Московский авиационный институт
(национальный исследовательский университет)

Факультет: «Компьютерные науки и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №3

по курсу «Численные методы»
Тема: «Приближение функций. Численные дифференцирование и интегрирование»

Студент: Мариничев И. А.
Группа: М8О-308Б-19
Преподаватель: Пивоваров Д. Е.
Оценка:

Москва
2022

1. Постановка задачи.

Вариант №12.

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

3.2. 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 $x = X^*$.

3.3. 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.

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

3.5. 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.

2. Описание методов.

3.1. Построение интерполяционных многочленов Лагранжа и Ньютона.

Решаемая задача: Полиномиальная интерполяция.

3.2. Построение кубического сплайна.

Решаемая задача: Сплайн-интерполяция.

3.3. Метод наименьших квадратов (построение приближающих многочленов 1 и 2 степени).

Решаемая задача: Аппроксимация.

3.4. Построение интерполяционного многочлена Ньютона и вычисление его 1 и 2 производной.

Решаемая задача: Численное дифференцирование.

3.5. Метод прямоугольников, метод трапеций, метод Симпсона.

Решаемая задача: Численное интегрирование.

3. Демонстрация работы программы.

3.1. Интерполяционные многочлены Лагранжа и Ньютона

Запустите в терминале программу на **C++** при помощи команд

```
make
make run
```

В двух появившихся файлах `answer_NN.txt` можно увидеть полученные интерполяционные полиномы. Возьмем их коэффициенты:

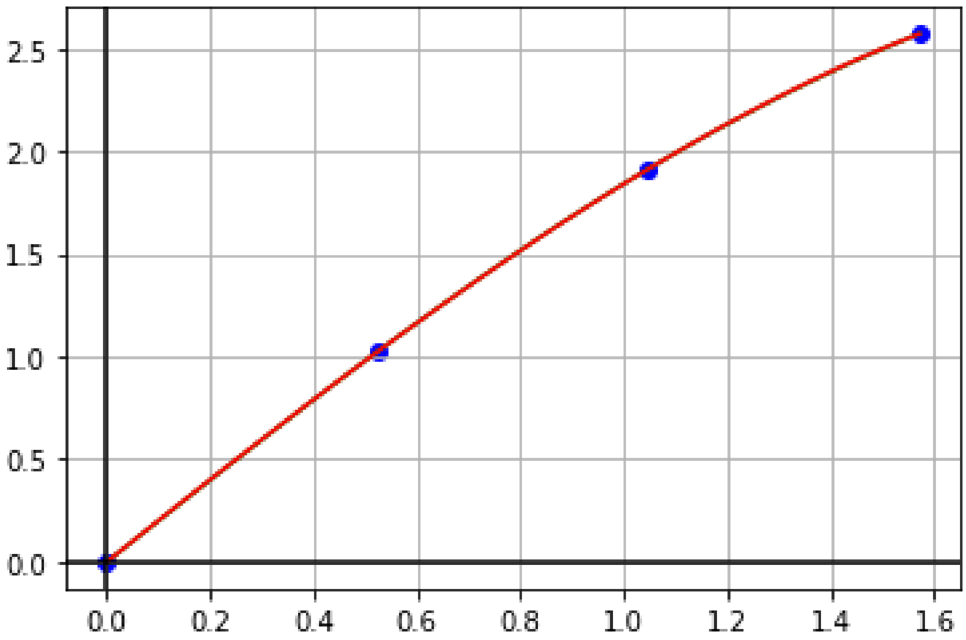
```
coefficients_a = [-0.113872, -0.0654712, 2.02043, 0]
coefficients_b = [-0.121411, -0.0496813, 2.01423, 0]
```

И посторим соответствующие графики

```
import numpy as np
import matplotlib.pyplot as plt


def function(x):
    return np.sin(x) + x


def draw(x, y, a):
    X = np.linspace(x[0], x[-1], 100)
    Y = a[0] * X ** 3 + a[1] * X ** 2 + a[2] * X + a[3]

    plt.plot(X, function(X), "-g")
    plt.plot(X, Y, "-r")

    plt.grid(True, which='both')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.scatter(x, y, c="blue")
    plt.show()
```

