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:	Я. А. Хайруллина
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: 26

$y = 1/x^2 + x^2, a)X_i = 0.1, 0.5, 0.9, 1.3;)X_i = 0.1, 0.5, 1.1, 1.3; X^* = 0.8$

2 Результаты работы

```
Lagrange method
for x1
L(x)
[100.01 4.25    2.04457 2.28172 ]
y(x)
[100.01 4.25    2.04457 2.28172 ]
delta(x)
[0      0       0       4.44089e-16     ]


for x2
L(x)
[100.01 4.25    2.04457 2.28172 ]
y(x)
[100.01 4.25    2.03645 2.28172 ]
delta(x)
[1.42109e-14    8.88178e-16     0       0       ]
```

```
Newton method
for x1
P(x)
[100.01 4.25    2.04457 2.28172 ]
y(x)
[100.01 4.25    2.04457 2.28172 ]
delta(x)
[0      0       3.9968e-15      1.95399e-14     ]


for x2
P(x)
[100.01 4.25    2.03645 2.28172 ]
y(x)
[100.01 4.25    2.03645 2.28172 ]
delta(x)
[0      0       4.70735e-14     5.15143e-14     ]
```

Рис. 1: Вывод программы в консоли

3 Исходный код

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

using namespace std;

double f(double x1, double x2) {
    return ((pow(1 / x1, 2) + pow(x1, 2)) - (pow(1 / x2, 2) + pow(x2, 2))) / (x1 - x2);
}

double f1(double x1, double x2, double x3) {
    return (f(x1, x2) - f(x2, x3)) / (x1 - x3);
}

double f2(double x1, double x2, double x3, double x4) {
    return (f1(x1, x2, x3) - f1(x2, x3, x4)) / (x1 - x4);
}

vector <double> omega_val(vector <double> x) {
    vector <double> omega(4, 0);
    omega[0] = (x[0] - x[1]) * (x[0] - x[2]) * (x[0] - x[3]);
    omega[1] = (x[1] - x[0]) * (x[1] - x[2]) * (x[1] - x[3]);
    omega[2] = (x[2] - x[0]) * (x[2] - x[1]) * (x[2] - x[3]);
    omega[3] = (x[3] - x[0]) * (x[3] - x[1]) * (x[3] - x[2]);
    return omega;
}

void Lagrange(vector <double> x, double X, vector <double>& L, vector <double>& y, vector <double>& delta) {
    vector <vector <double>> ans(5, vector<double>(4, 0));
    vector <double> omega = omega_val(x);
    for (int i = 0; i < x.size(); i++) {
        ans[0][i] = x[i];
        ans[1][i] = pow(1 / x[i], 2) + pow(x[i], 2);
        ans[2][i] = omega[i];
        ans[3][i] = (pow(1 / x[i], 2) + pow(x[i], 2)) / omega[i];
        ans[4][i] = X - x[i];
    }

    for (int i = 0; i < x.size(); i++) {
        L[i] = (ans[3][0] * (x[i] - ans[0][1]) * (x[i] - ans[0][2]) * (x[i] - ans[0][3]) \
            + ans[3][1] * (x[i] - ans[0][0]) * (x[i] - ans[0][2]) * (x[i] - ans[0][3]) \
            + ans[3][2] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][3]) \
```

```
            + ans[3][3] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][2]));
    }

    for (int i = 0; i < x.size(); i++) {
        y[i] = pow(1 / x[i], 2) + pow(x[i], 2);
    }

    for (int i = 0; i < x.size(); i++) {
        delta[i] = fabs(y[i] - L[i]);
    }
}


void Newton(vector <double> x, double X, vector <double>& P, vector <double>& y, vector <double>& delta) {
    vector <vector <double>> ans(5, vector<double>(4, 0));
    for (int i = 0; i < x.size(); i++) {
        ans[0][i] = x[i];
        ans[1][i] = pow(1 / x[i], 2) + pow(x[i], 2);
        if (i < 3) {
            ans[2][i] = f(x[i], x[i + 1]);
        }
        if (i < 2) {
            ans[3][i] = f1(x[i], x[i + 1], x[i + 2]);
        }
    }
    ans[4][0] = f2(x[0], x[1], x[2], x[3]);


