Оглавление

1. Постановка задачи

Темой курсовой работы является – нахождение экстремумов функции заданной слоджным образом. Условие задачи:

Вычислить с точность до $\epsilon = 10^{-5}$ все локальные и глобавльные экстремумы на отрезке $x \in [2,10]$ функции описываемой полиномом четвёртого порядка $P_4(x)$ определённым интерполяционными узлами (x_i, y_i). Здесь пары (x_i, y_i) – координаты векторов решения $\vec{y}$ и правой части $\vec{x}$ системы линейных алгебраических уравнений:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & -1 \\ 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 3 & -1 \\ -1 & 0 & 0 & -1 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{pmatrix}$$

2. Подзадачи

Данную задачу можно разделить на 3 этапа:
1. Решение исходной системы линейных алгебраических уравнений
2. Интерполяция по найденным узлам
3. Нахождение экстремумов интерполянта

Почему именно такие этапы: В начале нам надо найти значения вектора $\vec{y}$ решив исходную СЛАУ, далее находим интерполяционные узлы с помощью которых построим полином, и наконец найдем экстремумы функции.

3. Подзадача 1

3.1 Формулировка подзадачи 1

Дана система линейных алгебраических уравнений, которую необходимо решить:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & -1 \\ 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 3 & -1 \\ -1 & 0 & 0 & -1 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{pmatrix}$$

3.2 Методы для решения подзадачи. Выбор метода решения

Существует множество методов для решения СЛАУ, вот некоторые из них: Метод исключения Гаусса или Жордана-Гаусса, Применение LU разложения, Метод Холецкого, Метод прогонки и прочие. Тажке есть приближенные методы: Схема Якоби, Схема Зейделя и прочие. Для нахождение вектора $\vec{y}$ будем использовать метод Жордана-Гаусса по таким причинам:

1. Метод Гаусса-Жордана является улучшенной версией метода Гаусса, который используется для решения систем линейных уравнений. В случае метода Гаусса-Жордана, когда матрица приводится к диагональному виду, можно сразу получить решение системы уравнений.

2. Данная матрица имеет блочную структуру, где блоки нулей отделены от блоков ненулевых элементов. Метод Гаусса-Жордана хорошо подходит для таких типов матриц, так как он позволяет быстро и эффективно привести матрицу к диагональному виду, что упрощает решение системы уравнений.

Исходя из этих факторов, метод Гаусса-Жордана является хорошим выбором для данной матрицы.

3.3 Описание выбранного метода

Метод Гаусса-Жордана - это метод решения систем линейных уравнений путем приведения исходной матрицы к диагональному виду. Основная идея метода заключается в последовательном применении элементарных преобразований к матрице, чтобы привести ее к ступенчатому виду, а затем к диагональному виду.

Давайте подробно разберем метод Гаусса-Жордана для решения систем линейных уравнений. В нашем случае, у нас есть система линейных уравнений вида $A\vec{y}=\vec{x}$, где A - матрица коэффициентов, $\vec{y}$ - вектор неизвестных, $\vec{x}$ - вектор правой части. Метод Гаусса-Жордана позволяет привести матрицу A к диагональному виду, что упрощает решение системы. Вот основные шаги метода:

1. Создание расширенной матрицы: Объединяем матрицу коэффициентов A и вектор правой части $\vec{x}$ в одну расширенную матрицу $(A|\vec{x})$.

2. Приведение матрицы к ступенчатому виду: Используем элементарные преобразования строк для обнуления всех элементов ниже главной диагонали. Для этого выполняем следующие шаги:

a. Выбираем главный элемент (pivot) - элемент на диагонали, который мы будем использовать для обнуления других элементов в столбце.

b. Для каждой строки ниже текущей строки с pivot:
- Вычитаем из строки ниже текущей строки строку, умноженную на коэффициент, чтобы обнулить соответствующий элемент.