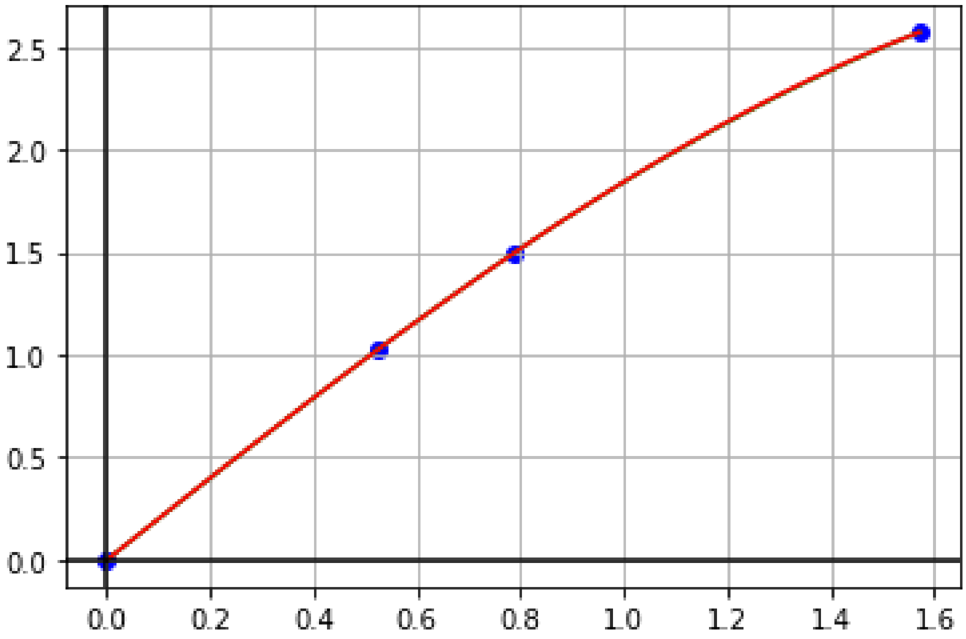
a)

```
X = [0, np.pi/6, 2*np.pi/6, 3*np.pi/6]
Y = [function(x) for x in X]

draw(X, Y, coefficients_a)
```



б)

```
X = [0, np.pi/6, np.pi/4, np.pi/2]
Y = [function(x) for x in X]

draw(X, Y, coefficients_b)
```



