

**Московский авиационный институт  
(национальный исследовательский университет)**

**Факультет информационных технологий и прикладной  
математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №1 по курсу «Численные методы»**

Студент: А. А. Литвина  
Преподаватель: И. Э. Иванов  
Группа: М8О-306Б  
Дата:  
Оценка:  
Подпись:

**Москва, 2020**

# Вариант №13

## 1 LU - разложение

### 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.

### 2 Описание

LU - разложение матрицы  $A$  представляет собой разложение матрицы  $A$  в произведение нижней и верхней треугольных матриц, т.е.  $A = LU$ , где  $L$  - нижняя треугольная матрица (матрица, у которой все элементы, находящиеся выше главной диагонали равны нулю, а элементы на главной диагонали равны 1),  $U$  - верхняя треугольная матрица (матрица, у которой все элементы, находящиеся ниже главной диагонали равны нулю, а элементы на главной диагонали произвольные).

LU - разложение эффективно используется при решении СЛАУ вида  $Ax = b$ . Если вместо  $A$  подставить  $LU$ , получим:  $LUx = b$ . Сделаем замену  $Ux = y$ , получим:  $Ly = b$ . Решим эту систему уравнений и найдем  $y$ . Далее подставим полученный  $y$  в уравнение  $Ux = y$  и найдем  $x$ .

Также с помощью LU - разложения легко можно найти определитель матрицы  $A$  - это результат перемножения элементов на главной диагонали матрицы  $U$ .

Обратная матрица ищется по тому же принципу, что и решение СЛАУ. В уравнение  $AA^{-1} = E$  подставим  $A = LU$  и получим  $LUA^{-1} = E$ , а дальше все так же, как и для решения СЛАУ.

### 3 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | using namespace std;
5 |
6 | void print(vector <vector<double>> v) {
7 |     for (int i=0; i<v.size(); i++) {
8 |         for (int j=0; j<v.size(); j++) {
9 |             cout << v[i][j] << "\t\t";
10 |        }
11 |        cout << endl;
12 |    }
13 | }
14 |
15 | double det(vector <vector<double>> v) {
16 |     double p=1;
17 |     for (int i=0; i<v.size(); i++) {
18 |         p*=v[i][i];
19 |     }
20 | }
21 |
22 | int main() {
23 |     int N;
24 |     cin >> N;
25 |     vector <vector <double>> A (N, vector <double>(N,0));
26 |     vector <vector <double>> A1 (N, vector <double>(N,0));
27 |     vector <vector <double>> Y1 (N, vector <double>(N,0));
28 |     vector <vector <double>> L (N, vector <double>(N,0));
29 |     vector <vector <double>> U (N, vector <double>(N,0));
30 |     vector <vector <double>> LU (N, vector <double>(N,0));
31 |     vector <double> b (N);
32 |     vector <double> x (N);
33 |     vector <double> y (N);
34 |     for (int i=0; i<N; i++) {
35 |         for (int j=0; j<N; j++) {
36 |             cin >> A[i][j];
37 |         }
38 |         cin >> b[i];
39 |     }
40 |
41 |     U=A;
42 |
43 |     for (int k=0; k<N-1; k++) {
44 |
45 |         for (int i=k; i<N; i++) {
46 |             for (int j=i; j<N; j++) {
47 |                 L[j][i]=U[j][i]/U[i][i];
```

