МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Ничипорук Роман Олегович

Приближение функций

Отчет по лабораторной работе № 1

Вариант 10

Численные методы

студента 3-го курса 4-ой группы

Преподаватель:
Левчук Е.А.

2024

ОГЛАВЛЕНИЕ

ГЛАВА 1

Введение

Лабораторная работа была выполнена с использованием языка програмирования Python.

При реализации функций были применены следующие модули:

- math - модуль предоставляет обширный функционал для работы с числами;
- os - модуль предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС, поэтому программы остаются переносимыми;
- subprocess - отвечает за выполнение следующих действий: порождение новых процессов, соединение с потоками стандартного ввода, стандартного вывода, стандартного вывода сообщений об ошибках и получение кодов возврата от этих процессов;
- functools - сборник функций высокого уровня: взаимодействующих с другими функциями или возвращающие другие функции;
- typing - модуль обеспечивает поддержку выполнения аннотации типов.

Вывод графиков осуществлялся с использованием исполняемого файла plot.exe.

ГЛАВА 2

Задание 1

2.1 Условие

1. Написать программу, которая для заданных в варианте функций строит интерполяционный многочлен Ньютона по равномерной сетке узлов.

2. С помощью написанной программы для каждой из функций построить интерполяционные многочлены степени n = 2, 4, 8, 16. Вывести аналитическое представление многочлена 2-й степени (в форме Ньютона).

3. Для каждой из функций построить 4 графика для сравнения интерполируемой функции и интерполяционного многочлена (см. пример ниже). Если построение графиков в вашем языке программирования слишком трудоемко, то можно воспользоваться сторонними программами. Например: в своей программе сделать таблицу значений аргумента и соответствующих значений функции, сохранить ее в файл, затем этот файл импортировать в программу для построения графиков (например, Excel).

2.2 Использованная теория

Случай неравноотстоящих узлов

Если все расстояния между соседними узлами различны, то многочлен Ньютона строится по формуле:

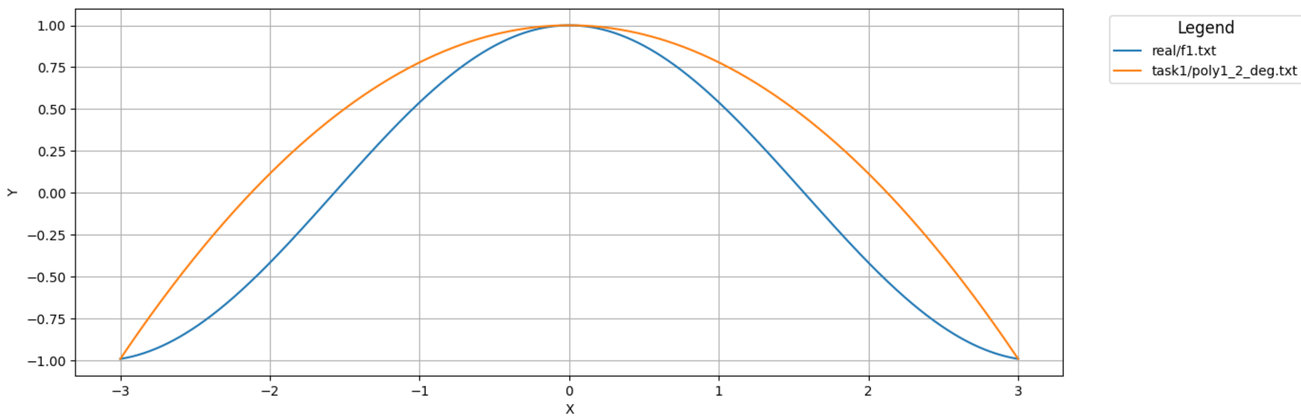
$$P_n = f(x_0) + f(x_0, x_1) * (x - x_0) + \\ f(x_0, x_1, x_1) * (x - x_0) * (x - x_1) + ... + \\ f(x_0, ..., x_n) * (x - x_0) * ... * (x - x_n) \quad (2.1)$$

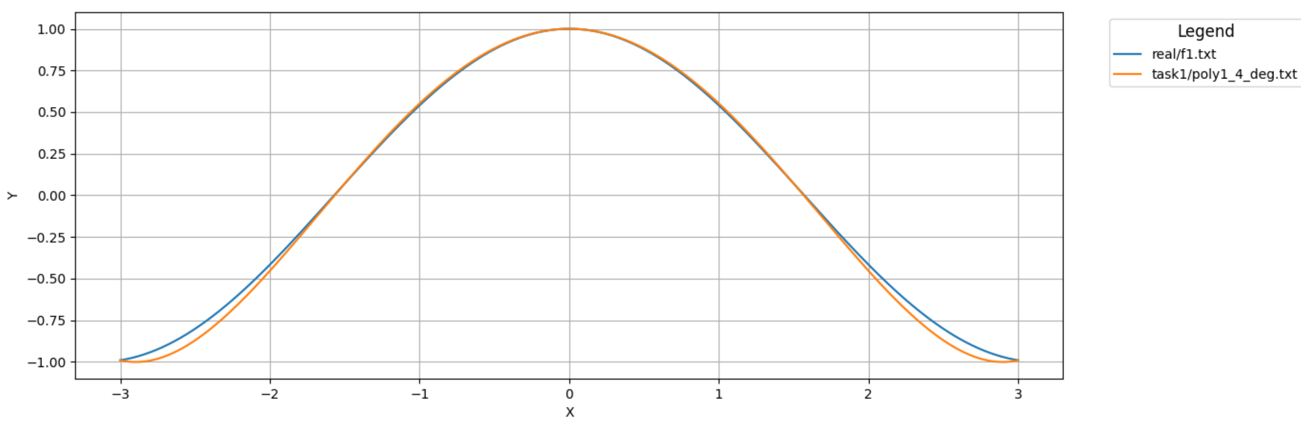
Где $f(x_0, ..., x_n)$ — разделённая разность порядка n.

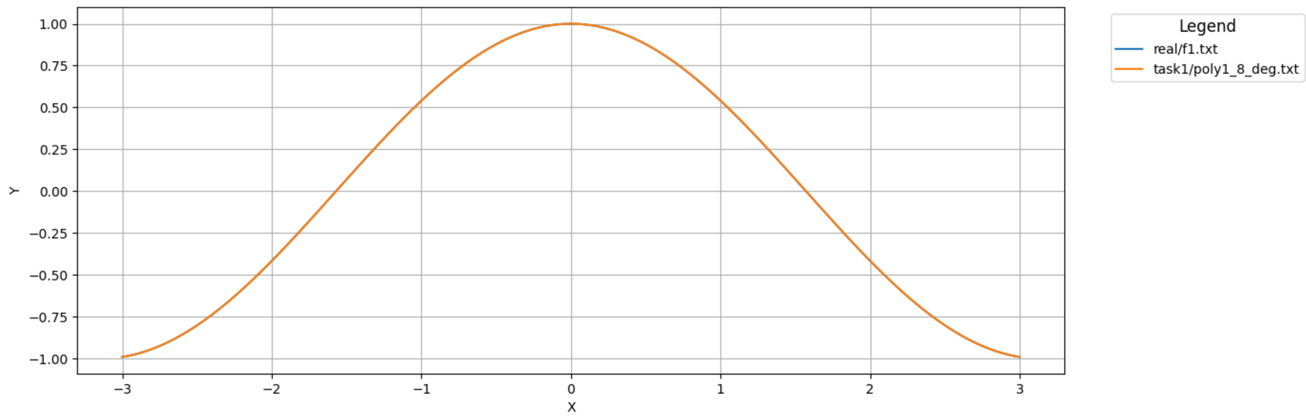
$$f(x_i, x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \quad (2.2)$$

