МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Интерполяция таблично заданных функций**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ЗАДАЧ»

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Сенокосова Владислава Владимировича

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Преподаватель  Аспирант | \_\_\_\_\_\_\_\_\_\_ | В.M.Шкатов |
|  | подпись, дата |  |

Саратов 2024

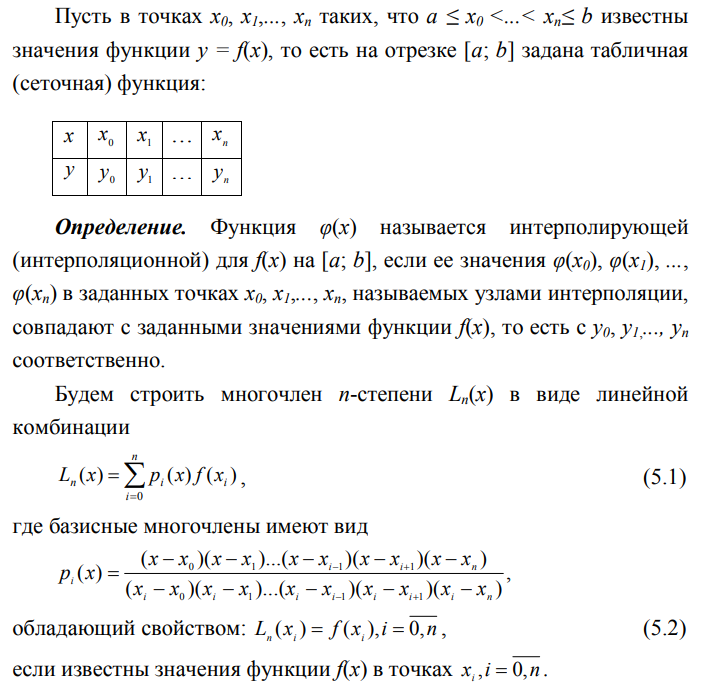
**Вариант 19:**

****

**Задания**

1. Реализовать интерполяционный многочлен Лагранжа
2. Найти таблицу конечных разностей
3. Найти таблицу разделенных разностей
4. Реализовать многочлен Ньютона
5. Реализовать систему линейного сплайна
6. Реализовать систему квадратичного сплайна

**Задание 1: «Многочлен Лагранжа»**

****

Функция, реализующая интерполяционный многочлен Лагранжа представлена в следующем виде:

def lagrange\_interpol\_polinom(system):

*# system => [[x1, x2, x3...], [y1, y2, y3...]]*

*# Вычисление базисных полиномов*

    x = sp.Symbol("x")

    base\_polinoms\_lst = []

    for i in range(len(system[0])):

        numerator, denominator = 1, 1

        for j, x\_i in enumerate(system[0]):

            if j != i:

                numerator \*= (x - x\_i)

                denominator \*= (system[0][i] - x\_i)

        base\_polinoms\_lst.append(numerator / denominator)

*# Вычисление многочлена лагранжа*

    L = 0

    for i, val in enumerate(system[1]):

        L += val \* base\_polinoms\_lst[i]

    return L

В результате работы получаем следующий многочлен для варианта 19:

0.168238551642772\*(x - 2.849)\*(x - 2.672)\*(x - 1.382)\*(x - 0.649) - 0.725338705421255\*(x - 2.849)\*(x - 2.672)\*(x - 1.382)\*(x - 0.234) + 1.51401969098111\*(x - 2.849)\*(x - 2.672)\*(x - 0.649)\*(x - 0.234) - 2.76608480073843\*(x - 2.849)\*(x - 1.382)\*(x - 0.649)\*(x - 0.234) + 2.80087625413611\*(x - 2.672)\*(x - 1.382)\*(x - 0.649)\*(x - 0.234)

В терминале при выводе:

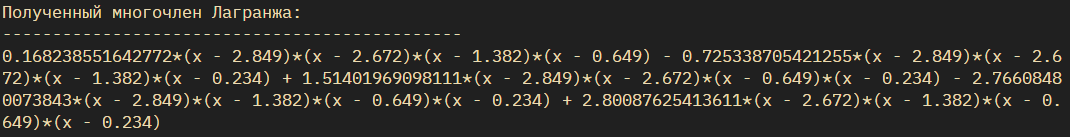
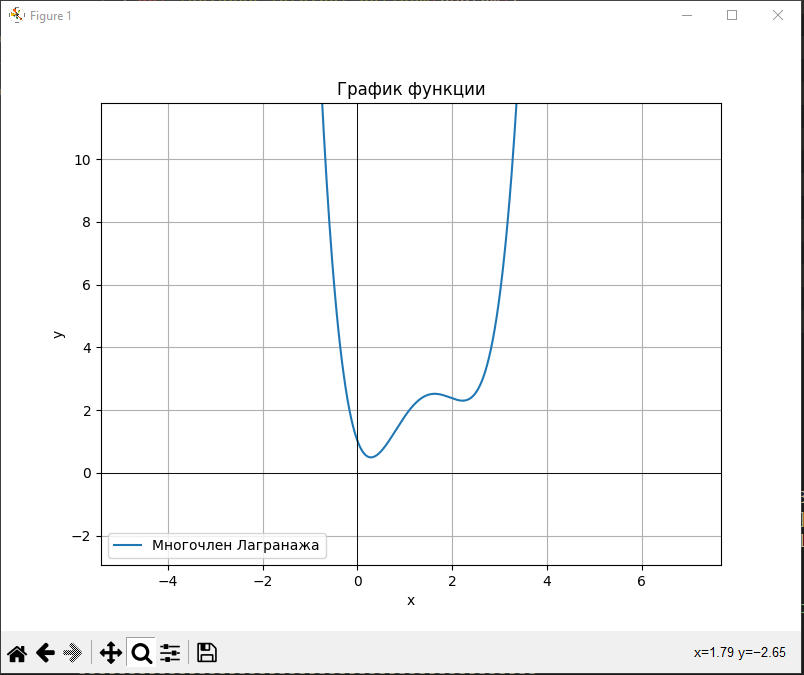


