.А. Теория и методы оптимизации.: Учебное пособие для академического бакалавриата / Е.А. Кочегурова. - Люберцы: Юрайт, 2016. - 133 c.

ПРИЛОЖЕНИЯ

Приложение А – листинг кода, для Симплекс метода.

Приложение Б - листинг кода, для метода Нелдера-Мида.

Приложение В - листинг кода, для метода градиентного спуска с постоянным шагом.

Приложение Г - листинг кода, для метода наискорейшего градиентного спуска.

Приложение Д – листинг кода, для метода Ньютона.

Приложение А

Листинг кода, для Симплекс метода.

Листинг А.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':  
 mass\_maximum = []  
 mass = [[9, 9]]  
 n = len(mass[0]) # размерость  
 m = 0.07 # длина ребра симплекса  
 e = 0.001 # точность  
  
 function = "10 \* x \*\* 2 + 3 \* x \* y + y \*\* 2 + 10 \* y" # 7  
 # function = "(2.8 \* y \*\* 2) + 1.9 \* x + (2.7 \* x \*\* 2) + 1.6 - 1.9 \* y"  
 # function = "x \*\* 2 - x \* y + 3 \* y \*\* 2 - x"  
  
 calculate\_increments(mass, m, n) # расчёт изначальныйх точек  
  
 for i in range(len(mass)): # расчёт изначальныйх функций для точек  
 value\_func = count\_target\_function(mass[i][0], mass[i][1])  
 mass[i].append(value\_func)  
  
 table\_output(mass) # вывод таблицы  
  
 iteration = 0  
 while iteration is not True:  
 # while iteration <= 2:  
 print('=' \* 100)  
 print('Итерация =', iteration)  
 iteration += 1  
 print('=' \* 100)  
 maximum\_value\_function(mass, mass\_maximum) # отбераем максимальную точку  
 center\_g = center\_of\_gravity(mass, mass\_maximum) # находим центер  
 new\_coordinate = finding\_coordinates\_reflected\_vertex(mass, mass\_maximum,  
 center\_g) # получем новые коодинаты точки  
 if increased\_or\_decreased(mass, mass\_maximum, new\_coordinate, e, n):  
 print('-' \* 100)  
 print('Так как все условия окончания поиска выполняются, то процесс итерации завершен.')  
 print(  
 f'В качестве приближенного решения x выбирается \nx{len(mass) - 1} = [{round(mass[-1][0], 7)};'  
 f' {round(mass[-1][1], 7)}]\n'  
 f'которой соответствует наименьшее значение целевой функции x{len(mass) - 1} = {round(mass[-1][-1], 7)}')  
 print('-' \* 100)  
 iteration = True  
 else:  
 print('-' \* 100)  
 print('Так как все условия окончания поиска не выполняются, то процесс итераций должен быть продолжен!')  
 print('-' \* 100)  
 print()  
 table\_output(mass)  
 out\_graph(mass)

Листинг А.2 – Метод расчёта изначальных точек

def calculate\_increments(mass, m, n):

print(mass, m, n)

d1 = round(((math.sqrt(n + 1) - 1) / (n \* math.sqrt(2))) \* m, 9)

print(f'd1 = ({(math.sqrt(n + 1) - 1)} / {(n \* math.sqrt(2))}) \* {m}')

print('d1 =', d1, '\n')

d2 = round(((math.sqrt(n + 1) + n - 1) / (n \* math.sqrt(2))) \* m, 9)

print(f'd2 = ({(math.sqrt(n + 1) + n - 1)} / {(n \* math.sqrt(2))}) \* {m}')

print('d2 =', d2, '\n')

new\_x\_1 = [mass[-1][0] + d1, mass[-1][1] + d2]

print(f'new\_x = [{mass[-1][0]} + {d1}; {mass[-1][1]} + {d2}]')

print('new\_x =', new\_x\_1, '\n')

new\_x\_2 = [mass[-1][0] + d2, mass[-1][1] + d1]

print(f'new\_x = [{mass[-1][0]} + {d2}; {mass[-1][1]} + {d1}]')

print('new\_x =', new\_x\_2, '\n')

mass.append(new\_x\_1)

mass.append(new\_x\_2)

Листинг А.3 – Вывод таблицы точек и значений функции

def table\_output(mass):

print('\nТаблица:')

for i in range(len(mass)):

print(f'Вершина №{i}: ', end='')

for j in range(len(mass[i]) - 1):

print(f'{round(mass[i][j], 7)} | ', end='')

print(f'{round(mass[i][-1], 7)}', end='')

print()

print()

Листинг А.4 – Метод отбора максимального значения функции

def maximum\_value\_function(mass, mass\_maximum):

maximum = [-99999999, 0]

for i in range(len(mass)):

if mass[i][-1] > maximum[0] and i not in mass\_maximum:

maximum = [mass[i][-1], i]

print(f'Максимум функции = {round(maximum[0], 7)}, номер вершины = {maximum[1]}')

print()

mass\_maximum.append(maximum[1])

Листинг А.5 – Метод расчёта центра масс симплекса

def center\_of\_gravity(mass, mass\_maximum):

x\_center = [0, 0]

for i in range(len(mass)):

if i not in mass\_maximum:

x\_center[0] += mass[i][0]

x\_center[1] += mass[i][1]

for i in range(len(x\_center)):

x\_center[i] \*= 0.5

print(f'Центер тяжести = [{round(x\_center[0], 7)}, {round(x\_center[1], 7)}]') return x\_center

Листинг А.6 – Метод расчёта новой координаты

def finding\_coordinates\_reflected\_vertex(mass, mass\_maximum, center\_g):

new\_coordinate = []

for i in range(n):

new\_coordinate.append(2 \* center\_g[i] - mass[mass\_maximum[-1]][i])

print(f'Новые координаты = [{round(new\_coordinate[0], 7)}; {round(new\_coordinate[1], 7)}]')

target\_func = count\_target\_function(new\_coordinate[0], new\_coordinate[1])

print('Целевая функция =', round(target\_func, 7), '\n')

new\_coordinate.append(target\_func)

# table\_output(mass)

return new\_coordinate

Листинг А.7 – Метод с критериями остановки

def increased\_or\_decreased(mass, mass\_maximum, new\_coordinate, e, n): # уменьшается функция или увеличивается

