Белорусский Государственный Университет

Методы Численного Анализа

Лабораторная работа 5

Интерполяционный многочлен Эрмита

Рак Алексей

1 июня 2017 г.

Постановка Задачи

Для заданной функции $f(x)$ построить интерполяционный многочлен Эрмита $P_n(x)$ в узлах $1.0, 1.5, 2.0$ (каждый из них кратности 2) для интерполирования заданной функции на отрезке $[1, 2]$. Указать погрешность, сравнить с предыдущими методами интерполирования.

Данные

Функция: $f(x) = 1.3 * e^x - 0.3 * sin(x)$
Значения функции, её первой и второй производных в точках $1.0, 1.5, 2.0$
$x^* = \frac{31}{30}$, $x^{**} = \frac{46}{30}$ $x^{***} = \frac{59}{30}$

Алгоритм решения и формулы

Пусть на отрезке $[a, b]$ даны m различных узлов интерполирования. Рассмотрим функцию $f(x)$ и будем считать что в точке x_1 заданы значения самой функции и $\alpha_1 - 1$ её первых производных, в точке x_2 аналогично значение самой функции и $\alpha_2 - 2$ её первых производных и т.д. Обозначим через $n + 1 = \alpha_1 + \alpha_2 + \cdots + \alpha_m$. Задача кратного интерполирования заключается в построении многочлена $P_n(x, x_0, \ldots, x_0, x_1, \ldots, x_1, \ldots, x_n, \ldots, x_n)$ по данным значениям. Слиянием нескольких узлов в один обеспечивают передачу не только значения функции в точке, но и наклона функции в этой точке (при передаче первой производной), кривизны (второй производно) и т.д. Оценка погрешности принимает вид:

$$\|f(x) - h_n(x)\| \le \frac{M_{n+1}}{(n+1)!}\|\Sigma_n(x)\|$$

$$M_{n+1} = \|f^{(n+1)}(\xi)\|$$

$$\Sigma_n(x) = \prod_{k=0}^{p}(x - x_k)^{\alpha_k}$$

Разделённая разность из α одинаковых элементов представляется в виде:

$$f(\underbrace{x_0, x_0, \ldots, x_0}_{\alpha}) = \frac{f^{(\alpha-1)}(x_0)}{(\alpha-1)!}$$

Остальные разделённые разности считаются также как и ранее.
Общее выражение интерполяционного многочлена Эрмита:

$$H_n(x) = \sum_{k=0}^{p}\sum_{m=0}^{\alpha_{k-1}}\sum_{q=0}^{a_{k-1}-m}\frac{f^{(m)}(x_k)}{m!q!}\left((x - x_k)^{m+q}\prod_{i\ne k}(x - x_i)^{a_i}\right)\left(\frac{d^q}{dx^q}\prod_{i\ne k}(x - x_j)^{-\alpha_i}\right)_{x=x_k}$$

В частном случае для трёх узлов кратности 2:

$$P_n(x) = f(x_0) + (x - x_0)f(x_0, x_0) + (x - x_0)^2 f(x_0, x_0, x_1) + (x - x_0)^2(x - x_1)f(x_0, x_0, x_1, x_1) +$$
$$+(x - x_0)^2(x - x_1)^2 f(x_0, x_0, x_1, x_1, x_2) + (x - x_0)^2(x - x_1)^2(x - x_2)f(x_0, x_0, x_1, x_1, x_2, x_2)$$

