**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 1** 3](#_Toc197083116)

[**Задание** 3](#_Toc197083117)

[**Вариант** 3](#_Toc197083118)

[**Ход лабораторной работы** 3](#_Toc197083119)

[Многочлен Лагранжа 3](#_Toc197083120)

[Многочлен Ньютона 4](#_Toc197083121)

[Результаты 4](#_Toc197083122)

[**2** **Задание 2** 5](#_Toc197083123)

[**Задание** 5](#_Toc197083124)

[**Вариант** 5](#_Toc197083125)

[**Ход лабораторной работы** 6](#_Toc197083126)

[Результаты 7](#_Toc197083127)

[**3** **Задание 3** 8](#_Toc197083128)

[**Задание** 8](#_Toc197083129)

[**Вариант** 8](#_Toc197083130)

[**Ход лабораторной работы** 9](#_Toc197083131)

[Результаты 9](#_Toc197083132)

[**4** **Задание 4** 10](#_Toc197083133)

[**Задание** 10](#_Toc197083134)

[**Вариант** 10](#_Toc197083135)

[**Ход лабораторной работы** 11](#_Toc197083136)

[Результаты 11](#_Toc197083137)

[**5** **Задание 5** 11](#_Toc197083138)

[**Задание** 11](#_Toc197083139)

[**Вариант** 11](#_Toc197083140)

[**Ход лабораторной работы** 12](#_Toc197083141)

[Результаты 13](#_Toc197083142)

[**6** **Выводы** 14](#_Toc197083143)

# **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]))

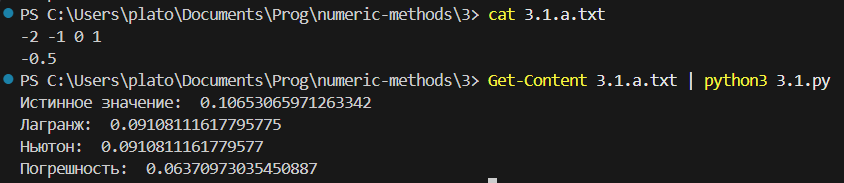
        res += polynom \* diffsCur[0]

        diffsPrev = copy(diffsCur)

    return res

### Результаты

Пункт а)



Пункт б)

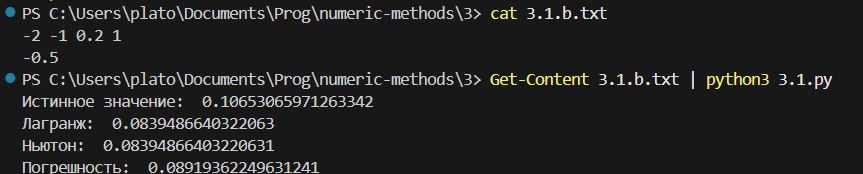
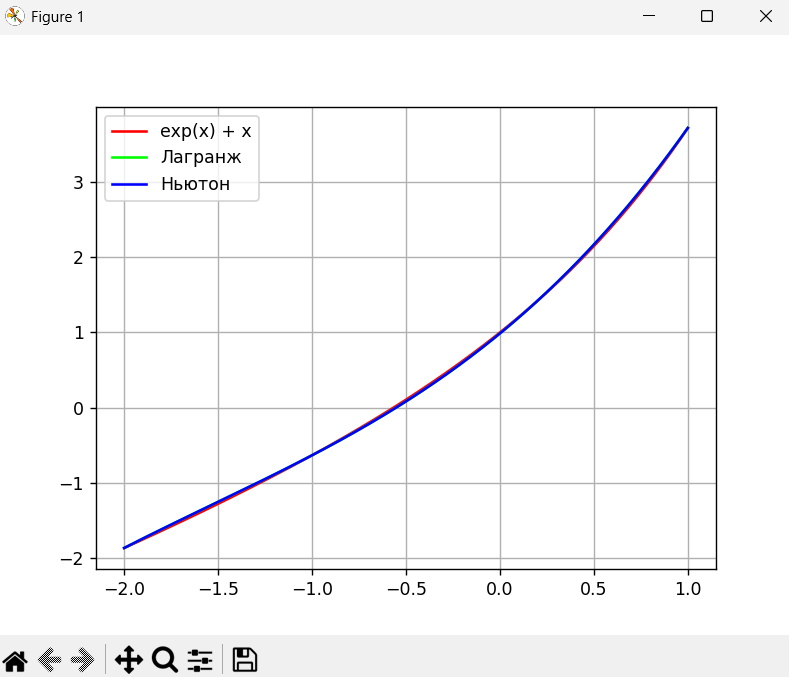