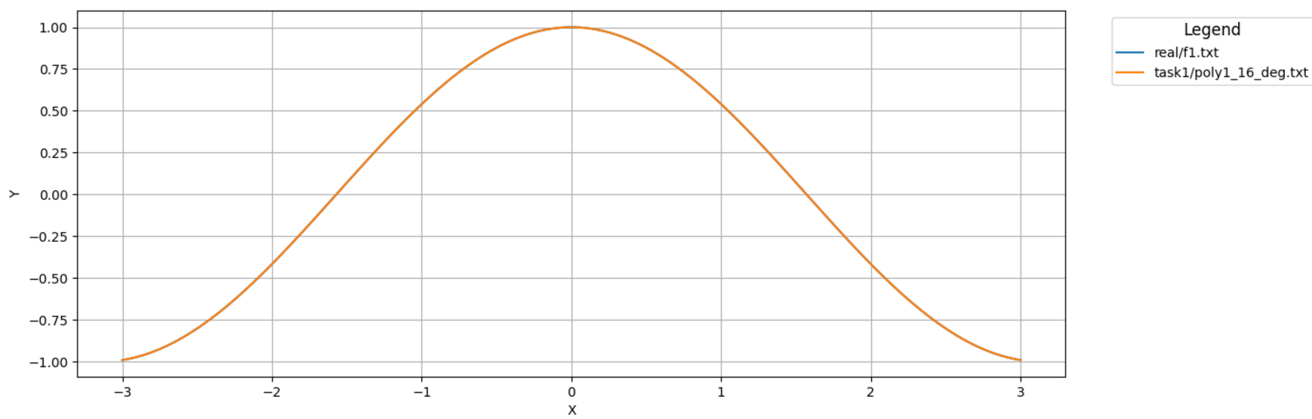
$$f(x_i, x_{i+1}, ..., x_{i+k}) = \frac{f(x_{i+1}, ..., x_{i+k}) - f(x_i, ..., x_{i+k})}{x_{i+k} - x_i} \tag{2.3}$$

