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)

Вычислительные алгоритмы.

Лабораторная работа №1.

«Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций»

Студент **Трошкин Николай Романович**

Группа **ИУ7-46Б**

Студент _____ Трошкин Н.Р.

подпись, дата *фамилия, и.о.*

Преподаватель _____ Градов В.М.

подпись, дата *фамилия, и.о.*

2021 г.

Цель работы

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

Задание

Исходные данные

1. Таблица функции и ее производных.

x	y	y’
0.00	1.000000	-1.00000
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.

Требуемый результат

1. Значения y(x) при степенях полиномов Ньютона и Эрмита n = 1, 2, 3, 4 для фиксированного x = 0.525.
2. Корень табличной функции, найденный с помощью обратной интерполяции полиномом Ньютона.

Краткий алгоритм

Полином Ньютона: набор точек сортируется по возрастанию аргумента, затем выбирается конфигурация узлов, аргументы которых окружают заданный аргумент x, на этой конфигурации вычисляются разделенные разности по формуле

$$y(x_i, x_j) = [\, y(x_i) - y(x_j)\,] \,/ (x_i - x_j),$$

$$y(x_i, x_j, x_k) = [\, y(x_i, x_j) - y(x_j, x_k)\,] \,/ (x_i - x_k),$$

и т.д. для более высоких порядков.

Затем строится полином, для которого находится значение функции в точке x, по формуле

$$y(x) \approx y(x_0) + \sum_{k=1}^{n} (x - x_0)(x - x_1)..(x - x_{k-1})\; y(x_0, .., x_k).$$

При **обратной интерполяции** все то же самое, но столбцы x и y меняются местами, а значение полинома ищется в точке 0.

Полином Эрмита: учитываются производные функции, то есть узлы учитываются в полиноме дважды. Если требуется найти разделенную разность для двух совпадающих узлов в некоторой точке, она равна производной функции в данной точке. В остальном алгоритм совпадает с нахождением полинома Ньютона.

Полученный результат

Степень полинома n	1	2	3	4
Полином Ньютона, P(0.525)	0.338	0.340	0.340	0.340
Полином Эрмита, H(0.525)	0.343	0.345	0.345	0.345

Корень заданной функции: x = 0.739 (получен при степени полинома Ньютона n = 1, 2, 3, 4).

Ответы на вопросы

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

Программа не упадет, но результат ее работы будет очень неточным: для такого полинома нужна всего одна точка и ее значение y0 будет являться значением полинома при любых значениях аргумента x.

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

Погрешность интерполяции можно оценить, анализируя коэффициенты полинома, полученные при вычислении разделенных разностей высоких порядков. Если коэффициент очень мал по модулю, то вклад высокой степени в итоговый результат уже не такой большой, значит, достигнута уже высокая точность.
Теоретическую оценку погрешности применить трудно, так как для полинома степени n требуется знать (n + 1)-ую производную исходной табличной функции, которая обычно неизвестна.

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

Минимально - первой, если не использовать информацию о производных и построить полином Ньютона.
Максимальная степень - третья, в этом случае используются все 4 условия и строится полином Эрмита.

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

Данная информация существенна при выборе конфигурации узлов. Если аргумент функции отсортирован, то удобнее искать узлы, наиболее близкие к заданному аргументу функции.

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

Выравнивающие переменные - переменные, используемые при интерполяции быстро меняющихся функций, для которых требуется создавать таблицы очень больших объемов. Чтобы сократить накладные расходы по памяти, выравнивающие переменные подбирают так, чтобы сделать график в этих переменных близким к прямой на отдельных участках. При этом интерполяция сначала находится в новых переменных, а затем преобразуется к старым обратным интерполированием.

Код программы

Основная функция приложения

```
int main(void)
{
    show_info();   // информация о программе
    FILE *file = get_file();    // получение файла с данными от пользователя
    mode_t mode = get_mode();    // получение вида интерполяции от пользователя
    int n = get_degree();    // получение степени полинома от пользователя

    double x, y;
    if (mode != INVERSE)
        x = get_argument();    // получение значения аргумента от пользователя
    else
        x = 0.0;    // обратная интерполяция - ищем корень, поэтому аргумент = 0.0

    size_t size = 0;
    record_t *data = export_to_array(file, &size);
    fclose(file);
```

```
    if (!data)
        return EXIT_FAILURE; // выходим, если не удалось выделить память

    sort(data, size, mode);
    // находим начало конфигурации узлов в массиве
    size_t start_index = get_config(data, size, mode, x, n);

    switch (mode)     // интерполяция в зависимости от выбора пользователя
    {
        case NEWTON:
            y = interp_newton(data, x, start_index, n);
            break;

        case HERMITE:
            y = interp_hermite(data, x, start_index, n);
            break;

        case INVERSE:
            y = interp_inverse(data, size, x, start_index, n);
            break;

        default:
            break;
    }

    printf("%.3lf", y);
    free(data);
    return EXIT_SUCCESS;
}
```

