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: Ковалец Кирилл

Группа: ИУ7-43Б

Оценка (баллы): _____

Преподаватель: Градов Владимир Михайлович

Москва
2021 г

Задание

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.

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

Теоретическая часть

Под аппроксимацией понимается некоторое приближение, приближенное представление.

Можно показать, что полином представим через разделенные разности в виде:

$$P_n(x) = P_n(x_0) + (x - x_0)P(x_0, x_1) + (x - x_0)(x - x_1)P(x_0, x_1, x_2) + \ldots$$

$$+ (x - x_0)(x - x_1)\ldots(x - x_{n-1})P(x_0, x_1, x_2, \ldots, x_n)$$

Интерполяционный многочлен Ньютона:

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

Погрешность многочлена Ньютона можно оценить по формуле

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

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

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

Полином Эрмита:

$$H_n(x) = P_n(x, \underbrace{x_0, x_0, \ldots, x_0}_{n_0}, x_1, x_1, \ldots, x_1, x_2, x_2, \ldots, x_2, \underbrace{x_k, x_k, \ldots, x_k}_{n_k}),$$

$$\sum_{l=0}^{k} n_l = n + 1,$$

$$y(x_0, x_0) = \lim_{x_1 \to x_0} \frac{y(x_0) - y(x_1)}{x_0 - x_1} = y'(x_0),$$

Алгоритм

1) Нахождение точек в таблице, которые будут участвовать при построении полинома Ньютона (узлы по возможности должны располагаться симметрично относительно значения X). Дальше работаем только с той частью таблицы, в которой содержаться нужные нам точки;

2) Нахождение разделённых разностей между соседними точками и запись результата в ячейку для хранения Y первой из двух выбранных точек. Параллельно с этим вычисляем полином Ньютона. Так продолжаем до тех пор, пока не будут вычислены все разделённые разности и вместе с ними сам полином.

3) Для нахождения полинома Эрмита вновь заполняем таблицу значений данными из файла. Увеличим таблицу в 2 раза таким образом, чтобы за каждой строкой была её копия. После чего найдём полином Ньютона для новой таблицы (при условии, что разделённая разность между двумя одинаковыми точками равна производной Y). Полученное значение будет являться полиномом Эрмита.

4) Для нахождения корня отсортируем исходную таблицу по Y и поменяем местами два первых столбца (X и Y). Далее воспользуемся алгоритмом поиска полинома Ньютона для X = 0. Полученное значение будет являться корнем табличной функции.

Код программы

```
EPS = 1e-6

def read_file(name_file):
    try:
        with open(name_file, "r") as f:
            table = [list(map(float, string.split())) for string in list(f)]
        return table

    except:
        print("Ошибка чтения файла!")
        return []

def read_data(size_table):
    try:
        n = int(input("Введите степень аппроксимирующего полинома - n: "))

        if (n <= 0):
            print("Степень палинома должна быть > 0!")
            return 1, 0, 0
        elif (n >= size_table):
            print("Слишком большая степень аппроксимирующего полинома для данной таблицы!")
            return 2, 0, 0

        x = float(input("Введите значение аргумента, для которого выполняется интерполяция: "))
        return 0, n, x

    except:
        print("Ошибка ввода данных!")
        return 3, 0, 0
```

```
def print_table(table):
    print("\n{:^12}{:^12}{:^12}\n".format("x", "y", "y`"))

    for i in range(len(table)):
        for j in range(len(table[i])):
            print("%-12f" %(table[i][j]), end = '')
        print()

    print()

def search_index(table, x, n):
    index = 0

    for i in table:
        if (i[0] > x):
            break
        index += 1

    l_border = index
    r_border = index

    while (n > 0):
        if (r_border - index == index - l_border):
            if (l_border > 0):
                l_border -= 1
            else:
                r_border += 1
        else:
            if (r_border < len(table) - 1):
                r_border += 1
            else:
                l_border -= 1
```

```
        n -= 1
    return l_border
```

```
68  def divided_difference(x0, y0, x1, y1, y_der):
69      if (abs(x0 - x1) > EPS):
70          return (y0 - y1) / (x0 - x1)
71      else:
72          return y_der
73
74  def newton_polynomial(table, n, x):
75      index = search_index(table, x, n)
76      np = table[index][1]
77
78      for i in range(n):
79          for j in range(n - i):
80              table[index + j][1] = divided_difference(
81                  table[index + j][0],         table[index + j][1],
82                  table[index + j + i + 1][0], table[index + j + 1][1],
83                  table[index + j][2])
84
85          mult = 1
86          for j in range(i + 1):
87              mult *= (x - table[index + j][0])
88
89          mult *= table[index][1]
90          np += mult
91
92      return np
93
94  def table_extension(table):
95      new_size = 2 * len(table)
96
97      for i in range (0, new_size, 2):
98          table.insert(i + 1, table[i][:])
99
100 def hermit_polynomial(name_file, n, x):
101     table = read_file(name_file)
102     table.sort(key = lambda array: array[0])
103     table_extension(table)
104
105     return newton_polynomial(table, n, x)
106
107 def root_search(name_file, n):
108     table = read_file(name_file)
109     table.sort(key = lambda array: array[1])
110
111     for i in table:
112         i[0], i[1] = i[1], i[0]
113
114     return newton_polynomial(table, n, 0)
115
116 def main():
117     name_file = "./data.txt"
118
119     table = read_file(name_file)
120     if (table == []):
121         return
122
123     table.sort(key = lambda array: array[0])
124     print_table(table)
125
126     r, n, x = read_data(len(table))
127     if (r):
128         return
129
130     np = newton_polynomial(table, n, x)
131     hp = hermit_polynomial(name_file, n, x)
132     root = root_search(name_file, n)
133
```

```
    print("\nNewton_p = %.6f" %(np))
    print("\nHermit_p = %.6f" %(hp))
    print("\nRoot = %.6f\n" %(root))

main()
```

