МГТУ им. Н. Э. Баумана

Отчет по лабораторной работе №2 по предмету «Вычислительные алгоритмы»
«Построение и программная реализация алгоритма многомерной интерполяции табличных функций»

Выполнила Параскун София,
ИУ7-44Б
Проверил
Градов В. М.

Москва, 2021 г.

Цель работы: получение навыков построения алгоритма интерполяции таблично заданных функций двух переменных.

1. Исходные данные

1. Таблица функции с количеством узлов 5x5.

y \ x	0	1	2	3	4
0	0	1	4	9	16
1	1	2	5	10	17
2	4	5	8	13	20
3	9	10	13	18	25
4	16	17	20	25	32

2. Степень аппроксимирующих полиномов — n_x и n_y.

3. Значение аргумента x, y, для которого выполняется интерполяция.

2. Код программы

Программа состоит из 5 файлов, которые собираются в app.exe. Имя файла с входными данными изначально зафиксировано в программе in.txt.

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "in_out.h"
#include "interpolation_func.h"

int main()
{
   int res = OK;
   FILE *f = fopen("in.txt", "r");
   struct polynomial data;
   res = input_data(f, &data);
   fclose(f);

   if (!res)
   {
      find_config(&data);
      calculations(&data);
```

```
    }

    return res;
}
```

in_out.h

```
#include <stdio.h>

#ifndef _IN_OUT_H_
#define _IN_OUT_H_

#define OK 0
#define ERR -1
#define COUNT 5

struct polynomial
{
    float data_f[COUNT * COUNT];
    int nx;
    int ny;
    float x;
    float y;
    int ind_x;
    int ind_y;
    float *difference;
    float result;
};

int input_data(FILE *f, struct polynomial *data);
void out_res(struct polynomial data);

#endif
```

in_out.c

```
#include <stdio.h>

#include <stdlib.h>
#include "in_out.h"

int file_reading(FILE *f, struct polynomial *data);

int input_data(FILE *f, struct polynomial *data)
{
    int res = OK;
```

```
    if (f)
    {
        res = file_reading(f, data);
    }

    if (!res)
    {
        printf("File-data is correct.\n");
        printf("Input nx and ny (integer): ");
        int rc = 0;
        rc = scanf("%d %d", &data->nx, &data->ny);
        if (rc == 2 && data->nx > 0 && data->nx < 4 && data->ny > 0 && data->ny < 4)
        {
            printf("Input polynomal degree x and y: ");
            rc = scanf("%f %f", &data->x, &data->y);
        }
        else
            res = ERR;

        if (rc != 2 || res)
        {
            printf("Wrong data!\n");
            res = ERR;
        }
        else
            printf("Correct data.\n");
    }

    return res;
}

int file_reading(FILE *f, struct polynomial *data)
{
    int res = OK, rc = 1;

    for (int i = 0; i < COUNT && rc; i++)
        for (int j = 0; j < COUNT && rc; j++)
            rc = fscanf(f, "%f", &(data->data_f[i * COUNT + j]));

    if (rc != 1 && feof(f))
    {
        printf("Wrong data!\n");
        res = ERR;
    }

    return res;
```

```
}

void out_res(struct polynomial data)
{
   printf("Interpolation equal %f.", data.result);
}
```

interpolation_func.h

```
#include <stdio.h>

#include "in_out.h"

#ifndef _INTERPOLATION_FUNC_H_
#define _INTERPOLATION_FUNC_H_

#define FIX 100

void find_config(struct polynomial *data);
void calculations(struct polynomial *data);

#endif
```

interpolation_func.c

```
#include <stdio.h>

#include <stdlib.h>
#include "in_out.h"
#include "interpolation_func.h"

void divided_difference_newton(struct polynomial *data, int column);
float find_polynom_newton(struct polynomial *data, int column);

void find_config(struct polynomial *data)
{
   data->ind_x = 0, data->ind_y = 0;
   for (int i = 0; i < COUNT; i++)
   {
      if (data->x >= i && data->x <= i + 1)
         data->ind_x = i;
      if (data->y >= i && data->y <= i + 1)
         data->ind_y = i;
   }

   int step_x = (data->nx + 1) / 2, step_y = (data->ny + 1) / 2;
```

```
if ((data->nx + 1) % 2 == 0)
{
   if (data->ind_x - step_x >= 0)
   {
      if (data->ind_x + step_x <= COUNT - 1)
         data->ind_x -= step_x - 1;
      else
         data->ind_x = COUNT - step_x * 2;
   }
   else
      data->ind_x = 0;
}
else
{
   if (data->ind_x - step_x + 1 >= 0)
   {
      if (data->ind_x + step_x <= COUNT - 1)
         data->ind_x -= step_x;
      else
         data->ind_x = COUNT - 1 - step_x * 2;
   }
   else
      data->ind_x = 0;
}
if ((data->ny + 1) % 2 == 0)
{
   if (data->ind_y - step_y >= 0)
   {
      if (data->ind_y + step_y <= COUNT - 1)
         data->ind_y -= step_y - 1;
      else
         data->ind_y = COUNT - step_y * 2;
   }
   else
      data->ind_y = 0;
}
else
{
   if (data->ind_y - step_y + 1 >= 0)
   {
      if (data->ind_y + step_y <= COUNT - 1)
         data->ind_y -= step_y;
      else
         data->ind_y = COUNT - 1 - step_y * 2;
   }
   else
```

