МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий
Кафедра «Прикладная математика и информатика»
Дисциплина «Методы оптимизации»

КУРСОВАЯ РАБОТА

Тема «Программная реализация методов оптимизации: метод Пауэлла, задача коммивояжера, алгоритм Прима»

Выполнил студент _____ /_____ В. Д. Шувалова_____/
подпись инициалы, фамилия

Курс 3 Группа ПМбд-31

Направление/ специальность 01.03.04 Прикладная математика

Руководитель доцент кафедры ПМИ, к.т.н., доцент
должность, ученая степень, ученое звание
Алексеева Венера Арифзяновна
фамилия, имя, отчество

Дата сдачи:
«____»_____20____г.

Дата защиты:
«____»_____20____г.

Оценка: _____

Ульяновск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий
Кафедра «Прикладная математика и информатика»
Дисциплина «Методы оптимизации»

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

студенту ПМбд-31 Шувалова В. Д.
группа фамилия, инициалы

Тема проекта (работы) «Программная реализация методов оптимизации: метод Пауэлла, задача коммивояжера, алгоритм Прима»

Срок сдачи законченной работы «___» _____20____г.

Исходные данные к работе Разработать программу для реализации методов оптимизации: метода Пауэлла, задачи коммивояжера, алгоритма Прима. Проверить работу программ на 3-х различных примерах. В программе предусмотреть ввод начального интервала и требуемой точности.
(базовое предприятие, характер курсового проекта (работы): задание кафедры, инициативная НИР, рекомендуемая литература, материалы практики)

Содержание пояснительной записки (перечень подлежащих разработке вопросов)
Введение 1. Теоретическая часть 1.1. Методы оптимизации 1.2. Метод Пауэлла 1.3. Задача коммивояжёра 1.4. Алгоритм Прима 2. Практическая часть 2.1. Описание работы программы для пользователя 2.2. Краткий обзор структуры и функций кода Заключение Список литературы Приложение

Перечень графического материала (с точным указанием обязательных чертежей)
_____

Руководитель доцент кафедры ПМИ _____ /В.А. Алексеева /
должность подпись инициалы, фамилия

«_____»_____20___г.

Студент _____ /_____В. Д. Шувалова_____/
подпись инициалы, фамилия

«_____»_____20___г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ОТЗЫВ
руководителя на курсовую работу

студента Шуваловой Валерии Дмитриевны
фамилия, имя и отчество

Факультет информационных систем и технологий группа ПМбд-31 курс 3
Дисциплина Методы оптимизации
Тема работы «Программная реализация методов оптимизации: метод Пауэлла, задача коммивояжера, алгоритм Прима»

Курсовая работа Шуваловой В.Д. рассматривает такие методы оптимизации как метод Пауэлла, алгоритм Прима, а также рассматривает такую задачу оптимизации как задача коммивояжера.

Целью курсовой работы является программная реализация указанных методов и задач оптимизации.

Для достижения этой цели были поставлены следующие задачи: изучить научную литературу, учебные пособия и электронные ресурсы, в которых рассматривается метод Пауэлла, задача коммивояжера и алгоритм Прима; программно реализовать метод Пауэлла, алгоритм Прима и любой метод решения задачи коммивояжёра; сделать выводы о достоинствах и недостатках указанных методов.

В результате была разработана программная реализация указанных методов с использованием языка программирования Python версии 3.12 и среды разработки PyCharm Community Edition версии 2024. При помощи программы были получены результаты решения различных задач.

Часть темы курсовой работы Шувалова В.Д. выбрала самостоятельно, что показывает заинтересованность в решении данной проблемы. При выполнении курсовой работы сообщала, на каком этапе реализации находится, консультировалась по интересующим вопросам. Студентка привела достаточно подробный теоретический материал и продемонстрировала его знание, а также продемонстрировала навыки программирования на языке программирования Python, знание структуры языка и алгоритмов.

Структура и содержание курсовой работы Шуваловой В.Д. полностью соответствуют заданию. Работа выполнена грамотно, написана понятным языком.

Руководитель доцент кафедры ПМИ, к.т.н., доцент _____ / В.А. Алексеева
должность, учёная степень, ученое звание подпись инициалы, фамилия

«_____»_____20____г.

Оглавление

Введение

В современной математике и информатике методы оптимизации занимают важное место, так как они позволяют находить наилучшие решения в различных задачах, начиная от минимизации функций и заканчивая оптимизацией сложных систем. Оптимизация является ключевым инструментом в таких областях, как машинное обучение, исследование операций, экономика, инженерия и многих других.

Одним из классических методов оптимизации является метод Пауэлла, который используется для поиска минимума функции. Он находит своё применение в физике, биологии и других науках, где требуется поиск экстремумов функции. Задача коммивояжера, в свою очередь, является одной из самых известных задач комбинаторной оптимизации, которая имеет множество практических применений, включая планирование маршрутов и управление цепочками поставок. Алгоритм Прима, относящийся к классу жадных алгоритмов, широко применяется для построения минимальных остовных деревьев в графах, что актуально в проектировании сетей и инфраструктуры.

Целью курсовой работы является программная реализация методов оптимизации: метод Пауэлла, задача коммивояжера, алгоритм Прима.

Для достижения этой цели были поставлены следующие задачи:

1. Изучить научную литературу, учебные пособия и электронные ресурсы, в которых рассматривается метод Пауэлла, задача коммивояжера и алгоритм Прима.

2. Программно реализовать метод Пауэлла, алгоритм Прима и любой метод решения задачи коммивояжёра.

3. Сделать выводы о достоинствах и недостатках рассмотренных методов.

Курсовая работа состоит из двух глав. В теоретической части описываются основные идеи метода Пауэлла для одномерной минимизации

функции, задачи коммивояжёра и полного перебора как метода её решения, алгоритма Прима. В практической части представлена реализация программы.

Код программы написан на языке программирования Python версии 3.12 с использованием среды разработки PyCharm Community Edition версии 2024 и прикреплён в приложении.

1. Теоретическая часть

1.1. Методы оптимизации

Оптимизация является одной из ключевых задач в математике и её приложениях, поскольку позволяет находить наилучшие решения в условиях ограниченных ресурсов или заданных критериев. Для решения задачи оптимизации необходимо построить её математическую модель, которая описывает объект или процесс с помощью соотношений между величинами, характеризующими его свойства. Основным критерием оптимальности выступает требование достижения экстремального значения (максимума или минимума) одной или нескольких функций, называемых целевыми [1].

Если целевая функция единственная, то задача относится к классу задач математического программирования. В случае, когда целевых функций несколько, задача относится к классу задач векторной оптимизации. В зависимости от количества переменных, от которых зависит целевая функция, задачи оптимизации делятся на одномерные и многомерные. В данной работе будет рассмотрен метод Пауэлла для одномерной оптимизации.

Также выделяют дискретную оптимизацию – область теории оптимизации, которая заключается в поиске оптимального решения среди конечного множества возможных решений [10]. Подразделом дискретной оптимизации является комбинаторная оптимизация, которая фокусируется на задачах, где решение представляет собой комбинацию элементов из конечного множества. Одной из классических задач комбинаторной оптимизации является задача коммивояжера, которая будет рассмотрена в данной работе. Также важной проблемой является задача о нахождении минимального остовного дерева, для решения которой далее будет рассмотрен алгоритм Прима.

1.2. Метод Пауэлла

Метод Пауэлла относится к методам полиномиальной аппроксимации. Общая идея этих методов состоит в том, что исходная функция

аппроксимируется полиномом какого-то порядка на некотором начальном интервале, который содержит минимум исходной функции. В качестве точки минимума исходной функции принимается точка минимума аппроксимируемого полинома. Тогда всё сводится к поиску точки минимума используемого полинома [6].

Чтобы повысить точность нахождения точки минимума, можно использовать два подхода: увеличить степень полинома или уменьшить начальный интервал. Второй способ является более предпочтительным, так как построение построение полинома выше третьего порядка – достаточно сложная задача, а сужение интервала – достаточно простая. Таким образом, на практике чаще всего используют полиномы второго и третьего порядков, то есть квадратичную и кубическую аппроксимацию [8].

