Московский авиационный институт

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа № 1
по дисциплине
"Численные методы"

Студент: Балес А.И.
Преподаватель: Ревизников Д.Л.
Дата:
Оценка:
Подпись:

Москва, 2016

Метод 1 — LUP–разложение

Задание
Реализовать алгоритм LUP–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: 3

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

Описание алгоритма
LUP–разложение матрицы A представляет собой разложение матрицы A в произведение нижней и верхней треугольных матриц, а также матрицы перестановок P, т.е.

$$PA = LU$$

где L — нижняя треугольная матрица (матрица, у которой все элементы, находящиеся выше главной диагонали, равны нулю, $l_{ij} = 0$ при $i < j$), U — верхняя треугольная матрица (матрица, у которой все элементы, находящиеся ниже главной диагонали, равны нулю, $u_{ij} = 0$ при $i > j$).

LUP–разложение может быть построено с использованием метода Гаусса, используя формулу ниже для обнуления поддиагональных элементов:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \mu_i^{(k)} a_{kj}^{(k-1)}, \mu_i^{(k)} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, i = \overline{k+1, n}, j = \overline{k, n}$$

На первом этапе решается СЛАУ $Lz = b$. Поскольку матрица системы — нижняя треугольная, решение можно записать в явном виде:

$$z_1 = b_1, z_i = b_i - \sum_{j=1}^{i-1} l_{ij} z_j, i = \overline{2, n}$$

На втором этапе решается СЛАУ $Ux = z$ с верхней треугольной матрицей. Здесь, как и на предыдущем этапе, решение представляется в явном виде:

$$x_n = \frac{z_n}{u_{nn}}, x_i = \frac{1}{u_{ii}}(z_i - \sum_{j=i+1}^{n} u_{ij}x_j), i = \overline{n-1, 1}$$

Отметим, что второй этап эквивалентен обратному ходу метода Гаусса, тогда как первый соответствует преобразованию правой части СЛАУ в процессе прямого хода.

В результате прямого хода метода Гаусса можно вычислить определитель матрицы A исходной СЛАУ:

$$detA = (-1)^p a_{11} a_{22}^1 a_{33}^2 \cdot ... \cdot a_{nn}^{n-1}$$

где p — число перестановок строк в процессе прямого хода, учитываются соответствующие перемены знаков вследствие перестановок строк.