График построенных многочленов:

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):

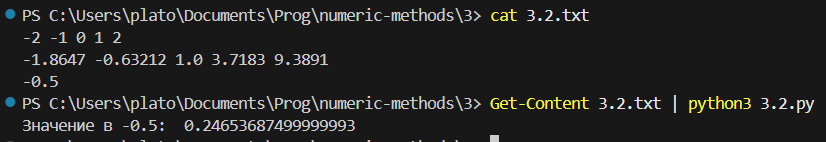
        if xs[i] <= x and x <= xs[i + 1]:

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

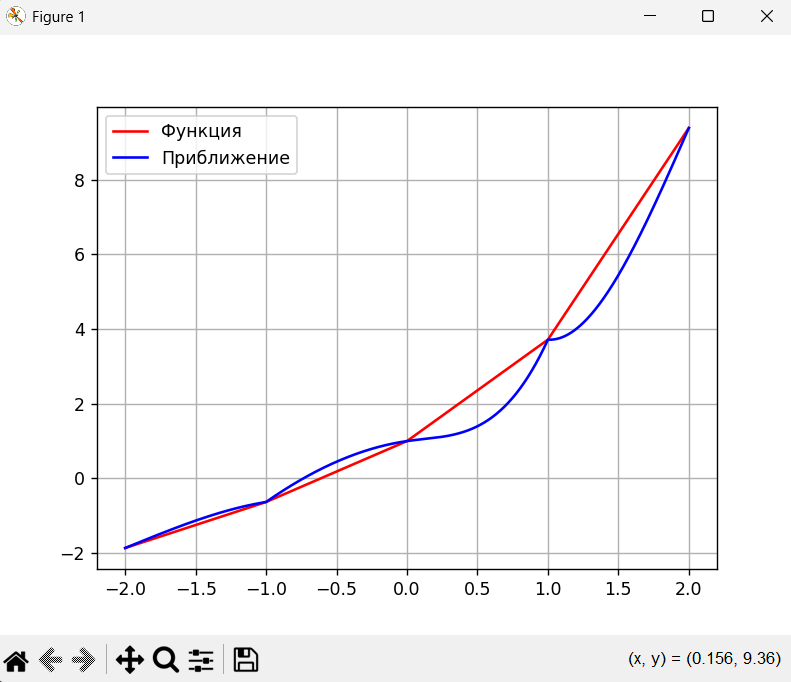
    return res

### Результаты

Вычисленное значение:



Построенный многочлен и соединенные прямыми точки исходной функции:



# **Задание 3**

## **Задание**

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены a) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

## **Вариант**

**Вариант 17**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
|  | -3.0 | -2.0 | -1.0 | 0.0 | 1.0 | 2.0 |
|  | -2.9502 | -1.8647 | -0.63212 | 1.0 | 3.7183 | 9.3891 |

## **Ход лабораторной работы**

Нахождение приближающего многочлена сводится к решению нормальной системы МНК относительно коэффициентов этого многочлена:

A = np.zeros((n, n))

b = np.zeros(n)

for i in range(n):

    for j in range(N):

        b[i] += ys[j] \* xs[j] \*\* i

    for j in range(n):

        for k in range(N):

