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 №2.

«Построение и программная реализация алгоритма многомерной 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 двух переменных.

Задание

Исходные данные

1. Таблица функции с количеством узлов 5x5.

y \ x	0	1	2	3	4
0	0	1	4	9	16
1	1	2	5	10	17
2	4	5	8	13	20
3	9	10	13	18	25
4	16	17	20	25	32

2. Степень аппроксимирующих полиномов - nx, ny.
3. Значение аргументов x и y, для которых выполняется интерполяция.

Требуемый результат

Результат интерполяции z(x,y) при степенях полиномов 1,2,3 для x=1.5, y=1.5.

Краткий алгоритм

Набор точек хранится в виде матрицы, в одном столбце которой находятся точки с одним и тем же аргументом x, а в одной строке - точки с одним и тем же аргументом y.

Матрица сортируется по возрастанию x в строке и возрастанию y в столбце, затем выбирается конфигурация узлов по x и по y так, что выбранные узлы окружают соответствующие введенные аргументы x и y.
Сначала производится ny + 1 линейных интерполяций (с помощью полинома Ньютона) для каждого фиксированного значения y[i] из выбранной конфигурации по аргументу x и вычисляется значение функции z(x, y) в точке x при y = y[i].
Затем по полученным значениям z производится одна линейная интерполяция по переменной y.

Полученный результат

x = 1.5, y = 1.5

Степени полиномов	ny = 1	ny = 2	ny = 3
nx = 1	5.000	4.750	4.750
nx = 2	4.750	4.500	4.500
nx = 3	4.750	4.500	4.500

Ответы на вопросы

1. Пусть производящая функция таблицы суть z(x,y)=x^2+y^2. Область определения по x и y - 0-5 и 0-5. Шаги по переменным равны 1. Степени nx = ny =1, x=y=1.5. Приведите по шагам те значения функции, которые получаются в ходе последовательных интерполяций по строкам и столбцу.

Выбранные x и y: x1 = 1, x2 = 2; y1 = 1, y2 = 2.
Интерполяция по строке при фиксированном y = 1:
z(x, y1) = 3.500
Интерполяция по строке при фиксированном y = 2:
z(x, y2) = 6.500
Интерполяция по столбцу по полученным выше значениям:
z(x, y) = 5.000

2. Какова минимальная степень двумерного полинома, построенного на четырех узлах? На шести узлах?

В обоих случаях минимальная степень двумерного полинома равна 2, т.к. полином первой степени строится максимум по трем узлам и имеет вид z = a + bx + cy.
4 узла для первой степени - уже слишком много.
Полином второй степени можно построить по шести узлам, и он примет вид z = a + bx + cy + dx^2 + gy^2 +hxy. Значит, и по 4 узлам можно построить полином второй степени.

3. Предложите алгоритм двумерной интерполяции при хаотичном расположении узлов, т.е. когда таблицы функции на регулярной сетке нет, и метод последовательной интерполяции не работает. Какие имеются ограничения на расположение узлов при разных степенях полинома?

При хаотичном расположении узлов строится полином, зависящий от двух переменных, например, имеющий вид:
z = a + bx + cy (первой степени)
или
z = a + bx + cy + dx^2 + gy^2 +hxy (второй степени)

В конфигурацию выбираются узлы в количестве, равном количеству неизвестных у полинома (3 для первой степени, 6 для второй). Узлы должны быть ближайшими к точке интерполяции. Подставив узлы (x, y, z) в общий вид полинома, получим систему уравнений, решив которую, получим коэффициенты аппроксимирующего полинома.

4. Пусть на каком-либо языке программирования написана функция, выполняющая интерполяцию по двум переменным. Опишите алгоритм использования этой функции для интерполяции по трем переменным.

Пусть функция трех переменных зависит от x, y, z. Можно зафиксировать nz + 1 значений аргумента z (nz - степень полинома по координате z), по каждому провести двумерную интерполяцию по

переменным x, y. Тогда будет получено nz + 1 значений функции u(x, y, z[i]) для каждого фиксированного z[i]. По этим значениям совершается одна линейная интерполяция по переменной z.

5. Можно ли при последовательной интерполяции по разным направлениям использовать полиномы несовпадающих степеней или даже разные методы одномерной интерполяции, например, полином Ньютона и сплайн?

Можно, от каждой интерполяции по сути нужен только конечный ее результат. Каким способом он получен - не так важно. Например, интерполировать по строкам можно со степенью полинома = 3, а по столбцу при степени = 2, по строкам использовать полином Эрмита или сплайн, а по столбцу - полином Ньютона.

6. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.

При двумерной интерполяции на треугольной конфигурации узлов алгоритм, как и при линейной интерполяции, сводится к вычислению разделенных разностей и “конструированию” из них полинома.
Схема вычисления разделенных разностей похожа на схему для одномерного случая:

$$z(x_0, y_0, y_1) = \frac{z(x_0, y_0) - z(x_0, y_1)}{y_0 - y_1}, z(x_0, x_1, y_0) = \frac{z(x_0, y_0) - z(x_1, y_0)}{x_0 - x_1}$$

Для разделенных разностей более высоких порядков формула аналогичная, в ней присутствуют уже ранее вычисленные разности меньших порядков.

