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 №3.

«Построение и программная реализация алгоритма сплайн-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 владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

Задание

Исходные данные

1. Таблица функции с количеством узлов N, заданная с помощью формулы y = x^2 в диапазоне [0..10] с шагом 1.

x	y
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

2. Значение аргумента x = 0.5 (в первом интервале) и x = 5.5 (в середине таблицы).

Требуемый результат

Значения y(x), их сравнение с результатом интерполяции полиномом Ньютона 3-й степени и с точным значением.

Краткий алгоритм

Набор узлов хранится в виде массива записей с полями x, y, отсортированного для удобства по возрастанию аргумента x.
Для каждого интервала i от 1 до N вычисляются коэффициенты a[i], b[i], c[i], d[i] кубических сплайнов с помощью метода прогонки. Он состоит из двух этапов:
Прямой ход. Вычисляются по порядку прогоночные коэффициенты до индекса N + 1 включительно, где N - количество интервалов.

$$\xi_{i+1} = -\frac{h_i}{h_{i-1}\xi_i + 2(h_{i-1} + h_i)}, \eta_{i+1} = \frac{f_i - h_{i-1}\eta_i}{h_{i-1}\xi_i + 2(h_{i-1} + h_i)}.$$

, где

$$f_i = 3(\frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}}) .$$

и $h_i = x_i - x_{i-1}, 1 \le i \le N.$

Вычисление начинается с индекса 2, причем xee[2] = etha[2] = 0, исходя из дополнительного условия c[1] = 0
Обратный ход. Вычисляются коэффициенты c[i] по формуле:

$$c_i = \xi_{i+1}c_{i+1} + \eta_{i+1},$$

При этом принимается, что c[N+1] = 0.
Затем рассчитываются и остальные коэффициенты сплайна.

$$a_i = y_{i-1}, 1 \le i \le N,$$

$$b_i = \frac{y_i - y_{i-1}}{h_i} - h_i\frac{c_{i+1} - 2c_i}{3}$$

$$d_i = \frac{c_{i+1} - c_i}{3h_i}$$

Теперь для введенных аргументов x можно рассчитывать значение соответствующего сплайна (если x такой, что x[i-1] < x < x[i], то ему соответствует сплайн с коэффициентами a[i], b[i], c[i], d[i]). Значение сплайна с заданными коэффициентами находится по формуле:

$$\psi(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$

Полученный результат

Алгоритм → Аргумент x ↓	Сплайны	Ньютон	Точное значение
x = 0.5	y = 0.342	y = 0.250	y = 0.250
x = 5.5	y = 30.250	y = 30.250	y = 30.250

Таким образом, в крайнем интервале вычисляемое с помощью сплайнов значение обладает ощутимой погрешностью.

Ответы на вопросы

1. Получить выражения для коэффициентов кубического сплайна, построенного на двух точках.

Две точки -> 1 интервал -> 4 условия:
y(x[0]) = a[1]
y(x[1]) = a[1] + b[1] * h[1] + c[i] * h^2[i] + d[i] * h^3[i]
y''(x[0]) = 0, т.е. c[1] = 0
y''(x[1]) = 0, т.е. c[2] = 0

Тогда:
a = y(x[0]),
b = (y[1] - y[0])/h[1],
c = 0, d = 0.

2. Выписать все условия для определения коэффициентов сплайна, построенного на 3-х точках.
3 точки -> 2 интервала -> 8 условий:

Равенство второй производной нулю на концах отрезка:
1) y''(x[0]) = 0 = c[1]
2) y''(x[2]) = 0 = c[3]
Равенство значений функции и сплайна в узле
3) y(x[0]) = a[1]
4) y(x[1]) = a[1] + b[1]*h[1] + c[1]*h^2[1] + d[1]*h^3[1]

5) y(x[1]) = a[2]
6) y(x[2]) = a[2] + b[2]*h[2] + c[2]*h^2[2] + d[2]*h^3[2]

Равенство первых и вторых производных во внутренних узлах:

7) b[2] = b[1] + 2*c[1]*h[1] + 3*d[1]*h^2[1]
8) c[2] = c[1] + 3*d[1]*h[1]

3. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо C1=C2.

C1 = xee[2]*C2 + etha[2]
C1 = 1*C2 + 0

Очевидно, что xee[2] = 1, etha[2] = 0.

4. Написать формулу для определения последнего коэффициента сплайна C[N] , чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано
k * C[N-1] + m * C[N] = p, где k,m и p - заданные числа.