Метод Пауэлла использует квадратичную аппроксимацию. Для применения данного метода рассматриваемая функция должна быть непрерывной и унимодальной на рассматриваемом отрезке, то есть должна иметь только один локальный минимум (или максимум). Это важно, потому что квадратичная аппроксимация предполагает, что функция ведёт себя как парабола на рассматриваемом отрезке, то есть имеет один экстремум. Также функция должна быть гладкой на рассматриваемом отрезке. Это означает, что функция не должна иметь резких изломов, разрывов, которые могут нарушить аппроксимацию параболой, поскольку не свойственны ей.

Квадратичная аппроксимация заключается в аппроксимации функции полиномом второго порядка. Для её построения достаточно трёх точек $M_1(x_1;\ y_1), M_2(x_2;\ y_2), M_3(x_3;\ y_3)$. Тогда полином записывается в виде: $P_2(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2)$. Коэффициенты $a_0,\ a_1,\ a_2$ выбираются так, чтобы выполнялось $P_2(x_1) = y_1,\ P_2(x_2) = y_2,\ P_2(x_3) = y_3$. Тогда они находятся по формулам:

$$a_0 = y_1, a_1 = \frac{y_2 - y_1}{x_2 - x_1}, a_2 = \frac{1}{x_3 - x_2}\left(\frac{y_3 - y_1}{x_3 - x_1} - \frac{y_2 - y_1}{x_2 - x_1}\right) \qquad (*)$$

Вершина параболы, называемая стационарной точкой, находится по формуле:

$$x^* = \frac{x_2 + x_1}{2} - \frac{a_1}{2a_2} \qquad (**)$$

Очевидно, что стационарная точка является оценкой искомого минимума.

Метод Пауэлла заключается в итеративном применении квадратичной аппроксимации, пока не будет достигнута заданная точность.

Ход метода Пауэлла:

1) Задать x_1 и шаг $h > 0$, точность $\varepsilon > 0$. Можно придать им следующие значения: $x_1 = a, h = \varepsilon$, где a – начало отрезка аппроксимации $[a; b]$.
2) Найти $x_2 = x_1 + h$, вычислить $f(x_1)$ и $f(x_2)$.
3) Если $f(x_1) > f(x_2)$, то $x_3 = x_1 + 2h$, иначе $x_3 = x_1 - h$.
4) Вычислить $f(x_3)$; определить $F_{min} = \min\{f(x_1), f(x_2), f(x_3)\}$, определить соответствующее x_{min}.
5) Найти x^* по формулам $(*), (**)$.
6) Если $|x^* - x_{min}| < \varepsilon$, то поиск окончен и x^* является точкой минимума. Иначе, если данное неравенство не выполняется, перейти к шагу 7.
7) Вычислить $f(x^*)$, если $f(x^*) < F_{min}$, то $x_1 = x^*$, иначе $x_1 = x_{min}$. Перейти к шагу 2 [8].

1.3. Задача коммивояжёра

Задача коммивояжёра – это одна из самых известных задач комбинаторной оптимизации. Название связано с её историческим происхождением: в 19-м и 20-м веке по городам ездили коммивояжёры (сейчас их называют торговыми представителями), которые предлагали людям купить товары.

Общая постановка задачи заключается в поиске самого выгодного маршрута, проходящего через указанные города, с соблюдением дополнительных условий. Выгодность маршрута определяется контекстом задачи: кратчайший, самый дешёвый и т. д. Дополнительными условиями могут быть лимиты на посещение городов, например, каждый город нужно посетить не менее одного раза, но не более трёх раз. Также могут быть заданы зависимости между городами: нужно отвезти документы в один город, но

перед этим забрать их в другом. Чем больше таких критериев, тем сложнее задача.

Классическая формулировка задачи коммивояжёра: найти самый короткий маршрут, проходящий через все города по одному разу с возвратом в город, с которого начали (то есть стартовый город посетить в итоге два раза). Именно эта формулировка будет рассматриваться далее.

Задача коммивояжёра относится к числу трансвычислительных задач, поскольку уже при небольшом количестве городов она не может быть решена методом перебора за адекватное время: даже самые мощные компьютеры будут решать её несколько миллиардов лет. Это объясняется её алгоритмической сложностью. Пусть есть n городов, коммивояжёр оказывается сначала в стартовом городе. Потом он встаёт перед выбором следующего города, который ещё не посетил, то есть имеет $(n-1)$ вариантов городов, далее - $(n-2)$ вариантов и т. д. Таким образом полный перебор всех маршрутов содержит $n!$ вариантов. То есть асимптотическая сложность представляет собой факториал - $O(n!)$. Таким образом, теоретический предел для компьютера – 66 городов. Если их будет 67, то число возможных комбинаций маршрутов будет равно $36 * 10^{93}$ – это больше предела Бремерманна. Он задаёт максимальную скорость вычислений автономной системы в материальной вселенной. Его суть заключается в том, что если представить компьютер размером с нашу планету, который будет работать столько, сколько она существует, то за всё время он сможет обработать 10^{93} бит информации, что в 36 раз меньше, чем требуется рассчитать при 67 городах [7].

Существует много методов решения данной задачи: точные и приближённые, доказанные и эвристические. В данной работе использован метод полного перебора (метод грубой силы – brute force), который даёт точное решение, но из-за факториальной сложности применяется для очень небольшого количества городов. Его суть проста и заключается в переборе всех возможных маршрутов коммивояжёра, удовлетворяющих заданным

условиям. Затем из них выбирается самый оптимальный маршрут или маршруты, если таких несколько. Для классической формулировки задачи оптимальным маршрутом является самый короткий, а условие заключается в поиске маршрута, проходящего через все города с возвратом в исходный город [5].

1.4. Алгоритм Прима

Алгоритм Прима решает задачу о нахождении минимального остовного дерева, которая относится к области комбинаторной оптимизации. Минимальное остовное дерево (минимальный остов) в связном взвешенном неориентированном графе – это такое поддерево этого графа, которое соединяет все его вершины и обладает наименьшим возможным весом, то есть суммой весов рёбер. Легко понять, что если в графе n рёбер, то остов будет содержать $n-1$ ребро. На рисунке ниже приведён пример минимального остова (рёбра графа, входящие в остов, покрашены в зелёный цвет):

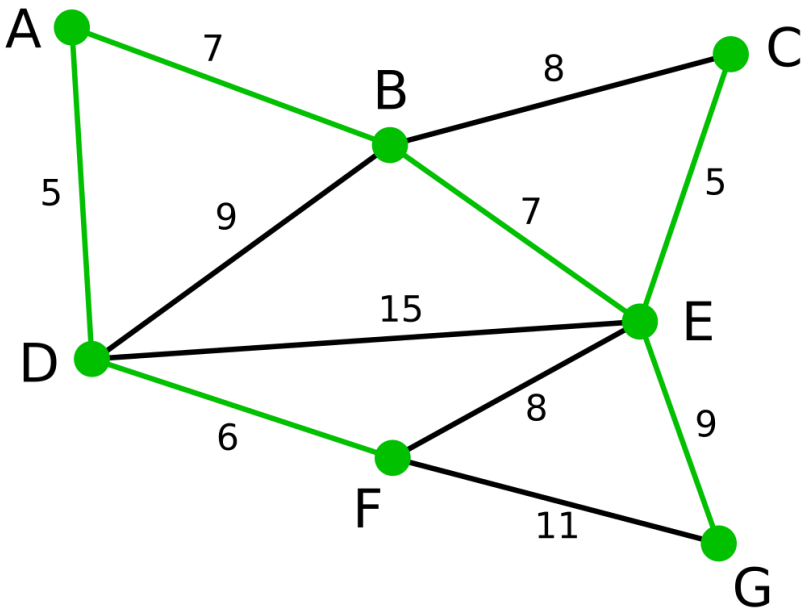

Рисунок 1. Пример минимального остовного дерева.

