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

Москва, 2024

1.1 LU - разложение матриц

1 Постановка задачи

Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

Вариант: 14

$$\begin{cases} -x_1 - 3x_2 - 4x_3 = -3 \\ 3x_1 + 7x_2 - 8x_3 + 3x_4 = 30 \\ x_1 - 6x_2 + 2x_3 + 5x_4 = -90 \\ -8x_1 - 4x_2 - x_3 - x_4 = 12 \end{cases}$$

2 Результаты работы

```
Decisions =
-3
6
-3
-9

det of SLAU matrxi= 2231

Inversed matrix SLAU =
0.12147 -0.0425818 -0.00582698 -0.15688
-0.164052 0.0685791 -0.0327208 0.0421336
-0.157329 -0.0407889 0.0259973 0.0076199
-0.158225 0.107127 0.151502 0.0788884
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
#include iostream
#include vector
#include utility

using namespace std;

using matrix = vectorvectordouble;

matrix multiple_matrix(const matrix& matrix1, const matrix& matrix2) {
    int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
    matrix res(n1, vectordouble(m2, 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;
}

pairmatrix, matrix lu_decomposition(matrix& coefficients, matrix& results) {
    int n1 = coefficients.size(), m1 = coefficients[0].size(), m2 = results[0].size();
    matrix L(n1, vectordouble(n1, 0)), U = coefficients;

    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);
}

double get_determinant(matrix& coefficients, matrix& results) {
    pairmatrix, matrix LU = lu_decomposition(coefficients, results);
    double det = 1;
    matrix U = LU.second;
    for (int i = 0; i coefficients.size(); i++) {
        det = U[i][i];
    }
    return det;
}

matrix calculate_decisions(matrix& coefficients, matrix& results) {
    pairmatrix, matrix LU = lu_decomposition(coefficients, results);
    matrix L = LU.first, U = LU.second;
    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 i = coefficients.size() - 1; i = 0; 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;
}

matrix get_inverse_matrix(matrix& matrix1) {
    matrix E(matrix1.size(), vectordouble(matrix1.size(), 0));
    for (int i = 0; i matrix1.size(); i++) {
        E[i][i] = 1;
    }
    return calculate_decisions(matrix1, E);
}

void print_matrix(const matrix& matrix1) {
    for (const auto& vect matrix1) {
        for (auto x vect) {
```

```
            cout x ;
        }
        cout endl;
    }
}

int main() {
    matrix coefficient_matrix{
            {-1, -3, -4, -0},
            {3, 7, -8, 3},
            {1, -6, 2, 5},
            {-8, -4, -1, -1}
    };

    matrix equation_roots = {
            {-3},
            {30},
            {-90},
            {12}
    };

    pairmatrix, matrix LU = lu_decomposition(coefficient_matrix, equation_roots);
    matrix l = LU.first;
    matrix u = LU.second;

    cout Decisions = endl;
    matrix decisions = calculate_decisions(coefficient_matrix, equation_roots);
    print_matrix(decisions);

    cout endl;
    cout det of SLAU matrxi= get_determinant(coefficient_matrix, equation_roots) endl;

    cout endl Inversed matrix SLAU = endl;
    matrix inversed = get_inverse_matrix(coefficient_matrix);
    print_matrix(inversed);
    return 0;
}
```

1.2 Метод прогонки

4 Постановка задачи

Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

Вариант: 14

$$
\begin{cases}
-x_1 - x_2 = -4 \\
7x_1 - 17x_2 - 8x_3 = 132 \\
-9x_2 + 19x_3 + 8x_4 = -59 \\
7x_3 - 20x_4 + 4x_5 = -193 \\
-4x_4 + 12x_5 = -40
\end{cases}
$$

