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

Тема: Алгоритм и программа построения интерполяционных полиномов Ньютона и Эрмита

Студент: Рядинский К. В.

Группа: ИУ7-43Б

Оценка (баллы): _________

Преподаватель: Градов В.М.

Москва. 2020 г.

Цель работы

Изучить метод нахождения значения функции в заданной точке с помощью интерполяционных полиномов Ньютона и Эрмита.

Задание

1. Найти **Pn(x)** и **Hn(x)**
2. Сравнить результаты вычисления значения функции обоими методами
3. Найти корень функции методом обратной интерполяции

Входные данные

1. Таблица координат
2. Координата точки по оси абсцисс
3. Степень искомых полиномов

Выходные данные

1. Значение функции в точке X, найденное двумя методами
2. Корень функции, найденный с помощью метода обратной интерполяции

Анализ алгоритма

В алгоритме подсчитываются разделенные разности.
Они вычисляются по формуле (1, 2, 3 степени):

$$y(x_i, x_j) = [y(x_i) - y(x_j)]/(x_i - x_j),$$

$$y(x_i, x_j, x_k) = [y(x_i, x_j) - y(x_j, x_k)]/(x_i - x_k),$$

$$y(x_i, x_j, x_k, x_l) = [y(x_i, x_j, x_k) - y(x_j, x_k, x_l)]/(x_i - x_l)$$

Далее с помощью этих разделенных разностей подсчитывается полином Ньютона, имеющий формулу:

$$P_n(x) = y_0 + \sum_{k=0}^{n} (x - x_n) \dots (x - x_{k-1}) y(x_0, x_1, \dots, x_k)$$

При нахождении полинома Эрмита, производные функции в точке рассматриваются в качестве предельного перехода от разделенных разностей, следовательно:

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

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

$$y(x_0, x_0, x_1, x_1) = \frac{y(x_0, x_0, x_1) - y(x_0, x_1, x_1)}{x_0 - x_1} = \frac{y'(x_0) - 2y(x_0, x_1) + y'(x_1)}{(x_0 - x_1)^2}.$$

И ход решения имеет вид

x_i	y_i	$y(x_k, x_m)$	$y(x_k, x_m, x_l)$
x_0	y_0	y_0'	$(y_0' - y(x_0, x_1))/(x_0 - x_1)$
x_0	y_0	$y(x_0, x_1)$	$(y(x_0, x_1) - y_1')/(x_0 - x_1)$
x_1	y_1	y_1'	и.т.д.
x_1	y_1	$y(x_1, x_2)$	
x_2	y_2	y_2'	
x_2	y_2		

При поиске корня обратной интерполяцией, столбцы меняются местами, а аргумент задается равным 0

Исходные данные

x	y	y’
0	1	-1
0,15	0,838	-1,149
0,30	0,655	-1,295
0,45	0,450	-1,434
0,6	0,225	-1,564
0,75	-0,018	-1,681
0,9	-0,278	-1,783
1,05	-0,552	-1,867

Результаты

X = 0.525	y		
Степень полинома	Полином Ньютона	Полином Эрмита	Значение корня
1	0,3375	0,34245	0,75
2	0,33975	0,339975	0,739316239
3	0,339875	0,34016875	0,739192378
4	0,339851562	0,340014062	0,739234906
5	0,339863281	0,340123047	0,739243464
6	0,33984375	0,340069336	0,739243797

Код

```python
import logging
import interpolation
import sys
import prettytable
import math as m

def main():
    logging.basicConfig(level=logging.INFO)

    if len(sys.argv) < 2:
        print("Few arguments.\nUse main.py filename")
        logging.error("Few arguments")
        sys.exit(1)

    n = int(input("Input polynom grade: "))
    x = float(input("Input x: "))

    interp = interpolation.interFunc(n)

    table = interp.get_table_from_file(sys.argv[1])
    table = interp.set_table_slice(x)

    x_y_dy_table = prettytable.PrettyTable()

    x_y_dy_table.field_names = ["x", 'y', 'dy/dx']
    for i in table:
        x_y_dy_table.add_row([i.x, i.y, i.dy])

    print(x_y_dy_table)

    ptable = prettytable.PrettyTable()
    ptable.field_names = ["x", "Newton", "Ermit"]

    ptable.add_row([x, round(interp.newton_polynom(x), 9), round(interp.ermit_polynom(x), 9)])
    print(ptable)

    interp.invert()
    print("Root: ", round(interp.newton_polynom(0), 9))


if __name__ == '__main__':
    main()

