**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 (ИУ-7)»

## **Лабораторная работа № 1**

**Тема** Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций.

**Студент** Сироткина П.Ю.

**Группа** ИУ7-46Б

## **Оценка (баллы)**

**Преподаватель** Градов В.М.

Москва, 2021 год

**Цель работы:**

Получение навыков построения алгоритма интерполяции таблично заданных функций полиномами Ньютона и Эрмита.

**Исходные данные**:

1. Таблица функции и ее производных (хранится в текстовом файле).
2. Степень аппроксимирующего полинома - n (в программе значение фиксировано и пробегает значения от 1 до 4 для сравнения эффективности).
3. Значение аргумента, для которого выполняется интерполяция (вводится с клавиатуры).

**Задание:**

1. Вычислить значения y(x) при степенях n = 1, 2, 3, 4 полиномов Ньютона и Эрмита при фиксированном x. Результаты свести в таблицу для сравнения полиномов.
2. Найти корень заданной табличной функции с помощью обратной интерполяции, используя полином Ньютона.

**Описание алгоритма:**

Введем некоторые понятия:

*Аппроксимация функции* - научный метод, заключающийся в нахождении близкой функции, но которая является более простой для анализа характеристик. Какую функцию стоит выбрать в том или ином случае, зависит от степени ответственности расчетов.

*Интерполяция* - нахождение промежуточных значений величины по имеющемуся дискретному набору значений.

*Полином* - многочлен от n переменных.

*Разделенная разность -* обобщение понятия производной для дискретного набора точек.

Таким образом, для двух точек:

**y(x0, x1) = (y0 – y1) / (x0 – x1)**

Для трех точек:

**y(x0, x1, x2) = (y(x0, x1) - y(x1, x2)) / (x0 – x2)**

Для n точек:

**y(x0, x1, …, x\_n) = (y(x0, x1, …, x\_n-1) - y(x1, x2, …, x\_n)) / (x0 – x\_n)**

На практике для осуществления интерполяции используют полином, т.к. это одна из самых простых функций.

Можно показать, что полином представим через разделенные разности в виде:

**P\_n(x) = P\_n(x0) + (x - x0)P(x0, x1) + (x – x0)(x – x1)P(x0, x1, x2) + … + (x – x0)…(x – x\_n-1)P(x0, x1,…,x\_n)**

Такой полином, представимый через разделенные разности, называется полиномом Ньютона.

Алгоритм построения интерполяционного полинома Ньютона сводится к следующим этапам:

1. При заданном n и x строится конфигурация узлов (интерполяционная база), на которых строится полином. По возможности они должны располагаться симметрично относительно заданного аргумента.
2. Строится таблица разделенных разностей.
3. Строится полином.
4. Вычисляется значение функции в заданной точке.

Помимо полинома Ньютона в практике вычислений находит применение еще один полином, называемый полиномом Эрмита. Он определяется не только значениями функции, но еще и значениями производных различного порядка. Вывод формулы для полинома Эрмита производят построением по узлам полинома Ньютона. В областях между узлами средние наклоны кривых и полинома совпадают. Если сближать узлы x\_i и x\_i+1, то средний наклон будет все точнее передавать первую производную функции. В пределе после совпадения узлов получают искомый многочлен с кратным узлом, являющийся полиномом Эрмита n-й степени.

Общее выражение для разделенных разностей при кратности узлов m имеет вид:

Y(x0, x0,...,x0\_m) = y`(m - 1)(x0) / (m – 1)!

*Обратная интерполяция* применяется для приближенного нахождения корня монотонных функций. Суть ее состоит в том, что в таблице этой функции меняются местами столбцы и выполняется обычная интерполяция при задании аргумента, равным нулю.