5 Результаты работы

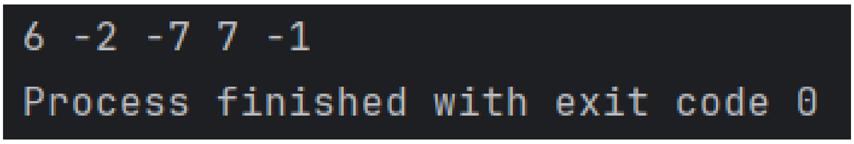

Рис. 2: Вывод программы

6 Исходный код

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

void print_result(std::vector<double> res, int n) {
    for (int i = 0; i < n; i++) {
        std::cout << res[i] << " ";
    }
}

void tridiagonalSolve(const std::vector<std::vector<double>>& A, const std::vector<
    double>& d, std::vector<double>& x) {
    int n = A.size();

    std::vector<double> P(n);
    std::vector<double> Q(n);

    for(int i = 0; i < n; ++i) {
        if(i == 0) {
            P[i] = -A[i][i+1] / A[i][i];
            Q[i] = d[i] / A[i][i];
        } else if(i == n - 1) {
            P[i] = 0;
            Q[i] = (d[i] - A[i][i - 1] * Q[i - 1]) / (A[i][i] + A[i][i - 1] * P[i - 1])
                ;
        } else {
            P[i] = -A[i][i+1] / (A[i][i] + A[i][i - 1] * P[i - 1]);
            Q[i] = (d[i] - A[i][i - 1] * Q[i - 1]) / (A[i][i] + A[i][i - 1] * P[i - 1])
                ;
        }
    }

    for(int i = n - 1; i >= 0; --i) {
        if(i == n - 1) x[i] = Q[i];
        else {
            x[i] = P[i] * x[i + 1] + Q[i];
        }
    }
}

int main() {
    std::vector<std::vector<double>> A = {
            {-1, -1, 0, 0, 0},
            {7, -17, -8, 0, 0},
            {0, -9, 19, 8, 0},
            {0, 0, 7, -20, 4},
            {0, 0, 0, -4, 12}
    };
```

```
    std::vector<double> d = {-4, 132, -59, -193, -40};
    int n = A.size();

    std::vector<double> x(n);
    tridiagonalSolve(A, d, x);

    print_result(x, n);