Пример работы программы

```
     x           y           y`

0.000000    1.000000    -1.000000
0.150000    0.838771    -1.149440
0.300000    0.655336    -1.295520
0.450000    0.450447    -1.434970
0.600000    0.225336    -1.564640
0.750000    -0.018310   -1.681640
0.900000    -0.278390   -1.783330
1.050000    -0.552430   -1.867420

Введите степень аппроксимирующего полинома - n: 2
Введите значение аргумента, для которого выполняется интерполяция: 0.525

Newton_p = 0.340208

Hermit_p = 0.340288

Root = 0.739174
```

Результаты работы программы при данных значениях

```

x = 0.525

-------------------------------------------------------------
N        |      1     |      2     |      3     |      4     |
-------------------------------------------------------------
Newton_p |  0.337891  |  0.340208  |  0.340314  |  0.340324  |
-------------------------------------------------------------
Hermit_p |  0.337891  |  0.340288  |  0.340323  |  0.340324  |
-------------------------------------------------------------

y = 0

-----------------------------------------------------------
N     |      1     |      2     |      3     |      4     |
-----------------------------------------------------------
Root  |  0.738727  |  0.739174  |  0.739095  |  0.739081  |
-----------------------------------------------------------

```

Вопросы при защите лабораторной работы.

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

Программа работать не будет, так как для интерполяции необходимо минимум 2 узла, а при степени полинома n = 0 будет только 1 узел.

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

Практически оценить погрешность интерполяции можно при помощи оценки первого отброшенного члена в полиноме Ньютона. При этом в полиноме остаются члены, которые больше заданной погрешности расчетов.

Теоретическую погрешность многочлена Ньютона можно оценить с помощью формулы:

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

где $M_{n+1} = max\ |y^{(n+1)}(\xi)|$ - 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 минимальной степени может быть построен на этих точках?

Минимальная степень для построения полинома равна 1, так как при данных значениях можно построить полином Ньютона при n = 1, полином Эрмита – при n = 1, 2, 3.

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

Для нахождения полинома нужно определить n + 1 узел так, чтобы они по возможности располагались симметрично относительно заданного аргумента. При неупорядоченном аргументе значение функции может получиться неточным или вовсе неверным.

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

Если функция, а точнее ее разделенные разности, значительно меняются на нескольких интервалах, то интерполяция обобщенным многочленом обычно не будет точной для дифференцирования данной функции. Поэтому для таких функций используется квазилинейная интерполяция, которая производится при помощи выравнивающих переменных. То есть выравнивающие переменные используются для того, чтобы повысить точность вычисления производной функции.