Министерство науки и высшего образования 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**

КАФЕДРА **Программное обеспечение ЭВМ и информационные технологии**

Лабораторная работа №1.

«Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций»

Студент: **Ивахненко Д. А**

Группа: **ИУ7-46Б**

Оценка (баллы): _____

Преподаватель: **Градов В.М.**

Москва

2021

Цель работы: получение навыков построения алгоритма интерполяции таблично заданных функций полиномами Ньютона и Эрмита.

Исходные данные:

1. Таблица функции и ее производных

x	y	y’
0.00	1.000000	-1.00000
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

2. Степень аппроксимирующего полинома – **n.**

3. Значение аргумента, для которого вычисляется интерполяция.

Описание алгоритма

Для решения задачи использую построение полиномов Ньютона и Эрмита. Для удобства практических вычислений вводится понятие разделенных разностей функции $y(x)$, заданной в узлах x_i

$$y(x_0, x_1) = \frac{y_0 - y_1}{x_0 - x_1}$$

$$y(x_0, x_1, x_2) = \frac{y(x_0, x_1) - y(x_1, x_2)}{x_0 - x_2}$$

$$y(x_0, x_1, x_2, x_3) = \frac{y(x_0, x_1, x_2) - y(x_1, x_2, x_3)}{x_0 - x_3}$$

И т.д.

Правило образования таких конструкций понятно из приведенной выше записи.

Тогда в результате получаем так называемый полином Ньютона, который имеет следующий вид:

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

Построение полинома Эрмита происходит схожим образов. Только в таблице исходных данных каждая точка дублируется, а для вычисления разделенной разности вида $y(x_i, x_i)$ используется значение производной функции, которое известно по условию.

В свою очередь для поиска приблизительного значения корня **монотонной функции** используется **обратная интерполяция**. Ее суть в том, что в таблице исходных данных меняются местами столбцы и выполняется обычная интерполяция при задании аргумента, равным 0.

Факт наличия корня у функции устанавливается по наличию смены знака функции.

Код программы

main.py

```
import numpy as np
import settings
from polynomes.hermite_polyn import hermite_polyn
from polynomes.newton_polyn import newton_polyn, search_root
from polynomes.utils import trim_table, read_table


def main():
    arg_value = float(input('Enter x: '))
    raw_table = read_table(settings.PATH_TO_TABLE)

    print(f'x = {arg_value}')
    print('|n|Newton |Hermite| Root  |')
    for polyn_degree in range(1, 5):
        newton_table = trim_table(raw_table, polyn_degree + 1, arg_value)[:, 0:2]
        hermite_table = trim_table(raw_table, (polyn_degree + 2) // 2, arg_value)

        ans_newton = newton_polyn(newton_table, arg_value)
        ans_hermite = hermite_polyn(hermite_table, arg_value, polyn_degree + 1)
        root = search_root(newton_table)

        print('|{}|{:.5f}|{:.5f}|{:.5f}|'.format(polyn_degree, ans_newton, ans_hermit
e, root))
```

```
if __name__ == '__main__':
    np.set_printoptions(precision=3)
    main()
```

Polynomes.hermite_polyn.py

```
import numpy as np

def _find_coef(x :np.ndarray, y :np.ndarray, dy :np.ndarray, n: int) -> np.ndarray:
        q = np.zeros((n, n - 1))
        q = np.concatenate((y[:n, None], q), axis=1)

        k = 0
        for i in range(1, n):
            for j in range(1, i + 1):
                if x[i] == x[i - j]:
                    q[i, j] = dy[k]
                    k += 1
                else:
                    q[i, j] = (q[i, j - 1] - q[i - 1, j - 1]) / (x[i] - x[i - j])
        return q.diagonal()

def hermite_polyn(table: np.ndarray, x: float, numof_params: int) -> np.float64:
    x_data, y_data = np.repeat(table[:, 0], 2), np.repeat(table[:, 1], 2)
    dy_data = table[:, 2]

    n = numof_params
    f = _find_coef(x_data, y_data, dy_data, n)
    p = f[0]

    for i in range(1, n):
        _tmp = 1
        for j in range(0, i):
            _tmp *= (x - x_data[j])
        p += _tmp * f[i]
    return p
```

