Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

Лабораторная работа №1

Тема: Алгоритм и программа построения интерполяционных полиномов Ньютона и Эрмита

Студент: Миронов Г.А.

Группа: ИУ7-43Б

Оценка (баллы): _____

Преподаватель: Градов В.М.

Москва.
2020 г.

Цель работы

Изучить метод нахождения значения функции в заданной точке с помощью интерполяционных полиномов Ньютона и Эрмита.

Задание

1. Найти **Pn(x)** и **Hn(x)**
2. Сравнить результаты вычисления значения функции обоими методами
3. Найти корень функции методом обратной интерполяции

Входные данные

1. Таблица координат
2. Координата точки по оси абсцисс
3. Степень искомых полиномов

Выходные данные

1. Значение функции в точке X, найденное двумя методами
2. Корень функции, найденный с помощью метода обратной интерполяции

Анализ алгоритма

В алгоритме подсчитываются разделенные разности.
Они вычисляются по формуле (1, 2, 3 степени):

$$y(x_i, x_j) = [y(x_i) - y(x_j)] / (x_i - x_j),$$

$$y(x_i, x_j, x_k) = [y(x_i, x_j) - y(x_j, x_k)] / (x_i - x_k),$$

$$y(x_i, x_j, x_k, x_l) = [y(x_i, x_j, x_k) - y(x_j, x_k, x_l)] / (x_i - x_l)$$

Далее с помощью этих разделенных разностей подсчитывается полином Ньютона, имеющий формулу:

$$P_n(x) = y_0 + \sum_{k=0}^{n} (x - x_n) \dots (x - x_{k-1}) y(x_0, x_1, \dots, x_k)$$

При нахождении полинома Эрмита, производные функции в точке рассматриваются в качестве предельного перехода от разделенных разностей, следовательно:

$$y(x_0, x_0) = \lim_{x_1 \to x_0} \frac{y(x_0) - y(x_1)}{x_0 - x_1} = y'(x_0),$$

$$y(x_0, x_0, x_1) = \frac{y(x_0, x_0) - y(x_0, x_1)}{x_0 - x_1} = \frac{y'(x_0) - y(x_0, x_1)}{x_0 - x_1},$$

$$y(x_0, x_0, x_1, x_1) = \frac{y(x_0, x_0, x_1) - y(x_0, x_1, x_1)}{x_0 - x_1} = \frac{y'(x_0) - 2y(x_0, x_1) + y'(x_1)}{(x_0 - x_1)^2}.$$

И ход решения имеет вид

x_i	y_i	$y(x_k, x_m)$	$y(x_k, x_m, x_l)$
x_0	y_0	y_0'	$(y_0' - y(x_0, x_1))/(x_0 - x_1)$
x_0	y_0	$y(x_0, x_1)$	$(y(x_0, x_1) - y_1')/(x_0 - x_1)$
x_1	y_1	y_1'	и.т.д.
x_1	y_1	$y(x_1, x_2)$	
x_2	y_2	y_2'	
x_2	y_2		

При поиске корня обратной интерполяцией, столбцы меняются местами, а аргумент задается равным 0

Исходные данные

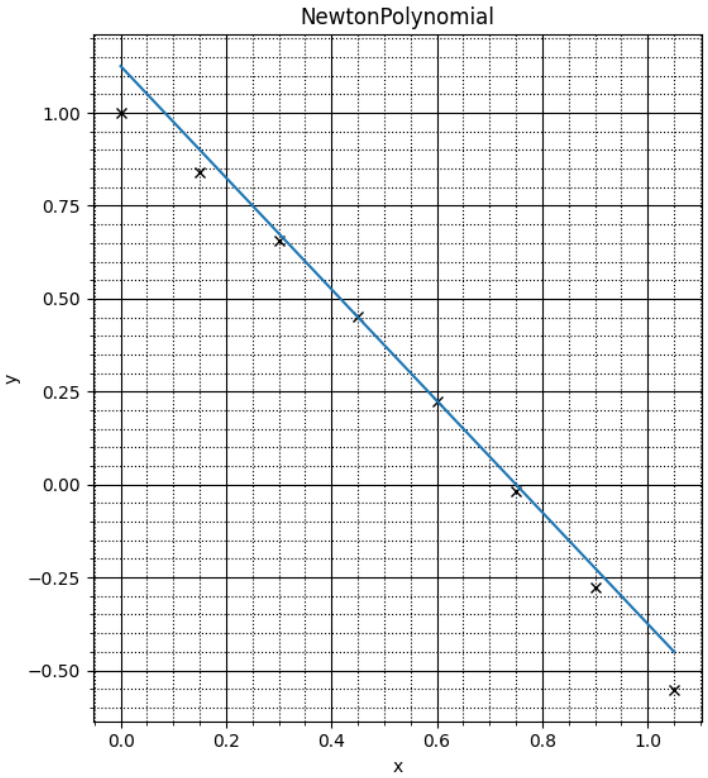
x	y	y’
0.00	1.000000	--1.000000
0.15	0.838771	-1.14944
0.30	0.655336	-1.29552
0.45	0.450447	-1.43497
0.60	0.225336	-1.56464
0.75	-0.018310	-1.68164
0.90	-0.278390	-1.78333
1.05	-0.552430	-1.86742