    for (int i = 0; i < x.size(); i++) {
        P[i] = (ans[1][0] + ans[2][0] * (x[i] - ans[0][0]) + ans[3][0] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) \
            + ans[4][0] * (x[i] - ans[0][0]) * (x[i] - ans[0][1]) * (x[i] - ans[0][2]));
    }

    for (int i = 0; i < x.size(); i++) {
        y[i] = pow(1 / x[i], 2) + pow(x[i], 2);
    }

    for (int i = 0; i < x.size(); i++) {
        delta[i] = fabs(y[i] - P[i]);
    }
}

int main() {
    int n = 4;
    vector <double> x1 = { 0.1, 0.5, 0.9, 1.3 };
```

```
    vector <double> x2 = { 0.1, 0.5, 1.1, 1.3 };
    double root = 0.8;
    vector <double> L(n, 0), y(n, 0), delta(n, 0), P(n, 0);
    std::cout << "Lagrange method" << std::endl;
    Lagrange(x1, root, L, y, delta);
    std::cout << "for x1" << std::endl << "L(x)\n" << "[";
    for (int i = 0; i < L.size(); i++) std::cout << L[i] << "\t";
    std::cout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
    std::cout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
    std::cout << "]\n";
    Lagrange(x2, root, P, y, delta);
    std::cout << "\n\nfor x2" << std::endl << "L(x)\n" << "[";
    for (int i = 0; i < L.size(); i++) std::cout << L[i] << "\t";
    std::cout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
    std::cout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
    std::cout << "]\n";
    std::cout << "\n\nNewton method" << std::endl;
    Newton(x1, root, P, y, delta);
    std::cout << "for x1" << std::endl << "P(x)\n" << "[";
    for (int i = 0; i < P.size(); i++) std::cout << P[i] << "\t";
    std::cout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
    std::cout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
    std::cout << "]\n";
    Newton(x2, root, P, y, delta);
    std::cout << "\n\nfor x2" << std::endl << "P(x)\n" << "[";
    for (int i = 0; i < P.size(); i++) std::cout << P[i] << "\t";
    std::cout << "]\n" << "y(x)\n" << "[";
    for (int i = 0; i < y.size(); i++) std::cout << y[i] << "\t";
    std::cout << "]\n" << "delta(x)\n" << "[";
    for (int i = 0; i < delta.size(); i++) std::cout << delta[i] << "\t";
    std::cout << "]\n";
}
```

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: 26

$$X_i = 0.8$$

i	0	1	2	3	4
x_i	0.1	0.5	0.9	1.3	1.7
f_i	100.01	4.2500	2.0446	2.2817	3.2360

Рис. 2: Условие

5 Результаты работы

