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 №4.

«Построение и программная реализация алгоритма наилучшего среднеквадратичного приближения»

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 метода наименьших квадратов с использованием полинома заданной степени при аппроксимации табличных функций с весами.

Задание

Исходные данные

1. Таблица функции с весами ρ с количеством узлов N. Таблица формируется со случайным разбросом точек. В интерфейсе предусмотрена удобная возможность изменения пользователем весов в таблицу.
2. Степень аппроксимирующего полинома n.

Требуемый результат

Графики, на которых изображены точки заданной табличной функции и кривые найденных полиномов.

Краткий алгоритм

Таблица функции хранится как массив записей с полями x, y, ρ(rho). Для степени полинома n вычисляются коэффициенты СЛАУ порядка n+1:

$\sum_{m=0}^{n}(\varphi_k,\varphi_m)a_m=(y,\varphi_k),\quad 0\le k\le n$. $(f,\varphi)=\sum_{i=1}^{N}\rho_i f(x_i)\varphi(x_i),\ \rho_i>0$. , где - скалярное произведение таблично заданных функций. В качестве функции φ_k используется функция x^k. Решив СЛАУ методом Гаусса, получаем коэффициенты аппроксимирующего полинома степени n, график которого затем строится (генерируется и выполняется скрипт для утилиты gnuplot) и сохраняется в файле.

Полученный результат

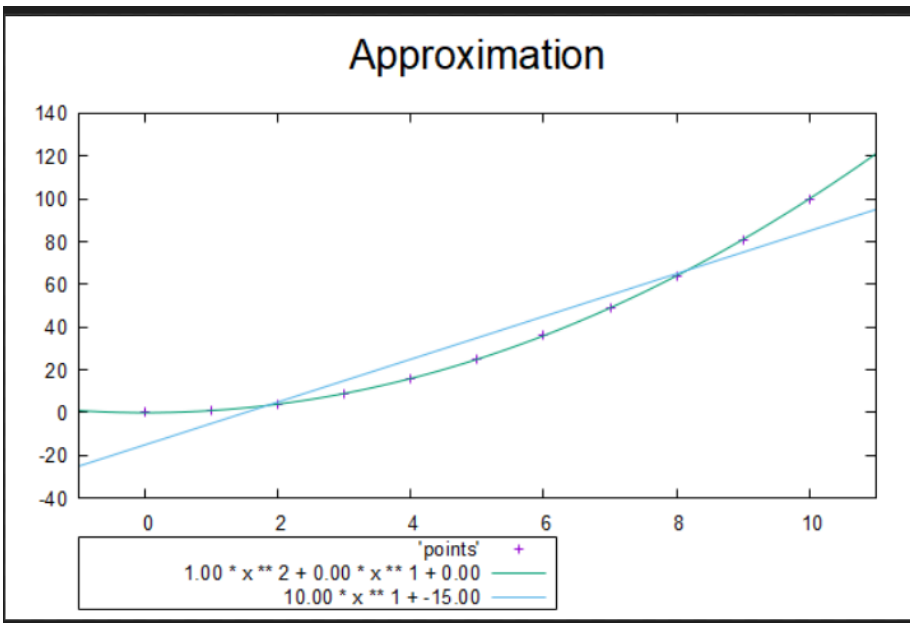
1. Веса всех точек одинаковы и равны единице.

Таблица:

Current function:

i	x	y	rho
0	0.00	0.00	1.00
1	1.00	1.00	1.00
2	2.00	4.00	1.00
3	3.00	9.00	1.00
4	4.00	16.00	1.00
5	5.00	25.00	1.00
6	6.00	36.00	1.00
7	7.00	49.00	1.00
8	8.00	64.00	1.00
9	9.00	81.00	1.00
10	10.00	100.00	1.00

Графики полиномов первой и второй степеней:



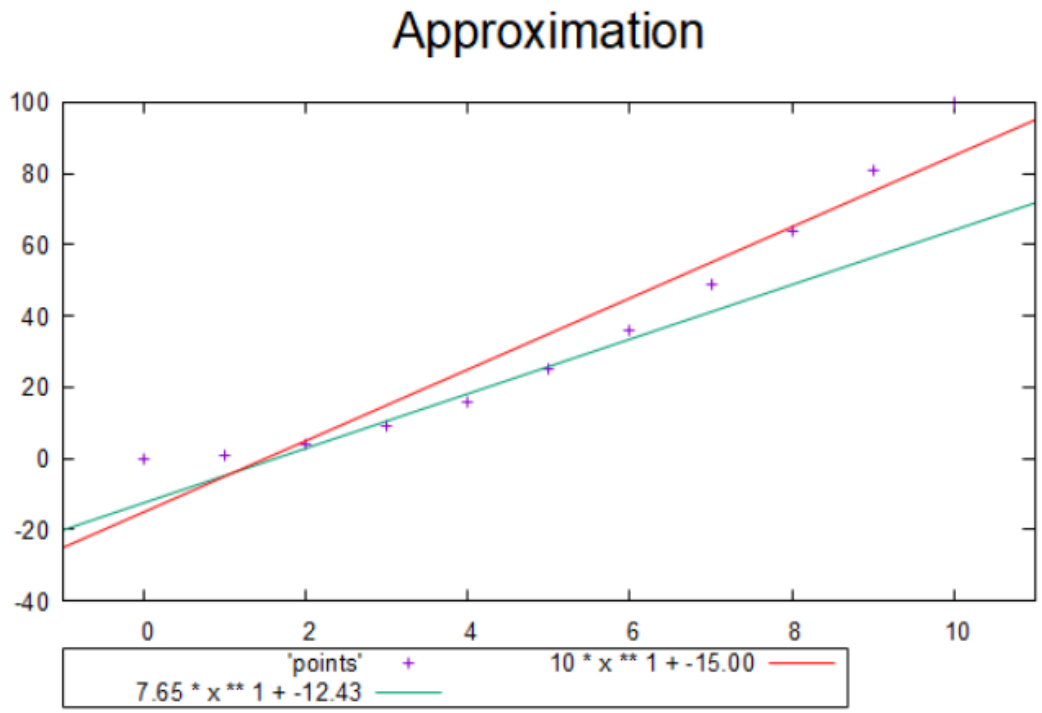
2. Веса точек разные.

Табличная функция используется та же, что и в первом пункте, но распределение весов изменилось:

Current function:

i	x	y	rho
0	0.00	0.00	0.50
1	1.00	1.00	1.00
2	2.00	4.00	2.00
3	3.00	9.00	4.00
4	4.00	16.00	8.00
5	5.00	25.00	3.00
6	6.00	36.00	0.80
7	7.00	49.00	0.40
8	8.00	64.00	0.25
9	9.00	81.00	0.10
10	10.00	100.00	0.01

Тогда при аппроксимации линейная функция изменит угловой коэффициент: красным показана линия при всех весах, равных единице, голубым - линия при неравномерном распределении весов



Ответы на вопросы

1. Что произойдет при задании степени полинома n=N-1 (числу узлов таблицы минус 1)?

На N различных точках можно определить единственный полином степени n-1, который будет проходить через эти точки. Тогда в выражении

$$\sum_{i=1}^{N} \rho_i [y(x_i) - \varphi(x_i)]^2 = min$$

выражение в скобках будет всегда нулевым для любого из N узлов, поэтому вне зависимости от весовой функции, будет построена одна и та же кривая.

2. Будет ли работать Ваша программа при n >= N? Что именно в алгоритме требует отдельного анализа данного случая и может привести к аварийной остановке?

При n >= N уравнения СЛАУ не будут линейно независимыми, что приведет к некорректному решению, так как в этом случае

определитель матрицы СЛАУ равен 0 и решения нет. При решении методом Гаусса это приводит к делению на ноль при приведении к единичной матрице, так как на главной диагонали матрицы коэффициентов СЛАУ найдется нулевой элемент. Лучше всего проверять условие n < N при вводе степени полинома пользователем.