График функции выглядит следующим образом:



**Задание 2: «Таблица конечных разностей»**

Функция, отвечающая за нахождение конечных разностей представлена следующим образом:

def finite\_differences(system):

*# Вычисление таблицы конечных разностей*

    row\_y = deepcopy(system[1])

    arr\_rows\_y = []

    count\_variables = len(row\_y)

    for \_ in range(count\_variables - 1):

        buff\_row = []

        for i in range(len(row\_y) - 1):

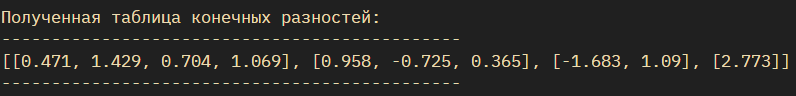
            buff\_row.append(round(row\_y[i + 1] - row\_y[i], 3))

        arr\_rows\_y.append(buff\_row)

        row\_y = buff\_row

    return arr\_rows\_y

Результат выполнения функции:



**Задание 3: «Таблица разделенных разностей»**

Функция, отвечающая за нахождение разделенных разностей представлена следующим образом:

def divided\_differences(system):

    row\_x = deepcopy(system[0])

    row\_y = deepcopy(system[1])

    arr\_rows\_x = []

    for j in range(len(system[0]) - 1):

        buff\_row = []

        for i in range(len(row\_y) - 1):

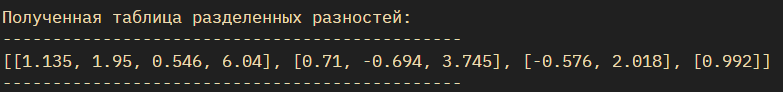
            buff\_row.append(round((row\_y[i + 1] - row\_y[i]) / (row\_x[i + 1 + j] - row\_x[i]), 3))

        arr\_rows\_x.append(buff\_row)

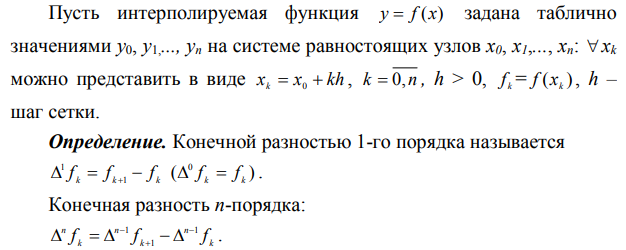
        row\_y = buff\_row

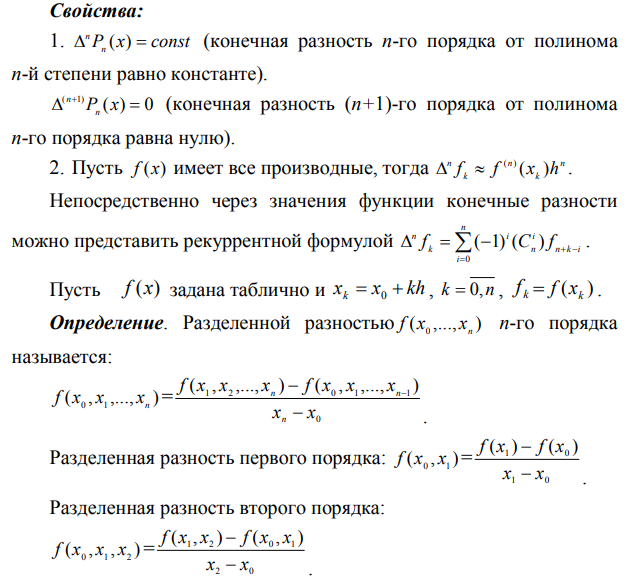
    return arr\_rows\_x

Результат выполнения функции:

****

**Задание 4: «Многочлен Ньютона»**

****

****

Функция, результатом которой является полином Ньютона, представлена следующим образом:

def newtown\_polinom(system):