    return 0;
}
```

1.3 Метод простых итераций. Метод Зейделя

7 Постановка задачи

Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

Вариант: 14

$$\begin{cases} -22x_1 - 2x_2 - 6x_3 + 6x_4 = 96 \\ 3x_1 - 17x_2 - 3x_3 + 7x_4 = -26 \\ 2x_1 + 6x_2 - 17x_3 + 5x_4 = 35 \\ -x_1 - 8x_2 + 8x_3 + 23x_4 = -234 \end{cases}$$

8 Результаты работы

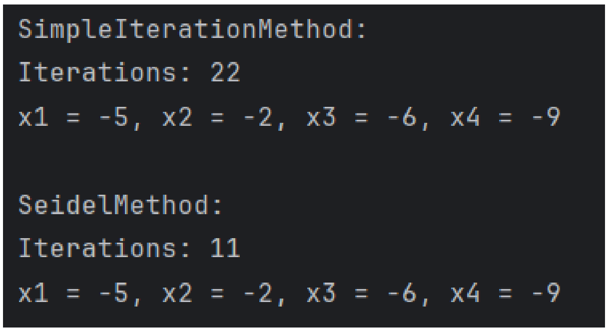

Рис. 3: Вывод программы

9 Исходный код

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

using namespace std;

vector<double> simpleIterationMethod(const vector<vector<double>>& A, const vector<
    double>& b, double epsilon) {
    int n = A.size();
    vector<double> x(n, 0.0);
```

```cpp
    vector<double> x_new(n);

    int iterations = 0;
    double error = epsilon + 1.0;

    while (error > epsilon) {
        for (int i = 0; i < n; ++i) {
            double sum = b[i];
            for (int j = 0; j < n; ++j) {
                if (j != i) {
                    sum -= A[i][j] * x[j];
                }
            }
            x_new[i] = sum / A[i][i];
        }

        error = 0.0;
        for (int i = 0; i < n; ++i) {
            error = max(error, abs(x_new[i] - x[i]));
            x[i] = x_new[i];
        }

        iterations++;
    }

    cout << "Iterations: " << iterations << endl;
    return x;
}

vector<double> gaussSeidelMethod(const vector<vector<double>>& A, const vector<double
    >& b, double epsilon) {
    int n = A.size();
    vector<double> x(n, 0.0);
    vector<double> x_new(n);

    int iterations = 0;
    double error = epsilon + 1.0;

    while (error > epsilon) {
        for (int i = 0; i < n; ++i) {
            double sum1 = 0.0;
            for (int j = 0; j < i; ++j) {
                sum1 += A[i][j] * x_new[j];
            }

            double sum2 = 0.0;
            for (int j = i + 1; j < n; ++j) {
                sum2 += A[i][j] * x[j];
            }
```

```

            x_new[i] = (b[i] - sum1 - sum2) / A[i][i];
        }

        error = 0.0;
        for (int i = 0; i < n; ++i) {
            error = max(error, abs(x_new[i] - x[i]));
            x[i] = x_new[i];
        }

        iterations++;
    }

    cout << "Iterations: " << iterations << endl;
    return x;
}

int main() {
    vector<vector<double>> A = {
            {-22, -2, -6, 6},
            {3, -17, -3, 7},
            {2, 6, -17, 5},
            {-1, -8, 8, 23}
    };

    vector<double> b = {96, -26, 35, -234};
    double epsilon = 1e-6;

    cout << "SimpleIterationMethod:" << endl;
    vector<double> solution_simple_iteration = simpleIterationMethod(A, b, epsilon);
    cout << "x1 = " << solution_simple_iteration[0] << ", x2 = " << solution_simple_iteration[1] << ", x3 = " << solution_simple_iteration[2] << ", x4 = " << solution_simple_iteration[3] << endl;

    cout << endl;

    cout << "SeidelMethod:" << endl;
    vector<double> solution_gauss_seidel = gaussSeidelMethod(A, b, epsilon);
    cout << "x1 = " << solution_gauss_seidel[0] << ", x2 = " << solution_gauss_seidel[1] << ", x3 = " << solution_gauss_seidel[2] << ", x4 = " << solution_gauss_seidel[3] << endl;

    return 0;
}
```

1.4 Метод вращений

10 Постановка задачи

Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

Вариант: 14

$$\begin{pmatrix} -7 & -5 & -9 \\ -5 & 5 & 2 \\ -9 & 2 & 9 \end{pmatrix}$$

11 Результаты работы

```
Iterations 7

Values:
-11.9023
4.06915
13.858

Vectors:
for lambda = -11.9023:
0.902897
-0.0289539
-0.428881

for lambda = 4.06915:
0.224018
0.883224
0.411985

for lambda = 13.858:
0.366869
-0.468057
0.803946
```

Рис. 4: Вывод программы

12 Исходный код

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

using namespace std;

const double epsilon = 1e-6;

double getMaxOffDiagonal(const vector<vector<double>>& A, int& p, int& q) {
    int n = A.size();
    double maxVal = 0.0;

    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (abs(A[i][j]) > maxVal) {
                maxVal = abs(A[i][j]);
                p = i;
                q = j;
            }
        }
    }

    return maxVal;
}

void rotateMatrix(vector<vector<double>>& A, int p, int q, vector<vector<double>>& V) {
    int n = A.size();
    double tau = (A[q][q] - A[p][p]) / (2.0 * A[p][q]);
    double t = (tau >= 0) ? 1.0 / (tau + sqrt(1.0 + tau * tau)) : -1.0 / (-tau + sqrt(1.0 + tau * tau));
    double c = 1.0 / sqrt(1.0 + t * t);
    double s = c * t;

