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

Задание

Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках X_i, $i = 0,\ldots,3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

Вариант

Вариант 17

17. $y = e^x + x$, а) $X_i = -2, -1, 0, 1$; б) $X_i = -2, -1, 0.2, 1$; $X^* = -0.5$.

Ход лабораторной работы

Определим заданную функцию, а также константу M 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

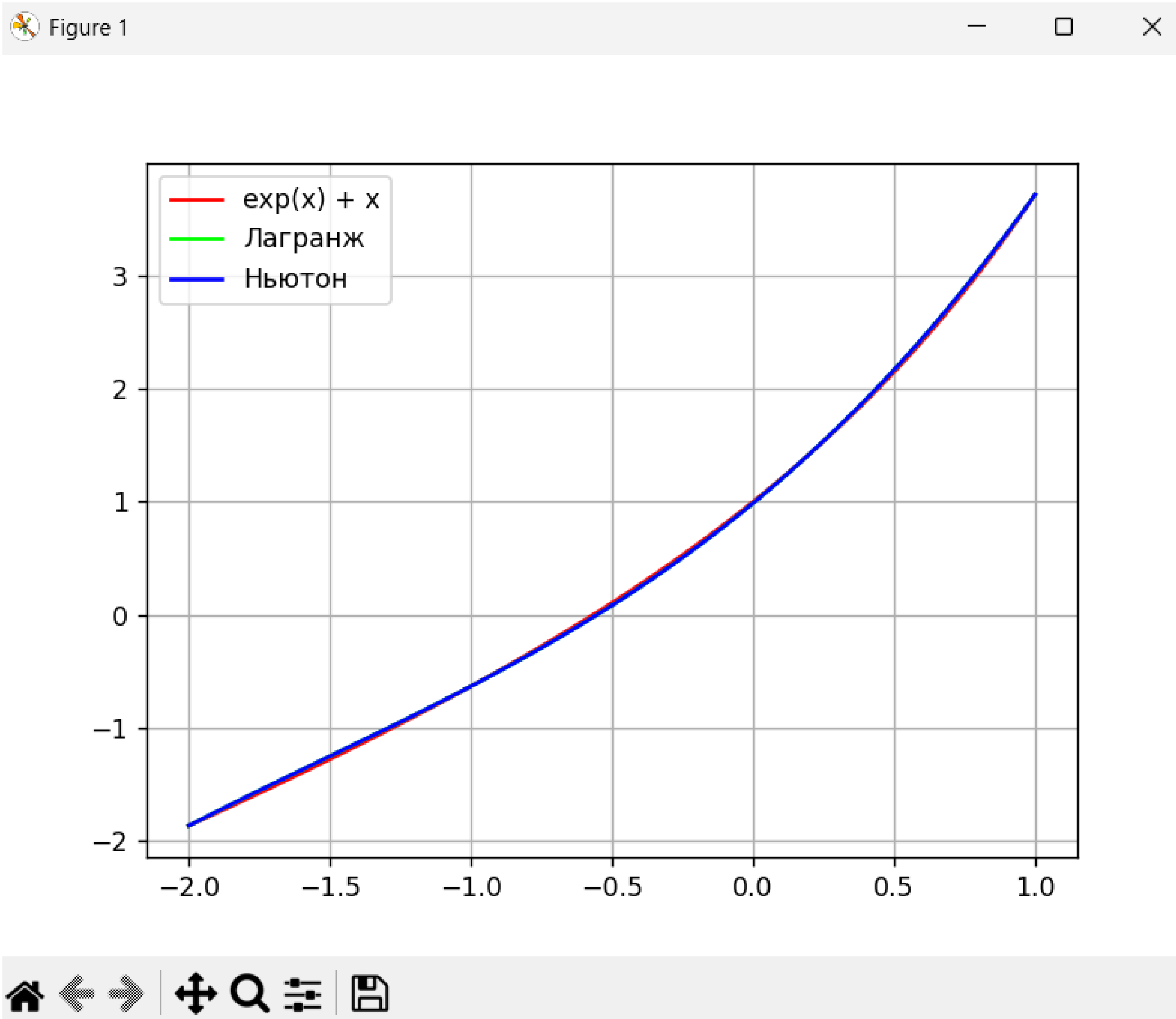
Пункт а)