*# Вычисление многочлена ньютона*

    x = sp.Symbol("x")

*# Выбираем начальное значение функции => y0*

    polinom\_newton = system[1][0]

*# Получаем таблицу разделенных разностей*

    arr\_rows\_x = divided\_differences(system)

    for i in range(len(system[0]) - 1):

        koef = arr\_rows\_x[i][0]

        j = 0

        while i >= 0:

            koef \*= (x - system[0][j])

            i -= 1

            j += 1

        polinom\_newton += koef

*# Вычисление значение полинома в точке x1 + x2*

    x1 = system[0][1]

    x2 = system[0][2]

    val\_func = polinom\_newton.subs({x: x1 + x2})

    return polinom\_newton, val\_func

В результате работы получаем следующий многочлен для варианта 19:

1.135\*x + (0.134784 - 0.576\*x)\*(x - 1.382)\*(x - 0.649) +

(0.71\*x - 0.16614)\*(x - 0.649) +

(0.992\*x - 0.232128)\*(x - 2.672)\*(x - 1.382)\*(x - 0.649) + 0.24541

В терминале при выводе:

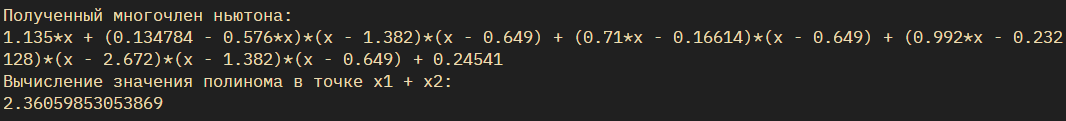
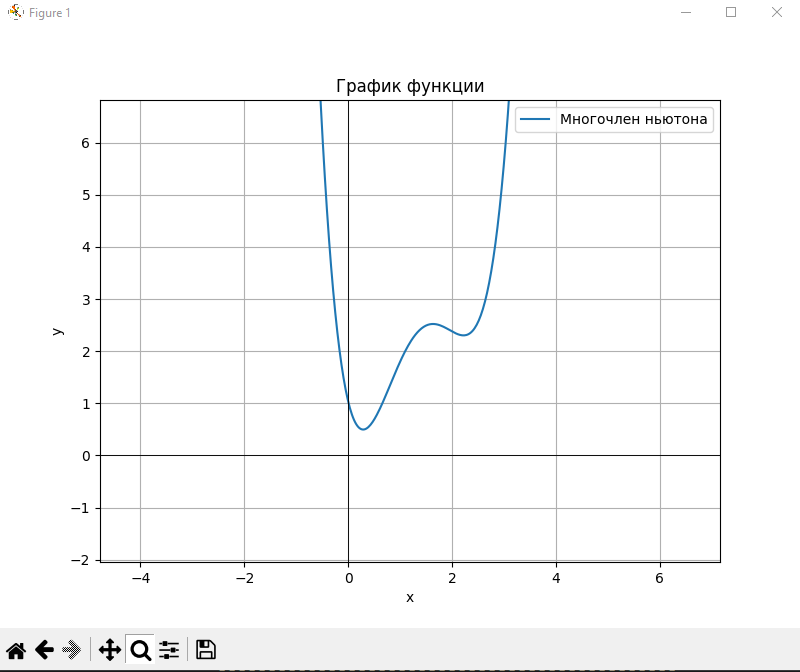
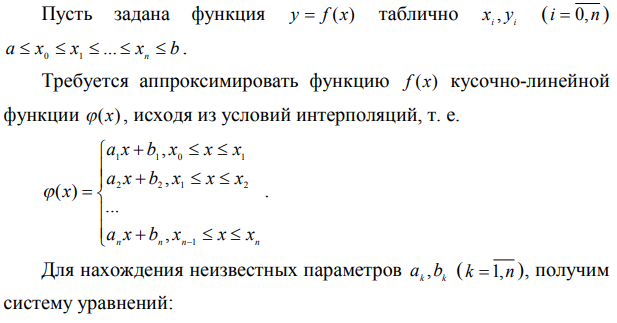
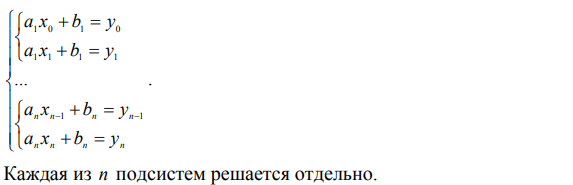


График выглядит следующим образом:



**Задание 5: «Система уравнений линейного сплайна»**

****

****

Функция, реализующая построение системы уравнений линейного сплайна представлена следующим образом:

def build\_linear\_spline(system):

*# Построим интерполяционные сплайн линейный*

    x = sp.Symbol("x")

*# Определяем количество параметров*

    params = []

    for i in range(len(system[0]) - 1):

        params\_for\_system = []

        params\_for\_system.append(sp.Symbol(f"a{i + 1}"))

        params\_for\_system.append(sp.Symbol(f"b{i + 1}"))

        params.append(params\_for\_system)