3.2. Кубический сплайн

Запустите в терминале программу на **C++** при помощи команд

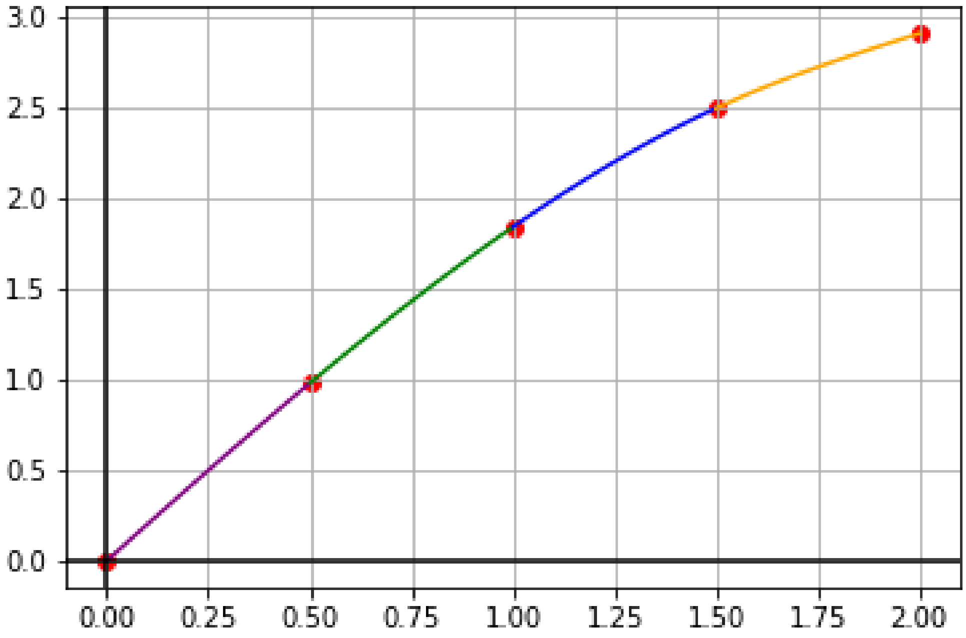
```
make
make run
```

В появившемся файле answer_NN.txt можно увидеть полученные коэффициенты участков сплайна:

```
coefficients = [
    [0.0000, 2.0010, 0.0000, -0.1686],
    [0.9794, 1.8746, -0.2529, -0.0958],
    [1.8415, 1.5498, -0.3966, -0.1579],
    [2.4975, 1.0347, -0.6334, 0.4223]
]
```

Проверим их

```
import numpy as np
import matplotlib.pyplot as plt


def draw(x, y, a):
    colors = ["purple", "green", "blue", "orange"]
    for i in range(1, len(x)):
        X = np.linspace(x[i - 1], x[i], 100)
        Y = a[i - 1][0] + a[i - 1][1] * (X - x[i - 1]) + a[i - 1][2] * (
            X - x[i - 1]) ** 2 + a[i - 1][3] * (X - x[i - 1]) ** 3
        plt.plot(X, Y, c=colors[i-1])
    plt.grid(True, which='both')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.scatter(x, y, c="red")
    plt.show()

X = [0.0, 0.5,     1.0,    1.5,    2.0]
Y = [0.0, 0.97943, 1.8415, 2.4975, 2.9093]

draw(X, Y, coefficients)
```



3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

3.3. МНК

Запустите в терминале программу на **C++** при помощи команд

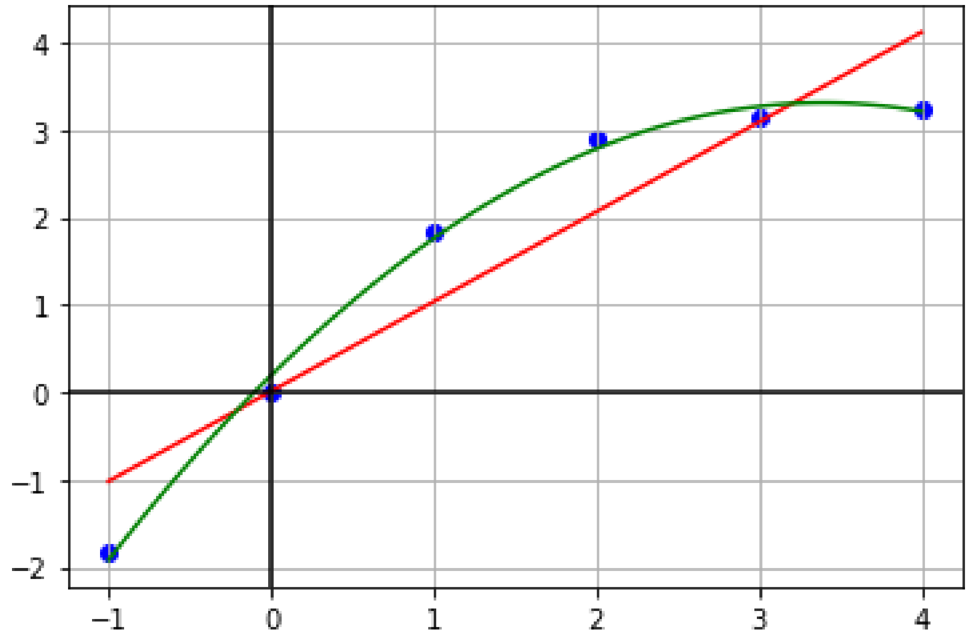
```
make
make run
```

В появившемся файле `answer_NN.txt` можно увидеть полученные коэффициенты приближающих многочленов первой и второй степени соответственно:

```
coefficients_1 = [0.0097, 1.0261]
coefficients_2 = [0.1899, 1.8370, -0.2703]
```

