Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 1

по курсу «Численные методы».

Тема: «Численные методы линейной алгебры».

Студент: Якимович А.И.
Группа: 80-308Б
Вариант: 1
Оценка:

Москва, 2017

Постановка задачи.

Реализовать методы для

решения СЛАУ:

1. LUP-разложение
2. Метод прогонки
3. Метод простых итераций (3.1) и метод Зейделя (3.2)

нахождения собственных векторов и собственных значений:

4. Метод вращений (Якоби)
5. QR-алгоритм

Описание методов.

1. Матрица A разлагается на произведение LU, где L – нижняя треугольная, U – верхняя треугольная. Для вычисления U выполнятся прямой ход метода Гаусса для которого вычисляются коэффициенты μ. Из этих коэффициентов строится матрица L. Решается система Lz = b, а затем система Ux = z. Эти действия эквивалентны обратному ходу метода Гаусса. Время работы O(n^3).
2. A – трехдиагональная матрица. b – коэффициенты элементов главной диагонали, a – под главной, c – над, d – элементы вектора . В ходе прямого хода определяются прогоночные коэффициенты P и Q. Затем обратным ходом вычисляется искомый вектор x. Для устойчивости необходимо: a[i] и c[i] отличны от нуля для i от 2 до n – 1, |b[i] >= |a[i]|+ |c[i]| для i от 1 до n, хотя бы для одного i достигается равенство. Время работы O(n).
3. 3.1 Применяется преимущественно для разреженных матриц. СЛАУ Ax = b приводится к эквивалентному виду x = β + αx (B – вектор, α – матрица). Один из способов такого приведения – способ Якоби. Используя его получаем итерационную формулу для метода простых итераций x[k] = β + αx[k - 1]. Достаточное условие сходимости || α || < 1.
   3.2 Применяется преимущественно для разреженных матриц. Является ускоренной версией предыдущего алгоритма. Ускорение достигается благодаря использованию на текущей итерации элементов вектора решения, уже вычисленных на текущей итерации (значения остальных компонент берутся из предыдущей итерации). Матрица α представляется в виде B + C, где B – нижняя треугольная(b[i][i] == 0), C – верхняя треугольная. Итерационная формула: x[k+1] = β + Bx[k+1] + Cx[k]. Достаточное условие сходимости || α || < 1.
4. A – симметрическая. Ищется преобразование подобия Λ = U^(-1) AU, где Λ – диагональная. Ее диагональные элементы будут собственными значениями. Векторы матрицы U – собственными векторами. На каждой итерации ищется максимальный по модулю элемент (его мы будем обнулять), в соответствии с ним строится матрица вращения U. Итерационная формула A[k+1] = U[k]^T * A[k] * U[k]. Критерий останова сумма поддиагональных элементов меньше заданной точности. Собственные значения – это диагональные элементы конечной матрицы A. Собственные векторы – векторы матрицы U = U[0]U[1]…U[k].

5. Ищется представление A = QR, где Q –ортогональная, R – верхняя треугольная, с помощью преобразования Хаусхолдера. Затем матрицы R и Q перемножаются в обратном порядке. То есть каждая итерация состоит из двух шагов A[k] = Q[k]R[k], A[k + 1] = R[k]Q[k]. Итерации продолжаются пока: для вещественных собственных значений – сумма поддиагональных элементов столбца не станет меньше или равной заданной точности, для комплексных |λ[k] – λ[k -1]| не станет меньше заданной точности. Комплексные λ находятся из решений квадратных уравнений, вещественные стоят на диагонали последней полученной матрицы A.

Общая информация.

Данная работа состоит из 5 модулей, которые позволяют решать различные задачи решения СЛАУ и нахождения собственных векторов и собственных значений. Полученные в ходе расчетов результаты сохраняются в отдельный файл. Поскольку составленные программы могут обрабатывать любой корректный ввод (в том числе все варианты заданий из лабораторных работ), они могут служить удобным примером для реализации собственных решателей, применяясь для сравнения результатов. Что касается технических деталей реализации, все программы написаны на языке C++.

Запуск программы.

Чтобы воспользоваться программой, необходимо скомпилировать файл main.cpp и запустить полученный исполняемый файл, например, для g++ на Windows:

g++ -std=c++11 main.cpp

a.exe

Исходные данные берутся из файлов с префиксом in папке data, выходные данные помещаются в файлы с префиксом out в папке data.