В естественной постановке задача о нахождении минимального остова звучит так: есть n городов, и для каждой пары известна стоимость соединения

их дорогой, либо известно, что соединить их нельзя. Требуется соединить все города так, чтобы можно было доехать из одного города в другой, а при этом стоимость прокладки дорог была минимальной.

Для решения данной задачи существует несколько методов, в данной работе будет рассмотрен один из них – алгоритм Прима. Этот алгоритм назван в честь американского математика Роберта Прима, который открыл этот алгоритм в 1957 году. Впрочем, иногда этот метод называют алгоритмом Дейкстры, так как он изобрёл его в 1959 году независимо от Прима [3].

Алгоритм Прима имеет простой вид:

1) Изначально остов полагается состоящим из одной вершины, её можно выбрать произвольно. Далее повторять шаг 2 до тех пор, пока не будет найден минимальный остов, то есть пока не будет найдено $n - 1$ рёбер.
2) Для текущего остова выбирается ребро минимального веса, исходящее из какой-нибудь вершины текущего остова в вершину, которая ещё не добавлена в остов. Это ребро добавляется в остов [9].

2. Практическая часть

В данной части сначала будет представлено описание работы программы для пользователя, чтобы он смог её запустить и использовать. Затем – краткий обзор структуры и функций кода. Он полезен для тех, кто хочет разобраться в коде, использовать его в своих проектах, доработать.

2.1. Описание работы программы для пользователя

Программа представлена тремя файлами, каждый из которых относится к своей задаче или методу оптимизации: powell_method.exe – к методу Пауэлла, travelling_salesman_problem.exe – к задаче коммивояжёра, prim_algorithm.exe – к алгоритму Прима соответственно. Для начала работы с программой пользователю необходимо запустить исполняемый файл с расширением .exe. Опишем работу с файлом powell_method.exe, для остальных файлов работа с ними аналогична.

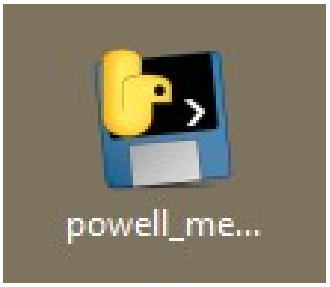

Рисунок 2. Запуск файла powell_method.exe.

После запуска файла пользователю будет предложено выбрать одну из трёх функций, для которой будет искаться точка минимума методом Пауэлла. Ему необходимо будет ввести номер выбранной функции и нажать Enter:

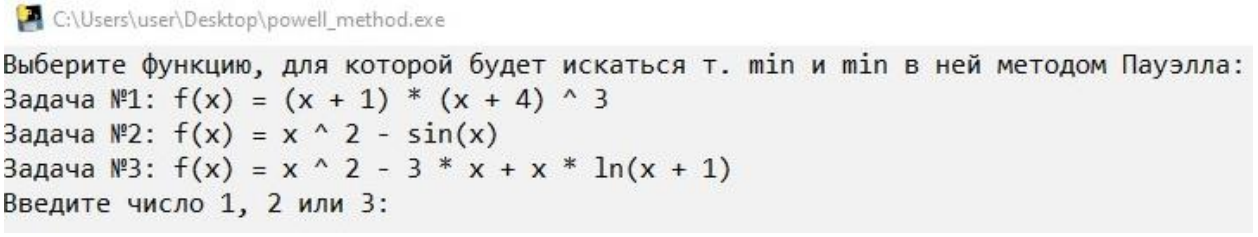

Рисунок 3. Приглашение ко вводу номера выбранной функции от пользователя.

Если пользователь введёт не целое число или число отличное от 1, 2 или 3, то ему будет выведено соответствующее сообщение и приглашение к повторному вводу:

```
Введите число 1, 2 или 3: f
Вы ввели не целое число - введите ещё раз
Введите число 1, 2 или 3: 3.0
Вы ввели не целое число - введите ещё раз
Введите число 1, 2 или 3: 5
Вы ввели число, отличное от 1, 2 или 3 - введите ещё раз
Введите число 1, 2 или 3:
```

Рисунок 4. Некорректный ввод номера функции от пользователя.

Приглашения ко вводу будут повторяться до тех пор, пока пользователь не введёт корректное число. Когда это произойдет, будет выведено сообщение с просьбой ввести начальный отрезок, и сразу после этого появится второе окно с графиком исходной функции, чтобы пользователь смог примерно оценить, где находится минимум и выбрать начальный отрезок. При наведении курсора на график в правом нижнем углу будут отображаться координаты точки наведения, чтобы пользователь смог ориентироваться не только на ось при выборе отрезка.

```
Введите число 1, 2 или 3: 1
Введите начальный отрезок [a; b]:
```

Рисунок 5. Приглашение ко вводу начального отрезка.

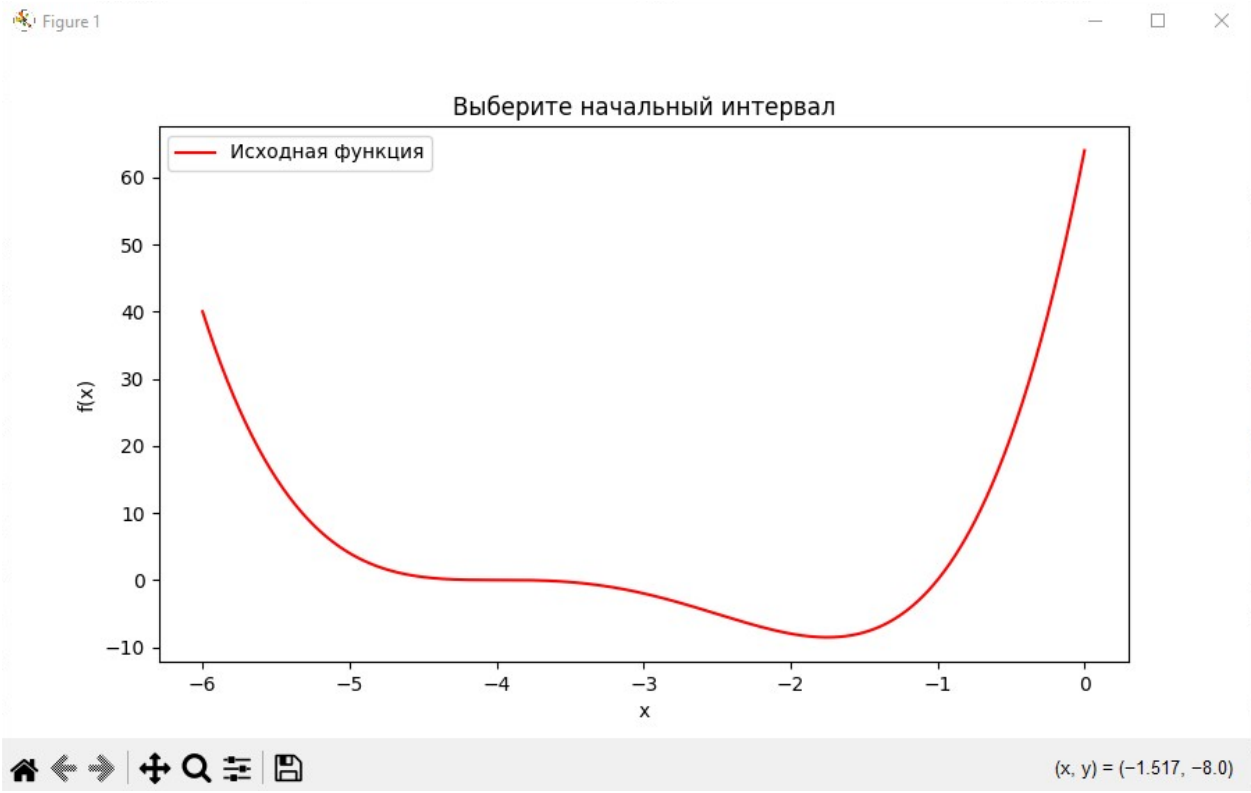

Рисунок 6. График исходной функции для выбора начального отрезка.