# print(mass)

if new\_coordinate[-1] < mass[mass\_maximum[-1]][-1]:

print(f'Наблюдается уменьшение целевой функции:')

print(f' x{len(mass) + 1} < x{mass\_maximum[-1]}')

print(f'{round(new\_coordinate[-1], 7)} < {round(mass[mass\_maximum[-1]][-1], 7)}')

mass.append(new\_coordinate)

return condition\_of\_the\_end\_search(mass, mass\_maximum, e, n)

elif new\_coordinate[-1] > mass[mass\_maximum[-1]][-1]:

print(f'Наблюдается увеличение целевой функции:')

print(f' x{len(mass) + 1} < x{mass\_maximum[-1]}')

print(f'{round(new\_coordinate[-1], 7)} > {round(mass[mass\_maximum[-1]][-1], 7)}')

mass\_maximum.append(len(mass) - 1)

minimum = find\_minimum\_value(mass, mass\_maximum)

generate\_new\_point(mass, mass\_maximum, minimum, m)

return condition\_of\_the\_end\_search(mass, mass\_maximum, e, n)

Листинг А.8 – Метод для визуализации графиков

def out\_graph(data):

import matplotlib.pyplot as plt

import numpy as np

fig = plt.figure(figsize=(16, 9))

ax = fig.add\_subplot(projection='3d')

mass\_x = []

mass\_y = []

value\_func = []

for i in range(-50, 0):

mass\_x.append(data[i][0])

mass\_y.append(data[i][1])

value\_func.append(data[i][-1])

colors = np.arange(len(mass\_x))

col\_1 = ax.scatter(mass\_x, mass\_y, value\_func, cmap="jet", c=colors)

ax.set\_xlabel('Значения первой переменной х(1)')

ax.set\_ylabel('Значения второй переменной х(2)')

ax.set\_zlabel('Значения целевой функции f(x)')

plt.colorbar(col\_1)

plt.savefig("static/plt\_1.png")

Продолжение Листинг А.8

fig2 = plt.figure(figsize=(16, 9))

ax\_2 = fig2.add\_subplot()

mass\_x = []

mass\_y = []

value\_func = []

for i in range(-10, 0):

mass\_x.append(data[i][0])

mass\_y.append(data[i][1])

value\_func.append(data[i][-1])

colors = np.arange(len(mass\_x))

col = ax\_2.scatter(mass\_x, mass\_y, cmap="jet", c=colors)

ax\_2.set\_xlabel('Значения первой переменной х(1)')

ax\_2.set\_ylabel('Значения второй переменной х(2)')

plt.colorbar(col)

ax\_2.grid()

plt.savefig("static/plt\_2.png")

Приложение Б

Листинг кода, для метода Нелдера-Мида.

Листинг Б.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':  
 min\_mean\_max = [1000000, 0, -1000000]  
 mass\_maximum = []  
 mass = [[9, 9]]  
 n = len(mass[0]) # размерость  
 m = 0.07 # длина ребра симплекса  
 B = 2.8 # параметр растяжения  
 y = 0.4 # параметр сжатия  
 e = 0.001 # точность  
  
 function = "10 \* x \*\* 2 + 3 \* x \* y + y \*\* 2 + 10 \* y" # 7  
  
 calculate\_increments(mass, m, n) # расчёт изначальныйх точек  
 for i in range(len(mass)): # расчёт изначальныйх функций для точек  
 value\_func = count\_target\_function(mass[i][0], mass[i][1])  
 mass[i].append(value\_func)  
  
 table\_output(mass) # вывод таблицы  
  
 iteration = 0  
 while iteration is not True:  
 # while iteration != 3:  
 print('=' \* 100)  
 print('Итерация =', iteration)  
 iteration += 1  
 print('=' \* 100)  
 determination\_of\_min\_mean\_max(min\_mean\_max, mass, mass\_maximum) # находим минимум максимум и среднее значение  
  
 maximum\_value\_function(mass, mass\_maximum) # отбераем максимальную точку  
  
 center\_g = center\_of\_gravity(mass, mass\_maximum) # находим центер  
  
 new\_coordinate = finding\_coordinates\_reflected\_vertex(mass, mass\_maximum,  
 center\_g, min\_mean\_max) # получем новые коодинаты точки  
  
 if changing\_the\_function(mass, mass\_maximum, new\_coordinate,  
 min\_mean\_max): # наблюдается ли уменьшение целевой функции?  
 if condition\_fulfillment(mass, mass\_maximum, min\_mean\_max, new\_coordinate):  
 new\_coordinate\_compressions = simplex\_compressions(mass, mass\_maximum, center\_g, y)  
 if new\_coordinate\_compressions[-1] < min\_mean\_max[-1][0]:  
 print('\n\nУчитывая, что условие сжатия выполнено,')  
 print(f' x{len(mass)} < x{min\_mean\_max[-1][1]}')  
 print('то добавим значение в таблицу')  
 mass.append(new\_coordinate\_compressions)  
 table\_output(mass)  
 if condition\_for\_the\_end\_of\_the\_search(mass, e, mass\_maximum):  
 iteration = True

Продолжение Листинг Б.1

else:  
 for i in range(len(mass) - 1):  
 if i not in mass\_maximum:  
 mass\_maximum.append(i)  
 print('\n\nУсловие растяжения не выполнено')  
 print(f'\nСформируем новый многогранник с уменьшенными вдвое сторонами и вершиной x{len(mass) - 1}')  
 new\_polyhedron(mass, mass\_maximum, min\_mean\_max) # генерация нового многоугольника  
 if condition\_for\_the\_end\_of\_the\_search(mass, e, mass\_maximum): # условие окончание поиска  
 iteration = True  
 else:  
 new\_coordinate\_stretching = simplex\_stretching(mass, mass\_maximum, center\_g, B)  
 if new\_coordinate\_stretching[-1] < mass[-1][-1]:  
 print('\n\nУсловие растяжения выполнено')  
 else:  
 for i in range(len(mass) - 1):  
 if i not in mass\_maximum:  
 mass\_maximum.append(i)  
 print('\n\nУсловие растяжения не выполнено')  
 print(f'\nСформируем новый многогранник с уменьшенными вдвое сторонами и вершиной x{len(mass) - 1}')  
 new\_polyhedron(mass, mass\_maximum, min\_mean\_max) # генерация нового многоугольника  
 if condition\_for\_the\_end\_of\_the\_search(mass, e, mass\_maximum): # условие окончание поиска  
 iteration = True  
 else:  
 mass\_maximum.append(min\_mean\_max[1][1])  
 mass\_maximum.append(min\_mean\_max[-1][1])  
 new\_polyhedron(mass, mass\_maximum, min\_mean\_max)  
 if condition\_for\_the\_end\_of\_the\_search(mass, e, mass\_maximum): # условие окончание поиска  
 iteration = True  
 if condition\_for\_the\_end\_of\_the\_search(mass, e, mass\_maximum): # условие окончание поиска  
 iteration = True  
  
 table\_output(mass)  
 out\_graph(mass)

