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

**Студент** Сучкова Т.М.
**Группа** ИУ7-42Б
**Оценка (баллы)** _____
**Преподаватель** Градов В.М.

*Москва*
*2021 г.*

**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 – **n_x** и **n_y**.

3. Значение аргументов **x, y**, для которого выполняется интерполяция.

## Алгоритм

На регулярной сетке последовательно проводим интерполяцию полиномами Ньютона: сначала по строкам, затем по столбцам.

## Код программы

**main.c:**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define OK 0
#define ERR -1

typedef struct point
{
   double x;
   double y;
} point_t;

double func(double x, double y)
{
   return x * x + y * y;
}
```

```
void matrix_free(double **matrix, int n)
{
   for (int i = 0; i < n; i++)
   {
      free(matrix[i]);
   }
   free(matrix);
}
int matrix_allocate(double ***matrix, int n)
{
   *matrix = calloc(n, sizeof(double *));
   if (!*matrix)
   {
      return ERR;
   }

   for (int i = 0; i < n; i++)
   {
      (*matrix)[i] = malloc(n * sizeof(double));
      if (!(*matrix)[i])
      {
         matrix_free(*matrix, i);
         return ERR;
      }
   }

   return OK;
}

// Initial values table
int initial_table_create(FILE *f, point_t **table, double ***z, int n)
{
   *table = malloc(n * sizeof(point_t));
   if (!*table)
   {
      return ERR;
   }

   if (matrix_allocate(z, n) == ERR)
   {
      free(*table);
      return ERR;
   }

   for (int i = 0; i < n; i++)
   {
      //printf("Input %d point data (x y): ", i + 1);
      fscanf(f, "%lf", &((*table)[i].x));
   }
   fscanf(f, "\n");

   for (int i = 0; i < n; i++)
```

```
    {
        //printf("Input %d point data (x y): ", i + 1);
        fscanf(f, "%lf",  &((*table)[i].y));

        for (int j = 0; j < n; j++)
        {
            fscanf(f, "%lf", &((*z)[i][j]));
        }
        fscanf(f, "\n");
    }

    return OK;
}

void initial_table_print(FILE *f, point_t *table, double **z, int n)
{
   fprintf(stdout, "%9s", "y\\x");

    for (int i = 0; i < n; i++)
    {
        //printf("Input %d point data (x y): ", i + 1);
        fprintf(f, "%9.2lf", table[i].x);
    }
    fprintf(f, "\n");

    for (int i = 0; i < n; i++)
    {
        //printf("Input %d point data (x y): ", i + 1);
        fprintf(f, "%9.2lf",  table[i].y);

        for (int j = 0; j < n; j++)
        {
            fprintf(f, "%9.2lf", z[i][j]);
        }
        fprintf(f, "\n");
    }
}

int min_point_i(point_t *points, int n_init, double x)
{
    int min_i = 0, min;
    int flag = 0;

    for (int i = 0; i < n_init; i++)
    {
        if (!flag)
        {
            min = abs(points[i].x - x);
            flag = 1;
        }
        else if (abs(points[i].x - x) < min)
        {
```

```
            min = abs(points[i].x - x);
            min_i = i;
        }
    }

    return min_i;
}

// Points selection
point_t *choose_points(point_t *points, double *z, int n_init, double x, int n, int flag)
{
    point_t *selected_points = malloc(n * sizeof(point_t));
    if (!selected_points)
    {
        return NULL;
    }

    int i_beg, i_flag = 0;
    int n_required = ceil(n / 2);
    int i_near = min_point_i(points, n_init, x);

    if (i_near + n_required + 1 > n_init)
    {
        i_beg = n_init - n;
    }
    else if (i_near < n_required)
    {
        i_beg = 0;
    }
    else
    {
        i_beg = i_near - n_required + 1;
    }

    if (flag)
    {
        i_flag = i_beg;
    }
    for (int i = 0; i < n; i++)
    {
        selected_points[i].x = points[i_beg + i].x;
        selected_points[i].y = z[i_beg - i_flag + i];
    }

    return selected_points;
}

void arr_print(double *arr, char *s, int n)
{
    printf("\n%s\n", s);

    for (int i = 0; i < n; i++)
```

```
    {
        printf(" %lf ", arr[i]);
    }
}

// Divided difference matrix
int dif_matrix_allocate(double ***dif_matrix, int n)
{
    *dif_matrix = calloc(n, sizeof(double *));
    if (!*dif_matrix)
    {
        return ERR;
    }

