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 интерполяционного полинома Ньютона

Студент Пересторонин Павел

Группа ИУ7-43Б

Оценка (баллы) ___________________

Преподаватель Градов Владимир Михайлович

Москва.
2020 г.

Теоретическая часть:
Аппроксимация функции f(x) - нахождение такой функции g(x) (называемой аппроксимирующей), которая была бы близка заданной. Критерии близости функций могут быть различные (например, метод наименьших квадратов (наименьшее квадратичное приближение)).
В случае если приближение строится на дискретном наборе точек, аппроксимацию называют точечной или дискретной.
Интерполяция — апроксимация по точкам (называемых узлами), заключающаяся в нахождении промежуточных значений по дискретному набору уже известных значений.
Интерполяционный полином Ньютона — полином вида:

$$P_N(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \dots +$$
$$+ c_N(x - x_0)(x - x_1) \cdot \dots \cdot (x - x_{N-1}) \equiv \sum_{i=0}^{N} c_i \prod_{j=0}^{i-1} (x - x_j)$$

где $c_i = y(x_1; x_2; \dots; x_i)$, а $y(x_i, \dots, x_j)$ называют разделенной разностью и расчитывают из следующих рекуррентных соотношений:
$y(x_i; x_j) = (y_i - y_j) / (x_i - x_j)$;
$y(x_i; x_j; x_k) = [y(x_i; x_j) - y(x_j; x_k)] / (x_i - x_k)$;
$y(x_0) = f(x_0) = y_0$;
Погрешность полинома n-ой степени $R_n(x)$ (критерий «близости» полученной функции):

$$|R_n(x)| \le \frac{M_{n+1}}{(n+1)!}|w_{n+1}(x)| \qquad w_{n+1} = (x - x_0)\dots(x - x_n)$$

где $M_{n+1} = \max_{x \in [\alpha, \beta]} |f^{n+1}(x)|$

Таким образом погрешность зависит от гладкости и полинома.
Экстраполяция — выход за пределы представленных в таблице значений (возникает сильное понижение точности).
Задание: вычислить значение функции в заданной точке, используя интерполяционный полином Ньютона.
Входные данные:
1. Таблица функции в файле.
2. Значение аргумента, для которого нужно найти значение.
3. Степень интерполяционного полинома.

Выходные данные:
1. Интерполяционный полином.
2. Найденное значение для заданного аргумента.
3. Корень табличной функции, найденный методом половинного деления.
4. Корень табличной функции, найденный методом обратной интерполяции.

АЛГОРИТМ:
1. Нахождение точек табличной функции, которые будут участвовать в построении полинома.
2. Расчет разделенных разностей для этих точек (если мы не рассчитывали у нас не осталось расчетов с прошлого раза).
3. Расчет значение полинома для заданного арумента.
(Примечание: аргумент `mode` принимает одно из 2 значений: NORMAL (= 0) (в таком случае 1 столбец принимается за аргумент) и REVERSED (= 1) (тогда 2 столбец — аргументы, а 1 — значения); нужен для более удобного расчета обратной интерполяции; аргумент `cache_usage` - флаг возможности использования «кэширования» (если мы в предыдущий раз рассчитывали разделенные разности для данной последовательности, то будем использовать прошлый расчет (начало последовательности и режим запоминаются, нет смысла пересчитывать)).

1.1. Нахождение 2 соседних аргументов (вернее нахождение 2ого из двух), между которыми лежит «наш» аргумент (для которого надо найти значение); эти 2 аргумента — центры последовательности, берущейся для построения полинома.
(1.1.* Если такие 2 «соседа» не найдены, то имеем экстраполяцию (расчета при ней нет в связи с маленькой точностью)
1.2. Вычисление начала последовательности (исходя из числа членов и длины последовательности с учетом количества оставшихся за «центром» членов).

2.0. Разделенные суммы хранятся в массиве в виде { [0, 1], … , [n - 1, n], [0, 1, 2], … , [n - 2, n - 1, n], … , [0, 1, 2, … , n] } , n — степень полинома, n + 1 = кол-во точек.
2.1. Заполнение разделенных разностей для 2 элементов (отдельный цикл, итерация по значениям функции).
2.2. Нахождение и заполнение оставшихся членов по рекуррентным соотношениям.

3.1. Расчет полинома Ньютона по известным значениям коэффициентов (разделенных разностей)

```c
double interpolation(data_t *const data, const double argument,
                     const int mode, const int cache_usage)
{
    int section_start = find_section(data, argument, mode);

    if (!(cache_usage && mode == data->is_cached &&
            section_start == data->cached_for))
        count_div_sums(data, section_start, mode);

    data->is_cached = mode;
    data->cached_for = section_start;

    return polynomial_value(data, argument, section_start, mode);
}
```

```c
int find_section(const data_t *const data, const double argument, const int mode)
{
    double sign = (data->table[mode ^ 0][0] < data->table[mode ^ 0][1]) - 0.5;
    int index = 0;
    while (index < data->size && (argument - data->table[mode ^ 0][index]) * sign > 0)
        index++;
    if (index == 0 || index == data->size)
    {
        index -= data->n;
        fprintf(OUTPUT, "Extrapolation!\n");
        return index > 0 ? index : 0;
    }

    int end_diff = data->n / 2 - (1 - (1 & data->n)) - (data->size - index - 1);

    if (end_diff < 0)
        end_diff = 0;

    index -= data->n / 2 + end_diff;
    return index > 0 ? index : 0;
}
```

```c
void count_div_sums(data_t *const data, const int section_start, const int mode)
{
    for (int i = section_start; i < section_start + data->n - 1; i++)
        data->divided_sums[i - section_start] = (data->table[1 ^ mode][i] -
                data->table[1 ^ mode][i + 1]) /
            (data->table[0 ^ mode][i] - data->table[0 ^ mode][i + 1]);

    double *cur_elem = data->divided_sums + data->n - 1;
    double *sum_ptr = data->divided_sums;

    for (int i = 1; i < data->n - 1; i++)
    {
        for (int j = 0; j < data->n - i - 1; j++)
        {
            *cur_elem = (*sum_ptr - *(sum_ptr + 1)) /
                (data->table[0 ^ mode][j] - data->table[0 ^ mode][j + 1 + i]);
            cur_elem++, sum_ptr++;
        }
        sum_ptr++;
    }
}
```

```c
static double polynomial_value(data_t *const data, const double argument,
                               const int section_start, const int mode)
{
    double multi = 1;
    double summary = data->table[1 ^ mode][section_start];
    int index = 0;

    for (int i = 0; i < data->n - 1; i++)
    {
        multi *= (argument - data->table[0 ^ mode][section_start + i]);
        summary += multi * data->divided_sums[index];
        index += data->n - i - 1;
    }
    return summary;
}
```