

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 1**

|  |  |
| --- | --- |
| **Тема:** Алгоритм и программа построения интерполяционного полинома Ньютона  **Студент:** Елгин И. Ю.  **Группа:** ИУ7-44Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Градов В. М. |  |

Москва.

2021 г.

**Цель работы**: Получение навыков построения алгоритма интерполяции таблично

заданных функций полиномами Ньютона и Эрмита.

Исходные данные.

**1 Исходные данные.**

1. Таблица функции и её производных

|  |  |  |
| --- | --- | --- |
| x | 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.

Программа считывает таблицу из файла “*input.txt*”, находящегося в директории программы. В первой строке находятся количество строк и столбцов в остальных данные таблицы.

Значения X и N вводятся с клавиатуры

**2 Код на c99.**

struct.h

#include <stdio.h>

#include <stdlib.h>

#define EPS 0.00001

**typedef** **struct** data\_table

{

**double** \*\*table;// таблица значений

**int** size\_x, size\_y;// размер таблицы

}data\_table;

// чтение таблицы из файла в структуру

**void** **input\_from\_file**(data\_table \*d\_table, **char** \*fname);

// вычисление полинома Ньютона или Эрмита

**double** **newton\_ermit\_polinom**(data\_table \*d\_table, **int** p, **int** n, **double** x);

// поиск индекса начального значения для апроксимации

**int** **select\_first\_number**(data\_table \*d\_table, **double** x, **int** n);

// отчистка данных таблицы

**void** **free\_table**(**double** \*\*table, **int** n);

functions.c

#include "struct.h"

**double** **my\_abs**(**double** d)

{

**return** (d < **0**) ? -d : d;

}

**void** **free\_table**(**double** \*\*table, **int** n)

{

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

free(table[i]);

free(table);

}

//создаёт и высчитывает таблицу разделённых разностей

**double** \*\***creat\_div\_diff\_table**(data\_table \*d\_table, **int** p, **int** n)

{

**double** \*\*div\_diff\_table = malloc(n \* **sizeof**(**double** \*));

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

{

div\_diff\_table[i] = malloc((n - i) \* **sizeof**(**double**));

// заполняет начало таблицы разделённых разностей корнями или

// кратными корнями в зависимости от числа столбцов входной таблицы

div\_diff\_table[i][**0**] = d\_table->table[p + i / (d\_table->size\_y -**1**)][**1**];

}

**for** (**int** i = **1**; i < n; i++)

**for** (**int** j = **0** ; j < n - i; j++)

{

//в случае не кратных корней вычисляет разделённую разницу

//иначе записывает значение соответствующей производной

**if** ((my\_abs(div\_diff\_table[j][i - **1**] - div\_diff\_table[j + **1**][i - **1**]) > EPS) || (d\_table->size\_y - **1**) <= i)

div\_diff\_table[j][i] = (div\_diff\_table[j][i - **1**] - div\_diff\_table[j + **1**][i - **1**]) / (d\_table->table[p + j / (d\_table->size\_y - **1**)][**0**] - d\_table->table[p + (i + j) / (d\_table->size\_y - **1**)][**0**]);

**else**

div\_diff\_table[j][i] = d\_table->table[p + j / (d\_table->size\_y - **1**)][i + **1**];

}

**return** div\_diff\_table;

}

// ищет значение приближённое к x

**int** **find\_medium**(**double** \*\*arr, **int** size, **double** x)

{

**int** result = **0**;

**double** delta = my\_abs(arr[**0**][**0**] - x);

**for** (**int** i = **1**; i < size; i++)

{

**if** (my\_abs(arr[i][**0**] - x) < delta)

{

delta = my\_abs(arr[i][**0**] - x);

result = i;

}

}

**return** result;

}

**int** **select\_first\_number**(data\_table \*d\_table, **double** x, **int** n)

{

n = n / (d\_table->size\_y - **1**) + ((n % (d\_table->size\_y - **1**)) ? **1** : **0**);