3. Приведение матрицы к диагональному виду: После приведения матрицы к ступенчатому виду, продолжаем процесс, чтобы привести ее к диагональному виду. Для этого:

a. Нормализуем строки: Делим каждую строку на соответствующий диагональный элемент, чтобы получить единицы на диагонали.

b. Идем обратно по ступенчатой матрице, обнуляя элементы выше главной диагонали.

4. Получение решения: После приведения матрицы к диагональному виду, решение системы линейных уравнений $\vec{y}$ находится в последнем столбце расширенной матрицы.

Это основные шаги метода Гаусса-Жордана для решения систем линейных уравнений. Этот метод является эффективным способом решения систем уравнений и нахождения обратной матрицы.

3.4 Программная реализация метода решения СЛАУ

```
def gauss_jordan(x: List[float], y: List[float], verbose=0) -> List[float]:
    m, n = x.shape

    augmented_mat: List[float] = np.zeros(shape=(m, n + 1))
    augmented_mat[:m, :n] = x
    augmented_mat[:, m] = y

    np.set_printoptions(precision=2, suppress=True)

    if verbose > 0:
        print('# Original augmented matrix')
        print(augmented_mat)

    outer_loop: List[List[float]] = [[0, m - 1, 1], [m - 1, 0, -1]]

    for d in range(2):
        for i in range(outer_loop[d][0], outer_loop[d][1], outer_loop[d][2]):
            inner_loop: List[List[float]] = [[i + 1, m, 1], [i - 1, -1, -1]]
            for j in range(inner_loop[d][0], inner_loop[d][1], inner_loop[d][2]):
                k: float = (-1) * augmented_mat[j, i] / augmented_mat[i, i]
                temp_row: List[float] = augmented_mat[i, :] * k
                if verbose > 1:
                    print('# Use line %2i for line %2i' % (i + 1, j + 1))
                    print('k=%.2f' % k, '*', augmented_mat[i, :], '=', temp_row)
                augmented_mat[j, :] = augmented_mat[j, :] + temp_row
                if verbose > 1:
                    print(augmented_mat)

    for i in range(0, m):
        augmented_mat[i, :] = augmented_mat[i, :] / augmented_mat[i, i]

    if verbose > 0:
        print('# Normalize the rows')
        print(augmented_mat)

    return augmented_mat[:, n]
```

Листинг 1. Функция вычисления вектора решения СЛАУ методом Гаусса-Жордана

Разберем его шаги работы приведенной в листинге 1 функции:
1. Создается расширенная матрица **augmented_mat**, куда помещаются исходная матрица коэффициентов **x** и вектор правой части **y**.

2. В цикле для каждого столбца и строки происходит обнуление элементов ниже и выше главной диагонали, делая элементы под диагональю равными нулю.

3. После обнуления элементов выполняется нормализация строк путем деления на диагональные элементы.

4. В конце возвращается решение системы уравнений в виде вектора значений переменных.

Результаты запуска кода:

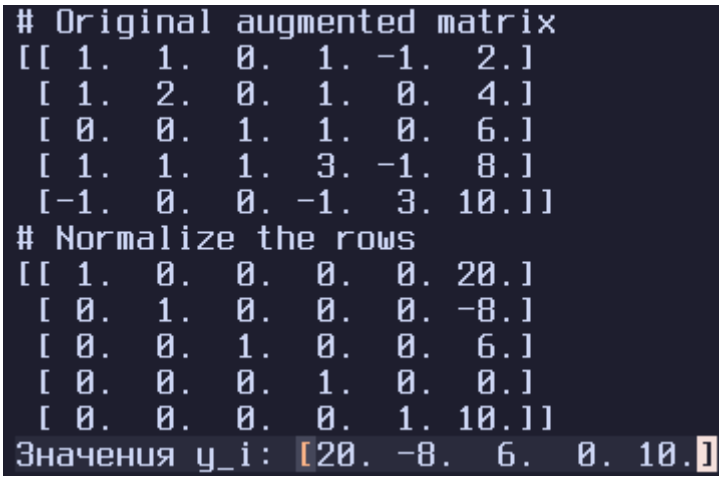

Рисунок 1. Результаты запуска кода, вектор решения СЛАУ.