Листинг Б.2 – Метод расчёта изначальных точек

def calculate\_increments(mass, m, n):

print(mass, m, n)

d1 = round(((math.sqrt(n + 1) - 1) / (n \* math.sqrt(2))) \* m, 9)

print(f'd1 = ({(math.sqrt(n + 1) - 1)} / {(n \* math.sqrt(2))}) \* {m}')

print('d1 =', d1, '\n')

d2 = round(((math.sqrt(n + 1) + n - 1) / (n \* math.sqrt(2))) \* m, 9)

print(f'd2 = ({(math.sqrt(n + 1) + n - 1)} / {(n \* math.sqrt(2))}) \* {m}')

print('d2 =', d2, '\n')

new\_x\_1 = [mass[-1][0] + d1, mass[-1][1] + d2]

print(f'new\_x = [{mass[-1][0]} + {d1}; {mass[-1][1]} + {d2}]')

print('new\_x =', new\_x\_1, '\n')

new\_x\_2 = [mass[-1][0] + d2, mass[-1][1] + d1]

print(f'new\_x = [{mass[-1][0]} + {d2}; {mass[-1][1]} + {d1}]')

print('new\_x =', new\_x\_2, '\n')

mass.append(new\_x\_1)

mass.append(new\_x\_2)

Листинг Б.3 – Вывод таблицы точек и значений функции

def table\_output(mass):

print('\nТаблица:')

for i in range(len(mass)):

print(f'Вершина №{i}: ', end='')

for j in range(len(mass[i]) - 1):

print(f'{round(mass[i][j], 7)} | ', end='')

print(f'{round(mass[i][-1], 7)}', end='')

print()

print()

Листинг Б.4 – Метод расчёта минимума, максимума и среднего значения

def determination\_of\_min\_mean\_max(min\_mean\_max, mass, mass\_maximum):

massive\_for\_search = []

for i in range(len(mass)):

if i not in mass\_maximum:

massive\_for\_search.append([mass[i][-1], i])

maximum = [-1000000, 123]

minimal = [1000000, 123]

for i in range(len(massive\_for\_search)):

if massive\_for\_search[i][0] > maximum[0]:

maximum = massive\_for\_search[i]

if massive\_for\_search[i][0] < minimal[0]:

minimal = massive\_for\_search[i]

mean = 0

for i in range(len(massive\_for\_search)):

if massive\_for\_search[i][1] != minimal[1] and massive\_for\_search[i][1] != maximum[1]:

mean = massive\_for\_search[i]

for i in range(len(min\_mean\_max)):

if i == 0:

min\_mean\_max[i] = minimal

elif i == 1:

min\_mean\_max[i] = mean

elif i == 2:

min\_mean\_max[i] = maximum

Листинг Б.5 – Метод отбора максимального значения функции

def maximum\_value\_function(mass, mass\_maximum):

maximum = [-99999999, 0]

for i in range(len(mass)):

if mass[i][-1] > maximum[0] and i not in mass\_maximum:

maximum = [mass[i][-1], i]

print(f'Максимум функции = {round(maximum[0], 7)}, номер вершины = {maximum[1]}')

print()

mass\_maximum.append(maximum[1])

Листинг Б.6 – Метод расчёта центра масс

def center\_of\_gravity(mass, mass\_maximum):

x\_center = [0, 0]

for i in range(len(mass)):

if i not in mass\_maximum:

x\_center[0] += mass[i][0]

x\_center[1] += mass[i][1]

for i in range(len(x\_center)):

x\_center[i] \*= 0.5

print(f'Центер тяжести = [{round(x\_center[0], 7)}, {round(x\_center[1], 7)}]') return x\_center

Листинг Б.7 – Метод расчёта новой координаты

def finding\_coordinates\_reflected\_vertex(mass, mass\_maximum, center\_g):

new\_coordinate = []

for i in range(n):

new\_coordinate.append(2 \* center\_g[i] - mass[mass\_maximum[-1]][i])

print(f'Новые координаты = [{round(new\_coordinate[0], 7)}; {round(new\_coordinate[1], 7)}]')

target\_func = count\_target\_function(new\_coordinate[0], new\_coordinate[1])

print('Целевая функция =', round(target\_func, 7), '\n')

new\_coordinate.append(target\_func)

# table\_output(mass)

return new\_coordinate

Листинг Б.8 – Метод уменьшается функция или увеличивается

def changing\_the\_function(mass, mass\_maximum, new\_coordinate, min\_mean\_max): # уменьшается функция или увеличивается

if new\_coordinate[-1] < mass[min\_mean\_max[-1][1]][-1]:

print(f'Наблюдается уменьшение целевой функции:')

print(f'x{len(mass)} < x{min\_mean\_max[-1][1]}')

print(f'{round(new\_coordinate[-1], 7)} < {round(mass[mass\_maximum[-1]][-1], 7)}')

mass.append(new\_coordinate)

return True

elif new\_coordinate[-1] > mass[min\_mean\_max[-1][1]][-1]:

print(f'Наблюдается увеличение целевой функции:')

print(f'x{len(mass)} > x{min\_mean\_max[-1][1]}')

print(f'{round(new\_coordinate[-1], 7)} > {round(mass[mass\_maximum[-1]][-1], 7)}\n\n')

return False

Листинг Б.9 – Функция определения выполнения условия

def condition\_fulfillment(mass, mass\_maximum, min\_mean\_max, new\_coordinate):

if min\_mean\_max[1][0] < new\_coordinate[-1] < min\_mean\_max[-1][0]:

print(f'\nТак как выполняется условие:')

print(f'x{min\_mean\_max[1][1]} < x{len(mass) - 1} < x{min\_mean\_max[-1][1]}')

