Белорусский Государственный Университет

Методы Численного Анализа

Лабораторная работа 3

Интерполирование на равностоящих узлах

Рак Алексей

25 мая 2017 г.

Постановка Задачи

Для заданной функции $f(x)$ на равномерной сетке узлов построить интерполяционный многочлен Ньютона $P_n(x)$ для интерполирования в начале таблицы и вычислить x^*, в конце таблицы и вычислить x^{***}, оценить разультат, сравнить с предыдущими результатами и указать степень конечной разности достаточной для обеспечения точности e^{-6}.

Данные

Функция: $f(x) = 1.3 * e^x - 0.3 * sin(x)$
Равномерная сетка узлов: $[1, 2]$
Шаг 0.1
$x^* = \frac{31}{30},\ x^{***} = \frac{59}{30}$

Алгоритм решения и формулы

Пусть x близка к x_0. Положим $x = x_0 + th$. Узлы интерполирвоания расположим в следующем порядке: $x_0, x_1, \ldots, x_n$:

$$P_n(x) = f(x_0) + (x - x_0)f(x_0, x_1) + \cdots + (x - x_0)(x - x_1)\ldots(x - x_{n-1})f(x_0, x_1, \ldots, x_n)$$

$$\begin{aligned}
&x - x_0 = th && f(x_0) = y_0 \\
&x - x_1 = (x - x_0) + (x_0 - x_1) = (t - 1)h && f(x_0, x_1) = \frac{\Delta y_0}{1!h} \\
&\ldots && \ldots \\
&x - x_{n-1} = (x - x_{n-2}) + (x_{n-2} - x_{n-1}) = (t - n + 1)h && f(x_0, x_1, \ldots, x_n) = \frac{\Delta^n y_0}{n!h^n}
\end{aligned}$$

Многочлен примет вид:

$$P_n(x_0 + th) = y_0 + t\frac{\Delta y_0}{1!} + t(t-1)\frac{\Delta^2 y_0}{2!} + \cdots + t(t-1)\ldots(t-n+1)\frac{\Delta^n y_0}{n!} \tag{1}$$

(1) - многочлен Ньютона для интерполирования в начале таблицы.

$$R_n(x_0 + th) = \frac{f^{(n+1)}(\xi)}{(n+1)!}t(t-1)\ldots(t-n)h^{n+1}$$

Пусть x близка к x_n. Положим $x = x_n + th$. Узлы интерполирования расположим в следующем порядке: $x_n, x_{n-1}, \ldots, x_0$.

$$P_n(x) = f(x_n) + (x - x_n)f(x_n, x_{n-1}) + \cdots + (x - x_n)(x - x_{n-1})\ldots(x - x_1)f(x_n, x_{n-1}, \ldots, x_0)$$

$$\begin{aligned}
&x - x_n = th && f(x_n) = y_n \\
&x - x_{n-1} = (x - x_n) + (x_n - x_{n-1}) = (t + 1)h && f(x_n, x_{n-1}) = \frac{\Delta y_{n-1}}{1!h} \\
&\ldots && \ldots \\
&x - x_1 = (x - x_2) + x_2 - x_1) = (t + n - 1)h && f(x_n, x_{n-1}, \ldots, x_0) = \frac{\Delta^n y_0}{n!h^n}
\end{aligned}$$

Многочлен примет вид:

$$P_n(x_n - th) = y_n + t\frac{\Delta y_{n-1}}{1!} + t(t+1)\frac{\Delta^2 y_{n-2}}{2!} + \cdots + t(t+1)\ldots(t+n-1)\frac{\Delta^n y_0}{n!} \tag{2}$$

(2) - многочлен Ньютона для интерполирования в конце таблицы.

