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 № 2

Тема: Построение и программная реализация алгоритма многомерной интерполяции табличных функций.

Студент Тартыков Л.Е.

Группа ИУ7-44Б

Оценка (баллы) _____

Преподаватель Градов В.М.

2021 г.

Содержание

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

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 - nx и ny .
3. Значение аргументов x, y, для которого выполняется интерполяция.

Код программы

Замечание: данная программа написана на языке Си (стандарт C99).

Листинг 1; main.c

```
#include <stdio.h>
#include "../inc/defines.h"
#include "../inc/struct.h"
#include "../inc/read.h"
#include "../inc/process_matrix.h"
#include "../inc/multiply_interpolation.h"

int main(int argc, char **argv)
{
   int code_error = ERR_OK;
   if (argc != 2)
   {
      printf("Ошибка. Некорректное количество переданных аргументов.\n");
      code_error = ERR_NOT_ENOUGH_ARGS;
   }
   else
   {
      matrix_t *table = NULL;
      if (read_data(argv[1], &table) == ERR_OK)
      {
         double result = 0;
         result = do_multiply_interpolation(&table);
         printf("Результат интерполяции: %lf\n", result);
      }

      free_table(&table);
   }
   return code_error;
}
```

Листинг 2; multiply_interpolation.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "../inc/multiply_interpolation.h"
#include "../inc/defines.h"
#include "../inc/struct.h"
#include "../inc/process_matrix.h"

void get_nodes(double *list_values, int degree, int table_size, double arg, int *index_low,
            int *index_high);
int find_index_near(double *x, int table_size, double arg);
void choose_need_table_place(double **table, double ***need_table, int index_low_x, int index_high_x,
                        int index_low_y, int index_high_y);
void get_f_values(double **f_received_values, double **need_table, double *x, double arg_x,
                int nx, int ny, int index_low, int index_high);
```

```
double interpolate_newton(double *x, double *y, double *matrix, int degree, double arg,
                          int index_low, int index_high);
void fill_newton_splitted_difference(double *matrix, int index_low, int index_high,
                               double step, int degree);


double do_multiply_interpolation(matrix_t **table)
{
   int index_low_x = 0, index_low_y = 0,
      index_high_x = 0, index_high_y = 0;
   double result = 0;

   get_nodes((*table)->x, (*table)->nx, (*table)->column, (*table)->arg_x, &index_low_x, &index_high_x);
   get_nodes((*table)->y, (*table)->ny, (*table)->row, (*table)->arg_y, &index_low_y, &index_high_y);

   double **need_table = NULL;
   choose_need_table_place((*table)->matrix, &need_table, index_low_x, index_high_x,
                           index_low_y, index_high_y);
   if (need_table != NULL)
   {
      double *f_received_values = NULL;
      get_f_values(&f_received_values, need_table, (*table)->x, (*table)->arg_x,
                   (*table)->nx, (*table)->ny, index_low_x, index_high_x);
      if (f_received_values != NULL)
      {
         double diff_matrix[15] = { 0 };
         result = interpolate_newton((*table)->x, f_received_values, diff_matrix, (*table)->ny,
                           (*table)->arg_y, index_low_y, index_high_y);
      }
      free(f_received_values);
   }
   free(need_table);

   return result;
}

void get_nodes(double *list_values, int degree, int table_size, double arg, int *index_low,
               int *index_high)
{
   //найти ближайшую точку
   int index_near = 0;
   index_near = find_index_near(list_values, table_size, arg);

   int need_space = degree / 2;

   if (index_near + need_space + 1 > table_size) //если не хватает снизу узлов
   {
      *index_low = table_size - 1 - degree;
      *index_high = table_size - 1;
   }
   else if (index_near < need_space) //если не хватает узлов сверху
```

```
    {
        *index_low = 0;
        *index_high = *index_low + degree;
    }
    else
    {
        *index_low = index_near - need_space;
        *index_high = *index_low + degree;
    }
}

int find_index_near(double *x, int table_size, double arg)
{
    int index_near = 0;
    double min_diff = fabs(x[0] - arg);

    for (int i = 1; i < table_size; i++)
    {
        if (fabs(x[i] - arg) < min_diff && fabs(fabs(x[i] - arg) - min_diff) >= EPS)
        {
            index_near = i;
            min_diff = fabs(x[i] - arg);
        }
    }