Реализация

```
TVector TSolve::_solveAx_is_b(const TMatrix& mL,
                              const TMatrix& mU,
                              const TVector& vB) {
    int tmpSz = vB.GetSize();
  TVector z(tmpSz);
    TVector x(tmpSz);
  z[0] = vB[0];
    for (int i = 1; i < tmpSz; ++i) {
        double tmpVal = 0.0;
        for (int j = 0; j < i; ++j)
            tmpVal += mL[i][j] * z[j];
        z[i] = vB[i] - tmpVal;
    }
    x[tmpSz - 1] = z[tmpSz - 1] / mU[tmpSz -
       1][tmpSz - 1];
    for (int i = tmpSz - 1; i >= 0; --i) {
        double tmpVal = 0.0;
        for (int j = i + 1; j < tmpSz; ++j) {
            tmpVal += mU[i][j] * x[j];
        }
        x[i] = (z[i] - tmpVal) / mU[i][i];
    }
    z.Clear();
```

```
    return x;
}

int TSolve::ToSolveByGauss() {
    TFRFF* tmpRead = _readFromFile(pathFrom, Gauss);
    if (tmpRead == NULL)
        return -1;
    _matrA.SetLink(tmpRead->matr);
    _vecB.SetLink(tmpRead->vec);
    ofstream log("solve1Gauss.log", ios::out);
    log << "|Method Gauss (LUP)| by Alexander Bales
       80-308" << endl << endl;
    TMatrix L(_matrA.GetSizeRow(),
       _matrA.GetSizeCol(), Identity);
    TMatrix U(_matrA);
    int cntSwitchRowsAndColumns = 0;
    int posPrecc = -1;
    int tmpSz = min(U.GetSizeRow(), U.GetSizeCol());
    for (int i = 0; i < tmpSz; ++i) {
        int posMax = U.FindPosMaxElemInColumn(i);
        if (tmpSz - 1 != i) {
            U.SwapRows(posMax, i);
            _matrA.SwapRows(posMax, i);
            _vecB.Swap(posMax, i);
            if (posMax != i)
                cntSwitchRowsAndColumns++;
            if (posPrecc != -1) {
                L.SwapRows(posMax, i);
                L.SwapColumns(posMax, i);
            }
        }

        for (int j = i + 1; j < tmpSz; ++j) {
            double koef = - U[j][i] / U[i][i];
            U[j][i] = 0.0;
            for (int k = i + 1; k < tmpSz; ++k) {
                U[j][k] = U[j][k] + koef * U[i][k];
            }
            L[j][i] = -koef;
        }
        posPrecc = posMax;
```

```
    }

    _vecX.SetLink(_solveAx_is_b(L, U, _vecB));
    L.Print(log, "L");
    U.Print(log, "U");
    output << "det(A) = ";
    double detA = 1.0;
    for (int i = 0; i < tmpSz; ++i)
        detA *= U[i][i];
    output << pow(-1.0, 1.0 *
       cntSwitchRowsAndColumns) * detA << endl <<
       endl;
    TMatrix reverseA(_matrA.GetSizeRow(),
       _matrA.GetSizeCol(), Zero);
    for (int i = 0; i < tmpSz; ++i) {
        TVector vec(_matrA.GetSizeRow());
        TVector calcVec;
        vec[i] = 1.0;
        calcVec.SetLink(_solveAx_is_b(L, U, vec));
        reverseA.AssignColumn(calcVec, i);
        calcVec.Clear();
        vec.Clear();
    }
    _writeToFile(pathTo);
    reverseA.Print(output, "A^(-1)");
    TMatrix check(_matrA * reverseA);
    check.Print(log, "A * A^(-1)");
    L.Clear();
    U.Clear();
    reverseA.Clear();
    check.Clear();
    _clear();
    delete tmpRead;
    return 0;
}

TVector TSolve::_solveAx_is_b(const TMatrix& mL,
                              const TMatrix& mU,
                              const TVector& vB) {
    int tmpSz = vB.GetSize();
  TVector z(tmpSz);
```

```
    TVector x(tmpSz);
  z[0] = vB[0];
    for (int i = 1; i < tmpSz; ++i) {
        double tmpVal = 0.0;
        for (int j = 0; j < i; ++j)
            tmpVal += mL[i][j] * z[j];
        z[i] = vB[i] - tmpVal;
    }
    x[tmpSz - 1] = z[tmpSz - 1] / mU[tmpSz -
       1][tmpSz - 1];
    for (int i = tmpSz - 1; i >= 0; --i) {
        double tmpVal = 0.0;
        for (int j = i + 1; j < tmpSz; ++j) {
            tmpVal += mU[i][j] * x[j];
        }
        x[i] = (z[i] - tmpVal) / mU[i][i];
    }
    z.Clear();
    return x;
}
```

Тестирование

Входной файл

```
4
9 -5 -6 3 -8
1 -7 1 0 38
3 -4 9 0 47
6 -1 9 8 -8
```

Выходной файл

```
solve1Gauss.log:

|Method Gauss (LUP)| by Alexander Bales 80-308

Matrix L:
1 0 0 0
0.111111 1 0 0
0.666667 -0.362069 1 0
0.333333 0.362069 0.764259 1
```

```
Matrix U:
9 -5 -6 3
0 -6.44444 1.66667 -0.333333
0 0 13.6034 5.87931
0 0 0 -5.37262

Matrix A * A^(-1):
1 5.55112e-17 0 2.22045e-16
6.93889e-18 1 -1.73472e-18 6.93889e-17
1.11022e-16 1.11022e-16 1 4.44089e-16
0 1.38778e-16 -2.77556e-17 1

res:

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

Vector B = (-8, 38, -8, 47)

Vector X = (0, -5, 3, -5)

Matrix A^(-1):
0.111 -0.149 -0.0418 0.133
0.0113 -0.168 -0.00425 0.0304
-0.0321 -0.0248 0.012 0.0804
-0.046 0.119 0.142 -0.186
```

Метод 2 — Метод прогонки

Задание
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: 3

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

Описание алгоритма
Метод прогонки является одним из эффективных методов решения СЛАУ с трех–диагональными матрицами, возникающих при конечно–разностной аппроксимации задач для обыкновенных дифференциальных уравнений (ОДУ) и уравнений в частных производных второго порядка и является частным случаем метода Гаусса. Рассмотрим следующую СЛАУ:

$$\begin{cases} a_1 = 0 \\ b_1x_1 + c_1x_2 = d_1 \\ a_2x_1 + b_2x_2 + c_2x_3 = d_2 \\ a_3x_2 + b_3x_3 + c_3x_4 = d_3 \\ ... \\ a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = d_{n-1} \\ a_nx_{n-1} + b_nx_n = d_n \\ c_n = 0 \end{cases}$$