print(f'{min\_mean\_max[1][0]} < {new\_coordinate[-1]} < {min\_mean\_max[-1][0]}')

mass\_maximum.append(len(mass) - 1)

return True

else:

print(f'\nУсловие не выполняется:')

print(f'x{min\_mean\_max[1][1]} <! x{len(mass) - 1} <! x{min\_mean\_max[-1][1]}')

print(f'x{min\_mean\_max[1][0]} <! x{new\_coordinate[-1]} <! x{min\_mean\_max[-1][0]}')

return False

Листинг Б.10 – Метод сжатия симплекса

def simplex\_compressions(mass, mass\_maximum, center\_g, y):

print('\nВыполним операцию сжатия симплекса:')

new\_coordinate\_compressions = [0, 0]

for i in range(len(new\_coordinate\_compressions)):

print(f'X{len(mass) - 1}[{i}] = {center\_g[i]} + {y} \*'

f' ({mass[-1][i]} - ({center\_g[i]}))')

new\_coordinate\_compressions[i] = round(center\_g[i] + y \* (mass[-1][i] - center\_g[i]), 9)

target\_func = count\_target\_function(new\_coordinate\_compressions[0], new\_coordinate\_compressions[1])

new\_coordinate\_compressions.append(target\_func)

print(f'\nПолучаем точку с координатами: [{new\_coordinate\_compressions[0]}; {new\_coordinate\_compressions[1]}]')

print(f'В полученной вершине значение целевой функции = {new\_coordinate\_compressions[-1]}')

return new\_coordinate\_compressions

Листинг Б.11 – Метод растяжения симплекса

def simplex\_stretching(mass, mass\_maximum, center\_g, B):

print('\nВыполним операцию растяжения симплекса:')

new\_coordinate\_stretching = [0, 0]

for i in range(n):

print(f'x{len(mass)} = {center\_g[i]} + {B} \*'

f' ({mass[-1][i]} - {center\_g[i]})')

new\_coordinate\_stretching[i] = round(center\_g[i] + B \* (mass[-1][i] - center\_g[i]), 9)

target\_func = count\_target\_function(new\_coordinate\_stretching[0], new\_coordinate\_stretching[1])

new\_coordinate\_stretching.append(target\_func)

print(f'Получаем точку с координатами: [{new\_coordinate\_stretching[0]}; {new\_coordinate\_stretching[1]}]')

print(f'И функцией = {target\_func}')

return new\_coordinate\_stretching

Листинг Б.12 – Метод генерации нового многоугольника

def new\_polyhedron(mass, mass\_maximum, min\_mean\_max):

for k in range(n):

new = [0, 0]

for l in range(n):

new[l] = round(mass[min\_mean\_max[0][1]][l] + 0.5 \*

(mass[mass\_maximum[-2 + k]][l] - mass[min\_mean\_max[0][1]][l]), 9)

print(f'{new[l]} = {mass[min\_mean\_max[-1][1]][l]} + 0.5 \* ({mass[mass\_maximum[-2 + k]][l]} - '

f'({mass[min\_mean\_max[-1][1]][l]}))')

new.append(count\_target\_function(new[0], new[1]))

print(f'Новые координаты = [{new[0]}; {new[1]}]')

print(f'Целевая функция равна: {new[-1]}\n')

mass.append(new)

Листинг Б.13 – Метод с критериями остановки

def condition\_for\_the\_end\_of\_the\_search(mass, e, mass\_maximum):

mass\_center\_gravity\_simplex = []

mass\_coof = []

for j in range(3):

minimum = [100000, 0]

for i in range(len(mass)):

if mass[i][-1] < minimum[0] and i not in mass\_coof and i not in mass\_maximum:

minimum = [mass[i][-1], i]

mass\_coof.append(minimum[1])

mass\_center\_gravity\_simplex.append(minimum)

print(mass\_center\_gravity\_simplex)

x\_center = [0, 0]

for i in range(len(mass\_center\_gravity\_simplex)):

x\_center[0] += mass[mass\_center\_gravity\_simplex[i][1]][0]

x\_center[1] += mass[mass\_center\_gravity\_simplex[i][1]][1]

for i in range(len(x\_center)):

x\_center[i] = round(x\_center[i] / 3, 9)

x\_center.append(count\_target\_function(x\_center[0], x\_center[1]))

print(f'\nПроверим условие окончания поиска\nЦентер тяжести = [{x\_center[0]}, {x\_center[1]}]')

print(f'В полученной вершине значение целевой функции = {x\_center[-1]}\n')

sigma = 0

for i in range(len(mass\_center\_gravity\_simplex)):

sigma += (mass[mass\_center\_gravity\_simplex[i][1]][-1] - x\_center[-1]) \*\* 2

sigma = sigma / (n + 1)

sigma = round(math.sqrt(sigma), 9)

if sigma < e:

print(f'Сигма = {sigma} < {e}')

print('Так как условие окончания поиска выполняется, то процесс итераций завершен.')

return True

else:

print(f'Сигма = {sigma} >= {e}')

print('Так как условие окончания поиска не выполняется, то процесс итерации должен быть продолжен.\n\n')

return False

Листинг Б.14 – Метод для визуализации графиков

def out\_graph(data):

import matplotlib.pyplot as plt

import numpy as np

fig = plt.figure(figsize=(16, 9))

ax = fig.add\_subplot(projection='3d')

mass\_x = []

mass\_y = []

value\_func = []

for i in range(-50, 0):

mass\_x.append(data[i][0])

mass\_y.append(data[i][1])

value\_func.append(data[i][-1])

colors = np.arange(len(mass\_x))

col\_1 = ax.scatter(mass\_x, mass\_y, value\_func, cmap="jet", c=colors)

ax.set\_xlabel('Значения первой переменной х(1)')

ax.set\_ylabel('Значения второй переменной х(2)')

ax.set\_zlabel('Значения целевой функции f(x)')

plt.colorbar(col\_1)

plt.savefig("static/plt\_1.png")

fig2 = plt.figure(figsize=(16, 9))

ax\_2 = fig2.add\_subplot()

mass\_x = []

mass\_y = []

value\_func = []

for i in range(-10, 0):

mass\_x.append(data[i][0])

mass\_y.append(data[i][1])

value\_func.append(data[i][-1])

colors = np.arange(len(mass\_x))

col = ax\_2.scatter(mass\_x, mass\_y, cmap="jet", c=colors)

ax\_2.set\_xlabel('Значения первой переменной х(1)')

ax\_2.set\_ylabel('Значения второй переменной х(2)')

plt.colorbar(col)

ax\_2.grid()

plt.savefig("static/plt\_2.png")

Приложение В

Листинг кода, для метода градиентного спуска с постоянным шагом.