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> cat 3.1.a.txt
-2 -1 0 1
-0.5
PS C:\Users\plato\Documents\Prog\numeric-methods\3> Get-Content 3.1.a.txt | python3 3.1.py
Истинное значение:  0.10653065971263342
Лагранж:  0.09108111617795775
Ньютон:  0.0910811161779577
Погрешность:  0.06370973035450887
```

Пункт б)

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> cat 3.1.b.txt
-2 -1 0.2 1
-0.5
PS C:\Users\plato\Documents\Prog\numeric-methods\3> Get-Content 3.1.b.txt | python3 3.1.py
Истинное значение:  0.10653065971263342
Лагранж:  0.0839486640322063
Ньютон:  0.08394866403220631
Погрешность:  0.08919362249631241
```

График построенных многочленов:



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

2 Задание 2

Задание

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

Вариант

Вариант 17

$X^* = $-0.5

i	0	1	2	3	4
x_i	-2.0	-1.0	0.0	1.0	2.0
f_i	-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):
```

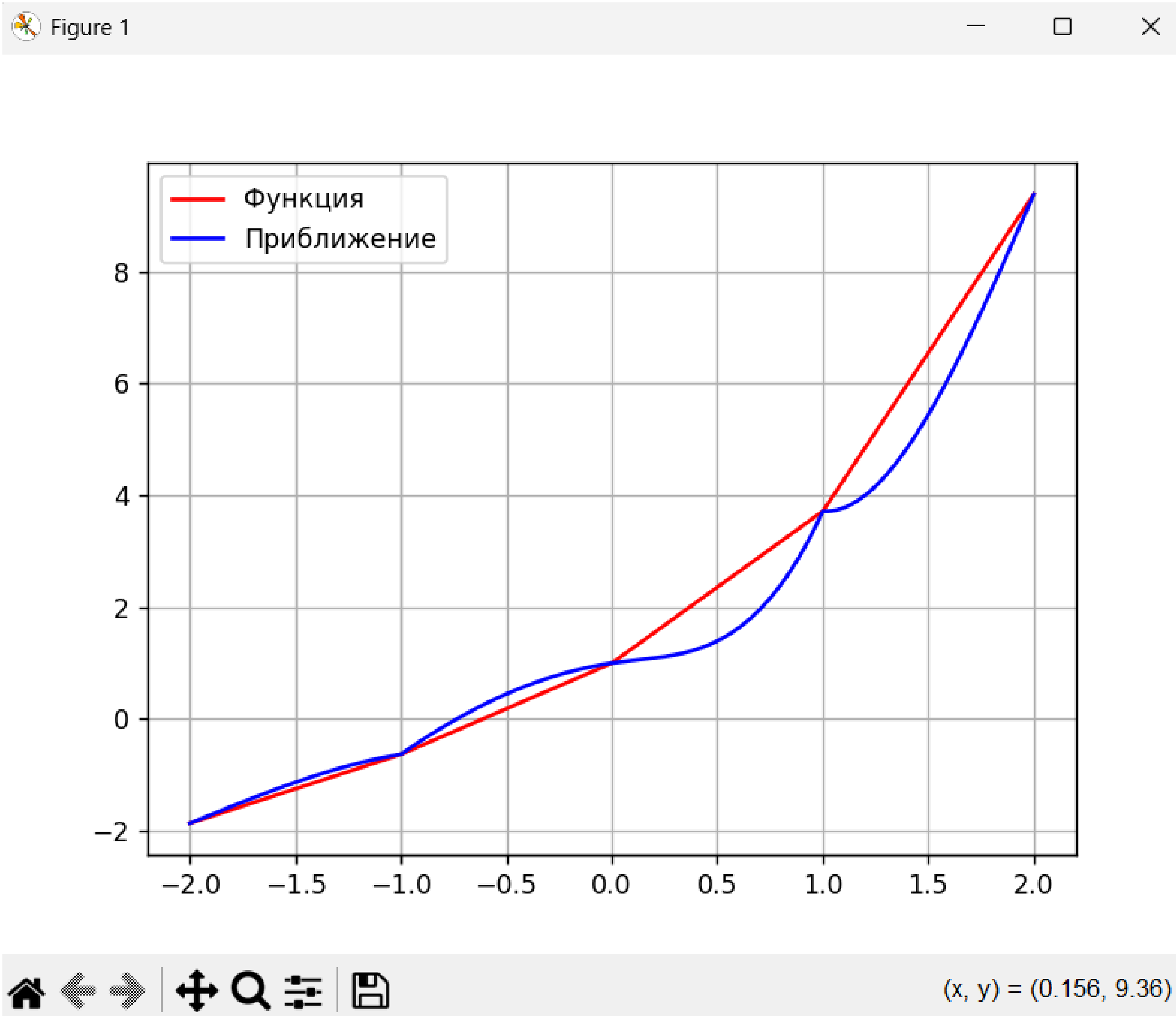
```
        bs[i] = (fs[i + 1] - fs[i]) / hs[i + 1] - 1/3 * hs[i + 1] * (cs[i + 1] +
2 * cs[i])
    bs[n - 2] = (fs[n - 1] - fs[n - 2]) / hs[n - 1] - 2/3 * hs[n - 1] * cs[n - 2]