Листинг

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

const double a = 1.3;
double df9max = a * exp(2.0) + (1 - a) * cos(2);

class Interpolation {
```

```
public:
    Interpolation(const std::vector<std::vector<double>> &function) :
            divided_diff_(10, std::vector<double>(9)) {
        for (int i = 0; i < 9; ++i) {
            divided_diff_[0][i] = function[0][i / 3];
            divided_diff_[1][i] = function[1][i / 3];
        }
        for (int i = 1; i < 9; ++i) {
            if (i % 3 == 0) {
                divided_diff_[2][i] = (divided_diff_[1][i] - divided_diff_[1][i - 1]) /
                                      (divided_diff_[0][i] - divided_diff_[0][i - 1]);
            } else {
                divided_diff_[2][i] = function[2][i / 3];
            }
        }
        for (int i = 2; i < 9; ++i) {
            if (i % 3 != 2) {
                divided_diff_[3][i] = (divided_diff_[2][i] - divided_diff_[2][i - 1]) /
                                      (divided_diff_[0][i] - divided_diff_[0][i - 2]);
            } else {
                divided_diff_[3][i] = function[3][i / 3] / 2;
            }
        }
        for (int i = 4; i < 10; ++i) {
            for (int j = i - 1; j < 9; ++j) {
                divided_diff_[i][j] = (divided_diff_[i - 1][j] - divided_diff_[i - 1][j - 1]) /
                                      (divided_diff_[0][j] - divided_diff_[0][j - i + 1]);
            }
        }
    }

    double value(double point) {
        double res = 0;
        double k = 1;
        for (int i = 1; i < 10; ++i) {
            res += k * divided_diff_[i][i - 1];
            k *= (point - divided_diff_[0][i - 1]);
        }
        return res;
    }

private:
    std::vector<std::vector<double>> divided_diff_;
};

double func(double x) {
    return a * exp(x) + (1 - a) * sin(x);
}

double grad(double x) {
    return a * exp(x) + (1 - a) * cos(x);
}

double ges(double x) {
    return a * exp(x) - (1 - a) * sin(x);
}

double calcRn(double x) {
    double ans = df9max;
    double xk = 1.0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; ++j) {
            ans = ans * fabs(xk - x) / (i * 3 + j + 1);
```

```
        }
        xk += 0.5;
    }
    return ans;
}

int main() {
    double a  = 1.3;
    std::vector<std::vector<double>> function{{1.0, 1.5, 2.0},
                                              {func(1.0), func(1.5), func(2.0)},
                                              {grad(1.0), grad(1.5), grad(2.0)},
                                              {ges(1.0), ges(1.5), ges(2.0)}};
    Interpolation interpolation(function);
    double x1 = 31.0 / 30;
    double x2 = 46.0 / 30;
    double x3 = 59.0 / 30;
    std::cout << "X1=" << interpolation.value(x1) << std::endl;
    std::cout << "X2=" << interpolation.value(x2) << std::endl;
    std::cout << "X3=" << interpolation.value(x3) << std::endl;
    std::cout << "RX1=" << calcRn(x1) << std::endl;
    std::cout << "RX2=" << calcRn(x2) << std::endl;
    std::cout << "RX3=" << calcRn(x3) << std::endl;
    std::cout << "RX1True=" << fabs(interpolation.value(x1) - func(x1)) << std::endl;
    std::cout << "RX2True=" << fabs(interpolation.value(x2) - func(x2)) << std::endl;
    std::cout << "RX3True=" << fabs(interpolation.value(x3) - func(x3)) << std::endl;
    return 0;
}
```

Результат

Выходные данные:
Значения функции в заданных точках:
$P(x^*) = 3.39584$
$P(x^{**}) = 5.72389$
$P(x^{***}) = 9.01406$
Максимальная погрешность:
$R_n(x^*) = 2.97744e^{-6}$
$R_n(x^{**}) = 9.06331e^{-7}$
$R_n(x^{***}) = 2.97744e^{-6}$
Настоящая погрешность:
$R(x^*) = 1.8224e^{-6}$
$R(x^{**}) = 5.92993e^{-7}$
$R(x^{***}) = 2.06969e^{-6}$

Вывод

Погрешность стала больше по сравнению с предыдущими методами из-за того, что ранее рассматривались интерполяционные многочлены 10-ой степени, а в данном задании используется многочлен 5-ой степени. Как и ранее наименьшая погрешность у точки находящейся наиболее близко к центральному узлу.