```
Function value in X with spline: 27.4016
```

Рис. 3: Вывод программы в консоли

6 Исходный код

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

using namespace std;

vector<double> solution(vector<vector<double>>& A, vector<double>& b) {
    int n = A.size();

    for (int i = 0; i < n; i++) {
        int max_row = i;
        for (int k = i + 1; k < n; k++) {
            if (abs(A[k][i]) > abs(A[max_row][i])) {
                max_row = k;
            }
        }
        swap(A[i], A[max_row]);
        swap(b[i], b[max_row]);
        for (int k = i + 1; k < n; k++) {
            double factor = A[k][i] / A[i][i];
            for (int j = i; j < n; j++) {
                A[k][j] -= factor * A[i][j];
            }
            b[k] -= factor * b[i];
        }
    }

    vector<double> x(n);
    for (int i = n - 1; i >= 0; i--) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
        x[i] /= A[i][i];
    }

    return x;
}


double Spline(vector<double>& x, vector<double>& y, double X) {
    vector<double> A(3, 0);
    vector<vector<double>> B(3, vector<double>(3, 0));
    vector<vector<double>> ans(4, vector<double>(4, 0));
    vector<double> roots(3, 0);
    double f = 0;
```

```

    for (int i = 2; i < 5; i++) {
        A[i - 2] = (3 * ((y[i] - y[i - 1]) / (x[i] - x[i - 1]) +
            (y[i - 1] - y[i - 2]) / (x[i - 1] - x[i - 2])));
        for (int j = 0; j < B.size(); j++) {
            for (int k = 0; k < B[0].size(); k++) {
                if (j == k) {
                    B[j][k] = 2 * ((x[i - 1] - x[i - 2]) + (x[i] - x[i - 1]));
                }
                else if (j < k && (k != B.size() - 1 || j != 0)) {
                    B[j][k] = (x[i] - x[i - 1]);
                }
                else if (j > k && (j != B.size() - 1 || k != 0)) {
                    B[j][k] = (x[i - 1] - x[i - 2]);
                }
            }
        }
    }

    roots = solution(B, A);
    roots.insert(roots.begin(), 0);

    for (int i = 0; i < ans.size(); i++) {
        for (int j = 0; j < ans[0].size(); j++) {
            if (i != ans.size() - 1) {
                if (j == 0) {
                    ans[i][j] = y[i];
                }
                if (j == 1) {
                    ans[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (
                        (x[i + 1] - x[i]) * roots[i + 1] + 2 * roots[i]) / 3;
                }
                if (j == 2) {
                    if (i == 0) {
                        ans[i][j] = 0;
                    }
                    else {
                        ans[i][j] = roots[i];
                    }
                }
                if (j == 3) {
                    ans[i][j] = (roots[i + 1] - roots[i]) / (3 * (x[i + 1] - x[i]));
                }
            }
            else {
                if (j == 0) {
                    ans[i][j] = y[i];
                }
                if (j == 1) {
```

```
                    ans[i][j] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (2 * (x[i + 1] - x[i]) * roots[i]) / 3;
                }
                if (j == 2) {
                    ans[i][j] = roots[i];
                }
                if (j == 3) {
                    ans[i][j] = -roots[i] / (3 * (x[i + 1] - x[i]));
                }
            }
        }
    }

    f = ans[1][0] + ans[1][1] * (X - x[1]) + ans[1][2] * pow((X - x[2]), 2) + ans[1][3] * pow((X - x[3]), 3);

    return f;
}


int main() {
    vector<double> x = { 0.1, 0.5, 0.9, 1.3, 1.7 };
    vector<double> y = { 100.01, 4.25, 2.0446, 2.2817, 3.236 };
    double X = 0.8;
    std::cout << "Function value in X with spline: " << Spline(x, y, X) << std::endl;

    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: 26

i	0	1	2	3	4	5
x_i	0.1	0.5	0.9	1.3	1.7	2.1
y_i	100.01	4.250	2.0446	2.2817	3.236	4.6368

Рис. 4: Условия

8 Результаты работы

```
Least squares method, 1st degree
Coefficients: [ 57.0983  -34.2622 ]
Sum of the error square: 4514.05
2st degree
Coefficients: [ 98.4497  -156.648  55.6298 ]
Sum of the error square: 1556.37
```

Рис. 5: Вывод программы в консоли

9 Исходный код

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

std::vector<double> x = { 0.1, 0.5, 0.9, 1.3, 1.7, 2.1 };
std::vector<double> y = { 100.01, 4.25, 2.0446, 2.2817, 3.236, 4.6368 };

double pol(std::vector<double> a, double x) {
    double sum = 0;
    for (int i = 0; i < a.size(); ++i) {
        sum += pow(x, i) * a[i];
    }
    return sum;
}

double error(std::vector<double> a, std::vector<double> x, std::vector<double> y) {
    double sum = 0;
    for (int i = 0; i < x.size(); ++i) {
        sum += pow(pol(a, x[i]) - y[i], 2);
    }
    return sum;
}



std::vector<std::vector<std::vector<double>>> getLU(std::vector<std::vector<double>> A
    ) {
    std::vector<std::vector<double>> U = A;

    std::vector<std::vector<double>> L(U.size(), std::vector<double>(U.size()));
    for (auto& l : L) std::fill(l.begin(), l.end(), 0.0);

    for (int k = 0; k < U.size(); ++k) {
        for (int i = k; i < U.size(); ++i) {
            L[i][k] = U[i][k] / U[k][k];
        }

        for (int i = k + 1; i < U.size(); ++i) {
            for (int j = k; j < U.size(); ++j) {
                U[i][j] = U[i][j] - L[i][k] * U[k][j];
            }
        }
    }

    return { L, U };
}