На рисунке 1 можно увидеть исходную матрицу, диагональную матрицу полученную методом Гаусса-Жордана и вектор решения $\vec{y}=(20,-8,6,0,10)^T$.

В результате решения подзадачи 1 получаем такие интерполяционные узлы:

x	2	4	6	8	10
y	20	-8	6	0	10

4. Подзадача 2

4.1 Формулировка подзадачи 2

При помощи найденных интерполяционных узлов построить полином.

4.2 Методы для решения подзадачи и особенность выбранного метода

Существуют такие методы нахождения интеполяционного многочлена: метод Лагранжа, метод Ньютона, алгоритм Эйткена, алгоритм Невилла.
В качестве метода нахождения полинома будем использовать метод Лагранжа по таким причинам:

1. **Универсальность**: Метод Лагранжа позволяет построить интерполяционный полином для любого набора узлов, включая равномерно распределенные значения, такие как x = (2, 4, 6, 8, 10) и y = (20, -8, 6, 0, 10).

2. **Простота реализации**: Формула Лагранжа для интерполяционного полинома относительно проста и легко вычисляется, что делает метод доступным для понимания и использования.

3. **Точное прохождение через узлы**: Метод Лагранжа гарантирует, что построенный интерполяционный полином будет проходить через все заданные точки (x, y), что может быть важным требованием в некоторых задачах.

4. **Минимальное отклонение**: При правильном выборе узлов метод Лагранжа может обеспечить минимальное отклонение интерполяционного полинома от исходных данных, что является важным для точности интерполяции.

Исходя из этих аргументов, использование метода Лагранжа на данных значениях x и y может быть предпочтительным для построения интерполяционного полинома.

4.3 Описание выбранного метода

Основная идея построения интерполяционного многочлена заключается в нахождении многочлена $L(x_i)=y_i$, для пар чисел $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$, где все x_i различны

Многочлен лагранжа в общем виде вычисляется так:

$$L(x)=\sum_{i=0}^{n} y_i(x),$$

где l_i – базисные полиномы, которые определяются по формуле:

$$l_i(x)=\prod_{j=0,\,j\neq i}^{n}\frac{x-x_i}{x_i-x_j}=\frac{x-x_0}{x_i-x_0}\ldots farc\,x-x_{i-1}x_i-x_{i+1}\cdot\frac{x-x_{i+1}}{x_i-x_{i+1}}\ldots\frac{x-x_n}{x_i-x_n},$$

Для любого i = 0, …, n многочлен l_i имеет степень n и

$$l_i(x_j)=\begin{cases}0, j\neq i\\ 1, j=i\end{cases}$$

В нашем случае значения $\vec{x}$ равномерно распределены на расстоянии $h=2$, т. е. $x_i=x_0+ih, i=0,\ldots,n$, для такого случая многочлен вычисляется таким образом:

$$l_j(x)=\prod_{i=0,\,i\neq j}^{n}\frac{(x-x_i)}{(x_j-x_i)}=\frac{\prod_{i=0,\,I\neq j}^{n}(x-x_0-ih)}{h^n\prod_{i=0,\,i\neq j}^{n}(j-i)}$$

4.4 Программная реализация выбранного метода

```
def L(X: List[float], Y: List[float]) -> Callable[[float], float]:
    if len(X) != len(Y): raise ValueError("Размерность X не совпадает с Y.")

    pairs: Iterable[float] = list(zip(X, Y))
    pairs.sort(key = lambda x: x[0])
    X, Y  = zip(*pairs)

    def polinom(x: float) -> float:
        result: float = 0
        for i in range(len(X)):
            term: float = Y[i]
            for j in range(len(X)):
                if j != i:
                    term *= (x - X[j]) / (X[i] - X[j])
            result += term
        return result

    return polinom
```