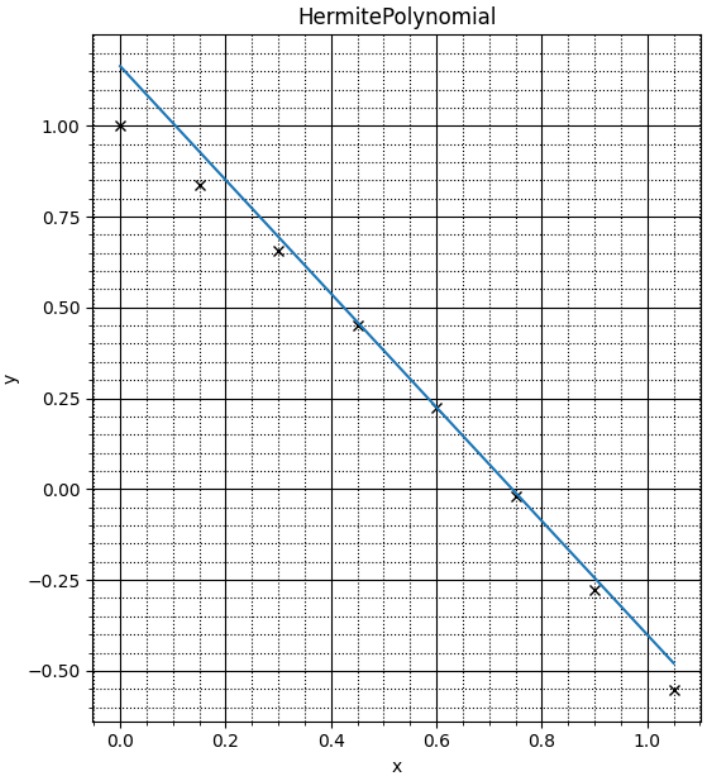
Результаты вычислений

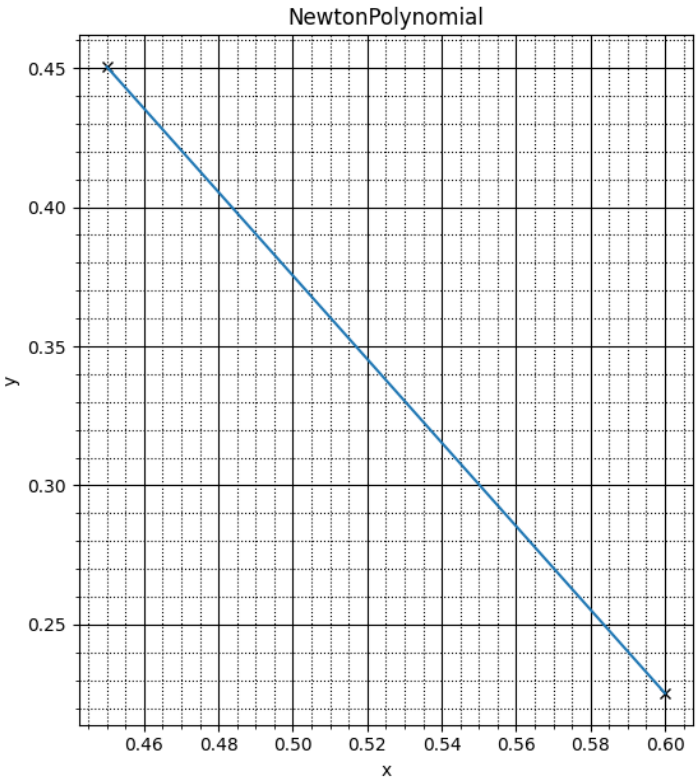
Аргумент x = 0.525	Значение функции		Корень
Степень полинома	Ньютон	Эрмит	
1	0.337891	0.342684	0.738727
2	0.340208	0.340288	0.739046
3	0.340314	0.340323	0.739095
4	0.340324	0.340324	0.739088
5	0.340324	0.340324	0.739085
6	0.340324	0.340324	0.739087