```

```
std::vector<double> solution(std::vector<std::vector<double>> A, std::vector<double> b
    ) {
    int n = A.size();
    std::vector<std::vector<std::vector<double>>> LU = getLU(A);
    std::vector<std::vector<double>> L = LU[0];
    std::vector<std::vector<double>> U = LU[1];

    std::vector<double> z(n);
    z[0] = b[0];
    for (int i = 1; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < i; ++j) {
            sum += L[i][j] * z[j];
        }
        z[i] = b[i] - sum;
    }

    std::vector<double> x(n);
    x[n - 1] = z[n - 1] / U[n - 1][n - 1];
    for (int i = n - 2; i >= 0; --i) {
        double sum = 0;
        for (int j = i + 1; j < n; ++j) {
            sum += U[i][j] * x[j];
        }

        x[i] = (z[i] - sum) / U[i][i];
    }
    return x;
}

std::vector<double> MNK(std::vector<double> x, std::vector<double> y, int m) {
    std::vector<std::vector<double>> A(m + 1, std::vector<double>(m + 1));
    std::vector<double> b(m + 1);

    for (int k = 0; k < m + 1; ++k) {
        for (int j = 0; j < m + 1; ++j) {
            double sum = 0;
            for (int i = 0; i < x.size(); ++i) {
                sum += pow(x[i], k + j);
            }
            A[k][j] = sum;
        }

        double sum = 0;
        for (int i = 0; i < x.size(); ++i) {
            sum += y[i] * pow(x[i], k);
        }
        b[k] = sum;
    }
```

```cpp

    std::vector<double> a = solution(A, b);
    return a;
}



int main() {
    std::cout << "Least squares method, 1st degree\n";
    std::vector<double> a = MNK(x, y, 1);
    std::cout << "Coefficients: [";
    for (int i = 0; i < a.size(); ++i) std::cout << " " << a[i] << " ";
    std::cout << "]\n";
    std::cout << "Sum of the error square: " << error(a, x, y) << "\n";

    std::cout << "2st degree\n";
    a = MNK(x, y, 2);
    std::cout << "Coefficients: [";
    for (int i = 0; i < a.size(); ++i) std::cout << " " << a[i] << " ";
    std::cout << "]\n";
    std::cout << "Sum of the error square: " << error(a, x, y) << "\n";

    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: 26 $X^* = 2.0$

i	0	1	2	3	4
x_i	0.0	1.0	2.0	3.0	4.0
y_i	0.0	0.86603	1.0	0.0	-2.0

Рис. 6: Условия

11 Результаты работы

```
First derivative in X: -0.23206
Second derivative in X: -0.73206
```

Рис. 7: Вывод программы в консоли

12 Исходный код

```cpp
#include <iostream>
#include <vector>

double derivative(std::vector<double> x, std::vector<double> y, double X) {
    int n = 0;
    double h = x[1] - x[0];
    for (int i = 0; i < x.size() - 1; ++i)
        if (X > x[i] && X < x[i + 1]) {
            if (i == 0) n = i;
            else if (i + 2 > x.size() - 1)
                n = i - (i + 2 - (x.size() - 1));
            else n = i - 1;
        }
    double a1 = (y[n + 1] - y[n]) / h;
    double a2 = ((y[n + 2] - 2 * y[n + 1] + y[n]) / (2 * h * h)) * (2 * X - x[n] - x[n + 1]);
    return a1 + a2;
}

double derivative2(std::vector<double> x, std::vector<double> y, double X) {
    int n = 0;
    double h = x[1] - x[0];
    for (int i = 0; i < x.size() - 1; ++i)
        if (X > x[i] && X < x[i + 1]) {
            if (i == 0) n = i;
            else if (i + 2 > x.size() - 1)
                n = i - (i + 2 - (x.size() - 1));
            else n = i - 1;
        }

    return (y[n + 2] - 2 * y[n + 1] + y[n]) / (h * h);
}

int main() {
    std::vector<double> x = { 0.0, 1.0, 2.0, 3.0, 4.0 };
    std::vector<double> y = { 0.0, 0.86603, 1.0, 0.0, -2.0 };
    double X = 2.0;
    std::cout << "First derivative in X: " << derivative(x, y, X) << "\n";
    std::cout << "Second derivative in X: " << derivative2(x, y, X) << "\n";

