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:	С. П. Сабурова
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: 18

$y = \sqrt{x} + x, a)X_i = 0, 1.7, 3.4, 5.1;)X_i = 0, 1.7, 4.0, 5.1; X^* = 3.0$

2 Результаты работы

```
Lagrange               Result: 4.75178          Loss: 0.0197268
Newton                 Result: 4.75178          Loss: 0.0197268
```

```
 Lagrange               Result: 4.77471         Loss: 0.0426626
 Newton                 Result: 4.77471         Loss: 0.0426626
```

Рис. 1: Вывод программы в консоли

3 Исходный код

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

double newton(double x, const vector<pair<double, double>>& xy) {
    vector<double> coefs(xy.size());
    int n = xy.size();
    for (int i = 0; i < n; ++i)
        coefs[i] = xy[i].second;
    for (int i = 1; i < n; ++i)
        for (int j = n - 1; j > i-1; --j)
            coefs[j] = (coefs[j] - coefs[j - 1]) / (xy[j].first - xy[j - i].first);
    for (int i = 1; i < n; ++i)
        for (int j = 0; j < i; ++j)
            coefs[i] *= x - xy[j].first;
    double res = 0;
    for(auto val: coefs)
        res += val;
    return res;
}

double lagrange(double x, const vector<pair<double, double>>& xy) {
    vector<double> coefs(xy.size());
    int n = xy.size();
    for (int i = 0; i < n; ++i)
        coefs[i] = xy[i].second;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j)
                coefs[i] /= xy[i].first - xy[j].first;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j)
                coefs[i] *= x - xy[j].first;
    double res = 0;
    for(auto val: coefs)
        res += val;
    return res;
}

int main() {
    vector<double> input_x = {0, 1.7, 3.4, 5.1};
    double star_x = 3.0;
    vector<pair<double, double>> xy;
    for (double x : input_x)
        xy.emplace_back(x, sqrt(x) + x);
    cout << "Lagrange" << "\t\tResult: " << lagrange(star_x, xy) << "\t\tLoss: " << abs
        (lagrange(star_x, xy) - sqrt(star_x) - star_x) << endl;
```

```cpp
    cout << "Newton " << "\t\tResult: " << newton(star_x, xy) << "\t\tLoss: " << abs(
        newton(star_x, xy) - sqrt(star_x) - star_x) << 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: 18

$$X_i = 3.0$$

i	0	1	2	3	4
x_i	0.0	1.7	3.4	5.1	6.8
f_i	0.0	3.0038	5.2439	7.3583	9.4077

Рис. 2: Условие

5 Результаты работы

```
Result: 4.75316
```

Рис. 3: Вывод программы в консоли

6 Исходный код

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

vector<vector<double>> multiple_matrix(vector<vector<double>>& matrix1, vector<vector<
    double>>& matrix2) {
    int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
    vector<vector<double>> res(n1);
    for (int i=0; i<n1; i++)
        for (int j=0; j<m2; j++)
            res[i].push_back(0);

    for (int i=0; i<n1; i++) {
        for (int j=0; j<m2; j++) {
            double cntr = 0;
            for (int k=0; k<m1; k++)
               cntr += matrix1[i][k] * matrix2[k][j];
            res[i][j] = cntr;
        }
    }
    return res;
}