Функции, использованные в начинке программы:

```
// функция находит первый индекс в упорядоченном массиве записей,
// с которого начинаются узлы выбранной конфигурации
size_t get_config(record_t *data, size_t size, mode_t mode, double x, int n)
{
    int dots = n + 1;    // количество записей, которое требуется для полинома
    if (mode == HERMITE)
        dots = (dots + 1) / 2;
```

```
    size_t i = 0;
    // поиск первого индекса, по которому значение аргумента больше x
    if (mode == INVERSE)
        while (i < size && data[i].y < x)
            i++;
    else
        while (i < size && data[i].x < x)
            i++;

    if (i == size)
        return size - dots;  // нижняя часть таблицы

    if (i < (size_t)dots / 2)
        return 0;  // верхняя часть таблицы

    if (size <= (size_t)dots && mode != HERMITE)
        return 0;

    return i - dots / 2;  // найдется поровну точек слева и справа от x
}

static double divided_difference(double y0, double y1, double x0, double x1)
{
    return (y0 - y1) / (x0 - x1);  // возвращает разделенную разность по 4 аргументам
}

// получение значения полинома степени n с заданными коэффициентами
// при заданном значении аргумента
static double get_polynom_value(double *factors, int n, double x,
record_t *data, size_t start_index)
{
    double y = 0.0;

    for (int i = 0; i <= n; i++)
    {
        double prod = factors[i];
        for (int j = 0; j < i; j++)
            prod *= (x - data[start_index + j].x);
        y += prod;
```

```
    }

    return y;
}

// линейная интерполяция с помощью полинома Ньютона
double interp_newton(record_t *data, double x, size_t start_index, int n)
{
    // в этом массиве сохраняются множители, которые будут в полученном полиноме
    double *factors = malloc((n + 1) * sizeof(double));

    // дополнительные промежуточные разности, из них получаются разности следующих
    // порядков
    double *extra_diffs = malloc((n + 1) * sizeof(double));

    for (int i = 0; i <= n; i++)
        extra_diffs[i] = data[start_index + i].y;

    factors[0] = extra_diffs[0];
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= n - i; j++)
            // из ранее вычисленных разностей получаем новую разделенную разность
            extra_diffs[j] = divided_difference(extra_diffs[j], extra_diffs[j + 1],
                data[start_index + j].x, data[start_index + j + i].x);

        factors[i] = extra_diffs[0];
    }

    return get_polynom_value(factors, n, x, data, start_index);
}

// линейная интерполяция с помощью полинома Эрмита
double interp_hermite(record_t *data, double x, size_t start_index, int n)
{
    double *factors = malloc((n + 1) * sizeof(double));
    double *extra_diffs = malloc((n + 1) * sizeof(double));

    int i;
    for (i = 0; i <= n - 1; i += 2)
        extra_diffs[i] = extra_diffs[i + 1] = data[start_index + i / 2].y;
```

```
    if (i == n)
        extra_diffs[i] = data[start_index + i / 2].y;

    factors[0] = extra_diffs[0];
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= n - i; j++)
            if (i == 1 && j % 2 == 0)
                extra_diffs[j] = data[start_index + j / 2].dy;
            else
                extra_diffs[j] = divided_difference(extra_diffs[j], extra_diffs[j + 1],
                    data[start_index + j / 2].x, data[start_index + (j + i) / 2].x);

        factors[i] = extra_diffs[0];
    }

    return get_polynom_value(factors, n, x, data, start_index);
}

// получение значения корня с помощью обратной интерполяции
double interp_inverse(record_t *data, size_t size, double x, size_t start_index, int n)
{
    // меняем местами поля x и y в записях
    for (size_t i = 0; i < size; i++)
    {
        double temp = data[i].x;
        data[i].x = data[i].y;
        data[i].y = temp;
    }

    // x = 0.0 - это предусмотрено в функции main()
    return interp_newton(data, x, start_index, n);
}
```