

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕТОДЫ ЧИСЛЕННОГО АНАЛИЗА

ЛАБОРАТОРНАЯ РАБОТА 2

ИНТЕРПОЛЯЦИОННЫЙ МНОГОЧЛЕН НЬЮТОНА

Рак Алексей

25 мая 2017 г.

Постановка Задачи

Для заданной функции $f(x)$ на равномерной сетке узлов построить интерполяционный многочлен Ньютона $P_n(x)$ и вычислить:

- 1) $P_n(x^*), P_n(x^{**}), P_n(x^{***})$
- 2) Погрешность $R_n(x^*), R_n(x^{**}), R_n(x^{***})$
- 3) Абсолютные погрешности $R_{true}(x^*), R_{true}(x^{**}), R_{true}(x^{***})$

Данные

Функция: $f(x) = 1.3 * e^x - 0.3 * \sin(x)$

Равномерная сетка узлов: $[1, 2]$

Шаг 0.1

$x^* = \frac{31}{30}, x^{**} = \frac{46}{30}, x^{***} = \frac{59}{30}$

Алгоритмы решения и формулы

Разностными отношениями нулевого порядка называются значения функции в узлах сетки. Разностными отношениями первого порядка называются величины $f(x_i, x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$. Разностные отношения порядка $n + 1, n = 1, 2, \dots$ определяются при помощи разностных отношений предыдущего n -го порядка по формуле:

$$f(x_0, x_1, \dots, x_{n+1}) = \frac{f(x_1, x_2, \dots, x_{n+1}) - f(x_0, x_1, \dots, x_n)}{x_{n+1} - x_0}$$

Таблицу вида:

$$\begin{array}{ccccccc} x_0 & f(x_0) & & & & & \\ x_1 & f(x_1) & f(x_0, x_1) & & & & \\ x_2 & f(x_2) & f(x_1, x_2) & \dots & & & \\ \dots & \dots & \dots & \dots & & & \\ x_n & f(x_n) & f(x_{n-1}, x_n) & \dots & f(x_0, x_1, \dots, x_n) & & \end{array} \quad (1)$$

называют таблицей разделённых разностей.

Будем рассматривать обычную задачу алгебраического интерполирования функции $f(x)$ по её значениям в узлах $x_i, i = 0, 1, \dots, n$ и пусть $P(x)$ - многочлен n -ой степени, значения которого в узлах интерполирования совпадают со значениями $f(x)$. Пусть x - произвольная точка, отличная от узлов интерполирования. Запишем следующую цепочку тождеств:

$$\begin{aligned} P(x) &= P(x_0) + (x - x_0)P(x, x_0) \\ P(x, x_0) &= P(x_0, x_1) + (x - x_1)P(x, x_0, x_1) \\ &\dots \end{aligned}$$

Эта цепочка соотношений конечна, так как разделённая разность $n + 1$ -го порядка от многочлена n -ой степени равна нулю.

Выражая $P(x)$ через разделённые разности и учитывая то, что в узлах значения $P(x)$ и $f(x)$ совпадают, получаем приближённую формулу:

$$f(x) \approx f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + \dots + (x - x_0)(x - x_1) \dots (x - x_{n-1})f(x_0, x_1, \dots, x_n)$$

Оценка погрешности:

$$R_n(x) \leq w(x)P(x, x_0, \dots, x_n)$$

Листинг

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

```

#include <iostream>

const int n = 11;
const double alpha = 1.3;

double func(double x, std::vector<std::vector<double>> dif) {
    double answer = 0;
    for (int i = 0; i < n; i++) {
        double add = dif[0][i + 1];
        for (int j = 0; j < i; j++) {
            add *= (x - dif[j][0]);
        }
        answer += add;
    }
    return answer;
}

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

void makeDif(std::vector<std::vector<double>> &dif, int _n) {
    for (int j = 2; j < _n + 1; j++) {
        for (int i = 0; i < _n - j + 1; i++) {
            dif[i][j] = (dif[i + 1][j - 1] - dif[i][j - 1]) / (dif[i + j - 1][0] - dif[i][0]);
        }
    }
}

double w(double x, std::vector<double> _x) {
    double answer = 1;
    for (int i = 0; i < n; i++) {
        answer *= (x - _x[i]);
    }
    return answer;
}

int main() {
    std::vector<double> x(n);
    x[0] = 1.0;
    for (int i = 1; i < n; i++) {
        x[i] = x[i - 1] + 0.1;
    }
    double x1 = 31.0 / 30, x2 = 46.0 / 30, x3 = 59.0 / 30;
    std::vector<std::vector<double>> dif(n + 1, std::vector<double>(n + 2));
    for (int i = 0; i < n; i++) {
        dif[i][0] = x[i];
        dif[i][1] = f(x[i]);
    }
    makeDif(dif, n);
    std::cout << "f(x1) = " << std::setprecision(15) << std::fixed << func(x1, dif) << std::endl; //
    std::cout << "f(x2) = " << std::setprecision(15) << std::fixed << func(x2, dif) << std::endl;
    std::cout << "f(x3) = " << std::setprecision(15) << std::fixed << func(x3, dif) << std::endl
    << std::endl;
    dif[n][0] = x1;
    dif[n][1] = f(x1);
    makeDif(dif, n + 1);
    std::cout << "Погрешность в x1 = " << std::setprecision(15) << std::fixed
    << dif[0][n + 1] * w(x1, x) << std::endl;
    std::cout << "Истинная погрешность в x1 = " << std::setprecision(15) << std::fixed
    << func(x1, dif) - f(x1) << std::endl << std::endl;
    dif[n][0] = x2;
    dif[n][1] = f(x2);
}

```

```

makeDif(dif, n + 1);
std::cout << "Погрешность в x2 = " << std::setprecision(15) << std::fixed
    << dif[0][n + 1] * w(x2, x) << std::endl;
std::cout << "Истинная погрешность в x2 = " << std::setprecision(15) << std::fixed
    << func(x2, dif) - f(x2) << std::endl << std::endl;
dif[n][0] = x3;
dif[n][1] = f(x3);
makeDif(dif, n + 1);
std::cout << "Погрешность в x3 = " << std::setprecision(15) << std::fixed
    << dif[0][n + 1] * w(x3, x) << std::endl;
std::cout << "Истинная погрешность в x3 = " << std::setprecision(15) << std::fixed
    << func(x3, dif) - f(x3) << std::endl;
return 0;
}

```

Результаты и вывод

Выходные данные:

$$P_n(x^*) = 3.395840703657075$$

$$P_n(x^{**}) = 5.723885855903852$$

$$P_n(x^{***}) = 9.014059843851850$$

$$R_n(x^*) \leq 5.83e^{-13}$$

$$R_n(x^{**}) \leq 7e^{-15}$$

$$R_n(x^{***}) \leq 6.30e^{-13}$$

$$R_{true}(x^*) = 5.84e^{-13}$$

$$R_{true}(x^{**}) = 8e^{-15}$$

$$R_{true}(x^{***}) = 6.29e^{-13}$$

Вывод:

Метод Ньютона нахождения интерполяционного многочлена позволяет достаточно точно интерполировать функцию, при этом абсолютная погрешность в рассматриваемых точках не превосходит $7e^{-13}$. Наиболее точные значения получаются вблизи центрального узла таблицы. Метод Ньютона даёт точность примерно 1.5 раза больше при такой же сложности, что и метод Лагранжа.