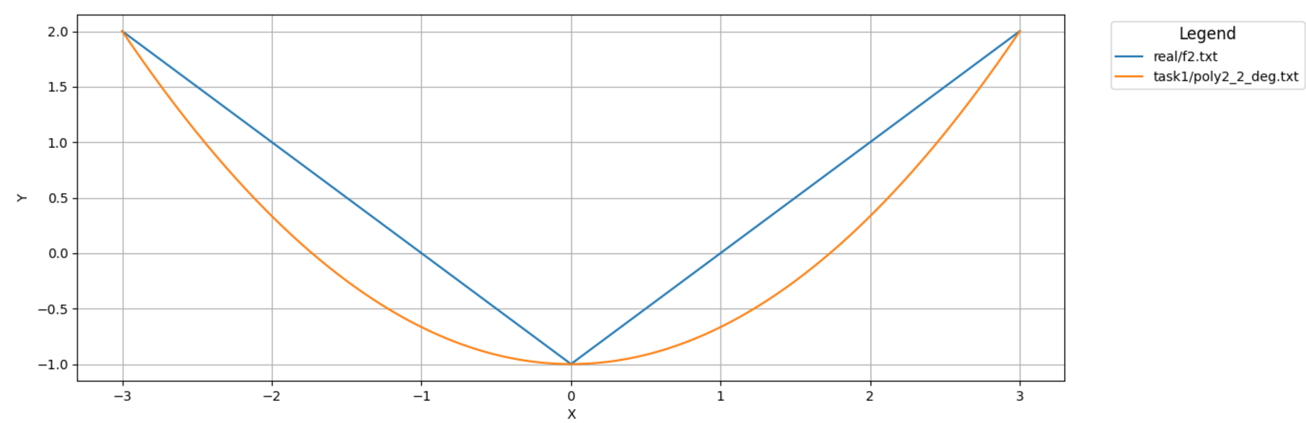
2.3 Результаты

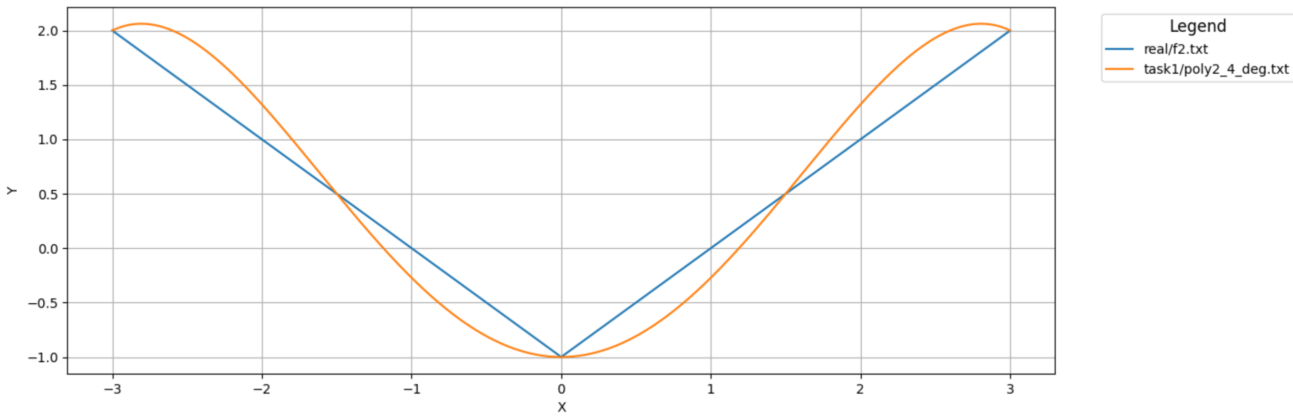


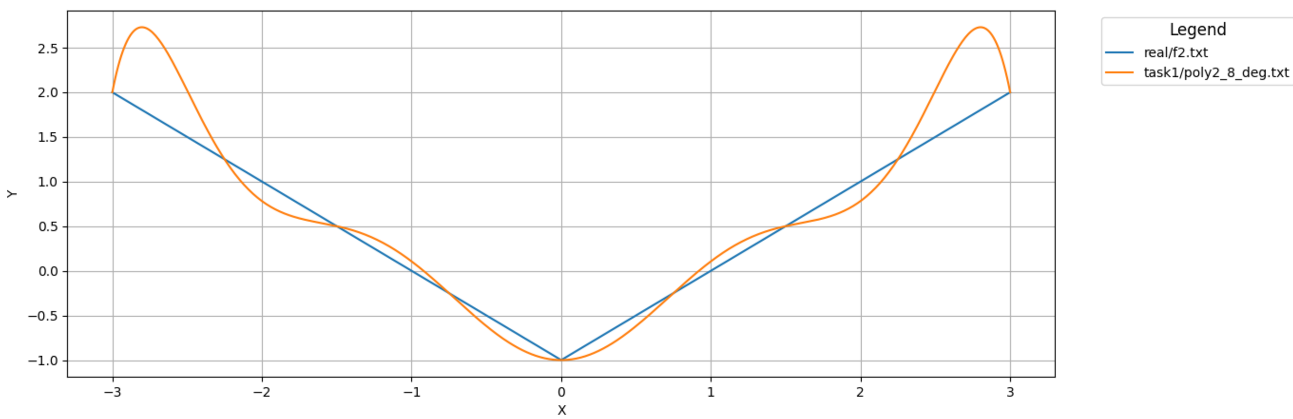


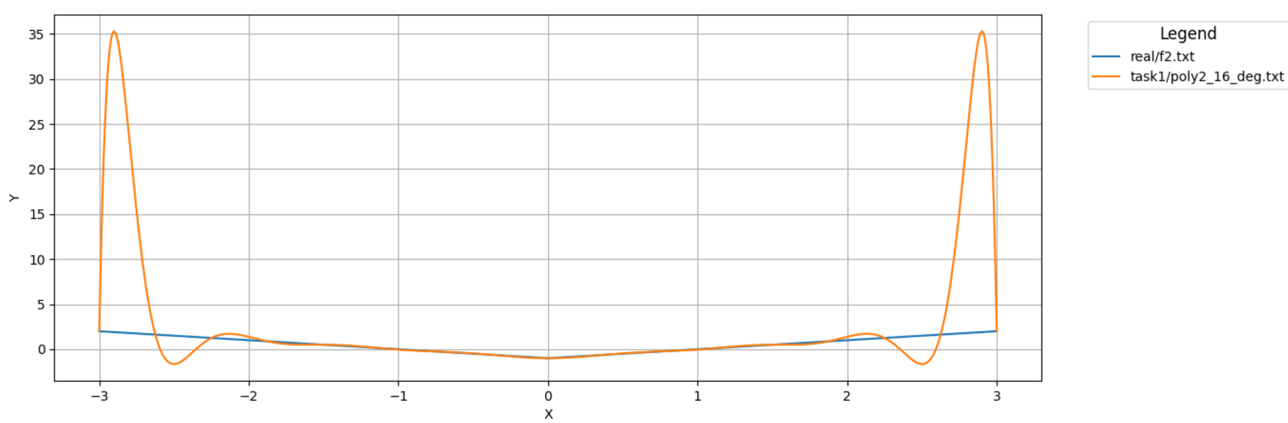

Legend
real/f1.txt
task1/poly1_8_deg.txt
Y
X


Legend
real/f1.txt
task1/poly1_16_deg.txt
Y
X


Legend
real/f2.txt
task1/poly2_2_deg.txt
Y
X


Legend
real/f2.txt
task1/poly2_4_deg.txt
X
Y


Legend
real/f2.txt
task1/poly2_8_deg.txt
X
Y


Legend
real/f2.txt
task1/poly2_16_deg.txt
X
Y

ГЛАВА 3

Задание 2

3.1 Условие

1. Написать программу, которая для заданных в варианте функций строит интерполяционный многочлен Ньютона по чебышевской сетке узлов.

2. С помощью написанной программы для каждой из функций построитьинтерполяционные многочлены степени n = 2, 4, 8, 16. Вывести аналитическое представление многочлена 2-й степени (в форме Ньютона).

3. Для каждой из функций построить 4 графика для сравнения интерполируемой функции и интерполяционного многочлена.

3.2 Использованная теория

Узлы Чебышёва

Для натурального числа n узлы Чебышёва на отрезке [-1, 1] задаются формулой:

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n. \tag{3.1}$$

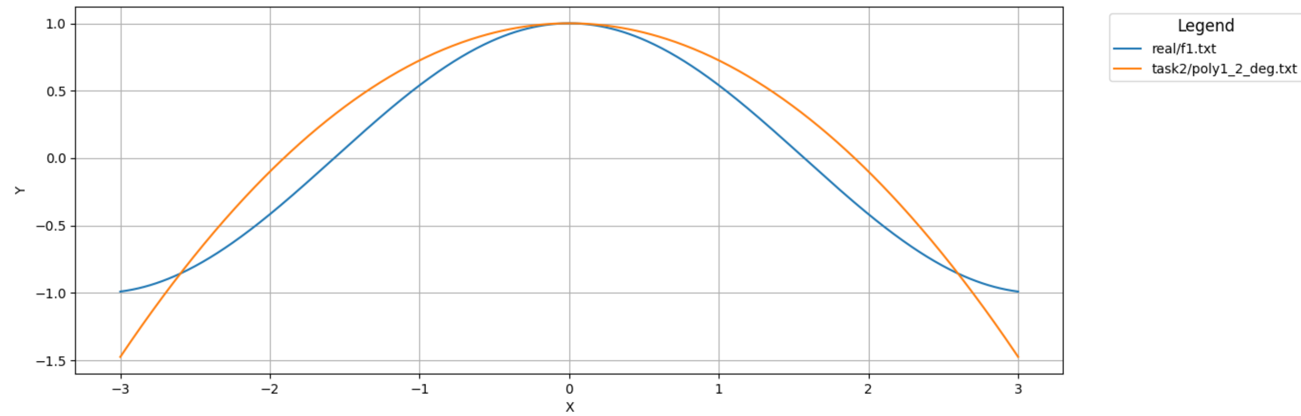
Это корни многочлена Чебышёва первого рода степени n. Для получения узлов на произвольном отрезке [a, b] можно применить аффинное преобразование отрезков:

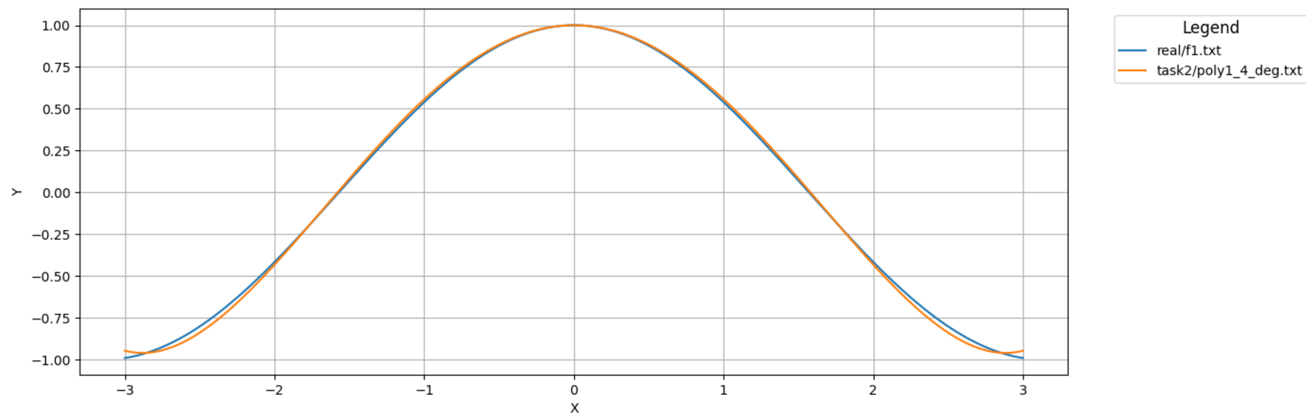
$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n. \tag{3.2}$$

