Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент:	М Р.Жалялетдинов
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-303Б-21
Дата:	
Оценка:	
Подпись:	

Москва, 2024

3.1

1 Постановка задачи

Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках $X_i, i = 0, ..3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

Вариант: 8

$y = \arcsin(x)$, а) $X_i = -0.4, -0.1, 0.2, 0.5$; б) $X_i = -0.4, 0, 0.2, 0.5$;

Рис. 1: Условие

2 Результаты работы

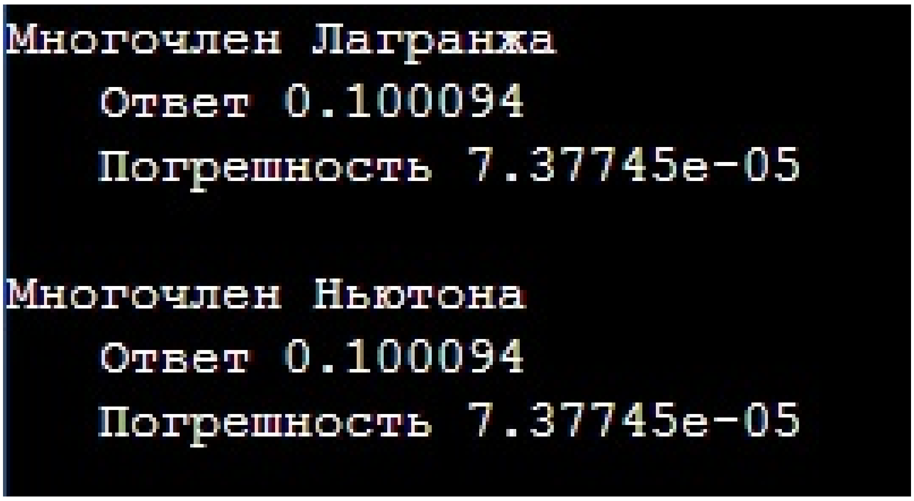

Рис. 2: Вывод программы в консоли

3 Исходный код

```
#include <bits/stdc++.h>
using namespace std;

double first_method(double x, vector<double>& x_v, int n) {
    vector<double> k_vect(x_v.size());

    for (int i = 0; i < n; i++)
        k_vect[i] = asin(x_v[i]);

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; ++j)
            if (i != j)
                k_vect[i] /= x_v[i] - x_v[j];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; ++j)
            if (i != j)
                k_vect[i] *= x - x_v[j];

    double answ = 0;
    for(int i=0; i<n; i++)
        answ += k_vect[i];
    return answ;
}

double second_method(double x, const vector<double>& x_v, int n) {
    vector<double> k_vect(x_v.size());

    for (int i = 0; i < n; i++)
        k_vect[i] = asin(x_v[i]);

    for (int i = 1; i < n; i++)
        for (int j = n - 1; j > i-1; --j)
            k_vect[j] = (k_vect[j] - k_vect[j - 1]) / (x_v[j] - x_v[j - i]);

    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; ++j) {
            k_vect[i] *= x - x_v[j];
        }
    }

    double answ = 0;
    for(int i=0; i<n; i++)
        answ += k_vect[i];
    return answ;
}

```

```
int main() {
    vector<double> x_vect = {-0.4, 0.0, 0.2, 0.5};
    double t = 0.1;

    cout << "\n \n" << "   " << first_method(t, x_vect, 4) << endl << "   " << abs(
        first_method(t, x_vect, 4) - asin(t)) << endl;
    cout << "\n \n" << "   " << second_method(t, x_vect, 4) << endl << "   " << abs(
        second_method(t, x_vect, 4) - asin(t)) << endl;

    return 0;
}
```

3.2

4 Постановка задачи

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

Вариант: 8

$$X_i = 0.8$$

$X^* =$ 0.1

i	0	1	2	3	4
x_i	-0.4	-0.1	0.2	0.5	0.8
f_i	-0.41152	-0.10017	0.20136	0.52360	0.92730