```

48     }
49 }
50
51 for (int i=k+1; i<N; i++) {
52     for (int j=k; j<N; j++) {
53         U[i][j]=U[i][j]-L[i][k]*U[k][j];
54     }
55 }
56 }
57
58
59 for (int i=0; i<N; i++) {
60     for (int j=0; j<N; j++) {
61         for (int k=0; k<N; k++) {
62             LU[i][j]+=L[i][k]*U[k][j];
63         }
64     }
65 }
66
67
68 for (int i=0; i<N; i++) {
69     double S=0;
70     for (int j=0; j<i; j++) {
71         S+=L[i][j]*y[j];
72     }
73     y[i]=b[i]-S;
74 }
75
76
77 for (int i=N-1; i>=0; i--) {
78     double S=0;
79     for (int j=N-1; j>i; j--) {
80         S+=U[i][j]*x[j];
81     }
82     x[i]=(y[i]-S)/U[i][i];
83 }
84
85
86 for (int k=0; k<N; k++) {
87     for (int i=0; i<N; i++) {
88         double S=0;
89         for (int j=0; j<i; j++) {
90             S+=L[i][j]*Y1[j][k];
91         }
92         if (i==k)
93             Y1[i][k]=1-S;
94         else
95             Y1[i][k]=-S;
96     }

```

```

97     }
98
99     for (int k=0; k<N; k++) {
100         for (int i=N-1; i>=0; i--) {
101             double S=0;
102             for (int j=N-1; j>i; j--) {
103                 S+=U[i][j]*A1[j][k];
104             }
105             A1[i][k]=(Y1[i][k]-S)/U[i][i];
106         }
107     }
108
109     cout << "\n\tMatrix L\n";
110     cout << "-----\n";
111     print(L);
112     cout << "\n\n\tMatrix U\n";
113     cout << "-----\n";
114     print(U);
115     cout << "\n\n\tMatrix L*U\n";
116     cout << "-----\n";
117     print(LU);
118     cout << "\n\n\tMatrix A1\n";
119     cout << "-----\n";
120     print(A1);
121     cout << "\n\nsolution=( ";
122     for (int i=0; i<N; i++) {
123         cout << x[i] << " ";
124     }
125     cout << ")\n\ndet=" << det(U) << endl;
126 }

```

## 4 Консоль

Входные данные:

```
4
-6 -5 -3 -8 101
5 -1 -5 -4 51
-6 0 5 5 -53
-7 -2 8 5 -63
```

Выходные данные:

Matrix L

```
-----
1 0 0 0
-0.833333 1 0 0
1 -0.967742 1 0
1.16667 -0.741935 8 1
```

Matrix U

```
-----
-6 -5 -3 -8
0 -5.16667 -7.5 -10.6667
0 0 0.741935 2.67742
0 -4.44089e-16 0 -15
```

Matrix L\*U

```
-----
-6 -5 -3 -8
5 -1 -5 -4
-6 0 5 5
-7 -2 8 5
```

Matrix A1

```
-----
-0.0724638 0.0724638 -0.202899 0.144928
0.0057971 -0.605797 -0.263768 -0.211594
-0.0202899 -0.37971 -0.576812 0.24058
-0.0666667 0.466667 0.533333 -0.0666667
```

solution=( -2 -3 -6 -7 )

det=-345

## 2 Метод прогонки

### 1 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.

### 2 Описание

Метод прогонки - один из наиболее эффективных и экономичных методов решения СЛАУ. Однако он весьма специфичен, потому что применяется только для трехдиагональных матриц, т.е. матриц с тремя диагоналями: главной, выше нее и ниже нее. Остальные элементы матрицы равны нулю.

Метод заключается в подборе прогоночных коэффициентов  $P_i, Q_i, i = 1..n$ , таких, что решение будет выглядеть в виде:  $x_i = P_i x_{i+1} + Q_i, i = 1..n$ .



### 3 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | using namespace std;
5 |
6 | int main() {
7 |     int N, i;
8 |     double y;
9 |     cin >> N;
10 |    vector <vector <double>> A (N, vector <double>(N,0));
11 |    vector <double> b (N);
12 |    vector <double> x (N);
13 |    vector <double> al (N);
14 |    vector <double> bet (N);
15 |
16 |    for (i=0; i<N; i++) {
17 |        for (int j=0; j<N; j++) {
18 |            cin >> A[i][j];
19 |        }
20 |        cin >> b[i];
21 |    }
22 |
23 |    y=A[0][0];
24 |    al[0]=-A[0][1]/y;
25 |    bet[0]=b[0]/y;
26 |    for (i=1; i<N-1; i++) {
27 |        y=A[i][i]+A[i][i-1]*al[i-1];
28 |        al[i]=-A[i][i+1]/y;
29 |        bet[i]=(b[i]-A[i][i-1]*bet[i-1])/y;
30 |    }
31 |
32 |    y=A[i][i]+A[i][i-1]*al[i-1];
33 |    bet[i]=(b[i]-A[i][i-1]*bet[i-1])/y;
34 |
35 |    x[N-1]=bet[N-1];
36 |    for (i=N-2; i>=0; i--) {
37 |        x[i]=al[i]*x[i+1]+bet[i];
38 |    }
39 |
40 |    cout << "\nsolution=( ";
41 |    for (i=0; i<N; i++) {
42 |        cout << x[i] << " ";
43 |    }
44 |    cout << ")\n";
45 |
46 | }
```