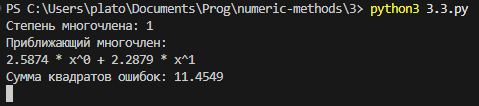
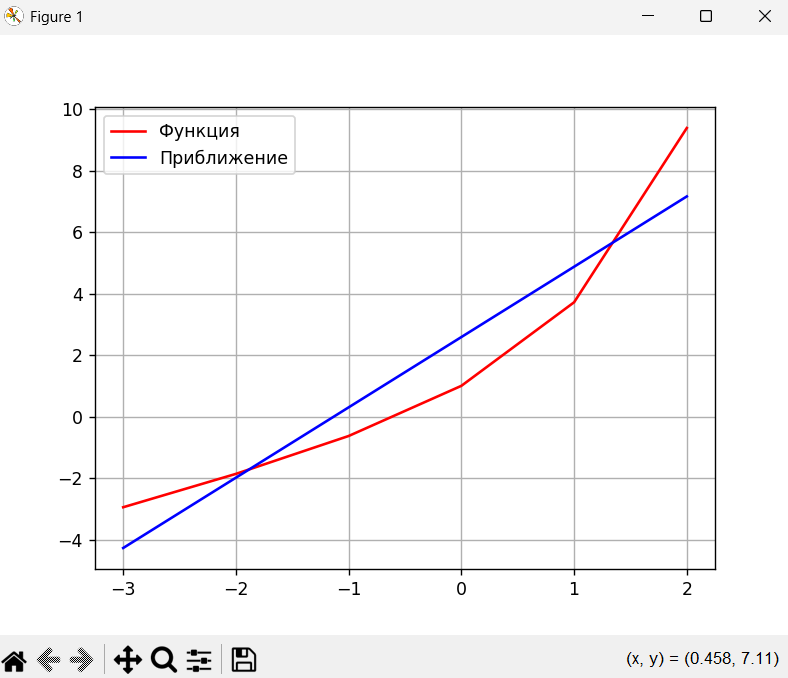
            A[i][j] += xs[k] \*\* (i + j)

as\_ = np.linalg.solve(A, b)

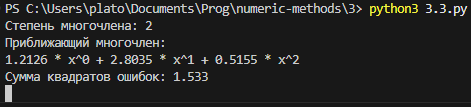
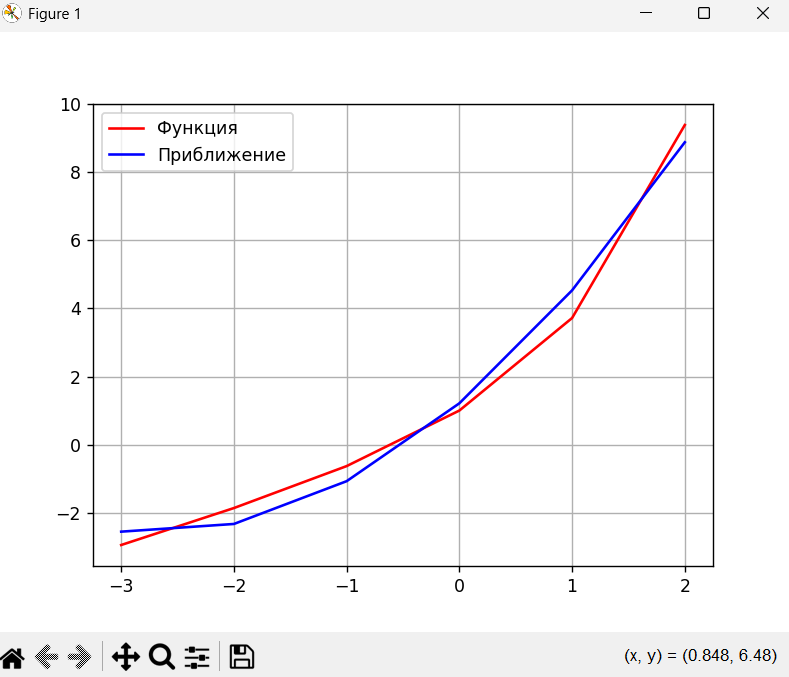
После нахождения коэффициентов многочлена не составит труда построить его график.

### Результаты

Многочлен первой степени:

Многочлен второй степени:

# **Задание 4**

## **Задание**

Вычислить первую и вторую производную от таблично заданной функции в точке .

