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 Алахов А.Г.

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 её 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.

   С учётом степени полинома выбирается n + 1 точек из таблицы.

3. Значение аргумента, для которого выполняется интерполяция.

   Исходя из значения аргумента выбираются ближайшие n + 1 точек из таблицы, по которым строится полином.

2 Код программы

Код программы представлен на листингах 1-2.

Листинг 1. functions.py

```
from math import fabs


def copy_table(table):
    new = []
    for i in table:
        new.append(i[:])
```

```
    return new


def neuton(table, x, n):
    n += 1
    func_table = copy_table(table)

    i = 0
    while func_table[i][0] < x and i < len(func_table):
        i += 1

    left_part = right_part = n // 2
    if n % 2:
        if (x - func_table[i - 1][0]) -
(func_table[i][0] - x) <= 0.000001:
            left_part += 1
        else:
            right_part += 1

    if i + right_part > len(func_table):
        left_part += i + right_part - len(func_table)

    start = max(i - left_part, 0)

    for j in range(1, n):
        for i in range(start+n-1, start+j-1, -1):
            func_table[i][1] = (func_table[i][1] -
func_table[i - 1][1]) / (func_table[i][0] - func_table[i
- j][0])

    result = 0
```

```
    for i in range(start, start + n):
        mult = func_table[i][1]
        for j in range(start, i):
            mult *= (x - func_table[j][0])
        result += mult

    return result


def hermit_table(table):
    new = []
    for i in table:
        new.append(i[:])
        new.append(i[:])

    return new


def hermit(table, x, n):
    n += 1
    func_table = hermit_table(table)

    i = 0
    while func_table[i][0] < x and i < len(func_table):
        i += 1

    left_part = right_part = n // 2
    if n % 2:
        if (x - func_table[i - 1][0]) -
(func_table[i][0] - x) <= 0.000001:
            left_part += 1
```

```
        else:
            right_part += 1

    if i + right_part > len(func_table):
        left_part += i + right_part - len(func_table)

    start = max(i - left_part, 0)

    for i in range(start + n - 1, start, -1):
        if fabs(func_table[i][0] - func_table[i - 1][0])
< 0.000001:
            func_table[i][1] = func_table[i][2]
        else:
            func_table[i][1] = ((func_table[i][1] -
func_table[i - 1][1]) / (func_table[i][0] - func_table[i
- 1][0]))

    for j in range(2, n):
        for i in range(start+n-1, start+j-1, -1):
            func_table[i][1] = ((func_table[i][1] -
func_table[i - 1][1]) / (func_table[i][0] - func_table[i
- j][0]))

    result = 0
    for i in range(start, start + n):
        mult = func_table[i][1]
        for j in range(start, i):
            mult *= (x - func_table[j][0])
        result += mult

    return result
```

```
def selection_sort(table):
    for i in range(len(table) - 1):
        smallest = i
        for j in range(i + 1, len(table)):
            if table[j][0] < table[smallest][0]:
                smallest = j
        table[i], table[smallest] = table[smallest],
table[i]



def find_root(table, n):
    func_table = copy_table(table)
    for i in func_table:
        temp = i[0]
        i[0] = i[1]
        i[1] = temp

    selection_sort(func_table)

    return neuton(func_table, 0, n)
```

Листинг 2. main.py

```
from valg_1 import *

def main():
    func_table = [[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]]

    x = 0.525

    print('Заданная таблица:\n\
|  X  |   Y(x)   |    Y\'    |')
    for i in func_table:
        print('|{:5.2f}|{:10.6f}|{:10.6f}|'.format(i[0], i[1], i[2]))

    print('\n\nЗначения полиномов при X = 0.525:\n\
| Полином |   n = 1   |   n = 2   |   n = 3   |   n = 4   |')

    print('| Ньютона |', end = '')
    for i in range(1, 5):
        print('{:10.6f} |'.format(neuton(func_table, x, i)), end = '')

    print('\n| Эрмита  |', end = '')
    for i in range(1, 5):
        print('{:10.6f} |'.format(hermit(func_table, x, i)), end = '')

    print('\n\nНахождение корня функции обратной интерполяцией, используя полином Ньютона:\n\
|   n = 1   |   n = 2   |   n = 3   |   n = 4   |\n|', end = '')
    for i in range(1, 5):
```

```
            print('{:10.6f} |'.format(find_root(func_table,
i)), end = '')


    n = int(input('\n\nВведите степень полинома: '))
    x = float(input('Введите значение X: '))


    print('\nЗначение, полученное полиномом Ньютона:
{:.6f}\n\
Значение, полученное полиномом Эрмита:
{:.6f}'.format(neuton(func_table,x,n),

hermit(func_table, x, n)))


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

3 Результаты работы

1. Значения y(x) при степенях полиномов Ньютона и Эрмита n= 1, 2, 3 и 4 при фиксированном x = 0.525

```
Значения полиномов при X = 0.525:
| Полином |   n = 1   |   n = 2   |   n = 3   |   n = 4   |
| Ньютона |  0.337891 |  0.340419 |  0.340314 |  0.340324 |
| Эрмита  |  0.337891 |  0.340358 |  0.340323 |  0.340324 |
```

2. Корень заданной выше табличной функции с помощью обратной интерполяции, используя полином Ньютона при различных степенях полинома.

```
Нахождение корня функции обратной интерполяцией, используя полином Ньютона:
|   n = 1   |   n = 2   |   n = 3   |   n = 4   |
|  0.738727 |  0.739046 |  0.739095 |  0.739088 |
```

4 Вопросы при защите лабораторной работы

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

Да, результатом работы будет значение y(x) самого близкого к введённому аргументу узла.

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. Если это происходит достаточно быстро, можно оставить только члены, начиная с определенного, значение которого и будет являться погрешностью.

Для применения теоретической оценки требуется знать производные интерполируемой функции, которые обычно не известны.

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

Так как заданы не только значения ф-и, но и её первых производных, то можно построить полином Эрмита. Так как доступно 4 узла, то степень полинома равна 3.

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

Эта информация существенна при нахождении узлов в таблице, ближайших к заданному аргументу. Если значения аргумента ф-и будут упорядочены, то не придётся тратить время либо на просмотр всей таблицы в поисках значений, либо на сортировку таблицы. Также смена знака у аргумента при обратной интерполяции говорит о наличии корня.

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

Выравнивающие переменные – это переменные элементарной функции, приближенной к начальной таблице. Для применения выравнивающих переменных, начальную функцию на протяжении нескольких шагов таблицы приближают к некой элементарной. Затем строится таблица выравнивающих переменных, их интерполяция и нахождение результата обратным преобразованием.