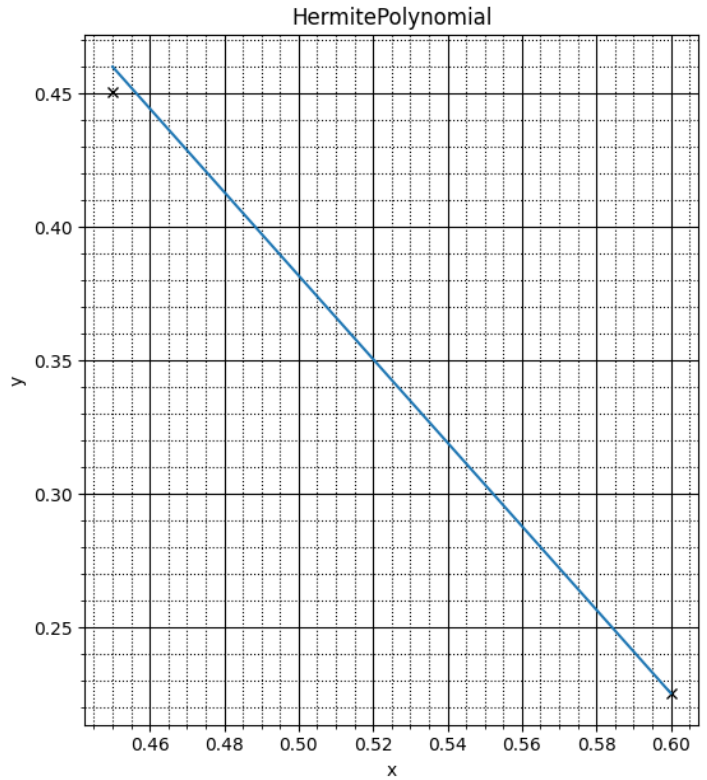
Графическое сравнение результатов

- 1 степень

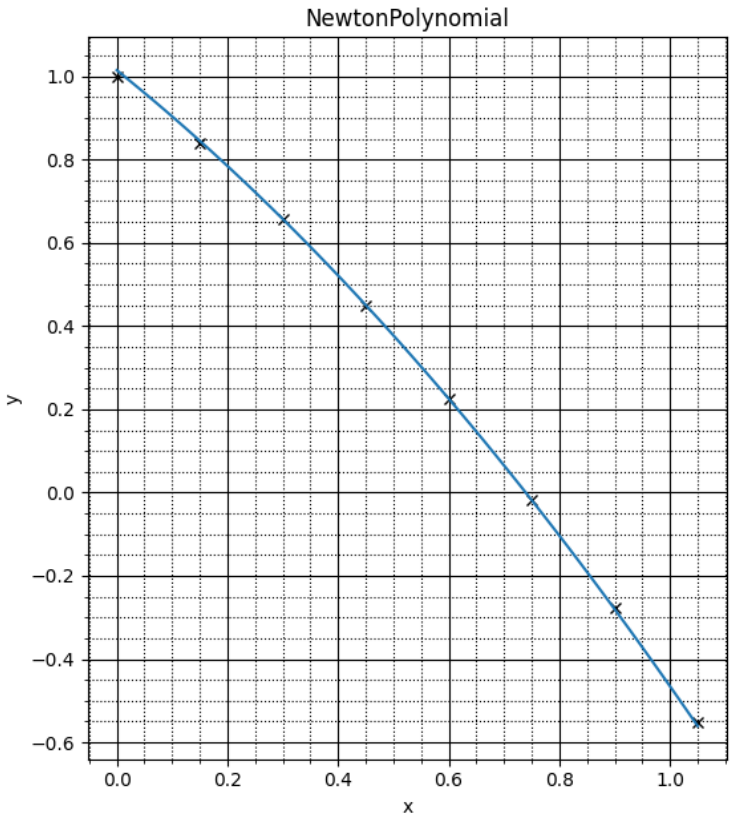


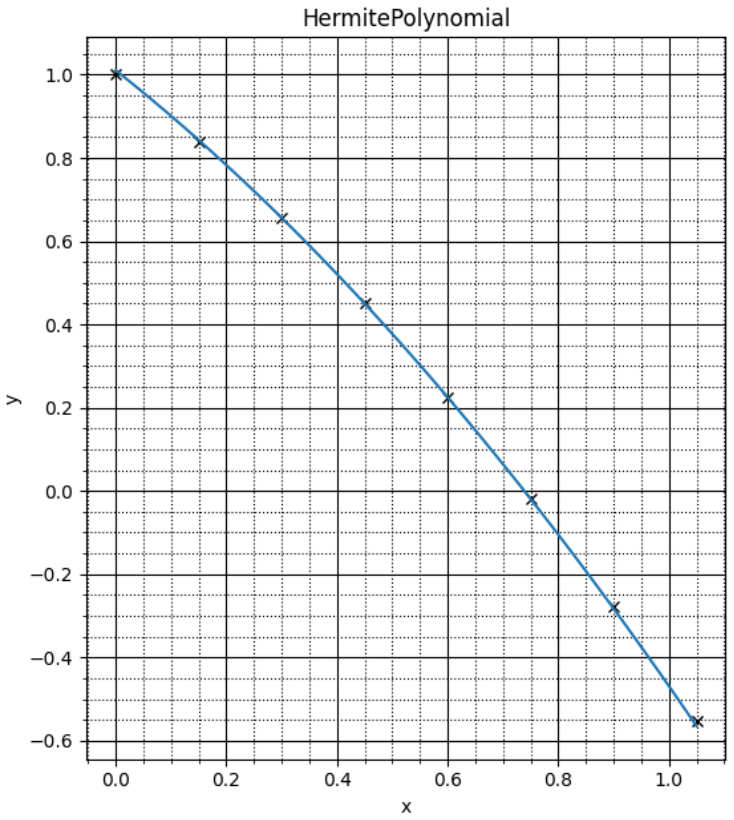


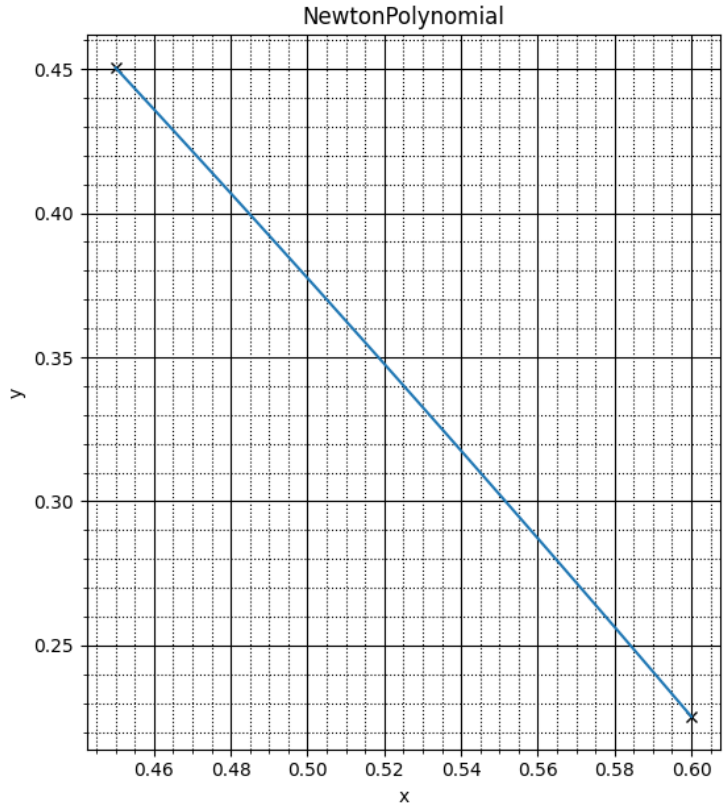


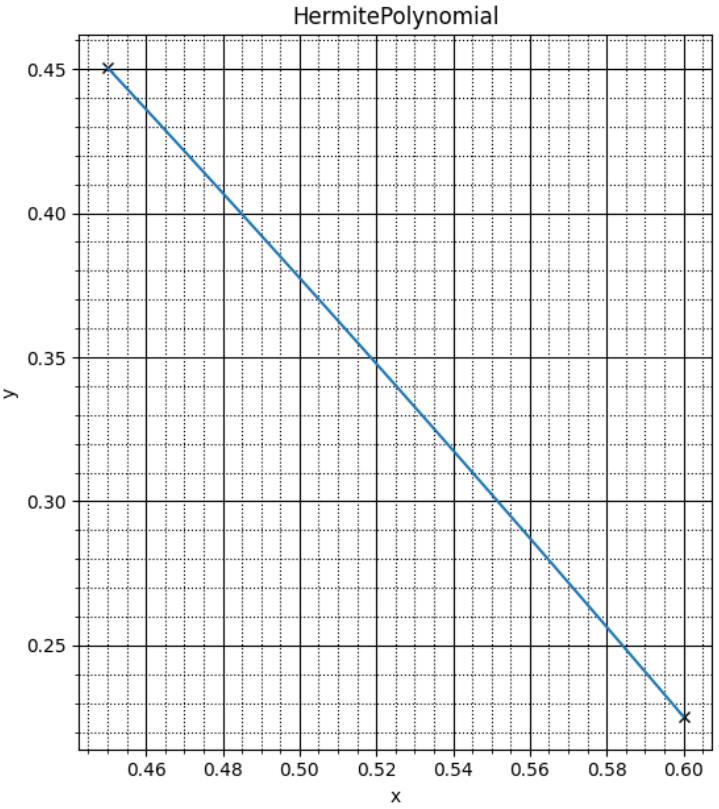


- 2 степень

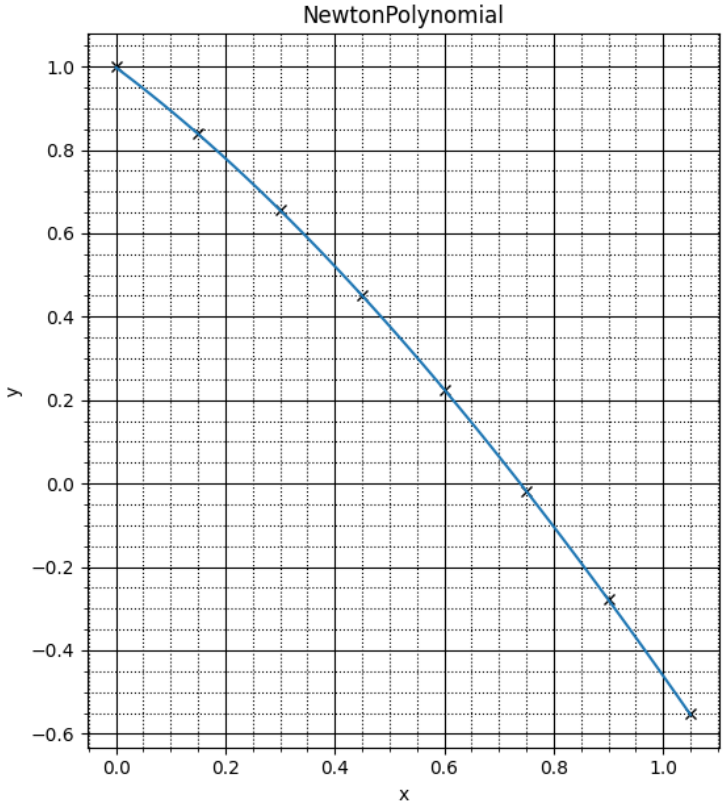

NewtonPolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x

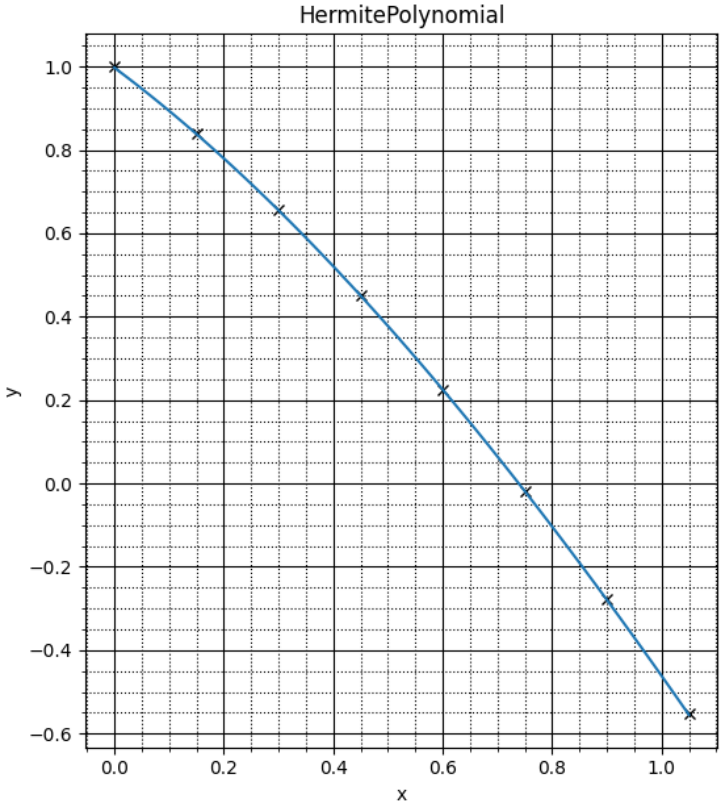

HermitePolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x

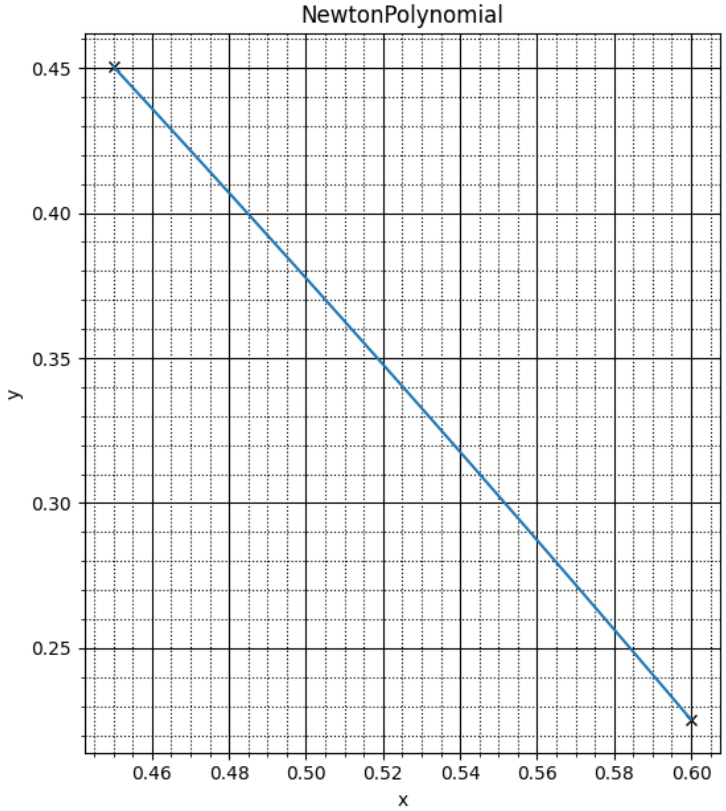

NewtonPolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

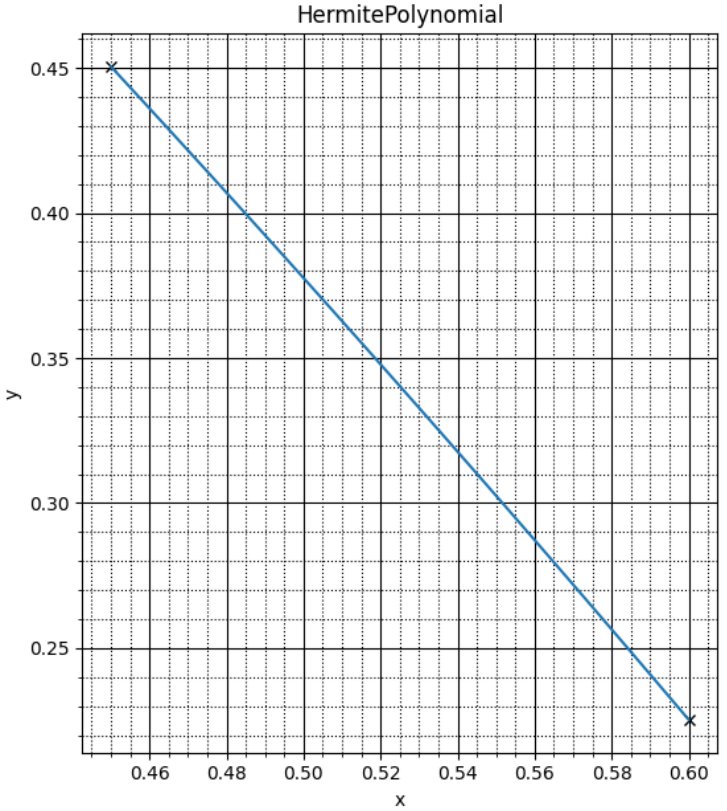

HermitePolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

- 3 степень

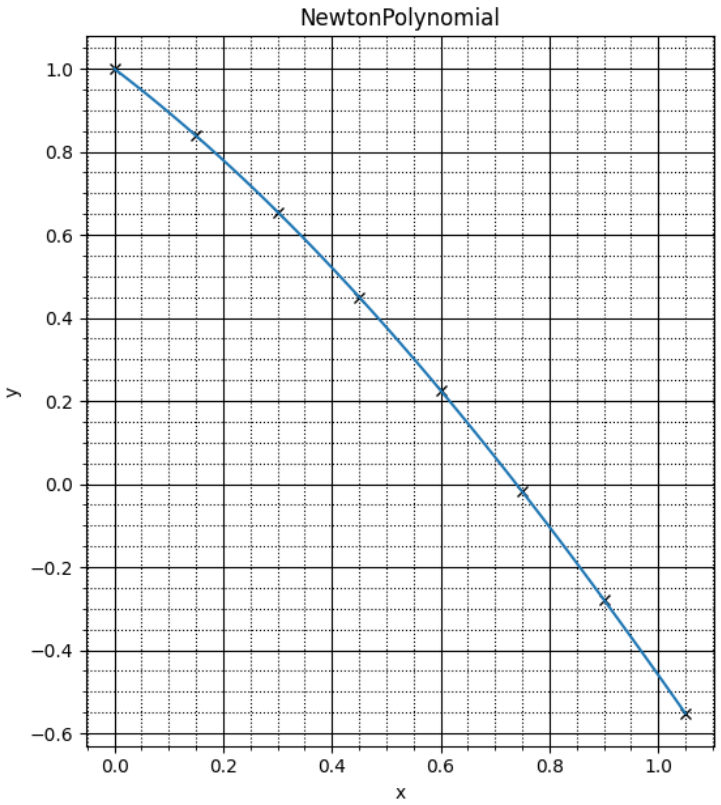

NewtonPolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x

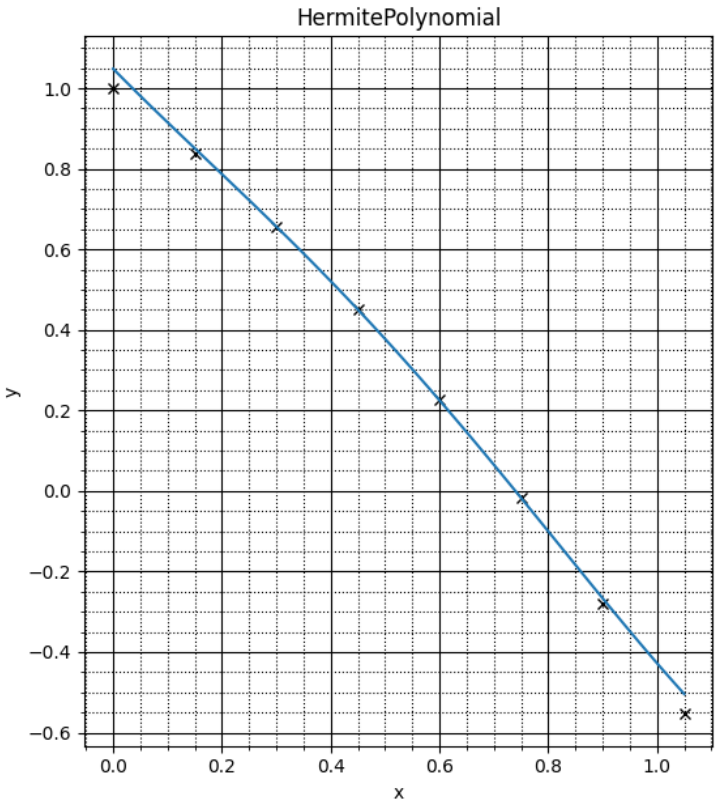

HermitePolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x

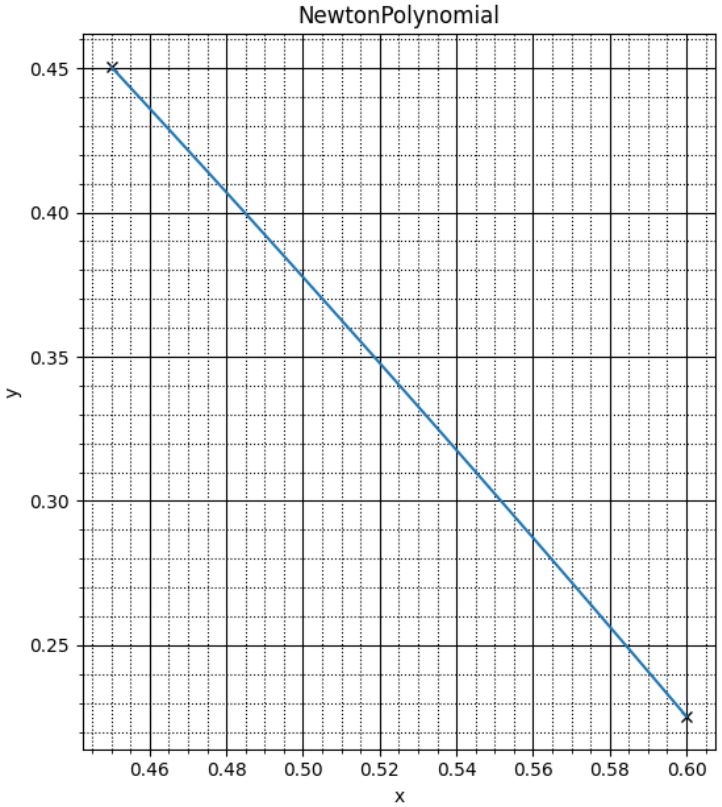

NewtonPolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

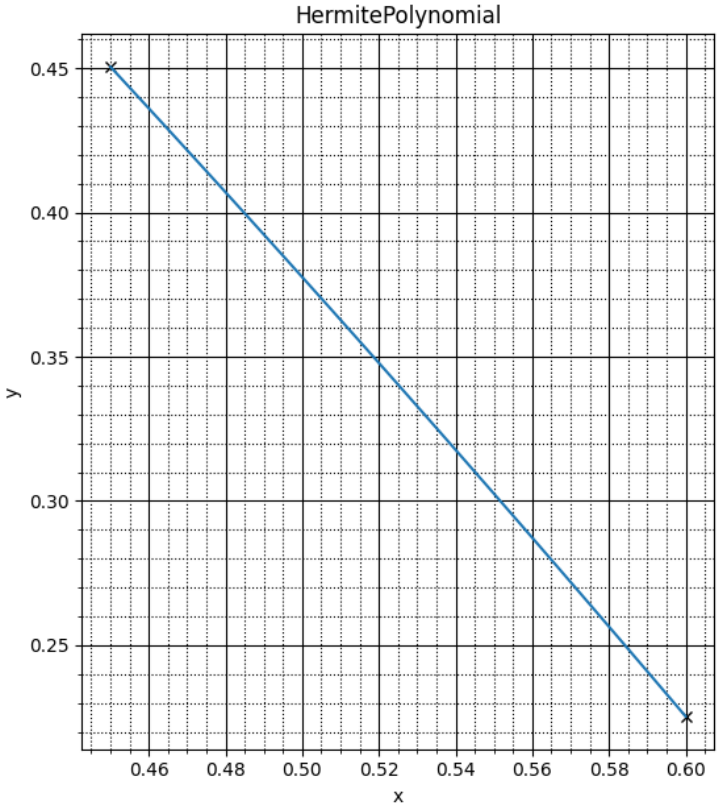

HermitePolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

- 4 степень


NewtonPolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x

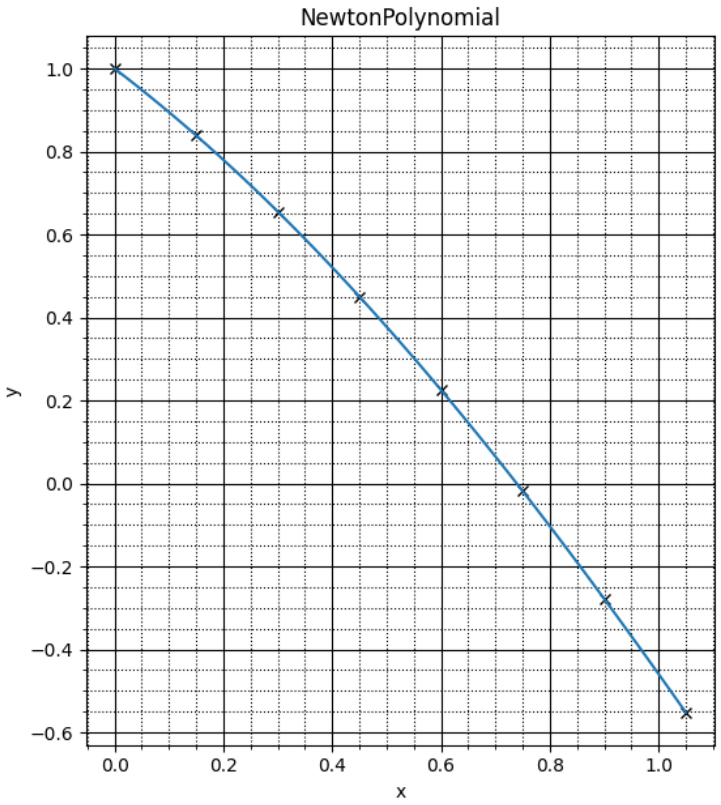

HermitePolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x

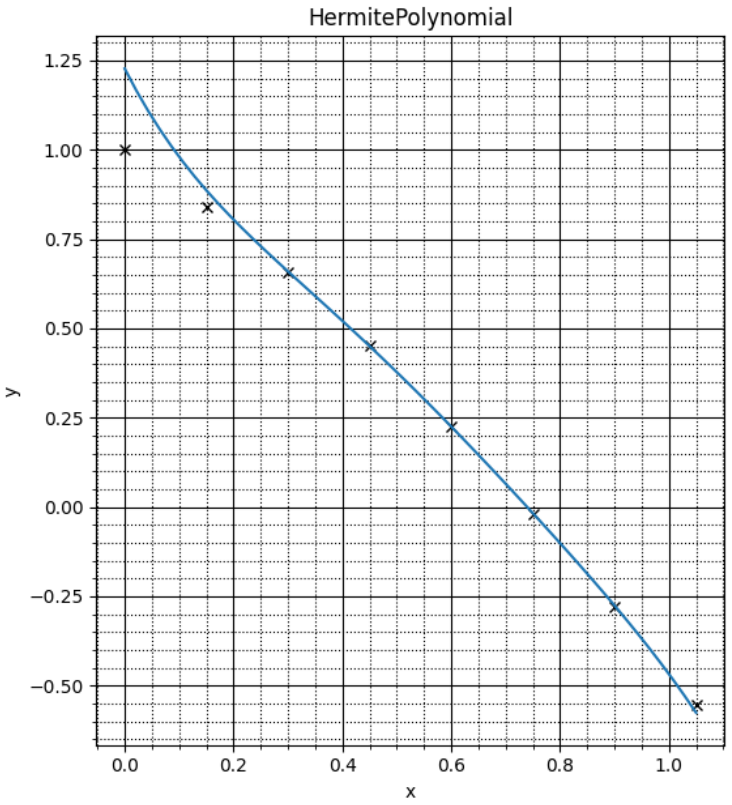

NewtonPolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

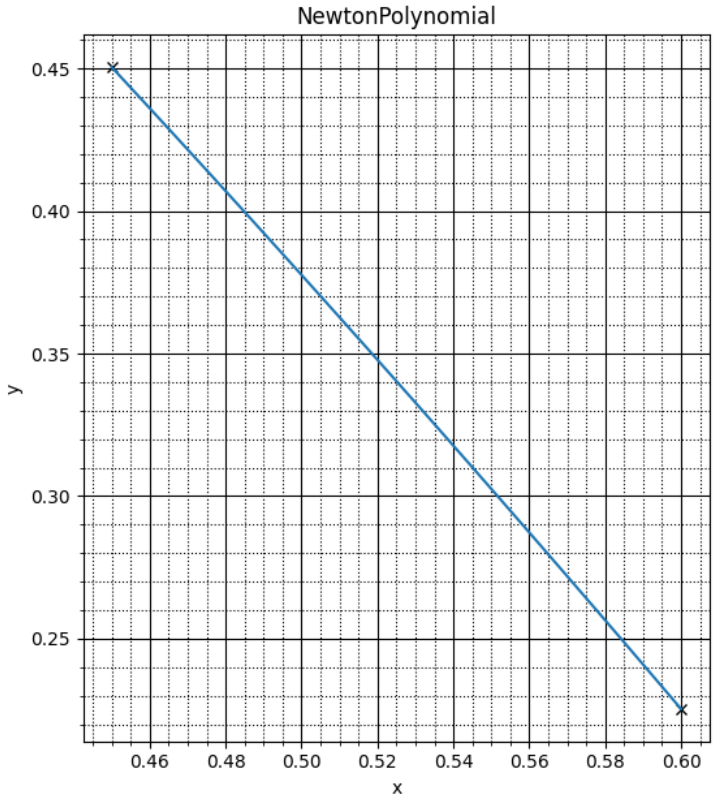

HermitePolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

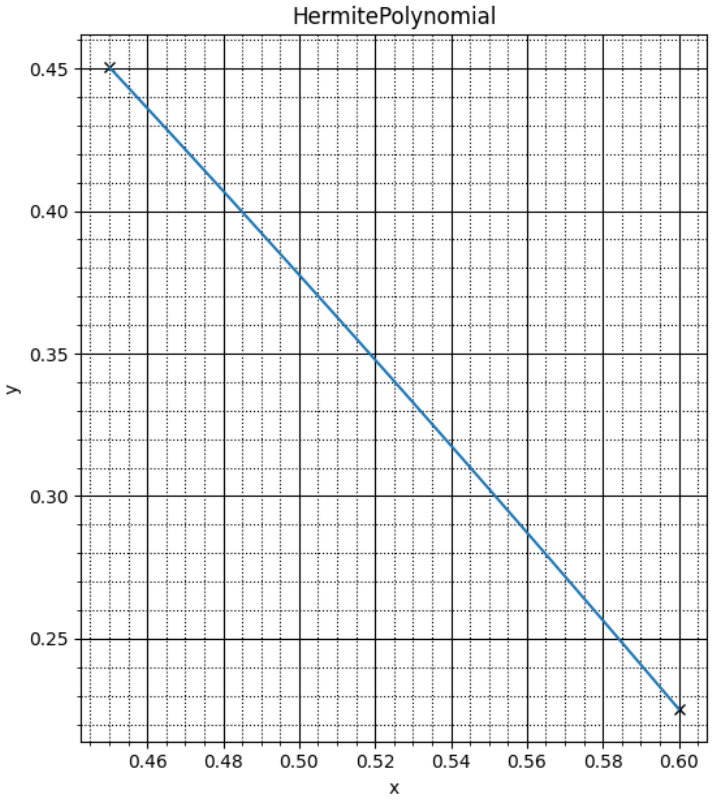
-

- 5 степень

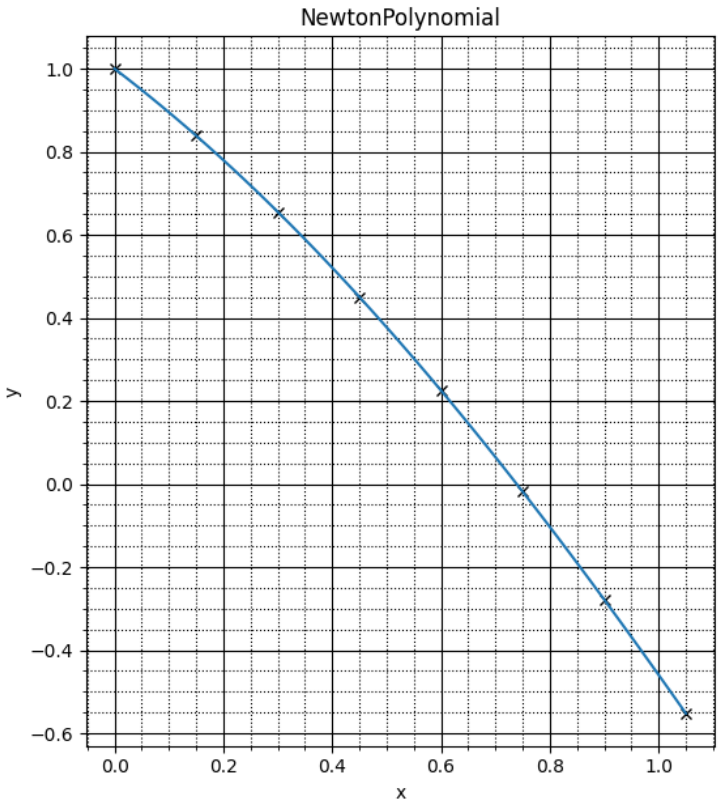

NewtonPolynomial
y
x
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0