Листинг 2. Код метода вычисления полинома Лагранжа

Для проверки правильности программы необходимо записать получившийся полином в алгебраической форме и сравнить получаемые значения в различных точках $\vec{x}$

Найдем полином Лагранжа и вычислим его значения в точках $\vec{x}$:

$$L(x)=20\frac{(x-4)\cdot(x-6)\cdot(x-8)\cdot(x-10)}{(2-4)\cdot(2-6)\cdot(2-8)\cdot(2-10)}-8\frac{(x-2)\cdot(x-6)\cdot(x-8)\cdot(x-10)}{(4-2)\cdot(4-6)\cdot(4-8)\cdot(4-10)}+$$
$$+6\frac{(x-2)\cdot(x-4)\cdot(x-8)\cdot(x-10)}{(6-2)\cdot(6-4)\cdot(6-8)\cdot(6-10)}+0\frac{(x-2)\cdot(x-4)\cdot(x-6)\cdot(x-10)}{(8-2)\cdot(8-4)\cdot(8-6)\cdot(8-10)}+10\frac{(x-2)\cdot(x-4)\cdot(x-6)\cdot(x-8)}{(10-2)\cdot(10-4)\cdot(10-6)\cdot(10-8)}$$

$$L(x)=20\frac{(x-4)\cdot(x-6)\cdot(x-8)\cdot(x-10)}{(-2)\cdot(-4)\cdot(-6)\cdot(-8)}-8\frac{(x-2)\cdot(x-6)\cdot(x-8)\cdot(x-10)}{2\cdot(-2)\cdot(-4)\cdot(-6)}+6\frac{(x-2)\cdot(x-4)\cdot(x-8)\cdot(x-10)}{4\cdot2\cdot(-2)\cdot(-4)}+$$
$$+10\frac{(x-2)\cdot(x-4)\cdot(x-6)\cdot(x-8)}{8\cdot6\cdot4\cdot2}$$

$$L(x)=20\frac{(x^2-10x+24)\cdot(x-8)\cdot(x-10)}{8\cdot(-6)\cdot(-8)}-8\frac{(x^2-8x+12)\cdot(x-8)\cdot(x-10)}{(-4)\cdot(-4)\cdot(-6)}+6\frac{(x^2-6x+8)\cdot(x-8)\cdot(x-10)}{8\cdot(-2)\cdot(-4)}+$$
$$+10\frac{(x^2-6x+8)\cdot(x-6)\cdot(x-8)}{48\cdot4\cdot2}$$

$$L(x)=20\frac{(x^4-28x^3+284x^2-1232x+1920)}{384}-8\frac{(x^4-26x^3+236x^2-856x+960)}{(-96)}+$$
$$+6\frac{(x^4-24x^3+196x^2-624x+640)}{64}+10\frac{(x^4-20x^3+140x^2-400x+384)}{384}$$

$$L(x)=\frac{5}{96}(x^4-28x^3+284x^2-1232x+1920)+\frac{1}{12}(x^4-26x^3+236x^2-856x+960)+$$
$$+\frac{3}{32}(x^4-24x^3+196x^2-624x+640)+\frac{5}{192}(x^4-20x^3+140x^2-400x+384)$$

$$L(x)=(\frac{5}{96}x^4-\frac{35}{24}x^3+\frac{355}{24}x^2-\frac{385}{6}x+100)+(\frac{1}{12}x^4-\frac{13}{6}x^3+\frac{59}{3}x^2-\frac{214}{3}x+80)+$$
$$+(\frac{3}{32}x^4-\frac{9}{4}x^3+\frac{147}{8}x^2-\frac{117}{2}x+60)+(\frac{5}{192}x^4-\frac{25}{48}x^3+\frac{175}{48}x^2-\frac{125}{12}x+10)$$

Найденный многочлен лагранжа

$$L(x)=\frac{49}{192}x^4-\frac{307}{48}x^3+\frac{2711}{48}x^2-\frac{2453}{12}x+250$$

