Московский Авиационный Институт
(национальный исследовательский университет)

Факультет Прикладной математики и физики
Кафедра Вычислительной математики и программирования

Лабораторная работа 1
по дисциплине
«Численные методы»

Вариант 3

Студент: Будникова В. П.
Группа: М08-307Б-19
Руководитель: Ревизников Д. Л.

Москва 2022

Постановка задачи

Часть 1_1:

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.

СЛАУ:

$$\begin{cases} 9 \cdot x_1 - 5 \cdot x_2 - 6 \cdot x_3 + 3 \cdot x_4 = -8 \\ x_1 - 7 \cdot x_2 + x_3 = 38 \\ 3 \cdot x_1 - 4 \cdot x_2 + 9 \cdot x_3 = 47 \\ 6 \cdot x_1 - x_2 + 9 \cdot x_3 + 8 \cdot x_4 = -8 \end{cases}$$

Часть 1_2:

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.

СЛАУ:

$$\begin{cases} 13 \cdot x_1 - 5 \cdot x_2 = -66 \\ -4 \cdot x_1 + 9 \cdot x_2 - 5 \cdot x_3 = -47 \\ -x_2 - 12 \cdot x_3 - 6 \cdot x_4 = -43 \\ 6 \cdot x_3 + 20 \cdot x_4 - 5 \cdot x_5 = -74 \\ 4 \cdot x_4 + 5 \cdot x_5 = 14 \end{cases}$$

Часть 1_3:

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.

СЛАУ:

$$\begin{cases} -23 \cdot x_1 - 7 \cdot x_2 + 5 \cdot x_3 + 2 \cdot x_4 = -26 \\ -7 \cdot x_1 - 21 \cdot x_2 + 4 \cdot x_3 + 9 \cdot x_4 = -55 \\ 9 \cdot x_1 + 5 \cdot x_2 - 31 \cdot x_3 - 8 \cdot x_4 = -58 \\ x_2 - 2 \cdot x_3 + 10 \cdot x_4 = -24 \end{cases}$$

Часть 1_4:

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.

Матрица:

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

Часть 1_5:

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.

Матрица:

$$\begin{pmatrix} 5 & -5 & -6 \\ -1 & -8 & -5 \\ 2 & 7 & -3 \end{pmatrix}$$

Реализация и результаты работы

Часть 1_1:

Алгоритм LU-разложения:

```
class LU_Decomposition {
private:
    int n;
    std::vector<std::vector<double>> a;
    std::vector<std::vector<double>> l;
    std::vector<std::vector<double>> u;
    std::vector<std::vector<double>> p;
public:
    LU_Decomposition(const std::vector<std::vector<double>> &_a) {
        if (_a.size() != _a[0].size()) throw "Wrong matrix size";
        n = _a.size();
        a = _a;
        l = std::vector<std::vector<double>> (n, std::vector<double>(n, 0));
        u = std::vector<std::vector<double>> (n, std::vector<double>(n, 0));
        p = E(n);
        Make_L_and_U();
    }
    void PrintElements() {
        PrintMatrix("A:", a);
        PrintMatrix("L:", l);
        PrintMatrix("U:", u);
        PrintMatrix("L * U", l * u);
        if (p != E(n)) PrintMatrix("P:", p);
    }
    std::vector<double> Solve(const std::vector<double> &b) {
        if (b.size() != n) throw "Wrong vector size";
        std::vector<double> z = Make_z(b);
        return Make_x(z);
    }
private:
    void Make_L_and_U() {
        std::vector<std::vector<double>> ak;
        ak = a;
        int ind = FindMaxInd(ak, 0);
        ChangeStr(ak, 0, ind);
        ChangeStr(l, 0, ind);
        ChangeStr(p, 0, ind);
        l[0][0] = 1;
        for(int k = 1; k < n; ++k) {
            ind = FindMaxInd(a, k);
            ChangeStr(ak, k, ind);
            ChangeStr(l, k, ind);
            ChangeStr(p, k, ind);
            std::vector<std::vector<double>> new_a = ak;
            for (int i = k; i < n; ++i) {
                double mu = 0;
                bool fl = false;
                for (int j = k - 1; j < n; ++j) {
                    if (!fl) {
                        mu =  ak[i][j] / ak[k - 1][k - 1];
                        if (i > j) l[i][j] = mu;
                    }
                    fl = true;
                    if (i == j) l[i][j] = 1;
                    new_a[i][j] = ak[i][j] - mu * ak[k - 1][j];
                }
            }
```

```
            ak = new_a;
        }
        u = ak;
    }
    int FindMaxInd(const std::vector<std::vector<double>> &m, const int &stl{
        double max = std::abs(m[stl][stl]);
        int ind = stl;
        for (int i = 0; i < n ; ++i) {
            if (std::abs(m[i][stl]) > max) {
                max = std::abs(m[i][stl]);
                ind = i;
            }
        }
        return ind;
    }
    std::vector<double> Make_z(const std::vector<double> &b) {
        std::vector<double> z(n);
        z[0] = b[0];
        for (int i = 0; i < n; ++i) {
            double sum = 0;
            for (int j = 0; j < i; ++j) {
                sum += l[i][j] * z[j];
            }
            z[i] = b[i] - sum;
        }
        return z;
    }
    std::vector<double> Make_x(const std::vector<double> &z) {
        std::vector<double> x(n);
        for (int i = n - 1; i > -1; --i) {
            double sum = 0;
            for (int j = n - 1; j > i; --j) {
                sum += u[i][j] * x[j];
            }
            x[i] = (z[i] - sum) / u[i][i];
        }
        return x;
    }
};
```

