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

«Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций»

Студент Маслова Марина Дмитриевна
фамилия, имя, отчество

Группа ИУ7-43Б

Оценка (баллы) ____________________

Преподаватель Градов Владимир Михайлович
фамилия, имя, отчество

2020 г.

Оглавление

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.

Описание алгоритма

Для представления полинома Ньютона используются разделенные разности, которые вычисляются по формулам:

$$y(x_i, x_j) = \frac{y(x_i) - y(x_j)}{x_i - x_j}$$

(для двух узлов)

$$y(x_i, x_j, x_k) = \frac{y(x_i, x_j) - y(x_j, x_k)}{x_i - x_k}$$

(для трех узлов)

$$y(x_i, x_j, \dots, x_n) = \frac{y(x_i, x_j, \dots, x_{n-1}) - y(x_j, \dots, x_n)}{x_i - x_n}$$

(для n узлов)

Через них полином Ньютона, а также значение функции при заданном значении аргумента выражается следующим образом:

$$y(x)=P_n(x)=y0+(x-x_0)y(x_0,x_1)+(x-x_0)(x-x_1)y(x_0,x_1,x_2)+\\+\ldots+(x-x_0)\ldots(x-x_{n-1})y(x_0,x_1,\ldots,x_n)$$

Для определение коэффициентов полинома Ньютона исходный набор узлов сортируется по возрастанию аргумента, после чего выбирается множество ближайших к заданному аргументу узлов, по которому строится таблица разделенных разностей и соответственно находятся коэффициенты полинома Ньютона, по найденным коэффициентам и заданному значению аргумента находится соответствующее значение функции.

Построение полинома Эрмита аналогично построению полинома Ньютона, но в силу того, что полином Эрмита строится на кратных узлах, значения разделенных разностей, при вычислении которых возникает неопределенность $0/0$, заменяются на значения производных.

Для нахождения корня заданной табличной функции с помощью обратной интерполяции используется алгоритм интерполяции полиномом Ньютона, но на вход ему подается значение аргумента, равное нулю, и обратная функция, то есть таблица, где столбец, отвечающий за аргументы, перенесен на место столбца значений функции, и наоборот.

Код программы

Код программы представлен на листингах 1-2.