Запишем полученный полином в программу и расчитаем значения в точках $\vec{x}$:

```
def f(x):
    return (49 / 192 * x**4) - (307 / 48 * x**3) + (2711 / 48 * x**2) - (2453 / 12 * x) + 250
```

Листинг 3. Полученный полином записанный в программе

f(x)	L(x)
20.0	20.0
-8.0	-8.0
6.0	6.0
0.0	0.0
10.0	10.0

Рисунок 3. Значения полученные про помощи расчитанного многочлена f(x) и программной реализации L(x) в точках вектора $\vec{x}$

Как мы можем увидеть программная реализация многочлена Лагранжа совпадает со значениями расчитанными вручную, что доказывает правильность алгоритма.

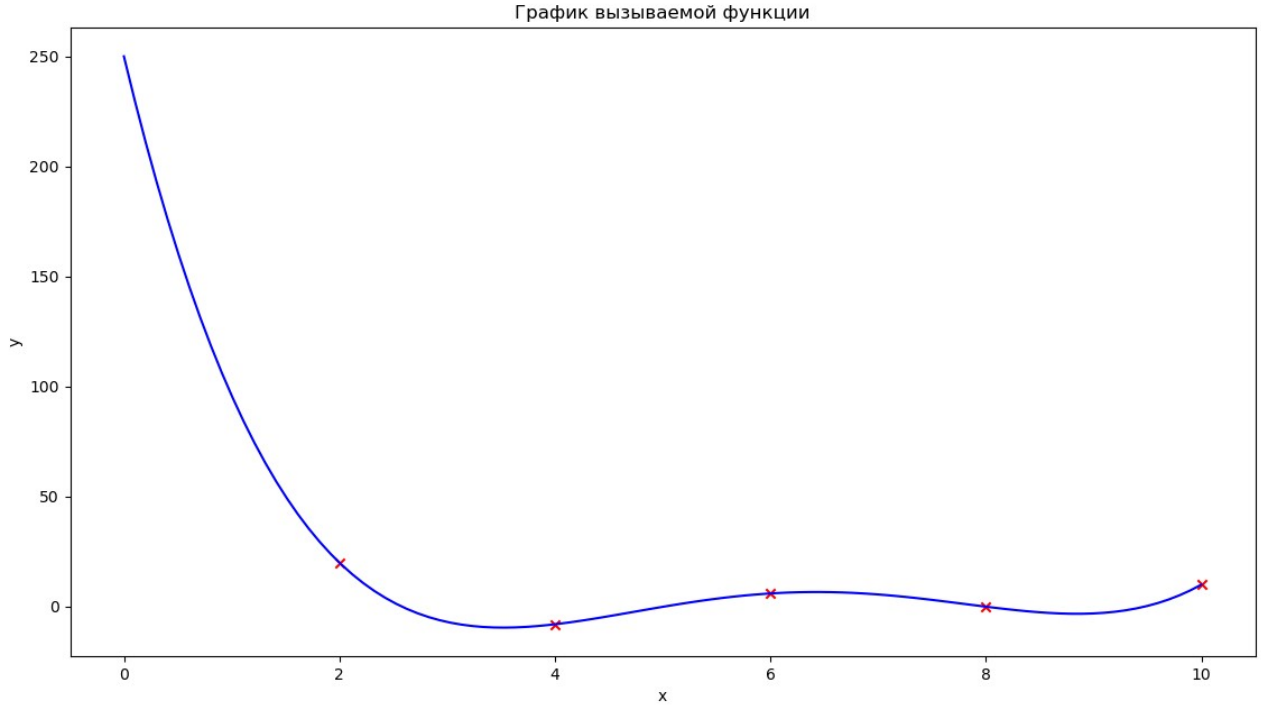

Рисунок 4. График полинома с отмеченными узлами интерполяции

5. Подзадача 3

5.1 Формулировка подзадачи 3

Найти экстремумы полинома:

$$L(x)=\frac{49}{192}x^4-\frac{307}{48}x^3+\frac{2711}{48}x^2-\frac{2453}{12}x+250$$