    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: 26

$y = x^2\sqrt{36 - x^2}$ $X_0 = 1, X_k = 5, h_1 = 1.0, h_2 = 0.5$

14 Результаты работы

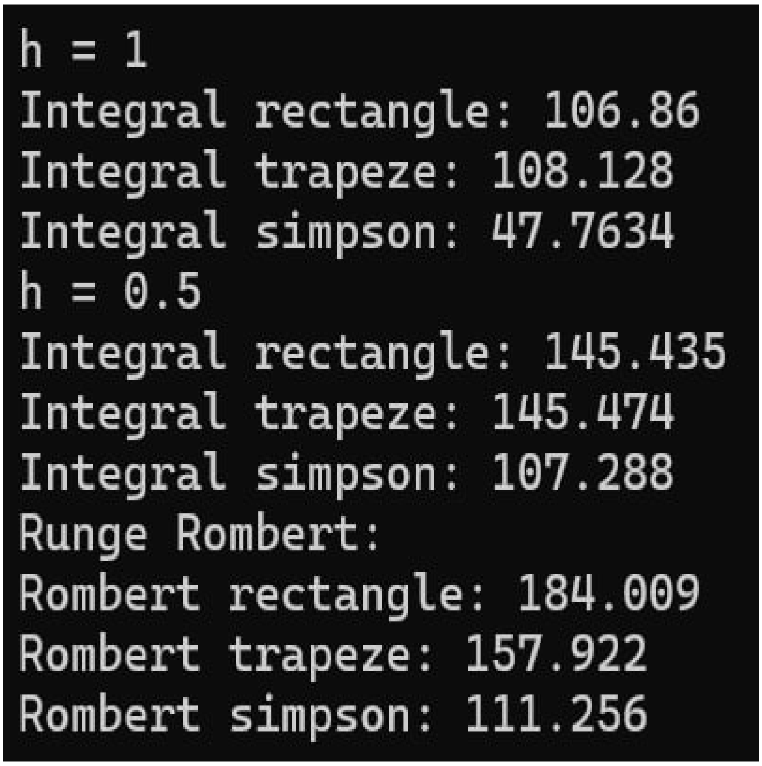

Рис. 8: Вывод программы в консоли

15 Исходный код

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

double f(double x) {
    return pow(x, 2) * pow(36 - pow(x, 2), 0.5);
}

double rectangle_method(double a, double b, double h) {
    double res = 0;
    while (a + h < b) {
        res += f(a + h / 2);
        a += h;
    }
    return res * h;
}

double trapez_method(double a, double b, double h) {
    double res = 0;
    while (a + h < b) {
        res += (f(a + h) + f(a));
        a += h;
    }
    return res * h * 0.5;
}

double simpson_method(double a, double b, double h) {
    double res = 0;
    a += h;
    while (a + h < b) {
        res += f(a - h) + 4 * f(a - h / 2) + f(a);
        a += h;
    }
    return res * h / 6;
}

double rungeRombert(double h1, double h2, double i1, double i2, double p) {
    return i1 + (i1 - i2) / (pow((h2 / h1), p) - 1);
}

int main() {
    double a = 1;
    double b = 5;
    double h1 = 1.0;
    double h2 = 0.5;
    std::cout << "h = " << h1 << "\n";
    std::cout << "Integral rectangle: " << rectangle_method(a, b, h1) << "\n";
    std::cout << "Integral trapeze: " << trapez_method(a, b, h1) << "\n";
```

```
    std::cout << "Integral simpson: " << simpson_method(a, b, h1) << "\n";
    std::cout << "h = " << h2 << "\n";
    std::cout << "Integral rectangle: " << rectangle_method(a, b, h2) << "\n";
    std::cout << "Integral trapeze: " << trapez_method(a, b, h2) << "\n";
    std::cout << "Integral simpson: " << simpson_method(a, b, h2) << "\n";
    std::cout << "Runge Rombert:\n";
    std::cout << "Rombert rectangle: " << rungeRombert(h1, h2, rectangle_method(a, b, h1), rectangle_method(a, b, h2), 1) << "\n";
    std::cout << "Rombert trapeze: " << rungeRombert(h1, h2, trapez_method(a, b, h1), trapez_method(a, b, h2), 2) << "\n";
    std::cout << "Rombert simpson: " << rungeRombert(h1, h2, simpson_method(a, b, h1), simpson_method(a, b, h2), 4) << "\n";

    return 0;
}
```