Проверим их

```python
import numpy as np
import matplotlib.pyplot as plt

def draw(l, r, x, y, a1,a2):
    X = np.linspace(l, r, 100)
    Y1 = a1[0] + a1[1] * X
    Y2 = a2[0] + a2[1] * X + a2[2] * X ** 2

    plt.plot(X, Y1, "-r")
    plt.plot(X, Y2, "-g")

    plt.grid(True, which='both')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')

    plt.scatter(x, y, c ="blue")
    plt.show()

X = [-1.0,    0.0, 1.0,    2.0,    3.0,    4.0]
Y = [-1.8415, 0.0, 1.8415, 2.9093, 3.1411, 3.2432]

draw(X[0], X[-1], X, Y, coefficients_1, coefficients_2)
```

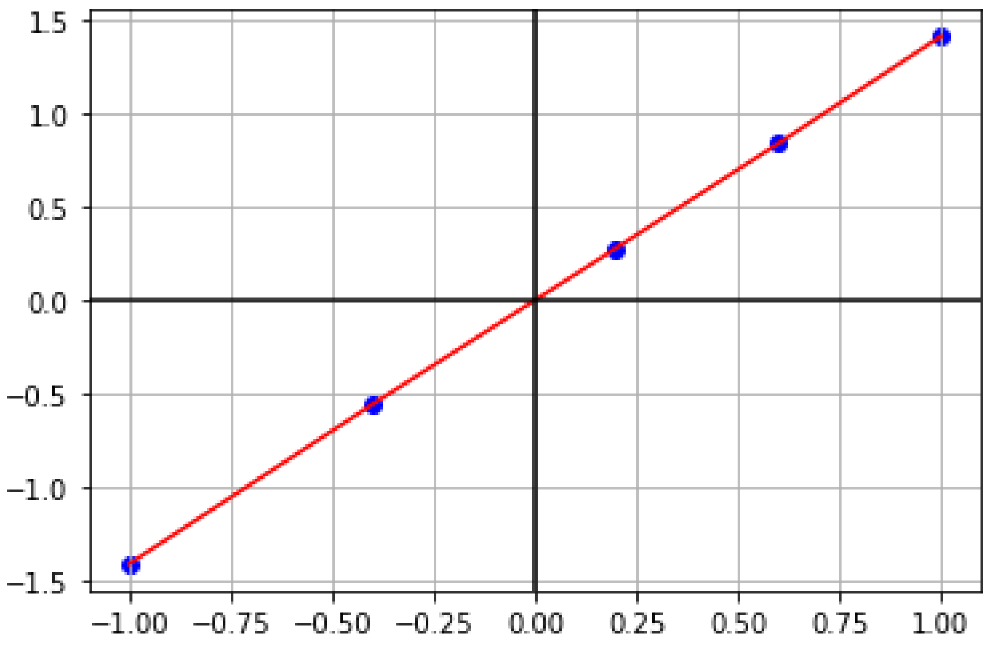


3.4. Анализ таблично заданной функции

i	0	1	2	3	4
x_i	-1.0	-0.4	0.2	0.6	1.0
y_i	-1.4142	-0.55838	0.27870	0.84008	1.4142

Построим ее график

```
import matplotlib.pyplot as plt


def draw(x, y):
    plt.plot(x, y, "-r")

    plt.grid(True, which='both')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')

    plt.scatter(x, y, c="blue")
    plt.show()


X = [-1.0,    -0.4,     0.2,     0.6,     1.0]
Y = [-1.4142, -0.55838, 0.27870, 0.84008, 1.4142]

draw(X, Y)
```



Как видим, это прямая. Составим ее уравнение:

```
k = (Y[0] - Y[-1]) / (X[0] - X[-1])
b = Y[-1] - k * X[-1]

print(f"f(x) = {k}*x + {b}")
```

```
f(x) = 1.4142*x + 0.0
```

Первая и вторая производные соответственно равны:

```python
x_0 = 0.2
print(f"f'({x_0}) = {k:0.2f}")
print(f"f''({x_0}) = {0:0.2f}")
```

```
f'(0.2) = 1.41
f''(0.2) = 0.00
```

Численное решение

Теперь запустите в терминале программу на **C++** при помощи команд

```
make
make run
```