    for (int i = 0; i < n; i++)
    {
        (*dif_matrix)[i] = malloc((n - i) * sizeof(double));
        if (!(*dif_matrix)[i])
        {
            matrix_free(*dif_matrix, i);
            return ERR;
        }
    }

    return OK;
}

void arr_null(double *arr, int n)
{
    for (int  i = 0; i < n; i++)
    {
        arr[i] = 0;
    }
}

void arr_copy(double *destination, double *source, int n)
{
    for (int i = 0; i < n; i++)
    {
        destination[i] = source[i];
    }
}

int dif_matrix_fill(double **dif_matrix, point_t *points, int n)
{
    double *tmp_arr = malloc(n *sizeof(double));
    if (!tmp_arr)
    {
        return ERR;
    }

    int cur_count;
```

```
    for (int i = 0; i < n; i++)
    {
        arr_null(tmp_arr, n);

        for (int j = 0; j < n - i; j++)
        {
            if (i == 0)
            {
                tmp_arr[j] = (points[j].y - points[j + 1].y) / (points[j].x - points[j + i + 1].x);

            }
            else
            {

                tmp_arr[j] = (dif_matrix[i - 1][j] - dif_matrix[i - 1][j + 1]) / \
                             (points[j].x - points[j + i + 1].x);

            }
        }
        cur_count = n - i - 1;

        arr_copy(dif_matrix[i], tmp_arr, cur_count);
        arr_print(dif_matrix[i], "DIF", cur_count);
    }

    free(tmp_arr);

    return OK;
}

int newton_interpolation(point_t *points, double *z, int n_init, \
                                 double x, int n, double *result, int flag)
{
    point_t *selected_points = calloc(n + 1, sizeof(point_t));
    selected_points = choose_points(points, z, n_init, x, n + 1, flag);
    if (!selected_points)
    {
        return ERR;
    }

    double **dif_matrix;
    dif_matrix_allocate(&dif_matrix, n);
    dif_matrix_fill(dif_matrix, selected_points, n + 1);

    double cur_k = 1;

    for (int i = 0; i < n + 1; i++)
    {
        if (i == 0)
        {
            *result += selected_points[i].y;
```

```
        }
        else
        {
            *result += cur_k * dif_matrix[i - 1][0];
        }

        cur_k *= x - selected_points[i].x;

        //arr_print(dif_matrix[i], "dif_matrix cur_i", n - i);
    }

    free(selected_points);
    matrix_free(dif_matrix, n);

    return OK;
}

void change_y_x(point_t *points, int n_init)
{
    double tmp;

    for (int i = 0; i < n_init; i++)
    {
        tmp = points[i].x;
        points[i].x = points[i].y;
        points[i].y = tmp;
    }
}

int interpolation(point_t *points, double **z, int n, \
                  double x, double y, int n_x, int n_y, double *result)
{
    double *cur_res = calloc((n_y + 1), sizeof(double));

    for (int i = 0; i < n_y + 1; i++)
    {
        if (newton_interpolation(points, z[i], n, x, n_y, &(cur_res[i]), 0) != OK)
        {
            free(points);
            return ERR;
        }
    }

    change_y_x(points, n);
    if (newton_interpolation(points, cur_res, n, y, n_x, result, 1) != OK)
    {
        free(points);
        return ERR;
    }

    free(cur_res);
```

```
    return OK;
}

int cmp_table_print(point_t *initial_table, double **z, \
                            int n_init, double x, double y, int n_max)
{
    double cur_res = 0.0;

    printf("\n|%10s |%19s |\n", "n_x = n_y", "Intorpolated value");

    for (int i = 1; i < n_max + 1; i++)
    {
        cur_res = 0.0;

        if (interpolation(initial_table, z, n_init, x, y, i, i, &cur_res) != OK)
        {
            free(initial_table);
            return ERR;
        }

        printf("|%10d |", i);
        printf("%20lf|\n", cur_res);
    }

    return OK;
}

int main(int argc, char **argv)
{
    int n_init = 0;
    setbuf(stdout, NULL);

    if (argc != 2)
    {
        return ERR;
    }

    printf("\nINITIAL TABLE DATA\n");