Листинг 1. interpolation.py

```
"""
    Модуль, реализующий полиномиальную
    интерполяцию табличных функций
"""


def read_table(file_name):
    """
        Чтение табличной функции из файла
    """

    func_table = []

    with open(file_name, "r") as file:
        for i, rec in enumerate(file):
            func_table.append(list(map(float, rec.split())))

            if len(func_table[i]) != 3:
                raise TypeError

    if not func_table:
        raise EOFError
```

```
    return func_table



def print_table(table):
    """
        Вывод табличной функции
    """

    if table:
        print("Загруженная таблица:")
        print("      x        y         y'")
    else:
        print("Пустой файл!")

    for rec in table:
        print("    {:.2f}  {:9.6f}  {:.5f}".format(rec[0], rec[1], rec[2]))


def print_list(res_list):
    """
        Вывод списка для таблицы
    """

    for y in res_list:
        print("{:9.6f}".format(y), end="  ")
    print()


def print_result(newton, hermit, root):
    """
        Вывод таблицы значений функции и корней при
        различных степенях полиномов Ньютона и Эрмита
    """

    print("\n----------------------------------------------------")
    print("   Вид                       Степень ")
    print(" Полинома        1         2         3         4")
    print("----------------------------------------------------")

    print("{:^11}".format("Ньютона"), end="  ")
    print_list(newton)

    print("{:^11}".format("Эрмита"), end="  ")
    print_list(hermit)

    print("{:^11}".format("Корень¹"), end="  ")
    print_list(root)

    print("----------------------------------------------------")

    print("\n¹ -- корень заданной табличной функции, полученный")
    print("     с помощью обратной интерполяции")


def find_x_position(table, arg):
    """
        Поиск положения заданного аргумента в таблице
    """

    prev_arg_index = 0
    num_table_args = len(table)

    while (prev_arg_index < num_table_args and
           arg > table[prev_arg_index][0]):
        prev_arg_index += 1
```

```python
    return prev_arg_index - 1


def create_calc_table(table, arg_position, coef_num):
    """
        Выбор значений для подсчета коэффициентов полинома
    """

    res_table = []

    begin = arg_position - coef_num // 2 + 1
    begin = begin if begin >= 0 else 0
    begin = begin if begin + coef_num < len(table) else len(table) - coef_num

    for i in range(begin, begin + coef_num):
        res_table.append([x for x in table[i]])

    return res_table


def delete_derivative(table):
    """
        Удаление слобца производных
    """

    for rec in table:
        rec.pop()


def calc_divided_difference(y0, y1, x0, x1):
    """
        Подсчет разделенной разности
    """

    return (y0 - y1) / (x0 - x1)


def calc_coef(calc_table, first_col):
    """
        Подсчет коэффициентов полинома
        с помощью разделенных разностей
    """

    for y in range(first_col, len(calc_table)):
        for i in range(0, len(calc_table) - y):
            calc_table[i].append(calc_divided_difference(
                calc_table[i][y],
                calc_table[i + 1][y],
                calc_table[i][0], calc_table[i + y][0]))


def calc_func(calc_table, arg):
    """
        Подсчет значения функции с помощью таблицы разделенных разностей
    """

    result = 0
    mul = 1

    for i in range(1, len(calc_table[0])):
        result += calc_table[0][i] * mul
        mul *= arg - calc_table[i - 1][0]

    return result


def newton_find_y(table, arg, power):
    """
        Поиск значения отсортированной по аргументу табличной
```

```
        функции с помощью интерполяции полиномом Ньютона
    """

    arg_position = find_x_position(table, arg)
    calculaton_table = create_calc_table(table, arg_position, power + 1)
    delete_derivative(calculaton_table)
    calc_coef(calculaton_table, 1)
    result = calc_func(calculaton_table, arg)

    return result


def create_node(nearest_nodes):
    """
        Создание узла таблицы
    """

    der = ([] if len(nearest_nodes) != 2
           else [calc_divided_difference(
               nearest_nodes[0][1],
               nearest_nodes[1][1],
               nearest_nodes[0][0],
               nearest_nodes[1][0]
           )]
          )

    return [nearest_nodes[0][:2] + der]


def add_node(table, power):
    """
        Добавление повторных узлов
    """

    needed_num = power + 1
    cur_num = len(table)

    i = 0
    while cur_num < needed_num:
        new_node = create_node(table[i:i + 2])
        table = table[:i + 1] + new_node + table[i + 1:]
        cur_num += 1
        i += 2

    i = power

    table[i] = table[i][:2]

    return table


def hermit_find_y(table, arg, power):
    """
        Поиск значения отсортированной по аргументу табличной
        функции с помощью интерполяции полиномом Эрмита
    """

    arg_position = find_x_position(table, arg)
    calculaton_table = create_calc_table(table, arg_position, power // 2 + 1)
    calculaton_table = add_node(calculaton_table, power)
    calc_coef(calculaton_table, 2)


    return calc_func(calculaton_table, arg)


def newton_find_root(table, power):
    """
        Поиск корня с помощью обратной интерполяции
```

```
    """

    deep_copy = []

    for rec in table:
        deep_copy.append([])
        for arg in rec:
            deep_copy[len(deep_copy) - 1].append(arg)

    for rec in deep_copy:
        rec[0], rec[1] = rec[1], rec[0]

    deep_copy.sort(key=lambda row: row[0])

    res = newton_find_y(deep_copy, 0, power)

    return res
```