vector<vector<double>> tridiagonal_algorithm(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() {
    double star_x = 3.0;
    vector<double> x = {0, 1.7, 3.4, 5.1, 6.8}, y = {0.0, 3.0038, 5.2439, 7.3583,
        9.4077}, h = {0.0};
    for (int i = 0; i < 4; ++i)
        h.push_back(x[i + 1] - x[i]);
    vector<vector<double>> Xdata = {{2 * (h[1] + h[2]), h[2], 0}}, Ydata = {};
    for (int i=3; i<4; i++)
        Xdata.push_back({h[i - 1], 2 * (h[i - 1] + h[i]), h[i]});
    for (int i=0; i<4-1; i++)
        Ydata.push_back({3 * ((y[i + 2] - y[i + 1]) / h[i + 2] - (y[i + 1] - y[i]) / h[
            i + 1])});
    Xdata.push_back({0, h[4 - 1], 2 * (h[4 - 1] + h[4])});
    vector<vector<double>> X(Xdata), Y(Ydata);
    vector<double> a(y.begin(), y.end() - 1), b, c = {0}, d;
    auto result = tridiagonal_algorithm(X, Y);
    for (auto val : result)
        c.push_back(val[0]);
    for (int i = 1; i < 4; ++i)
        b.push_back((y[i] - y[i - 1]) / h[i] - h[i] * (c[i] + 2 * c[i - 1]) / 3);
    b.push_back((y[4] - y[4 - 1]) / h[4] - 2 * h[4] * c[4 - 1] / 3);
    for (int i = 0; i < 4 - 1; ++i)
        d.push_back((c[i + 1] - c[i]) / (3 * h[i + 1]));
    d.push_back(-c[4 - 1] / (3 * h[4]));
    for (int i = 0; i < 4; ++i)
        if (x[i] <= star_x && star_x <= x[i + 1]) {
            double res = a[i] + b[i]*(star_x-x[i]) + c[i]*(star_x-x[i])*(star_x-x[i]) +
                d[i]*(star_x-x[i])*(star_x-x[i])*(star_x-x[i]);
            cout << "Result: " << res << endl;
            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: 18

i	0	1	2	3	4	5
x_i	0.0	1.7	3.4	5.1	6.8	8.5
y_i	0.0	3.0038	5.2439	7.3583	9.4077	11.415

Рис. 4: Условия

8 Результаты работы

```
Coefficents 0.471371 1.31767
Loss = 0.48743

Coefficents 0.129389 1.61941 -0.0354999
Loss = 0.0944722
```

Рис. 5: Вывод программы в консоли

9 Исходный код

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

using namespace std;

vector<vector<double>> multiple_matrix(vector<vector<double>>& matrix1, vector<vector<
    double>>& matrix2) {
    int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
    vector<vector<double>> res(n1);
    for (int i=0; i<n1; i++)
        for (int j=0; j<m2; j++)
            res[i].push_back(0);

    for (int i=0; i<n1; i++) {
        for (int j=0; j<m2; j++) {
            double cntr = 0;
            for (int k=0; k<m1; k++)
               cntr += matrix1[i][k] * matrix2[k][j];
            res[i][j] = cntr;
        }
    }
    return res;
}

pair<vector<vector<double>>, vector<vector<double>>> lu_decomposition(vector<vector<
    double>>& coefficients, vector<vector<double>>& results) {
    int n1=coefficients.size(), m1=coefficients[0].size(), m2 = results[0].size();
    vector<vector<double>> 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);
}

vector<vector<double>> calculate_decisions(vector<vector<double>>& coefficients, vector<vector<double>>& results) {
    auto [L, U] = lu_decomposition(coefficients, results);
    vector<vector<double>> 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 = {0.0, 1.7, 3.4, 5.1, 6.8, 8.5}, y = {0.0, 3.0038, 5.2439, 7.3583, 9.4077, 11.415};
    double element_one = 0, element_two = 0, element_three = 0, element_four = 0, element_five = 0, element_six = 0, element_seven = 0;
    for (int i=0; i<6; i++){
        element_one += x[i];
        element_two += x[i]*x[i];
        element_three += x[i]*x[i]*x[i];
        element_four += x[i]*x[i]*x[i]*x[i];
        element_five += y[i];
        element_six += y[i]*x[i];
        element_seven += y[i]*x[i]*x[i];
    }

    vector<vector<double>> X = {
        {6.0, element_one},
        {element_one, element_two}
```