**Код программы:**

**Main.c**

**#include "io.h"**

**#include "calc.h"**

**int main(void)**

**{**

**FILE \*f;**

**char name[MAX\_NAME\_LEN];**

**int rc = read\_filename(name, &f);**

**point\_t points[MAX\_POINTS\_COUNT];**

**int n = 0;**

**double x;**

**if (!rc)**

**{**

**rc = read\_data(f, points, &n);**

**if (!rc)**

**{**

**print(points, n);**

**rc = get\_x(&x);**

**if (!rc)**

**{**

**newton(points, n, x);**

**hermite(points, n, x);**

**root(points, n);**

**fclose(f);**

**}**

**}**

**}**

**return rc;**

**}**

**Io.c**

**#include "io.h"**

**int read\_filename(char name[], FILE \*\*f)**

**{**

**printf("\nВведите имя файла для чтения данных: ");**

**if (!fgets(name, MAX\_NAME\_LEN, stdin))**

**{**

**printf("Ошибка во время считывания названия: превышена максимальная длина названия.\n");**

**return ERR\_READ;**

**}**

**name[strlen(name) - 1] = '\0';**

**if (!strlen(name))**

**{**

**printf("Ошибка во время считывания названия: введена пустая строка.\n");**

**return ERR\_LEN;**

**}**

**\*f = fopen(name, "r");**

**if (!(\*f))**

**{**

**printf("\nОшибка: файла с таким именем не существует.\n");**

**return ERR\_FOPEN;**

**}**

**printf("\nИмя файла успешно считано.\n");**

**return EXIT\_SUCCESS;**

**}**

**int read\_data(FILE \*f, point\_t points[], int \*n)**

**{**

**printf("\nЧтение данных из файла...\n");**

**rewind(f);**

**char buf[MAX\_INPUT\_LEN];**

**int i = 0;**

**\*n = 0;**

**while (fgets(buf, sizeof(buf), f))**

**{**

**if (sscanf(buf, "%lf%lf%lf", &(points[i].x), &(points[i].y), &(points[i].d\_y)) != 3)**

**{**

**printf("\nОшибка: файл содержит некорректные данные в %d-ой строке.\n", i + 1);**

**return ERR\_DATA;**

**}**

**i++;**

**}**

**if (!feof(f))**

**{**

**printf("\nОшибка: файл содержит некорректные данные.\n");**

**return ERR\_DATA;**

**}**

**if (!i)**

**{**

**printf("\nОшибка: пустой файл.\n");**

**return ERR\_EMPTY;**

**}**

**\*n = i;**

**printf("\nДанные успешно считаны из файла.\n");**

**return EXIT\_SUCCESS;**

**}**

**void print(point\_t points[], int n)**

**{**

**printf("\nПечать данных...\n\n┏━━━━━━━━━━┳━━━━━━━━━━┳━━━━━━━━━━┓\n");**

**printf("┃ x ┃ y ┃ dy ┃\n");**

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

**{**

**printf("┣━━━━━━━━━━╋━━━━━━━━━━╋━━━━━━━━━━┫\n");**

**printf("┃%-10lf┃%-10lf┃%-10lf┃\n", points[i].x, points[i].y, points[i].d\_y);**

**}**

**printf("┗━━━━━━━━━━┻━━━━━━━━━━┻━━━━━━━━━━┛\n");**

**}**

**int get\_x(double \*x)**

**{**

**printf("\nВведите точку для аппроксимизации: ");**

**if (scanf("%lf", x) != 1)**

**{**

**printf("\nОшибка: введено некорректное число.\n");**

**return ERR\_DATA;**

**}**

**printf("\nТочка успешно введена.\n");**

**return EXIT\_SUCCESS;**

**}**

**Calc.c**

**#include "calc.h"**

**int newton(point\_t points[], int n, double x)**

**{**

**if (n < 4)**

**{**

**printf("\nНевозможно произвести анализ: необходимо минимум 4 точки.