В появившемся файле `answer_NN.txt` можно увидеть полученное численное решение:

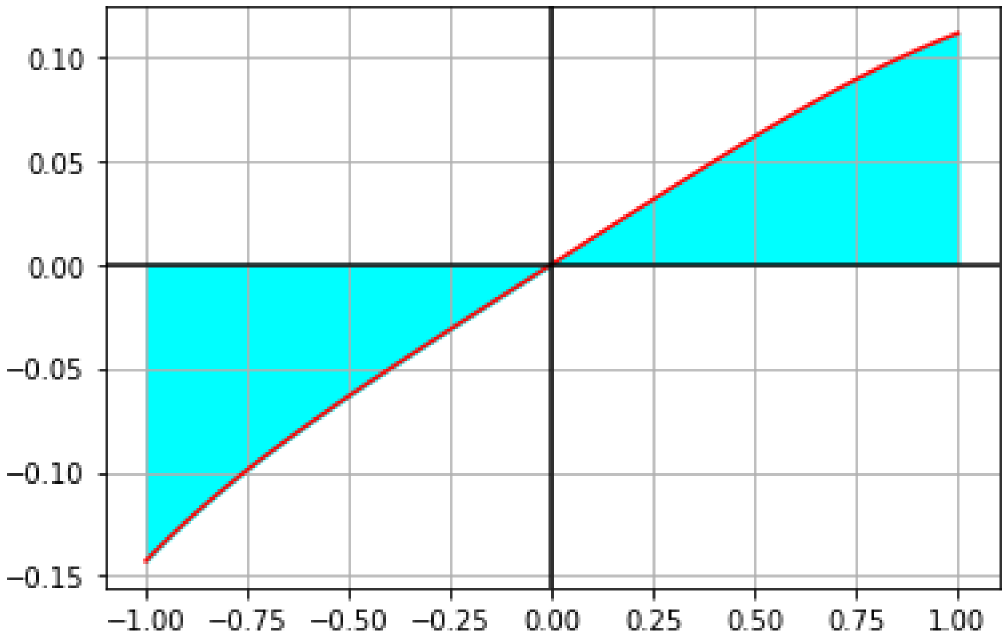
```python
answer_nn = [1.4001, 0.0166]
print(f"f'({x_0}) = {answer_nn[0]:0.2f}")
print(f"f''({x_0}) = {answer_nn[1]:0.2f}")
```

```
f'(0.2) = 1.40
f''(0.2) = 0.02
```

3.5. Анализ функции

Посмотрим на график функции:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-1, 1, 100)
y = x / (x ** 3 + 8.0) # вариант 12

plt.plot(x, y,color="red")
plt.fill_between(x, y, np.zeros_like(y), color='cyan')

plt.grid(True, which='both')
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')

plt.show()
```



Посмотрим на значение интеграла:

```
from scipy.integrate import quad
def integrand(x):
    return x / (x ** 3 + 8.0)

I, EPS = quad(integrand, -1, 1)
print(I)
```

```
-0.006294843240543573
```

Численное решение

Теперь запустите в терминале программу на **C++** при помощи команд

```
make
make run
```

В появившемся файле answer_NN.txt можно увидеть полученные численные решения:

```
half_step = [-0.00501867,
             -0.00891331,
             -0.00631688]

quarter_step = [-0.0059615,
                -0.00696599,
                -0.00629633]
print("Значения интеграла, полученные с шагом 0.5:", half_step)
print("Значения интеграла, полученные с шагом 0.25:", quarter_step)
```

```
Значения интеграла, полученные с шагом 0.5: [-0.00501867, -0.00891331, -
0.00631688]
Значения интеграла, полученные с шагом 0.25: [-0.0059615, -0.00696599, -
0.00629633]
```

4. Выводы.

В ходе данной лабораторной работы я изучил базовые численные методы, решающие задачи приближения функций, численного дифференцирования и интегрирования.

Стоит отметить, что те варианты реализаций, которые были написаны мной на C++ носят скорее учебный характер, так как передо мной стояла задача понять алгоритмы при реализации, а не написать максимально оптимальные решатели тех или иных задач.