Ввести начальный отрезок можно будет только после закрытия окна с графиком. Тогда сначала появится приглашение ввести начало отрезка a, затем – конец отрезка b. Пользователь может вводить как целые, так и вещественные числа, только дробную часть в вещественных числах необходимо писать через точку, а не через запятую. Также начало отрезка должно быть строго меньше его конца. Отрезок должен быть внутри области определения функции и функция должна пройти проверку на унимодальность на нём. При невыполнении хотя бы одного их вышеперечисленных критериев пользователю будут выводиться соответствующие ошибки, и программа будет продолжать выводить приглашения ко вводу. Они будут повторяться до тех пор, пока пользователь не осуществит корректный ввод.

```
Введите начальный отрезок [a; b]:
a = -2,5
Некорректный ввод: нужно вводить числа и не использовать запятую
a = -2.0
b = f
Некорректный ввод: нужно вводить числа и не использовать запятую
a = 3
b = 2
Некорректный ввод: a должно быть меньше b
a = -3
b = -1
Функция не унимодальна на данном отрезке: повторите ввод
a = -2.0
b = -1
```

Рисунок 7. Некорректный ввод начального отрезка от пользователя.

Далее пользователю необходимо ввести точность вычислений.

```
Введите точность eps от 10^-13 до 10^-1
в формате [0.0...0[ненулевое число]],
например: eps = 0.001

Для точности eps < 10^-13
решение задачи невозможно в силу
ограниченных вычислительных возможностей Python.
eps =
```

Рисунок 8. Приглашение ко вводу точности.

Пользователь может вводить как целые, так и вещественные числа, только дробную часть в вещественных числах необходимо писать через точку, а не через запятую. Также точность должна принадлежать диапазону $[10^{-13}; 10^{-1}]$. Нижняя граница диапазона равна 10^{-13}, так как вычислительные возможности Python ограничены, как и у любого другого языка программирования. Если превысить эту точность, то начнут накапливаться ошибки округления и погрешности вычислений, в результате программа будет работать некорректно. Также пользователь должен вводить число в формате 0.0 ... [ненулевое число]. При невыполнении хотя бы одного

их вышеперечисленных критериев пользователю будут выводиться соответствующие ошибки, и программа будет продолжать выводить приглашения ко вводу. Они будут повторяться до тех пор, пока пользователь не осуществит корректный ввод.

```
eps = f
Вы ввели не число или использовали запятую - введите ещё раз
eps =
Вы ввели не число или использовали запятую - введите ещё раз
eps = 0,001
Вы ввели не число или использовали запятую - введите ещё раз
eps = 1
Вы ввели число вне диапазона [10^-13; 10^-1] - введите ещё раз
eps = 0.5
Вы ввели число вне диапазона [10^-13; 10^-1] - введите ещё раз
eps = 0.00000000000000000000000000000000000000000000001
Вы ввели число вне диапазона [10^-13; 10^-1] - введите ещё раз
eps = 0.00012
Вы ввели число не в формате [0.0...0[ненулевое число]] - введите ещё раз
eps = 0.00010
Вы ввели число не в формате [0.0...0[ненулевое число]] - введите ещё раз
eps = 0.0010001
Вы ввели число не в формате [0.0...0[ненулевое число]] - введите ещё раз
eps = 0.001
```

Рисунок 9. Некорректный ввод точности от пользователя.

Далее будет выведена полная таблица расчётов и ответ, а также появится второе окно с графиком функции на выбранном отрезке и отображённой на нём найденной точкой минимума.

```
Полная таблица расчётов:
+---+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+----------+--------+--------+-------+
| n |   x1   |   x2   |   x3   | f(x1)  | f(x2)  | f(x3)  |   a0   |   a1   |   a2   | x_min  | f(x_min) |   x*   | f(x*)  | eps_n |
+---+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+----------+--------+--------+-------+
| 1 | -2.000 | -1.999 | -1.998 | -8.000 | -8.004 | -8.008 | -8.000 | -3.994 | 6.015  | -1.998 |  -8.008  | -1.667 | -8.471 | 0.331 |
| 2 | -1.667 | -1.666 | -1.668 | -8.471 | -8.469 | -8.472 | -8.471 | 1.807  | 11.651 | -1.668 |  -8.472  | -1.745 | -8.543 | 0.076 |
| 3 | -1.745 | -1.744 | -1.746 | -8.543 | -8.543 | -8.543 | -8.543 | 0.121  | 10.223 | -1.746 |  -8.543  | -1.750 | -8.543 | 0.004 |
| 4 | -1.750 | -1.749 | -1.751 | -8.543 | -8.543 | -8.543 | -8.543 | 0.011  | 10.125 | -1.750 |  -8.543  | -1.750 | -8.543 | 0.000 |
+---+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+----------+--------+--------+-------+

Итог: точка минимума x_min = -1.750 и минимум f_min = -8.543
```

Рисунок 10. Полная таблица расчётов и точка минимума.

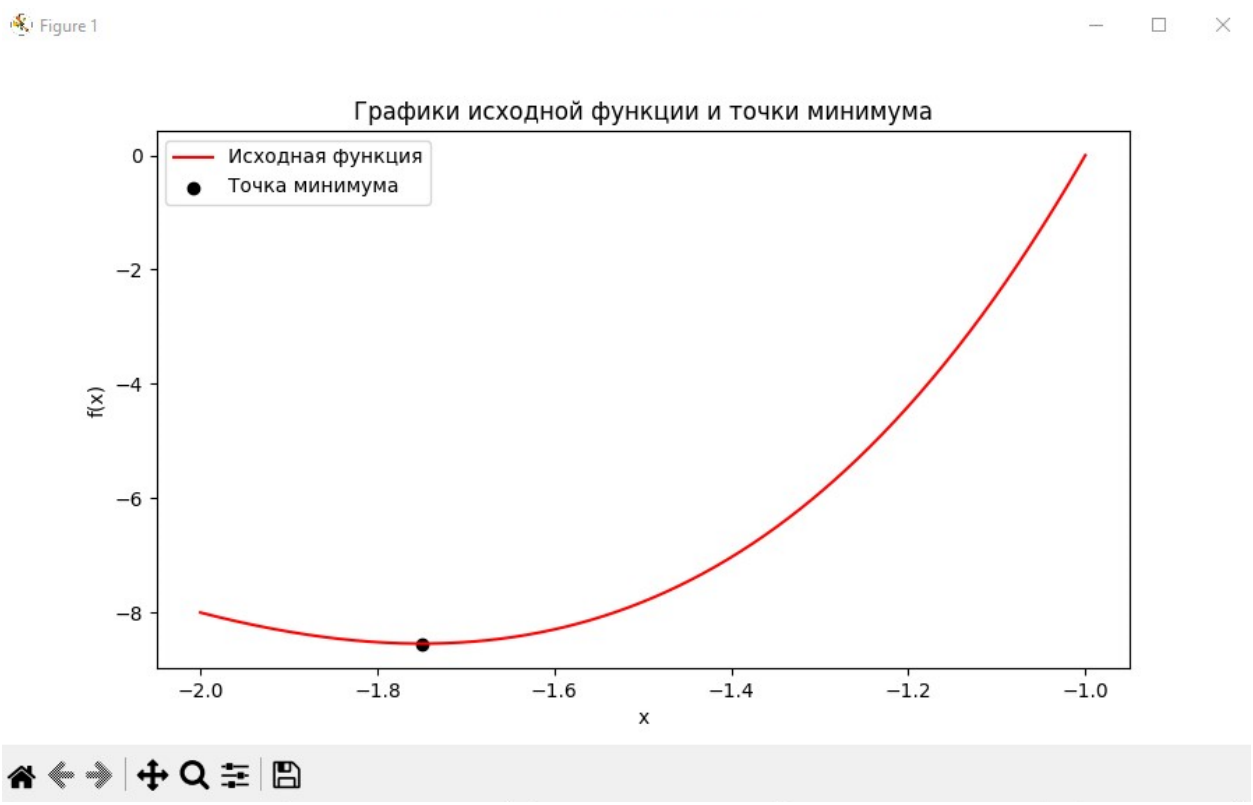

Рисунок 11. График исходной функции и точки минимума.