    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:

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> cat 3.2.txt
-2 -1 0 1 2
-1.8647 -0.63212 1.0 3.7183 9.3891
-0.5
PS C:\Users\plato\Documents\Prog\numeric-methods\3> Get-Content 3.2.txt | python3 3.2.py
Значение в -0.5:  0.24653687499999993
```

Построенный многочлен и соединенные прямыми точки исходной функции:



3 Задание 3

Задание

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 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

i	0	1	2	3	4	5
x_i	-3.0	-2.0	-1.0	0.0	1.0	2.0
y_i	-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:

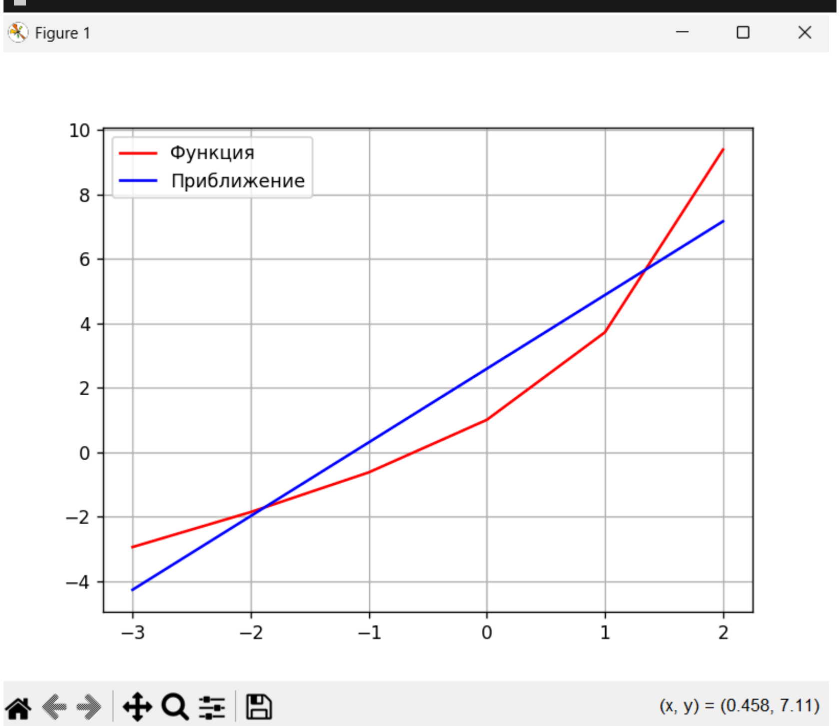
```
A = np.zeros((n, n))
b = np.zeros(n)
for i in range(n):
    for j in range(N):
        b[i] += ys[j] * xs[j] ** i