## 4 Консоль

Входные данные:

5

14 9 0 0 0 125

-8 14 6 0 0 -56

0 -5 -17 8 0 144

0 0 1 5 -2 36

0 0 0 -4 -10 70

Выходные данные:

solution=( 7 3 -7 5 -9 )

## 3 Метод простых итераций

### 1 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.

### 2 Описание

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

Сначала приводим СЛАУ к виду  $x = \beta + \alpha x$ . Задаем начальное значение  $x^{(0)} = \beta$ . Метод простых итераций на  $k$ -ой итерации имеет вид:  $x^{(k)} = \beta + \alpha x^{(k-1)}$ ,  $k = 1..n$ . Итерационный процесс останавливается, когда значение  $\frac{\|\alpha\|}{1-\|\alpha\|} \|x^{(k)} - x^{(k-1)}\|$  станет меньше или равно заданной точности вычислений.

Метод Зейделя во многом схож с методом простых итераций, но является некоторым его ускорением за счет того, что при вычислении компонента  $x_i^{k+1}$  вектора неизвестных на  $(k+1)$ -ой итерации используются  $x_1^{k+1}$ ,  $x_2^{k+1}$ , ...,  $x_{i-1}^{k+1}$ , уже вычисленные на  $(k+1)$ -ой итерации. Метод Зейделя на  $k+1$ -ой итерации имеет вид:  $x^{k+1} = \beta + Bx^{k+1} + Cx^k$ , где  $B$  - нижняя треугольная матрица с диагональными элементами, равными нулю, а  $C$  - верхняя треугольная матрица с диагональными элементами, отличными от нуля,  $\alpha = B + C$ . В этом случае итерационный процесс останавливается, когда заданная точность вычислений станет больше или равной значению  $\frac{\|C\|}{1-\|\alpha\|} \|x^{(k)} - x^{(k-1)}\|$ .

Как можно увидеть ниже из результата работы программы, метод простых итераций находит решение за 12 итераций, в то время как метод Зейделя справляется с поставленной задачей всего за 9 итераций, что является довольно существенным выигрышем.

## 3 Исходный код

### 3.1 Метод простых итераций

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5  using namespace std;
6
7  const double eps=0.0001;
8
9  int main() {
10     int N, it;
11     double max, S, maxA, max1;
12     cin >> N;
13     vector <vector <double>> A (N, vector <double>(N,0));
14     vector <vector <double>> A1 (N, vector <double>(N,0));
15     vector <double> b (N);
16     vector <double> x (N,0);
17     vector <double> y (N,0);
18     vector <double> e (N,0);
19
20     for (int i=0; i<N; i++) {
21         for (int j=0; j<N; j++) {
22             cin >> A[i][j];
23         }
24         cin >> b[i];
25     }
26
27     A1=A;
28
29     for (int i=0; i<N; i++) {
30         if (A1[i][i]!=0) {
31             b[i]/=A1[i][i];
32             for (int j=0; j<N; j++) {
33                 if (i!=j)
34                     A1[i][j]/=A1[i][i];
35             }
36             A1[i][i]=0;
37         }
38     }
39
40     maxA=0;
41     for (int i=0; i<N; i++) {
42         S=0;
43         for (int j=0; j<N; j++) {
44             S+=abs(A1[i][j]);
45         }
```