Чтобы закрыть программу, пользователь может нажать на крестик в углу окна с ответами или нажать Enter после закрытия окна с графиком.

2.2. Краткий обзор структуры и функций кода

Код написан на языке программирования Python версии 3.12 с использованием среды разработки PyCharm Community Edition версии 2024. Для обеспечения совместимости кода можно использовать любую среду разработки, однако версия Python не должна быть ниже 3.8, иначе некоторые синтаксические конструкции, например f-строки, будут работать некорректно.

Программная реализация методов оптимизации состоит из трёх файлов: powell_method.py, travelling_salesman_problem.py, prim_algorithm.py. Каждый из них содержит код относящийся к своей задаче или методу оптимизации: метод Пауэлла, задача коммивояжёра, алгоритм Прима соответственно.

Структура каждого файла одинакова. Сначала подключаются необходимые модули и библиотеки:

```
# импортируем необходимые библиотеки
import math, decimal, numpy as np, matplotlib.pyplot as plt
from decimal import Decimal
from prettytable import PrettyTable
```

Рисунок 12. Подключение библиотек для файла powell_method.py.

Модуль – это файл с кодом, который содержит готовые инструменты, которые можно использовать в своих программах. Библиотека – это набор связанных модулей, предоставляющих более широкий функционал. Далее будут описаны модули и библиотеки, которые используются в файлах с кодом.

Модуль math является встроенным в стандартную библиотеку Python. Он предоставляет доступ к различным математическим функциям и константам. Модуль decimal также является встроенным, он используется для более точной работы с вещественными числами. Модуль prettytable является сторонним модулем, который предоставляет инструменты для форматирования и отображения табличных данных. Библиотека numpy является сторонней библиотекой, которая предоставляет поддержку многомерных массивов и высокоуровненые математические функции, работающие с этими массивами. Библиотека matplotlib также является сторонней, она используется для визуализации данных: создания графиков и диаграмм.

Встроенные модули не требуют установки для их использования. Сторонние же модули и библиотеки нужно устанавливать. Далее будет описан процесс установки сторонних инструментов для среды PyCharm Community Edition. Для других сред разработки, например, Visual Studio Code, процесс может отличаться.

В среде разработки PyCharm Community Edition создаётся проект с выбранной версией интерпретатора Python (в данном случае версией 3.12) и с использованием виртуального окружения venv. Виртуальное окружение позволяет устанавливать сторонние модули и библиотеки. Кроме того, виртуальное окружение создаёт изолированную среду. Это значит, что

библиотеки и модули устанавливаются только в venv только для этого проекта и не затрагивают другие проекты. Это позволяет избежать проблем с конфликтом разных версий инструментов в разных проектах. Для установки сторонних инструментов используется пакетный менеджер pip: в терминале прописывается команда: pip install название_библиотеки. После этого библиотеку (или модуль) можно использовать. Обычный пользователь избавлен от необходимости это делать, так как пользуется созданным исполняемым файлом с расширением .exe для каждого файла с кодом: powell_method.exe, travelling_salesman_problem.exe, prim_algorithm.exe соответственно. Но если в программе требуется что-то менять, то необходимо осуществить эту установку любым доступным способом.

Структура каждого файла с кодом состоит из функций: главной функции main, вспомогательных функций и основной логической функции, которая реализует метод оптимизации: powell_method() для powell_method.py, complete_search_of_all_ways() для travelling_salesman_problem.py и prim_algorithm() для prim_algorithm.py. Функция – это блок кода, который выполняет определённую задачу и может быть вызван в различных частях программы. Они повышают читаемость кода, разделяя его на составляющие.

Опишем функции для файла powell_method.py, для остальных файлов структура аналогична.

```
def f_1(x):...
3 usages
def f_2(x):...
2 usages
def f_3(x):...
1 usage
def check_unimodality(func, a, b):...
4 usages
def graph_function(func, left, right, x_min = None, f_min = None):...
1 usage
def get_function():...
1 usage
def get_interval(func):...
1 usage
def get_accuracy():...
1 usage
def powell_method(f, a, b, eps, eps_signs):...
1 usage
def main():...
main()
input()  # чтобы консоль не закрылась
```

Рисунок 13. Схема всех функций для файла powell_method.py.

Функции f_1(), f_2(), f_3() предоставляют задачи на выбор – функции для минимизации. Функция check_unimodality() проверяет унимодальность функции на указанном отрезке. Функция graph_function() рисует график исходной функции, а также добавляет на него найденную точку минимума. Функция get_interval() получает от пользователя отрезок, на котором рассматривается функция и ищется точка минимума. Функция get_accuracy() запрашивает у пользователя точность. Все вышеперечисленные функции являются вспомогательными. Основная логическая функция powell_method() реализуем метод Пауэлла, используя переданные ей аргументы: f – функция, a – начало отрезка, b – конец отрезка, eps – точность, eps_signs – количество знаков после точки, которые нужно вывести в ответе. Основная функция main() вызывает все вышеперечисленные методы, чтобы запросить от

пользователя входные данные, вызвать функцию powell_method() для решения задачи и вывести ответ. В самом конце программы main() вызывается – это точка входа программы. Последняя строчка input() нужна для того, чтобы исполняемый файл с расширением .exe, созданный из исходного файла .py, не закрывался сразу после вывода результатов.

```
def get_problem():...
1 usage
def get_start_vertex(n):...
2 usages
def complete_search_of_all_ways(n, adjacency_matrix, start_v, all_ways, cur_v,
                                total_v, total_dist, total_way, used_v):...
1 usage
def main():...
main()
input()  # чтобы консоль не закрылась
```

Рисунок 14. Схема всех функций для файла travelling_salesman_problem.py.

```
def get_problem():...
1 usage
def draw_graph(n, m, all_edges, chosen_edges):...
1 usage
def prim_algorithm(n, m, edge_list):...
1 usage
def main():...
main()
input()  # чтобы консоль не закрылась
```

Рисунок 15. Схема всех функций для файла prim_algorithm.py.

Заключение

Цель и задачи данной курсовой работы выполнены. Были изучены такие методы оптимизации как метод одномерной оптимизации Пауэлла для нахождения точки минимума функции, метод полного перебора для решения задачи коммивояжёра, а также алгоритм Прима для решения задачи о поиске минимального остовного дерева в графе. На языке программирования Python была написана программная реализация указанных методов оптимизации. Каждый из этих методов имеет свои достоинства и недостатки, которые определяют их применимость в различных задачах.

Метод Пауэлла не требует вычисления производных функции, что делает его применимым для функций, у которых производные сложно вычислить или они недоступны. Однако на него накладываются определённые ограничения из-за квадратичной аппроксимации функции: она должна иметь только один локальный экстремум и должна быть гладкой на рассматриваемом отрезке. Это важно, потому что квадратичная аппроксимация предполагает, что функция ведёт себя как парабола на рассматриваемом отрезке, то есть она должна иметь один экстремум и не должна иметь резких изломов и разрывов, которые ей не свойственны.

Метод полного перебора для решения задачи коммивояжёра гарантирует нахождение точного решения задачи и прост в реализации, но его асимптотическая сложность представляет собой факториал от количества городов, что делает его неприменимым для задач большой размерности. Поэтому данный метод подходит только для задач, где количество городов невелико.

Алгоритм Прима относится к классу жадных алгоритмов, а потому имеет простую идею и реализацию, а также гарантирует точное нахождение минимального остова с асимптотической сложностью $O(NM)$, где N – количество вершин в графе, M – количество рёбер. Однако для плотных графов, где количество рёбер близко к квадрату вершин, алгоритм может быть менее эффективен, так как деградирует до $O(n^3)$. Поэтому его лучше

применять для разреженных графов, где количество рёбер значительно меньше квадрата количества вершин.

Дальнейшая модификация программы может заключаться в добавлении других методов для решения данных задач оптимизации, в уменьшении расхода используемой памяти с помощью использования массивов библиотеки numpy, в добавлении графического интерфейса с помощью библиотеки customtkinter.