5.2 Методы решения подзадачи

Задача нахождения экстремумов функции заключается в уточнении значений аргумента x, при которых достигаются локальные экстремумы (максимумы и минимумы), после того как эти значения были предварительно локализованы с использованием графических или аналитических методов. При этом методы уточнения позволяют получить значения экстремумов с заданной точностью. Существуют такие методы нахождения экстремумов как: метод дихотомии, метод золотого сечения, метод Фибоначчи, методы полиномиальной аппроксимации, метод квадратичной аппрокси-мации, также есть методы с использованием производных: метод средней точки (бисекция) и метод Ньютона.

В программной реализации будет использован метод Ньютона. Давайте разберемся в его работе, преимуществах и недостатках.

Метод Ньютона - это итерационный метод нахождения корней функции, основанный на геометрическом смысле производной. Он начинается с начального приближения x0 и использует значение функции y0 в этой точке и ее производную для вычисления следующего приближения x1. Метод продолжается до тех пор, пока полученная погрешность не окажется приемлемо малой.

Важно учитывать недостатки метода Ньютона:

• Функция должна быть дифференцируемой и непрерывной на всем исследуемом интервале.
• Производная функции должна существовать и быть непрерывной на всем интервале.
• Метод Ньютона требует единственности решения на интервале.
• Неподходящее начальное приближение может привести к неверным результатам.

В отличие от этого, поиск экстремумов заключается в поиске точек, где производная функции равна нулю. Точка максимума или минимума идентифицируется в зависимости от знака второй производной.

Итерационная формула метода Ньютона:

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

5.3 Программная реализация метода

```
def derivative(f):
    def df(x, h=1e-5):
        return (f(x + h) - f(x)) / h
    return df
```

```
def newton(f: Callable[[float], float], df: Callable[[float], float], x0: float,
           eps: float=1e-7, kmax: int=1e3) -> float:

    x, x_prev, i = x0, x0 + 2 * eps, 0

    while abs(x - x_prev) >= eps and i < kmax:
        x, x_prev, i = x - f(x) / df(x), x, i + 1

    return x
```

Листинг 5. Код функции нахождения производной и метод Ньютона

В результате вычислений получаем экстремумы в точках: x=2.58094, x=5.00943, x=8.0, x=9.47085