решение которой будем искать в виде:

$$x_i = P_ix_{i+1} + Q_i, i = \overline{1,n} \tag{1}$$

где $P_i, Q_i, i = \overline{1,n}$ — прогоночные коэффициенты, подлежащие определению.

$$P_i = \frac{-c_i}{b_i + a_iP_{i-1}}, Q_i = \frac{d_i - a_iQ_{i-1}}{b_i + a_iP_{i-1}}, i = \overline{2,n-1}$$

$$P_1 = \frac{-c_1}{b_1}, Q_1 = \frac{d_1}{b_1}, i = 1$$

$$P_n = 0, Q_n = \frac{d_n - a_n Q_{n-1}}{b_n + a_n P_{n-1}}, i = n$$

Обратный ход метода прогонки осуществляется в соответствии с выражением (1):

$$\begin{cases} x_n = P_n x_{n+1} + Q_n = 0 \cdot x_{n+1} + Q_n = Q_n \\ x_{n-1} = P_{n-1} x_n + Q_{n-1} \\ ... \\ x_1 = P_1 x_2 + Q_1 \end{cases}$$

Общее число операций в методе прогонки равно $8n + 1$, т.е. пропорционально числу уравнений. Такие методы решения СЛАУ называют *экономичными*. Для сравнения число операций в методе Гаусса пропорционально n^3.

Для устойчивости метода прогонки достаточно выполнение следующих условий:

$$a_i \neq 0, c_i \neq 0, i = \overline{2, n-1}$$

$$|b_i| \geq |a_i| + |c_i|, i = \overline{1, n}$$

Реализация

```
int TSolve::ToSolveByTripleDiagMatrix() {
    TFRFF* tmpRead = _readFromFile(pathFrom,
       TripleDiagMatrix);
    if (tmpRead == NULL)
        return -1;
    _matrA.SetLink(tmpRead->matr);
    _vecB.SetLink(tmpRead->vec);
    ofstream log("solve1TripleDiagMatrix.log",
       ios::out);
    log << "|Method TripleDiagMatrix| by Alexander
       Bales 80-308" << endl << endl;
```

```
    double P, Q;
    try {
        P = -_matrA[0][1] / _matrA[0][0];
        Q = _vecB[0] / _matrA[0][0];
    }
    catch (const out_of_range& e) {
        cerr << "Out of range: " << e.what() <<
           endl;
    }
    _vecX = _vecB;
    _findSolve(P, Q, 1, _vecX, log);
    _writeToFile(pathTo);
    _clear();
    delete tmpRead;
    return 0;
}

void TSolve::_findSolve(double P, double Q, int n,
   TVector& x, ofstream& log) {
    if (n == x.GetSize()) {
        x[n - 1] = Q;
        log << "P_" << n << " = " << 0 << endl;
        log << "Q_" << n << " = " << Q << endl;
    } else {
        try {
            _findSolve(-_matrA[n][2] /
               (_matrA[n][0] * P + _matrA[n][1]),
                        (_vecB[n] - _matrA[n][0] *
                           Q) / (_matrA[n][0] * P +
                           _matrA[n][1]),
                        n + 1, x, log);
            log << "P_" << n << " = " << P << endl;
            log << "Q_" << n << " = " << Q << endl;
            x[n - 1] = P * x[n] + Q;
        }
        catch (const out_of_range& e) {
            cerr << e.what() << endl;
        }
    }
}
```

Тестирование

Входной файл

```
5
13 -5 0 -66
-4 9 5 -47
-1 -12 -6 -43
6 20 -5 -74
0 4 5 14
```

Выходной файл

```
solve1TripleDiagMatrix.log:

|Method TripleDiagMatrix| by Alexander Bales 80-308

P_5 = 0
Q_5 = 3.5
P_4 = 0.29722
Q_4 = -6.03646
P_3 = -0.529572
Q_3 = 4.59145
P_2 = -0.670103
Q_2 = -9.02062
P_1 = 0.384615
Q_1 = -5.07692

res:

Matrix A:
13 -5 0
-4 9 5
-1 -12 -6
6 20 -5
0 4 5

Vector B = (-66, -47, -43, -74, 14)

Vector X = (-10.4, -13.9, 7.24, -5, 3.5)
```

Метод 3 — Метод простых итераций/метод Зейделя

Задание
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: 3

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

Описание алгоритма
При большом числе уравнений прямые методы решения СЛАУ (за исключением метода прогонки) становятся труднореализуемыми на ЭВМ прежде всего из–за сложности хранения и обработки матриц большой размерности.