*# Составляем системы*

    systems = []

    j, i = 1, 0

    for \_ in range(len(params)):

        buff\_sys = []

        j -= 1

        for \_ in range(len(params[0])):

            buff\_sys.append(system[0][j] \* params[i][0] + params[i][1] - system[1][j])

            j += 1

        i += 1

        systems.append(buff\_sys)

*# Находим корни систем*

    roots = []

*# Берем произвольные стартовые приближения*

    intervals = [10 for \_ in range(len(params[0]))]

    for i in range(len(systems)):

        root, \_ = method\_newton(params[i], systems[i], intervals)

        roots.append(list(root))

*# Находим итоговую систему линейного сплайна и диапозоны для каждой из строк системы*

    linear\_spline = []

    for pair\_root in roots:

        linear\_spline.append(pair\_root[0] \* x + pair\_root[1])

*# Определяем диапозоны*

    diaposon = []

    i = 1

    for \_ in range(len(system[0]) - 1):

        buff = []

        i -= 1

        for \_ in range(2):

            buff.append(system[0][i])

            i += 1

        diaposon.append(buff)

*# Объединяем наши уравнения и диапозоны в кортеж*

*# (уравнение, диапозон принимаемых значений x)*

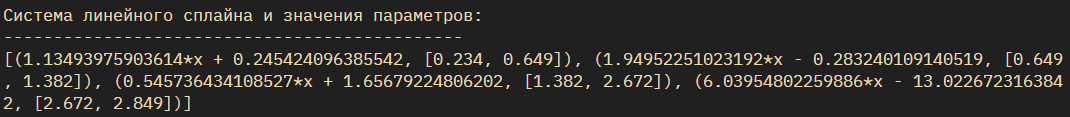
    res\_system = []

    for i in range(len(diaposon)):

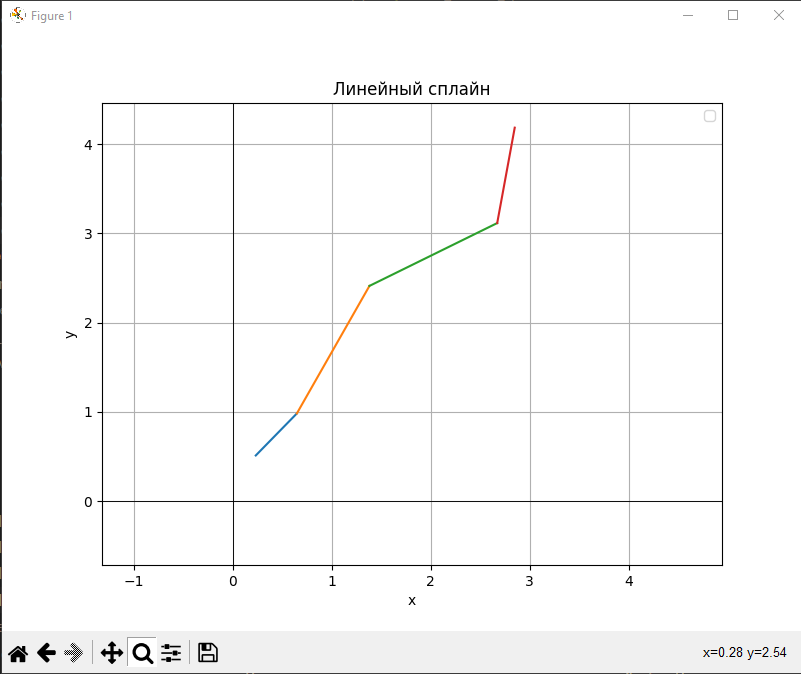
        res\_system.append((linear\_spline[i], diaposon[i]))

    return res\_system

В качестве системы уравнений выступает массив, в котором каждый элемент представлен парой, где первый элемент – это уравнение, а второй элемент – диапазон значений аргумента. В результате выполнения функции получаем следующий вывод:

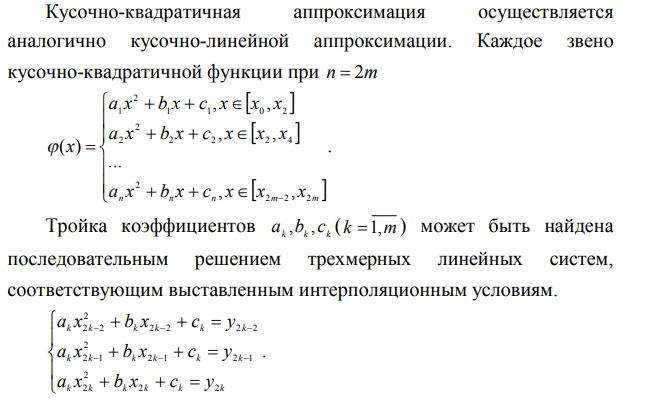


Полученная система уравнений имеет следующий график:



Можно заметить, что эта функция приближенно напоминает исходную функцию.

