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 Бугаенко А.П.

Группа ИУ7-35Б

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
3. Значение аргумента, для которого выполняется интерполяция.

2 Код программы

Код программы представлен на листинге ниже.

Листинг main.py

```
import math as m
import pandas as pd

table = [
    {"x": 0.00, "y": 1.000000, "y'": -1.000000},
    {"x": 0.15, "y": 0.838771, "y'": -1.14944},
    {"x": 0.30, "y": 0.655336, "y'": -1.29552},
    {"x": 0.45, "y": 0.450447, "y'": -1.43497},
    {"x": 0.60, "y": 0.225336, "y'": -1.56464},
    {"x": 0.75, "y": -0.018310, "y'": -1.68164},
    {"x": 0.90, "y": -0.278390, "y'": -1.78333},
    {"x": 1.05, "y": -0.552430, "y'": -1.86742}]

def Divided_diff(x, y):
    new_y = []
    n = len(x) - len(y)
    for i in range(0, len(y) - 1):
        new_y.append((y[i] - y[i + 1]) / (x[i] - x[i + n + 1]))
    return new_y

def SortTableNewton(table, x, n):
    table = sorted(table, key=lambda d: abs(d["x"] - x))
    table = sorted(table[:n+1], key=lambda t: t["x"])
    return table
```

```
def FormXYNewton(table):
    X = []
    Y = []
    for row in table:
        X.append(row["x"])
        Y.append(row["y"])
    return X, Y

def CountDiffDivNewton(X, Y):
    Y_arr = [Y]
    while len(Y_arr[-1]) != 1:
        Y = Divided_diff(X, Y)
        Y_arr.append(Y)
    return Y_arr

def CountPolynomNewton(Y_arr):
    polym = []
    for Y in Y_arr:
        polym.append(Y[0])
    return polym

def GetValApproxNewton(polym, X, x):
    x_mult = 1
    result = 0
    for i in range(0, len(polym)):
        result += x_mult * polym[i]
        x_mult = x_mult * (x - X[i])
    return result

def NewtonApprox(x, n, table):

    table = SortTableNewton(table, x, n)
    X, Y = FormXYNewton(table)

    Y_arr = CountDiffDivNewton(X, Y)
    polym = CountPolynomNewton(Y_arr)

    return GetValApproxNewton(polym, X, x)

def SortTableErmit(table, x, n):
    table = sorted(table, key=lambda d: abs(d["x"] - x))
    table = sorted(table[:m.floor(n/2) + 1], key=lambda t:
t["x"])
    return table

def FormXYY_Ermit(table):
    X = []
    Y = []
    Y_ = []
    for row in table:
        X.append(row["x"])
        Y.append(row["y"])
        Y_.append(row["y'"])
    return X, Y, Y_

def Divided_diff_ermit(x, y, n):
    new_y = []
    n = n + 1 - len(y)
    for i in range(0, len(y) - 1):
        new_y.append((y[i] - y[i + 1]) / (x[int(i/2)] -
x[int((i + n + 1)/2)]))
    return new_y
```

```
def FormY_arr(n, X, Y, Y_):
    Y_arr = [Y]
    new_Y = []
    j = 0
    k = 0
    Y = Divided_diff(X, Y)
    for i in range(0, n):
        if i % 2 == 0:
            new_Y.append(Y_[j])
            j += 1
        else:
            new_Y.append(Y[k])
            k += 1
    if len(new_Y) != 0:
        Y_arr.append(new_Y)
    return Y_arr

def CountDiffDivErmit(X, Y, Y_, n):
    Y_arr = FormY_arr(n, X, Y, Y_)
    if len(Y_arr) != 1:
        Y = Y_arr[-1]
    while len(Y_arr[-1]) != 1:
        Y = Divided_diff_ermit(X, Y, n)
        Y_arr.append(Y)
    return Y_arr

def CountPolynomErmit(Y_arr):
    polym = []
    for Y in Y_arr:
        polym.append(Y[0])
    return polym

def GetValApproxErmit(polym, X, x):
    x_mult = 1
    result = 0
    for i in range(0, len(polym)):
        result += x_mult * polym[i]
        x_mult = x_mult * (x - X[int(i/2)])
    return result

def ErmitApprox(x, n, table):
    table = SortTableErmit(table, x, n)
    X, Y, Y_ = FormXYY_Ermit(table)

    Y_arr = CountDiffDivErmit(X, Y, Y_, n)
    polym = CountPolynomErmit(Y_arr)

    return GetValApproxErmit(polym, X, x)

def SortTableNewtonRoot(table, y, n):
    table = sorted(table, key=lambda d: abs(d["y"] - y))
    table = sorted(table[:n+1], key=lambda t: t["y"])
    return table

def NewtonRootApprox(n, table):
    y = 0
    table = SortTableNewtonRoot(table, y, n)
    Y, X = FormXYNewton(table)

    Y_arr = CountDiffDivNewton(X, Y)
    polym = CountPolynomNewton(Y_arr)

    return GetValApproxNewton(polym, X, y)
```

```
def FormResultApproxTable(x, table, out_name):
    results = {"n": [], "y for Newton": [], "y for
Ermit": []}
    for n in range(0, 5):
        results["n"].append(n)
        results["y for
Newton"].append(round(NewtonApprox(x, n, table), 5))
        results["y for
Ermit"].append(round(ErmitApprox(x, n, table), 5))
    df = pd.DataFrame(results)
    df.to_csv(out_name + ".csv")

def FormRootApproxTable(table, out_name):
    results = {"n": [], "x for Newton": []}
    for n in range(0, 5):
        results["n"].append(n)
        results["x for
Newton"].append(round(NewtonRootApprox(n, table), 5))
    df = pd.DataFrame(results)
    df.to_csv(out_name + ".csv")

FormResultApproxTable(0.525, table, "out_1")
FormResultApproxTable(0.55, table, "out_2")
FormResultApproxTable(0.575, table, "out_3")
FormRootApproxTable(table, "out_root")
```