```
    };
    vector<vector<double>> Y = {
        {element_five},
        {element_six}
    };
    vector<vector<double>> coeffs = calculate_decisions(X, Y);
    cout << "Coefficents " << coeffs[0][0] << " " << coeffs[1][0] << endl;
    double loss = 0;
    for (int i = 0; i < 6; i++)
        loss += pow(coeffs[0][0] + coeffs[1][0] * x[i] - y[i], 2);
    cout << "Loss = " << loss << endl << endl;

    X = {
        {6, element_one, element_two},
        {element_one, element_two, element_three},
        {element_two, element_three, element_four}
    };
    Y = {
        {element_five},
        {element_six},
        {element_seven}
    };
    coeffs = calculate_decisions(X, Y);
    cout << "Coefficents " << coeffs[0][0] << " " << coeffs[1][0] << " " << coeffs[2][0] << endl;
    loss = 0;
    for (int i = 0; i < 6; i++)
        loss += pow(coeffs[0][0] + coeffs[1][0] * x[i] + coeffs[2][0] * pow(x[i], 2) - y[i], 2);
    cout << "Loss = " << loss << 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: 18

$X^* = 0.2$

i	0	1	2	3	4
x_i	-0.2	0.0	0.2	0.4	0.6
y_i	1.5722	1.5708	1.5694	1.5593	1.5273

Рис. 6: Условия

11 Результаты работы

```
Left derivative = -0.007        Right derivative = -0.0505        Second derivative = -0.2175
```

Рис. 7: Вывод программы в консоли

12 Исходный код

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

int main() {
    vector<double> x = {-0.2, 0.0, 0.2, 0.4, 0.6}, y = {1.5722, 1.5708, 1.5694, 1.5593, 1.5273}, dy, ddy;
    double star_x = 0.2;
    for (int i = 0; i < 5; i++)
        dy.push_back((y[i + 1] - y[i]) / (x[i + 1] - x[i]));
    for (int i = 0; i < 4; i++)
        ddy.push_back(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]));
    for (int i = 0; i < 5; i++)
        if (x[i] == star_x) {
            cout << "Left derivative = " << dy[i - 1] << "\tRight derivative = " << dy[i];
            break;
        } else if (x[i] < star_x && star_x < x[i + 1])
            cout << "First derivative = " << dy[i];
    for (int i = 0; i < 4; i++)
        if (x[i] <= star_x && star_x <= x[i + 1]) {
            cout << "\tSecond derivative = " << ddy[i] << endl;
            break;
        }
    return 0;
}
```

3.5

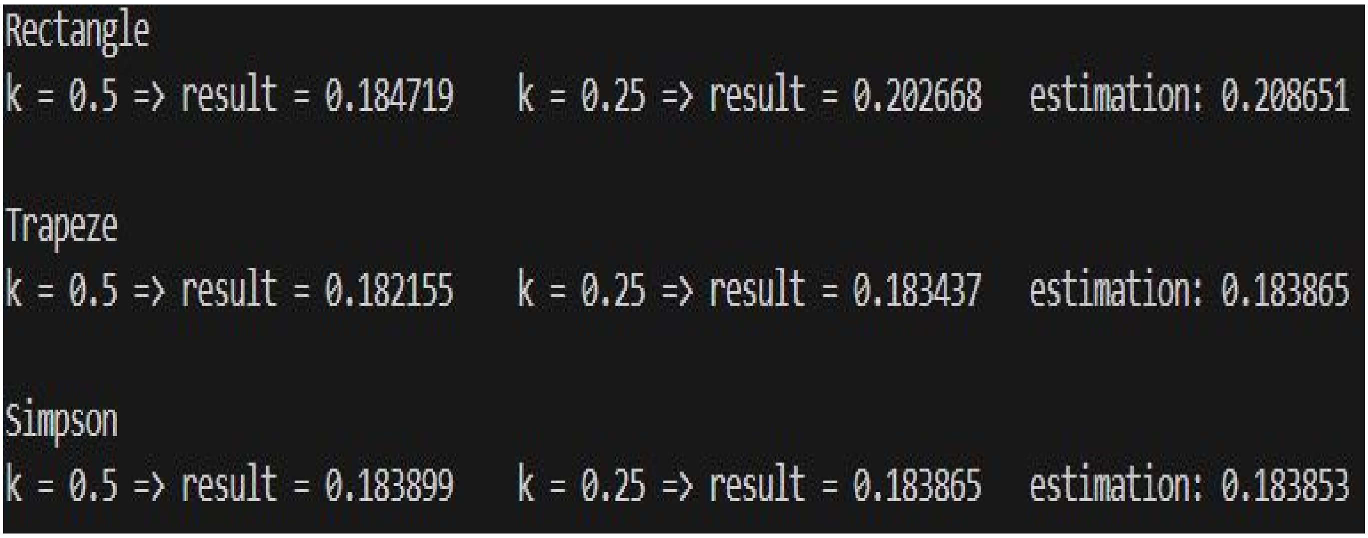
13 Постановка задачи

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: 18

$y = \frac{x}{16-x^4}$ $X_0 = -1, X_k = 1, h_1 = 0.5, h_2 = 0.25$

14 Результаты работы