**Задание 6: «Система уравнений квадратичного сплайна»**

****

Функция, реализующая построение системы уравнений квадратичного сплайна представлена следующим образом:

def build\_kvadro\_spline(system):

*# Построим сплайн квадратичный*

    x = sp.Symbol("x")

*# Определяем параметры*

    params\_kvadro = []

*# Определяем количество систем состоящих из 3 уравнений*

    count\_system = int(len(system[0]) / 3 + 0.99)

    for i in range(count\_system):

        params\_for\_system = []

        params\_for\_system.append(sp.Symbol(f"a{i + 1}"))

        params\_for\_system.append(sp.Symbol(f"b{i + 1}"))

        params\_for\_system.append(sp.Symbol(f"c{i + 1}"))

        params\_kvadro.append(params\_for\_system)

*# Составляем систему уравнений*

    kvadro\_system = []

    k = 0

    j = 0

    index\_system = 0

    count\_itr = len(params\_kvadro[0])

    buff\_row = None

    for i in range(count\_system):

        if buff\_row == None:

            syst = []

        else:

            syst = [buff\_row]

        for \_ in range(count\_itr):

            last\_row = pow(system[0][k], 2) \* params\_kvadro[index\_system][0] \

                        + system[0][k] \* params\_kvadro[index\_system][1] \

                        + params\_kvadro[index\_system][2] - system[1][j]

            syst.append(last\_row)

            k += 1

            j += 1

        index\_system += 1

        count\_itr -= 1

        kvadro\_system.append(syst)

        if index\_system < len(params\_kvadro):

            buff\_row = pow(system[0][k - 1], 2) \* params\_kvadro[index\_system][0] \

                            + system[0][k - 1] \* params\_kvadro[index\_system][1] \

                            + params\_kvadro[index\_system][2] - system[1][j - 1]

*# Вычисление корней систем*

    roots = []

    intervals = [10 for i in range(len(params\_kvadro[0]))]

    for i in range(len(params\_kvadro)):

        root, \_ = method\_newton(params\_kvadro[i], kvadro\_system[i], intervals)

        roots.append(list(root))

*# Вычисление итоговой системы уравнений*

    res\_func = []

    for row in roots:

        buff = 0

        for i in range(len(row)):

            buff += row[i] \* x \*\* (2 - i)

        res\_func.append(buff)

*# Объединяем с диапозоном значений x*

    res = []

    start = 0

    end = len(res\_func)

    for i in range(len(res\_func)):

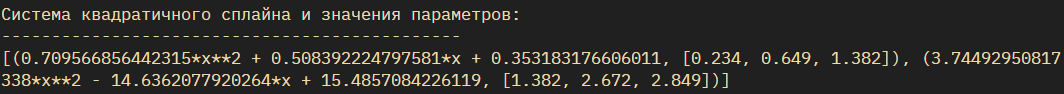
        res.append((res\_func[i], system[0][start:end + 1]))

        start = end

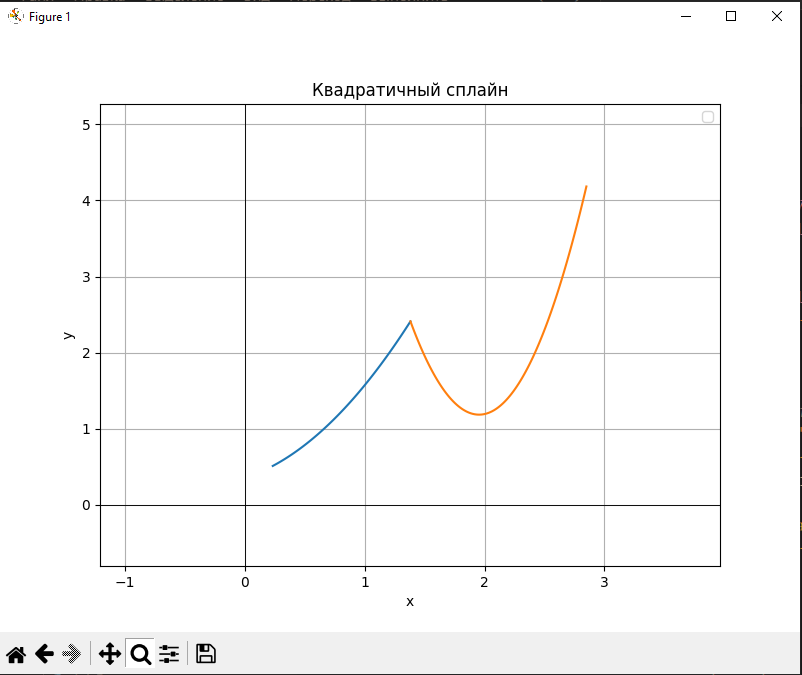
        end = end + len(res\_func)

    return res

В качестве системы уравнений выступает массив, в котором каждый элемент представлен парой, где первый элемент – это уравнение, а второй элемент – диапазон значений аргумента. В результате выполнения функции получаем следующий вывод:

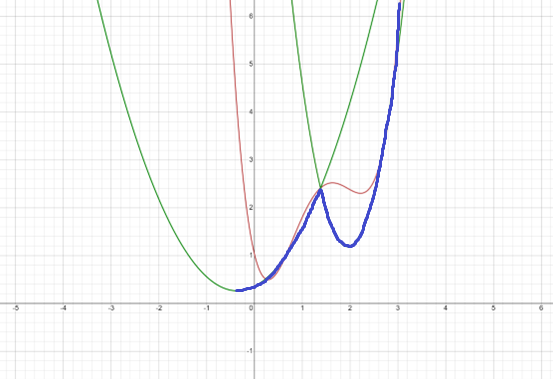


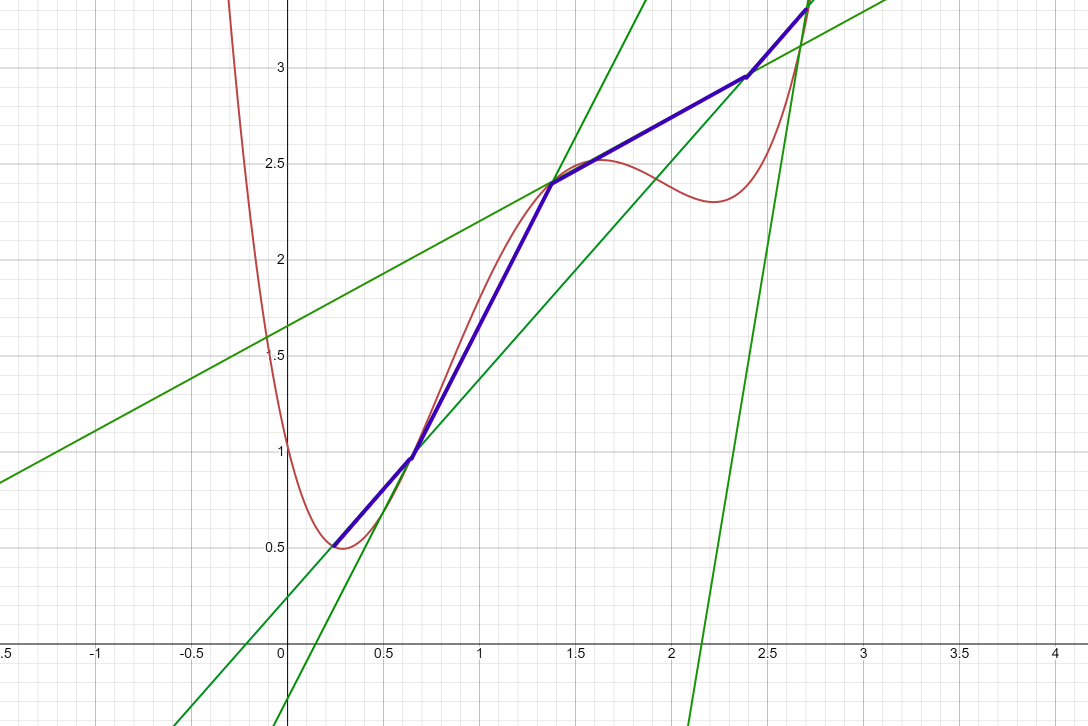
Полученная система уравнений имеет следующий график:



То есть приближение происходит при помощи квадратичных функций.

Отобразим все функции на одном графике. Для простоты анализа было принято решение разделить на два графика. Первый из них демонстрирует исходную функцию (красный) и квадратичный сплайн (зеленый – обведен синем), второй демонстрирует функцию (красный) и линейный сплайн (зеленый – обведен синем):





**Выводы:**

Эх, вот это я, сука, попал... как пять крон в говно.

-Золтан Хивай

**ПРИЛОЖЕНИЕ**

**Листинг программы**

import sympy as sp

import matplotlib.pyplot as plt

import numpy as np

from copy import deepcopy

from lab4 import method\_newton

def lagrange\_interpol\_polinom(system):

*# system => [[x1, x2, x3...], [y1, y2, y3...]]*

*# Вычисление базисных полиномов*

    x = sp.Symbol("x")

    base\_polinoms\_lst = []

    for i in range(len(system[0])):

        numerator, denominator = 1, 1

        for j, x\_i in enumerate(system[0]):

            if j != i:

                numerator \*= (x - x\_i)

                denominator \*= (system[0][i] - x\_i)

        base\_polinoms\_lst.append(numerator / denominator)

*# Вычисление многочлена лагранжа*

    L = 0

    for i, val in enumerate(system[1]):

        L += val \* base\_polinoms\_lst[i]

    return L

def finite\_differences(system):