    //   A
    double apq = A[p][q];
    A[p][q] = 0.0;
    A[q][p] = 0.0;
    A[p][p] = c * c * A[p][p] - 2.0 * c * s * apq + s * s * A[q][q];
    A[q][q] = s * s * A[p][p] + 2.0 * c * s * apq + c * c * A[q][q];

    for (int i = 0; i < n; ++i) {
        if (i != p && i != q) {
            double api = A[p][i];
            double aqi = A[q][i];
            A[p][i] = c * api - s * aqi;
            A[i][p] = A[p][i];
```

```
            A[q][i] = s * api + c * aqi;
            A[i][q] = A[q][i];
        }
    }

    for (int i = 0; i < n; ++i) {
        double vip = V[i][p];
        double viq = V[i][q];
        V[i][p] = c * vip - s * viq;
        V[i][q] = s * vip + c * viq;
    }
}

void findEigenvaluesAndEigenvectors(const vector<vector<double>>& A, vector<double>&
    eigenvalues, vector<vector<double>>& eigenvectors) {
    int n = A.size();
    eigenvectors = vector<vector<double>>(n, vector<double>(n, 0.0));

    for (int i = 0; i < n; ++i) {
        eigenvectors[i][i] = 1.0;
    }

    vector<vector<double>> B = A;

    int iterations = 0;
    while (true) {
        int p, q;
        double maxOffDiagonal = getMaxOffDiagonal(B, p, q);
        if (maxOffDiagonal < epsilon) //
            break;

        rotateMatrix(B, p, q, eigenvectors);

        iterations++;
    }

    cout << endl << "Iterations " << iterations << endl << endl;

    eigenvalues.clear();
    for (int i = 0; i < n; ++i) {
        eigenvalues.push_back(B[i][i]);
    }
}

int main() {
    vector<vector<double>> A = {
            {-7, -5, -9},
            {-5, 5, 2},
            {-9, 2, 9}
```

```
    };

    vector<double> eigenvalues;
    vector<vector<double>> eigenvectors;
    findEigenvaluesAndEigenvectors(A, eigenvalues, eigenvectors);

    cout << "Values:" << endl;
    for (double eigenvalue : eigenvalues) {
        cout << eigenvalue << " " << endl;
    }
    cout << endl;

