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 № 1

Тема: Построение и программная реализация алгоритма полиномиальной 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 полиномами Ньютона и Эрмита.

Исходные данные

1. Таблица функции и её производных

x	y	y’
0.00	1.000000	-1.000000
0.15	0.838771	-1.14944
0.30	0.655336	-1.29552
0.45	0.450447	-1.43497
0.60	0.225336	-1.56464
0.75	-0.018310	-1.68164
0.90	-0.278390	-1.78333
1.05	-0.552430	-1.86742

2. Степень аппроксимирующего полинома - n.

3. Значение аргумента, для которого выполняется интерполяция.

Код программы

Замечание: данная программа написана на языке Си (стандарт C99)
Код данной программы представлен на листингах 1 — 5

Листинг 1; main.c

```
#include <stdio.h>
#include "../inc/defines.h"
#include "../inc/read.h"
#include "../inc/struct.h"
#include "../inc/io.h"
#include "../inc/interpolation.h"

int main(int argc, char **argv)
{
    int code_error = ERR_OK;
    int table_size = 0;
    double arg = 0;
    double matrix[15] = { 0 };
    double matrix_ermite[60] = { 0 };
    table_t *table = NULL;

    read_table(&table, argv[1], &table_size);
    print_table(table, table_size);

    read_interpol_argument(&arg);
    printf("Введенный аргумент = %lf\n", arg);

    printf("Результирующая таблица:\n"
           "|    Степень полинома    |    Полином Ньютона    |    Полином Эрмита    |\n"
           "|------------------------|-----------------------|----------------------|\n");
    for (int i = 1; i < 5; i++)
    {
        double res_newton = 0, res_ermite = 0;
        int index_low = 0, index_high = 0;
        get_nodes(table->x, i, table_size, arg, &index_low, &index_high);

        //интерполяция функции (ньютон)
        res_newton = newton_interpolate(table->x, table->y, matrix, table_size, i, arg, index_low, index_high);

        int index_low_ermite = 0, index_high_ermite = 0;
        int need_degree = 0;
        need_degree = (int)((i + 1) % 2 != 0 ? (i + 1) / 2 + 1 : (i + 1) / 2);
        get_nodes(table->x, need_degree - 1, table_size, arg, &index_low_ermite, &index_high_ermite);

        res_ermite = create_ermite_polynome(table, matrix_ermite, i, arg, index_low_ermite, index_high_ermite,
                            table->x[index_low_ermite + 1] - table->x[index_low_ermite]);
        printf("|%26d|%25lf|%23lf|\n", i, res_newton, res_ermite);
    }

    //найти корень функции - обратная интерполяция
    int index_low = 0, index_high = 0;
    double matrix_new[15] = { 0 }, result = 0;
    insert_sort_array(&table->y, &table->x, table_size);
    get_nodes(table->y, 4, table_size, arg, &index_low, &index_high);

    result = newton_interpolate(table->y, table->x, matrix_new, table_size, 4, ROOT, index_low, index_high);
    printf("Результат обратной интерполяции = %lf\n", result);

    return code_error;
}
```

Листинг 2; io.c

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

void print_table(table_t *table, int table_size)
{
   printf("Исходная таблица:\n"
          "|     x     |     y     |     dy    |\n"
          "|-----------|-----------|-----------|\n");
   for (int i = 0; i < table_size; i++)
   {
      printf("|%11lf|%11lf|%11lf|\n", table->x[i], table->y[i], table->dy[i]);
   }
}
```

```
#include <stdio.h>
#include <math.h>
#include "../inc/interpolation.h"
#include "../inc/struct.h"
#include "../inc/defines.h"

int find_index_near(double *x, int table_size, double arg);
void fill_newton_splitted_difference(double *matrix, int index_low, int index_high,
                                     double step, int degree);

void fill_ermite_split_difference(double *x, double *y, double *dy, double *matrix_ermite, int index_low_ermite, int
index_high_ermite,
                                  double shift);
void swap_elem(double *first, double *second);

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

   for (int i = index_low, j = 0; i <= index_high; 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;
}
```

Листинг 3; interpolation.c