Обобщенная запись полинома от двух переменных имеет вид:

$$P_n(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n-1} z(x_0, \ldots, x_i, y_0, \ldots, y_j) \prod_{p=0}^{i-1} (x - x_p) \prod_{q=0}^{j-1} (y - y_q) \ .$$

По данному полиному можно вычислить значение функции z(x, y) в заданной точке.

Код программы

Основная функция приложения

```
int main(void)
{
    show_info();   // информация о программе
    FILE *file = get_file();    // получение файла с данными от пользователя
    int nx, ny;
    get_degrees(&nx, &ny);    // получение степеней полиномов от пользователя

    double x, y, z;
    get_arguments(&x, &y);    // получение значений аргументов от пользователя

    size_t rows = 0, cols = 0;
    record_t *data = export_to_matrix(file, &rows, &cols);
    fclose(file);

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

    // сортировка матрицы записей для удобства нахождения конфигурации
    sort(data, rows, cols);

    z = mult_interp(data, rows, cols, x, y, nx, ny);    // сама интерполяция
    printf("%.3lf", z);
    free(data);
    return EXIT_SUCCESS;
}
```

Функции, использованные в начинке программы:

```
// функция получает первый индекс в упорядоченной матрице записей,
// с которого начинаются строки выбранной конфигурации (ближайшие к y)
static size_t get_y_config(record_t *data, size_t rows, size_t cols, double y, int ny)
{
```

```
    int count_y = ny + 1;

    size_t i = 0;
    // поиск первой строки, по которой значение аргумента y больше введенного y
    while (i < rows && data[i * cols].y < y)
        i++;

    if (i == rows)
        return rows - count_y;  // нижняя часть таблицы

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

    if (rows <= (size_t)count_y)
        return 0;

    return i - count_y / 2;  // найдется поровну точек сверху и снизу от y
}
```

Аналогичная функция есть для поиска ближайших значений к аргументу x, она такая же, но бежит по столбцам матрицы

```
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 от одной переменной 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, int flag)
{
    // флаг flag равен 1, если собираем полином, зависящий от x при
    // фиксированном y, иначе полином, зависящий от y
    double z = 0.0;

    for (int i = 0; i <= n; i++)
    {
        double prod = factors[i];
        for (int j = 0; j < i; j++)
            if (flag == 1)
                prod *= (x - data[start_index + j].x);
```

```
            else
                prod *= (x - data[start_index + j].y);
        z += prod;
    }

    free(factors);
    return z;
}

// линейная интерполяция с помощью полинома Ньютона
static double interp_newton(record_t *data, size_t cols, double x,
size_t start_index, int n, int row)
{
    // 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));

    // row = -1 означает, что передана не матрица, а массив со значениями,
    // полученными при интерполяции по строкам, в нем отсчитывать
    // количество строк не нужно
    for (int i = 0; i <= n; i++)
        if (row != -1)
            extra_diffs[i] = data[start_index + row * cols + i].z;
        else
            extra_diffs[i] = data[start_index + i].z;

    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
        // в зависимости от того, интерполируем по строкам или по столбцу
            if (row != -1)
                extra_diffs[j] = divided_difference(extra_diffs[j], extra_diffs[j + 1],
                    data[start_index + j + row * cols].x, data[start_index + j + i + row * cols].x);
            else
                extra_diffs[j] = divided_difference(extra_diffs[j], extra_diffs[j + 1],
                    data[start_index + j].y, data[start_index + j + i].y);
```

```
        factors[i] = extra_diffs[0];
    }

    free(extra_diffs);

    if (row != -1)
        return get_polynom_value(factors, n, x, data, start_index + row * cols, 1);

    return get_polynom_value(factors, n, x, data, start_index, 0);
}

// многомерная (в данном случае двумерная) интерполяция
double mult_interp(record_t *data, size_t rows, size_t cols, double x, double y, int nx, int ny)
{
    // выясняем, по каким строкам и столбцам располагаются ближайшие к
    // аргументам значения, чтобы по ним интерполировать табличную функцию
    size_t start_y = get_y_config(data, rows, cols, y, ny);
    size_t start_x = get_x_config(data, cols, x, nx);

    // в этом массиве будут записываться значения z(x, y_i) для фиксированных
    // значений y
    record_t *func_y_fixed = malloc((ny + 1) * sizeof(record_t));

    for (size_t i = start_y; i <= ny + start_y; i++)
    {
        func_y_fixed[i - start_y].y = data[i * cols].y;
        // интерполяция по строке для фиксированного y
        func_y_fixed[i - start_y].z = interp_newton(data, cols, x, start_x, nx, i);
    }

    // после интерполяции по строкам интерполируем по полученным значениям
    // переменной y
    double z = interp_newton(func_y_fixed, ny + 1, y, 0, ny, -1);
    free(func_y_fixed);

    return z;
}
```