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

Тема Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций

Студент Климов И.С.

Группа ИУ7-42Б

Оценка (баллы) _____

Преподаватель Градов В.М.

Москва.
2021 г

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

1. Исходные данные

1) Таблица функции и ее производных

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

2) Степень аппроксимирующего полинома – n

3) Значение аргумента, для которого выполняется интерполяция

2. Код программы

Листинг 1. data.py

```
y0 = {
    0.00: 1.000000,
    0.15: 0.838771,
    0.30: 0.655336,
    0.45: 0.450447,
    0.60: 0.225336,
    0.75: -0.018310,
    0.90: -0.278390,
    1.05: -0.552430
}

y_reverse = {
    1.000000: 0.00,
    0.838771: 0.15,
    0.655336: 0.30,
    0.450447: 0.45,
    0.225336: 0.60,
    -0.018310: 0.75,
    -0.278390: 0.90,
    -0.552430: 1.05
}

y_diff = {
    0.00: -1.000000,
    0.15: -1.14944,
```

```
    0.30: -1.29552,
    0.45: -1.43497,
    0.60: -1.56464,
    0.75: -1.68164,
    0.90: -1.78333,
    1.05: -1.86742
}
```

Листинг 2. func.py

```
from data import y0, y_diff


def get_diff(y, *args):
    if len(args) == 0:
        return None
    elif len(args) == 1:
        return y[args[0]]
    elif len(args) == 2 and len(args) != len(set(args)):
        return y_diff[args[0]]
    else:
        return (get_diff(y, *args[:-1]) - get_diff(y, *args[1:])) / (args[0] -
args[-1])


def get_xi_newton(n, x, start, finish, step):
    if n < 0:
        raise ValueError('n can\'t be negative')

    if n > len(y0):
        raise ValueError(f'n can\'t be more than {len(y0)}')

    array_x = []
    first_element = -1
    for value, i in y0.items():
        if value <= x:
            first_element = value
    array_x.append(first_element)
    inc = 1
    dec = 1

    for _ in range(1, n // 2 + 1):
        value_up = round(first_element + inc * step, 2)
        value_down = round(first_element - dec * step, 2)
        if value_up <= finish:
            array_x.append(value_up)
            inc += 1
        else:
            array_x.append(value_down)
            dec += 1

        value_down = round(first_element - dec * step, 2)
        value_up = round(first_element + inc * step, 2)
        if value_down >= start:
            array_x.append(value_down)
            dec += 1
        else:
            array_x.append(value_up)
            inc += 1

    if n % 2 != 0:
        value_up = round(first_element + inc * step, 2)
```

```
        value_down = round(first_element - dec * step, 2)
        if value_up <= finish:
            array_x.append(value_up)
        else:
            array_x.append(value_down)

    return sorted(array_x)


def get_xi_hermit(n, x, start, finish, step):
    if n < 0:
        raise ValueError('n can\'t be negative')

    if n > len(y0):
        raise ValueError(f'n can\'t be more than {len(y0)}')

    array_x = []
    xi_newton = get_xi_newton(n // 2, x, start, finish, step)
    for x in xi_newton:
        array_x.append(x)
        array_x.append(x)
    array_x = array_x[:(n + 1)]
    return array_x


def get_xi_reverse(n):
    if n < 0:
        raise ValueError('n can\'t be negative')

    if n > len(y0):
        raise ValueError(f'n can\'t be more than {len(y0)}')

    values = list(y0.values())
    first_index = 4
    first_element = values[first_index]
    array_x = [first_element]
    array_x.extend(values[(first_index - n // 2):first_index])
    array_x.extend(values[(first_index + 1):(first_index + n // 2) + 1])
    if n % 2 != 0:
        array_x.append(values[first_index + n // 2 + 1])

    return sorted(array_x)
```