Результаты работы алгоритма:

```
ЗАДАНИЕ 1.1
A
9           -5          -6          3
1           -7          1           0
3           -4          9           0
6           -1          9           8

b:
-8 38 47 -8

LU-разложение:
A:
9           -5          -6          3
1           -7          1           0
3           -4          9           0
6           -1          9           8

L:
1           0           0           0
0.111           1           0           0
0.333           0.362           1           0
0.667           -0.362          1.31            1

U:
9           -5          -6          3
0           -6.44           1.67            -0.333
0           0           10.4            -0.879
0           0           0           7.03

L * U
9           -5          -6          3
1           -7          1           0
3           -4          9           0
6           -1          9           8

x :
0 -5 3 -5
```

Часть 1_2:

Метод прогонки:

```
class Tridiagonal_Matrix_Algorithm {
private:
    int n;
    std::vector<double> a;
    std::vector<double> b;
    std::vector<double> c;
    std::vector<double> d;
    std::vector<double> p;
    std::vector<double> q;
    std::vector<double> x;
    std::vector<std::vector<double>> matr_a;

public:

    Tridiagonal_Matrix_Algorithm(const std::vector<std::vector<double>> &_a,
const std::vector<double> &_d) {

        if (_a.size() != _d.size()) throw "Wrong matrix or vector size";

        n = _d.size();
        a = std::vector<double>(n);
        b = std::vector<double>(n);
        c = std::vector<double>(n);
        d = _d;
        p = std::vector<double>(n);
        q = std::vector<double>(n);
        matr_a = _a;

        a[0] = 0;
        b[0] = matr_a[0][0];
        c[0] = matr_a[0][1];
        for (int i = 1; i < n - 1; ++i) {
            a[i] = matr_a[i][i - 1];
            b[i] = matr_a[i][i];
            c[i] = matr_a[i][i + 1];
        }
        a[n - 1] = matr_a[n - 1][n - 2];
        b[n - 1] = matr_a[n - 1][n - 1];
        c[n - 1] = 0;

        Make_P_andQ();
        x = Make_x();
    }
    void PrintElements() {
        PrintMatrix("A:", matr_a);
        PrintVec("d:", d);
        PrintVec("a:", a);
        PrintVec("b:", b);
        PrintVec("c:", c);
        PrintVec("d:", d);
        PrintVec("p:", p);
        PrintVec("q:", q);
    }
    void PrintAnswer() {
        PrintVec("x:", x);
    }
    std::vector<double> Ans() {
        return x;
    }
private:
    void Make_P_andQ() {
         p[0] = - c[0] / b[0];
```

```
        q[0] = d[0] / b[0];
        for (int i = 1; i < n - 1; ++i) {
            p[i] = - c[i] / (b[i] + a[i] * p[i - 1]);
            q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
        }
    }
    std::vector<double> Make_x() {
        std::vector<double> ans(n);
        ans[n - 1] = (d[n - 1] - a[n - 1] * q[n - 2]) / (b[n - 1] + a[n - 1]
* p[n - 2]);
        for(int i = n - 2; i > -1; --i) {
            ans[i] = p[i] * ans[i + 1] + q[i];
        }

        return ans;
    }

};
```

