Белорусский Государственный Университет

Методы Численного Анализа

Лабораторная работа 7

Методы наименьших квадратов построения элемента наилучшего приближения

Рак Алексей

28 мая 2017 г.

Постановка Задачи

Построить элемент наилучшего приближения $\phi(x)$ степени $n = 5$ методом наименьших квадратов по равномерной сетке узлов $x_i \in [1, 2]$, $x_i = 1 + ih$, $h = \frac{2-1}{m}$, $i = \overline{0, m}$, $m = 10$ для функции $f(x) = ae^x + (1 - a)sinx$, где $a = 1.3$ используя полиномиальную аппроксимацию, т.е. $\phi(x) = x^i, i = \overline{0, n}$. Рассчитать истинную погрешность в точках:

$$x^* = x_0 + \frac{h}{3}$$
$$x^{**} = x_5 + \frac{h}{3}$$
$$x^{***} = x_{10} - \frac{h}{3}$$

Алгоритм решения и формулы

1) Для определения коэффициентов полинома $\alpha_i, i = \overline{0, n}$ составим систему:

$$\sum_{i=0}^{n} \alpha_i (g_i, g_j) = (f, g_j), j = \overline{0, m}$$

где $(g_i, g_j) = \sum_{k=0}^{m} x_k^{i+j}, (f, g_j) = \sum_{k=0}^{m} x_k^j f(x_k)$
Данную систему решим методом квадратного корня, т.к. матрица системы является симметрической.
2) Элемент наилучшего приближения определим по формуле:

$$\phi(x) = \sum_{i=0}^{n} \alpha_i x^i$$

3) Посчитаем значения элемента наилучшего приближения в узлах
4) Для точек x^*, x^{**}, x^{***} считаем истинную погрешность:

$$R_{true}(x) = |f(x) - \phi(x)|$$

Листинг

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

double KOEF = 1.3;
int POINTS_AMOUNT = 11;
double BEGIN = 1;
double END = 2;
int POWER = 5;
double WEIGHT = 1;
double STEP = (END - BEGIN) / (POINTS_AMOUNT - 1);
std::vector<double> CHECK_POINTS{
        BEGIN + STEP * 0 + STEP / 3.0,
        BEGIN + STEP * 5 + STEP / 3.0,
        BEGIN + STEP * 10 - STEP / 3.0
};

double f(double x){
    return (KOEF * pow(exp(1), x) + (1 - KOEF) * sin(x));
}

static double q(double x, const std::vector<double>& a){
    double res = 0;
```

```
    for (int i = 0; i <= POWER; i++){

        res += pow(x, i) * a[i];

    }
    return res;
}

std::vector<std::vector<double» straightProcess(std::vector<std::vector<double» matrix, int n){
    double maximum;
    int ind;
    for (int i = 0; i < n - 1; i++){
        maximum = matrix[i][i];
        ind = i;
        for (int j = i + 1; j < n; j++){
            if (maximum < matrix[j][i]){
                ind = j;
                maximum = matrix[j][i];
            }
        }
        double buf;
        for (int j = 0; j <= n; j++){
            buf = matrix[i][j];
            matrix[i][j] = matrix[ind][j];
            matrix[ind][j] = buf;
        }
        for (int j = i + 1; j < n; j++){
            for (int k = i + 1; k <= n; k++){
                matrix[j][k] -= matrix[i][k] * (matrix[j][i] / matrix[i][i]);
            }
        }
    }
    return matrix;
}

std::vector<double> reversedProcess(std::vector<std::vector<double» matrix, int n){
    std::vector<double> a(n);
    for (int i = n - 1; i > 0; i-){
        matrix[i][n] /= matrix[i][i];
        a[i] = matrix[i][n];
        for (int j = i - 1; j > -1; j-){
            matrix[j][n] -= matrix[i][n] * matrix[j][i];
        }
    }
    a[0] = matrix[0][n] / matrix[0][0];
    return a;
}

std::vector<double> gaus(std::vector<std::vector<double» matrix, int n){
    return reversedProcess(straightProcess(matrix, n), n);
}

std::vector<double> approximation(const std::vector<double>& a){
    std::vector<double> r(3);
    for (int i = 0; i < 3; i++){

        r[i] = fabs(f(CHECK_POINTS[i]) - q(CHECK_POINTS[i], a));

    }
    return r;
}

void output(const std::vector<double>& a, const std::vector<double>& r){
```

```
    for (int i = 0; i < POINTS_AMOUNT; i++){
        std::cout « q(BEGIN + STEP * i, a) « " ";
    }
    std::cout « std::endl;
    for (int i = 0; i <= POWER; i++){
        std::cout « a[i] « " ";
    }
    std::cout « std::endl « "f:" « std::endl;
    for (int i = 0; i < 3; i++){
        std::cout « f(CHECK_POINTS[i]) « " ";
    }
    std::cout « std::endl « "q:" « std::endl;
    for (int i = 0; i < 3; i++){
        std::cout « q(CHECK_POINTS[i], a) « " ";
    }
    std::cout « std::endl;
    for (int i = 0; i < 3; i++){
        std::cout « r[i] « std::endl;
    }
}

int main(){
    std::vector<std::vector<double» equations(POWER + 1, std::vector<double>(POWER + 2));
    for (int i = 0; i <= POWER; i++){
        for (int j = 0; j <= POWER; j++){
            for (int k = 0; k < POINTS_AMOUNT; k++){
                equations[i][j] += WEIGHT * pow((BEGIN + STEP * k), i + j);
            }
        }
        for (int k = 0; k < POINTS_AMOUNT; k++){
            equations[i][POWER + 1] += WEIGHT * f(BEGIN + STEP * k) * pow(BEGIN + STEP * k, i);
        }
    }
    std::vector<double> a = gaus(equations, POWER + 1);
    std::vector<double> r = approximation(a);
    output(a, r);
    return 0;
}
```

Результаты

Элемент наилучшего приближения:

$$\phi(x) = 1.23199 + 1.28038x + 0.179989x^2 + 0.673151x^3 - 0.133061x^4 + 0.0488768x^5$$

Таблица узлов:

1.0	3.28132
1.1	3.63806
1.2	4.03654
1.3	4.48101
1.4	4.97613
1.5	5.52695
1.6	6.13907
1.7	6.81863
1.8	7.57238
1.9	8.40778
2.0	9.33298

x	1.0(3)	1.5(3)	1.9(6)
$f(x)$	3.39584	5.72389	9.01406
$\phi(x)$	3.39585	5.72389	9.01406
$R_{true}(x)$	$4.48641e^{-6}$	$3.47124e^{-6}$	$4.94157e^{-6}$

Вывод

Полученные показатели погрешности соответствуют степени построенного элемента наилучшего приближения ($n = 5$). Однако система $g_i(x) = x^i$ не ортогональна при больших значениях n система плохо обусловлена. Можно обойти эту трудность, строя и используя многочлены, ортогональные с заданным весом на заданной системе точчек. В нашем же случае (при $n = 5$) обусловленность задачи удовлетворительна.