```

46
47     if (S>maxA) {
48         maxA=S;
49     }
50 }
51
52 it=1;
53
54 cout << endl;
55
56 while ((max>eps)|| (it==1)) {
57     cout << "-----\n";
58     cout << it << "\n-----\nx=( ";
59     for (int j=0; j<N; j++) {
60         x[j]=b[j];
61         for (int k=0; k<N; k++) {
62             x[j]-=A1[j][k]*y[k];
63         }
64     }
65
66     max=0;
67     for (int j=0; j<N; j++) {
68         e[j]=abs(x[j]-y[j]);
69         if (e[j]>max)
70             max=e[j];
71         max1=max*maxA/(1-maxA);
72         y[j]=x[j];
73         cout << x[j] << " ";
74     }
75     cout << ")\n" << "eps=" << max1 << endl;
76     it++;
77 }
78
79 cout << endl;
80 }

```

### 3.2 Метод Зейделя

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5  using namespace std;
6
7  const double eps=0.0001;
8
9  int main() {
10     int N, it;
11     double max, maxA, S, maxC, max1;
12     cin >> N;

```

```

13 vector <vector <double>> A (N, vector <double>(N,0));
14 vector <vector <double>> A1 (N, vector <double>(N,0));
15 vector <vector <double>> C (N, vector <double>(N,0));
16 vector <double> b (N);
17 vector <double> x (N,0);
18 vector <double> y (N,0);
19 vector <double> e (N,0);
20
21 for (int i=0; i<N; i++) {
22     for (int j=0; j<N; j++) {
23         cin >> A[i][j];
24     }
25     cin >> b[i];
26 }
27
28 A1=A;
29
30 for (int i=0; i<N; i++) {
31     if (A1[i][i]!=0) {
32         b[i]/=A1[i][i];
33         for (int j=0; j<N; j++) {
34             if (i!=j)
35                 A1[i][j]/=A1[i][i];
36         }
37         A1[i][i]=0;
38     }
39 }
40
41 maxA=0;
42 for (int i=0; i<N; i++) {
43     S=0;
44     for (int j=0; j<N; j++) {
45         S+=abs(A1[i][j]);
46     }
47
48     if (S>maxA) {
49         maxA=S;
50     }
51 }
52
53 for (int i=0; i<N; i++) {
54     for (int j=i; j<N; j++) {
55         C[i][j]=A1[i][j];
56     }
57 }
58
59 maxC=0;
60 for (int i=0; i<N; i++) {
61     S=0;

```

```

62     for (int j=0; j<N; j++) {
63         S+=abs(C[i][j]);
64     }
65
66     if (S>maxC) {
67         maxC=S;
68     }
69 }
70
71 it=1;
72
73 cout << endl;
74
75 while ((max>eps)|| (it==1)) {
76     cout << "-----\n";
77     cout << it << "\n-----\nx=( ";
78     for (int j=0; j<N; j++) {
79         x[j]=b[j];
80         if (it!=1) {
81             for (int k=0; k<N; k++) {
82                 x[j]-=A1[j][k]*x[k];
83             }
84         }
85     }
86
87     max=0;
88     for (int j=0; j<N; j++) {
89         e[j]=abs(x[j]-y[j]);
90         if (e[j]>max)
91             max=e[j];
92         max1=max*maxC/(1-maxA);
93         y[j]=x[j];
94         cout << x[j] << " ";
95     }
96     cout << ")\n" << "eps=" << max1 << endl;
97     it++;
98 }
99
100 cout << endl;
101 }

```

## 4 Консоль

Входные данные:

```
5
14 9 0 0 0 125
-8 14 6 0 0 -56
0 -5 -17 8 0 144
0 0 1 5 -2 36
0 0 0 -4 -10 70
```