\n");**

**return ERR\_NOT\_ENOUGH\_DATA;**

**}**

**int res = get\_nearest\_pos(points, n, x);**

**int start;**

**if (res < 2)**

**start = 0;**

**else if (res + 2 > n)**

**start = n - 5;**

**else**

**start = res - 2;**

**double diff[4][4];**

**for (int i = 0; i < 4; i++)**

**diff[i][0] = (points[i + start].y - points[i + start + 1].y) / (points[i + start].x - points[i + start + 1].x);**

**for (int i = 1; i < 4; i++)**

**for (int j = 0; j < 4 - i; j++)**

**{**

**if (points[j].x == points[j + i + 1].x)**

**{**

**printf("\nОшибка: деление на ноль.\n");**

**return ERR\_ZERO\_DIV;**

**}**

**diff[j][i] = (diff[j][i - 1] - diff[j + 1][i - 1]) / (points[j].x - points[j + i + 1].x);**

**}**

**printf("\n┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓");**

**printf("\n┃ Полином Ньютона ┃");**

**printf("\n┣━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━┫");**

**printf("\n┃ n = 1 ┃ n = 2 ┃ n = 3 ┃ n = 4 ┃");**

**printf("\n┣━━━━━━━━━━━╋━━━━━━━━━━━╋━━━━━━━━━━━╋━━━━━━━━━━━┫\n");**

**double ans = points[start].y, temp = (x - points[start].x);**

**for (int i = 0; i < 4; i++)**

**{**

**ans += temp \* diff[0][i];**

**printf("┃%11lf", ans);**

**temp \*= (x - points[start + i + 1].x);**

**}**

**printf("┃\n┗━━━━━━━━━━━┻━━━━━━━━━━━┻━━━━━━━━━━━┻━━━━━━━━━━━┛\n");**

**return EXIT\_SUCCESS;**

**}**

**int hermite(point\_t points[], int n, double x)**

**{**

**if (n < 4)**

**{**

**printf("\nНевозможно произвести анализ: необходимо минимум 4 точки.\n");**

**return ERR\_NOT\_ENOUGH\_DATA;**

**}**

**int res = get\_nearest\_pos(points, n, x);**

**int start;**

**if (res < 3)**

**start = 0;**

**else**

**start = res - 1;**

**double diff[4][4];**

**diff[0][0] = points[start].d\_y;**

**diff[1][0] = (points[start].y - points[start + 1].y) / (points[start].x - points[start + 1].x);**

**diff[2][0] = points[start + 1].d\_y;**

**diff[3][0] = (points[start + 1].y - points[start + 2].y) / (points[start + 1].x - points[start + 2].x);**

**for (int i = 1; i < 4; i++)**

**for (int j = 0; j < 4 - i; j++)**

**if (points[j].x == points[j + i + 1].x)**

**diff[i][j] = points[j].d\_y;**

**else**

**diff[j][i] = (diff[j][i - 1] - diff[j + 1][i - 1]) / (points[j].x - points[j + i + 1].x);**

**printf("\n┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓");**

**printf("\n┃ Полином Эрмита ┃");**

**printf("\n┣━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━┫");**

**printf("\n┃ n = 1 ┃ n = 2 ┃ n = 3 ┃ n = 4 ┃");**

**printf("\n┣━━━━━━━━━━━╋━━━━━━━━━━━╋━━━━━━━━━━━╋━━━━━━━━━━━┫\n");**

**double ans = points[start].y, temp = (x - points[start].x);**

**for (int i = 0; i < 4; i++)**

**{**

**ans += temp \* diff[0][i];**

**printf("┃%11lf", ans);**

**temp \*= (x - points[start + i + 1].x);**

**}**

**printf("┃\n┗━━━━━━━━━━━┻━━━━━━━━━━━┻━━━━━━━━━━━┻━━━━━━━━━━━┛\n");**

**return EXIT\_SUCCESS;**

**}**

**int root(point\_t points[], int n)**

**{**

**if (n < 4)**

**{**

**printf("\nНевозможно произвести анализ: необходимо минимум 4 точки.