    FILE *f = fopen(argv[1], "r");
    if (!f)
    {
        return ERR;
    }

    if (fscanf(f, "%d", &n_init) != 1 || n_init < 1)
    {
        return ERR;
    }
    fscanf(f, "\n");

    point_t *initial_table;
```

```
    double **z = NULL;

    if (initial_table_create(f, &initial_table, &z, n_init) != OK)
    {
        return ERR;
    }

    fclose(f);

    printf("\n");
    initial_table_print(stdout, initial_table, z, n_init);

    int n_x, n_y;
    double x, y;

    printf("\nInput n_x n_y: ");
    if (scanf("%d%d", &n_x, &n_y) != 2 || n_x < 0 || n_y < 0)
    {
        free(initial_table);
        matrix_free(z, n_init);
        return ERR;
    }

    printf("Input x y: ");
    if (scanf("%lf%lf", &x, &y) != 2)
    {
        free(initial_table);
        matrix_free(z, n_init);
        return ERR;
    }

    cmp_table_print(initial_table, z, n_init, x, y, n_y);

    printf("\nz(x, y)     : %lf", func(x, y));

    matrix_free(z, n_init);
    free(initial_table);

    return OK;
}
```

## Результаты работы программы.

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

```
INITIAL TABLE DATA

      y\x     0.00     1.00     2.00     3.00     4.00
     0.00     0.00     1.00     4.00     9.00    16.00
     1.00     1.00     2.00     5.00    10.00    17.00
     2.00     4.00     5.00     8.00    13.00    20.00
     3.00     9.00    10.00    13.00    18.00    25.00
     4.00    16.00    17.00    20.00    25.00    32.00

Input n_x n_y: 3 3
Input x y: 1.5 1.5
```

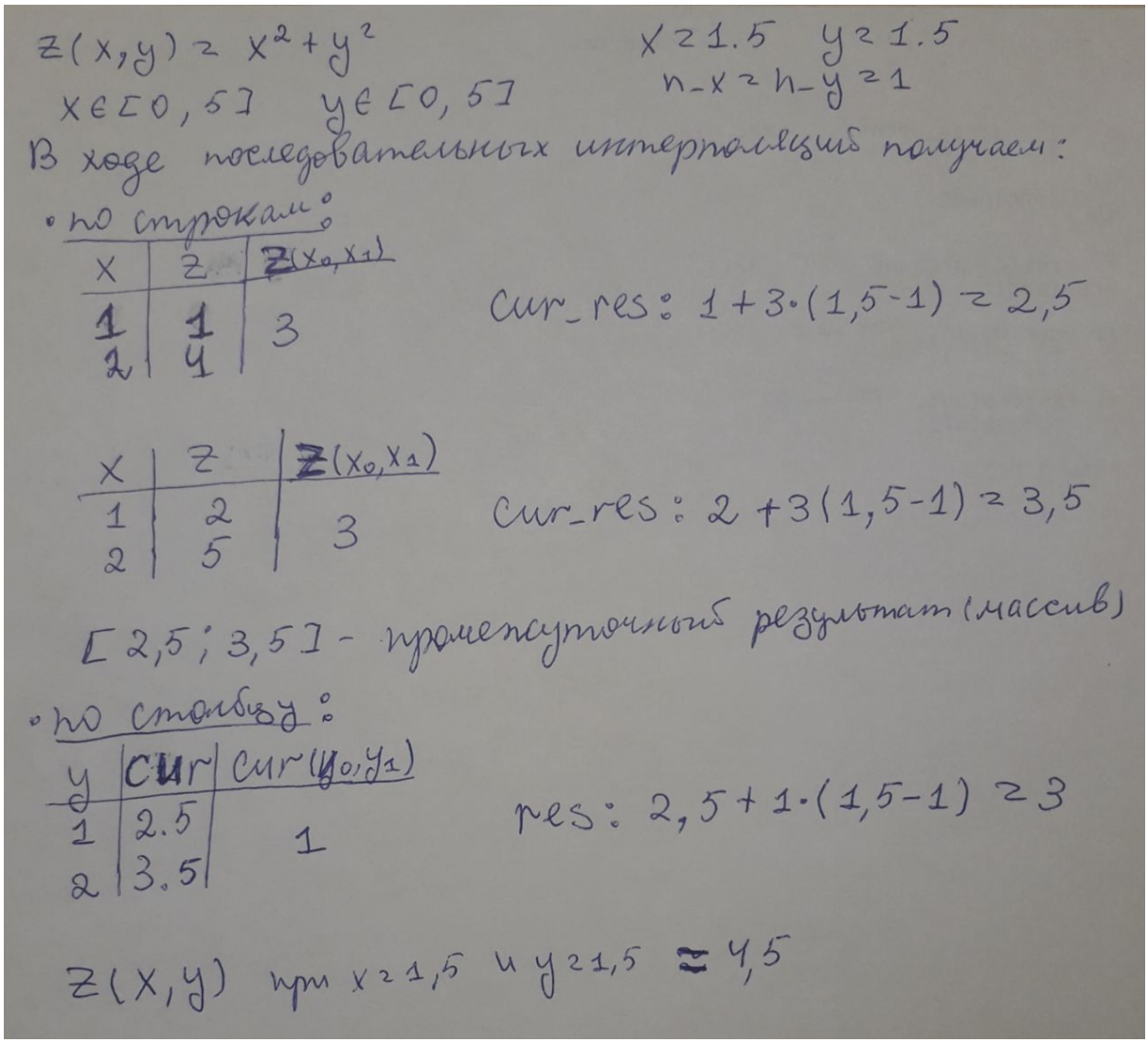
```
| n_x = n_y | Intorpolated value |
|         1 |              1.750000|
|         2 |              5.906250|
|         3 |              4.500000|