polynomes.newton_polyn.py

```
import numpy as np

def _find_coef(x :np.ndarray, y :np.ndarray) -> np.ndarray:
        n = x.size
        q = np.zeros((n, n - 1))
        q = np.concatenate((y[:, None], q), axis=1)

        for i in range(1, n):
            for j in range(1, i + 1):
                q[i, j] = (q[i, j - 1] - q[i - 1, j - 1]) / (x[i] - x[i - j])
```

```
        return q.diagonal(), n

def newton_polyn(table: np.ndarray, x: float) -> np.float64:
    x_data, y_data = table[:, 0], table[:, 1]
    f, n = _find_coef(x_data, y_data)

    p = f[0]
    for i in range(1, n):
        _tmp = 1
        for j in range(0, i):
            _tmp *= (x - x_data[j])
        p += _tmp * f[i]
    return p


def search_root(raw_table: np.ndarray) -> np.float64:
    x = raw_table[:, 0]
    y = raw_table[:, 1]
    table = np.concatenate((y[:, None], x[:, None]), axis=1)
    return newton_polyn(table, 0.0)
```

polynomes.utils.py

```
import numpy as np

def read_table(path :str) ->np.ndarray:
    data = np.loadtxt(path)
    data = data[data.argsort(axis=0)[:, 0]]
    return data

def print_polyn(f, x_data, n):
    print("The polynomial is:")
    print("P(x)={:+.3f}".format(f[0]), end="")
    for i in range(1, n):
        print("{:+.3f}".format(f[i]), end="")
        for j in range(0, i):
            print("(x{:+.3f})".format(x_data[j] * -1), end="")
    print("")

def trim_table(table: np.ndarray, numof_rows: int, x: float) ->np.ndarray:
    args = table[:, 0]
    length = len(args)

    pos = 0
    while (pos < length and x >= args[pos]):
        pos += 1
```

```
    start = pos - numof_rows // 2
    end = pos + numof_rows // 2

    if (start >= 0) and (end <= length):
        if (numof_rows % 2):
            if (end == length) or (abs(x - args[start - 1]) < abs(x - args[end]))
:
                start -= 1
            else:
                end += 1
        new_table = table[start:end]
    elif start < 0:
        new_table = table[:numof_rows]
    elif end > length:
        new_table = table[length - numof_rows:]

    return new_table
```

Результаты работы

1. Значения функции при степенях полиномов Ньютона и Эрмита при n=1, 2, 3, 4 при фиксированном x = 0.525. Результаты свести в таблицу.

Степень (n)	Полином Ньютона	Полином Эрмита
1	0.33789	0.34268
2	0.34021	0.34036
3	0.34031	0.73905
4	0.34032	0.73908

2. Найти корень заданной табличной функции с помощью обратной интерполяции, используя полином Ньютона.

$$y(x) = 0 \text{ при } x \approx 0.73873$$

Ответы на контрольные вопросы

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

Да, программа будет работать. При вычислении значения функции в некоторой точке выведется значение функции в ближайшей точке.

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

Точность расчетов удобно оценивать, наблюдая за тем, насколько быстро убывают члены ряда, составляющего полином. Если это происходит достаточно быстро, можно оставлять только те члены, которые больше заданной погрешности расчётов.

Погрешность многочлена Ньютона можно оценить по формуле

$$\left|y(x)-P_n(x)\right| \le \frac{M_{n+1}}{(n+1)!}\left|\varpi_n(x)\right|,$$

где $M_{n+1}=max\left|y^{(n+1)}(\xi)\right|$ - максимальное значение производной интерполируемой функции на отрезке между наименьшим и наибольшим из значений $x_0, x_1, x_2, \ldots, x_n$, а полином

$$\varpi_n(x)=\prod_{i=0}^{n}(x-x_i).$$

Трудность использования указанных теоретических оценок на практике состоит в том, что производные интерполируемой функции обычно неизвестны, поэтому для определения погрешности удобнее воспользоваться оценкой первого отброшенного члена

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

При данных условиях имеем 4 параметра, а значит, можно построить полином степени 3.

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

Для повышения эффективности программы и экономии времени при поиске точек для интерполяции и построения полинома удобно, когда таблица содержит отсортированные данные

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

Выравнивающие переменные – это производные функции от исходных переменных. Используя данные переменные можно получить график, который на отдельных участках будет близок к прямой, что упрощает работу с **быстроизменяющимися функциями.** Выравнивающие преобразования подбирают несложными, чтобы их производные находились точно.