**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа №3   
по курсу «Численные методы»

Группа: М8О-307Б-22

Студент(ка): П. В. Лебедько

Преподаватель: Д. Л. Ревизников

Оценка:

Дата: 16.04.2025

Москва, 2025

**ОГЛАВЛЕНИЕ**

[**1** **Задание** 3](#_Toc194481276)

[**2** **Вариант** 3](#_Toc194481277)

[**3** **Ход лабораторной работы** 4](#_Toc194481278)

[**4** **Выводы** 5](#_Toc194481279)

# **Задание 1**

## **Задание**

Используя таблицу значений  функции , вычисленных в точках  построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки . Вычислить значение погрешности интерполяции в точке .

## **Вариант**

**Вариант 17**

****

## **Ход лабораторной работы**

Определим заданную функцию, а также константу М для расчета погрешности:

def f(x):

    return math.exp(x) + x

def M(x):

    return math.exp(x)

def error(xs, x):

    n = len(xs)

    res = M(xs[-1]) / math.factorial(n)

    for i in range(n):

        res \*= x - xs[i]

    return abs(res)

### Многочлен Лагранжа

Построение многочлена и вычисление его значения в точке x:

def lagrange(xs, x):

    n = len(xs)

    res = 0

    for i in range(n):

        resCur = f(xs[i])

        for j in range(n):

            if (i == j):

                continue

            resCur \*= (x - xs[j]) / (xs[i] - xs[j])

        res += resCur

    return res

### Многочлен Ньютона

Построение многочлена и вычисление его значения в точке x с использованием разделенных разностей (поскольку узлы интерполяции не являются равноудаленными друг от друга):

def newton(xs, x):

    n = len(xs)

    res = f(xs[0])

    polynom = 1

    diffsPrev = [f(x) for x in xs]

    for i in range(2, n + 1): # сколько аргументов у разделенной разности

        diffsCur = []

        polynom \*= x - xs[i - 2]

        for j in range(n - i + 1): # с какого икса начинаем

            diffsCur.append((diffsPrev[j] - diffsPrev[j + 1]) / (xs[j] - xs[j + i - 1]))

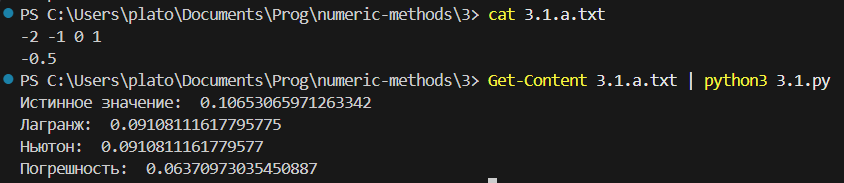
        res += polynom \* diffsCur[0]

        diffsPrev = copy(diffsCur)

    return res

### Результаты

Пункт а)



Пункт б)

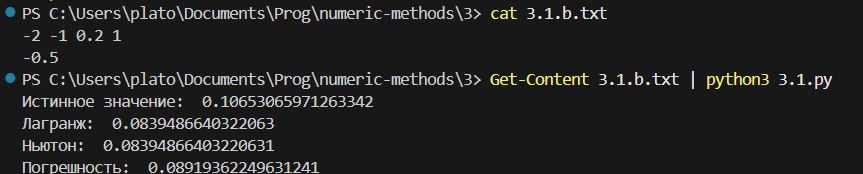
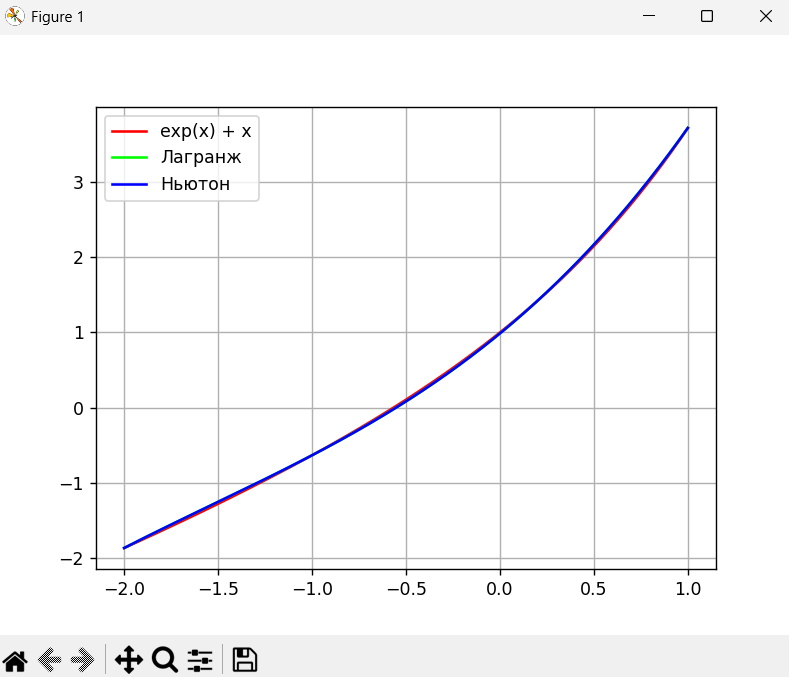


График построенных многочленов:

Видно, что интерполяционные многочлены хорошо приблизили функцию на отрезке [-2; 1]. Разницы между многочленами Лагранжа и Ньютона визуально не заметно.