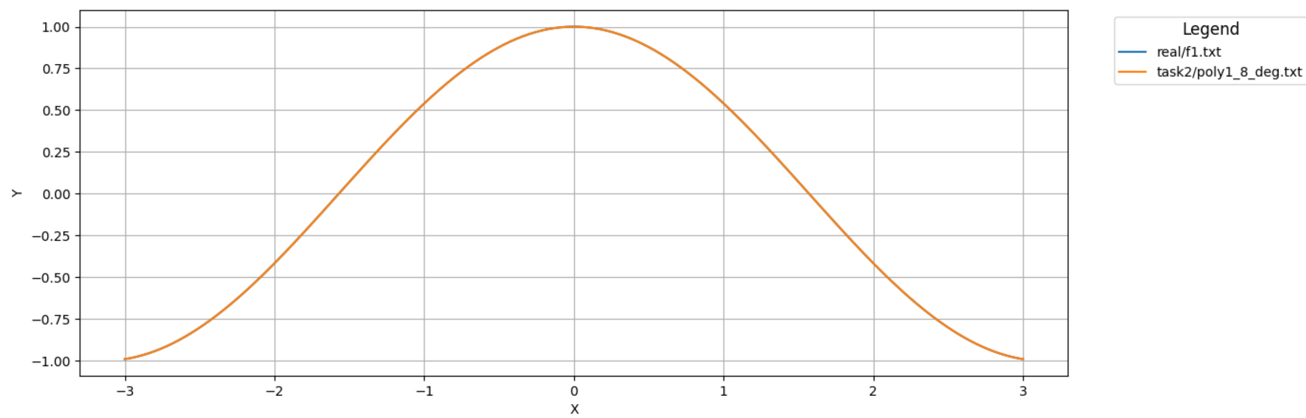
Равноотстоящие узлы

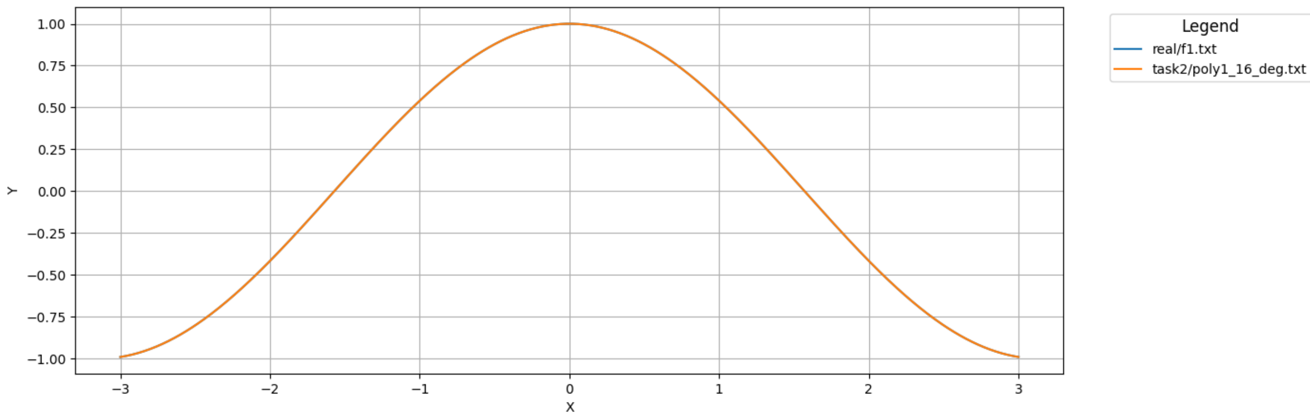
$$x_k = a + k * \frac{b-a}{n}, \quad k = 1, \ldots, n. \tag{3.3}$$