    return index_near;
}

void choose_need_table_place(double **table, double ***need_table, int index_low_x, int index_high_x,
                             int index_low_y, int index_high_y)
{
    *need_table = allocate_matrix(index_high_y - index_low_y + 1, index_high_x - index_low_x + 1);
    if (*need_table != NULL)
    {
        for (int i = index_low_y, k = 0; i <= index_high_y; i++, k++)
            for (int j = index_low_x, l = 0; j <= index_high_x; j++, l++)
                (*need_table)[k][l] = table[i][j];
    }
}

void get_f_values(double **f_received_values, double **need_table, double *x, double arg_x,
                  int nx, int ny, int index_low, int index_high)
{
    *f_received_values = malloc((ny + 1) * sizeof(double));
    if (f_received_values != NULL)
    {
        double diff_matrix[15] = { 0 };
        for (int i = 0; i < ny + 1; i++)
        {
            (*f_received_values)[i] = interpolate_newton(x, need_table[i], diff_matrix, nx, arg_x,
                                      index_low, index_high);
```

```
        }
    }
}

double interpolate_newton(double *x, double *y, double *matrix, int degree, double arg,
                          int index_low, int index_high)
{
    double result = 0, temp = 1;

    for (int i = 0, j = 0; i <= index_high - index_low; i++, j++)
        matrix[j] = y[i];

    fill_newton_splitted_difference(matrix, index_low, index_high, x[1] - x[0], degree);

    for (int i = 0, k = 0, p = degree + 1; i <= degree; i++, k += p, p--)
    {
        result += matrix[k] * temp;
        temp *= (arg - x[index_low + i]);
    }

    return result;
}

void fill_newton_splitted_difference(double *matrix, int index_low, int index_high,
                                     double step, int degree)
{
    int k = 1, koeff = 1, p = 0, shift = 1;
    for (int i = index_high - index_low; i > 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            matrix[degree + p + 1] = (matrix[shift] - matrix[shift - 1]) / (step * koeff);
            shift++, p++;
        }
        koeff++, k++, shift++;
    }
}
```

Листинг 3; process_matrix.c

```
#include <stdio.h>
#include <stdlib.h>
#include "../inc/process_matrix.h"
#include "../inc/struct.h"

double **allocate_matrix(int row, int column)
{
    double **table = NULL;
    table = malloc(row * sizeof(double*) + row * column * sizeof(double));
    if (table != NULL)
    {
```

```
        for (int i = 0; i < row; i++)
        {
            table[i] = (double*)((char*)table + row * sizeof(int*) + i * column * sizeof(double));
        }
    }

    return table;
}

void free_table(matrix_t **table)
{
    free((*table)->matrix);
    free((*table)->x);
    free((*table)->y);
    free(*table);
}

void print_matrix(double **matrix, int row, int column)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < column; j++)
            printf("%lf ", matrix[i][j]);
        printf("\n");
    }
}
```

Листинг 4; read.c

```
#include <stdio.h>
#include <stdlib.h>
#include "../inc/read.h"
#include "../inc/struct.h"
#include "../inc/defines.h"
#include "../inc/process_matrix.h"

void read_powers(int *nx, int *ny);
void read_args(double *x, double *y);
int fill_table_from_file(FILE *file, double ***table, int row, int column);

int read_data(char *file_name, matrix_t **table)
{
    int code_error = ERR_OK;
    FILE *file = fopen(file_name, "r");
    if (file != NULL)
    {
        *table = malloc(sizeof(matrix_t));
        if (fscanf(file, "%d%d", &(*table)->row, &(*table)->column) != 2)
        {
            printf("Неверный размер таблицы.\n");
            code_error = ERR_INCORRECT_SIZE_TABLE;
        }
```

```
        else
        {
            (*table)->x = malloc((*table)->row * sizeof(double));
            (*table)->y = malloc((*table)->column * sizeof(double));
            for (int i = 0; i < (*table)->row; i++)
                (*table)->x[i] = i, (*table)->y[i] = i;
            fill_table_from_file(file, &(*table)->matrix, (*table)->row, (*table)->column);
        }

        if (code_error == ERR_OK)
        {
            read_powers(&(*table)->nx, &(*table)->ny);
            read_args(&(*table)->arg_x, &(*table)->arg_y);
        }
        fclose(file);
    }
    else
        code_error = ERR_NO_FILE;

