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 №2

Тема: Билинейная интерполяция

Студент: Миронов Г.А.

Группа: ИУ7-43Б

Оценка (баллы): _________

Преподаватель: Градов В.М.

Москва.
2020 г.

Задание

1. Задана матрица значений функции вида X, Y, F(X, Y). С помощью интерполяции, используя метод полинома Ньютона, найти приближенное значение функции от введенных X и Y

Входные данные

1. Таблица координат
2. Координата точки по осям абсцисс и ординат
3. Степени полинома для X и Y

Выходные данные

1. Значение функции в точке (X, Y)

Анализ алгоритма

Билинейная интерполяция основывается на линейной интерполяции.

Для 2-х точек:

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

Для 3-х:

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

Соответственно, для n-точек:

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

Отсюда искомый полином будет равен:

$$\mathcal{P}_n(x) = y_0 + (x - x_0)y(x_0, x_1) + (x - x_0)y(x_0, x_1)y(x_0, x_1, x_3) + ... + (x - x_0)...(x - x_n)y(x_0, x_1, ..., x_n)$$

Что эквивалентно

$$\mathcal{P}_n(x) = \sum a_k x^k$$

Первым шагом линейно интерполируется значение вспомогательных точек R1 и R2 вдоль оси абсцисс, где

$$R_1 = (x, y_1)$$

$$R_2 = (x, y_2)$$

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

Теперь проводится линейная интерполяция между вспомогательными точками R1 и R2. Это и есть интерполирующее значение ф-ии F(X, Y)

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

Таким образом, общая формула имеет вид

$$f(x, y) = \sum_{i=0}^{1} \sum_{j=0}^{1} a_{ij} x^i y^j = a_{00} + a_{10}x + a_{01}y + a_{11}xy$$

Исходные данные

Y \ X	0	1	2	3	4
0	0.00	1.00	4.00	9.00	16.00
1	1.00	2.00	5.00	10.00	17.00
2	4.00	5.00	8.00	13.00	20.00
3	9.00	10.00	13.00	18.00	25.00
4	16.00	17.00	20.00	25.00	32.00

Результаты вычислений

Для полиномов 1-4 степеней в точке (1.5, 1.5)

	$n_x = 1$	$n_x = 2$	$n_x = 3$	$n_x = 4$
$n_y = 1$	5	4.75	4.75	4.75
$n_y = 2$	4.75	4.5	4.5	4.5
$n_y = 3$	4.75	4.5	4.5	4.5
$n_y = 4$	4.75	4.5	4.5	4.5

Исходный код

Код программы представлен на листингах 1-3

Листинг 1. `utils.py`

```
from polynomial import Dot


def read_dots(fname: str) -> list[list[Dot]]:
    dots = []

    with open(fname) as fin:
        dots += [[Dot(x, None)
                  for x in list(map(float, fin.readline().split()))]]

        for line in fin.readlines():
            dots += [[Dot(x, None) for x in list(map(float, line.split()))]]

    return sorted(dots, key=lambda p: p[0].x)


def print_dots(dots: list[list[Dot]]) -> None:
    print(" Y \ X ", end="")
    for i in dots[0]:
        print("{: >6.2f}".format(i.x), end=" ")
    print()

    for row in dots[1:]:
        for col in row:
            print("{: >6.2f}".format(col.x), end=" ")
        print()


def read_func_data() -> tuple[int, int, float, float]:
    degx, degy, x, y = map(float, input().split())

    return int(degx), int(degy), x, y
```