Результаты работы алгоритма:

```
ЗАДАНИЕ 1.2
A
13          -5          0           0           0
-4          9           -5          0           0
0           -1          -12         -6          0
0           0           6           20          -5
0           0           0           4           5

d:
-66 -47 -43 -74 14

Метод прогонки:
x:
-7 -5 6 -4 6

LU-разложение:
x:
-7 -5 6 -4 6
```

Часть 1_3:

Метод простых итераций

```
class Simple_Iteration_Method {
private:
    int n;
    double eps;
    double count;
    std::vector<double> b;
    std::vector<double> bet;
    std::vector<double> x;
    std::vector<std::vector<double>> a;
    std::vector<std::vector<double>> al;

public:
    Simple_Iteration_Method(const std::vector<std::vector<double>> &_a, const
std::vector<double> &_b, double _eps) {

        if (_a.size() != _b.size()) throw "Wrong matrix or vector size";
        n = _a.size();
        eps = _eps;
        b = _b;
        a = _a;
        Make_Alpha_Betta();
        x = Make_x();
    }

    void PrintElements() {
        PrintMatrix("A:", a);
        PrintVec("b:", b);
        PrintMatrix("Alpha:", al);
        PrintVec("Betta:", bet);
    }

    void PrintAnswer() {
        PrintVec("x:", x);
    }

    void PrintAnalysis() {
        std::cout << count << " итерации(ий) потребовалось для получения
ответа.\n\n";
    }
private:
    void Make_Alpha_Betta() {
        al = a;
        bet = b;
        for (int i = 0; i < n; ++i) {
            double diag_el = a[i][i];
            bet[i] /= diag_el;
            for (int j = 0; j < n; ++j) {
                al[i][j] /= -diag_el;
                if (i == j) al[i][j] = 0;
            }
        }
    }

    std::vector<double> Make_x() {
        double eps_k = eps + 1;
        x = bet;
        std::vector<double> x_prev = x;
        count = 0;
        double norm_al = Norm1(al);
        while(eps_k > eps) {
```

```
            x_prev = x;
            x = al*x_prev + bet;
            if (norm_al < 1) {
                eps_k = norm_al * Norm1(x - x_prev) / (1 - norm_al);
            } else {
                eps_k = Norm1(x - x_prev);
            }
            ++count;
        }
        return x;
    }
};
```