Рис. 3: Условие

5 Результаты работы

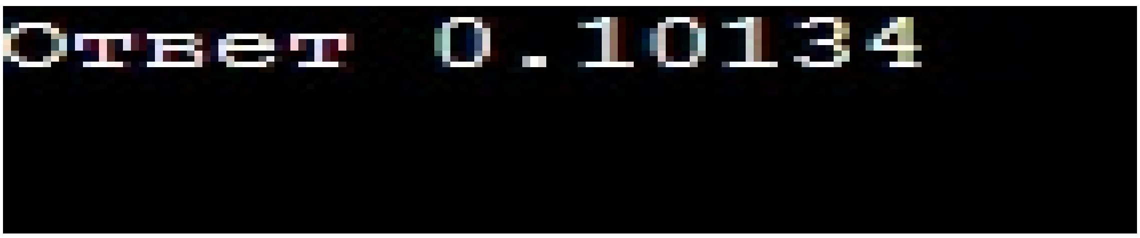

Рис. 4: Вывод программы в консоли

6 Исходный код

```
#include <bits/stdc++.h>

using namespace std;

vector<vector<double>> get_res(vector<vector<double>>& coefficients, vector<vector<
    double>>& results) {
    double a, b, c, d;
    a = 0;
    b = coefficients[0][0];
    c = coefficients[0][1];
    d = results[0][0];
    vector<double> P(coefficients[0].size(), 0), Q(coefficients[0].size(), 0);

    P[0] = -c/b;
    Q[0] = d/b;
    for (int i=1; i < coefficients.size() - 1; i++){
        a = coefficients[i][i-1];
        b = coefficients[i][i];
        c = coefficients[i][i+1];
        d = results[i][0];

        P[i] = -c/(b + a*P[i-1]);
        Q[i] = (d - a*Q[i-1])/(b + a*P[i-1]);
    }

    a = coefficients[coefficients.size()-1][coefficients[0].size()-2];
    b = coefficients[coefficients.size()-1][coefficients[0].size()-1];
    c = 0;
    d = results[results.size()-1][0];

    Q[Q.size()-1] = (d - a * Q[Q.size()-2]) / (b + a * P[P.size()-2]);

    vector<vector<double>> decision(results.size());
    for(int i=0; i<decision.size(); i++)
        decision[i].push_back(0);

    decision[decision.size()-1][0] = Q[Q.size()-1];
    for (int i = decision.size()-2; i > -1; i--)
        decision[i][0] = P[i]*decision[i+1][0] + Q[i];

    return decision;
}

int main() {
    vector<double> x = {-0.4, -0.1, 0.2, 0.5, 0.8}, y = {-0.41152, -0.10017, 0.20136,
        0.52360, 0.92730}, h = {0.0};
    double t = 0.1;
```

```

    for (int i = 0; i < 4; ++i)
        h.push_back(x[i + 1] - x[i]);

    vector<vector<double>> X_content = {{2 * (h[1] + h[2]), h[2], 0}}, Y_content = {};

    for (int i=3; i<4; i++)
        X_content.push_back({h[i - 1], 2 * (h[i - 1] + h[i]), h[i]});

    for (int i=0; i<4-1; i++)
        Y_content.push_back({3 * ((y[i + 2] - y[i + 1]) / h[i + 2] - (y[i + 1] - y[i])
            / h[i + 1])});

    X_content.push_back({0, h[3], 2 * (h[3] + h[4])});

    vector<vector<double>> X(X_content), Y(Y_content);

    vector<double> k_a(y.begin(), y.end() - 1);
    vector<double> k_c = {0};
    auto result = get_res(X, Y);
    for (auto val : result) {
        k_c.push_back(val[0]);
    }
    vector<double> k_b;
    for (int i = 1; i < 4; ++i) {
        k_b.push_back((y[i] - y[i - 1]) / h[i] - h[i] * (k_c[i] + 2 * k_c[i - 1]) / 3);
    }
    k_b.push_back((y[4] - y[3]) / h[4] - 2 * h[4] * k_c[3] / 3);

    vector<double> k_d;
    for (int i = 0; i < 3; ++i) {
        k_d.push_back((k_c[i + 1] - k_c[i]) / (3 * h[i + 1]));
    }
    k_d.push_back(-k_c[3] / (3 * h[4]));

    bool marker = false;
    for (int i = 0; i < 4; ++i) {
        if (x[i] <= t && t <= x[i + 1]) {
            double res = k_a[i] + k_b[i]*(t-x[i]) + k_c[i]*(t-x[i])*(t-x[i]) + k_d[i]*(
                t-x[i])*(t-x[i])*(t-x[i]);
            cout << " " << res << endl;
            marker = true;
            break;
        }
    }

    return 0;
}
```

3.3

7 Постановка задачи

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

Вариант: 8

i	0	1	2	3	4	5
x_i	-0.7	-0.4	-0.1	0.2	0.5	0.8
y_i	-0.7754	-0.41152	-0.10017	0.20136	0.5236	0.9273

Рис. 5: Условия

8 Результаты работы

```
Многочлен первого порядка 0.00552648 + 1.1067 * x
Погрешность 0.00394166

Многочлен первого порядка -0.0069917 + 1.10189 * x + 0.0481468 * x^2
Погрешность 0.00324066
```