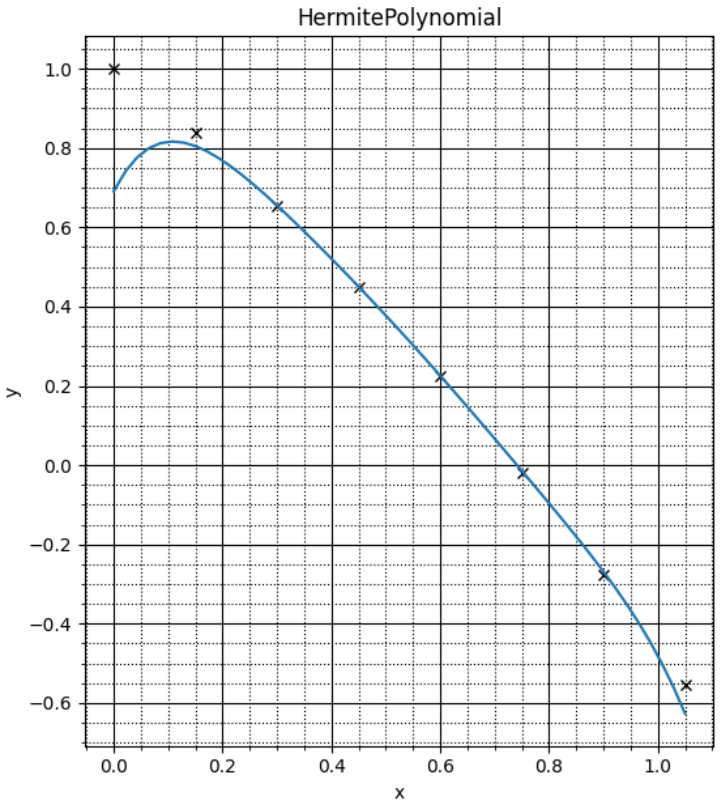

HermitePolynomial
y
x
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0.0
0.2
0.4
0.6
0.8
1.0

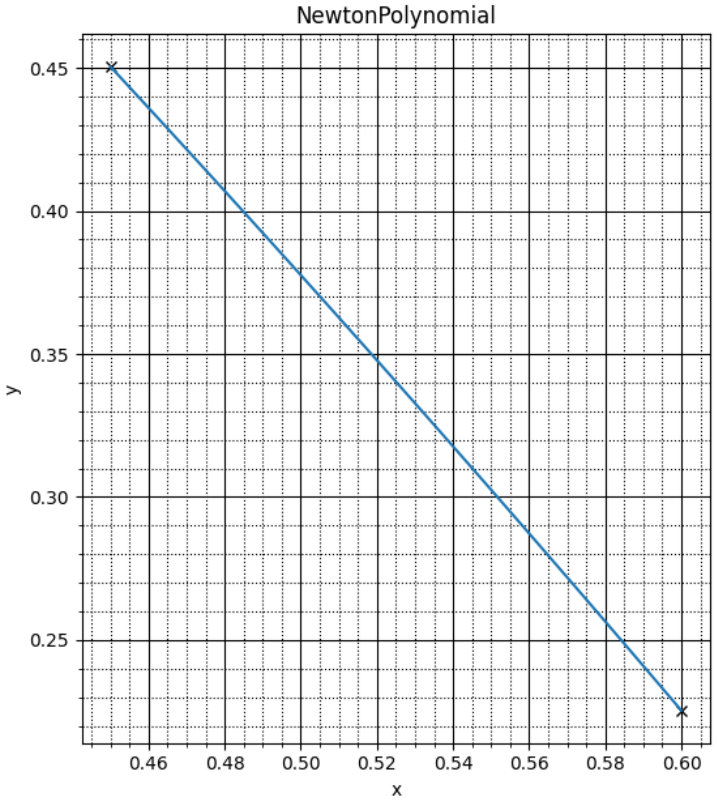

NewtonPolynomial
y
x
0.45
0.40
0.35
0.30
0.25
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60

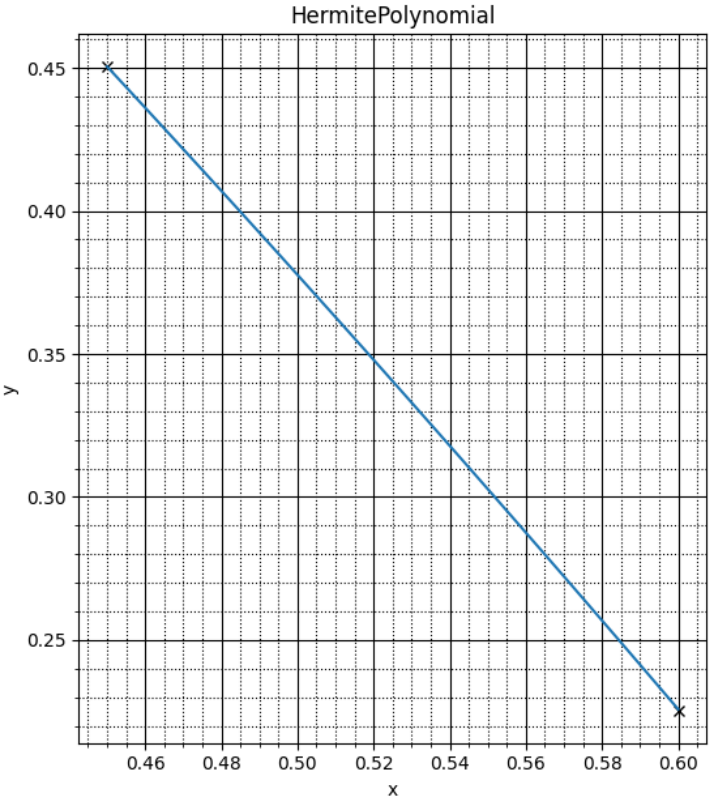

HermitePolynomial
y
x
0.45
0.40
0.35
0.30
0.25
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60