Листинг В.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':  
 mass = [[9, 9]]  
 n = len(mass[0]) # размерость  
 h = 0.4 # начальная величина шага  
 mass\_h = [h]  
 e = 0.001 # точность  
 grad = 0  
 mass\_grad = []  
  
 function = "10 \* x \*\* 2 + 3 \* x \* y + y \*\* 2 + 10 \* y" # 7  
  
 iteration = 0  
  
 print(f'Вычислим значение целевой функции f(x{len(mass) - 1})\n')  
 mass[-1].append(count\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Значение целевой функции f(x{len(mass) - 1}) = {mass[-1][2]}\n')  
 print(f'Вычислим градиент ∇f(x{len(mass) - 1})')  
 mass\_grad.append(count\_grad\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Градиент функции ∇f(x{len(mass) - 1}): [{mass\_grad[-1][0]}; {mass\_grad[-1][1]}]\n')  
  
 # while iteration != 2:  
 while iteration != -404:  
 print('=' \* 100)  
 print('Итерация =', iteration)  
 print('=' \* 100)  
 iteration += 1  
 print(f'Найдем новую координату x[{len(mass)}]\n')  
 new\_coordinate = generate\_new\_coordinate(mass[-1], mass\_grad[-1], mass\_h)  
 print(f'Сравним f(x{len(mass) - 1}) и f (𝑥({len(mass)}))')  
 if changing\_the\_function(mass, new\_coordinate, grad, h, iteration) is True:  
 iteration = -404  
 table\_output(mass)  
 out\_graph(mass)

Листинг В.2 – Метод вычисление новой координаты

def generate\_new\_coordinate(mass, grad, mass\_h):  
 print(mass)  
 new\_coordinate = [0, 0]  
 for k in range(2):  
 print(f'{round\_value(mass[k] - mass\_h[-1] \* grad[k])} = {mass[k]} - {mass\_h[-1]} \* {grad[k]}')  
 new\_coordinate[k] = mass[k] - mass\_h[-1] \* grad[k]  
 new\_coordinate[k] = round\_value(new\_coordinate[k])  
 print(f'Новая координата со значением [{new\_coordinate[0]}; {new\_coordinate[1]}]')  
 new\_coordinate.append(count\_target\_function(new\_coordinate[0], new\_coordinate[1]))  
 print(f'Целевая функция: {new\_coordinate[-1]}\n\n')  
 return new\_coordinate

Листинг В.3 – Метод вычисление новой координаты

def generate\_new\_coordinate(mass, grad, mass\_h):  
 print(mass)  
 new\_coordinate = [0, 0]  
 for k in range(2):  
 print(f'{round\_value(mass[k] - mass\_h[-1] \* grad[k])} = {mass[k]} - {mass\_h[-1]} \* {grad[k]}')  
 new\_coordinate[k] = mass[k] - mass\_h[-1] \* grad[k]  
 new\_coordinate[k] = round\_value(new\_coordinate[k])  
 print(f'Новая координата со значением [{new\_coordinate[0]}; {new\_coordinate[1]}]')  
 new\_coordinate.append(count\_target\_function(new\_coordinate[0], new\_coordinate[1]))  
 print(f'Целевая функция: {new\_coordinate[-1]}\n\n')  
  
 return new\_coordinate

Листинг В.4 – Метод замена шага

def changing\_the\_function(mass, new\_coordinate, grad, h, iteration):  
 if new\_coordinate[-1] < mass[-1][-1]:  
 print(f'Поскольку и f(x{len(mass)}) < f(x{len(mass) - 1})')  
 print(f'Условие убывания функции выполнено.\n')  
 mass.append(new\_coordinate)  
 return condition\_end\_search(mass, iteration)  
 else:  
 print(f'Поскольку и f(x{len(mass)}) > f(x{len(mass) - 1})')  
 print(f'Условие убывания функции не выполнено.\n'  
 f'Уменьшим величину шага h = h/2 = {mass\_h[-1] / 2}\n')  
  
 mass\_h.append(mass\_h[-1] / 2)  
 print(f'Повторим вычисления координат точки x[{len(mass)}]')  
 new\_coordinate = generate\_new\_coordinate(mass[-1], mass\_grad[-1], mass\_h)  
 print(f'Сравним f(x{len(mass) - 1}) и f (𝑥({len(mass)}))')  
  
 return changing\_the\_function(mass, new\_coordinate, mass\_grad[-1], mass\_h, iteration)

Листинг В.5 – Метод расчёта функции

def count\_target\_function(x, y):  
 target\_function = eval(function)  
 return round\_value(target\_function)

Листинг В.6 – Метод для округления

def round\_value(value):  
 return round(value, 10)

Листинг В.7 – Метод для расчёта производной

def calculating\_the\_derivative():  
 result\_diff = []  
 x, y = symbols('x y')  
 target\_function = eval(function)  
 result\_diff.append(str(target\_function.diff(x)))  
 result\_diff.append(str(target\_function.diff(y)))  
 return result\_diff

Листинг В.8 – Метод условия остановки функции

def condition\_end\_search(mass, iteration):  
 print(f'Проверим условие окончания поиска.\n'  
 f'Для этого вычислим вектор градиента ∇f(x{len(mass) - 1}) в точке'  
 f' x({len(mass) - 1})\n')  
 mass\_grad.append(count\_grad\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Градиент функции ∇f(x{len(mass) - 1}): [{mass\_grad[-1][0]}; {mass\_grad[-1][1]}]\n')  
 return find\_norm\_gradient\_vector(mass, mass\_grad[-1], e, iteration)

Листинг В.9 – Метод расчёта нормы градиента

def find\_norm\_gradient\_vector(mass, grad, e, iteration):  
 print(f'Найдем норму вектора градиента x{len(mass) - 1}')  
 norm = round\_value(math.sqrt((grad[0] \*\* 2) + (grad[1] \*\* 2)))  
 print(f'norm = √(({grad[0]}\*\*2) + ({grad[1]} \*\* 2)) = {norm}')  
 if norm >= e:  
 print(f'norm = {norm} >= {e}')  
 print('Итерации продолжаются.\n\n\n')  
 return iteration  
 else:  
 print(f'norm = {norm} < {e}')  
 print('Итерации завершаются.\n\n\n')  
 return True