Исходный код.

```
//main.cpp
#include <iostream>
#include <algorithm>
#include <cmath>
#include <fstream>
#include "method1.hpp"

int main()
{
```

```
    method1();
    method2();
    method3_1();
    method3_2();
    method4();
    method5();
    return 0;
}


//matrix.hpp
#ifndef __matrix__
#define __matrix__

#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <fstream>


using namespace std;

class Matrix
{
public:

    friend ofstream &operator<<(ofstream &ofstr,  Matrix &m);
    friend ifstream &operator>>(ifstream &ifstr,  Matrix &m);
    Matrix();
    Matrix(int size);
    Matrix(int rows, int cols);
    void resize(int rows, int cols);
    double get(int row, int col);
    void set(int row, int col, double value);
    int getSize();
    int getM();
    int getN();
    Matrix sub(Matrix other);
    Matrix mul(double value);
    Matrix mul(Matrix other);
    vector <double> mul(vector <double> other);
    Matrix transpose();
    void identity();
    void copy(Matrix other);
    void swapRows(int index1, int index2);
    string toString();

private:
    vector < vector<double> > m_mat;
```

```
};



Matrix::Matrix() {}

Matrix::Matrix(int size)
{
   m_mat.resize(size);
   for (int i = 0; i < size; i++) {
      m_mat[i].resize(size);
   }
}


Matrix::Matrix(int rows, int cols)
{
   m_mat.resize(rows);
   for (int i = 0; i < cols; i++) {
      m_mat[i].resize(cols);
   }
}


void Matrix::resize(int rows, int cols)
{
   m_mat.resize(rows);
   for (int i = 0; i < cols; i++) {
      m_mat[i].resize(cols);
   }
}

double Matrix::get(int row, int col)
{
   return m_mat[row][col];
}

void Matrix::set(int row, int col, double value)
{
   m_mat[row][col] = value;
}

int Matrix::getSize()
{
   return m_mat.size();
}

int Matrix::getM()
{
   return getSize();
```

```
}

int Matrix::getN()
{
   return m_mat[0].size();
}

Matrix Matrix::sub(Matrix other)
{
   int n = getSize();
   Matrix res(n);

   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
         res.set(i, j, get(i, j) - other.get(i, j));
      }
   }

   return res;
}

Matrix Matrix::mul(double value)
{
   int n = getSize();
   Matrix res(n);

   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
         res.set(i, j, get(i, j) * value);
      }
   }

   return res;
}

Matrix Matrix::mul(Matrix other)
{
   int n = getSize();
   Matrix res(n);

   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
         double sum = 0.0;

         for (int k = 0; k < n; ++k) {
            sum += get(i, k) * other.get(k, j);
         }

         res.set(i, j, sum);
      }
```

```
    }

    return res;
}

vector <double> Matrix::mul(vector <double> other)
{
    int n = getSize();
    vector <double> res(n);

    for (int i = 0; i < n; ++i) {
        double sum = 0.0;

        for (int j = 0; j < n; ++j) {
            sum += get(i, j) * other[j];
        }

        res[i] = sum;
    }

    return res;
}

Matrix Matrix::transpose()
{
    int n = getSize();
    Matrix res(n);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            res.set(i, j, get(j, i));
        }
    }

    return res;
}

void Matrix::identity()
{
    int n = getSize();

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j) {
                set(i, j, 1.0);
            } else {
                set(i, j, 0.0);
            }
        }
    }
```

```
}

void Matrix::copy(Matrix other)
{
   for (int i = 0; i < getM(); ++i) {
      for (int j = 0; j < getN(); ++j) {
         set(i, j, other.get(i, j));
      }
   }
}

void Matrix::swapRows(int index1, int index2)
{
   swap(m_mat[index1], m_mat[index2]);
   //double tmp = m_mat[index1];

   //m_mat[index1] = m_mat[index2];