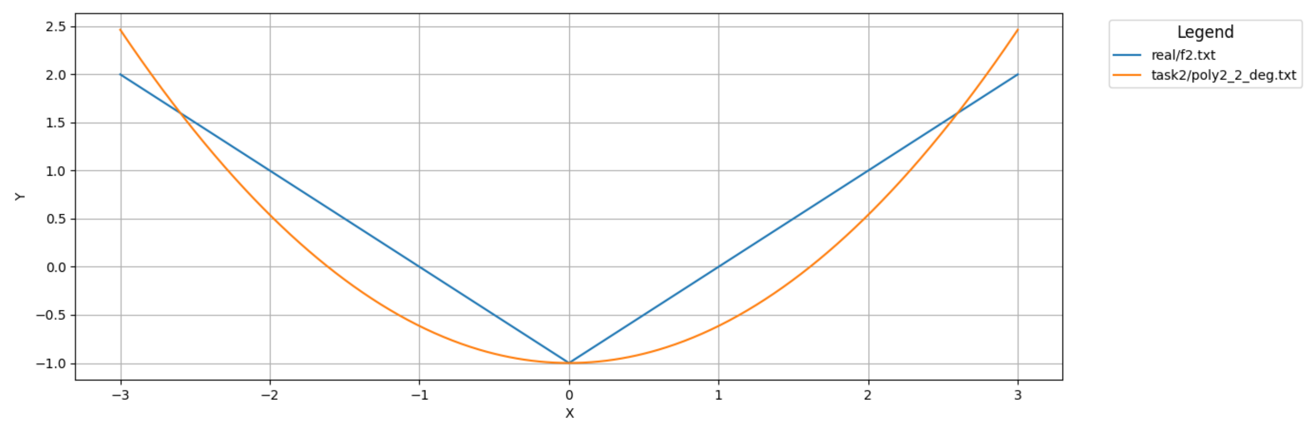
3.3 Результаты

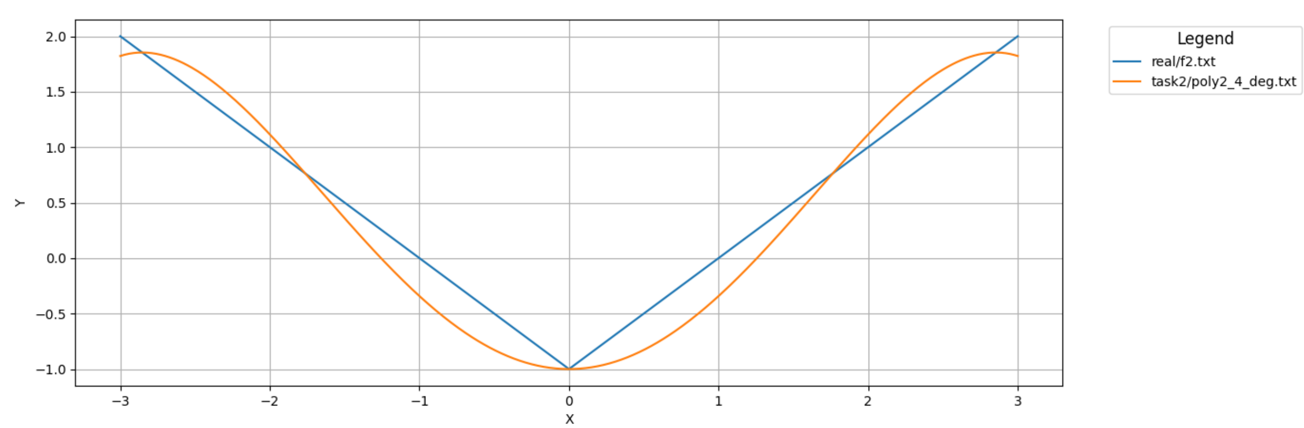


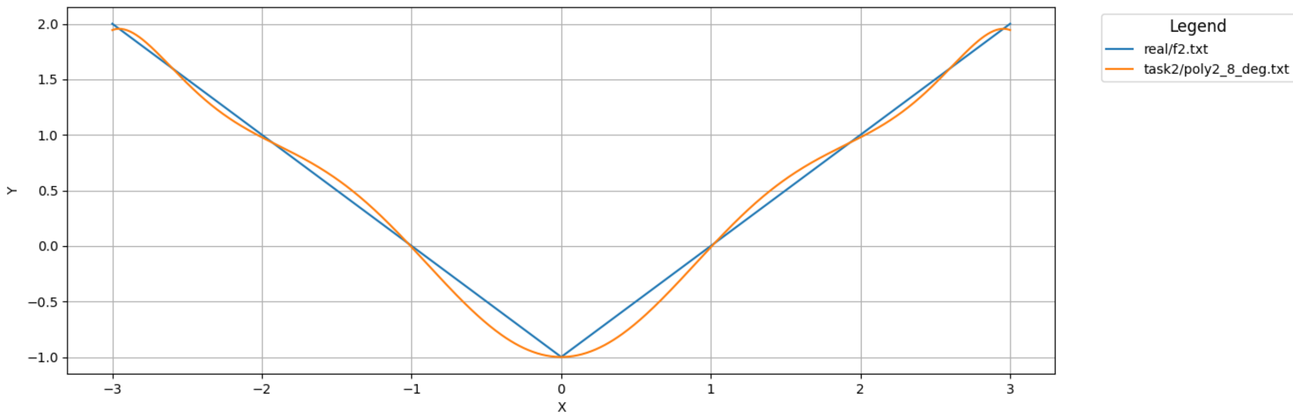


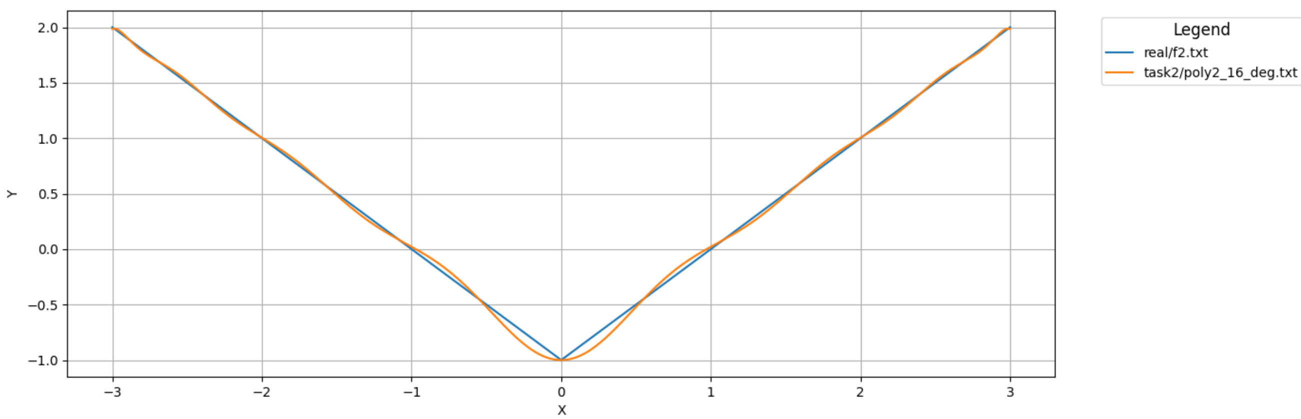



Legend
real/f1.txt
task2/poly1_16_deg.txt
X
Y


Legend
real/f2.txt
task2/poly2_2_deg.txt
X
Y


Legend
real/f2.txt
task2/poly2_4_deg.txt
X
Y


Legend
real/f2.txt
task2/poly2_8_deg.txt
Y
X


Legend
real/f2.txt
task2/poly2_16_deg.txt
Y
X

ГЛАВА 4

Задание 3

4.1 Условие

1. Написать программу, которая для заданных в варианте функций строит интерполяционный кубический сплайн по равномерной сетке узлов. В качестве дополнительных условий использовать значения вторых производных на границах отрезка. Для решения СЛАУ использовать любой подходящий метод, реализованный в прошлом семестре, или реализовать подходящий метод заново.

2. С помощью написанной программы для каждой из функций построить интерполяционные кубические сплайны по n + 1 узлам: n = 2, 4, 8, 16.

3. Для каждой из функций построить 4 графика для сравнения интерполируемой функции и интерполяционного кубического сплайна.

4.2 Использованная теория

Для построения кубических сплайнов используют каноническую запись:

$$S(x) = \begin{cases} S_i(x) = a_i + b_i * (x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6} * (x - x_i)^3 \\ x \in [x_{i-1}, x_i], \quad i = 1, ...n \end{cases} \tag{4.1}$$

Теорема 1. Для любой функции f (x), значения которой заданы на сетке x_i, $i = 0, ..., n$ отрезка [a, b], интерполяционный кубический сплайн $S(x)$ с граничными условиями вида:

$$S"(b) = \mu_1; \quad S"(b) = \mu_2 \tag{4.2}$$

существует и является единственным. Для нахождения его коэффициентов c_i, $i = 1, ..., n$ необходимо решить СЛАУ:

$$\begin{cases} c_0 = \mu_1 \\ c_{i-1}h_i + 2(h_i + h_{i+1})c_i + c_{i+1}h_{i+1} = 6(\frac{f_{i+1}-f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i})c_n = \mu_2 \end{cases} \tag{4.3}$$

и затем вычислить остальные его коэффициенты a_i, b_i, d_i , $i = 1, ..., n$ по формулам:

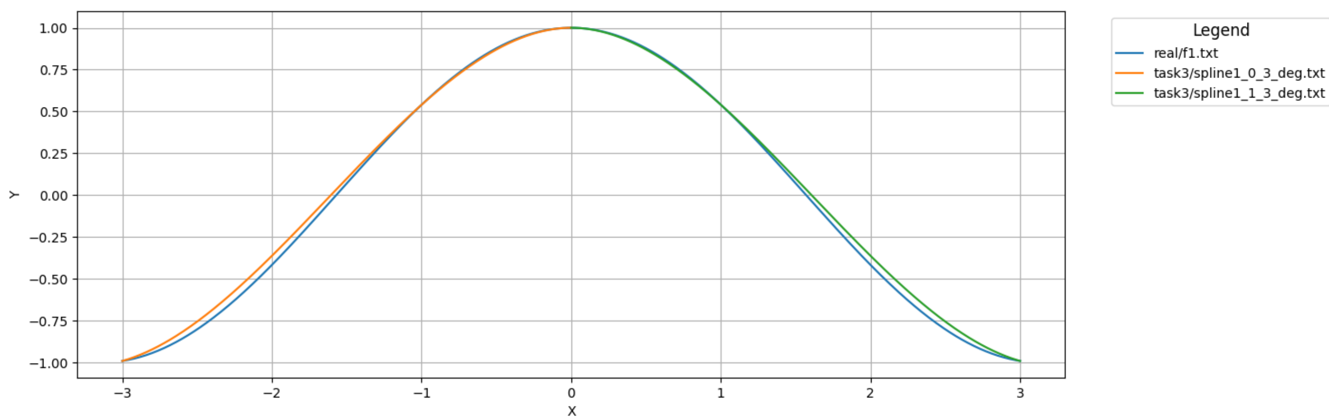
$$a_i = f_i, \quad i = 1, ..., n \tag{4.4}$$

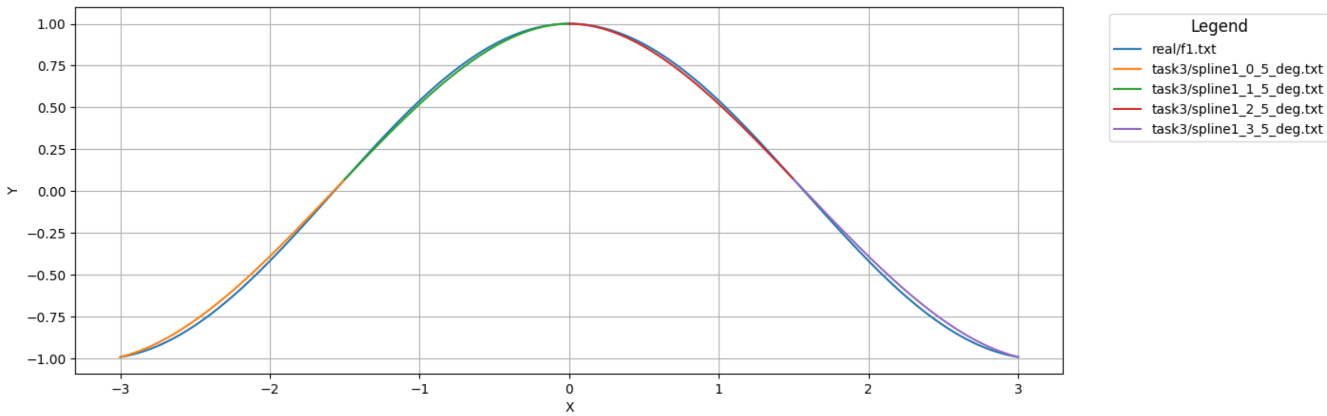
$$d_i = \frac{c_i - c_{i-1}}{h_i}, \quad i = 1, ..., n \tag{4.5}$$