Листинг 2. `polynomials.py`

```python
from __future__ import annotations
from copy import deepcopy
from math import prod


class Dot(object):
   x: float
   y: float

   def __init__(self, x: float, y: float):
       self.x, self.y = x, y

   def __str__(self):
       return "(" + str(self.x) + ";" + str(self.y) + ")"


class Polynomial(object):
   terms: list[callable[float]:float]

   def __init__(self, terms: list[callable[float]:float]):
       self.terms = terms

   def __call__(self, arg: float) -> float:
       return sum([term(arg) for term in self.terms])


class NewtonPolynomial(Polynomial):

   @staticmethod
   def build(points: list[Dot], arg: Dot, n: int) -> NewtonPolynomial:
       table = NewtonPolynomial._make_table(points, arg, n)

       return NewtonPolynomial(
           [lambda x:table[1][0]] +
           [NewtonPolynomial._term(table[i][0],  table[0][:i - 1])
            for i in range(2, len(table))]
       )

   @staticmethod
   def _term(va: float, vl: list[float]) -> callable:
       return lambda x: va * prod(map(lambda a: (x - a), vl))

   @staticmethod
   def _make_table(points: list[Dot], arg: Dot, n: int) -> list[list[Dot]]:
       base = sorted(
           sorted(points, key=lambda p: abs(p.x - arg.x))[:n + 1],
           key=lambda p: p.x
       )
```

```
        t = [[None for i in range(len(base))] for j in range(n + 2)]

        for i in range(len(t[0])):
            t[0][i], t[1][i] = base[i].x, base[i].y

        for i in range(2, len(t)):
            for j in range(len(base) - i + 1):
                t[i][j] = (t[i - 1][j] - t[i - 1][j + 1]) / \
                    (t[0][j] - t[0][j + i - 1])

        return t


class BiNewtonPolynomial(Polynomial):

    __second_interp_set: list
    __ny: float

    def __init__(self, temp: list, ny: int):
        self.__second_interp_set = temp
        self.__ny = ny

    @staticmethod
    def build(dots: list[list[Dot]], arg: Dot, nx: int, ny: int) ->
BiNewtonPolynomial:
        points = deepcopy(dots)
        xrow, ycol, matrix = BiNewtonPolynomial.__split_data(points)
        baseX, baseY = BiNewtonPolynomial.__get_bases(xrow, ycol, arg, nx, ny)

        t = [[None for i in range(nx + 1)] for j in range(ny + 1)]

        k = 0
        for i in range(xrow.index(baseX[0]), xrow.index(baseX[0]) + nx + 1):
            l = 0
            for j in range(ycol.index(baseY[0]), ycol.index(baseY[0]) + ny + 1):
                t[l][k] = matrix[i][j]
                l += 1
            k += 1

        second_set = []
        for i in range(len(t)):
            for j in range(len(baseX)):
                t[i][j].y = baseX[j].x
                t[i][j].x, t[i][j].y = t[i][j].y, t[i][j].x
            second_set += [
                Dot(
                    baseY[i].x,
                    NewtonPolynomial.build(t[i], Dot(arg.y, arg.x), nx)
                )
            ]
```

```
        return BiNewtonPolynomial(
            second_set,
            ny
        )

    @staticmethod
    def __split_data(matrix: list[list[Dot]]) -> tuple[list[Dot], list[Dot],
list[list[Dot]]]:
        xrow = matrix[0]
        matrix = matrix[1:]
        ycol = [row[0] for row in matrix]
        matrix = [row[1:] for row in matrix]

        return xrow, ycol, matrix

    @staticmethod
    def __get_bases(x: list[Dot], y: list[Dot], arg: Dot, nx: int, ny: int):
        baseX = sorted(
            sorted(x, key=lambda p: abs(p.x - arg.x))[:nx + 1],
            key=lambda p: p.x
        )
        baseY = sorted(
            sorted(y, key=lambda p: abs(p.x - arg.y))[:ny + 1],
            key=lambda p: p.x
        )
        return baseX, baseY

    def __call__(self, arg: Dot) -> float:
        t = [Dot(i.x, i.y(arg.y)) for i in self.__second_interp_set]
        return NewtonPolynomial.build(t, arg, self.__ny)(arg.x)
```