Методы последовательных приближений, в которых при вычислении последующего приближения решения используются предыдущие, уже известные приближенные решения, называются *итерационными*.

Рассмотрим СЛАУ:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{cases}$$

с невырожденной матрицей ($detA \neq 0$).

Приведем СЛАУ к эквивалентному виду:

$$\begin{cases} x_1 = \beta_1 + a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \\ x_2 = \beta_2 + a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \\ ... \\ x_n = \beta_n + a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n \end{cases}$$

или в векторно–матричной форме:

$$x = \beta + \alpha x$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \alpha = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ldots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

Такое приведение может быть выполнено различными способами. Один из них показан ниже.

$$\beta_i = \frac{b_i}{a_{ii}}; a_{ij} = -\frac{a_{ij}}{a_{ii}}, i, j = \overline{1, n}, i \neq j; a_{ij} = 0, i = j, i = \overline{1, n}$$

Метод простых итераций сходится к единственному решению СЛАУ при любом начальном приближении $x^{(0)}$, если какая-либо норма матрицы α эквивалентной системы меньше единицы $\|\alpha\| < 1$

Если используется метод Якоби, то достаточным условием сходимости является *диагональное преобладание матрицы* A, т.е. $|a_{ii}| > \sum\limits_{j=1, i\neq j}^{n} |a_{ij}|$ (для каждой строки матрицы A модули элементов, стоящих на главной диагонали, больше суммы модулей недиагональных элементов).

Поскольку $\|\alpha\| < 1$ является только достаточным (не необходимым) условием сходимости метода простых итераций, то итерационный процесс может сходиться и в случае, если оно не выполнено. Тогда критерием окончания итераций может служить неравенство $\|x^{(k)} - x^{(k-1)}\| \leq \varepsilon$.

Метод Зейделя

Метод простых итераций довольно медленно сходится. Для его ускорения существует *метод Зейделя*, заключающийся в том, что при вычислении компонента x_i^{k+1} вектора неизвестных на $(k+1)$–й итерации используется $x_1^{k+1}, x_2^{k+1}, ..., x_{i-1}^{k+1}$ уже вычисленные на $(k+1)$–й итерации. Значения остальных компонент $x_{i+1}^k, x_{i+2}^k, ..., x_n^k$ берутся из предыдущей итерации.

$$x^{k+1} = \beta + Bx^{k+1} + Cx^k$$

где B — нижняя треугольная матрица с диагональными элементами, равными нулю, а C — верхняя треугольная матрица с диагональными элементами, отличными от нуля, $\alpha = B + C$.

Реализация

```
int TSolve::ToSolveBySimpleIterations() {
    TFRFF* tmpRead = _readFromFile(pathFrom,
       SimpleIter);
    if (tmpRead == NULL)
        return -1;
    _matrA.SetLink(tmpRead->matr);
    _vecB.SetLink(tmpRead->vec);
    if (abs(_matrA.GetNorm()) >= 1.0) {
        cerr << "Error!!!" << endl;
        return -1;
    }
    log.open("solve1SimpleIter.log", ofstream::out);
    log << "|Method Simple Iterations| by Alexander
       Bales 80-308" << endl << endl;
    log << "|A| = " << _matrA.GetNorm() << endl <<
       endl;
    _vecX = _vecB;
    TVector vecRes = _vecX + (_matrA *
       _vecX.Rotate()).Rotate();
    _vecB.Print(log, "B");
    _matrA.Print(log, "A");
    double tmp = abs(_matrA.GetNorm());
    double eps_k = 1.0 * tmp / (1.0 - tmp) *
       (vecRes - _vecX).GetNorm();
    for (int i = 1; eps_k > eps; ++i) {
        _vecX = vecRes;
        log << "x_" << i - 1 << " = (";
        _vecX.Print(log);
        log << "); ";
        vecRes = _vecB + (_matrA *
           _vecX.Rotate()).Rotate();
```

```
            log << "x_" << i << " = (";
            vecRes.Print(log);
            log << "); ";
            eps_k = tmp / (1.0 - tmp) * (vecRes -
               _vecX).GetNorm();
            log << "eps(" << i << ") = " << eps_k <<
               endl;
        }
        for (int i = 0; i < vecRes.GetSize(); ++i) {
            output << vecRes[i] << endl;
        }
        _writeToFile(pathTo);
        _clear();
        delete tmpRead;
        return 0;
}
```

Тестирование

Входной файл

```
4
-23 -7 5 2 -26
-7 -21 4 9 -55
9 5 -31 -8 -58
0 1 -2 10 -24
```