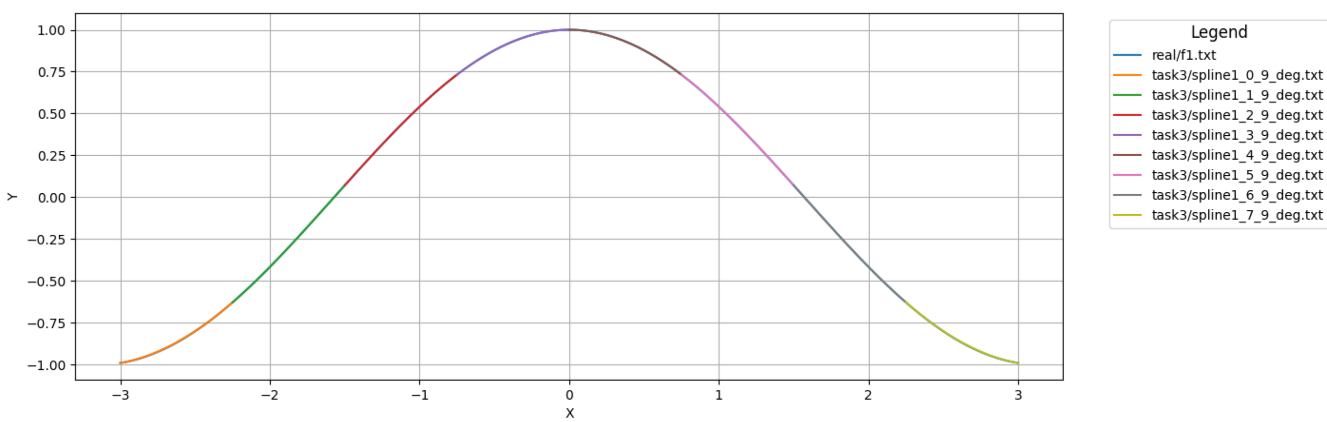
$$b_i = \frac{f_i - f_{i-1}}{h_i} + c_i\frac{h_i}{3} + c_{i-1}\frac{h_i}{6}, \quad i = 1, ..., n \tag{4.6}$$

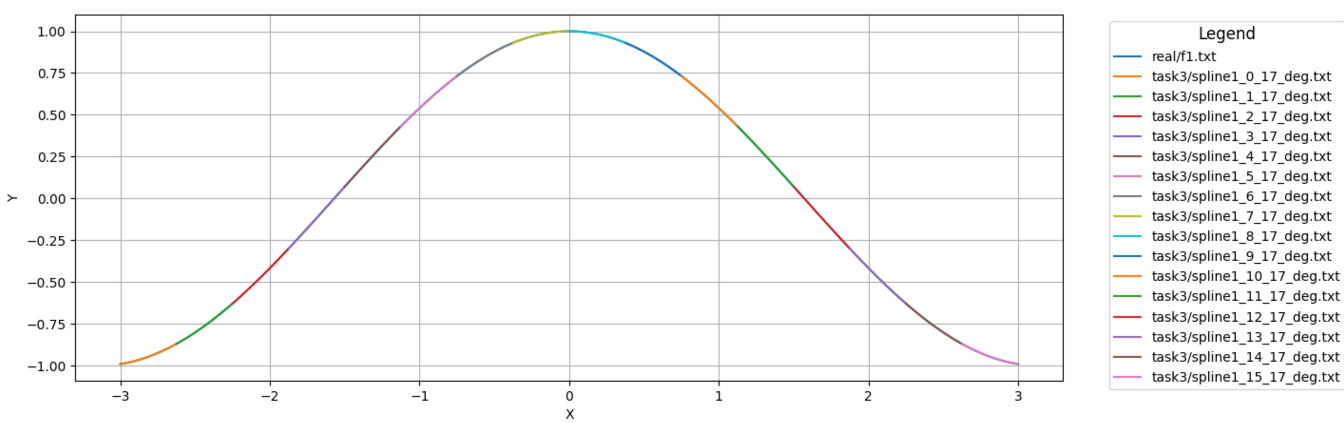
СЛАУ (4.3) может быть решена методом прогонки.

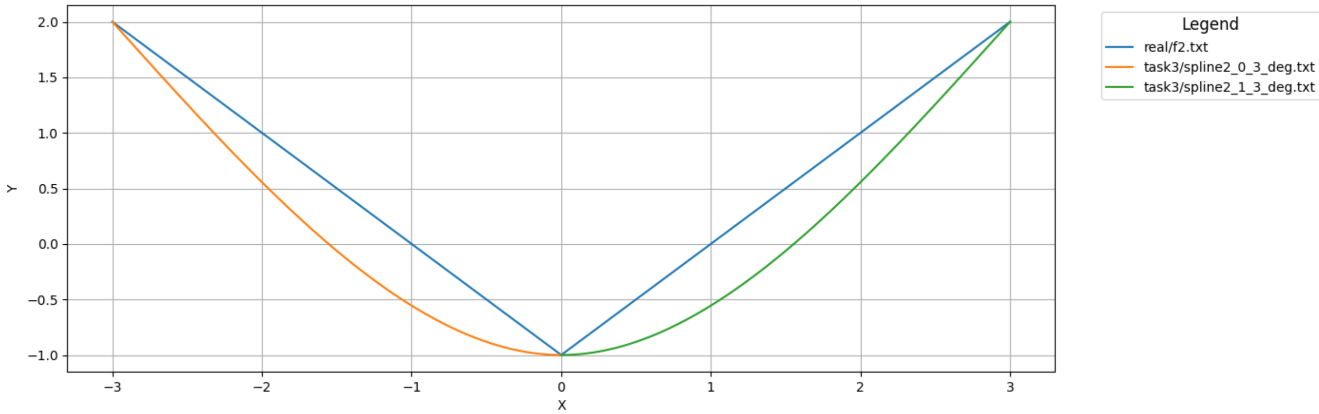
4.3 Результаты

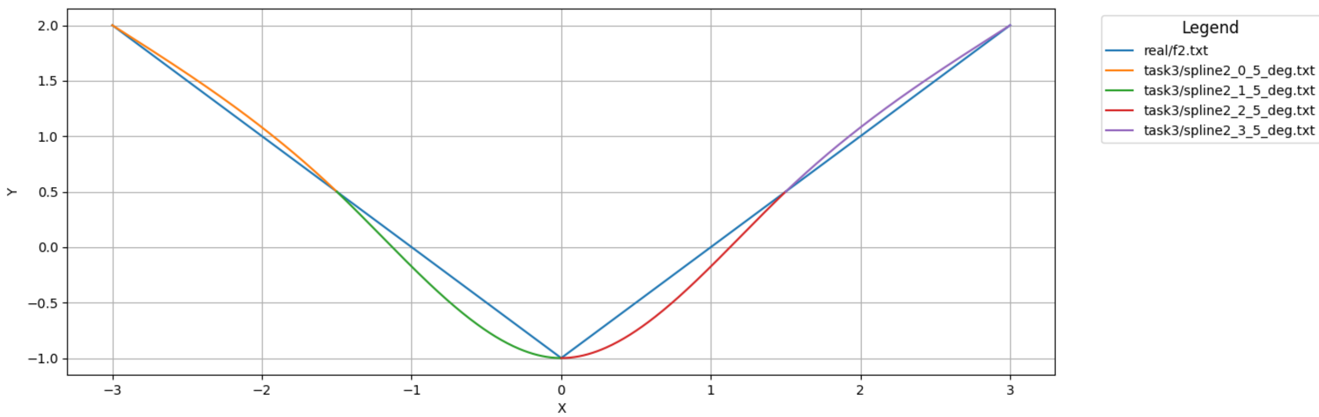


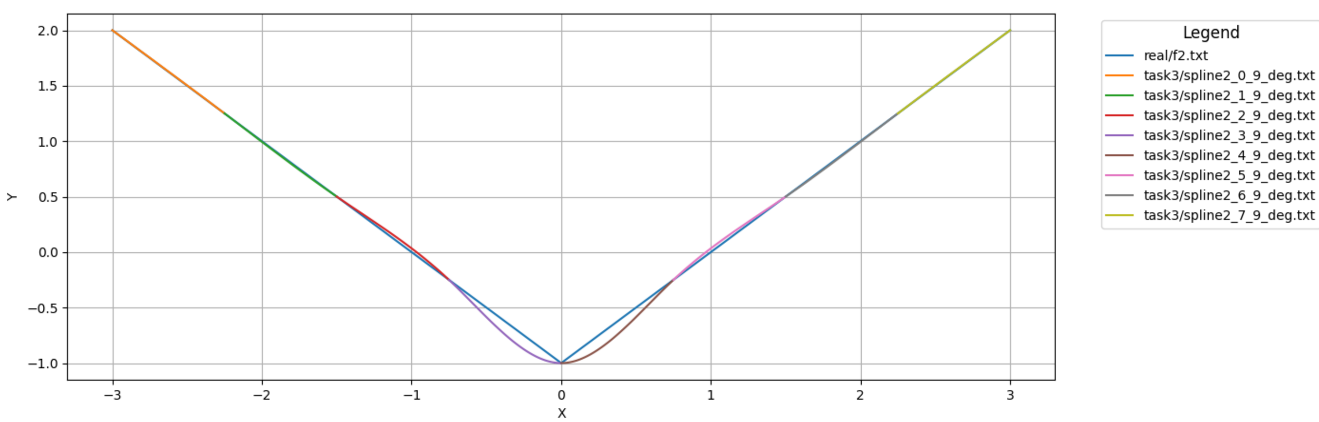

Legend
real/f1.txt
task3/spline1_0_5_deg.txt
task3/spline1_1_5_deg.txt
task3/spline1_2_5_deg.txt
task3/spline1_3_5_deg.txt
X
Y