    cout << "Vectors:" << endl;
    for (int i = 0; i < eigenvectors.size(); ++i) {
        cout << "for lambda = " << eigenvalues[i] << ": " << endl;
        for (double component : eigenvectors[i]) {
            cout << component << endl;
        }
        cout << endl;
    }
    return 0;
}
```

1.5 QR – разложение матриц

13 Постановка задачи

Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

Вариант: 14

$$\begin{pmatrix} 2 & -4 & 5 \\ -5 & -2 & -3 \\ 1 & -8 & -3 \end{pmatrix}$$

14 Результаты работы

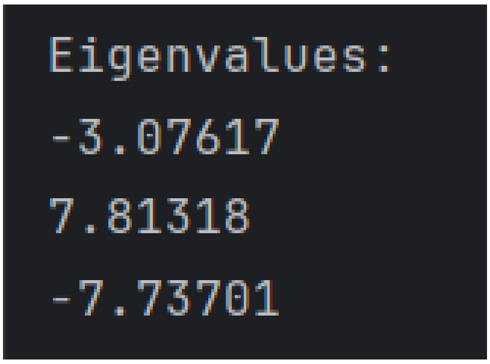

Рис. 5: Вывод программы

15 Исходный код

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

using namespace std;

vector<vector<double>> transposing(vector<vector<double>> matrix) {
    int n = matrix.size();
    vector<vector<double>> result(n, vector<double>(n));
    for (int i = 0; i < n; ++i) {
```

```
        for (int j = 0; j < n; ++j) {
            result[i][j] = matrix[j][i];
        }
    }
    return result;
}

double normalise(vector <double> vector){
    int n = vector.size();
    double mag = 0;
    for (int i=0; i<n; i++){
        mag += vector[i] * vector[i];
    }

    return sqrt(mag);
}

double dot_product(vector<double> first, vector<double> second){
    double sum = 0;
    int n = first.size();
    for (int i=0; i<n; i++){
        sum += first[i] * second[i];
    }
    return sum;
}

void standardize_matrix(vector<vector <double>>& matrix) {
    int n = matrix.size();
    for (int i = 0; i<n; i++){
        double mag = normalise(matrix[i]);
        for (int j=0;j<n;j++){
            matrix[i][j] /= mag;
        }
    }
}

vector <double> subtract_vector(vector<double> first, vector<double> second){
    int n = first.size();
    vector <double> res(n,0);
    for (int i = 0; i<n; i++){
        res[i] = first[i] - second[i];
    }
    return res;
}

vector<vector <double>> orthogonalize(vector<vector<double>> matrix) {
    int n = matrix.size();
    vector <vector <double>> transposed_matrix = transposing(matrix);
    vector <vector <double>> B(n, vector<double>(n,0));
```

```
    B[0] = transposed_matrix[0];
    for (int i=1; i<n; i++){
        vector <double> projection(n,0);
        for (int count=0; count<i; count++){
            double k = dot_product(transposed_matrix[i], B[count]) / dot_product(B[count], B[count]);
            for (int j = 0; j<n; j++){
                projection[j] += k * B[count][j];
            }
        }
        B[i] = subtract_vector(transposed_matrix[i], projection);
    }
    return B;
}

vector <vector <double>> matrix_product(vector <vector <double>> A, vector <vector <double>> B){
    int n = A.size();
    vector <vector <double>> R(n, vector<double>(n,0));
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            for(int k = 0; k < n; k++)
                R[i][j] += A[i][k] * B[k][j];
    return R;
}

void decompose_QR(vector<vector <double>> &matrix, vector<vector <double>> &Q, vector<vector <double>> &R){
    Q = orthogonalize(matrix);
    standardize_matrix(Q);
    Q = transposing(Q);
    R = matrix_product(transposing(Q), matrix);
}

vector <double> calculate_eigenvalues(vector<vector<double>> matrix, double epsilon)
{
    int n = matrix.size();
    vector <double> prev_eigenvalues(n);
    vector<vector<double>> Q, R;
    vector <double> cur_eigenvalues(n,0);
    while (true) {
        decompose_QR(matrix,Q,R);
        matrix = matrix_product(R,Q);
        for (int i = 0; i < n; i++) {
            if (i < n - 1 && abs(matrix[i + 1][i]) > 1e-7) {
                double b = -(matrix[i][i] + matrix[i + 1][i + 1]);
                double c = matrix[i][i] * matrix[i + 1][i + 1] - matrix[i][i + 1] * matrix[i + 1][i];
                double discriminant = b * b - 4 * c;
```

```cpp

                if (discriminant > 0) {
                    cur_eigenvalues[i] = 0.5 * (-b - sqrt(discriminant));
                    cur_eigenvalues[i + 1] = 0.5 * (-b + sqrt(discriminant));
                    i++;
                } else {
                    cur_eigenvalues[i] = (-b / 2);
                    cur_eigenvalues[i + 1] = (-b / 2);
                    i++;
                }
            } else {
                cur_eigenvalues[i] = matrix[i][i];
            }
        }
        bool ok = true;
        for (int i = 0; i < n; i++) {
            ok = ok && abs(cur_eigenvalues[i] - prev_eigenvalues[i]) < epsilon;
        }
        if (ok)
            break;
        prev_eigenvalues = cur_eigenvalues;
    }
    return prev_eigenvalues;
}

int main() {
    vector<vector<double>> matrix = {
            {2, -4, 5},
            {-5, -2, -3},
            {1, -8, -3}
    };

    vector<vector<double>> Q, R;
    decompose_QR(matrix, Q, R);

    double epsilon = 0.1;
    vector<double> eigenvalues = calculate_eigenvalues(matrix, epsilon);
    cout << "Eigenvalues:" << endl;
    for(int i = 0; i < 3; i++){
        cout << eigenvalues[i] << endl;
    }
    return 0;
}
```