Выходной файл

```
solve1SimpleIter.log:
|Method Simple Iterations| by Alexander Bales 80-308

|A| = 0.952

Vector B = (1.13, 2.62, 1.87, -2.4)

Matrix A:
0 -0.304 0.217 0.087
-0.333 0 0.19 0.429
0.29 0.161 0 -0.258
-0 -0.1 0.2 0
```

```
x_0 = (0.531 1.57 3.24 -2.29); x_1 = (1.16 2.08 2.87 -1.91); eps(1) = 37.7
x_1 = (1.16 2.08 2.87 -1.91); x_2 = (0.955 1.96 3.04 -2.03); eps(2) = 12.2
x_2 = (0.955 1.96 3.04 -2.03); x_3 = (1.02 2.01 2.99 -1.99); eps(3) = 3.94
x_3 = (1.02 2.01 2.99 -1.99); x_4 = (0.997 2 3 -2); eps(4) = 1.13
x_4 = (0.997 2 3 -2); x_5 = (1 2 3 -2); eps(5) = 0.31
x_5 = (1 2 3 -2); x_6 = (1 2 3 -2); eps(6) = 0.0857
x_6 = (1 2 3 -2); x_7 = (1 2 3 -2); eps(7) = 0.0218
x_7 = (1 2 3 -2); x_8 = (1 2 3 -2); eps(8) = 0.0059

solve1Zeydel.log:
|Method Zeydel| by Alexander Bales 80-308

Vector B = (1.13, 2.62, 1.87, -2.4)

Matrix A:
0 -0.304 0.217 0.087
-0.333 0 0.19 0.429
0.29 0.161 0 -0.258
-0 -0.1 0.2 0

x_0 = (0.531 1.77 2.93 -1.99); x_1 = (1.06 1.97 3.01 -2); eps(1) = 136
x_1 = (1.06 1.97 3.01 -2); x_2 = (1.01 2 3 -2); eps(2) = 14.2
x_2 = (1.01 2 3 -2); x_3 = (1 2 3 -2); eps(3) = 2.21
x_3 = (1 2 3 -2); x_4 = (1 2 3 -2); eps(4) = 0.171
x_4 = (1 2 3 -2); x_5 = (1 2 3 -2); eps(5) = 0.0257
x_5 = (1 2 3 -2); x_6 = (1 2 3 -2); eps(6) = 0.00353

res:
Matrix A:
0 -0.304 0.217 0.087
-0.333 0 0.19 0.429
0.29 0.161 0 -0.258
-0 -0.1 0.2 0

Vector B = (1.13, 2.62, 1.87, -2.4)

Vector X = (1, 2, 3, -2)
```

Метод 4 — Метод вращений

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

Описание алгоритма
Метод вращений Якоби применим только для симметрических матриц $A_{n\times n}$ $(A = A^T)$ и решает полную проблему собственных значений и собственных векторов таких матриц. Он основан на отыскании с помощью итерационных процедур матрицы U в преобразовании подобия $\Lambda = U^{-1}AU$, а поскольку для симметрических матриц A матрица преобразования подобия U является ортогональной $(U^{-1} = U^T)$, то $\Lambda = U^T AU$, где Λ — диагональная матрица с собственными значениями на главной диагонали.

$$\Lambda = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \ddots & \cdots \\ 0 & \cdots & \lambda_n \end{pmatrix}$$

Пусть дана симметрическая матрица A. Требуется для нее вычислить с точностью ε все собственные значения и соответствующие им собственные векторы. Алгоритм метода вращений следующий:

Пусть известна матрица $A^{(k)}$ на k–й итерации, при этом для $k = 0$ $A^{(0)} = A$.
1. Выбирается максимальный по модулю недиагональный элемент $a_{ij}^{(k)}$ матрицы $A^{(k)}(|a_{ij}^{(k)}|) = \max\limits_{l<m} |a_{lm}^{(k)}|$.
2. Ставится задача найти такую ортогональную матрицу $U^{(k)}$, чтобы в результате преобразования подобия $A(k+1) = U^{(k)T}A^{(k)}U^{(k)}$ произошло обнуление элемента $a_{ij}^{(k+1)}$ матрицы $A^{(k+1)}$. В качестве ортогональной

матрицы выбирается матрица вращения, имеющая следующий вид:

$$U^{(k)} = \begin{array}{c} \\ i \\ j \end{array} \begin{array}{c} \begin{array}{cc} i & j \end{array} \\ \begin{pmatrix} cos\varphi^{(k)} & -sin\varphi^{(k)} \\ sin\varphi^{(k)} & cos\varphi^{(k)} \end{pmatrix} \end{array}$$

Угол вращения $\varphi^{(k)}$ определяется из условия $a_{ij}^{(k+1)} = 0$:

$$\varphi^{(k)} = \frac{1}{2} arctg \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}}; a_{ii}^{(k)} = a_{jj}^{(k)}, \varphi^{(k)} = \frac{\pi}{4}$$

3. Строится матрица $A^{(k+1)}$:

$$A^{(k+1)} = U^{(k)T} A^{(k)} U^{(k)}$$

в которой элемент $a_{ij}^{(k+1)} \approx 0$.