    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

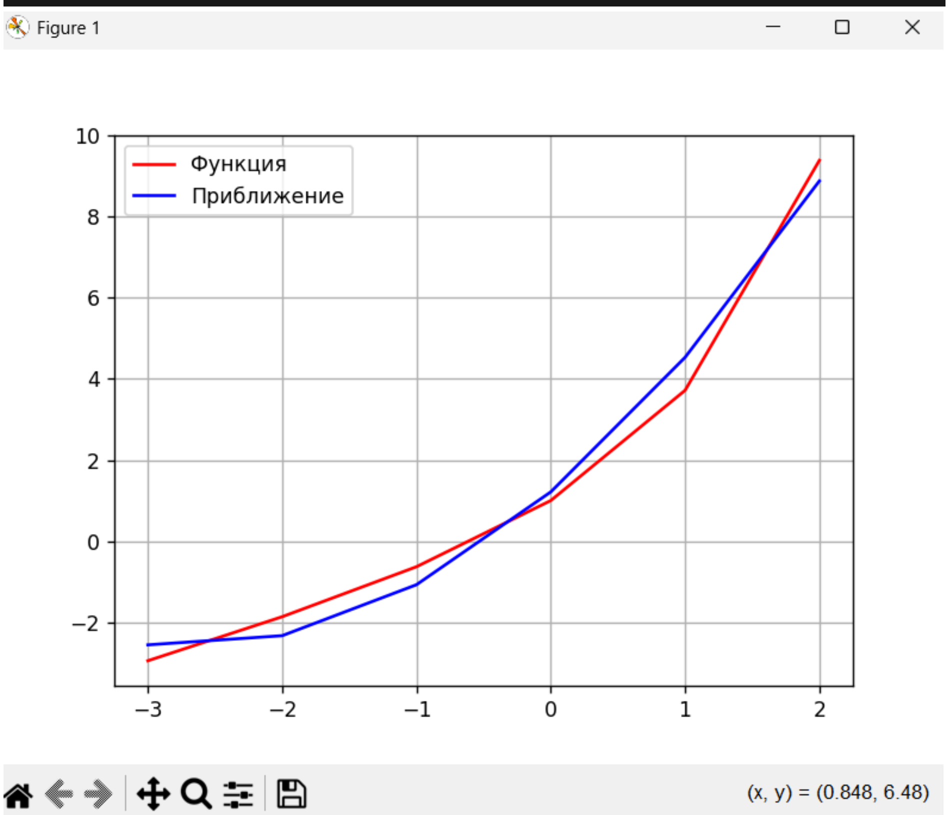
Многочлен первой степени:

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> python3 3.3.py
Степень многочлена: 1
Приближающий многочлен:
2.5874 * x^0 + 2.2879 * x^1
Сумма квадратов ошибок: 11.4549
```



Многочлен второй степени:

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> python3 3.3.py
Степень многочлена: 2
Приближающий многочлен:
1.2126 * x^0 + 2.8035 * x^1 + 0.5155 * x^2
Сумма квадратов ошибок: 1.533
```



4 Задание 4

Задание

Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i)$, $\mathrm{i} = 0,1,2,3,4$ в точке $x = X^*$.

Вариант

Вариант 17

$X^* = 0.2$

i	0	1	2	3	4
x_i	-0.2	0.0	0.2	0.4	0.6
y_i	-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

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> python3 3.4.py
Первая производная: 2.0288
Вторая производная: 0.22000000000000242
```

5 Задание 5

Задание

Вычислить определенный интеграл $F = \int_{X_0}^{X_1} y\,dx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга

Вариант

Вариант 17

$y = \dfrac{1}{256 - x^4}$, $\quad X_0 = -2, \quad X_k = 2, \quad h_1 = 1.0, \quad h_2 = 0.5$;

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

```
PS C:\Users\plato\Documents\Prog\numeric-methods\3> python3 3.5.py
Истинное значение: 0.015827402395857202
Шаг 1
Метод прямоугольников
        Значение: 0.015784519894108937
        Абсолютная погрешность: 4.2882501748265495e-05
Метод трапеций
        Значение: 0.015916053921568626
        Абсолютная погрешность: 8.86515257114244e-05
Метод Симпсона
        Значение: 0.01582836456992883
        Абсолютная погрешность: 9.62174071628824e-07
==================================================
Шаг 0.5
Метод прямоугольников
        Значение: 0.015816058350960477
        Абсолютная погрешность: 1.1344044896725164e-05
Метод трапеций
        Значение: 0.01585028690783878
        Абсолютная погрешность: 2.2884511981579453e-05
Метод Симпсона
        Значение: 0.01582746786991991
        Абсолютная погрешность: 6.547406270970835e-08
==================================================
Уточненные значения
Метод прямоугольников
        Апостериорная оценка: 1.0512818950512287e-05
        Значение: 0.01582657116991099
        Абсолютная погрешность: 8.312259462128768e-07
Метод трапеций
        Апостериорная оценка: 2.192233790994716e-05
        Значение: 0.015828364569928834
        Абсолютная погрешность: 9.621740716322935e-07
Метод Симпсона
        Апостериорная оценка: 5.97800005955329e-08
        Значение: 0.015827408089919316
        Абсолютная погрешность: 5.694062114175447e-09
PS C:\Users\plato\Documents\Prog\numeric-methods\3>
```

6 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.