# **Задание 2**

## **Задание**

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при  и . Вычислить значение функции в точке .

## **Вариант**

**Вариант 17**

-0.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 |
|  | -1.8647 | -0.63212 | 1.0 | 3.7183 | 9.3891 |

## **Ход лабораторной работы**

Решать систему с трехдиагональной матрицей будем, используя реализацию tridiagonalMatrixAlgorithm из предыдущей работы.

Будем вычислять коэффициенты для всех многочленов, затем определять, каким многочленом приближать значение в заданной точке.

def spline(xs, fs: list, x):

    n = len(xs)

    hs = [xs[i] - xs[i - 1] for i in range(1, n)]

    hs.insert(0, 0)

    A = np.zeros((n - 2, 3))

    A[0][0] = 2 \* (hs[1] + hs[2])

    A[0][1] = hs[2]

    A[0][2] = 0

    A[n - 3][0] = 0

    A[n - 3][1] = hs[n - 2]

    A[n - 3][2] = 2 \* (hs[n - 2] + hs[n - 1])

    for i in range(3, n - 1):

        A[i - 2][0] = hs[i - 1]

        A[i - 2][1] = 2 \* (hs[i - 1] + hs[i])

        A[i - 2][2] = hs[i]

    b = np.zeros(n - 2)

    for i in range(n - 2):

        b[i] = 3 \* ((fs[i + 2] - fs[i + 1]) / (hs[i + 2]) - (fs[i + 1] - fs[i]) / (hs[i + 1]))

    cs = tridiagonalMatrixAlgorithm(A, b)

    cs = np.concatenate((np.zeros(1), cs))

    as\_ = copy(fs)

    as\_.pop()

    as\_ = np.array(as\_)

    bs = np.zeros(n - 1)

    for i in range(n - 2):

        bs[i] = (fs[i + 1] - fs[i]) / hs[i + 1] - 1/3 \* hs[i + 1] \* (cs[i + 1] + 2 \* cs[i])

    bs[n - 2] = (fs[n - 1] - fs[n - 2]) / hs[n - 1] - 2/3 \* hs[n - 1] \* cs[n - 2]

    ds = np.zeros(n - 1)

    for i in range(n - 2):

        ds[i] = (cs[i + 1] - cs[i]) / (3 \* hs[i + 1])

    ds[n - 2] = - cs[n - 2] / (3 \* hs[n - 1])

    res = 0

    for i in range(n - 1):

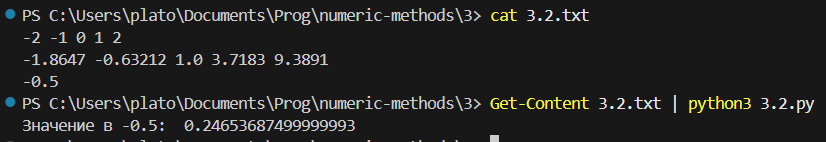
        if xs[i] <= x and x <= xs[i + 1]:

            res = as\_[i] + bs[i] \* (x - xs[i]) + cs[i] \* (x - xs[i]) \*\* 2 + ds[i] \* (x - xs[i]) \*\* 3

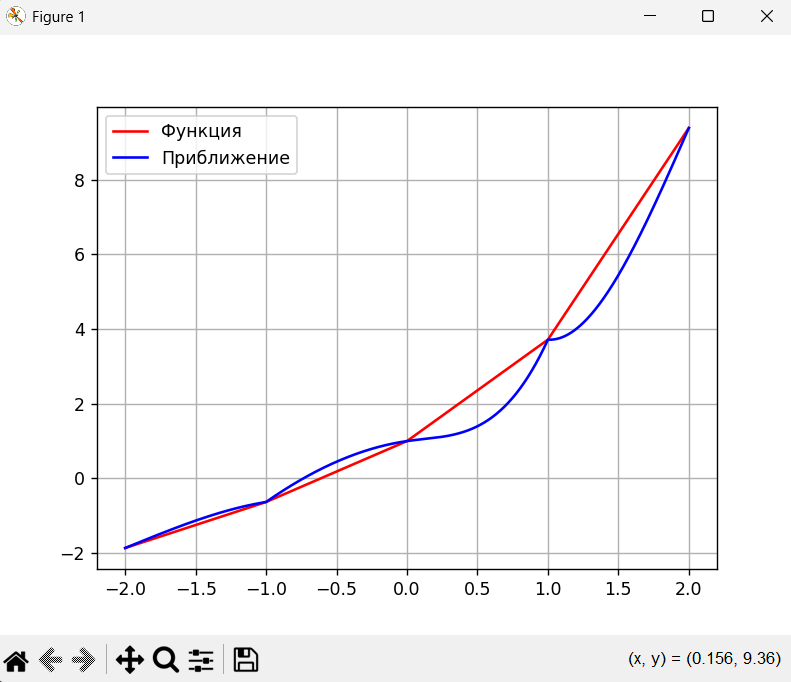
    return res

### Результаты

Вычисленное значение:



Построенный многочлен и соединенные прямыми точки исходной функции:



# **Выводы**