	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
«Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций»

Студент группы ИУ7-41Б
Костев Дмитрий

Москва 2021

Цель работы

Получение навыков построения алгоритма интерполяции таблично заданных функций полиномами Ньютона и Эрмита.

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.

Код программы

```
# Лабораторная работа №1

class Point:
    def __init__(self, x, y, derivative):
        self.x = x
        self.y = y
        self.derivative = derivative


def getIndex(points, x):
    dif = abs(points[0].x - x)
    index = 0
    for i in range(len(points)):
        if abs(points[i].x - x) < dif:
            dif = abs(points[i].x - x)
            index = i
    return index


def getWorkingPoints(points, index, n):
    left = index
    right = index
    for i in range(n - 1):
        if i % 2 == 0:
            if left == 0:
                right += 1
            else:
                left -= 1
        else:
            if right == len(points) - 1:
                left -= 1
            else:
                right += 1
    return points[left:right + 1]
```

```
def Newton(points, x, n):
    index = getIndex(points, x)
    workingPoints = getWorkingPoints(points, index, n)

    vals = [p.y for p in workingPoints]
    args = [p.x for p in workingPoints]
    difs = [vals[0]]
    col_num = len(vals)
    for i in range(1, col_num):
        for j in range(col_num - 1):
            vals[j] = (vals[j] - vals[j + 1]) / (args[j] -
args[j + i])
        difs.append(vals[0])
        col_num -= 1

    result = 0
    multiplier = 1
    for i in range(n):
        result += (difs[i] * multiplier)
        multiplier *= (x - args[i])
    return result


def Hermite(points, x, n):
    index = getIndex(points, x)
    workingPoints = getWorkingPoints(points, index, n)
    count = n

    vals = []
    args = []
    for i in range(count):
        args.append(workingPoints[int(i / 2)].x)
        vals.append(workingPoints[int(i / 2)].y)
    difs = [vals[0]]
    for j in range(count - 1):
        if j % 2 == 0:
            vals[j] = workingPoints[int(j / 2)].derivative
        else:
```

```
            vals[j] = (vals[j] - vals[j + 1]) / (args[j] -
args[j + 1])
    difs.append(vals[0])
    count -= 1
    for i in range(2, count + 1):
        for j in range(count - 1):
            vals[j] = (vals[j] - vals[j + 1]) / (args[j] -
args[j + i])
        difs.append(vals[0])
        count -= 1

    result = 0
    multiplier = 1
    for i in range(n):
        result += (difs[i] * multiplier)
        multiplier *= (x - args[i])
    return result


def result1(points, x):
    points.sort(key=lambda point: point.x)
    print('\nЗадача 1: Полиномы Ньютона и Эрмита\n')
    print("|-------------------------------------------------
|")
    print("|         |   n=1   |   n=2   |   n=3   |   n=4
|")
    print("|---------|---------|---------|---------|---------
|")
    print("| Ньютон  |", end="")
    for n in range(1, 5):
        print("{:9.6f}|".format(Newton(points, x, n + 1)),
end="")
    print("\n|---------|---------|---------|---------|-------
--|")
    print("|  Эрмит  |", end="")
    for n in range(1, 5):
        print("{:9.6f}|".format(Hermite(points, x, n + 1)),
end="")
```

```
    print("\n|-----------------------------------------------
--|")


def result2(points):
    print('\n\nЗадача 2: Обратная интерполяция\n')
    print("|---------------------------------------|")
    print("|         |   n=2   |   n=3   |   n=4   |")
    print("|---------|---------|---------|---------|")
    print("|  Корни  |", end="")
    for p in points:
        p.x, p.y = p.y, p.x
    points.sort(key=lambda point: point.x)
    x = 0
    for n in range(2, 5):
        print("{:9.6f}|".format(Newton(points, x, n + 1)),
end="")
    print("\n|---------------------------------------|")


if __name__ == "__main__":
    x = float(input("Введите x: "))
    points = [Point(0.00, 1.000000, -1.000000),
              Point(0.15, 0.838771, -1.14944),
              Point(0.30, 0.655336, -1.29552),
              Point(0.45, 0.450447, -1.43497),
              Point(0.60, 0.225336, -1.56464),
              Point(0.75, -0.018310, -1.68164),
              Point(0.90, -0.278390, -1.78333),
              Point(1.05, -0.552430, -1.86742)]
    result1(points, x)
    result2(points)
```

Результат работы программы

1.Значения y(x) при степенях полиномов Ньютона и Эрмита n = 1, 2, 3, 4(Значение x вводится с клавиатуры).

	n = 1	n = 2	n = 3	n = 4
Ньютон	0.337891	0.340208	0.340314	0.340324
Эрмит	0.342824	0.340358	0.340312	0.340323

2.Корень заданной выше табличной функции с помощью обратной интерполяции с использованием полинома Ньютона.

n = 2	n = 3	n = 4
0.739046	0.739079	0.739088

Ответы на вопросы

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

Да, но результат будет очень грубым, поскольку он будет представлять собой значение функции в точке, аргумент которой ближе всего к заданному значению (точка из исходной таблицы).

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

Чтобы оценить погрешность, можно проследить за убыванием членов ряда, и, если они убывают достаточно быстро, отбросить все после определённого. Его значение и будет погрешностью. Сложность использования теоретической оценки состоит в том, что производные интерполируемой функции обычно неизвестны.

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

Полином 3 степени (так как условий 4).

4. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?
При формировании конфигурации из n + 1 узлов, по возможности симметрично расположенных относительно аргумента, для которого выполняется интерполяция (если точки упорядочены по значению аргумента, то формирование такой конфигурации значительно упрощается, ведь все точки расположены друг за другом).

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

Для интерполяции быстроменяющихся функций возникает необходимость создавать таблицы очень больших объемов, что в ряде случаев неприемлемо. Тогда для повышения точности интерполяции применяют методвыравнивающих переменных: строят преобразование n = n(y) и e = e(x) так, чтобы в новых переменных (n, e) график на отдельных участках был близок к прямой, затем интерполируют и обратным интерполированием находят y = y(n).