   //m_mat[index2] = tmp;
}

string Matrix::toString()
{
   // string res = Arrays.toString(m_mat[0]);

   // for (int i = 1; i < getM(); ++i) {
   //    res += '\n' + Arrays.toString(m_mat[i]);
   // }
   string res = "To string doesnt work\n";
   return res;
}



ifstream &operator >> (ifstream &ifstr,  Matrix &m)
{
   for (int i = 0; i < m.m_mat.size(); ++i) {
      for (int j = 0; j < m.m_mat[0].size(); ++j) {
         ifstr >> m.m_mat[i][j];
      }
   }
   return ifstr;
}

ofstream &operator << (ofstream &ofstr,  Matrix &m)
{
   for (int i = 0; i < m.m_mat.size(); ++i) {
      for (int j = 0; j < m.m_mat[0].size(); ++j) {
         ofstr.width(10);
         ofstr << m.get(i, j) << " ";
      }
```

```
        ofstr << "\n";
    }
    return ofstr;
}

#endif


//vector.hpp
#include <vector>
#include <cmath>
#include <fstream>

#define MAX_ITERATIONS 1000

using namespace std;

vector <double> vecAdd(vector <double> a, vector <double> b);
vector <double> vecSub(vector <double> a, vector <double> b);
double vecNormC(vector <double> v);
void inputVec(vector <double> &vec, ifstream &ifs);
void printVec(vector <double> &vec, ofstream &ofs);


vector <double> vecAdd(vector <double> a, vector <double> b)
{
    vector <double> c(a.size());
    for (int i = 0; i < a.size(); ++i) {
        c[i] = a[i] + b[i];
    }
    return c;
}


vector <double> vecSub(vector <double> a, vector <double> b)
{
    vector <double> c(a.size());
    for (int i = 0; i < a.size(); ++i) {
        c[i] = a[i] - b[i];
    }
    return c;
}


double vecNormC(vector <double> v)
{
    double res = 0.0;

    for (int i = 0; i < v.size(); ++i) {
        res = max(res, abs(v[i]));
```

```
    }
    return res;
}


void inputVec(vector <double> &vec, ifstream &ifs)
{
    for (int i = 0; i < vec.size(); ++i) {
        ifs >> vec[i];
    }
}


void printVec(vector <double> &vec, ofstream &ofs)
{
    for (int i = 0; i < vec.size(); ++i) {
        ofs.width(10);
        ofs << vec[i] << " ";
    }
    ofs << "\n";
}


//complex.hpp
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <fstream>

class Complex
{
public:
    Complex()
    {
        m_re = 0;
        m_im = 0;
    }

    Complex(double re, double im)
    {
        m_re = re;
        m_im = im;
    }

    double getRe()
    {
        return m_re;
    }
```

```
    void setRe(double re)
    {
        m_re = re;
    }

    double getIm()
    {
        return m_im;
    }

    void setIm(double im)
    {
        m_im = im;
    }

    Complex sub(Complex other)
    {
        return Complex(m_re - other.getRe(), m_im - other.getIm());
    }

    double abs()
    {
        return sqrt(pow(m_re, 2.0) + pow(m_im, 2.0));
    }


    // std::string toString()
    // {
    //     std::string res = string.valueOf(m_re);

    //     if (m_im != 0.0) {
    //         if (m_im > 0.0) {
    //             res += "+";
    //         }

    //         res += m_im + "*i";
    //     }

    //     return res;
    // }

private:
    double m_re;
    double m_im;
};


//method1.hpp
#ifndef __METHOD_1__
#define __METHOD_1__
```

```
#include <vector>
#include <cmath>
#include <fstream>
#include <iostream>
#include "method2.hpp"


using namespace std;

int m_detSign = 1;

void m_frontSub(Matrix &mat, vector <double> &vec, vector <double> &vecP, vector <double> &vecZ)
{
   int n = mat.getSize();

   for (int i = 0; i < n; ++i) {
      double sum = 0.0;

      for (int j = 0; j < i; ++j) {
         sum += mat.get(i, j) * vecZ[j];
      }

      vecZ[i] = vec[(int)vecP[i]] - sum;
   }
}


void m_backSub(Matrix &mat, vector <double> &vec, vector <double> &vecX)
{
   int n = mat.getSize();