В качестве критерия окончания итерационного процесса используется условие малости суммы квадратов внедиагональных элементов:

$$t(A^{(k+1)}) = (\sum_{l,m;l<m} (a_{lm}^{(k+1)})^2)^{\frac{1}{2}}$$

Реализация

```
int TSolve::ToSolveByRotateMethod() {
    TFRFF* tmpRead = _readFromFile(pathFrom,
       Rotate);
    if (tmpRead == NULL)
        return -1;
    _matrA.SetLink(tmpRead->matr);
    ofstream log("solve1RotateMethod.log",
       ios::out);
    log << "|Method Rotate| by Alexander Bales
       80-308" << endl << endl;
    TMatrix rotateMatr( _matrA.GetSizeRow(),
                        _matrA.GetSizeCol(),
                        Identity );
    TMatrix A(_matrA);
    TMatrix OwnVectors(rotateMatr);
    //ofstream urs("tmp.log", ios::out);
    int cnt = 0;
```

```
while (_t(A) > eps) {
    cout << _t(A) << endl;
    log << "|" << cnt++ + 1 << " iteration|" <<
       endl;
    log <<
       "*******************************************"
       << endl;
    A.Print(log, "A");
    pair<int, int> pos =
       A.FindPosMaxNotDiagElem();
    int i = pos.first, j = pos.second;
    log << "maxPos = (" << i + 1 << "; " << j +
       1 << ");" << endl << endl;
    double angel = 0.0;
    try {
        angel = A[i][i] == A[j][j] ? M_PI / 4 :
                        .5 * atan(2 * A[i][j] /
                                (A[i][i] -
                                   A[j][j]));
    }
    catch (const out_of_range& e) {
        cerr << "Out of range: " << e.what() <<
           endl;
    }
    rotateMatr[i][i] = rotateMatr[j][j] =
       cos(angel);
    rotateMatr[i][j] = -sin(angel);
    rotateMatr[j][i] = -rotateMatr[i][j];
    rotateMatr.Print(log, "rotateMatr");
    A = rotateMatr.Rotate() * A * rotateMatr;
    OwnVectors = OwnVectors * rotateMatr;
    rotateMatr[i][i] = rotateMatr[j][j] = 1.0;
    rotateMatr[i][j] = rotateMatr[j][i] = 0.0;
    OwnVectors.Print(log,
       "curMultiplyRotateMatrix");
    log <<
       "###########################################"
       << endl << endl;
}
_writeToFile(pathTo);
for (int i = 0; i < min(A.GetSizeRow(),
```

```
        A.GetSizeCol()); i++)
          output << "l_" << i + 1 << " = " << A[i][i]
             << endl;
      output << endl;
      for (int i = 0; i < OwnVectors.GetSizeCol();
         i++) {
          output << "x_" << i << " = (";
          for (int j = 0; j <
             OwnVectors.GetSizeRow(); j++) {
              output << OwnVectors[j][i];
              if (j != OwnVectors.GetSizeRow() - 1)
                  output << ", ";
          }
          output << ");" << endl;
      }
      _clear();
      A.Clear();
      OwnVectors.Clear();
      rotateMatr.Clear();
      return 0;
}
```