Метод Зейделя

```
class Zeidel_Method {
private:
    int n;
    double eps;
    double count;
    std::vector<double> b;
    std::vector<double> bet;
    std::vector<double> x;
    std::vector<std::vector<double>> a;
    std::vector<std::vector<double>> al;
    std::vector<std::vector<double>> al2;
public:
    Zeidel_Method(const std::vector<std::vector<double>> &_a, const
std::vector<double> &_b, double _eps) {

        if (_a.size() != _b.size()) throw "Wrong matrix or vector size";
        n = _a.size();
        eps = _eps;
        b = _b;
        a = _a;
        Make_Alpha_Betta();
        al2 = Al2(al);
        x = Make_x();
    }

    void PrintElements() {
        PrintMatrix("A:", a);
        PrintVec("b:", b);
        PrintMatrix("Alpha:", al);
        PrintMatrix("Alpha2:", al2);
        PrintVec("Betta:", bet);
    }

    void PrintAnswer() {
        PrintVec("x:", x);
    }

    void PrintAnalysis() {
        std::cout << count << " итерации(ий) потребовалось для получения
ответа.\n\n";
    }

private:

    void Make_Alpha_Betta() {
        al = a;
        bet = b;
        for (int i = 0; i < n; ++i) {
```

```cpp
            double diag_el = a[i][i];
            bet[i] /= diag_el;
            for (int j = 0; j < n; ++j) {
                al[i][j] /= -diag_el;
                if (i == j) al[i][j] = 0;
            }
        }
    }

    std::vector<double> Make_x() {
        double eps_k = eps + 1;
        count = 0;
        x = bet;
        std::vector<double> x_prev = x;


        double norm_al = Norm1(al);
        double norm_al2 = Norm1(al2);

        while(eps_k > eps) {
            x_prev = x;
            x = Zend_Multi(al, x_prev) + bet;
            if (norm_al < 1) {
                eps_k = norm_al2 * Norm1(x - x_prev) / (1 - norm_al);
            } else {
                eps_k = Norm1(x - x_prev);
            }
            ++count;
        }
        return x;
    }

    std::vector<double> Zend_Multi(const std::vector<std::vector<double>> &m,
const std::vector<double> &v) {
        std::vector<double> ans(n, 0);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                ans[i] += m[i][j] * (ans[j] + bet[j]);
            }
            for (int j = i; j < n; ++j) {
                ans[i] += m[i][j] * v[j];
            }
        }
        return ans;
    }

    std::vector<std::vector<double>> Al2(const
std::vector<std::vector<double>> &al) {
        std::vector<std::vector<double>> al2(n, std::vector<double>(n, 0));
        for(int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                al2[i][j] = al[i][j];
            }
        }
        return al2;
    }
};
```

Результаты работы алгоритмов:

```
ЗАДАНИЕ 1.3

A
-23         -7          5           2
-7          -21         4           9
9           5           -31         -8
0           1           -2          10

d:
-26 -55 -58 -24

точность:0.01

Метод простой итерации:
x:
1 2 3 -2

8 итерации(ий) потребовалось для получения ответа.

Метод Зейделя:
x:
1 2 3 -2

5 итерации(ий) потребовалось для получения ответа.
```

Часть 1_4:

Метод вращений:

```
class Rotation_Method {
private:
    int n;
    double eps;
    std::vector<double> eigenvalues;
    std::vector<std::vector<double>> a;
    std::vector<std::vector<double>> eigenvectors;
public:
    Rotation_Method(const std::vector<std::vector<double>> &_a, double _eps)
{
        n = _a.size();
        eps = _eps;
        a = _a;
        Go();
    }
    void PrintElements() {
        PrintMatrix("A:", a);
        std::cout << "eps:" << eps << '\n';
        PrintMatrix("Eigenvectors: ", eigenvectors);
```

```
        PrintMatrix("Fundamental Eigenvectors: ", FundamentalEigenvectors());
        PrintVec("Eigenvalues: ", eigenvalues);
    }
    void PrintAnswer() {
        PrintMatrix("Собственные вектора: ", eigenvectors);
        PrintVec("Собственные значения: ", eigenvalues);

    }
private:
    void Go() {
        std::vector<std::vector<double>> uk(n, std::vector<double>(n));
        std::vector<std::vector<double>> ak = a;
        eigenvalues = std::vector<double>(n);

        std::pair<int, int> max = FindMaxElem(ak);
        uk = Make_U(max, ak);
        ak = Transpose(uk) * ak * uk;
        double eps_k = Make_eps(ak);
        eigenvectors = uk;

        while(eps_k > eps) {
            max = FindMaxElem(ak);
            uk = Make_U(max, ak);
            ak = Transpose(uk) * ak * uk;
            eps_k = Make_eps(ak);
            eigenvectors = eigenvectors * uk;
        }
        for(int i = 0; i < n; ++i) {
            eigenvalues[i] = ak[i][i];
        }
    }

    std::vector<std::vector<double>> Make_U(const std::pair<int, int> &max,
const std::vector<std::vector<double>> &ak) {
        std::vector<std::vector<double>> u(n, std::vector<double>(n));

        double phi = std::atan(2 * ak[max.first][max.second] / (ak[max.first]
[max.first] - ak[max.second][max.second])) / 2;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (i == j) u[i][j] = 1;
                if (i != j) u[i][j] = 0;
                if (i == max.first and j == max.first) u[i][j] =
std::cos(phi);
                if (i == max.second and j == max.second) u[i][j] =
std::cos(phi);
                if (i == max.first and j == max.second) u[i][j] = -
std::sin(phi);
                if (i == max.second and j == max.first) u[i][j] =
std::sin(phi);
            }
        }

        return u;
    }

    std::pair<int, int> FindMaxElem(const std::vector<std::vector<double>>
&ak) {
        std::pair<int, int> max(1, 0);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                if (std::abs(ak[i][j]) > std::abs(ak[max.first][max.second]))
{
                    max.first = i;
                    max.second = j;
```

```
                }
            }
        }
        return max;
    }

    double Make_eps(const std::vector<std::vector<double>> &ak) {
        double eps = 0;
        for(int i = 0; i < n; ++i) {
            for(int j = i + 1; j < n; ++j) {
                eps += ak[i][j] * ak[i][j];
            }
        }
        return std::sqrt(eps);
    }

    std::vector<std::vector<double>> FundamentalEigenvectors() {
        std::vector<std::vector<double>> rez = eigenvectors;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                rez[i][j] /= eigenvectors[n - 1][j];
            }
        }
        return rez;
    }

};
```

Результаты работы алгоритма:

```
ЗАДАНИЕ 1.4

A
5            5            3
5            -4           1
3            1            2

точность:0.1

Собственные вектора:
0.831            -0.415            -0.371
0.36             0.909             -0.21
0.425            0.0409            0.904

Собственные значения:
8.71 -6.24 0.534
```

Часть 1_5:

Алгоритм QR – разложения:

```
class QR_Algorithm {
private:
    int n;
    double eps;
    Matrix a;
    Matrix q;
    Matrix r;
    std::vector<Complex> eigenvalues;

public:
    QR_Algorithm(const Matrix &_a, const double &_eps) {

        if (_a.size() != _a[0].size()) throw "Wrong matrix size";

        n = _a.size();
        eps = _eps;
        a = _a;
        q = Matrix (n, Vector(n, 0));
        r = Matrix (n, Vector(n, 0));

        Algorithm();
    }

    void PrintElements() {
        PrintMatrix("A:", a);
        PrintMatrix("Q:", q);
        PrintMatrix("R:", r);


        for (int i = 0; i < n; ++i) {
            std::cout.precision(3);
            std::cout << "lambda" << i + 1 << " = " << eigenvalues[i] <<
"\n";
        }
    }

private:
    std::pair<Matrix, Matrix> QR_Decomposition(const Matrix &a) {
        Matrix q, r;
        Matrix ak, hk;
        Vector vk(n);
        ak = a;

        vk[0] = ak[0][0] + Sign(ak[0][0])* EuclidianNorm(ak, 0);
        for (int i = 1; i < n; ++i) {
            vk[i] = a[i][0];
        }

        hk = E(n) - (2 / Scalar(vk, vk)) * (vk * vk);

        q = hk;
        ak = hk * ak;
        for (int k = 1; k < n - 1; ++k) {

            for (int i = 0; i < k; ++i) {
                vk[i] = 0;
            }
            vk[k] = ak[k][k] + Sign(ak[k][k])* EuclidianNorm(ak, k);
            for (int i = k + 1; i < n; ++i) {
                vk[i] = a[i][k];
            }
            hk = E(n) - (2 / Scalar(vk, vk)) * (vk * vk);
```

```
            q = q * hk;
            ak = hk * ak;
        }

        r = ak;

        return std::pair<Matrix, Matrix>{q, r};
    }

    void Algorithm() {
        double eps2_k = eps + 1;
        Matrix ak;
        ak = a;
        std::pair<Matrix, Matrix> qr = QR_Decomposition(ak);

        q = qr.first;
        r = qr.second;

        int counter = 0;
        while (eps2_k > eps) {

            ak = Transpose(qr.first) * ak * qr.first;

            qr = QR_Decomposition(ak);

            eps2_k = Eps(ak);

            ++counter;
        }

        FindEigenvalues(ak);

    }
    double Eps(Matrix ak) {
        double eps = 0;
        for (int i = 0; i < n - 2; ++i) {
            eps += ak[i + 2][i] * ak[i + 2][i];
        }
        return std::sqrt(eps);
    }
    void FindEigenvalues(const Matrix &m) {
        for (int i = 0; i < n - 1; ++i) {
            if (std::abs(m[i + 1][i]) > eps) {
                Solve(m[i][i], m[i][i + 1], m[i + 1][i], m[i + 1][i + 1]);
            } else {
                Complex l(m[i][i], 0);
                eigenvalues.push_back(l);
            }
        }
    }
    void Solve(const double &a1, const double &a2, const double &a3, const
double &a4) {
        double a = 1;
        double b = -(a1 + a4);
        double c = a1 * a4 - a2 * a3;
        double d = b * b - 4 * a * c;
        if (d > 0) {
            Complex l1 ((-b + std::sqrt(d)) / (2 * a), 0);
            Complex l2 ((-b - std::sqrt(d)) / (2 * a), 0);
            eigenvalues.push_back(l1);
            eigenvalues.push_back(l2);
        } else if (d == 0) {
            Complex l ((-b) / (2 * a), 0);
            eigenvalues.push_back(l);
        } else {
```

```
            Complex l1 ((-b) / (2 * a), std::sqrt(-d) / (2 * a));
            Complex l2 ((-b) / (2 * a), -std::sqrt(-d) / (2 * a));
            eigenvalues.push_back(l1);
            eigenvalues.push_back(l2);
        }
    }

    int Sign(const double &a) {
        return a < 0 ? -1 : a > 0 ? 1 : 0;
    }

};
```

Результаты работы алгоритма:

```
ЗАДАНИЕ 1.5

A
1           3           1           5           6
1           1           4           2           3
4           3           1           4           5
6           9           8           7           3
1           2           4           2           5

точность:0.001

A:
1           3           1           5           6
1           1           4           2           3
4           3           1           4           5
6           9           8           7           3
1           2           4           2           5

Q:
-0.135          -0.928          -0.00935          0.289           -0.194
-0.135          -0.209          0.762           -0.542          0.251
-0.539          -0.133          -0.551          -0.351          0.515
-0.809          0.262           0.195           0.161           -0.461
-0.135          0.095           0.278           0.688           0.649

R:
-7.42           -9.71           -8.23           -9.03           -7.01
-2.03e-16           -0.84           0.581           -3.56           -5.6
-2.97e-17           1.4         5.16            1.2         1.45
-1.27e-16           2.1         1.81            1.46            2.28
3.63e-16            -1.64           0.233           -0.338          4.03

lambda1 = 18.4 + i*(0)
lambda2 = -2.36 + i*(2.12)
lambda3 = -2.36 + i*(-2.12)
lambda4 = -1.52 + i*(0)
lambda5 = -1.63 + i*(0)
```

Выводы:

В данной лабораторной работе я реализовала LU-алгоритм разложения матриц, метод прогонки, метод простых итераций, метод Зейделя, с помощью этих алгоритмов решила заданные вариантом системы линейных уравнений. Также были реализованы: метод вращений, с помощью которого были найдены собственные векторы и собственные значения заданной вариантом симметрической матрицы; и алгоритм QR-разложения, с помощью которого были найдены собственные значения матриц (в том числе и комплексные).