   for (int i = n - 1; i >= 0; --i) {
      double sum = 0.0;

      for (int j = i + 1; j < n; ++j) {
         sum += mat.get(i, j) * vecX[j];
      }
      vecX[i] = (vec[i] - sum) / mat.get(i, i);
   }
}


bool m_lup(Matrix &mat, Matrix &matL, Matrix &matU, vector <double> &vecP)
{
   int n = mat.getSize();

   m_detSign = 1;
   matU.copy(mat);
```

```
    for (int i = 0; i < n; ++i) {
        vecP[i] = i;
    }

    for (int j = 0; j < n; ++j) {
        int row = -1;
        double max = 0.0;

        for (int i = j; i < n; ++i) {
            double element = abs(matU.get(i, j));

            if (element > max) {
                max = element;
                row = i;
            }
        }

        if (row == -1) {
            return false;
        }

        if (row != j) {
            m_detSign *= -1;
        }

        matU.swapRows(j, row);
        matL.swapRows(j, row);
        matL.set(j, j, 1);

        swap(vecP[j], vecP[row]);

        for (int i = j + 1; i < n; ++i) {
            double ratio = matU.get(i, j) / matU.get(j, j);

            for (int k = j; k < n; ++k) {
                matU.set(i, k, matU.get(i, k) - matU.get(j, k) * ratio);
            }

            matL.set(i, j, ratio);
        }
    }

    return true;
}



void matInverse(Matrix &mat, Matrix &matInv)
{
    int n = mat.getSize();
```

```
    vector <double> vec1(n);
    vector <double> vec2(n);
    vector <double> vecP(n);
    vector <double> vecX(n);
    Matrix matL(n);
    Matrix matU(n);
    Matrix matE(n);

    m_lup(mat, matL, matU, vecP);
    matE.identity();

    for (int j = 0; j < n; ++j) {
        for (int i = 0; i < n; ++i) {
            vec1[i] = matE.get(i, j);
        }

        m_frontSub(matL, vec1, vecP, vec2);
        m_backSub(matU, vec2, vecX);

        for (int i = 0; i < n; ++i) {
            matInv.set(i, j, vecX[i]);
        }
    }
}

double matDet(Matrix &mat)
{
    int n = mat.getSize();
    double res = 1.0;
    Matrix matL(n);
    Matrix matU(n);
    vector <double> vecP(n);

    m_lup(mat, matL, matU, vecP);

    for (int i = 0; i < n; ++i) {
        res *= matU.get(i, i);
    }

    return res * m_detSign;
}



bool lup(Matrix &mat, vector <double> &vec, vector <double> &vecX)
{
    int n = mat.getSize();
    Matrix matL(n);
    Matrix matU(n);
    vector <double> vecP(n);
```

```
    vector <double> vecZ(n);

    if (!m_lup(mat, matL, matU, vecP)) {
        return false;
    }

    m_frontSub(matL, vec, vecP, vecZ);
    m_backSub(matU, vecZ, vecX);

    return true;
}


void method1()
{
    ifstream fin("../data/in1.txt");
    ofstream fout("../data/out1.txt");
    int n, tmp;


    fin >> n;//int n;
    Matrix mat(n, n);
    vector <double> vec(n);
    vector <double> vecX(n);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            fin >> tmp;
            mat.set(i, j, tmp);
        }
    }

    for (int i = 0; i < n; ++i) {
        fin >> vec[i];
    }

    Matrix matInv(n);

    if (!lup(mat, vec, vecX)) {
        fout << "Degenerate matrix\n";
    } else {
        matInverse(mat, matInv);

        fout << "Answer (vector):\n";
        for (int i = 0; i < n; ++i) {
            fout << vecX[i] << " ";
        }
        fout << "\nInverse matrix:\n";
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
```

```
            fout << matInv.get(i, j) << " ";
        }
        fout << "\n";
    }
    fout << "Determinant: " << matDet(mat) << "\n";
  }
}

#endif


//method2.hpp
#ifndef __method_2__
#define __method_2__

#include <vector>
#include <cmath>
#include <fstream>
#include "method3_1.hpp"


using namespace std;


bool m_tmaCheck(Matrix &mat)
{
  int n = mat.getSize();