Листинг 2. main.py

```
"""
    Модуль для запуска программы
    ЛАБОРАТОРНАЯ РАБОТА #1
    ПОСТРОЕНИЕ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОЛИНОМИАЛЬНОЙ
    ИНТЕРПОЛЯЦИИ ТАБЛИЧНЫХ ФУНКЦИЙ
"""

import argparse

import interpolation as interp


LOWER = 1
UPPER = 4


def create_args():
    """
        Добавление аргументов командной строки
    """

    parser = argparse.ArgumentParser()
    parser.add_argument('file_name', nargs='?', default='data/data_01.txt')
    args = parser.parse_args()

    return args


if __name__ == "__main__":
    ARGS = create_args()

    try:
        func_table = interp.read_table(ARGS.file_name)

        func_table.sort(key=lambda table: table[0])
        interp.print_table(func_table)

        x = float(input("\nВведите значения аргумента для интерполяции: "))

    except FileNotFoundError:
        print("\nТакого файла не существует!")

    except ValueError:
        print("\nНечисловые данные недопустимы!")
        print("\nПроверьте содержимое файла или введенный аргумент!")

    except EOFError:
        print("\nПустой файл!")
```

```
    except TypeError:
        print("\nВ файле должно быть три столбца с данными!")

    else:
        newton = []
        hermit = []
        root = []
        for n in range(LOWER, UPPER + 1):
            newton.append(interp.newton_find_y(func_table, x, n))
            hermit.append(interp.hermit_find_y(func_table, x, n))
            root.append(interp.newton_find_root(func_table, n))

        interp.print_result(newton, hermit, root)
```

Результат работы

Значения $y(x)$ и корня заданной табличной функции при различных степенях полиномов для :

Степень полинома	1	2	3	4
Полином Ньютона	0.337891	0.340208	0.340314	0.340324
Полином Эрмита	0.342684	0.340358	0.340323	0.340324
Корень функции	0.738727	0.739174	0.739095	0.739081

Контрольные вопросы

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

При степени полинома $n=0$ программа будет работать, она выберет значение аргумента, ближайшее к заданному x, и выдаст значение функции при выбранном аргументе. Такое поведение программы основано на том, что полином степени $n=0$ задает константу, а участок между двумя соседними узлами представляет отрезок, параллельный оси абсцисс при значении ординаты, равном её значению у одного из выбранных узлов.

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

На практике для оценки точности расчета используют эмпирическое правило, заключающееся в наблюдении за скоростью убывания членов суммы:

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

При быстром убывании оставляют только те члены, которые больше допустимой погрешности; таким образом определяют, сколько узлов требуется включить в расчет. Если же количество узлов задано, то для определения погрешности используют оценку первого отброшенного члена.

Для теоретической же оценки требуется наличие значений производных заданной функции, которые обычно не известны, поэтому применить такую оценку сложно.

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

При задании в двух точках значений функций и её производных имеем 4 условия, которые должны быть использованы. Из 4-ех условий можно найти 4 коэффициента, то есть минимально возможная степень полинома будет равна 3.

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

При определении значения функции методом интерполяции важна точность найденного значения, при этом чем ближе друг к другу расположены узлы, тем точнее полином будет заменять исходную функцию. При построении полинома n -ой степени должен быть выбран $n+1$ узел. Для большей точности необходимо выбрать ближайшие к заданному аргументу узлы, что достигается упорядоченностью аргумента функции.

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

При большом шаге сетки или при быстро меняющихся функциях погрешность может увеличиваться при увеличении порядка точности формулы. Для повышения точности интерполяции используют преобразование переменных $\eta = \eta(y)$ и $\epsilon = \epsilon(x)$ такое, что в новых переменных, называющихся выравнивающими, $\eta(\epsilon)$ мало отличается от прямой хотя бы на отдельных участках. Тогда в переменных $\eta(\epsilon)$ интерполяция полиномом невысокой степень дает хорошую точность. А для вычисления значения функции в исходных переменных используют обратное преобразование $y = y(\eta)$.