Рис. 6: Вывод программы в консоли

9 Исходный код

```
#include <bits/stdc++.h>

using namespace std;
using matrix = vector<vector<double>>;

pair<matrix, matrix> lu_decomposition(matrix& coefficients, matrix& results) {
    int n1=coefficients.size(), m1=coefficients[0].size(), m2 = results[0].size();
    matrix L(n1), U = coefficients;
    for (int i=0; i<n1; i++)
        for (int j=0; j<m1; j++)
            L[i].push_back(0);

    for (int k=0; k<n1; k++) {
        if (U[k][k] == 0) {
            for (int i=k+1; i<n1; i++) {
                if (U[i][k] != 0) {
                    swap(U[k], U[i]);
                    swap(L[k], L[i]);
                    swap(coefficients[k], coefficients[i]);
                    swap(results[k], results[i]);
                    break;
                }
            }
        }
        L[k][k] = 1;
        for (int i=k+1; i<n1; i++) {
            L[i][k] = U[i][k]/U[k][k];
            if (U[i][k] == 0)
                continue;
            for(int j=k; j<m1; j++)
                U[i][j] -= L[i][k]*U[k][j];

        }
    }

    return make_pair(L, U);
}

matrix calculate_results(matrix& coefficients, matrix& results) {
    auto [L, U] = lu_decomposition(coefficients, results);
    matrix res = results;

    for (int k=0; k<res[0].size(); k++)
        for (int i=0; i<res.size(); i++)
            for (int j=0; j<i; j++)
                res[i][k] -= res[j][k]*L[i][j];
    for (int k=0; k<res[0].size(); k++) {
```

```
        for (int i=coefficients.size()-1; i>-1; i--) {
            for (int j=i+1; j<results.size(); j++) {
                res[i][k] -= res[j][k]*U[i][j];
            }
            res[i][k] /= U[i][i];
        }
    }

    return res;
}

int main() {
    vector<double> x = {-0.7, -0.4, -0.1, 0.2, 0.5, 0.8}, y = {-0.7754, -0.41152, -0.10017, 0.20136, 0.5236, 0.9273};

    double n = x.size();
    double ex = 0, ex2 = 0, ex3 = 0, ex4 = 0, ey = 0, eyx = 0, eyx2 = 0;
    for (int i=0; i<x.size(); i++){
        ex += x[i]; ex2 += pow(x[i], 2); ex3 += pow(x[i], 3); ex4 += pow(x[i], 4); ey += y[i]; eyx += y[i]*x[i]; eyx2 += y[i]*pow(x[i], 2);
    }

    matrix X = {{n, ex},{ex, ex2}};
    matrix Y = {{ey},{eyx}};
    matrix results = calculate_results(X, Y);

    cout << "    " << results[0][0] << " + " << results[1][0] << " * x" << endl;
    double diff_f1 = 0;
    for (int i = 0; i < n; i++)
        diff_f1 += pow(results[0][0] + results[1][0] * x[i] - y[i], 2);
    cout << " " << diff_f1 << endl << endl;

    X = {{n, ex, ex2},{ex, ex2, ex3},{ex2, ex3, ex4}};
    Y = {{ey},{eyx},{eyx2}};
    results = calculate_results(X, Y);
    cout << "    " << results[0][0] << " + " << results[1][0] << " * x + " << results[2][0] << " * x^2" << endl;
    double diff_f2 = 0;
    for (int i = 0; i < n; i++)
        diff_f2 += pow(results[0][0] + results[1][0] * x[i] + results[2][0] * pow(x[i], 2) - y[i], 2);
    cout << " " << diff_f2 << endl;

    return 0;
}
```

3.4

10 Постановка задачи

Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i), i = 0, 1, 2, 3, 4$ в точке $x = X_i$.

Вариант: 8 $X^* = 2.0$

$X^* =$ 1.0

i	0	1	2	3	4
x_i	-1.0	0.0	1.0	2.0	3.0
y_i	-0.7854	0.0	0.78540	1.1071	1.249

Рис. 7: Условия

11 Результаты работы

```
Правосторонняя первая производная 0.7854
Правосторонняя первая производная 0.3217
Вторая �роизводная -0.4637
Вторая производная -0.1798
```

Рис. 8: Вывод программы в консоли

12 Исходный код

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    double t = 1.0;
    vector<double> x = {-1.0, 0.0, 1.0, 2.0, 3.0}, y = {-0.7854, 0.0, 0.78540, 1.1071, 1.249};

    vector<double> first;
    for (int i = 0; i < 4; ++i)
        first.push_back((y[i + 1] - y[i]) / (x[i + 1] - x[i]));

    vector<double> second;
    for (int i = 0; i < 3; ++i) {
        double val = 2 * ((y[i+2]-y[i+1])/(x[i+2]-x[i + 1])-(y[i+1]-y[i])/(x[i+1]-x[i]))/(x[i+2]-x[i]);
        second.push_back(val);
    }

    for (int i = 0; i < 4; ++i) {
        if (x[i] == t) {
            cout << "   " << first[i - 1] << endl;
            cout << "   " << first[i] << endl;
            break;
        }
    }