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      if (i == j - 1 || i == j || i == j + 1) {
        if (mat.get(i, j) == 0.0) {
          return false;
        }
      } else if (mat.get(i, j) != 0.0) {
        return false;
      }
    }
  }

  if (abs(mat.get(0, 0)) < abs(mat.get(0, 1)) ||
      abs(mat.get(n - 1, n - 1)) < abs(mat.get(n - 1, n - 2))) {
    return false;
  }

  for (int i = 1; i < n - 1; ++i) {
    if (abs(mat.get(i, i)) < abs(mat.get(i, i - 1)) + abs(mat.get(i, i + 1))) {
      return false;
    }
  }
```

```
    return true;
}



bool tma(Matrix &mat, vector <double> &vec, vector <double> &vecX, bool checkCond)
{
    if (checkCond && !m_tmaCheck(mat)) {
        return false;
    }

    int n = mat.getSize();
    vector <double> vecP(n, 0);
    vector <double> vecQ(n);

    vecP[0] = -mat.get(0, 1) / mat.get(0, 0);
    vecQ[0] = vec[0] / mat.get(0, 0);

    for (int i = 1; i < n - 1; ++i) {
        double a = mat.get(i, i - 1);
        double b = mat.get(i, i);
        double c = mat.get(i, i + 1);

        vecP[i] = -c / (b + a * vecP[i - 1]);
        vecQ[i] = (vec[i] - a * vecQ[i - 1]) / (b + a * vecP[i - 1]);
    }

    double resUp = vec[n - 1] - mat.get(n - 1, n - 2) * vecQ[n - 2];
    double resDown = mat.get(n - 1, n - 1) + mat.get(n - 1, n - 2) * vecP[n - 2];

    vecQ[n - 1] = resUp / resDown;
    vecX[n - 1] = vecQ[n - 1];

    for (int i = n - 2; i >= 0; --i) {
        vecX[i] = vecP[i] * vecX[i + 1] + vecQ[i];
    }

    return true;
}

void method2()
{
    ifstream fin("../data/in2.txt");
    ofstream fout("../data/out2.txt");
    //fout.setf(ios::left);
    int n;
    fin >> n;
    Matrix mat(n);
    fin >> mat;
```

```
    vector <double> vec(n);

    for (int i = 0; i < n; ++i) {
        fin >> vec[i];
    }

    vector <double> vecX(n);

    fout << "Метод 2: Метод прогонки\n";
    fout << "Исходная матрица:\n";
    fout << mat;
    fout << "Исходный вектор:\n" ;
    printVec(vec, fout);


    if (!tma(mat, vec, vecX, true)) {
        fout << "Матрица A не является трехдиагональной или не выполнено условие |b| >= |a| +
|c|\n";
    } else {
        fout << "Решение:\n";
        printVec(vecX, fout);
    }
}

#endif


//method3_1.hpp
#ifndef __method_3_1__
#define __method_3_1__

#include <vector>
#include <cmath>
#include <fstream>
#include <iostream>
#include "method3_2.hpp"

using namespace std;

bool simpleIteration(Matrix &mat, vector <double> &vec, vector <double> &vecX, double &eps, int
&cnt)
{
    int n = mat.getSize();
    Matrix matA(n);
    vector <double> vecB(n);
    vector <double> vecPrev(n);
    int iterCnt = 1;

    vecB = vec;
```

```
for (int j = 0; j < n; ++j) {
   if (mat.get(j, j) == 0) {
      int rowForSwap = -1;
      for (int i = 0; i < n; ++i) {
         if (j != i) {
            if (mat.get(i, j) != 0 && mat.get(j, i) != 0) {
               rowForSwap = i;
               break;
            }
         }
      }
      if (rowForSwap == -1) {
         return false;
      }
      mat.swapRows(j, rowForSwap);
   }
}


for (int i = 0; i < n; ++i) {

   if (mat.get(i, i) == 0) {

      return false;
   }

   for (int j = 0; j < n; ++j) {
      if (i != j) {
         matA.set(i, j, -mat.get(i, j) / mat.get(i, i));
      } else {
         matA.set(i, j, 0);
      }
   }

   vecB[i] = vecB[i] / mat.get(i, i);
}

vecPrev = vecB;

while (true) {
   vecX = matA.mul(vecPrev);//
   for (int i = 0; i < vecB.size(); ++i) {
      vecX[i] += vecB[i];//
   }

   // if (m_logger != Complex(0.0, 0.0)) {
   //     m_logger.writeln("Итерация #" + iterCnt + ": " + vecX);
   // }
   double normC = 0;
   for (int i = 0; i < n; ++i) {
```

```