```
        data->ind_y = 0;
    }
    printf("Config x: %d - %d\n", data->ind_x, data->ind_x + data->nx);
    printf("Config y: %d - %d\n", data->ind_y, data->ind_y + data->ny);
}

void calculations(struct polynomial *data)
{
    float dif_x[data->nx + 1];
    for (int i = 0; i < data->nx + 1; i++)
    {
        divided_difference_newton(data, i + data->ind_x);
        dif_x[i] = find_polynom_newton(data, i + data->ind_x);
        printf("%f\n", dif_x[i]);
    }

    data->difference = malloc(sizeof(float) * (data->nx + 1) * (data->nx + 1));
    if (data->difference)
    {
        //вычисление разностей аргументов и первых разделенных разностей
        for (int i = 0; i < data->nx; i++)
        {
            data->difference[i * (data->nx + 1)] = data->x - (i + data->ind_x);
            data->difference[i * (data->nx + 1) + 1] = (dif_x[i] - dif_x[i + 1]) / (-1);
        }

        //вычисление оставшихся разделенных разностей
        for (int j = 2; j < data->nx + 1; j++)
            for (int i = 0; i < data->nx - j + 1; i++)
                data->difference[i * (data->nx + 1) + j] = (data->difference[i * (data->nx + 1) + j - 1] - data-
>difference[(i + 1) * (data->nx + 1) + j - 1]) / (-j);

        data->result = dif_x[0];
        float buf_x = 1;
        for (int i = 1; i < data->nx + 1; i++)
        {
            buf_x *= data->difference[(i - 1) * (data->nx + 1)];
            data->result += buf_x * data->difference[i];
        }
        free(data->difference);
        printf("RES %f\n", data->result);
    }
    else
        printf("Memory allocate error!\n");
}
```

```
void divided_difference_newton(struct polynomial *data, int column)
{
   data->difference = malloc(sizeof(float) * (data->ny + 1) * (data->ny + 1));
   if (data->difference)
   {
      //вычисление разностей аргументов и первых разделенных разностей
      for (int i = 0; i < data->ny; i++)
      {
         data->difference[i * (data->ny + 1)] = data->y - (i + data->ind_y);
         data->difference[i * (data->ny + 1) + 1] = (data->data_f[(i + data->ind_y) * COUNT + column] -
data->data_f[(i + data->ind_y + 1) * COUNT + column]) / (-1);
      }

      //вычисление оставшихся разделенных разностей
      for (int j = 2; j < data->ny + 1; j++)
         for (int i = 0; i < data->ny - j + 1; i++)
            data->difference[i * (data->ny + 1) + j] = (data->difference[i * (data->ny + 1) + j - 1] - data-
>difference[(i + 1) * (data->ny + 1) + j - 1]) / (-j);
   }
   else
      printf("Memory allocate error!\n");
}

float find_polynom_newton(struct polynomial *data, int column)
{
   float res = data->data_f[data->ind_y * COUNT + column];
   float buf_x = 1;
   for (int i = 1; i < data->ny + 1; i++)
   {
      buf_x *= data->difference[(i - 1) * (data->ny + 1)];
      res += buf_x * data->difference[i];
   }
   free(data->difference);
   return res;
}
```

3. Результаты работы

Результатом интерполяции z(x, y) при степенях полиномов 1, 2, 3 для x = y = 1.5.

Степень	$\mathbf{n_x = 1}$	$\mathbf{n_x = 2}$	$\mathbf{n_x = 3}$
$\mathbf{n_y = 1}$	5.00	4.75	4.75
$\mathbf{n_y = 2}$	4.75	4.5	4.5
$\mathbf{n_y = 3}$	4.75	4.5	4.5

4. Вопросы при защите лабораторной работы

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.

При заданных значениях n_x, n_y, x, y конфигурация узлов по x и y будет одинакова — от 1 до 2.
Проведем интерполяцию сначала по строкам (по x):
y = 1: z(x, y) = 3.5
y = 2: z(x, y) = 6.5
Теперь интерполируем по столбцу (по y) при полученных значениях:
z(x, y) = 5

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

Двумерный полином имеет вид $z = a + bx + cy + dx^2 + gy^2 + hxy$. Для данного полинома используется не больше 6 узлов, значит что для 4 и 6 узлов минимальное значение степени полинома равна 2.

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 данной ситуации можно отсортировать данные, попадающие под область конфигурации, и выбирать узлы, максимально приближенные к окрестности точки интерполяции (3 узла для 1 степени, 6 узлов для 2 степени и т.д.). За выбором узлов следует их подстановка в полином и образование системы с последующим выражением требуемых коэффициентов.

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

Сначала необходимо сформировать конфигурацию узлов по третьей переменной, затем провести двумерную интерполяцию для каждого узла третьей переменной и после провести одномерную интерполяцию по полученным в предыдущем шаге значениям.

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

Так как при дальнейших интерполяциях по другим направлениям нас интересует только результат, можно воспользоваться разностепенными полиномами или разными методами интерполяций. Способ получения и/или степени полиномов влияют лишь на точность результата.

6. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.

Данный алгоритм сводится к вычислению разделенных разностей и полинома Ньютона.

$$z(x_0, y_0, y_1) = \frac{z(x_0, y_0) - z(x_0, y_1)}{y_0 - y_1}, z(x_0, x_1, y_0) = \frac{z(x_0, y_0) - z(x_1, y_0)}{x_0 - x_1}$$

Тогда многочлен в данном алгоритме можно представить как обобщение одномерного варианта записи:

$$P_n(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n-1} z(x_0, \ldots, x_i, y_0, \ldots, y_i) \prod_{p=0}^{i-1} (x - x_p) \prod_{q=0}^{j-1} (y - y_q) \ .$$