3 Результаты работы

1. Значения y(x) при степенях полиномов Ньютона и Эрмита n= 1, 2, 3 и 4 при фиксированном x, например, x=0.525 (середина интервала 0.45- 0.60). Результаты свести в таблицу для сравнения полиномов.

x = 0.525		
n	y for Newton	y for Ermit
0	0.22534	0.22534
1	0.33789	0.34268
2	0.34021	0.34036
3	0.34031	0.34032
4	0.34032	0.34032

x = 0.55		
n	y for Newton	y for Ermit
0	0.22534	0.22534
1	0.30037	0.30357
2	0.30243	0.30257
3	0.30252	0.30252
4	0.30252	0.30252

x = 0.575		
n	y for Newton	y for Ermit
0	0.22534	0.22534
1	0.26285	0.26445
2	0.26414	0.26422
3	0.26419	0.26419
4	0.26419	0.26419

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

n	x for Newton
0	0.75
1	0.73873
2	0.73905
3	0.73909
4	0.73909

4 Вопросы при защите лабораторной работы

1. Будет ли работать программа при степени полинома n=0?
Да, будет, поскольку если мы выбираем степень полинома равную 0, то он будет состоять только из одного свободного члена. Который в свою очередь является y_0, т.е. первым значением y из отсортированной таблицы значений. Это хорошо видно в нашем случае, поскольку при попытке рассчитать значение со степенью создаваемого полинома 0 мы получаем значение 0.22534, и если мы посмотрим таблицу, то мы увидим, что это значение y, которое соответствует x, такому, который находится ближе всего к x, для которого нам нужно рассчитать значение, в нашем случае это были x = 0.525, 0.55 и 0.575. При обратной интерполяции происходит то же самое, но уже со значением y функции, которое при n = 0 принимает значение табличного x, для которого табличный y находится ближе всего к значению 0.

2. Как практически оценить погрешность интерполяции? Почему сложно применить для этих целей теоретическую оценку?
Практически погрешность вычисляется с помощью построения полинома со степенью, на 1 больше заданной (n + 1), и тогда погрешность будет $R_n(x) \approx |L_{n+1}(x) - L_n(x)|$, где $R_n(x)$ - погрешность, а $L_{n+1}(x)$ и $L_n(x)$ - полиномы n и n+1 степеней. Данная оценка погрешности является менее точной, чем оценка с помощью максимального значения производной, однако последняя (как и остальные оценки, зависящие от производных) редко применимы на практике ввиду необходимости вычисления производных интерпретируемой функции вплоть до n+1 порядка для их реализации, что в реальных случаях мало возможно.

3. Если в двух точках заданы значения функции и ее первых производных, то полином какой минимальной степени может быть построен на этих точках?
Минимальная степень полинома, который может быть построен на двух точках равняется 0, если же рассматривать максимальную степень, которой можно достичь, то в случае полинома Ньютона мы получаем одну разделённую разность, и как следствие, полином будет второй степени.
Максимальное значение степени достигается при использовании полинома Эрмита, поскольку мы можем использовать производные, так как у нас только две точки, то из них мы можем извлечь только одну разделённую разность, и использовав две производные вычислить коэффициенты для полинома Эрмита третьей степени.

4. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?
Информация об упорядоченности аргумента функции существенна при оценке точности расчётов. Наблюдение за тем, насколько быстро возрастают или убывают члены ряда позволяет, в случае если изменения происходят достаточно быстро, оставлять только те члены ряда, которые больше заданной погрешности расчётов.

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

Выравнивающие переменные это переменные, которые используются в методе выравнивания, который используется для более точной интерполяции быстроменяющихся функций. С помощью преобразований переменных $n = n(y)$ и $e = e(x)$ можно добиться чтобы в новых переменных график $n(e)$ был близок к прямой хотя бы на некоторых участках. Если переменные получается подобрать, то интерполяцию проводят в переменных (n, e), а затем находят $y_i = y(n_i)$ с помощью обратного интерполирования.