```

```python
import logging
import math

class funcTable:
    def __init__(self, x:float, y:float, dy:float) -> None:
        self.x = x
        self.y = y
        self.dy = dy


class interFunc:
    table = []
    diffs = []

    ermit_table = []
    ermit_diffs = []

    def __init__(self, polynom_size:int):
        self.polynom_size = polynom_size

    def get_table_from_file(self, filename:str, ) -> list:
        try:
            with open(filename, "r") as f:
                for line in f:
                    x, y, dy = map(float, line.split())
                    record = funcTable(x, y, dy)
                    self.table.append(record)
            self.table.sort(key=lambda x: x.x)
        except OSError:
            logging.error("Unexisting file")
            print("That file doesn't exists. Try again.")
            return None

        return self.table

    def set_table_slice(self, x:float) -> list:
        num_of_records = self.polynom_size + 1

        for ind, rec in enumerate(self.table):
            if rec.x >= x:
                logging.debug(f"Index of {ind}\tnum of records {num_of_records}")
                half = num_of_records / 2
                half = math.ceil(half)

                indent = num_of_records

```

```python
                while ind < len(self.table) and half > 0:
                    half -= 1
                    ind += 1

                ind -= indent

                if (ind < 0):
                    ind = 0


                logging.debug(f"Index of {ind}\tnum of records {num_of_records}")
                self.table = self.table[ind:ind + self.polynom_size + 1]


                self.__set_diffs()
                self.__double_table()
                self.__set_ermit_diffs()
                return self.table

        self.table = self.table[len(self.table) - 1 - self.polynom_size:len(self.table)]

        self.__set_diffs()
        self.__double_table()
        self.__set_ermit_diffs()

        return self.table

    def get_sep_diff(self, vals:tuple) -> float:
        if len(vals) == 2:
            return (self.table[vals[0]].y - self.table[vals[1]].y) / (self.table[vals[0]].x - self.table[vals[1]].x)

        return (self.get_sep_diff(vals[0:len(vals) - 1]) - self.get_sep_diff(vals[1:len(vals)])) / (self.table[vals[0]].x - self.table[vals[-1]].x)


    def newton_polynom(self, x:float) -> float:
        y_z = self.table[0].y

        for i in range(self.polynom_size):
            val = self.diffs[i]

            for j in range(i + 1):
                val *= (x - self.table[j].x)
            y_z += val

        return y_z

```

```python

    def invert(self):
        for i in range(len(self.table)):
            self.table[i].x, self.table[i].y = self.table[i].y, self.table[i].x
        self.__set_diffs()

    def __set_diffs(self):
        ind = [0, 1]
        self.diffs = []

        for i in range(self.polynom_size):
            self.diffs.append(self.get_sep_diff(tuple(ind)))
            ind.append(i + 2)


    # ------------ ERMIT ------------

    def get_ermit_sep_diff(self, vals:tuple) -> float:
        if len(vals) == 2:
            if (self.ermit_table[vals[0]].x == self.ermit_table[vals[1]].x): # (x_0, x_0)
                return self.ermit_table[vals[0]].dy
            return (self.ermit_table[vals[0]].y - self.ermit_table[vals[1]].y) / (self.ermit_table[vals[0]].x - self.ermit_table[vals[1]].x)

        return (self.get_ermit_sep_diff(vals[0:len(vals) - 1]) - self.get_ermit_sep_diff(vals[1:len(vals)])) / (self.ermit_table[vals[0]].x - self.ermit_table[vals[-1]].x)

    def __double_table(self):
        for record in self.table:
            self.ermit_table.append(record)
            self.ermit_table.append(record)

    def __set_ermit_diffs(self):
        ind = [0, 1]
        self.ermit_diffs = []

        for i in range(self.polynom_size):
            self.ermit_diffs.append(self.get_ermit_sep_diff(tuple(ind)))
            ind.append(i + 2)


    def ermit_polynom(self, x:float):
        y_z = self.ermit_table[0].y

        for i in range(self.polynom_size):
            val = self.ermit_diffs[i]

            for j in range(i + 1):
                val *= (x - self.ermit_table[j].x)
            y_z += val

        return y_z

```

Контрольные вопросы

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

Да будет, но будет использоваться только свободный член полинома

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

Практически можно оценить через первый отброшенный член полинома.
Теоретическую провести тяжело, так как для нее нужна производная, которая не всегда имеется.

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

Минимальная 0
Максимальная 3. Так как на n степень нужно n + 1 значений

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

При вычислении раздельных разностей. Удобнее всего отсортировать в порядке возрастания

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

Это переменные, в которых график функции близок к линейному.
Часто применяются в функциях, которые быстро меняются. Чтобы повысить точность, строят полином Ньютона для этих переменных, а потом вместо них используют x и y