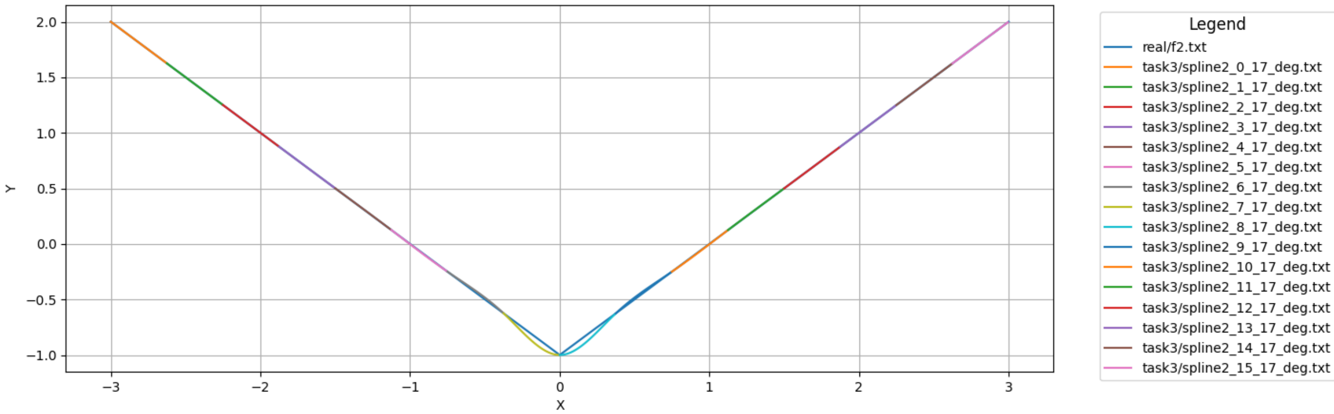

Legend
real/f1.txt
task3/spline1_0_9_deg.txt
task3/spline1_1_9_deg.txt
task3/spline1_2_9_deg.txt
task3/spline1_3_9_deg.txt
task3/spline1_4_9_deg.txt
task3/spline1_5_9_deg.txt
task3/spline1_6_9_deg.txt
task3/spline1_7_9_deg.txt
X
Y


Legend
real/f1.txt
task3/spline1_0_17_deg.txt
task3/spline1_1_17_deg.txt
task3/spline1_2_17_deg.txt
task3/spline1_3_17_deg.txt
task3/spline1_4_17_deg.txt
task3/spline1_5_17_deg.txt
task3/spline1_6_17_deg.txt
task3/spline1_7_17_deg.txt
task3/spline1_8_17_deg.txt
task3/spline1_9_17_deg.txt
task3/spline1_10_17_deg.txt
task3/spline1_11_17_deg.txt
task3/spline1_12_17_deg.txt
task3/spline1_13_17_deg.txt
task3/spline1_14_17_deg.txt
task3/spline1_15_17_deg.txt
X
Y


Legend
real/f2.txt
task3/spline2_0_3_deg.txt
task3/spline2_1_3_deg.txt
X
Y


Legend
real/f2.txt
task3/spline2_0_5_deg.txt
task3/spline2_1_5_deg.txt
task3/spline2_2_5_deg.txt
task3/spline2_3_5_deg.txt
X
Y


Legend
real/f2.txt
task3/spline2_0_9_deg.txt
task3/spline2_1_9_deg.txt
task3/spline2_2_9_deg.txt
task3/spline2_3_9_deg.txt
task3/spline2_4_9_deg.txt
task3/spline2_5_9_deg.txt
task3/spline2_6_9_deg.txt
task3/spline2_7_9_deg.txt
X
Y



Legend
real/f2.txt
task3/spline2_0_17_deg.txt
task3/spline2_1_17_deg.txt
task3/spline2_2_17_deg.txt
task3/spline2_3_17_deg.txt
task3/spline2_4_17_deg.txt
task3/spline2_5_17_deg.txt
task3/spline2_6_17_deg.txt
task3/spline2_7_17_deg.txt
task3/spline2_8_17_deg.txt
task3/spline2_9_17_deg.txt
task3/spline2_10_17_deg.txt
task3/spline2_11_17_deg.txt
task3/spline2_12_17_deg.txt
task3/spline2_13_17_deg.txt
task3/spline2_14_17_deg.txt
task3/spline2_15_17_deg.txt
X
Y

ГЛАВА 5

Листинг программы

```python
import utility

utility.run()
utility.make_plots()
```