Листинг 3. `main.py`

```
from sys import argv

import matplotlib.pyplot as plt
import numpy as np

from copy import deepcopy

from utils import *
from polynomial import *


def plot(dots: list[list[Dot]], polynomial: BiNewtonPolynomial) -> None:
   plt.clf()

   matrix = deepcopy(dots)
   xrow = matrix[0]
   matrix = matrix[1:]
   ycol = [row[0] for row in matrix]
   matrix = [row[1:] for row in matrix]

   x_s, z_s, y_s = [], [], []
   for i in range(len(xrow)):
       for j in range(len(ycol)):
           x_s.append(xrow[i].x)
           y_s.append(ycol[j].x)
           z_s.append(matrix[i][j].x)

   axis = plt.figure(1).add_subplot(111, projection='3d')
   axis.set_xlabel('X')
   axis.set_ylabel('Y')
   axis.set_zlabel('Z')

   axis.scatter(x_s, y_s, z_s, color='red')

   x_arg = np.linspace(min([x.x for x in dots[0]]),
                       max([x.x for x in dots[0]]), 20)
   y_arg = np.linspace(min([y.x for y in [row[0] for row in dots]]),
                       max([y.x for y in [row[0] for row in dots]]), 20)
   z_arg = np.empty([len(x_arg), len(y_arg)])

   for i in range(len(x_arg)):
       for j in range(len(y_arg)):
           z_arg[i][j] = polynomial(Dot(x_arg[i], y_arg[j]))

   X, Y = np.meshgrid(x_arg, y_arg)
   axis.plot_wireframe(X, Y, z_arg)

   plt.show()
```

```python
if __name__ == "__main__":
    dots = read_dots(argv[1])

    print("Loaded table:")

    print_dots(dots)

    print("Enter polynomial X, Y degree and X value:")
    degx, degy, x, y = read_func_data()

    polynomial = BiNewtonPolynomial.build(dots, Dot(x, y), degx, degy)

    print(
        "Value in ({: >5.6g}; {: >5.6g}) is {: <5.6g}".format(
            x,
            y,
            polynomial(Dot(x, y))
        )
    )

    plot(dots, polynomial)
```

Контрольные вопросы

1. Пусть производящая функция таблицы суть $z(x,y)=x^2+y^2$. Область определения по x и y 0-5 и 0-5. Шаги по переменным равны 1. Степени $n_x = n_y = 1$, $x=y=1.5$. Приведите по шагам те. значения функции, которые получаются в ходе последовательных интерполяций. по строкам и столбцу.

По строкам:

1.0	3.5
2.0	6.5

По столбцу: 5.0

2. Какова минимальная степень двумерного полинома, построенного на четырех узлах? На шести узлах?

4 узла: Берем базу из 2-х элементов для интерполяции по X и по Y (“квадрат”)
Выходит, 2 полинома 1 степени при первой интерполяции. Аналогично, второй этап дает полином 1 степени
6 узлов: Берем базу из 2-х и 3-х элементов для интерполяции по X и по Y (“прямоугольник”). Конкретные размерности баз для осей не принципиальны.
Пусть, по X - 3 элемента, по Y - 4.
Первая интерполяция даст 3 полинома 1 степени, вторая - полином 2 степени.

3. Предложите алгоритм двумерной интерполяции при хаотичном расположении узлов, т.е. когда таблицы функции на регулярной сетке нет, и метод последовательной интерполяции не работает. Какие имеются ограничения на расположение узлов при разных степенях полинома?

В таком случае, обычно ограничиваются малыми степенями многочленов, приравнивая их к табличным значениям функции (предполагаемой). Для первой степени:

$$z \approx a + bx + cy,$$
$$z_i = a + bx_i + cy_i, \qquad i = 1,\ 2,\ 3.$$

Или

$$\begin{vmatrix} z & 1 & x & y \\ z_1 & 1 & x_1 & y_1 \\ z_2 & 1 & x_2 & y_2 \\ z_3 & 1 & x_3 & y_3 \end{vmatrix} = 0.$$

Откуда

$$z = [z_1 \Delta(\boldsymbol{r}, \boldsymbol{r}_2, \boldsymbol{r}_3) + z_2 \Delta(\boldsymbol{r}_1, \boldsymbol{r}, \boldsymbol{r}_3) + \\ + z_3 \Delta(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r})] \,/\, \Delta(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3).$$

4. Пусть на каком-либо языке программирования написана функция, выполняющая интерполяцию по двум переменным. Опишите алгоритм использования этой функции для интерполяции по трем переменным.

1. Выбрать ближайшие к интерполируемой точке nz+1 плоскостей, заданных в таблице
2. Провести двумерную интерполяцию в каждой из плоскостей
3. По полученным значениям провести одномерную интерполяцию

5. Можно ли при последовательной интерполяции по разным направлениям использовать полиномы несовпадающих степеней или даже разные методы одномерной интерполяции, например, полином Ньютона и сплайн?

Можно.

6. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.

Треугольная конфигурация задает (n+1)(n+2)/2 узлов. Т.е. однозначно задает многочлен n-ой степени.
Это удобно записать в виде формы Ньютона, введя разделенные разности функции 2-х переменных

$$z(x_0, x_1; y) = [z(x_0, y) - z(x_1, y)]/(x_0 - x_1),$$
$$z(x; y_0, y_1) = [z(x, y_0) - z(x, y_1)]/(y_0 - y_1)$$

и т.д.