Список литературы

1. Алексеева В. А. Специальные разделы математики: учебное пособие / В. А. Алексеева. – Ульяновск : УлГТУ, 2019. – 138 с. – ISBN 978-5-9795-1887-9.

2. Борознов В. О. Исследование решения задачи коммивояжера // Вестник АГТУ. Серия: Управление, вычислительная техника и информатика. 2009. №2. URL: https://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera (дата обращения: 25.11.2024).

3. Иванов М. Минимальное остовное дерево. Алгоритм Прима. // Сайт e-maxx.ru. – 2011. [электронный ресурс] – URL: http://e-maxx.ru/algo/mst_prim (дата обращения: 24.11.2024).

4. Кормен Т. Х. Алгоритмы: построение и анализ. / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн — М.: Вильямс, 2011. — 1296 с. – ISBN 978-5-8459-0857-5

5. Левитин А. В. Глава 3. Метод грубой силы: Задача коммивояжёра // Алгоритмы. Введение в разработку и анализ – М.: Вильямс, 2006. – 576 с. – ISBN 978-5-8459-0987-9

6. Мицель А. А. Методы оптимизации. В 2 ч. Ч. 1 : учебное пособие / А. А. Мицель, А. А. Шелестов, В. В. Романенко. – Томск : Изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2020. – 350 с. – ISBN 978-5-86889-893-8

7. Полянин М. Что такое “задача коммивояжёра” // Журнал Яндекс Практикума “Код”. – 2024. [электронный ресурс] – URL: https://thecode.media/komm/ (дата обращения: 25.11.2024).

8. Попова Т. М. Методы одномерной оптимизации: методические указания и задания к выполнению лабораторных работ по дисциплине “Методы оптимизации” / Т. М. Попова. – Хабаровск : ТОГУ, 2010. – 26 с.

9. Слотин С. Алгоритм Прима. // Сайт Алгоритмика. – 2022. [электронный ресурс] – URL: https://ru.algorithmica.org/cs/spanning-trees/prim/ (дата обращения: 20.11.2024).

10. Тюхтина А. А. Методы дискретной оптимизации: учебно-методическое пособие / А. А. Тюхтина. – Нижний Новгород : Нижегородский госуниверситет, 2014. – 62 с.

Приложение

Файл powell_method.py:

```
"""Метод Пауэлла"""

# импортируем необходимые библиотеки
import math, decimal, numpy as np, matplotlib.pyplot as plt
from decimal import Decimal
from prettytable import PrettyTable

# 3 функции на выбор:

def f_1(x):  # функция №1: f(x) = (x + 1) * (x + 4) ^ 3
    return (x + Decimal("1")) * (x + Decimal("4")) ** Decimal("3")


def f_2(x):  # функция №2: f(x) = x ^ 2 - sin(x)
    return x ** Decimal("2") - Decimal(math.sin(x))


def f_3(x):  # функция №3: f(x) = x ^ 2 - 3 * x + x * ln(x + 1)
    return x ** Decimal("2") - Decimal("3") * x + x * Decimal(math.log(x + Decimal("1")))


def check_unimodality(func, a, b):  # проверяет унимодальность функции на выбранном начальном
отрезке
    if func == f_1:
        func_dxdx = lambda x: Decimal("12") * x ** Decimal("2") + Decimal("78") * x + Decimal("120")
    elif func == f_2:
        func_dxdx = lambda x: Decimal("2") + Decimal(math.sin(x))
    else:
        func_dxdx = lambda x: ((Decimal("2") * x ** Decimal("2") + Decimal("5") * x + Decimal("4")) /
                               ((x + Decimal("1")) ** Decimal("2")))
    # функция унимодальна на [a; b], если её вторая производная >= 0 на [a, b]
    x = np.linspace(a, b, 1000)
    for i in x:
        if func_dxdx(Decimal(i)) < Decimal("0"):
            return False
    return True


def graph_function(func, left, right, x_min = None, f_min = None):
    x = np.linspace(left, right, 1000)
    y = [func(Decimal(i)) for i in x]
    plt.figure(figsize=(9, 5))
    plt.plot(x, y, color="red", label="Исходная функция")
    if x_min == f_min == None:
        title = "Выберите начальный интервал"
    else:
        title = "Графики исходной функции и точки минимума"
        plt.scatter(x_min, f_min, color="black", label="Точка минимума")
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("f(x)")
```

```
    plt.legend()
    plt.show()


def get_function():  # пользователь выбирает функцию
    print("Выберите функцию, для которой будет искаться т. min и min в ней методом Пауэлла:")
    print("Задача №1: f(x) = (x + 1) * (x + 4) ^ 3")
    print("Задача №2: f(x) = x ^ 2 - sin(x)")
    print("Задача №3: f(x) = x ^ 2 - 3 * x + x * ln(x + 1)")
    while True:
        try:
            number_choice = int(input("Введите число 1, 2 или 3: "))
        except ValueError:
            print("Вы ввели не целое число - введите ещё раз")
        else:
            if number_choice not in (1, 2, 3):
                print("Вы ввели число, отличное от 1, 2 или 3 - введите ещё раз")
            else:
                if number_choice == 1:
                    return f_1
                elif number_choice == 2:
                    return f_2
                else:
                    return f_3


def get_interval(func):  # пользователь вводит начальный отрезок
    print("Введите начальный отрезок [a; b]:")
    # выведем пользователю график исходной функции, чтобы он выбрал начальный отрезок
    if func == f_1:  # для функции №1 желательно брать отрезок [-2; -1]
        graph_function(func, -6, 0)
    elif func == f_2:  # для функции №2 желательно брать отрезок [-2; 3]
        graph_function(func, -6, 7)
    else:  # для функции №3 желательно брать отрезок [0; 2]
        graph_function(func, -0.9, 4)
    while True:
        try:
            a = float(input("a = "))
            b = float(input("b = "))
        except ValueError:
            print("Некорректный ввод: нужно вводить числа и не использовать запятую")
        else:
            if Decimal(a) >= Decimal(b):
                print("Некорректный ввод: a должно быть меньше b")
            else:
                if func == f_3 and a <= Decimal("-1"):  # проверка области определения для функции №3
                    print("Вышли за область определения функции: повторите ввод")
                elif not check_unimodality(func, a, b):
                    print("Функция не унимодальна на данном отрезке: повторите ввод")
                else:
                    return Decimal(a), Decimal(b)


def get_accuracy():  # пользователь вводит точность
```

```
    print("\nВведите точность eps от 10^-13 до 10^-1\nв формате [0.0...0[ненулевое
число]],\nнапример: eps = 0.001\n")
    print("Для точности eps < 10^-13\nрешение задачи невозможно в силу",
          "\nограниченных вычислительных возможностей Python.", sep="")
    while True:
        try:
            eps_str = input("eps = ")
            eps = Decimal(eps_str)
        except decimal.InvalidOperation:
            print("Вы ввели не число или использовали запятую - введите ещё раз")
        else:
            if not (Decimal("0.0000000000001") <= eps <= Decimal("0.1")):
                print("Вы ввели число вне диапазона [10^-13; 10^-1] - введите ещё раз")
            elif eps_str.count('0') != (len(eps_str) - 2) or eps_str[-1] == "0":
                print("Вы ввели число не в формате [0.0...0[ненулевое число]] - введите ещё раз")
            else:
                break
    eps_signs = len(eps_str) - 2  # столько знаков после точки надо будет оставлять при выводе
ответов
    return eps, eps_signs


def powell_method(f, a, b, eps, eps_signs):  # метод Пауэлла
    x_1, h = a, eps  # задаём сами
    iter = 0
    table = PrettyTable()
    table.field_names = ["n", "x1", "x2", "x3", "f(x1)", "f(x2)", "f(x3)", "a0", "a1", "a2", "x_min", "f(x_min)",
                         "x*", "f(x*)", "eps_n"]

    while True:
        iter += 1
        x_2 = x_1 + h
        f_x_1, f_x_2 = f(x_1), f(x_2)
        if f(x_1) > f(x_2):
            x_3 = x_1 + Decimal("2") * h
        else:
            x_3 = x_1 - h
        f_x_3 = f(x_3)
        f_min = min(f_x_1, f_x_2, f_x_3)
        if f_min == f_x_1:
            x_min = x_1
        elif f_min == f_x_2:
            x_min = x_2
        else:
            x_min = x_3
        a_0 = f_x_1
        a_1 = (f_x_2 - f_x_1) / (x_2 - x_1)
        a_2 = (Decimal("1") / (x_3 - x_2)) * (((f_x_3 - f_x_1) / (x_3 - x_1)) - ((f_x_2 - f_x_1) / (x_2 - x_1)))
        x_stat = ((x_2 + x_1) / Decimal("2")) - (a_1 / (Decimal("2") * a_2))
        f_x_stat = f(x_stat)
        eps_n = abs(x_stat - x_min)
        table.add_row([iter, f"{x_1:.{eps_signs}f}", f"{x_2:.{eps_signs}f}", f"{x_3:.{eps_signs}f}",
                       f"{f_x_1:.{eps_signs}f}", f"{f_x_2:.{eps_signs}f}", f"{f_x_3:.{eps_signs}f}",
                       f"{a_0:.{eps_signs}f}",
```

```
                f"{a_1:.{eps_signs}f}", f"{a_2:.{eps_signs}f}", f"{x_min:.{eps_signs}f}",
                f"{f_min:.{eps_signs}f}",
                f"{x_stat:.{eps_signs}f}", f"{f_x_stat:.{eps_signs}f}", f"{eps_n:.{eps_signs}f}"])
        if eps_n < eps:
            break
        if f_x_stat < f_min:
            x_1 = x_stat
        else:
            x_1 = x_min

    return x_stat, f_x_stat, table


def main():  # главная функция программы
    # получаем данные от пользователя
    f = get_function()
    a, b = get_interval(f)
    eps, eps_signs = get_accuracy()
    # применение метода Пауэлла
    x_min, f_min, table = powell_method(f, a, b, eps, eps_signs)
    # вывод ответов
    print("\nПолная таблица расчётов:")
    print(table)
    print(f"\nИтог: точка минимума x_min = {x_min:.{eps_signs}f} и минимум f_min =
{f_min:.{eps_signs}f}")
    graph_function(f, a, b, x_min, f_min)  # вывод графика функции с найденной точкой


main()
input()  # чтобы консоль не закрылась
```