## **Вариант**

**Вариант 17**

 0.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | -0.2 | 0.0 | 0.2 | 0.4 | 0.6 |
|  | -0.40136 | 0.0 | 0.40136 | 0.81152 | 1.2435 |

## **Ход лабораторной работы**

Для аппроксимации функции на отрезках будем использовать интерполяционные многочлены второй степени и примем, что производная функции равна производной этого многочлена. Тогда для каждой точки x останется понять, какому из отрезков она принадлежит, и вычислить соответствующий многочлен в этой точке.

def getIndex(x, xs):

    for i in range(1, len(xs)):

        if x <= xs[i]:

            return i - 1

def firstDerivative(x, xs, ys):

    i = getIndex(x, xs)

    return (ys[i + 1] - ys[i]) / (xs[i + 1] - xs[i]) + ((ys[i + 2] - ys[i + 1]) / (xs[i + 2] - xs[i + 1]) - (ys[i + 1] - ys[i]) / (xs[i + 1] - xs[i])) / (xs[i + 2] - xs[i]) \* (2 \* x - xs[i] - xs[i + 1])

def secondDerivative(x, xs, ys):

    i = getIndex(x, xs)

    return 2 \* ((ys[i + 2] - ys[i + 1]) / (xs[i + 2] - xs[i + 1]) - (ys[i + 1] - ys[i]) / (xs[i + 1] - xs[i])) / (xs[i + 2] - xs[i])

### Результаты



# **Задание 5**

## **Задание**



## **Вариант**

**Вариант 17**

, ;

## **Ход лабораторной работы**

Реализуем разбиение отрезка на точки с заданным шагом:

def splitting(x0, xk, h):

    xs = []

    x = x0

    while x < xk:

        xs.append(x)

        x += h

    xs.append(xk)

    return xs

Реализуем формулы численного интегрирования: формулу прямоугольников, трапеций и Сипсона:

def rectangles(x0, xk, h):

    integral = 0

    xs = splitting(x0, xk, h)

    for i in range(1, len(xs)):

        integral += h \* f((xs[i - 1] + xs[i]) / 2)

    return integral

def trapezoids(x0, xk, h):

    integral = 0

    xs = splitting(x0, xk, h)

    for i in range(1, len(xs)):

        integral += 0.5 \* h \* (f(xs[i - 1]) + f(xs[i]))

    return integral

def simpson(x0, xk, h):

    integral = 0

    xs = splitting(x0, xk, h)

    for i in range(1, len(xs)):

        integral += 1/3 \* h/2 \* (f(xs[i - 1]) + 4 \* f((xs[i - 1] + xs[i]) / 2) + f(xs[i]))

    return integral

Реализуем уточнение результата с помощью метода Рунге-Ромберга-Ричардсона:

def rungeError(values, hs, p):

    k = hs[0] / hs[1]

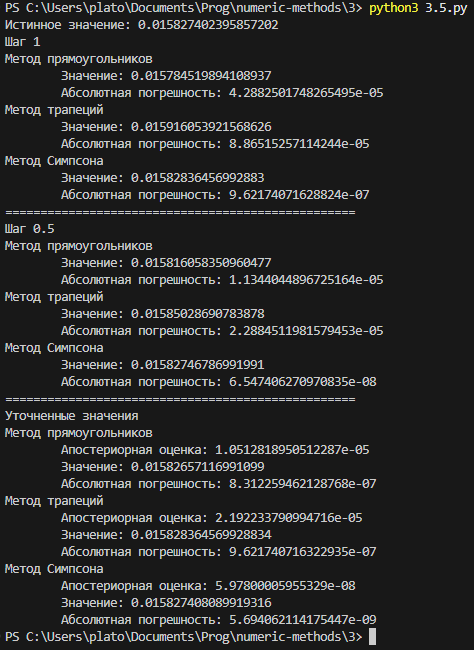
    return (values[1] - values[0]) / (k \*\* p - 1)

def runge(values, hs, p):

    return values[1] + rungeError(values, hs, p)

Затем вычислим значение интеграла по разным формулам и для различных шагов разбиения. Найдем абсолютную погрешность в каждом случае, а также апостериорную оценку погрешности по Рунге, которая на самом деле будет являться уточнением значения интеграла.

### Результаты



# **Выводы**

В ходе данной лабораторной работы я реализовал интерполяцию функций, численное нахождение производных и интегралов. Применил метод Рунге-Ромберга-Ричардсона для уточнения результатов и нахождения апостериорной оценки погрешности.