### 4.1 Метод простых итераций

Выходные данные:

```
-----
1
-----
x=( -7.91667 1.33333 6.45833 1.13333 )
eps=30.0833
-----
2
-----
x=( -6.64028 2.28519 6.63403 -0.788889 )
eps=7.30444
-----
3
-----
x=( -6.013 2.24784 6.91769 -1.10796 )
eps=2.38365
-----
4
-----
x=( -5.92344 2.04662 6.96982 -1.08903 )
eps=0.764646
-----
5
-----
x=( -5.97659 2.00097 7.01022 -1.00952 )
eps=0.302132
-----
6
-----
```



```

x=( -5.99643 1.99204 7.00036 -1.00019 )
eps=0.0753658
-----
7
-----
x=( -6.00223 1.99877 7.00195 -0.996627 )
eps=0.0255674
-----
8
-----
x=( -6.0006 1.99956 6.99937 -0.999916 )
eps=0.0124992
-----
9
-----
x=( -6.00025 2.00032 7.00017 -0.999781 )
eps=0.00303601
-----
10
-----
x=( -5.99991 2 6.99986 -1.00017 )
eps=0.00146919
-----
11
-----
x=( -6 2.00004 7.00005 -0.999973 )
eps=0.000752839
-----
12
-----
x=( -5.99998 1.99998 6.99998 -1.00002 )
eps=0.000287553

```

## 4.2 Метод Зейделя

Выходные данные:

```

-----
1
-----
x=( -7.91667 1.33333 6.45833 1.13333 )
eps=23.75

```

```

-----
2
-----
x=( -6.64028 1.85972 6.32095 -0.896034 )
eps=6.0881
-----
3
-----
x=( -6.17142 2.18494 6.9285 -1.07587 )
eps=1.82265
-----
4
-----
x=( -5.94533 2.01453 7.01738 -1.00448 )
eps=0.678255
-----
5
-----
x=( -5.99212 1.99451 7.0023 -0.997584 )
eps=0.140361
-----
6
-----
x=( -6.00162 1.99949 6.99944 -0.999831 )
eps=0.0285074
-----
7
-----
x=( -6.00027 2.00018 6.99992 -1.00008 )
eps=0.00405952
-----
8
-----
x=( -5.99995 2.00002 7.00002 -1.00001 )
eps=0.00095751
-----
9
-----
x=( -5.99999 1.99999 7 -0.999998 )
eps=0.000123933

```

## 4 Метод вращений

### 1 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.

### 2 Описание

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.

Сначала выбираем максимальный по модулю элемент в наддиагональной части матрицы  $A$ . Затем находим матрицу вращения  $H$ , соответствующую этому элементу. Далее строится матрица  $A^{(k+1)} = H^{(k)T} A^{(k)} H^{(k)}$ . В качестве критерия окончания итерационного процесса используется условие малости суммы квадратов наддиагональных элементов:

$$t(A^{(k+1)}) = \left( \sum_{l,m:l < m} (a_{lm}^{(k+1)})^2 \right)^{1/2}.$$

Если  $t(A^{(k+1)}) > \varepsilon$ , то итерационный процесс продолжается. Когда итерационный процесс останавливается, в качестве искоемых собственных значений принимаются  $\lambda_1 = a_{11}^{(k+1)}$ ,  $\lambda_2 = a_{22}^{(k+1)}$ ,  $\lambda_3 = a_{33}^{(k+1)}$ . Столбцами собственных векторов матрицы  $A$  будут столбцы матрицы  $H = H^{(0)} H^{(1)} \dots H^{(k)}$ .

Проверить правильность работы программы можно подсчитав след исходной матрицы  $A$  и матрицы  $A$ , полученной в результате работы программы, и убедившись, что они совпадают.