    return code_error;
}

int fill_table_from_file(FILE *file, double ***table, int row, int column)
{
    int code_error = ERR_OK;
    *table = allocate_matrix(row, column);
    if (table != NULL)
    {
        for (int i = 0; code_error == ERR_OK && i < row; i++)
            for (int j = 0; code_error == ERR_OK && j < column; j++)
                if (fscanf(file, "%lf", &(*table)[i][j]) != 1)
                    code_error = ERR_INCORRECT_ELEM_TABLE;
    }
    print_matrix(*table, row);
    return code_error;
}

void read_powers(int *nx, int *ny)
{
    printf("Введите степени апроксимирующих полиномов (nx, ny): ");
    while (scanf("%d%d", nx, ny) != 2)
    {
        fflush(stdin);
        printf("Ошибка: некорректные значения.\n\n");
        printf("Введите степени апроксимирующих полиномов (nx, ny): ");
    }
}

void read_args(double *x, double *y)
{
    printf("Введите аргументы x и y через пробел: ");
```

```
    while(scanf("%lf%lf", x, y) != 2)
    {
        fflush(stdin);
        printf("Ошибка: некорректные значения.\n\n");
        printf("Введите аргументы x и y через пробел: ");
    }
}
```

Листинг 5; defines.h

```
#ifndef _DEFINES_H_
#define _DEFINES_H_

#define ERR_OK 0
#define ERR_NOT_ENOUGH_ARGS 1
#define ERR_INCORRECT_SIZE_TABLE 2
#define ERR_INCORRECT_ELEM_TABLE 3
#define ERR_NO_FILE 4
#define EPS 1e-6

#endif
```

Листинг 6; multiply_interpolation.h

```
#ifndef _MULTIPLY_INTERPOLATION_H_
#define _MULTIPLY_INTERPOLATION_H_

#include "../inc/struct.h"
double do_multiply_interpolation(matrix_t **table);

#endif
```

Листинг 7; process_matrix.h

```
#ifndef _PROCESS_MATRIX_H_
#define _PROCESS_MATRIX_H_

#include "../inc/struct.h"

double **allocate_matrix(int row, int column);
void free_table(matrix_t **table);
void print_matrix(double **matrix, int row, int column);

#endif
```

Листинг 8; read.h

```
#ifndef _READ_H_
#define _READ_H_

#include "../inc/struct.h"
```

```
int read_data(char *file_name, matrix_t **table);

#endif
```

Листинг 9; struct.h

```
#ifndef _STRUCT_H_
#define _STRUCT_H_

typedef struct matrix matrix_t;
struct matrix
{
   double **matrix;
   int row, column;
   double *x, *y;
   int nx, ny;
   double arg_x, arg_y;
};

#endif
```

Результаты работы

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

nx	ny	z(x,y)
1	1	5
1	2	4,75
1	3	4,75
2	1	4,75
2	2	4,5
2	3	4,5
3	1	4,75
3	2	4,5
3	3	4,5

Ответы на 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.

Значения, полученные в ходе последовательной интерполяции:

По первой строке: 3,5

По второй строке: 6,5

По столбцу: 5,0

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

В обоих случаях минимально возможная степень полинома — нуль. В таком случае можно использовать всего лишь один узел и получить какое-то число в зависимости от $f(x_0, y_0)$.

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?

При хаотичном расположении узлов получается некоторая «плоскость» из этих точек. Эту плоскость разбиваем на «треугольники» и интерполируем уже внутри них. Главное ограничение при таком разбиении — узлы не должны лежать на одной прямой.

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

Принцип построения трехмерной интерполяции схож тому, как строится двумерная интерполяция: двумерная интерполяция строится через использование одномерной, так и трехмерная строится через двумерную.

Пусть исходная функция f имеет три параметра x,y,z => f(x,y,z). Выполняем двумерную интерполяцию сначала для параметров (x, y) nz раз (nz- степень полинома z). После этого выполняется интерполяция для значения z. Получаем искомое значение.

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. Не важно, в какой последовательности будет выполняться интерполяция (сначала по одному параметру, затем — по другому, или наоборот; выполняется один метод интерполяции или другой) — результат от этого не изменится. Главное, не отклоняться от такой последовательности выполнения интерполяции.

6. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.

Принцип выполнения интерполяции на треугольной конфигурации узлов аналогичен интерполяции на прямоугольной конфигурации. Отличие заключается только в ограничении выполнения одномерной интерполяции (будет выполняется условие n-i-1 вместо n-1, которое определяет диагональ этой матрицы).

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