z(x,y)      : 4.500000
```

## Вопросы при защите лабораторной работы.

Ответы на вопросы дать письменно в Отчете о лабораторной работе.

**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 nx = ny =1, x=y=1.5. Приведите по шагам те. значения функции, которые получаются в ходе последовательных интерполяций. по строкам и столбцу**




Можем видеть, что полученное значение при степени **nx = ny =1** вычислено неточно, причиной чему стал выбор малой степени.

Точное значение:
```
z(x, y): 4.500000
```

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

На 4 узлах степень полинома **nx=ny** может находиться в промежутке **[0; 3]**.
На 6 узлах степень полинома **nx=ny** может находиться в промежутке **[0; 5]**.

**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 использовании нерегулярной сетки, ограничиваясь интерполяционным полиномом первой степени, имеем **z=a+bx+cy**, и его коэффициенты находят по трем узлам, выбираемым в окрестности точки интерполяции:

$$z_i = a + bx_i + cy_i,\ 0 \le i \le 2,$$ здесь i - номер узла.

Точно так же можно использовать полином второй степени

$$z_i = a + bx_i + cy_i + dx_i^2 + gy_i^2 + hx_iy_i,\ 0 \le i \le 5.$$

Понятно, что выбираются 6 узлов, ближайших к точке интерполяции.

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

Пусть дана функция трех переменных **t = f(x,y,z)**. Для проведения интерполяции по трем переменным воспользуемся алгоритмом двумерной интерполяции.

Выбираем начальное направление, например, по x. Проводим по нему **n_yz + 1** одномерных интерполяций при выбранных значениях **yz[k], k = 0,1,..,n**. Вычисляем значения функции **f(x,yz[k]), k = 0,1,...,n**. По этим значениям функции, которые теперь привязаны к **yz[k]**, проводим одну интерполяцию по **yz**.

Таким образом, 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?**

При последовательной интерполяции по разным направлениям можно использовать полиномы несовпадающих степеней, если речь идет, например, о двумерной интерполяции. Тогда если z = f(x, y) и при этом количество значений x и y не совпадает (т. е. таблица является прямоугольной, а не квадратной), степени **n_x** и **n_y** будут различны.

Использовать различные методы интерполяции для разных направлений так же допустимо из-за того, что последовательная интерполяция в каждом из отдельных направлений является одномерной, а то, что должно быть найдено в ее результате, от метода интерполяции не зависит.

**6. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.**

При многомерной интерполяции расположение узлов не может быть произвольным. Например, при интерполяции полиномом первой степени $P_1(x,y)$ узлы не должны лежать на одной прямой в плоскости. Проверять условия подобного типа достаточно сложно, поэтому на практике целесообразно строить регулярные сетки, как правило, прямоугольные и равномерные, когда узлы являются точками пересечения двух взаимно перпендикулярных систем параллельных прямых. На этой сетке проводят простую последовательную интерполяцию: с**начала по строкам, а затем по столбцам.**

Покажем, как строится алгоритм на примере интерполяции двумерной табличной функции **z = f(x, y)**. Задаются степени интерполяционных полиномов по двум координатам x y **n_x ,n_y** и значения аргументов **x, y** . Вначале проводится интерполяцию, например, по x. При этом выполняется **n_y + 1** одномерных интерполяций при выбранных значениях **y_j, j = 0,1,…n_y**, и вычисляются значения функции **f(x, y_j), j=0,1,...n_y**. А затем по полученным значениям функции, привязанным теперь к **y_j**, совершается одна интерполяция по **y**.
При треугольной конфигурации расположения узлов степень многочлена будет минимальной. Многочлен n-й степени в форме Ньютона для двумерной интерполяции в этом случае можно представить как обобщение одномерного варианта записи:

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