            normC = max(normC, abs(vecX[i] - vecPrev[i]));//
        }
        if (normC <= eps) {
            break;
        }

        vecPrev = vecX;
        ++iterCnt;

        if (iterCnt > MAX_ITERATIONS) {
            return false;
        }
    }
    cnt = iterCnt;
    return true;
}


void method3_1()
{
    ifstream fin("../data/in3_1.txt");
    ofstream fout("../data/out3_1.txt");

    int n;
    double eps;
    fin >> n >> eps;
    Matrix mat(n);
    fin >> mat;
    vector <double> vec(n);
    inputVec(vec, fin);
    vector <double> vecX(n);

    fout << "Метод 3: Метод простых итераций\n";
    fout << "Исходная матрица:\n";
    fout << mat;
    fout << "Исходный вектор:\n";
    printVec(vec, fout);
    int cnt = 0;

    if (!simpleIteration(mat, vec, vecX, eps, cnt)) {
        fout << "Превышен лимит итераций\n";
        return;
    }

    fout << "Решение:\n";
    printVec(vecX, fout);
    fout << "Количество интераций " << cnt;
}

#endif
```

```
//method3_2.hpp
#ifndef __method_3_2__
#define __method_3_2__

#include <vector>
#include <cmath>
#include <fstream>
#include "method4.hpp"

using namespace std;

bool seidel(Matrix &mat, vector <double> &vec, vector <double> &vecX, double &eps, int &cnt)
{
   int n = mat.getSize();
   Matrix matA(n);
   Matrix matB(n);
   Matrix matC(n);
   vector <double> vecB(n);
   vector <double> vecPrev(n);
   int iterCnt = 1;

   vecB = vec;

   for (int j = 0; j < n; ++j) {
      if (mat.get(j, j) == 0) {
         int rowForSwap = -1;
         for (int i = 0; i < n; ++i) {
            if (j != i) {
               if (mat.get(i, j) != 0 && mat.get(j, i) != 0) {
                  rowForSwap = i;
                  break;
               }
            }
         }
         if (rowForSwap == -1) {
            return false;
         }
         mat.swapRows(j, rowForSwap);
      }
   }



   for (int i = 0; i < n; ++i) {
      if (mat.get(i, i) == 0) {
         return false;
      }
```

```
      for (int j = 0; j < n; ++j) {
         if (i != j) {
            matA.set(i, j, -mat.get(i, j) / mat.get(i, i));
         }
      }

      vecB[i] = vecB[i] / mat.get(i, i);
   }

   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
         if (j < i) {
            matB.set(i, j, matA.get(i, j));
         } else {
            matC.set(i, j, matA.get(i, j));
         }
      }
   }

   vecPrev = vecB;

   while (true) {
      vecX = vecAdd(matB.mul(vecX), vecAdd(matC.mul(vecPrev), vecB));

      if (vecNormC(vecSub(vecX, vecPrev)) <= eps) {
         cnt = iterCnt;
         return true;
      }

      vecPrev = vecX;
      ++iterCnt;

      if (iterCnt > MAX_ITERATIONS) {
         return false;
      }
   }
   cnt = iterCnt;
   return true;
}



void method3_2()
{
   ifstream fin("../data/in3_2.txt");
   ofstream fout("../data/out3_2.txt");
   int n;
   double eps;
   fin >> n >> eps;
   Matrix mat(n);
```

```
    fin >> mat;
    vector <double> vec(n);
    inputVec(vec, fin);
    vector <double> vecX(n);
    int cnt = 0;

    fout << "Метод 4: Метод Зейделя\n";
    fout << "Исходная матрица:\n";
    fout << mat;
    fout << "Исходный вектор:\n";
    printVec(vec, fout);

    seidel(mat, vec, vecX, eps, cnt);

    fout << "Решение:\n";
    printVec(vecX, fout);

    fout << "Количество интераций " << cnt;
}


#endif


//method4.hpp
#ifndef __method_4__
#define __method_4__

#include <vector>
#include <cmath>
#include <fstream>
#include "method5.hpp"

#define PI acos(-1.0)

using namespace std;

bool m_rotationCheck(Matrix &mat)
{
    int n = mat.getSize();

    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (mat.get(i, j) != mat.get(j, i)) {
                return false;