**int** m = find\_medium(d\_table->table, d\_table->size\_x, x);

**if** (m <= n / **2**)

**return** **0**;

**else** **if** (d\_table->size\_x - m < (n / **2** + n % **2**))

**return** d\_table->size\_x - n;

**else**

**return** m - n / **2**;

}

**double** **newton\_ermit\_polinom**(data\_table \*d\_table, **int** p, **int** n, **double** x)

{

**double** k = **1**, result = d\_table->table[p][**1**];

**double** \*\*div\_diff\_table = creat\_div\_diff\_table(d\_table, p, n + **1**);

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

{

k \*= (x - d\_table->table[p + i / (d\_table->size\_y - **1**)][**0**]);

result += k \* div\_diff\_table[**0**][i + **1**];

}

free\_table(div\_diff\_table, n + **1**);

**return** result;

}

**void** **input\_from\_file**(data\_table \*d\_table, **char** \*fname)

{

**FILE** \*f = fopen(fname, "r");

fscanf(f, "%d %d", &(d\_table->size\_x), &(d\_table->size\_y));

d\_table->table = malloc(d\_table->size\_x \* **sizeof**(**double** \*));

**for** (**int** i = **0**; i < d\_table->size\_x; i++)

{

d\_table->table[i] = malloc(d\_table->size\_y \* **sizeof**(**double**));

**for** (**int** j = **0**; j < d\_table->size\_y; j++)

fscanf(f, "%lf", d\_table->table[i] + j);

}

}

main.c

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include "struct.h"  **int** **main**()  {  setbuf(stdout, NULL);  data\_table d\_table;  **int** n;  **double** x;  // чтение таблицы из файла в структуру  input\_from\_file(&d\_table, "input.txt");  // ввод x и n  printf("Введите значение X и степени полинома: ");  scanf("%lf %d", &x, &n);  // поиск индекса начального значения для апроксимации  **int** p = select\_first\_number(&d\_table, x, (n + **1**));  // вычисление полинома Ньютона или Эрмита и вывод результата  printf("**\n**Апроксимированное значение в точке: %lf**\n**",\  newton\_ermit\_polinom(&d\_table, p, n, x));  free\_table(d\_table.table, d\_table.size\_x);  **return** **0**;  } |

**3 Результаты работы.**

Формула для полинома Ньютона



Для вычисления полинома Эрмита берём кратные корни а в разделённых разностях записываем значения производных.

1. Значения y(x) при степенях полиномов Ньютона и Эрмита n= 1, 2, 3 и 4 при x = 0.386

|  |  |  |
| --- | --- | --- |
| n | Ньютон | Эрмит |
| 1 | 0.537866 | 0.542285 |
| 2 | 0.540340 | 0.540450 |
| 3 | 0.540411 | 0.540421 |
| 4 | 0.540422 | 0.540422 |

2.

Чтобы найти корень заданной выше табличной функции с помощью обратной интерполяции, поменяем столбцы x и y и отсортируем по y. Введём в программу X=0, n=4;

Результат:

X = 0.739088

**4 Вопросы при защите лабораторной работы.**

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

*Будет, тогда программа выдаст значение из таблицы наиболее приближенное к заданному, поэтому смысла в этом особо нет.*

2. Как практически оценить погрешность интерполяции? Почему сложно применить для

этих целей теоретическую оценку?

*Для практической оценки погрешности можно воспользоваться оценкой первого отброшенного члена.*

*Трудность использования теоретических оценок на практике состоит в*

*том, что производные интерполируемой функции обычно неизвестны.*

3. Если в двух точках заданы значения функции и ее первых производных, то полином

какой минимальной степени может быть построен на этих точках?

*Можно получить 4 корня 2 простых и 2 кратных, значит можно получить полином 3 степени, так как он строится по n+1 узлам.*

4. В каком месте алгоритма построения полинома существенна информация об

упорядоченности аргумента функции (возрастает, убывает)?

*При формировании конфигурации из n+1 узла.*

5. Что такое выравнивающие переменные и как их применить для повышения точности

интерполяции?

*Метод выравнивания заключается в том, что выяснив качественное поведение функции, стараются подобрать такие переменные выравнивания чтобы в новых переменных график мало отличался от прямой на протяжении нескольких шагов таблицы. Тогда в переменных интерполяция многочленом невысокой степени будет давать хорошую точность.*