Листинг В.10 – Метод вывода таблицы значений

def table\_output(mass):  
 print('\nТаблица:')  
 for i in range(len(mass)):  
 print(f'Вершина №{i}: ', end='')  
 for j in range(len(mass[i]) - 1):  
 print(f'{round(mass[i][j], 10)} | ', end='')  
 print(f'{round(mass[i][-1], 10)}', end='')  
 print()  
 print()

Листинг В.11 – Метод для вывода графиков

def out\_graph(data):  
 import matplotlib.pyplot as plt  
 import numpy as np  
  
 fig0 = plt.figure(figsize=(16, 9))  
 ax0 = fig0.add\_subplot(projection='3d')  
 mass\_x = []  
 mass\_y = []  
 value\_func = []  
 for i in range(-(len(mass)), 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
 colors = np.arange(len(mass\_x))  
 col\_1 = ax0.scatter(mass\_x, mass\_y, value\_func, cmap="jet", c=colors)  
 ax0.set\_xlabel('Значения первой переменной х(1)')  
 ax0.set\_ylabel('Значения второй переменной х(2)')  
 ax0.set\_zlabel('Значения целевой функции f(x)')  
 plt.colorbar(col\_1)  
 plt.savefig("static/plt\_0.png")  
  
 fig = plt.figure(figsize=(16, 9))  
 ax = fig.add\_subplot(projection='3d')  
 mass\_x = []  
 mass\_y = []  
 value\_func = []  
 for i in range(-50, 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
 colors = np.arange(len(mass\_x))  
 col\_1 = ax.scatter(mass\_x, mass\_y, value\_func, cmap="jet", c=colors)  
 ax.set\_xlabel('Значения первой переменной х(1)')  
 ax.set\_ylabel('Значения второй переменной х(2)')  
 ax.set\_zlabel('Значения целевой функции f(x)')  
 plt.colorbar(col\_1)  
 plt.savefig("static/plt\_1.png")  
  
 fig2 = plt.figure(figsize=(16, 9))  
 ax\_2 = fig2.add\_subplot()  
 mass\_x = []  
 mass\_y = []  
 value\_func = []  
 for i in range(-10, 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
 colors = np.arange(len(mass\_x))  
 col = ax\_2.scatter(mass\_x, mass\_y, cmap="jet", c=colors)  
 ax\_2.set\_xlabel('Значения первой переменной х(1)')  
 ax\_2.set\_ylabel('Значения второй переменной х(2)')  
 plt.colorbar(col)  
 ax\_2.grid()  
 plt.savefig("static/plt\_2.png")

Приложение Г

Листинг кода, для метода наискорейшего градиентного спуска.

Листинг Г.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':  
 mass = [[9, 9]]  
 n = len(mass[0]) # размерость  
 mass\_grad = []  
 e = 0.001 # точность  
 iteration = 0  
 function = "10 \* x \*\* 2 + 3 \* x \* y + y \*\* 2 + 10 \* y" # 7  
  
 print(f'Вычислим значение целевой функции f(x{len(mass) - 1})\n')  
 mass[-1].append(count\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Значение целевой функции f(x{len(mass) - 1}) = {mass[-1][2]}\n')  
 print(f'Вычислим градиент ∇f(x{len(mass) - 1})')  
 mass\_grad.append(count\_grad\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Градиент функции ∇f(x{len(mass) - 1}): [{mass\_grad[-1][0]}; {mass\_grad[-1][1]}]\n')  
 while iteration != -404:  
 print('=' \* 100)  
 print('Итерация =', iteration)  
 print('=' \* 100)  
 iteration += 1  
 print(f'Найдем новую координату x[{len(mass)}]\n')  
 new\_coordinate = generate\_new\_coordinate(mass, mass\_grad[-1])  
 iteration = condition\_end\_search(mass)  
 if iteration is False:  
 iteration = -404  
 table\_output(mass)  
 out\_graph(mass)

Листинг Г.2 – Метод расчёта функции

def count\_target\_function(x, y):  
 target\_function = eval(function)  
 return round\_value(target\_function)

Листинг Г.3 – Метод для округления

def round\_value(value):  
 return round(value, 10)

Листинг Г.4 – Метод вывода таблицы значений

def table\_output(mass):  
 print('\nТаблица:')  
 for i in range(len(mass)):  
 print(f'Вершина №{i}: ', end='')  
 for j in range(len(mass[i]) - 1):  
 print(f'{round(mass[i][j], 10)} | ', end='')  
 print(f'{round(mass[i][-1], 10)}', end='')  
 print()  
 print()

Листинг Г.5 – Метод для расчёта шага

def count\_h(value\_x, value\_y):  
 derivative = derivative\_for\_h()  
 print(derivative)  
 print(value\_x)  
 print(value\_y)  
 derivative = derivative.replace('x', value\_x)  
 derivative = derivative.replace('y', value\_y)  
 print('\nchange ====', derivative, )  
 value, h = calculating\_the\_derivative\_for\_h(derivative)  
 if h >= 0:  
 h = - value / h  
 else:  
 h = value / h  
 h = round\_value(h)  
 print(h)  
 if h > 0:  
 return h  
 else:  
 return False

Листинг Г.6 – Метод условия остановки поиска

def condition\_end\_search(mass):  
 print(f'Проверим условие окончания поиска.\n'  
 f'Для этого вычислим вектор градиента ∇f(x{len(mass) - 1}) в точке'  
 f' x({len(mass) - 1})\n')  
 mass\_grad.append(count\_grad\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Градиент функции ∇f(x{len(mass) - 1}): [{mass\_grad[-1][0]}; {mass\_grad[-1][1]}]\n')  
 return find\_norm\_gradient\_vector(mass, mass\_grad[-1], e)

Листинг Г.7 – Метод расчёта градиента функции

def count\_grad\_target\_function(value\_x, value\_y):  
 massive\_derivative\_calculating = []  
 derivative = calculating\_the\_derivative\_for()  
 for index in range(2):  
 y = value\_y  
 x = value\_x  
 print(f'Производная для {index} переменной: {derivative[index]}')  
 relust = round\_value(eval(f'{derivative[index]}'))  
 print(f'Производная для {index} переменной = {relust}')  
 massive\_derivative\_calculating.append(relust)  
 return massive\_derivative\_calculating