            }
        }
    }

    return true;
```

```
}


bool rotation(Matrix &mat, Matrix &matX, vector <double> &vecX, double &eps, int &iterCnt)
{
   if (!m_rotationCheck(mat)) {
      return false;
   }

   int n = mat.getSize();
   Matrix matA(n);
   iterCnt = 1;

   matA.copy(mat);
   matX.identity();

   while (true) {
      double max = 0.0;
      int maxRow = -1;
      int maxCol = -1;

      for (int i = 0; i < n; ++i) {
         for (int j = 0; j < n; ++j) {
            if (i < j) {
               double element = abs(matA.get(i, j));

               if (element > max) {
                  max = element;
                  maxRow = i;
                  maxCol = j;
               }
            }
         }
      }

      double a1 = matA.get(maxRow, maxRow);
      double a2 = matA.get(maxCol, maxCol);
      double fiZero = PI / 4.0;

      if (a1 != a2) {
         fiZero = 0.5 * atan(2.0 * matA.get(maxRow, maxCol) / (a1 - a2));
      }

      double fiCos = cos(fiZero);
      double fiSin = sin(fiZero);
      Matrix matU(n);

      matU.identity();
      matU.set(maxRow, maxRow, fiCos);
      matU.set(maxRow, maxCol, -fiSin);
```

```
      matU.set(maxCol, maxRow, fiSin);
      matU.set(maxCol, maxCol, fiCos);

      matX.copy(matX.mul(matU));
      matA = matU.transpose().mul(matA).mul(matU);

      // if (m_logger != Complex(0.0, 0.0)) {
      //     m_logger.writeln("Итерация #" + iterCnt + ":\n" + matA);
      // }

      double sum = 0.0;

      for (int i = 0; i < n; ++i) {
         for (int j = 0; j < n; ++j) {
            if (i < j) {
               sum += pow(matA.get(i, j), 2.0);
            }
         }
      }

      if (sqrt(sum) <= eps) {
         break;
      }

      ++iterCnt;

      if (iterCnt > MAX_ITERATIONS) {
         return false;
      }
   }

   for (int i = 0; i < n; ++i) {
      vecX[i] = matA.get(i, i);
   }

   return true;
}



void method4()
{
   ifstream fin("../data/in4.txt");
   ofstream fout("../data/out4.txt");
   int n, iterCnt;
   fin >> n;
   double eps;
   fin >> eps;
   Matrix mat(n);
   fin >> mat;
```

```
    Matrix matX(n);
    vector <double> vecX(n);

    fout << "Метод 4: Метод вращений\n";
    fout << "Исходная матрица:\n";
    fout <<  mat;

    //m_method.setLogger(logger);

    if (!rotation(mat, matX, vecX, eps, iterCnt)) {
        fout << "Матрица A не является симметричной или превышено количество итераций\n";
    } else {
        fout << "Количество интераций: " << iterCnt << "\n";
        fout << "Собственные значения:\n";

        for (int i = 0; i < n; ++i) {
            fout << "Lambda #" << i + 1 << ": " << vecX[i] << "\n";
        }

        fout << "Матрица собственных векторов:\n";
        fout << matX;
    }

    // output.close();
    // logger.close();
    // reader.close();
}

#endif


//method5.hpp
#ifndef __method_5__
#define __method_5__
#include <vector>
#include <cmath>
#include <fstream>
#include "matrix.hpp"
#include "vector.hpp"
#include "complex.hpp"


using namespace std;

void m_qr(Matrix &mat, Matrix &matQ, Matrix &matR)
{
    int n = mat.getSize();
    Matrix matA(n);
    Matrix matE(n);
    Matrix matResQ(n);
```

```
    matA.copy(mat);
    matE.identity();
    matResQ.identity();

    for (int j = 0; j < n - 1; ++j) {
      vector <double> vec(n);
      Matrix ratioBottom(n);
      double sum = 0.0;

      for (int i = j; i < n; ++i) {
        sum += pow(matA.get(i, j), 2.0);
      }

      vec[j] = matA.get(j, j) + (matA.get(j, j) >= 0.0 ? 1.0 : -1.0) * sqrt(sum);

      for (int i = j + 1; i < n; ++i) {
        vec[i] = matA.get(i, j);
      }

      sum = 0.0;

      for (int i = 0; i < n; ++i) {