*# Вычисление таблицы конечных разностей*

    row\_y = deepcopy(system[1])

    arr\_rows\_y = []

    count\_variables = len(row\_y)

    for \_ in range(count\_variables - 1):

        buff\_row = []

        for i in range(len(row\_y) - 1):

            buff\_row.append(round(row\_y[i + 1] - row\_y[i], 3))

        arr\_rows\_y.append(buff\_row)

        row\_y = buff\_row

    return arr\_rows\_y

def divided\_differences(system):

    row\_x = deepcopy(system[0])

    row\_y = deepcopy(system[1])

    arr\_rows\_x = []

    for j in range(len(system[0]) - 1):

        buff\_row = []

        for i in range(len(row\_y) - 1):

            buff\_row.append(round((row\_y[i + 1] - row\_y[i]) / (row\_x[i + 1 + j] - row\_x[i]), 3))

        arr\_rows\_x.append(buff\_row)

        row\_y = buff\_row

    return arr\_rows\_x

def newtown\_polinom(system):

*# Вычисление многочлена ньютона*

    x = sp.Symbol("x")

*# Выбираем начальное значение функции => y0*

    polinom\_newton = system[1][0]

*# Получаем таблицу разделенных разностей*

    arr\_rows\_x = divided\_differences(system)

    for i in range(len(system[0]) - 1):

        koef = arr\_rows\_x[i][0]

        j = 0

        while i >= 0:

            koef \*= (x - system[0][j])

            i -= 1

            j += 1

        polinom\_newton += koef

*# Вычисление значение полинома в точке x1 + x2*

    x1 = system[0][1]

    x2 = system[0][2]

    val\_func = polinom\_newton.subs({x: x1 + x2})

    return polinom\_newton, val\_func

def build\_linear\_spline(system):

*# Построим интерполяционные сплайн линейный*

    x = sp.Symbol("x")

*# Определяем количество параметров*

    params = []

    for i in range(len(system[0]) - 1):

        params\_for\_system = []

        params\_for\_system.append(sp.Symbol(f"a{i + 1}"))

        params\_for\_system.append(sp.Symbol(f"b{i + 1}"))

        params.append(params\_for\_system)

*# Составляем системы*

    systems = []

    j, i = 1, 0

    for \_ in range(len(params)):

        buff\_sys = []

        j -= 1

        for \_ in range(len(params[0])):

            buff\_sys.append(system[0][j] \* params[i][0] + params[i][1] - system[1][j])

            j += 1

        i += 1

        systems.append(buff\_sys)

*# Находим корни систем*

    roots = []

*# Берем произвольные стартовые приближения*

    intervals = [10 for \_ in range(len(params[0]))]

    for i in range(len(systems)):

        root, \_ = method\_newton(params[i], systems[i], intervals)

        roots.append(list(root))

*# Находим итоговую систему линейного сплайна и диапозоны для каждой из строк системы*

    linear\_spline = []

    for pair\_root in roots:

        linear\_spline.append(pair\_root[0] \* x + pair\_root[1])

*# Определяем диапозоны*

    diaposon = []

    i = 1

    for \_ in range(len(system[0]) - 1):

        buff = []

        i -= 1

        for \_ in range(2):

            buff.append(system[0][i])

            i += 1

        diaposon.append(buff)

*# Объединяем наши уравнения и диапозоны в кортеж*

*# (уравнение, диапозон принимаемых значений x)*

    res\_system = []

    for i in range(len(diaposon)):

        res\_system.append((linear\_spline[i], diaposon[i]))

    return res\_system

def build\_kvadro\_spline(system):

*# Построим сплайн квадратичный*

    x = sp.Symbol("x")

*# Определяем параметры*

    params\_kvadro = []

*# Определяем количество систем состоящих из 3 уравнений*

    count\_system = int(len(system[0]) / 3 + 0.99)

    for i in range(count\_system):

        params\_for\_system = []

        params\_for\_system.append(sp.Symbol(f"a{i + 1}"))

        params\_for\_system.append(sp.Symbol(f"b{i + 1}"))

        params\_for\_system.append(sp.Symbol(f"c{i + 1}"))

        params\_kvadro.append(params\_for\_system)

*# Составляем систему уравнений*

    kvadro\_system = []

    k = 0

    j = 0

    index\_system = 0

    count\_itr = len(params\_kvadro[0])

    buff\_row = None

    for i in range(count\_system):

        if buff\_row == None:

            syst = []

        else:

            syst = [buff\_row]

        for \_ in range(count\_itr):

            last\_row = pow(system[0][k], 2) \* params\_kvadro[index\_system][0] \

                        + system[0][k] \* params\_kvadro[index\_system][1] \

                        + params\_kvadro[index\_system][2] - system[1][j]

            syst.append(last\_row)

            k += 1

            j += 1

        index\_system += 1

        count\_itr -= 1

        kvadro\_system.append(syst)

        if index\_system < len(params\_kvadro):

            buff\_row = pow(system[0][k - 1], 2) \* params\_kvadro[index\_system][0] \

                            + system[0][k - 1] \* params\_kvadro[index\_system][1] \

                            + params\_kvadro[index\_system][2] - system[1][j - 1]

*# Вычисление корней систем*

    roots = []

    intervals = [10 for i in range(len(params\_kvadro[0]))]

    for i in range(len(params\_kvadro)):

        root, \_ = method\_newton(params\_kvadro[i], kvadro\_system[i], intervals)

        roots.append(list(root))