Листинг Г.8 – Метод расчёта нормы градиента

def find\_norm\_gradient\_vector(mass, grad, e):  
 print(f'Найдем норму вектора градиента x{len(mass) - 1}')  
 norm = round\_value(math.sqrt((grad[0] \*\* 2) + (grad[1] \*\* 2)))  
 print(f'norm = √(({grad[0]}\*\*2) + ({grad[1]} \*\* 2)) = {norm}')  
 if norm >= e:  
 print(f'norm = {norm} >= {e}')  
 print('Итерации продолжаются.\n\n\n')  
 return iteration  
 else:  
 print(f'norm = {norm} < {e}')  
 print('Итерации завершаются.\n\n\n')  
 return False

Листинг Г.9 – Метод для вывода графиков

def out\_graph(data):  
 import matplotlib.pyplot as plt  
 import numpy as np  
  
 fig = plt.figure(figsize=(16, 9))  
 ax = fig.add\_subplot(projection='3d')  
  
 mass\_x = []  
 mass\_y = []  
 value\_func = []  
 for i in range(-len(data), 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
 colors = np.arange(len(mass\_x))  
 col\_1 = ax.scatter(mass\_x, mass\_y, value\_func, cmap="jet", c=colors)  
 ax.set\_xlabel('Значения первой переменной х(1)')  
 ax.set\_ylabel('Значения второй переменной х(2)')  
 ax.set\_zlabel('Значения целевой функции f(x)')  
 plt.colorbar(col\_1)  
 plt.savefig("static/plt\_1.png")  
  
 fig2 = plt.figure(figsize=(16, 9))  
 ax\_2 = fig2.add\_subplot()  
 mass\_x = []  
 mass\_y = []  
 value\_func = []  
 for i in range(-5, 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
 colors = np.arange(len(mass\_x))  
 col = ax\_2.scatter(mass\_x, mass\_y, cmap="jet", c=colors)  
 ax\_2.set\_xlabel('Значения первой переменной х(1)')  
 ax\_2.set\_ylabel('Значения второй переменной х(2)')  
 plt.colorbar(col)  
 ax\_2.grid()  
 plt.savefig("static/plt\_2.png")

Приложение Д

Листинг кода, для метода Ньютона.

Листинг Д.1 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':  
 mass = [[9, 9]]  
 n = len(mass[0]) # размерость  
 mass\_grad = []  
 mass\_p = [] # направление спуска  
 e = 0.001 # точность  
 iteration = 0  
  
 function = "10 \* x \*\* 2 + 3 \* x \* y + y \*\* 2 + 10 \* y" # 7  
  
 print(f'Вычислим значение целевой функции f(x{len(mass) - 1})')  
 mass[-1].append(count\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Значение целевой функции f(x{len(mass) - 1}) = {mass[-1][2]}\n')  
 print(f'Вычислим градиент ∇f(x{len(mass) - 1})')  
 mass\_grad.append(count\_grad\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Градиент функции ∇f(x{len(mass) - 1}): [{mass\_grad[-1][0]}; {mass\_grad[-1][1]}]\n')  
 while iteration != -404:  
 # while iteration != 3:  
 print('=' \* 100)  
 print('Итерация =', iteration)  
 print('=' \* 100)  
  
 iteration += 1  
 print(f'Определим в формуле x({len(mass)}) = '  
 f'x({len(mass) - 1}) + p({len(mass) - 1}) направление спуска p({len(mass) - 1}).')  
 print(f'Для этого проведем анализ матрицы Гессе в точке x({len(mass) - 1}):H(x({len(mass) - 1}))')  
  
 hesse\_matrix = calculating\_hesse\_matrix(mass) # матрица Гессе  
 if angular\_minors\_of\_hesse\_matrix(hesse\_matrix):  
 p = determining\_direction\_descent(mass, hesse\_matrix, mass\_grad[-1])  
 calculating\_coordinates\_new\_point(mass, p)  
 if condition\_end\_search(mass) == -404:  
 iteration = -404  
 else:  
 calculate\_step(mass)  
 table\_output(mass)  
 out\_graph(mass)

Листинг Д.2 – Метод расчёта функции

def count\_target\_function(x, y):  
 target\_function = eval(function)  
 return round\_value(target\_function)

Листинг Д.3 – Метод для округления

def round\_value(value):  
 return round(value, 10)

Листинг Д.4 – Метод вывода таблицы значений

def table\_output(mass):  
 print('\nТаблица:')  
 for i in range(len(mass)):  
 print(f'Вершина №{i}: ', end='')  
 for j in range(len(mass[i]) - 1):  
 print(f'{round(mass[i][j], 10)} | ', end='')  
 print(f'{round(mass[i][-1], 10)}', end='')  
 print()  
 print()

Листинг Д.5 – Метод расчёта градиента функции

def count\_grad\_target\_function(value\_x, value\_y):  
 massive\_derivative\_calculating = []  
 derivative = calculating\_the\_derivative()  
 y = value\_y  
 x = value\_x  
 print(f'x = {x}; y = {y}')  
 for index in range(2):  
 print(f'Производная для {index} переменной: {derivative[index]}')  
 relust = round\_value(eval(f'{derivative[index]}'))  
 print(f'Производная для {index} переменной = {relust}')  
 massive\_derivative\_calculating.append(relust)  
 return massive\_derivative\_calculating

Листинг Д.6 – Метод расчёта производой

def calculating\_the\_derivative():  
 result\_diff = []  
 x, y = symbols('x y')  
 target\_function = eval(function)  
 result\_diff.append(str(target\_function.diff(x)))  
 result\_diff.append(str(target\_function.diff(y)))  
 return result\_diff

Листинг Д.7 – Метод расчёта матрицы Хессе

def calculating\_hesse\_matrix(mass):  
 hesse\_matrix = []  
 x, y = symbols('x y')  
 target\_function = eval(function)  
 hesse\_matrix.append([round\_value(float(target\_function.diff(x, x))),  
 round\_value(float(target\_function.diff(x, y)))])  
 hesse\_matrix.append([round\_value(float(target\_function.diff(y, x))),  
 round\_value(float(target\_function.diff(y, y)))])  
  
 string\_for\_out\_on\_scren = f'Матрица Гессе для x({len(mass) - 1}) = '  
 print(string\_for\_out\_on\_scren + f'| {hesse\_matrix[0][0]} {hesse\_matrix[0][1]} |')  
 print(f' ' \* len(string\_for\_out\_on\_scren) + f'| {hesse\_matrix[1][0]} {hesse\_matrix[1][1]} |\n')  
 return hesse\_matrix