C[N-1] = xee[N] * C[N] + etha[N] - из метода прогонки
C[N-1] = -m/k * C[N] + p/k - из граничного условия

Следовательно, **xee[N] = -m/k, etha[N] = p/k**

C[N] = xee[N+1] * C[N+1] + etha[N+1] = etha[N+1], т.к. C[N+1] = 0

Тогда, вычислив etha[N+1], получим значение:

C[N] = (F[N] - h[N-1] * p / k) / (2 * (h[N - 1] + h[N]) + h[N - 1] * m / k),

$$f_i = 3(\frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}}) .$$

где $h_i = x_i - x_{i-1}, 1 \le i \le N .$ и

Код программы

Основная функция приложения

```c
int main(void)
{
    show_info(); // информация о программе
    FILE *file = get_file();    // получение файла с данными от пользователя

    double x1, x2, y1, y2;
    get_arguments(&x1, &x2);    // получение значений аргументов от пользователя

    size_t size = 0;   // size - количество узлов
    // считывание табличной функции в массив
    record_t *data = export_to_array(file, &size);
    fclose(file);

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

    // сортировка массива записей для удобства
    sort(data, size);

    // вычисление коэффициентов кубических сплайнов
    spline_t *splines = calc_splines(data, size);

    if (!splines)
        return EXIT_FAILURE; // выход, если ошибка выделения памяти

    // нахождение значения функции как значения соответствующего сплайна
    y1 = get_spline_value(splines, data, size, x1);
    y2 = get_spline_value(splines, data, size, x2);

    printf("%.3lf %.3lf", y1, y2);
    free(splines);
    free(data);
    return EXIT_SUCCESS;
}
```

Функции, использованные в начинке программы:

```
// прямой ход метода прогонки - вычисление прогоночных коэффициентов
static void forward_stroke(record_t *data, double *xee, double *etha, size_t size)
{
   // первые два элемента не используются - для удобства чтения алгоритма
   xee[2] = etha[2] = 0;  // начальные значения коэффициентов

   // вычисление коэффициентов с 3 по N + 1 = size
   for (size_t i = 2; i < size; i++)
   {
      double D_i = data[i].x - data[i - 1].x;
      double B_i = 2 * (data[i - 2].x - data[i].x);
      double A_i = data[i - 1].x - data[i - 2].x;
      double F_i = -3 * ((data[i].y - data[i - 1].y) / D_i - \
                  (data[i - 1].y - data[i - 2].y) / A_i);

      xee[i + 1] = D_i / (B_i - A_i * xee[i]);
      etha[i + 1] = (A_i * etha[i] + F_i) / (B_i - A_i * xee[i]);
   }
}

// обратный ход метода прогонки - вычисление коэффициентов сплайнов
static void reverse_stroke(spline_t *splines, record_t *data, double *xee,
double *etha, size_t size)
{
   // 0-й элемент массива сплайнов не используется для удобства чтения

   splines[size].c = 0;   // size = N + 1, N - количество интервалов
   for (size_t i = size - 1; i >= 1; i--)
   {
      splines[i].c = xee[i + 1] * splines[i + 1].c + etha[i + 1];
      splines[i].a = data[i - 1].y;
      double h = data[i].x - data[i - 1].x;
      splines[i].b = (data[i].y - data[i - 1].y) / h - \
                  h / 3.0 * (2 * splines[i].c + splines[i + 1].c);
      splines[i].d = (splines[i + 1].c - splines[i].c) / h / 3.0;
   }
}
```

```
// расчет коэффициентов для сплайнов
spline_t *calc_splines(record_t *data, size_t size)
{
    // spline_t - структура с полями a, b, c, d - коэффициентами сплайна
    spline_t *splines = malloc((size + 1) * sizeof(spline_t));

    // прогоночные коэффициенты
    double *xee = malloc((size + 1) * sizeof(double));
    double *etha = malloc((size + 1) * sizeof(double));

    if (!xee || !etha || !splines)
    {
        free(xee);
        free(etha);
        free(splines);
        return NULL;
    }

    // метод прогонки
    forward_stroke(data, xee, etha, size);
    reverse_stroke(splines, data, xee, etha, size);

    free(etha);
    free(xee);
    return splines;
}

// получение значения сплайна, соответствующего данному аргументу
double get_spline_value(spline_t *splines, record_t *data, size_t size, double x)
{
    size_t i;
    for (i = 1; i < size - 1 && x > data[i].x; i++);

    double y = splines[i].a;
    y += splines[i].b * (x - data[i - 1].x);
    y += splines[i].c * (x - data[i - 1].x) * (x - data[i - 1].x);
    y += splines[i].d * (x - data[i - 1].x) * (x - data[i - 1].x) \
                                    * (x - data[i - 1].x);
    return y;
}
```