```
void get_nodes(double *x, 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(x, table_size, arg);

   int need_space = degree / 2;

   if (need_space + index_near + 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 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++;
   }
}



double create_ermite_polynome(table_t *table, double *matrix_ermite, int degree, double arg, int index_low_ermite, int index_high_ermite,
                        double shift)
{
   fill_ermite_split_difference((*table).x, (*table).y, (*table).dy, matrix_ermite, index_low_ermite, index_high_ermite, shift);
   double result = 0, temp = 1;
   for (int i = 0, k = 0, p = (index_high_ermite - index_low_ermite + 1)  * 2 + 1, t = 0;
         i < (index_high_ermite - index_low_ermite + 1)  * 2; i++, k += p - 1, p--)
   {
      result += matrix_ermite[k] * temp;
      temp *= (arg - (*table).x[index_low_ermite + t]);
      if (i % 2 != 0)
         t++;
   }
   return result;
}

void fill_ermite_split_difference(double *x, double *y, double *dy, double *matrix_ermite, int index_low_ermite, int index_high_ermite,
                              double shift)
{
   int step = (index_high_ermite - index_low_ermite + 1)  * 2, temp_step = step;
   //заполняем повторяющиеся столбцы
   for (int i = 0, j = 0; i <= step; i += 2, j++)
   {
      matrix_ermite[i] = y[index_low_ermite + j];
      matrix_ermite[i + 1] = y[index_low_ermite + j];
   }
   temp_step--;

   //заполняем колонку с производными
   for (int i = step, p = 0, t = 0; i > 1; i--, p++)
   {
      if (p % 2 == 0)
      {
         matrix_ermite[step + p] = dy[t + index_low_ermite];
         t++;
      }
      else
         matrix_ermite[step + p] = (matrix_ermite[p + 1] - matrix_ermite[p]) / (shift);
   }
   step += temp_step - 1, temp_step--;

   int k = 1, koeff = 1, p = 0, new_shift = 0;
   for (int i = temp_step, step_2 = (index_high_ermite - index_low_ermite + 1) * 2, t = 1; i > 0; i--, step_2++)
   {
      for (int j = 0; j < i; j++)
      {
```

```
            matrix_ermite[step + p + 1] = (matrix_ermite[step_2 + new_shift + 1] - matrix_ermite[step_2 + new_shift]) /
(x[index_low_ermite + t] - x[index_low_ermite]);
            new_shift++, p++;
        }
        if (i % 2 != 0)
            t++;
        koeff++, k++;
    }
}

void insert_sort_array(double **array,  double **array_1, int size_array)
{
    double item = 0, item_1 = 0;
    for (int i = 1, j = 0; i < size_array; i++)
    {
        item = (*array)[i], item_1 = (*array_1)[i];
        j = i - 1;
        while (j >= 0 && (*array)[j] > item)
        {
            (*array)[j + 1] = (*array)[j];
            (*array_1)[j + 1] = (*array_1)[j];
            j -= 1;
        }
        (*array)[j + 1] = item;
        (*array_1)[j + 1] = item_1;
    }
}
```

Листинг 4; read.c

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

void init_table(table_t **table, int table_size);

void read_degree(int *degree)
{
    printf("Введите степень полинома (от 1 до 4): ");
    while (fscanf(stdin, "%d", degree) != 1 || *degree <= 0 || *degree >= 5)
    {
        fflush(stdin);
        printf("Некорректное значение!\n"
               "Введите степень полинома (от 1 до 4): ");
    }
}

int read_table(table_t **table, char *table_name, int *table_size)
{
    int code_error = ERR_OK;
    FILE *file = fopen(table_name, "r");
    if (file != NULL)
```

```
    {
        if (fscanf(file, "%d\n", table_size) != 1 || *table_size <= 0)
        {
            printf("Некорректный размер таблицы!\n");
        }
        else
        {
            *table = NULL;
            *table = malloc(sizeof(table_t));
            init_table(table, *table_size);
            for (int i = 0; code_error == ERR_OK && feof(file) == 0; i++)
            {
                if (fscanf(file, "%lf%lf%lf\n", &(*table)->x[i], &(*table)->y[i], &(*table)->dy[i]) != 3)
                    code_error = ERR_NOT_READ_VALUES;
            }
        }
    }
    else
        code_error = ERR_NOT_CORRECT_FILE;

    return code_error;
}

void init_table(table_t **table, int table_size)
{
    (*table)->x = NULL, (*table)->y = NULL, (*table)->dy = NULL;
    (*table)->x = malloc(table_size * sizeof(double));
    (*table)->y = malloc(table_size * sizeof(double));
    (*table)->dy = malloc(table_size * sizeof(double));
}

void read_interpol_argument(double *arg)
{
    printf("Введите значение аргумента, для которого выполняется интерполяция: ");
    while (scanf("%lf", arg) != 1)
    {
        fflush(stdin);
        printf("Некорректное значение!\n"
               "Введите значение аргумента, для которого выполняется интерполяция: ");
    }
}
```