Листинг 5.1 main.py

```python
import math
import os
import subprocess

import newton_interpolation
import spline

a = -3
b = 3
degrees = [2, 4, 8, 16]


def f1(x):
    return math.cos(x);


def d2f1(x):
    return -math.cos(x);


def f2(x):
    return math.fabs(x) - 1


def d2f2(x):
    return 0


def forward_run_through(b, c, a, vector):
    n = len(b)
    vector[0] /= c[0]
    b[0] /= -c[0]
    for i in range(1, n):
        b[i] /= -(c[i] + a[i - 1] * b[i - 1])
        vector[i] = (vector[i] - a[i - 1] * vector[i - 1]) / (c[i] + a[i - 1] * b[i - 1])
    vector[n] = (vector[n] - a[n - 1] * vector[n - 1]) / (c[n] + a[n - 1] * b[n - 1])


def reverse_run_through(b, vector):
    n = len(b)
```

```python
    solution = [0] * (n + 1)
    solution[n] = vector[n]
    for i in range(n - 1, -1, -1):
        solution[i] = b[i] * solution[i + 1] + vector[i]
    return solution


def solve_system_using_run_through_method(b, c, a, vector):
    forward_run_through(b, c, a, vector)
    return reverse_run_through(b, vector)


def write(filename, function, l, r):
    with open(filename, 'w') as file:
        for i in range(int((r - l) / 0.01) + 1):
            x = l + i * 0.01
            file.write(f"{x} {function(x)}\n")


def create_folders():
    folders = ["task1", "task2", "task3", "real"]
    for folder in folders:
        if not os.path.exists(folder):
            os.makedirs(folder)


def run():
    create_folders()
    write("real/f1.txt", f1, a, b)
    write("real/f2.txt", f2, a, b)
    for deg in degrees:
        x1 = newton_interpolation.get_equal_points(deg, a, b)
        x2 = newton_interpolation.get_chebyshev_points(deg, a, b)
        y1_1 = []
        y2_1 = []
        y1_2 = []
        y2_2 = []
        for i in range(len(x1)):
            y1_1.append(f1(x1[i]))
            y2_1.append(f2(x1[i]))
            y1_2.append(f1(x2[i]))
            y2_2.append(f2(x2[i]))
        poly1_1 = newton_interpolation.create_polynomial(x1, y1_1)
        poly2_1 = newton_interpolation.create_polynomial(x1, y2_1)
        poly1_2 = newton_interpolation.create_polynomial(x2, y1_2)
        poly2_2 = newton_interpolation.create_polynomial(x2, y2_2)
        write("task1/poly1_" + str(deg) + "_deg.txt", poly1_1, a, b)
        write("task1/poly2_" + str(deg) + "_deg.txt", poly2_1, a, b)
        write("task2/poly1_" + str(deg) + "_deg.txt", poly1_2, a, b)
        write("task2/poly2_" + str(deg) + "_deg.txt", poly2_2, a, b)
        functions1 = spline.get_spline(x1, f1, d2f1)
        functions2 = spline.get_spline(x1, f2, d2f2)
        for i in range(len(functions1)):
            write("task3/spline1_" + str(i) + "_" + str(deg + 1) + "_deg.txt",
   functions1[i], x1[i], x1[i + 1])
            write("task3/spline2_" + str(i) + "_" + str(deg + 1) + "_deg.txt",
```

```
        functions2[i], x1[i], x1[i + 1])


def make_plots():
    subprocess.run(["plot.exe", "real/f1.txt", "task1/poly1_2_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task1/poly1_4_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task1/poly1_8_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task1/poly1_16_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task1/poly2_2_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task1/poly2_4_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task1/poly2_8_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task1/poly2_16_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task2/poly1_2_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task2/poly1_4_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task2/poly1_8_deg.txt"])
    subprocess.run(["plot.exe", "real/f1.txt", "task2/poly1_16_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task2/poly2_2_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task2/poly2_4_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task2/poly2_8_deg.txt"])
    subprocess.run(["plot.exe", "real/f2.txt", "task2/poly2_16_deg.txt"])

    subprocess.call("plot.exe real/f1.txt task3/spline1_0_3_deg.txt task3/spline1_1_3_deg.txt", shell=True)
    subprocess.call("plot.exe real/f1.txt task3/spline1_0_5_deg.txt task3/spline1_1_5_deg.txt task3/spline1_2_5_deg.txt task3/spline1_3_5_deg.txt", shell=True)
    subprocess.call("plot.exe real/f1.txt task3/spline1_0_9_deg.txt task3/spline1_1_9_deg.txt task3/spline1_2_9_deg.txt task3/spline1_3_9_deg.txt task3/spline1_4_9_deg.txt task3/spline1_5_9_deg.txt task3/spline1_6_9_deg.txt task3/spline1_7_9_deg.txt", shell=True)
    subprocess.call("plot.exe real/f1.txt task3/spline1_0_17_deg.txt task3/spline1_1_17_deg.txt task3/spline1_2_17_deg.txt task3/spline1_3_17_deg.txt task3/spline1_4_17_deg.txt task3/spline1_5_17_deg.txt task3/spline1_6_17_deg.txt task3/spline1_7_17_deg.txt task3/spline1_8_17_deg.txt task3/spline1_9_17_deg.txt task3/spline1_10_17_deg.txt task3/spline1_11_17_deg.txt task3/spline1_12_17_deg.txt task3/spline1_13_17_deg.txt task3/spline1_14_17_deg.txt task3/spline1_15_17_deg.txt", shell=True)
    subprocess.call("plot.exe real/f2.txt task3/spline2_0_3_deg.txt task3/spline2_1_3_deg.txt", shell=True)
    subprocess.call("plot.exe real/f2.txt task3/spline2_0_5_deg.txt task3/spline2_1_5_deg.txt task3/spline2_2_5_deg.txt task3/spline2_3_5_deg.txt", shell=True)
    subprocess.call("plot.exe real/f2.txt task3/spline2_0_9_deg.txt task3/spline2_1_9_deg.txt task3/spline2_2_9_deg.txt task3/spline2_3_9_deg.txt task3/spline2_4_9_deg.txt task3/spline2_5_9_deg.txt task3/spline2_6_9_deg.txt task3/spline2_7_9_deg.txt", shell=True)
    subprocess.call("plot.exe real/f2.txt task3/spline2_0_17_deg.txt task3/spline2_1_17_deg.txt task3/spline2_2_17_deg.txt task3/spline2_3_17_deg.txt task3/spline2_4_17_deg.txt task3/spline2_5_17_deg.txt task3/spline2_6_17_deg.txt task3/spline2_7_17_deg.txt task3/spline2_8_17_deg.txt task3/spline2_9_17_deg.txt task3/spline2_10_17_deg.txt task3/spline2_11_17_deg.txt task3/spline2_12_17_deg.txt task3/spline2_13_17_deg.txt task3/spline2_14_17_deg.txt task3/spline2_15_17_deg.txt", shell=True)
```

Листинг 5.2 utility.py

```python
import math
from functools import partial
from typing import List

def newton_interpolation(x, y):
    n = len(x)
    divided_differences = y[:]
    res = []
    for i in range(1, n + 1):
        next_divided_differences = []
        for j in range(n - i):
            divided_difference = (divided_differences[j + 1] - divided_differences[j]) / (x[j + i] - x[j])
            next_divided_differences.append(divided_difference)
        res.append(divided_differences[0])
        divided_differences = next_divided_differences
    return res


def create_polynomial(x: List[float], y: List[float]):
    coeffs = newton_interpolation(x, y)
    if len(x) == 3:
        print("\nAnalytical representation of the 2nd-degree polynomial: ", end="")
        print(f"{coeffs[0]} + {coeffs[1]} * (x - {x[0]}) + {coeffs[2]} * (x - {x[0]}) * (x - {x[1]})")

    def polynomial(point):
        res = 0
        for i in range(len(x)):
            term = coeffs[i]
            for j in range(i):
                term *= (point - x[j])
            res += term
        return res

    return partial(polynomial)


def get_equal_points(n, a, b):
    step = (b - a) / n
    points = []
    for i in range(n + 1):
        points.append(a + i * step)
    return points


def get_chebyshev_points(n, a, b):
    step = (b - a) / 2
    middle = (a + b) / 2
    points = []
    for i in range(n + 1):
        point = middle + step * math.cos(((2 * i + 1) * math.pi) / (2 * (n + 1)))
        points.append(point)
```

```
    return points
```

Листинг 5.3 newton$_{i}nterpolation.py$

```
import utility


def get_spline(x, f, d2f):
    n = len(x)
    h = x[1] - x[0]
    vector = [0] * n
    ac = [0] * (n - 1)
    cc = [0] * n
    bc = [0] * (n - 1)
    matrix = [[0] * n for _ in range(n)]
    cc[0] = 1
    cc[n - 1] = 1
    vector[0] = d2f(x[0])
    vector[n - 1] = d2f(x[n - 1])
    for i in range(1, n - 1):
        vector[i] = 6 * ((f(x[i + 1]) - f(x[i])) / h - (f(x[i]) - f(x[i - 1])) / h)
        ac[i - 1] = h
        cc[i] = 2 * (h + h)
        bc[i] = h
    c = utility.solve_system_using_run_through_method(bc, cc, ac, vector)
    functions = []
    for i in range(1, n):
        temp = i
        functions.append(lambda point, f=f, c=c, x=x, h=h, temp=temp: \
            f(x[temp]) + ((f(x[temp]) - f(x[temp - 1])) / h + c[temp] * h / 3 + c[temp - 1] * h / 6) * (point - x[temp]) + \
            c[temp] / 2 * (point - x[temp]) * (point - x[temp]) + \
            (c[temp] - c[temp - 1]) / (6 * h) * (point - x[temp]) * (point - x[temp]) * (point - x[temp]))

    return functions
```

Листинг 5.4 spline.py

ГЛАВА 6

Вывод

Сравнивая f1 и f2 можно заметить, что все методы приближения работают лучше для f1, так как на отрезке [-2,2] она является более «простой». Сравнивая методы приближения лучше всего сработал метод из Задания 3. Также стоит отметить тот факт, что при увелечении n увеличивается точность приближения. Касаемо выбора построения сетки узлов сложно однозначно определить лучший, но не исключено преимущество какого-либо из построений на функциях, отличных от f1 и f2.