\n");**

**return ERR\_NOT\_ENOUGH\_DATA;**

**}**

**double x = 0;**

**int res = get\_nearest\_pos(points, n, x);**

**int start;**

**if (res < 2)**

**start = 0;**

**else if (res + 2 > n)**

**start = n - 5;**

**else**

**start = res - 2;**

**double diff[4][4];**

**for (int i = 0; i < 4; i++)**

**diff[i][0] = (points[i + start].x - points[i + start + 1].x) / (points[i + start].y - points[i + start + 1].y);**

**for (int i = 1; i < 4; i++)**

**for (int j = 0; j < 5 - i - 1; j++)**

**{**

**if (points[j].y == points[j + i + 1].y)**

**{**

**printf("\nОшибка: деление на ноль.\n");**

**return ERR\_ZERO\_DIV;**

**}**

**diff[j][i] = (diff[j][i - 1] - diff[j + 1][i - 1]) / (points[j].y - points[j + i + 1].y);**

**}**

**printf("\n┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓");**

**printf("\n┃ Нахождение корня с помощью полинома Ньютона ┃");**

**printf("\n┣━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━┳━━━━━━━━━━━┫");**

**printf("\n┃ n = 1 ┃ n = 2 ┃ n = 3 ┃ n = 4 ┃");**

**printf("\n┣━━━━━━━━━━━╋━━━━━━━━━━━╋━━━━━━━━━━━╋━━━━━━━━━━━┫\n");**

**double ans = points[start].x, temp = (x - points[start].y);**

**for (int i = 0; i < 4; i++)**

**{**

**ans += temp \* diff[0][i];**

**printf("┃%11lf", ans);**

**temp \*= (x - points[start + i + 1].y);**

**}**

**printf("┃\n┗━━━━━━━━━━━┻━━━━━━━━━━━┻━━━━━━━━━━━┻━━━━━━━━━━━┛\n");**

**return EXIT\_SUCCESS;**

**}**

**int get\_nearest\_pos(point\_t points[], int n, double x)**

**{**

**int res = 0;**

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

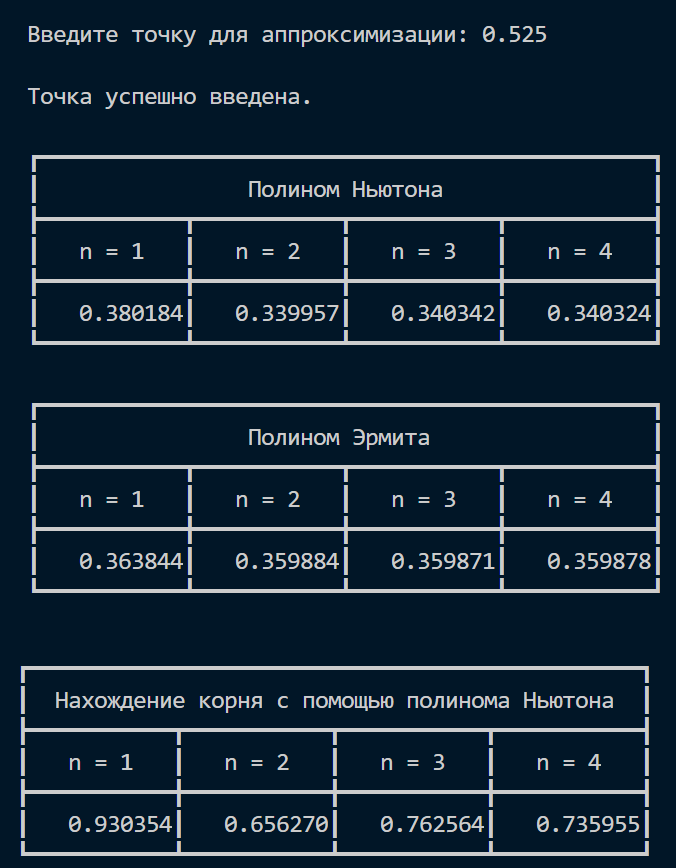
**if (fabs(points[i].x - x) < (fabs(points[res].x - x) - EPS))**

**res = i;**

**return res;**

**}**

**Результаты работы с данными из условия задачи:**



**Контрольные вопросы:**

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

Программа работать не будет, т.к. алгоритм не будет вычислять коэффициенты при степенях полинома. В ответ попадет лишь свободный член, то есть y0 - очевидно, что в общем случае это неверный ответ.

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

Практически погрешность интерполяции определяется по модулю последнего слагаемого полинома. (В ходе расчетов в ЛР практическая погрешность составила меньше одного процента).

Трудность использования теоретических оценок на практике состоит в том, что производные интерполируемой функции обычно неизвестны, поэтому для определения погрешности удобнее воспользоваться оценкой первого отброшенного члена.

*3. Если в двух точках заданы значения функции и ее первых производных, то полином какой минимальной степени может быть построен на этих точках?*

Может быть построен полином как минимум 3-ей степени.

*4. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?*

В той части алгоритма, когда выбирается место, куда следует вставить точку при поиске интерполяционной базы.

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

Выравнивающие переменные применяются для аппроксимации быстропеременных функций, т.к. для быстропеременных функций нужно учитывать гораздо больше точек, чтобы расчеты были достоверными. Интерполяция проводится для выравнивающих переменных, а затем применяют обратное интерполирование. Такие переменные можно применять, например, в моделировании физических явлений, потому что там часто фигурируют показательные функции.