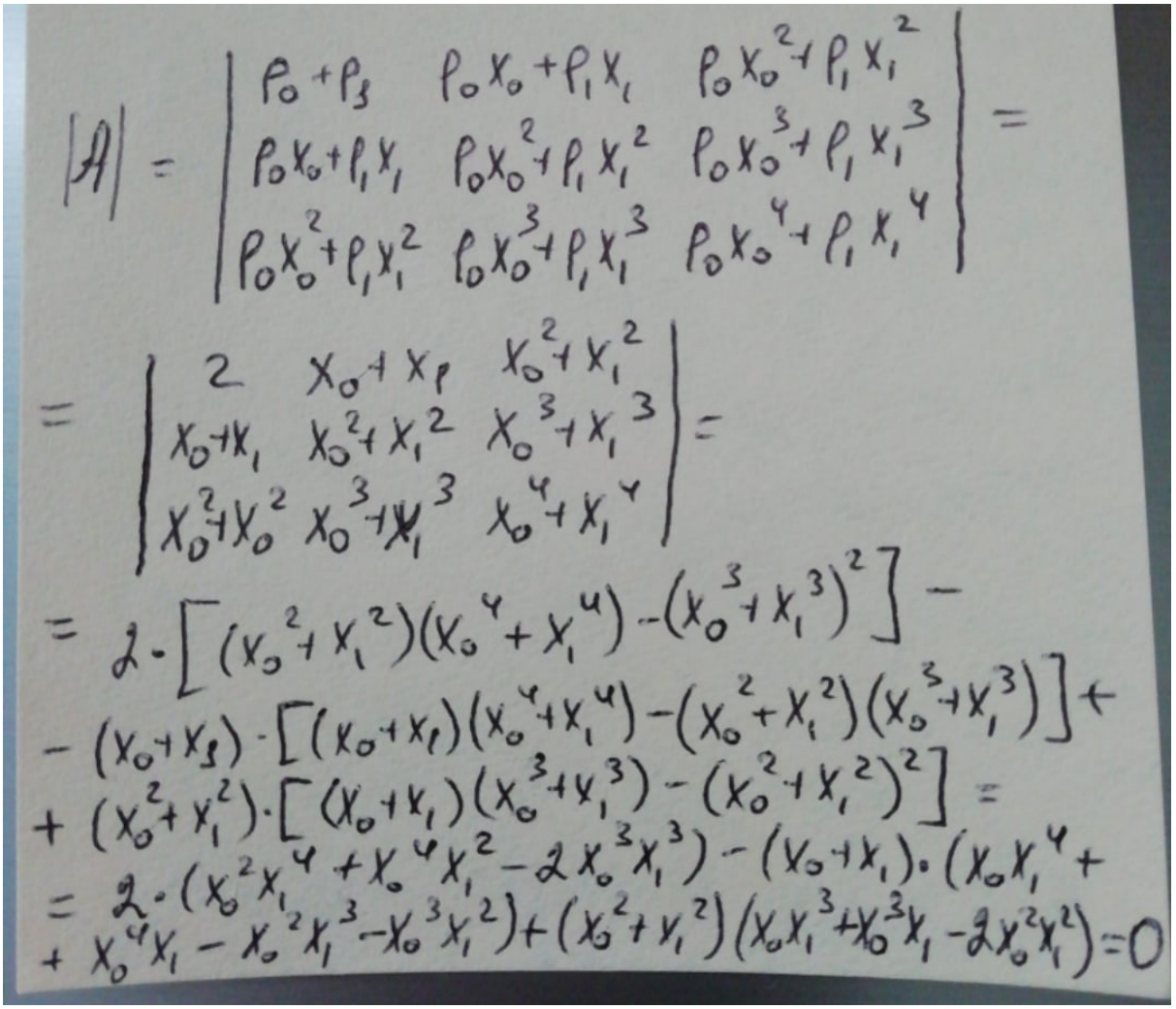
3. Получить формулу для коэффициента полинома a0 при степени полинома n=0. Какой смысл имеет величина, которую представляет данный коэффициент?

Система состоит из одного уравнения с одной неизвестной:

$\sum_{i=0}^{N} \rho_i * a_0 = \sum_{i=0}^{N} \rho_i y_i$, откуда $a_0 = \frac{\sum_{i=0}^{N} \rho_i y_i}{\sum_{i=0}^{N} \rho_i}$. Разделив на сумму весов, получим выражение, равное математическому ожиданию:

$a_0 = \sum_{i=0}^{N} p_i y_i$, где p_i- отношение веса ρ_iк сумме всех весов.

4. Записать и вычислить определитель матрицы СЛАУ для нахождения коэффициентов полинома для случая, когда n=N=2. Принять все ρ_i=1.



5. Построить СЛАУ при выборочном задании степеней аргумента полинома $\varphi(x) = a_0 + a_1 x^m + a_2 x^n$, причем степени n и m в этой формуле известны.