- 6 степень


NewtonPolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x


HermitePolynomial
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
y
0.0
0.2
0.4
0.6
0.8
1.0
x


NewtonPolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x


HermitePolynomial
0.45
0.40
0.35
0.30
0.25
y
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
x

Исходный код

Код программы представлен на листингах 1-3

Листинг 1. `utils.py`

```
def read_dots(fname: str) -> list[list[float]]:
    dots = []

    with open(fname) as fin:
        for line in fin.readlines():
            dots += [list(map(float, line.split()))]

    return sorted(dots, key=lambda p: p[1])


def print_dots(dots: list[list[float]) -> None:
    for d in dots:
        print(d)


def read_func_data() -> tuple[int, float]:
    deg, x = map(float, input().split())

    return int(deg), x
```

Листинг 2. `polynomial.py`

```
from __future__ import annotations
from math import prod


class Polynomial(object):
   terms: list[callable[float]:float]

   def __init__(self, terms: list[callable[float]:float]):
       self.terms = terms

   def __call__(self, arg: float) -> float:
       return sum([term(arg) for term in self.terms])


class NewtonPolynomial(Polynomial):

   @staticmethod
   def build(points: list[list[float]], arg: float, n: int) -> NewtonPolynomial:
       table = NewtonPolynomial._make_table(points, arg, n)

       return NewtonPolynomial(
           [lambda x:table[1][0]] +
           [NewtonPolynomial._term(table[i][0],  table[0][:i - 1])
            for i in range(2, len(table))]
       )

   @staticmethod
   def _term(va: float, vl: list[float]) -> callable:
       return lambda x: va * prod(map(lambda a: (x - a), vl))

   @staticmethod
   def _make_table(points: list[list[float]], arg: float, n: int) -> list[list[float]]:
       base = sorted(sorted(points, key=lambda p: abs(p[0] - arg))[:n+1])

       t = [[None for i in range(len(base))] for j in range(n + 2)]

       for i in range(len(t[0])):
           t[0][i], t[1][i] = base[i][0], base[i][1]

       for i in range(2, len(t)):
           for j in range(len(base) - i + 1):
               t[i][j] = (t[i - 1][j] - t[i - 1][j + 1]) / \
                   (t[0][j] - t[0][j + i - 1])

       return t


class HermitePolynomial(NewtonPolynomial):
```

```
    @ staticmethod
    def build(points: list[list[float]], arg: float, n: int) -> HermitePolynomial:
        table = HermitePolynomial._make_table(points, arg, n)

        return HermitePolynomial(
            [lambda x:table[1][0]] +
            [HermitePolynomial._term(table[i][0],  table[0][:i - 1])
             for i in range(2, len(table))]
        )

    @ staticmethod
    def _make_table(points: list[list[float]], arg: float, n: int) -> list[list[float]]:
        base = []

        for p in sorted(points, key=lambda p: abs(p[0] - arg)):
            base += [p[:j] for j in range(2, len(p) + 1)]

        base = sorted(base[:n + 1], key=lambda p: (p[0], -len(p)))

        t = [[None for i in range(len(base))] for j in range(n + 2)]

        for i in range(len(base)):
            p = base[i]
            for j in range(len(p)):
                t[j][i] = p[j]

        for i in range(2, len(t)):
            for j in range(len(base) - i + 1):
                if t[i][j]:
                    continue

                t[i][j] = (t[i - 1][j] - t[i - 1][j + 1]) / \
                          (t[0][j] - t[0][j + i - 1])

        return t
```

Листинг 3. `main.py`

```
from sys import argv

import matplotlib.pyplot as plt
import numpy as np

from utils import *
from polynomial import *

import pylab as py


def plot(dots: list[float], polynomials: list[Polynomial]) -> None:
    x, y = [p[0] for p in dots], [p[1] for p in dots]
    a, b = min(x), max(x)
    x_arg = np.linspace(a, b)

    plt.clf()

    for i in range(len(polynomials)):
        plt.subplot(1, len(polynomials), i + 1)
        plt.title(type(polynomials[i]).__name__)
        plt.xlabel("x")
        plt.ylabel("y")
        plt.grid(which='minor', color='k', linestyle=':')
        plt.grid(which='major', color='k')
        plt.minorticks_on()

        plt.plot(x, y, "xk")

        plt.plot(x_arg, [polynomials[i](arg) for arg in x_arg])

    plt.show()


if __name__ == "__main__":
    dots = read_dots(argv[1])

    print("Loaded table:")

    print_dots(dots)

    print("Enter polynomial degree and X value:")
    deg, x = read_func_data()

    newton = NewtonPolynomial.build(dots, x, deg)

    print("Newton value in {: <5.6g} is {: <5.6g}".format(x, newton(x)))

    hermite = HermitePolynomial.build(dots, x, deg)
```

```
print("Hermite value in {: <5.6g} is {: <5.6g}".format(x, hermite(x)))

inv_dots = [[p[1], p[0]] for p in dots]
root = NewtonPolynomial.build(inv_dots, 0.0, deg)(0.0)

print("Root found by inverted interpolation method is {: <5.6g}".format(root))

plot(dots, [newton, hermite])
```

Контрольные вопросы

1. Будет ли работать программа при степени полинома n=0?

   Будет.
   В качестве полинома будет использоваться некоторая константа (Свободный член)

2. Как практически оценить погрешность интерполяции? Почему сложно применить для этих целей теоретическую оценку?

   Через первый отброшеннй член полинома.
   Теоретическую провести тяжело, т.к. для нее нужна производная, которую, зачастую, невозможно установить

3. Если в двух точках заданы значения функции и ее первых производных, то полином какой минимальной степени может быть построен на этих точках?

   Минимальная - 0. В этом случае в качестве полинома будет использоваться константа (Свободный член).
   Максимальная - 3 (Т.к. на n степень нужно n+1 значений)

4. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?

   При построении таблицы раздельных разностей.
   Для нее необходимо отсортированть данные в порядке возрастания

5. Что такое выравнивающие переменные и как их применить для повышения точности интерполяции?

   Это такие переменные, в которых график функций близок к прямолинейному
   Часто используются к функциям, которые быстро меняются
   Чтобы повысить точность, строят полином Ньютона для этих переменных, а потом вместо них используют x и y