Тестирование

Входной файл

```
3
5 5 3
5 -4 1
3 1 2
0.001
```

Выходной файл

```
solve1RotateMethod.log:
|Method Rotate| by Alexander Bales 80-308

|1 iteration|
*******************************************
Matrix A:
5 5 3
5 -4 1
```

```
3 1 2

maxPos = (1; 2);

Matrix rotateMatr:
0.9135 -0.406839 0
0.406839 0.9135 0
0 0 1

Matrix curMultiplyRotateMatrix:
0.9135 -0.406839 0
0.406839 0.9135 0
0 0 1

##########################################

|2 iteration|
******************************************
Matrix A:
7.22681 4.44089e-16 3.14734
4.44089e-16 -6.22681 -0.307016
3.14734 -0.307016 2

maxPos = (1; 3);

Matrix rotateMatr:
0.905216 0 -0.424952
0 1 0
0.424952 0 0.905216

Matrix curMultiplyRotateMatrix:
0.826915 -0.406839 -0.388194
0.368277 0.9135 -0.172887
0.424952 0 0.905216

##########################################

|3 iteration|
******************************************
Matrix A:
8.70433 -0.130467 0
```

```
-0.130467 -6.22681 -0.277915
8.32667e-17 -0.277915 0.522485

maxPos = (2; 3);

Matrix rotateMatr:
1 0 0
0 0.999156 -0.0410727
0 0.0410727 0.999156

Matrix curMultiplyRotateMatrix:
0.826915 -0.422439 -0.371157
0.368277 0.905628 -0.210261
0.424952 0.0371796 0.904452

##############################################

|4 iteration|
*********************************************
Matrix A:
8.70433 -0.130357 0.00535863
-0.130357 -6.23824 0
0.00535863 -2.08167e-17 0.53391

maxPos = (1; 2);

Matrix rotateMatr:
0.999962 0.00872288 0
-0.00872288 0.999962 0
0 0 1

Matrix curMultiplyRotateMatrix:
0.830568 -0.41521 -0.371157
0.360363 0.908806 -0.210261
0.424612 0.040885 0.904452

##############################################

|5 iteration|
*********************************************
Matrix A:
```

```
8.70546 -1.38778e-17 0.00535843
0 -6.23937 4.67427e-05
0.00535843 4.67427e-05 0.53391

maxPos = (3; 1);

Matrix rotateMatr:
1 0 -0.000655741
0 1 0
0.000655741 0 1

Matrix curMultiplyRotateMatrix:
0.830324 -0.41521 -0.371701
0.360225 0.908806 -0.210497
0.425205 0.040885 0.904173

###############################################

res:
Matrix A:
5 5 3
5 -4 1
3 1 2

l_1 = 8.71
l_2 = -6.24
l_3 = 0.534

x_0 = (0.83, 0.36, 0.425);
x_1 = (-0.415, 0.909, 0.0409);
x_2 = (-0.372, -0.21, 0.904);
```

Метод 5 — QR–разложение

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

Описание алгоритма
При решении полной проблемы собственных значений для несимметричных матриц эффективным является подход, основанный на приведении матриц к подобным, имеющим треугольный или квазитреугольный вид. Одним из наиболее распространенных методов этого класса является QR–алгоритм, позволяющий находить как вещественные, так и комплексные собственные значения.

В основе QR–алгоритма лежит представление матрицы в виде $A = QR$, где Q — ортогональная матрица ($Q^{-1} = Q^T$), а R — верхняя треугольная. Такое разложение существует для любой квадратной матрицы. Одним из возможных подходов к построению QR разложения является использование преобразования Хаусхолдера, позволяющего обратить в нуль группу поддиагональных элементов столбца матрицы.

Преобразование Хаусхолдера осуществляется с использованием матрицы Хаусхолдера, имеющей следующий вид:

$$H = E - \frac{2}{v^T v} v v^T$$

где v — произвольный ненулевой вектор–столбец, E — единичная матрица, vv^T — квадратная матрица того же размера.

Вектор v определяется следующим образом:

$$v = b + sign(b_1)\|b\|_2 e_1$$

где $\|b\|_2 = (\sum\limits_i b_i^2)^{\frac{1}{2}}$ — евклидова норма вектора, $e_1 = (1, 0, ..., 0)^T$.

Применяя описанную процедуру с целью обнуления поддиагональных элементов каждого из столбцов исходной матрицы, можно за фиксированное число шагов получить ее QR–разложение.

Процедура QR–разложения многократно используется в QR–алгоритме вычисления собственных значений. Строится следующий итерационный процесс:

$A^{(0)} = A$
$A^{(0)} = Q^{(0)}R^{(0)}$ — производится QR–разложение
$A^{(1)} = R^{(0)}Q^{(0)}$ — производится перемножение матриц
...
$A^{(k)} = Q^{(k)}R^{(k)}$ — разложение
$A^{(k+1)} = R^{(k)}Q^{(k)}$ — перемножение

Таким образом, каждому вещественному собственному значению будет соответствовать столбец со стремящимися к нулю поддиагональными элементами и в качестве критерия сходимости итерационного процесса для таких собственных значений можно использовать следующее неравенство:

$$(\sum_{l=m+1} (a_{lm}^{(k)})^2)^{\frac{1}{2}} \leq \varepsilon$$

При этом соответствующее собственное значение принимается равным диагональному элементу данного столбца.