Имеем три линейно независимых функции x^0, x^n, x^m
По ним можно построить следующую СЛАУ:

$$a_0(x^0, x^0) + a_1(x^0, x^m) + a_2(x^0, x^n) = (y, x^0);$$
$$a_0(x^m, x^0) + a_1(x^m, x^m) + a_2(x^m, x^n) = (y, x^m);$$
$$a_0(x^n, x^0) + a_1(x^n, x^m) + a_2(x^n, x^n) = (y, x^n);$$

Выводится из условия $\frac{\partial I}{\partial a_k} = 0$, где $I = ((y - \varphi), (y - \varphi))$

6. Предложить схему алгоритма решения задачи из вопроса 5, если степени n и m подлежат определению наравне с коэффициентами a_k, т.е. количество неизвестных равно 5.

Можно организовать перебор всех возможных комбинаций m и n, которых конечное число (m и n ограничены сверху количеством заданных узлов). Для каждого найти значение I и в итоге выбрать те m и n, при которых значение I будет наименьшим.

Код программы

Основная функция приложения:

```
int main(void)
{
    show_info(); // информация о программе
    size_t size = 0, bufsize = 0;
    record_t *data = NULL, *buf = NULL;
    func_t func;
    do
    {
        func = get_func();
        switch (func)
        {
            case LOAD:
            {
                // получение файла с данными от пользователя
                FILE *file = get_file();
                // считывание табличной функции в массив
                bufsize = 0;
                buf = export_to_array(file, &bufsize);
                fclose(file);

                if (!buf)
                    return EXIT_FAILURE;
                free(data);
                // перенос данных из временного буфера
                data = buf;
                size = bufsize;
```

```
            break;
        }
        case SHOW:
        {
            // вывод таблицы
            output_table(data, size);
            break;
        }
        case ADJUST:
        {
            // изменение веса узла
            size_t index = get_index(size - 1);
            double weight = get_weight();
            change_weight(data, index, weight);
            break;
        }
        case APPROX:
        {
            int n = 0;
            get_degree(&n);    // получение степени полинома от пользователя
            double *coeffs = get_polynom_coeffs(data, size, n);
            plot(coeffs, data, n, size);
            free(coeffs);
            break;
        }
        default: break;
      }
   } while (func);
   free(data);
   return EXIT_SUCCESS;
}
```

Функции, использованные в начинке программы:

```
// решение СЛАУ методом Гаусса: массив Y - правые части ур-ий -> столбец корней
static void solve_slae(double **A, double *Y, int n)
{
   // прямой ход метода Гаусса
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
```

```
        {
            if (i == j)
                continue;
            double mult = A[j][i] / A[i][i];
            for (int k = 0; k < n; k++)
                A[j][k] -= mult * A[i][k];
            Y[j] -= mult * Y[i];
        }
    }

    // обратный ход метода Гаусса
    for (int i = 0; i < n; i++)
    {
        double mult = A[i][i];
        for (int j = 0; j < n; j++)
            A[i][j] /= mult;
        Y[i] /= mult;
    }
}

double *get_polynom_coeffs(record_t *data, size_t size, int n)
{
    n++;
    double **slae_A = malloc(n * sizeof(double *) + n * n * sizeof(double));
    double *slae_Y = malloc(n * sizeof(double));

    if (!slae_A || !slae_Y)
    {
        free(slae_A);
        free(slae_Y);
        return NULL;
    }
    // начальная инициализация динамической матрицы
    double *split = (double *)((char *)slae_A + n * sizeof(double *));
    for (int i = 0; i < n; i++)
        slae_A[i] = split + i * n;

    // вычисление скалярных произведений - элементов матрицы СЛАУ
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
```

```
        {
            slae_A[i][j] = 0;
            for (size_t k = 0; k < size; k++)
            {
                double prod = data[k].rho;
                for (int l = 0; l < i + j; l++)
                    prod *= data[k].x;
                slae_A[i][j] += prod;
            }
        }
    // вычисление скалярных произведений - правых частей СЛАУ
    for (int i = 0; i < n; i++)
    {
        slae_Y[i] = 0;
        for (size_t k = 0; k < size; k++)
        {
            double prod = data[k].rho * data[k].y;
            for (int l = 0; l < i; l++)
                prod *= data[k].x;
            slae_Y[i] += prod;
        }
    }

    solve_slae(slae_A, slae_Y, n);

    free(slae_A);
    return slae_Y;
}
```