### 3 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 | using namespace std;
6 |
7 | const double eps=0.0001;
8 |
9 | int main() {
10 |     int N, maxi, maxj;
11 |     double max, phi, sl, t;
12 |     cin >> N;
13 |     vector <vector <double>> A (N, vector <double>(N,0));
14 |     vector <vector <double>> H (N, vector <double>(N,0));
15 |     vector <vector <double>> Hp (N, vector <double>(N,0));
16 |     vector <vector <double>> Htemp (N, vector <double>(N,0));
17 |     vector <vector <double>> Ht (N, vector <double>(N,0));
18 |     vector <vector <double>> HtA (N, vector <double>(N,0));
19 |     vector <vector <double>> HtAH (N, vector <double>(N,0));
20 |
21 |     for (int i=0; i<N; i++) {
22 |         for (int j=0; j<N; j++) {
23 |             cin >> A[i][j];
24 |         }
25 |     }
26 |
27 |     max=abs(A[0][N-1]);
28 |     maxi=0;
29 |     maxj=N-1;
30 |     t=0;
31 |
32 |     for (int i=0; i<N; i++) {
33 |         for (int j=i+1; j<N; j++) {
34 |             if (abs(A[i][j])>max) {
35 |                 max=abs(A[i][j]);
36 |                 maxi=i;
37 |                 maxj=j;
38 |             }
39 |             t+=pow(A[i][j],2);
40 |         }
41 |     }
42 |     t=pow(t,0.5);
43 |
44 |     for (int i=0; i<N; i++) {
45 |         Hp[i][i]=1;
46 |     }
47 | }
```

```

48 while (max>eps) {
49     phi=atan(2*A[maxi][maxj]/(A[maxi][maxi]-A[maxj][maxj]))/2;
50     for (int i=0; i<N; i++)
51         H[i][i]=1;
52
53     H[maxi][maxi]=cos(phi);
54     H[maxi][maxj]=-sin(phi);
55     H[maxj][maxi]=sin(phi);
56     H[maxj][maxj]=cos(phi);
57
58     for (int i=0; i<N; i++) {
59         for (int j=0; j<N; j++) {
60             for (int k=0; k<N; k++) {
61                 Htemp[i][j]+=Hp[i][k]*H[k][j];
62             }
63         }
64     }
65
66     Hp=Htemp;
67
68     Ht=H;
69
70     for (int i=0; i<N; i++) {
71         for (int j=i+1; j<N; j++) {
72             double tmp=Ht[i][j];
73             Ht[i][j]=Ht[j][i];
74             Ht[j][i]=tmp;
75         }
76     }
77
78     for (int i=0; i<N; i++) {
79         for (int j=0; j<N; j++) {
80             for (int k=0; k<N; k++) {
81                 HtA[i][j]+=Ht[i][k]*A[k][j];
82             }
83         }
84     }
85
86     for (int i=0; i<N; i++) {
87         for (int j=0; j<N; j++) {
88             for (int k=0; k<N; k++) {
89                 HtAH[i][j]+=HtA[i][k]*H[k][j];
90             }
91         }
92     }
93
94     A=HtAH;
95
96     max=abs(A[0][N-1]);

```

```

97     maxi=0;
98     maxj=N-1;
99     t=0;
100
101     for (int i=0; i<N; i++) {
102         for (int j=i+1; j<N; j++) {
103             if (abs(A[i][j])>max) {
104                 max=abs(A[i][j]);
105                 maxi=i;
106                 maxj=j;
107             }
108             t+=pow(A[i][j],2);
109         }
110     }
111     t=pow(t,0.5);
112
113     H.assign(N, vector<double>(N,0));
114     Ht.assign(N, vector<double>(N,0));
115     HtA.assign(N, vector<double>(N,0));
116     HtAH.assign(N, vector<double>(N,0));
117     Htemp.assign(N, vector<double>(N,0));
118 }
119
120 cout << "\n Eigenvectors:\n\n";
121 for (int i=1; i<=N; i++) {
122     cout << "X" << i << "\t\t";
123 }
124 cout << "\n-----\n";
125
126 for (int i=0; i<N; i++) {
127     for (int j=0; j<N; j++) {
128         cout << Hp[i][j] << " \t";
129     }
130     cout << endl;
131 }
132 cout << "-----\n";
133
134 s1=0;
135 cout << "\n Eigenvalues:\n";
136 for (int i=0; i<N; i++) {
137     s1+=A[i][i];
138     cout << A[i][i] << " ";
139 }
140
141 cout << "\n\n Track " << s1 << "\n\n";
142 }

```

## 4 Консоль

Входные данные:

```
3
8 0 -2
0 5 4
-2 4 -6
```

Выходные данные:

Собственные векторы:

X1	X2	X3
-----		
0.95324	0.27661	0.121743
-0.229944	0.925237	-0.301764
-0.196112	0.259659	0.945578
-----		

Собственные значения:

```
8.41146  6.12256  -7.53402
```

След равен 7

## 5 QR - разложение

### 1 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.

### 2 Описание

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. Матрица Хаусхолдера имеет вид:

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

Произведем разложение  $A = QR$   $n - 1$  раз, где  $Q = H_1 H_2 \dots H_{n-1}$ . Далее произведем перемножение матриц в обратном порядке:  $A^{(k+1)} = R^{(k)} Q^{(k)}$ . Итерационный процесс продолжается, пока сумма квадратов поддиагональных элементов первого столбца достаточно велика, т.е.

$$\left( \sum_{l=m+1}^n (a_{lm}^{(k)})^2 \right)^{1/2} > \varepsilon.$$

Когда критерий окончания нарушается, диагональный элемент  $a_{11}^{(k)}$  может быть принят в качестве собственного значения, а элементы  $a_{22}^{(k)}$ ,  $a_{23}^{(k)}$ ,  $a_{32}^{(k)}$ ,  $a_{33}^{(k)}$  составляют комплексно-сопряженные пары собственных значений, поэтому остальные с.з. определяются из решения квадратного уравнения  $(a_{22}^{(k)} - \lambda^{(k)})(a_{33}^{(k)} - \lambda^{(k)}) = a_{23}^{(k)} a_{32}^{(k)}$ .



### 3 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 | using namespace std;
6 |
7 | const double eps=0.0001;
8 |
9 | int sign(double a) {
10 |     if (a>0) return 1;
11 |     if (a<0) return -1;
12 |     return 0;
13 | }
14 |
15 | void print(vector <vector<double>> v) {
16 |     for (int i=0; i<v.size(); i++) {
17 |         for (int j=0; j<v.size(); j++) {
18 |             cout << v[i][j] << "\t\t";
19 |         }
20 |         cout << endl;
21 |     }
22 | }
23 |
24 | int main() {
25 |     int N;
26 |     cin >> N;
27 |     vector <vector <double>> A (N, vector <double>(N,0));
28 |     vector <vector <double>> newA (N, vector <double>(N,0));
29 |     vector <vector <double>> H (N, vector <double>(N,0));
30 |     vector <vector <double>> Q (N, vector <double>(N,0));
31 |     vector <vector <double>> newQ (N, vector <double>(N,0));
32 |     vector <vector <double>> R (N, vector <double>(N,0));
33 |     vector <vector <double>> vvt (N, vector <double>(N,0));
34 |     vector <double> v (N,0);
35 |
36 |     for (int i=0; i<N; i++) {
37 |         for (int j=0; j<N; j++) {
38 |             cin >> A[i][j];
39 |         }
40 |     }
41 |
42 |     for (int i=0; i<N; i++) {
43 |         Q[i][i]=1;
44 |     }
45 |
46 |     double max=0;
47 |     for (int k=1; k<N; k++) {
```

```

48     if (abs(A[k][0])>eps) {
49         max+=pow(A[k][0],2);
50     }
51 }
52 max=pow(max,0.5);
53
54 while (max>eps) {
55     for (int i=0; i<N-1; i++) {
56         int j;
57         for (j=0; j<i; j++) {
58             v[j]=0;
59         }
60         double sum=0;
61         for (int k=i; k<N; k++) {
62             sum+=pow(A[k][i],2);
63         }
64
65         sum=pow(sum,0.5);
66         v[j]=A[j][i]+sign(A[j][i])*sum;
67         for (j=i+1; j<N; j++) {
68             v[j]=A[j][i];
69         }
70
71         for (int k=0; k<N; k++) {
72             for (int l=0; l<N; l++) {
73                 vvt[k][l]=v[k]*v[l];
74             }
75         }
76
77         double vtv=0;
78         for (int k=0; k<N; k++) {
79             vtv+=v[k]*v[k];
80         }
81
82         for (int k=0; k<N; k++) {
83             for (int l=0; l<N; l++) {
84                 if (k==l)
85                     H[k][l]=1-2*vvt[k][l]/vtv;
86                 else
87                     H[k][l]=-2*vvt[k][l]/vtv;
88             }
89         }
90
91         for (int k=0; k<N; k++) {
92             for (int l=0; l<N; l++) {
93                 for (int m=0; m<N; m++) {
94                     newQ[k][l]+=Q[k][m]*H[m][l];
95                 }
96             }