Листинг 5; defines.h

```
#ifndef _DEFINES_H_
#define _DEFINES_H_

#define ERR_OK 0
#define ERR_NOT_CORRECT_X 1
#define ERR_NOT_CORRECT_FILE 2
#define ERR_NOT_READ_VALUES 3
#define EPS 1e-6
#define ROOT 0.000000

#endif
```

Листинг 6; interpolation.h

```
#ifndef _INTERPOLATION_H_
#define _INTERPOLATION_H_

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

double newton_interpolate(double *x, double *y, double *matrix, int table_size, int degree, double arg, int index_low, int
index_high);
void get_nodes(double *x, int degree, int table_size, double arg, int *index_low, int *index_high);
double create_ermite_polynome(table_t *table, double *matrix_ermite, int degree, double arg, int index_low_ermite, int
index_high_ermite,
                        double shift);

void insert_sort_array(double **array,  double **array_1, int size_array);

#endif
```

Листинг 7; io.h

```
#ifndef _IO_H_
#define _IO_H_

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

void print_table(table_t *table, int table_size);

#endif
```

Листинг 8; read.h

```
#ifndef _READ_H_
#define _READ_H_

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

void read_degree(int *degree);
int read_table(table_t **table, char *table_name, int *table_size);
void read_interpol_argument(double *arg);

#endif
```

Листинг 9; struct.h

```
#ifndef _STRUCT_H_
#define _STRUCT_H_

typedef struct table table_t;
struct table
{
   double *x, *y, *dy;
};

#endif
```

Результаты работы

1. **Значения y(x) при степенях полиномов Ньютона и Эрмита n=1, 2, 3 и 4 при фиксированном x, например, x=0.525 (середина интервала 0.45-0.60). Результаты свести в таблицу для сравнения полиномов.**

Результирующая таблица (при значении x = 0.525):

Степень полинома	Полином Ньютона	Полином Эрмита
1	0.337891	0.342824
2	0.340208	0.340288
3	0.340314	0.340323
4	0.340324	0.340427

2. **Найти корень заданной выше табличной функции с помощью обратной интерполяции, используя полином Ньютона.**
Методом обратной интерполяции найденный корень при степени полинома n = 4
x = 0.739978

Ответы на вопросы при защите лабораторной работы.

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

При вычислении методом полинома Ньютона для степени n = 0 требуется 1 узел. Для полинома Эрмита также требуется 1 узел (по формуле (n + 1) / 2 получается 1 узел (если не делится, берем большее число узлов)). Функция вернет значение, которое находится ближе к итоговому результату, но имеет большую погрешность по сравнению с другими, высшими, степенями полиномов (характерно для полинома Ньютона).

Степень полинома	Полином Ньютона	Полином Эрмита
0	0.450447	0.342824
1	0.337891	0.342824
2	0.340208	0.340288
3	0.340314	0.340323
4	0.340324	0.340427

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

Для практической оценки погрешности интерполяции можно воспользоваться при помощи оценки первого отброшенного члена в полиноме Ньютона.

Трудность использования указанных теоретических оценок состоит в том, что производные интерполируемой функции обычно неизвестны (а также возникают дополнительные трудности при вычислении теоретической оценки при помощи формулы).

$$\left|y(x)-P_n(x)\right| \le \frac{M_{n+1}}{(n+1)!}\left|\varpi_n(x)\right|,$$

где $M_{n+1}=max\left|y^{(n+1)}(\xi)\right|$ - максимальное значение производной интерполируемой функции на отрезке между наименьшим и наибольшим из значений $x_0, x_1, x_2, \ldots, x_n$, а полином

$$\varpi_n(x)=\prod_{i=0}^{n}(x-x_i).$$

3. Если в двух точках заданы значения функции и ее первых производных, то полином какой минимальной степени может быть построен на этих точках?
Из условия имеем 4 условия (2 значения функции и 2 значения её производной). Следовательно, полином будет иметь 3-ю степень (максимальная).
Полином Ньютона: 0, 1, 2, 3 степени (минимальная — 0).
Полином Эрмита: 0, 1 степени (минимальная — 0).

4. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?
Данная информация о сортированных значениях необходима при вычислении корня функции методом обратной интерполяции (когда меняются местами значения.

5. Что такое выравнивающие переменные и как их применить для повышения точности интерполяции?
Выравнивающие переменные используются для того, чтобы можно было повысить точность вычисления производной функции. Они применяются, когда разделенные разности функции на некоторых интервалах меняются значительно.