        sum += pow(vec[i], 2.0);

        for (int k = 0; k < n; ++k) {
          ratioBottom.set(i, k, vec[i] * vec[k]);
        }
      }

      Matrix matH = matE.sub(ratioBottom.mul(2.0 / sum));

      matA = matH.mul(matA);
      matResQ = matResQ.mul(matH);
    }

    matQ.copy(matResQ);
    matR.copy(matA);
}


void m_qrSolveBlock(Matrix &mat, int &col, Complex &c1, Complex &c2)
{
    double b = -(mat.get(col, col) + mat.get(col + 1, col + 1));
    double c = mat.get(col, col) * mat.get(col + 1, col + 1) - mat.get(col, col + 1) * mat.get(col + 1, col);
    double d = pow(b, 2.0) - 4.0 * c;

    if (d >= 0.0) {
      double dRoot = sqrt(d);
```

```
        c1.setRe((-b - dRoot) / 2.0);
        c2.setRe((-b + dRoot) / 2.0);
    } else {
        double dRoot = sqrt(-d);

        c1.setRe(-b / 2.0);
        c1.setIm(-dRoot / 2.0);
        c2.setRe(-b / 2.0);
        c2.setIm(dRoot / 2.0);
    }
}


double m_qrNorm(Matrix &mat, int &col)
{
    double res = 0.0;

    for (int i = col + 1; i < mat.getSize(); ++i) {
        res += pow(mat.get(i, col), 2.0);
    }

    return sqrt(res);
}


bool qr(Matrix &mat, vector <Complex> &res, double &eps, int &iterCnt)
{
    int n = mat.getSize();
    Matrix matA(n);
    vector <Complex> prev(n, Complex(1e+6, 0.0));
    vector <bool> isComplex(n, true);
    iterCnt = 0;
    double error = eps + 1.0;

    if ((n & 1) == 1) {
        isComplex[n - 1] = false;
    }

    matA.copy(mat);

    while (error > eps) {
        Matrix matQ(n);
        Matrix matR(n);
        double errorRat = 0.0;
        double errorCom = 0.0;

        ++iterCnt;

        m_qr(matA, matQ, matR);
        matA = matR.mul(matQ);
```

```
      for (int j = 0; j < n; ++j) {
        if (isComplex[j]) {
          if (m_qrNorm(matA, j) <= eps) {
            if (j + 2 < n) {
              for (int i = n - 1; i > j; --i) {
                isComplex[i] = isComplex[i - 1];
                prev[i] = prev[i - 1];
                res[i] = res[i - 1];
              }
            } else {
              isComplex[n - 1] = false;
              prev[n - 1] = Complex(0.0, 0.0);
            }

            isComplex[j] = false;
            prev[j] = Complex(0.0, 0.0);
            res[j] = Complex(0.0, 0.0);
            --j;
          } else {
            m_qrSolveBlock(matA, j, res[j], res[j + 1]);

            errorCom = max(errorCom, res[j].sub(prev[j]).abs());
            errorCom = max(errorCom, res[j + 1].sub(prev[j + 1]).abs());

            ++j;
          }
        } else {
          res[j].setRe(matA.get(j, j));
          errorRat = max(errorRat, m_qrNorm(matA, j));
        }
      }

      for (int i = 0; i < n; ++i) {
        if (isComplex[i]) {
          prev[i].setRe(res[i].getRe());
          prev[i].setIm(res[i].getIm());
        }
      }

      error = max(errorRat, errorCom);

      if (iterCnt > MAX_ITERATIONS) {
        return false;
      }
    }
    return true;
  }
```

```
void method5()
{
   ifstream fin("../data/in5.txt");
   ofstream fout("../data/out5.txt");
   int n, iterCnt;
   fin >> n;
   double eps;
   fin >> eps;
   Matrix mat(n);
   fin >> mat;
   vector <Complex> res(n);

   fout << "Метод 5: QR - алгоритм\n";
   fout << "Исходная матрица:\n";
   fout << mat;

   for (int i = 0; i < n; ++i) {
      res[i] = Complex(0.0, 0.0);
   }

   if (!qr(mat, res, eps, iterCnt)) {
      fout << "Превышен лимит итераций.\n";
      return;
   }

   fout << "Количество интераций: " << iterCnt << "\n";
   fout << ("Собственные значения:\n");

   for (int i = 0; i < n; ++i) {
      fout << "Lambda #" << i + 1 << ": Re = " << res[i].getRe() << " Im = " << res[i].getIm() << "\n";
   }
}

#endif
```