*# Вычисление итоговой системы уравнений*

    res\_func = []

    for row in roots:

        buff = 0

        for i in range(len(row)):

            buff += row[i] \* x \*\* (2 - i)

        res\_func.append(buff)

*# Объединяем с диапозоном значений x*

    res = []

    start = 0

    end = len(res\_func)

    for i in range(len(res\_func)):

        res.append((res\_func[i], system[0][start:end + 1]))

        start = end

        end = end + len(res\_func)

    return res

def draw\_func\_system(system, name):

*# Определение переменной x как символ из библиотеки SymPy*

    x = sp.Symbol('x')

*# Построение графиков*

    plt.figure(figsize=(8, 6))

    for equation, x\_range in system:

*# Создание массива значений x в указанном диапазоне*

        x\_values = np.linspace(x\_range[0], x\_range[-1], 100)

*# Преобразование уравнения в функцию*

        equation\_func = sp.lambdify(x, equation, 'numpy')

*# Вычисление значений y для каждого значения x*

        y\_values = equation\_func(x\_values)

*# Построение графика уравнения*

        plt.plot(x\_values, y\_values)

*# Настройка осей координат*

    plt.axhline(0, color='black',linewidth=0.5)

    plt.axvline(0, color='black',linewidth=0.5)

*# Отображение четвертей координатной плоскости*

    plt.axhline(0, color='black',linewidth=0.5)

    plt.axvline(0, color='black',linewidth=0.5)

    plt.xlim(-25, 25)

    plt.ylim(-25, 25)

*# Добавление подписей к осям*

    plt.xlabel('x')

    plt.ylabel('y')

*# Добавление заголовка*

    plt.title(name)

*# Добавление легенды*

    plt.legend()

*# Отображение графика*

    plt.grid(True)

    plt.show()

def draw\_func(func, name):

    x = sp.Symbol('x')

*# Создание массива значений x*

    x\_values = np.linspace(-10, 10, 1000)

*# Вычисление значений функции для каждого значения x*

    y\_values = [func.subs({x: i})for i in x\_values]

*# Построение графика функции*

    plt.figure(figsize=(8, 6))

    plt.plot(x\_values, y\_values, label=f"{name}")

*# Настройка осей координат*

    plt.axhline(0, color='black',linewidth=0.5)

    plt.axvline(0, color='black',linewidth=0.5)

*# Отображение четвертей координатной плоскости*

    plt.axhline(0, color='black',linewidth=0.5)

    plt.axvline(0, color='black',linewidth=0.5)

    plt.xlim(-25, 25)

    plt.ylim(-25, 25)

*# Добавление подписей к осям*

    plt.xlabel('x')

    plt.ylabel('y')

*# Добавление заголовка*

    plt.title('График функции')

*# Добавление легенды*

    plt.legend()

*# Отображение графика*

    plt.grid(True)

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

*# Система из варианта 19*

    system\_1 = [[0.234, 0.649, 1.382, 2.672, 2.849],

                [0.511, 0.982, 2.411, 3.115, 4.184]]

*# Система из примера*

    system\_2 = [[0.351, 0.867, 3.315, 5.013, 6.432],

                [-0.572, -2.015, -3.342, -5.752, -6.911]]

    test\_system = deepcopy(system\_1)

*# Нахождение интерполяционного многочлена Лагранжа*

    print("Полученный многочлен Лагранжа:")

    print("----------------------------------------------")

    polinom1 = lagrange\_interpol\_polinom(test\_system)

    print(polinom1)

    draw\_func(polinom1, "Многочлен Лагранажа")

    print("----------------------------------------------")

*# Построение таблицы конечных разностей*

    print("Полученная таблица конечных разностей:")

    print("----------------------------------------------")

    print(finite\_differences(test\_system))

    print("----------------------------------------------")

*# Построение таблицы разделенных разностей*

    print("Полученная таблица разделенных разностей:")

    print("----------------------------------------------")

    print(divided\_differences(test\_system))

    print("----------------------------------------------")

*# Построение полинома Ньютона*

    print("----------------------------------------------")

    polinom2, val = newtown\_polinom(test\_system)

    print("Полученный многочлен ньютона:")

    print(polinom2)

    draw\_func(polinom2, "Многочлен ньютона")

    print("Вычисление значения полинома в точке x1 + x2:")

    print(val)

    print("----------------------------------------------")

*# Построение линейного сплайна*

    print("Система линейного сплайна и значения параметров:")

    print("----------------------------------------------")

    system1 = build\_linear\_spline(test\_system)

    print(system1)

    draw\_func\_system(system1, "Линейный сплайн")

    print("----------------------------------------------")

*# Построение квадратичного сплайна*

    print("Система квадратичного сплайна и значения параметров:")

    print("----------------------------------------------")

    system2 = build\_kvadro\_spline(test\_system)

    print(system2)

    draw\_func\_system(system2, "Квадратичный сплайн")

    print("----------------------------------------------")