    for (int i = 0; i < 3; ++i)
        if (x[i] <= t && t <= x[i + 1])
            cout << "  " << second[i] << endl;

    return 0;
}
```

3.5

13 Постановка задачи

Вычислить определенный интеграл $\int\limits_{X_0}^{X_1} ydx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:
Вариант: 8

$$y = \frac{1}{x^2+4}, \qquad X_0 = -2,\ \ X_k = 2,\ \ h_1 = 1.0,\ \ h_2 = 0.5;$$

Рис. 9: Условие

14 Результаты работы

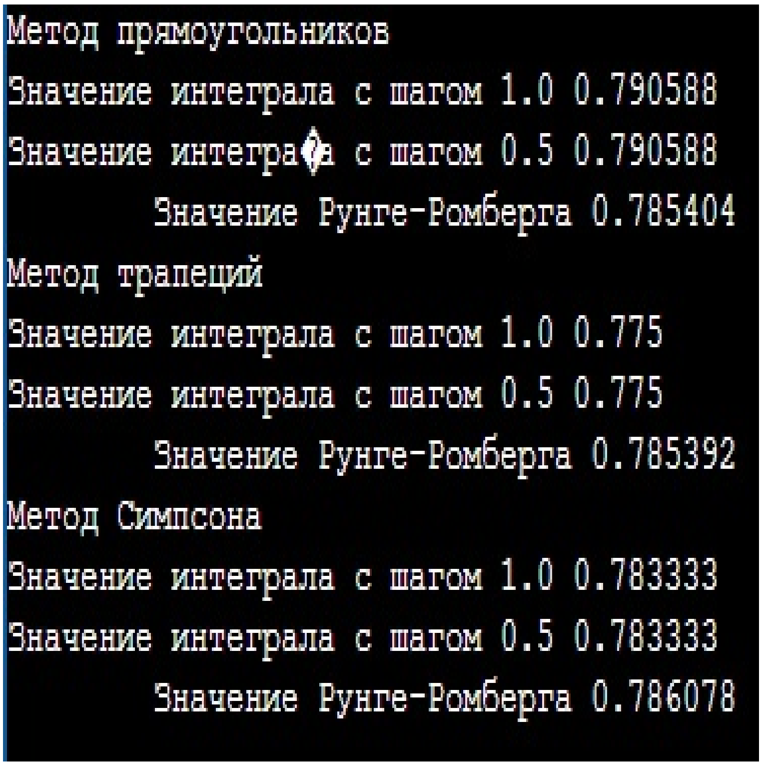

Рис. 10: Вывод программы в консоли

15 Исходный код

```
#include <bits/stdc++.h>
using namespace std;

double rectangular(function<double(double)> f, double x_begin, double x_end, double k)
    {
    double x = x_begin, res = 0;
    while (x < x_end){
        res += f((2*x + k)/2);
        x += k;
    }
    return k*res;
}


double trapeze(function<double(double)> f, double x_begin, double x_end, double k){
    double x = x_begin+k, res = f(x_begin)/2 + f(x_end)/2;
    while (x < x_end){
        res += f(x);
        x += k;
    }
    return k * res;
}


double simpson(function<double(double)> f, double x_begin, double x_end, double k){
    double x = x_begin + k, res = f(x_begin) + f(x_end);
    bool flag = true;
    while (x < x_end){
        if (flag)
            res += f(x) * 4;
        else
            res += f(x) * 2;
        x += k;
        flag = !flag;
    }
    return k * res / 3;
}

double Runge(double v1, double v2, double k1, double k2, double p){
    return v1 + (v1 - v2)/(pow((k2/k1), p) - 1);
}

int main() {
    auto y = [](double x) {
        return 1 / (pow(x, 2) + 4);
    };
    double x0 = -2, xk = 2, val1, val2;
```

```
    cout << " " << endl;
    val1 = rectangular(y, x0, xk, 1.0);
    val2 = rectangular(y, x0, xk, 0.5);
    cout << "    1.0 " << val1 << endl;
    cout << "    0.5 " << val1 << endl;
    cout << "\t - " << Runge(val1, val2, 1.0, 0.5, 2) << endl;

    cout << " " << endl;
    val1 = trapeze(y, x0, xk, 1.0);
    val2 = trapeze(y, x0, xk, 0.5);
    cout << "    1.0 " << val1 << endl;
    cout << "    0.5 " << val1 << endl;
    cout << "\t - " << Runge(val1, val2, 1.0, 0.5, 2) << endl;

    cout << " " << endl;
    val1 = simpson(y, x0, xk, 1.0);
    val2 = simpson(y, x0, xk, 0.5);
    cout << "    1.0 " << val1 << endl;
    cout << "    0.5 " << val1 << endl;
    cout << "\t - " << Runge(val1, val2, 1.0, 0.5, 2) << endl;

    return 0;
}
```