Листинг 3. main.py

```
from func import get_diff, get_xi_newton, get_xi_hermit, get_xi_reverse
from data import y0, y_reverse


def get_polynomial(xi, y):
    polynomial = []

    for i in range(len(xi)):
        coefficients = xi[:i]
        polynomial.append(coefficients)
        diff = get_diff(y, *xi[:(i + 1)])
        polynomial.append(diff)
    return polynomial


def take_x(brackets, x):
    if not brackets:
```

```
            return 1
        result = 1
        for bracket in brackets:
            result *= (x - bracket)
        return result


def count_value(polynomial, x):
        result = 0
        for i in range(0, len(polynomial), 2):
            result += take_x(polynomial[i], x) * polynomial[i + 1]
        return result


def main():
        start = 0.00
        finish = 1.05
        step = 0.15
        try:
            x = float(input('Введите x: '))
        except ValueError:
            return print('Вы должны ввести число')

        print('\n|', '-' * 7, '|', '-' * 14, '|', '-' * 14, '|', '-' * 14, '|', sep='')
        print('|Степень| Для полинома | Для полинома | Для обратной |')
        print('|        |    Ньютона   |     Эрмита   | интерполяции |')
        for n in range(1, 5):
            print('|', '-' * 7, '|', '-' * 14, '|', '-' * 14, '|', '-' * 14, '|',
sep='')

            xi_newton = get_xi_newton(n, x, start, finish, step)
            polynomial_newton = get_polynomial(xi_newton, y0)
            print(f'|    {n}    |    {count_value(polynomial_newton, x):.6f}', end='   ')

            xi_hermit = get_xi_hermit(n, x, start, finish, step)
            polynomial_hermit = get_polynomial(xi_hermit, y0)
            print(f'|    {count_value(polynomial_hermit, x):.6f}', end='   ')

            xi_reverse = get_xi_reverse(n)
            polynomial_reverse = get_polynomial(xi_reverse, y_reverse)
            print(f'|    {count_value(polynomial_reverse, 0):.6f}   |')

        print('|', '-' * 7, '|', '-' * 14, '|', '-' * 14, '|', '-' * 14, '|', sep='')


if __name__ == '__main__':
        main()
```

3. Результаты работы

1) Значения y(x) при степенях полинома Ньютона и Эрмита (второй и третий столбцы) при n = 1, 2, 3, 4 при x = 0.525
2) Корень функции, найденный с помощью обратной интерполяции, используя полином Ньютона (четвертый столбец) при аргументе, равном 0

```
Введите x: 0.525

|-------|--------------|--------------|--------------|
|Степень| Для полинома | Для полинома | Для обратной |
|       |    Ньютона   |    Эрмита    | интерполяции |
|-------|--------------|--------------|--------------|
|   1   |   0.337891   |   0.342824   |   0.738727   |
|-------|--------------|--------------|--------------|
|   2   |   0.340419   |   0.340358   |   0.739174   |
|-------|--------------|--------------|--------------|
|   3   |   0.340314   |   0.340323   |   0.739095   |
|-------|--------------|--------------|--------------|
|   4   |   0.340324   |   0.340323   |   0.739081   |
|-------|--------------|--------------|--------------|
```

4. Вопросы при защите лабораторной работы

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

Программа выдаст результат. Мы получим значения, которые будут ближайшими в таблице. То есть будут выведены значения от ближайшего к аргументу x, и ответ получится совсем не точным

```
Введите x: 0.525

|-------|--------------|--------------|--------------|
|Степень| Для полинома | Для полинома | Для обратной |
|       |    Ньютона   |    Эрмита    | интерполяции |
|-------|--------------|--------------|--------------|
|   0   |   0.450447   |   0.450447   |   0.600000   |
|-------|--------------|--------------|--------------|
```

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

Практически оценить погрешность интерполяции можно оценивать, наблюдая за тем, насколько быстро убывают члены ряда, то есть путем оценки первого отброшенного члена. Теоретически же погрешность многочлена Ньютона можно оценить по формуле:

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

Однако есть определенная сложность в связи с тем, что зачастую производные интерполируемой функции неизвестны (в формуле M_{n+1} – максимальное значение производной интерполируемой функции на отрезке между наибольшим и наименьшем из значений).

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

На этих точках минимально может быть построен полином третьей степени (с четырьмя коэффициентами).

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

Информация об упорядоченности аргумента функции существенна при формировании конфигурации, так как значения необходимо брать по возможности симметрично, для чего нужно знать между какими значениями в исходной таблице находится значение x.

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

Выравнивающие переменные – это переменные, преобразованием которых можно добиться того, что график будет близко к прямой хотя бы на некоторых участках. Тогда интерполяцию проводят в новых переменных, а затем обратным интерполированием находят нужные значения функции.