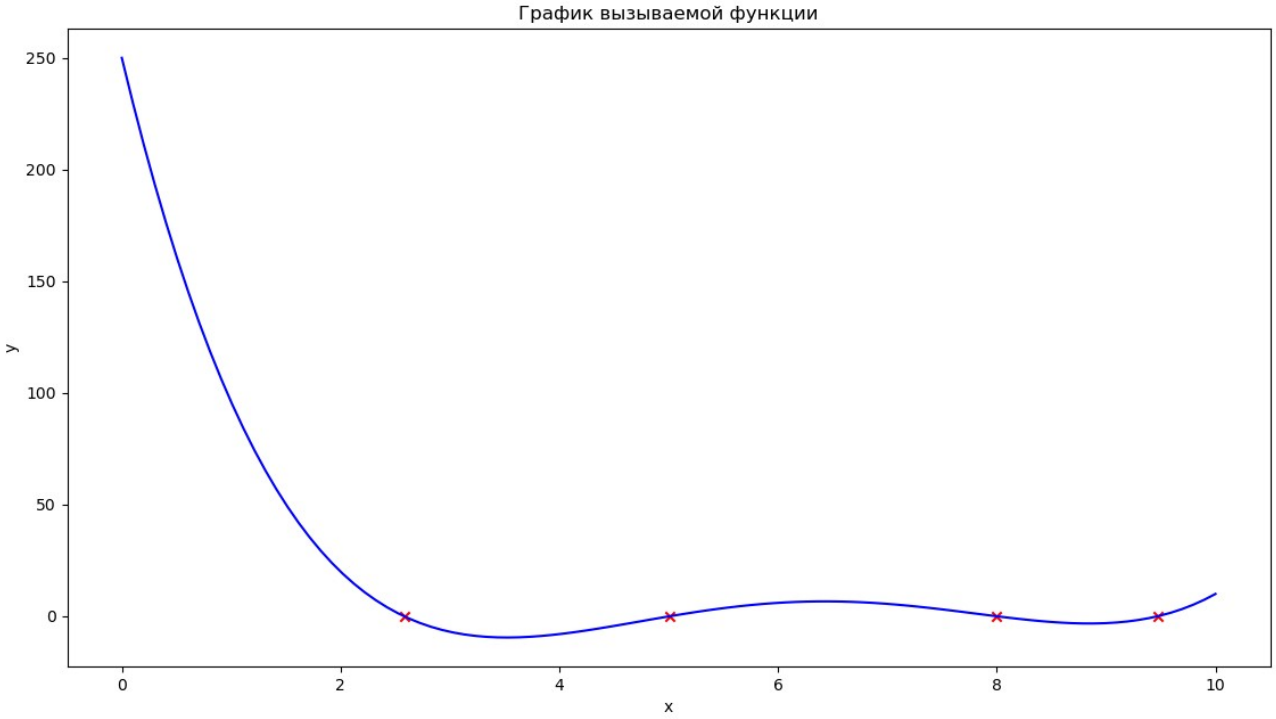

Рисунок 5. Визуализация найденных экстремумов на графике

Полученные результаты

Вектор решений исходной СЛАУ: $\vec{y}=(20,-8,6,0,10)^T$ и правая часть слау $\vec{x}=(2,4,6,8,10)^T$
Интрполяционные узлы в виде таблицы:

$\vec{x}$	2	4	6	8	10
$\vec{y}$	20	-8	6	0	10

Интерполяционный полином в форме Лагранжа:
$L(x)=\frac{49}{192}x^4-\frac{307}{48}x^3+\frac{2711}{48}x^2-\frac{2453}{12}x+250$

Экстремумы функции в точках: $x=2.58094$, $x=5.00943$, $x=8.0$, $x=9.47085$

Краткое описание выполненной работы

В ходе курсовой работы решалась задача по нахождению эктремумов функции заданной сложным образом. Задача была разбита на 3 подзадачи:
1. Найти вектор решения СЛАУ
2. Найти интерполяционные узлы
3. Найти экстремумы интерполянта

Подзадача 1 была решена при помощи метода Жордана-Гаусса, в результате был получаен вектор решения: $\vec{y}=(20,-8,6,0,10)^T$. На следующем шаге требовалось найти интерполяционные узлы. Найдены они были при помощи построения полинома Лагранжа:
$L(x)=\frac{49}{192}x^4-\frac{307}{48}x^3+\frac{2711}{48}x^2-\frac{2453}{12}x+250$
Далее было доказана правильность найденых значений полинома вручную.
Наконец были найдены экстремумы функции методом Ньютона с точностью 10^{-5}:
$x=2.58094, x=5.00943, x=8.0, x=9.47085$

Список литературы

1. Вержбицкий В.М. Основы численных методов: Учебник для вузов/В.М. Вержбицкий. — М.: Высш. шк., 2002. — 840 с.: ил.
2. Киреев В.И. Численные методы в примерах и задачах: Учеб.пособие/В.И. Киреев, А.В. Пантелеев. — 3-е изд. стер. — М.: Высш.шк., 2008. — 480 с: ил.
3. Волков Е. А. Численные методы: Учеб. пособие для вузов.—2-е изд., испр. — М: Наука. Гл. ред. физ.-мат. лит., 1987.— 248 с.
4. Турчак Л. И., Плотников П. В. Основы численных методов: Учебное пособие. — 2-е изд., перераб. и доп. — М.: ФИЗМАТЛИТ, 2003. — 304 с.
5. Численные методы: Учеб, пособие для студ. вузов / М.П.Лапчик, М.И.Рагулина, Е.К.Хеннер; Подред. М.П.Лапчика. - М.: Издательский центр «Академия», 2004. —384 с