```
Rectangle
k = 0.5 => result = 0.184719    k = 0.25 => result = 0.202668   estimation: 0.208651

Trapeze
k = 0.5 => result = 0.182155    k = 0.25 => result = 0.183437   estimation: 0.183865

Simpson
k = 0.5 => result = 0.183899    k = 0.25 => result = 0.183865   estimation: 0.183853
```

Рис. 8: Вывод программы в консоли

15 Исходный код

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

double y(double x) {
    return x / (x*x + 9);
}

int main() {
    double x_0 = 0, x_k = 2, h1 = 0.5, h2 = 0.25;
    double F1, F2;
    cout << "Rectangle" << endl;
    double x = x_0, res = 0;
    while (x < x_k){
        res += y((2*x + h1)/2);
        x += h1;
    }
    F1 = h1*res;
    x = x_0, res = 0;
    while (x < x_k){
        res += y((2*x + h1)/2);
        x += h2;
    }
    F2 = h2*res;
    cout << "k = 0.5 => result = " << F1 << "\tk = 0.25 => result = " << F2 << "\testimation: " << F1 + (F1 - F2)/(pow((h2/h1), 2) - 1) << endl << endl;
    cout << "Trapeze" << endl;
    x = x_0+h1, res = y(x_0)/2 + y(x_k)/2;
    while (x < x_k){
        res += y(x);
        x += h1;
    }
    F1 = h1 * res;
    x = x_0+h2, res = y(x_0)/2 + y(x_k)/2;
    while (x < x_k){
        res += y(x);
        x += h2;
    }
    F2 = h2 * res;
    cout << "k = 0.5 => result = " << F1 << "\tk = 0.25 => result = " << F2 << "\testimation: " << F1 + (F1 - F2)/(pow((h2/h1), 2) - 1) << endl << endl;
    cout << "Simpson" << endl;
    x = x_0 + h1, res = y(x_0) + y(x_k);
    bool flag = true;
    while (x < x_k){
        res += y(x) * ((flag) ? 4 : 2);
        x += h1;
        flag = !flag;
```

```
    }
    F1 = h1 * res / 3;
    x = x_0 + h2, res = y(x_0) + y(x_k);
    flag = true;
    while (x < x_k){
        res += y(x) * ((flag) ? 4 : 2);
        x += h2;
        flag = !flag;
    }
    F2 = h2 * res / 3;
    cout << "k = 0.5 => result = " << F1 << "\tk = 0.25 => result = " << F2 << "\
        testimation: " << F1 + (F1 - F2)/(pow((h2/h1), 2) - 1) << endl << endl;
    return 0;
}
```