Листинг Д.8 – Метод расчёта миноров в матрице Хессе

def angular\_minors\_of\_hesse\_matrix(hesse\_matrix):  
 minor\_1 = round\_value(hesse\_matrix[0][0])  
 minor\_2 = round\_value(hesse\_matrix[0][0] \* hesse\_matrix[1][1] - (hesse\_matrix[0][1] \* hesse\_matrix[1][0]))  
  
 print(f'Минор №1 = {minor\_1}')  
 print(f'Минор №2 = {hesse\_matrix[0][0]} \* {hesse\_matrix[1][1]} -'  
 f' ({hesse\_matrix[0][1]} \* {hesse\_matrix[1][0]}) = {minor\_2}')  
  
 if minor\_1 >= 0 and minor\_2 >= 0:  
 print('Так как знаки угловых миноров строго положительны,\n'  
 'то согласно критерию Сильвестра матрица Гессе положительна определена.')  
 return True  
 else:  
 print('страх')  
 exit(123)  
 return False

Листинг Д.9 – Метод расчёта направления спуска по формуле Ньютона

def determining\_direction\_descent(mass, hesse\_matrix, grad):  
 print(f'\nСледовательно, направление спуска определяем по формуле Ньютона')  
 print(f'p({len(mass) - 1}) = - 𝐻−1(x({len(mass) - 1})) \* 𝛻f(x({len(mass) - 1}))')  
 matrix\_grad = np.array([grad[0], grad[1]])  
 mass\_to\_inv = np.array([[hesse\_matrix[0][0], hesse\_matrix[0][1]], [hesse\_matrix[1][0], hesse\_matrix[1][1]]])  
 mass\_to\_inv = inv(mass\_to\_inv)  
 for i in range(len(mass\_to\_inv)):  
 for j in range(len(mass\_to\_inv[i])):  
 mass\_to\_inv[i][j] = round\_value(mass\_to\_inv[i][j])  
 p = -(np.dot(mass\_to\_inv, matrix\_grad))  
 print(f'p({len(mass) - 1}) = | {p[0]} {p[1]} |')  
 return p

Листинг Д.10 – Метод расчёта координат новой точки

def calculating\_coordinates\_new\_point(mass, p):  
 matrix\_start\_point = np.array([mass[-1][0], mass[-1][1]])  
 value = matrix\_start\_point + p  
 new\_coordinate = [round\_value(value[0]), round\_value(value[1]),  
 count\_target\_function(round\_value(value[0]), round\_value(value[1]))]  
  
 print(f'Вычислим координаты точки x({len(mass)})')  
 print(f'x({len(mass)}) = x({len(mass) - 1}) + p({len(mass) - 1}))')  
 print(f'x({len(mass)}) = {matrix\_start\_point} − {p}')  
 print(f'Новая координата со значением [{new\_coordinate[0]}; {new\_coordinate[1]}]')  
 print(f'Целевая функция: {new\_coordinate[-1]}\n\n')  
 mass.append(new\_coordinate)

Листинг Д.11 – Метод расчёта условия окончаний поиска

def condition\_end\_search(mass):  
 print(f'Проверим условие окончания поиска.\n'  
 f'Для этого вычислим вектор градиента ∇f(x{len(mass) - 1}) в точке'  
 f' x({len(mass) - 1})\n')  
 mass\_grad.append(count\_grad\_target\_function(mass[-1][0], mass[-1][1]))  
 print(f'Градиент функции ∇f(x{len(mass) - 1}): [{mass\_grad[-1][0]}; {mass\_grad[-1][1]}]\n')  
 return find\_norm\_gradient\_vector(mass, mass\_grad[-1], e)

Листинг Д.12 – Метод расчёта условия окончаний поиска

def find\_norm\_gradient\_vector(mass, grad, e):  
 print(f'Найдем норму вектора градиента x{len(mass) - 1}')  
 norm = round\_value(math.sqrt((grad[0] \*\* 2) + (grad[1] \*\* 2)))  
 print(f'norm = √(({grad[0]}\*\*2) + ({grad[1]} \*\* 2)) = {norm}')  
 if norm >= e:  
 print(f'norm = {norm} >= {e}')  
 print('Итерации продолжаются.\n\n\n')  
 return False  
 else:  
 print(f'norm = {norm} < {e}')  
 print('Итерации завершаются.\n\n\n')  
 return -404

Листинг Д.13 – Метод расчёта условия окончаний поиска

def out\_graph(data):  
 import matplotlib.pyplot as plt  
 import numpy as np  
  
 fig = plt.figure(figsize=(16, 9))  
 ax = fig.add\_subplot(projection='3d')  
  
 mass\_x = []  
 mass\_y = []  
 value\_func = []  
 for i in range(-len(data), 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
 colors = np.arange(len(mass\_x))  
 col\_1 = ax.scatter(mass\_x, mass\_y, value\_func, cmap="jet", c=colors)  
 ax.set\_xlabel('Значения первой переменной х(1)')  
 ax.set\_ylabel('Значения второй переменной х(2)')  
 ax.set\_zlabel('Значения целевой функции f(x)')  
 plt.colorbar(col\_1)  
 plt.savefig("static/plt\_1.png")

Продолжение Листинга Д.13

fig2 = plt.figure(figsize=(16, 9))  
ax\_2 = fig2.add\_subplot()  
mass\_x = []  
mass\_y = []  
value\_func = []  
for i in range(-2, 0):  
 mass\_x.append(data[i][0])  
 mass\_y.append(data[i][1])  
 value\_func.append(data[i][-1])  
colors = np.arange(len(mass\_x))  
col = ax\_2.scatter(mass\_x, mass\_y, cmap="jet", c=colors)  
ax\_2.set\_xlabel('Значения первой переменной х(1)')  
ax\_2.set\_ylabel('Значения второй переменной х(2)')  
plt.colorbar(col)  
ax\_2.grid()  
plt.savefig("static/plt\_2.png")