Приложение

```
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
from typing import List, Callable



def gauss_jordan(x: List[float], y: List[float], verbose=0) -> List[float]:
    m, n = x.shape

    augmented_mat: List[float] = np.zeros(shape=(m, n + 1))
    augmented_mat[:m, :n] = x
    augmented_mat[:, m] = y

    np.set_printoptions(precision=2, suppress=True)

    if verbose > 0:
        print('Исходная матрица:')
        print(augmented_mat)

    outer_loop: List[List[float]] = [[0, m - 1, 1], [m - 1, 0, -1]]

    for d in range(2):
        for i in range(outer_loop[d][0], outer_loop[d][1], outer_loop[d][2]):
            inner_loop: List[List[float]] = [[i + 1, m, 1], [i - 1, -1, -1]]
            for j in range(inner_loop[d][0], inner_loop[d][1], inner_loop[d][2]):
                k: float = (-1) * augmented_mat[j, i] / augmented_mat[i, i]
                temp_row: List[float] = augmented_mat[i, :] * k
                if verbose > 1:
                    print('Используем строку %2i для строки %2i' % (i + 1, j + 1))
                    print('k=%.2f' % k, '*', augmented_mat[i, :], '=', temp_row)
                augmented_mat[j, :] = augmented_mat[j, :] + temp_row
                if verbose > 1:
                    print(augmented_mat)

    for i in range(0, m):
        augmented_mat[i, :] = augmented_mat[i, :] / augmented_mat[i, i]
```

```
    if verbose > 0:
        print('Диагональная матрица:')
        print(augmented_mat)

    return augmented_mat[:, n]


def L(X: List[float], Y: List[float]) -> Callable[[float], float]:
    if len(X) != len(Y): raise ValueError("Размерность X не совпадает с Y.")

    pairs: Iterable[float] = list(zip(X, Y))
    pairs.sort(key = lambda x: x[0])
    X, Y  = zip(*pairs)

    def polinom(x: float) -> float:
        result: float = 0
        for i in range(len(X)):
            term: float = Y[i]
            for j in range(len(X)):
                if j != i:
                    term *= (x - X[j]) / (X[i] - X[j])
            result += term
        return result

    return polinom


def derivative(f):
    def df(x, h=1e-5):
        return (f(x + h) - f(x)) / h
    return df

def newton(f: Callable[[float], float], df: Callable[[float], float], x0: float,
           eps: float=1e-7, kmax: int=1e3) -> float:

    x, x_prev, i = x0, x0 + 2 * eps, 0

    while abs(x - x_prev) >= eps and i < kmax:
        x, x_prev, i = x - f(x) / df(x), x, i + 1

    return x
```

```python
def main() -> None:
    m_A = np.array([
        [1, 1, 0, 1, -1],
        [1, 2, 0, 1, 0],
        [0, 0, 1, 1, 0],
        [1, 1, 1, 3, -1],
        [-1, 0, 0, -1, 3]
    ])
    m_X = np.array([2, 4, 6, 8, 10])
    m_Y = gauss_jordan(m_A, m_X, 1)

    print(f"Значения y_i: {m_Y}")

    lagr = []
    f: Callable[[float], float] = L(m_X, m_Y)
    for i in range(len(m_X)):
        lagr.append(f(m_X[i]))
    print(f"Многочлен лагранжа в x: {lagr}")

    e = []
    x_e = []
    for i in range(20):
        n = newton(f, derivative(f), i)
        if round(n, 5) not in e:
            e.append(round(n, 5))

    for i in e:
        x_e = f(e)
    print(f"Экстремумы: {e}")

    x = np.linspace(0, 10, 100)
    y = list(map(f, x))
    plt.plot(x, y, color='blue', linestyle='-')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('График вызываемой функции')
    #plt.scatter(m_B, lagr, color='red', marker='x')
    plt.scatter(e, x_e, color='red', marker='x')
```

```
    plt.show()

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

Приложение 1. Код программы