Файл travelling_salesman_problem.py:

```
"""Решение задачи коммивояжёра полным перебором"""

# импортируем необходимые библиотеки
from prettytable import PrettyTable

# 3 задачи на выбор:
# граф №1 - симметричный из 5 вершин
n_1 = 5
adjacency_matrix_1 = [
    [None, 5, 6, 14, 15],
    [5, None, 7, 10, 6],
    [6, 7, None, 8, 7],
    [14, 10, 8, None, 9],
    [15, 6, 7, 9, None]
]
# граф №2 - асимметричный из 5 вершин
n_2 = 5
adjacency_matrix_2 = [
    [None, 25, 40, 31, 27],
    [5, None, 17, 30, 25],
    [19, 15, None, 6, 1],
    [9, 50, 24, None, 6],
```

```
    [22, 8, 7, 10, None]
]
# граф №3 - асимметричный из 10 вершин
n_3 = 10
adjacency_matrix_3 = [
    [None, 12, 16, 18, 13, 6, None, 11, 4, 4],
    [2, None, 13, 4, None, 8, 8, 11, 20, 5],
    [None, 3, None, 3, 18, 6, 17, 8, 13, None],
    [None, 6, 11, None, 14, 9, 14, None, 17, 10],
    [5, None, 3, 2, None, 4, 10, 19, None, 20],
    [None, None, 8, None, None, 3, None, 17, None, None],
    [None, None, 20, 9, None, 7, None, 19, 3, 4],
    [None, 16, 9, 17, None, 5, 9, None, 8, 13],
    [None, 13, 16, 3, 2, 4, 6, 10, None, 6],
    [7, None, 17, 4, 7, None, 16, 13, 6, None]
]


def get_problem():  # пользователь выбирает задачу
    print("Выберите задачу коммивояжёра, которая будет решаться полным перебором:")
    print("Задача №1: 5 городов, симметричная матрица смежности между городами:")
    table_1 = PrettyTable()
    table_1.field_names = ["Город", "1", "2", "3", "4", "5"]
    for i in range(n_1):
        row = [f"{i + 1}"]
        for j in range(n_1):
            row.append(adjacency_matrix_1[i][j] if adjacency_matrix_1[i][j] != None else "-")
        table_1.add_row(row)
    print(table_1)
    print("Задача №2: 5 городов, асимметричная матрица смежности между городами:")
    table_2 = PrettyTable()
    table_2.field_names = ["Город", "1", "2", "3", "4", "5"]
    for i in range(n_2):
        row = [f"{i + 1}"]
        for j in range(n_2):
            row.append(adjacency_matrix_2[i][j] if adjacency_matrix_2[i][j] != None else "-")
        table_2.add_row(row)
    print(table_2)
    print("Задача №3: 10 городов, асимметричная матрица смежности между городами:")
    table_3 = PrettyTable()
    table_3.field_names = ["Город", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"]
    for i in range(n_3):
        row = [f"{i + 1}"]
        for j in range(n_3):
            row.append(adjacency_matrix_3[i][j] if adjacency_matrix_3[i][j] != None else "-")
        table_3.add_row(row)
    print(table_3)

    while True:
        try:
            number_choice = int(input("Введите число 1, 2 или 3: "))
        except ValueError:
            print("Вы ввели не целое число - введите ещё раз")
        else:
```

```
        if number_choice not in (1, 2, 3):
            print("Вы ввели число, отличное от 1, 2 или 3 - введите ещё раз")
        else:
            if number_choice == 1:
                return n_1, adjacency_matrix_1
            elif number_choice == 2:
                return n_2, adjacency_matrix_2
            else:
                return n_3, adjacency_matrix_3


def get_start_vertex(n):  # пользователь выбирает стартовую вершину
    while True:
        try:
            number_choice = int(input(f"Введите номер стартовой вершины от 1 до {n}: "))
        except ValueError:
            print("Вы ввели не целое число - введите ещё раз")
        else:
            if number_choice not in (elem for elem in range(1, n + 1)):
                print(f"Вы ввели число, отличное от диапазона от 1 до {n} - введите ещё раз")
            else:
                return number_choice - 1  # так как в матрицах нумерация вершин начинается с 0


def complete_search_of_all_ways(n, adjacency_matrix, start_v, all_ways, cur_v, total_v, total_dist,
total_way, used_v):
    # n, adjacency_matrix, start_v - информация, all_ways - ссылка на таблицу всех путей, которая
пополняется,
    # cur_v - текущая вершина, total_v - вершин пройдено на данный момент,
    # total_dist - расстояние пройденное на данный момент, total_way - текущий путь на данный
момент,
    # used_v - список посещённых вершин
    if total_v == n:
        if adjacency_matrix[cur_v][start_v] != None:
            total_dist += adjacency_matrix[cur_v][start_v]
            total_way += f" -> {start_v + 1}"
            all_ways.append((total_dist, total_way))
    else:
        for next_v in range(n):
            if used_v[next_v] == 0 and adjacency_matrix[cur_v][next_v] != None:
                used_v[next_v] = 1
                complete_search_of_all_ways(n, adjacency_matrix, start_v, all_ways, next_v, total_v + 1,
total_dist +
                                            adjacency_matrix[cur_v][next_v], total_way + f" -> {next_v + 1}", used_v)
                used_v[next_v] = 0


def main():  # главная функция программы
    # получаем данные от пользователя
    n, adjacency_matrix = get_problem()
    start_v = get_start_vertex(n)
    # получаем решение задачи коммивояжёра полным перебором
    all_ways = []  # таблица всех возможных путей, в ф-цию передаём ссылку на неё, чтобы она
пополнялась
```

```
    complete_search_of_all_ways(n, adjacency_matrix, start_v, all_ways, start_v, 1, 0,
                                f"{start_v + 1}", [0 if x != start_v else 1 for x in range(n)])
    # теперь таблица all_ways сформирована, можно обработать её, превратив в красивую таблицу
и найти кратчайшие пути
    if len(all_ways) > 0:
        all_ways_prettytable = PrettyTable()
        all_ways_prettytable.field_names = ["Расстояние", "Путь"]
        min_dist, optimal_ways = 10000000000000000000000000000000, []
        for dist, way in all_ways:
            all_ways_prettytable.add_row([dist, way])
            if dist < min_dist:
                min_dist = dist
                optimal_ways = [way]
            elif dist == min_dist:
                optimal_ways.append(way)
        # вывод ответов
        print(all_ways_prettytable)
        print(f"Минимальное расстрояние = {min_dist}")
        print("Оптимальные пути:", *optimal_ways, sep="\n")
    else:
        print("Не существует ни одного подходящего пути")


main()
input()  # чтобы консоль не закрылась
```