$$R_n(x_n + th) = \frac{f^{(n+1)}(\xi)}{(n+1)!}t(t+1)\ldots(t+n)h^{n+1}$$

Листинг

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

const int n = 11;
const double alpha = 1.3;
const double x1 = 1.03333333333;
const double x3 = 1.96666666667;
std::vector<double> nodes;
const double x0 = 1;
const double step = 0.1;
std::vector<std::vector<double» konechRazn;
const double accuracy = 1e-6;
const double df11max = alpha * exp(2) - (1 - alpha) * cos(1);

double f(double x) {
    return alpha * exp(x) + (1 - alpha) * sin(x);
}

static double f1(double x) {
    return alpha * exp(x) + (1 - alpha) * cos(x);
}

static double f2(double x) {
    return alpha * exp(x) - (1 - alpha) * sin(x);
}

static void makeNodes() {
    nodes.resize(n);
    for (int i = 0; i < n; i++) {
        nodes[i] = x0 + i * (2 - 1) / 10.;
    }
}

static void makeKR() {
    konechRazn.resize(n);
    for (int i = 0; i < n; i++) {
        konechRazn[i].resize(n);
        konechRazn[i][0] = f(nodes[i]);
    }
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            konechRazn[j][i] = konechRazn[j + 1][i - 1] - konechRazn[j][i - 1];
        }
    }
}

static int factorial(int x) {
    if (x == 1 || x == 0)
        return 1;
    return x * factorial(x - 1);
}

double begin(double x) {
    double t = (x - nodes[0]) / step;
    double answer = 0, k = 1;
    for (int i = 0; i < n; i++) {
        answer += konechRazn[0][i] * k;
        k *= (t - i) / (i + 1);
    }
```

```
    return answer;
}

double end(double x) {
    double t = (x - nodes[n - 1]) / step;
    double answer = 0, k = 1;
    for (int i = 0; i < n; i++) {
        answer += konechRazn[n - 1 - i][i] * k;
        k *= (t + i) / (i + 1);
    }
    return answer;
}

int beginAndAc(double x) {
    double t = (x - nodes[0]) / step;
    double answer = 0, k = 1;
    for (int i = 0; i < n; i++) {
        answer += konechRazn[0][i] * k;
        k *= (t - i) / (i + 1);
        if (fabs(answer - f(x)) < accuracy)
            return i;
    }
    return n;
}

int endAndAc(double x) {
    double t = (x - nodes[n - 1]) / step;
    double answer = 0, k = 1;
    for (int i = 0; i < n; i++) {
        answer += konechRazn[n - 1 - i][i] * k;
        k *= (t + i) / (i + 1);
        if (fabs(answer - f(x)) < accuracy)
            return i;
    }
    return n;
}

double expectedEnd(double x) {
    double t = (x - nodes[n - 1]) / step;
    double k = 1;
    for (int i = 0; i < n; i++)
        k *= (t + i) / (i + 1);
    return pow(step, n) * k * df11max;

}

double expectedBegin(double x) {
    double t = (x - nodes[0]) / step, k = 1;
    for (int i = 0; i < n; i++)
        k *= (t - i) / (i + 1);
    return pow(step, n) * k * df11max;
}

static void print() {
    std::cout « "x*: " « std:: endl « "resalt: " « begin(x1) « std::endl;
    std::cout « "expected: " « expectedBegin(x1) « std::endl;
    std::cout « "true: " « fabs(begin(x1) - f(x1)) « std::endl;
    std::cout « "degree with <= E-6: " « beginAndAc(x1) « std::endl;
    std::cout « "x***: " « std::endl « "resalt: " « end(x3) « std::endl;
    std::cout « "expected: " « expectedEnd(x3) « std::endl;
    std::cout « "true: " « fabs(end(x3) - f(x3)) « std::endl;
    std::cout « "degree with <= E-6: " « endAndAc(x3) « std::endl;
```

```
}

int main() {
    makeNodes();
    makeKR();
    print();
    return 0;
}
```

Результаты и вывод

Выходные данные:
$P_n(x^*) = 3.39584$
$P_n(x^{***}) = 9.01406$
$R_n(x^*) \leq 1.00346e^{-12}$
$R_n(x^{***}) \leq 1.00346e^{-12}$
$R_{true}(x^*) = 5.8753e^{-13}$
$R_{true}(x^{***}) = 6.18172e^{-13}$
Вывод:
Модификация многочлена Ньютона для равномерной сетки не дала выигрыша в точности: и для x^*, и для x^{***} мы получили точности того же порядка, что и без неё. Если учитывать тот факт, что модификации проводились как для конца, так и для начала таблицы, а исходный многочлен был построен в единственном экземпляре и дал сравнимые результаты, то можно сказать, что при необходимости нахождения максимально точного значения данная модификация не даёт достаточного выигрыша, однако она удобна, если требуется найти значения с наперёд заданной точностью.