```

```

97     }
98
99     for (int k=0; k<N; k++) {
100         for (int l=0; l<N; l++) {
101             for (int m=0; m<N; m++) {
102                 newA[k][l] += H[k][m] * A[m][l];
103             }
104         }
105     }
106
107     A=newA;
108     Q=newQ;
109     newA.assign(N, vector<double>(N,0));
110     newQ.assign(N, vector<double>(N,0));
111 }
112
113 R=A;
114 A.assign(N, vector<double>(N,0));
115 for (int k=0; k<N; k++) {
116     for (int l=0; l<N; l++) {
117         for (int m=0; m<N; m++) {
118             A[k][l] += R[k][m] * Q[m][l];
119         }
120     }
121 }
122
123 max=0;
124 for (int k=1; k<N; k++) {
125     if (abs(A[k][0])>eps) {
126         max+=pow(A[k][0],2);
127     }
128 }
129 max=pow(max,0.5);
130
131 Q.assign(N, vector<double>(N,0));
132 for (int i=0; i<N; i++) {
133     Q[i][i]=1;
134 }
135 }
136
137 cout << "lambda1 = " << A[0][0] << endl;
138 double d;
139 d=pow(A[1][1]+A[2][2],2)-4*(A[1][1]*A[2][2]-A[1][2]*A[2][1]);
140 if (d>=0) {
141     d=pow(d,0.5);
142     cout << "lambda2 = " << (A[1][1]+A[2][2]+d)/2 << endl;
143     cout << "lambda3 = " << (A[1][1]+A[2][2]-d)/2 << endl;
144 }
145 else if (d<0) {

```

```

146 || d=pow(-d,0.5);
147 || cout << "lambda2 = " << (A[1][1]+A[2][2])/2 << " + " << d/2 << "i\n";
148 || cout << "lambda3 = " << (A[1][1]+A[2][2])/2 << " - " << d/2 << "i\n";
149 || }
150 || }

```

## 4 Консоль

Входные данные:

```
3
-1 2 9
9 3 4
8 -4 -6
```

Выходные данные:

```
lambda1 = -13.0064
lambda2 = 4.50319 + 2.80382i
lambda3 = 4.50319 -2.80382i
```

## Выводы

В этой лабораторной работе я познакомилась с численными методами решения СЛАУ и задач на собственные значения и собственные векторы матриц. Я узнала такие методы, как метод Гауса и LU-разложения, метод прогонки, метод простых итераций и метод Зейделя, метод вращений и метод QR-разложения.

Все эти методы имеют свои особенности, свои достоинства и недостатки. Поэтому подбирать метод решения следует исходя из специфики задачи.

Данная лабораторная работа была полезна для меня, потому что, реализовав все эти методы в виде программ, я стала лучше понимать, как они работают.