Если в ходе итераций прослеживается комплексно–сопряженная пара собственных значений, соответствующая блоку, образуемому элементами j–го и $(j+1)$–го столбцов $a_{jj}^{(k)}, a_{jj+1}^{(k)}, a_{j+1j}^{(k)}, a_{j+1j+1}^{(k)}$, то несмотря на значительное изменение в ходе итераций самих этих элементов, собственные значения, соответствующие данному блоку и определяемые из решения квадратного уравнения $(a_{jj}^{(k)} - \lambda^{(k)})(a_{j+1j+1}^{(k)} - \lambda^{(k)}) = a_{jj+1}^{(k)} a_{j+1j}^{(k)}$, начиная с некоторого k, отличаются незначительно. В качестве критерия окончания итераций для таких блоков может быть использовано следующее условие $|\lambda^{(k)} - \lambda^{(k-1)}| \leq \varepsilon$.

Реализация

Тестирование

Входной файл

```
3
5 -5 -6
-1 -8 -5
2 7 -3
0.001
```

Выходной файл

```
solve1QR.log:
|Method QR| by Alexander Bales 80-308

3.03333 5.19283 5.11535
0.71339 -5.70348 10.9534
1.73876 -2.92221 -3.32986

5.79843 2.11649 -1.48932
0.575435 -4.01297 -8.26022
5.19363 5.04254 -7.78546

1.55173 8.27555 7.6834
-3.83406 -2.73465 5.423
4.96303 -2.94102 -4.81708

-3.14749 7.58946 -1.34069
0.526922 -5.21176 -10.7171
-3.99439 2.2088 2.35925

-2.20408 8.58656 -0.389965
-4.89394 -8.16322 -0.455668
-6.52049 -2.6219 4.3673

-2.49884 4.71682 -8.49962
-6.74093 -5.65205 -7.03694
-2.36348 0.383827 2.15088

-8.11358 7.18704 4.65476
-4.5264 -3.58066 -5.25446
-1.9268 1.33936 5.69424
```

```
-5.33289 8.49936 -4.29515
-4.61994 -5.4185 6.67382
-0.9889 0.274396 4.75139

-3.2153 6.87261 -2.15913
-5.16699 -7.19904 -8.79677
-0.609817 -0.0419114 4.41434

-6.11476 4.05064 7.46761
-8.4366 -5.04646 2.2089
-0.512829 0.115725 5.16122

-7.17506 7.22887 -7.01184
-5.29099 -3.51479 4.76206
-0.231564 0.14895 4.68985

-5.02926 8.31725 2.50335
-4.01722 -5.78925 -8.02337
-0.124846 0.0359831 4.81852

-3.28602 6.2672 2.90602
-6.2133 -7.6045 7.68895
-0.0937216 -0.00772327 4.89051

-6.52824 4.40073 -8.3292
-8.02614 -4.25755 -1.08913
-0.0627453 0.0237435 4.78578

-6.97305 7.72097 5.99533
-4.69775 -3.87494 -5.76463
-0.0294156 0.021379 4.84799

-4.59877 8.21127 -1.79164
-4.23437 -6.23585 8.1257
-0.016857 0.00435829 4.83461

-3.35687 5.57105 -4.20784
-6.85538 -7.4661 -7.21033
-0.0129455 -0.000149771 4.82297
```

```
-7.19273 4.99117 8.30146
-7.44142 -3.64487 -0.613455
-0.00811517 0.00422724 4.8376

-6.57876 8.02773 -5.34196
-4.40678 -4.25095 6.39708
-0.00379228 0.00279957 4.8297

-4.18394 7.98265 0.872999
-4.44798 -6.64698 -8.28887
-0.00229859 0.0004847 4.83091

-3.72875 4.87821 5.4361
-7.55493 -7.10406 6.3125
-0.00181818 0.000117299 4.83281

-7.5035 5.68994 -8.06995
-6.74255 -3.32729 2.08056
-0.00104359 0.000658023 4.83078

-6.16162 8.23673 4.61023
-4.19537 -4.67014 -6.94114
-0.000503125 0.000339114 4.83175

-6.16162 8.23673 4.61023
-4.19537 -4.67014 -6.94114
-0.000503125 0.000339114 4.83175

res:
Matrix A:
5 -5 -6
-1 -8 -5
2 7 -3

eigenvalue[1] = -5.42 + 5.83i
eigenvalue[2] = -5.42 - 5.83i
eigenvalue[3] = 4.83
```