Файл prim_algorithm.py:

```
"""Алгоритм Прима"""

# импортируем необходимые библиотеки
import math, matplotlib.pyplot as plt
from prettytable import PrettyTable
from matplotlib.ticker import FixedLocator


# 3 задачи на выбор
# граф №1 - 5 вершин, 7 рёбер
n_1, m_1 = 5, 7
# список рёбер в формате (вершина №1, вершина №2, вес ребра) без повторений, т. к. граф
неориентированный
edge_list_1 = [
    (0, 1, 3), (0, 2, 4), (0, 4, 1), (1, 2, 5), (2, 3, 2), (2, 4, 6), (3, 4, 7)
]
# граф №2 - 7 вершин, 11 рёбер
n_2, m_2 = 7, 11
edge_list_2 = [
    (0, 1, 7), (0, 3, 5), (1, 2, 8), (1, 3, 9), (1, 4, 7), (2, 4, 5), (3, 4, 15), (3, 5, 6), (4, 5, 8), (4, 6, 9),
    (5, 6, 11)
]
# граф №3 - 10 вершин, 21 ребро
n_3, m_3 = 10, 21
edge_list_3 = [
    (0, 1, 6), (0, 2, 3), (0, 4, 9), (1, 2, 4), (1, 3, 2), (1, 6, 9), (2, 3, 2), (2, 4, 9), (2, 5, 9), (3, 5, 8),
    (3, 6, 9), (4, 5, 8), (4, 9, 18), (5, 6, 7), (5, 8, 9), (5, 9, 10), (6, 7, 4), (6, 8, 5), (7, 8, 1), (7, 9, 4),
```

```
    (8, 9, 3)
]


def get_problem():  # пользователь выбирает задачу
    print("Выберите задачу, для которой будет искаться минимальный остов алгоритмом Прима:")
    print("Задача №1: список рёбер для неориентированного связного графа из 5 вершин:")
    table_1 = PrettyTable()
    table_1.field_names = ["Ребро №", "Вершина A", "Вершина B", "Вес ребра между A и B"]
    for i in range(m_1):
        table_1.add_row([i + 1, edge_list_1[i][0] + 1, edge_list_1[i][1] + 1, edge_list_1[i][2]])
    print(table_1)
    print("Задача №2: список рёбер для неориентированного связного графа из 7 вершин:")
    table_2 = PrettyTable()
    table_2.field_names = ["Ребро №", "Вершина A", "Вершина B", "Вес ребра между A и B"]
    for i in range(m_2):
        table_2.add_row([i + 1, edge_list_2[i][0] + 1, edge_list_2[i][1] + 1, edge_list_2[i][2]])
    print(table_2)
    print("Задача №3: список рёбер для неориентированного связного графа из 10 вершин:")
    table_3 = PrettyTable()
    table_3.field_names = ["Ребро №", "Вершина A", "Вершина B", "Вес ребра между A и B"]
    for i in range(m_3):
        table_3.add_row([i + 1, edge_list_3[i][0] + 1, edge_list_3[i][1] + 1, edge_list_3[i][2]])
    print(table_3)

    while True:
        try:
            number_choice = int(input("Введите число 1, 2 или 3: "))
        except ValueError:
            print("Вы ввели не целое число - введите ещё раз")
        else:
            if number_choice not in (1, 2, 3):
                print("Вы ввели число, отличное от 1, 2 или 3 - введите ещё раз")
            else:
                if number_choice == 1:
                    return n_1, m_1, edge_list_1
                elif number_choice == 2:
                    return n_2, m_2, edge_list_2
                else:
                    return n_3, m_3, edge_list_3


def draw_graph(n, m, all_edges, chosen_edges):  # рисует граф с выделением минимального остова
в нём
    # создание и настройка графика
    fig = plt.figure(figsize=(6, 6))
    p = fig.add_subplot()
    plt.xlim(-1.5, 1.5)
    plt.ylim(-1.5, 1.5)
    plt.axis("off")  # скрытие осей
    # вершины графа будут располагаться на единичной тригонометрической окружности
    v_coords = []
    fi = (2 * math.pi) / n
    total_fi = 0
```

```
    for i in range(n):
        total_fi += fi
        if total_fi in (math.pi / 2, math.pi, (3 * math.pi) / 2, 2 * math.pi):
            if total_fi == (math.pi / 2):
                x, y = 0, 1
            elif total_fi == math.pi:
                x, y = -1, 0
            elif total_fi == ((3 * math.pi) / 2):
                x, y = 0, -1
            else:
                x, y = 1, 0
        elif 0 < total_fi < (math.pi / 2):
            x, y = math.cos(total_fi), math.sin(total_fi)
        elif (math.pi / 2) < total_fi < math.pi:
            x, y = -math.cos(math.pi - total_fi), math.sin(math.pi - total_fi)
        elif math.pi < total_fi < ((3 * math.pi) / 2):
            x, y = -math.cos(total_fi - math.pi), -math.sin(total_fi - math.pi)
        else:
            x, y = math.cos((2 * math.pi) - total_fi), -math.sin((2 * math.pi) - total_fi)
        p.scatter([x], [y], marker="o", color="orange")
        p.text(x + 0.1, y, f"{i + 1}", fontweight='bold')
        v_coords.append((x, y))
    for edge in all_edges:
        v_1, v_2, w = edge
        if edge in chosen_edges:
            edge_color = "orange"
        else:
            edge_color = "grey"
        p.plot([v_coords[v_1][0], v_coords[v_2][0]], [v_coords[v_1][1], v_coords[v_2][1]],
color=edge_color)
        x_mid, y_mid = (v_coords[v_1][0] + v_coords[v_2][0]) / 2, (v_coords[v_1][1] + v_coords[v_2][1]) / 2
        p.text(x_mid, y_mid + 0.05, w, color=edge_color)
    plt.show()  # вывод графика


def prim_algorithm(n, m, edge_list):
    used_v, edges = [0] * n, []
    used_v[0] = 1  # изначально добавили в остов 1-ую вершину
    total_v, total_w = 1, 0
    for i in range(n - 1):
        w_min, v_add, edge_add = 10000000000000000000000000000000, None, None
        for j in range(m):
            edge = edge_list[j]
            v_1, v_2, w = edge
            if (used_v[v_1] + used_v[v_2]) == 1:
                if w < w_min:
                    w_min = w
                    v_add = v_1 if used_v[v_1] == 0 else v_2
                    edge_add = edge
        used_v[v_add] = 1
        edges.append(edge_add)
        total_v += 1
        total_w += w_min
    return total_w, edges
```

```
def main():
    # получаем данные от пользователя
    n, m, edge_list = get_problem()
    # получаем минимальный остов алгоритмом Прима
    total_w_min, edges_min = prim_algorithm(n, m, edge_list)
    # выводим ответ
    table = PrettyTable()
    table.field_names = ["Вершина A", "Вершина B", "Вес ребра между A и B"]
    for edge in edges_min:
        table.add_row([edge[0] + 1, edge[1] + 1, edge[2]])
    print(f"Вес минимального остовного дерева = {total_w_min}")
    print("Рёбра графа, входящие в остов:")
    print(table)
    draw_graph(n, m, edge_list, edges_min)


main()
input()  # чтобы консоль не закрылась
```