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

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

using namespace std;

void print(vector <vector<double>> v) {
  for (int i=0; i<v.size(); i++) {
    for (int j=0; j<v.size(); j++) {
      cout << v[i][j] << "\t\t";
    }
    cout << endl;
  }
}

double det(vector <vector<double>> v) {
  double p=1;
  for (int i=0; i<v.size(); i++) {
    p*=v[i][i];
  }
}

int main() {
  int N;
  cin >> N;
  vector <vector <double>> A (N, vector <double>(N,0));
  vector <vector <double>> A1 (N, vector <double>(N,0));
  vector <vector <double>> Y1 (N, vector <double>(N,0));
  vector <vector <double>> L (N, vector <double>(N,0));
  vector <vector <double>> U (N, vector <double>(N,0));
  vector <vector <double>> LU (N, vector <double>(N,0));
  vector <double> b (N);
  vector <double> x (N);
  vector <double> y (N);
  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      cin >> A[i][j];
    }
    cin >> b[i];
  }

  U=A;

  for (int k=0; k<N-1; k++) {

    for (int i=k; i<N; i++) {
      for (int j=i; j<N; j++) {
        L[j][i]=U[j][i]/U[i][i];
```

```
    }
  }

  for (int i=k+1; i<N; i++) {
    for (int j=k; j<N; j++) {
      U[i][j]=U[i][j]-L[i][k]*U[k][j];
    }
  }
}


for (int i=0; i<N; i++) {
  for (int j=0; j<N; j++) {
    for (int k=0; k<N; k++) {
      LU[i][j]+=L[i][k]*U[k][j];
    }
  }
}


for (int i=0; i<N; i++) {
  double S=0;
  for (int j=0; j<i; j++) {
      S+=L[i][j]*y[j];
  }
  y[i]=b[i]-S;
}


for (int i=N-1; i>=0; i--) {
  double S=0;
  for (int j=N-1; j>i; j--) {
      S+=U[i][j]*x[j];
  }
  x[i]=(y[i]-S)/U[i][i];
}


for (int k=0; k<N; k++) {
  for (int i=0; i<N; i++) {
    double S=0;
    for (int j=0; j<i; j++) {
      S+=L[i][j]*Y1[j][k];
    }
    if (i==k)
      Y1[i][k]=1-S;
    else
      Y1[i][k]=-S;
  }
```

```
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: $x_i = P_i x_{i+1} + Q_i$, $i = 1..n$. Прогоночные коэффициенты вычисляются по формулам:

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

$$P_i = \frac{-c_i}{b_i + a_i P_{i-1}}, Q_i = \frac{d_i - a_i Q_{i-1}}{b_i + a_i P_{i-1}}; i = 2, .., n-1$$

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

Для устойчивости метода прогонки достаточно выполнение следующих условий:

$$a_i \neq 0, c_i \neq 0; i = 2, .., n-1$$

$$|b_i| \geq |a_i| + |c_i|; i = 1, .., n$$

причем строгое неравенство имеет место хотя бы при одном i. Здесь устойчивость понимается в смысле ненакопления погрешности решения в ходе вычислительного процесса при малых погрешностях входных данных (правых частей и элементов матрицы СЛАУ).

3 Исходный код

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

using namespace std;

int main() {
  int N, i;
  double y;
  cin >> N;
  vector <vector <double>> A (N, vector <double>(N,0));
  vector <double> b (N);
  vector <double> x (N);
  vector <double> al (N);
  vector <double> bet (N);

  for (i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      cin >> A[i][j];
    }
    cin >> b[i];
  }

  y=A[0][0];
  al[0]=-A[0][1]/y;
  bet[0]=b[0]/y;
  for (i=1; i<N-1; i++) {
    y=A[i][i]+A[i][i-1]*al[i-1];
    al[i]=-A[i][i+1]/y;
    bet[i]=(b[i]-A[i][i-1]*bet[i-1])/y;
  }

  y=A[i][i]+A[i][i-1]*al[i-1];
  bet[i]=(b[i]-A[i][i-1]*bet[i-1])/y;

  x[N-1]=bet[N-1];
  for (i=N-2; i>=0; i--) {
    x[i]=al[i]*x[i+1]+bet[i];
  }

  cout << "\nsolution=( ";
  for (i=0; i<N; i++) {
    cout << x[i] << " ";
  }
  cout << ")\n";

}
```

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: $x^{(k)} = \beta + \alpha x^{(k-1)}$, $k = 1..n$. Итерационнный процесс отстанавливается, 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 значения $\frac{\|C\|}{1-\|\alpha\|}\|x^{(k)} - x^{(k-1)}\|$.

Оба метода сходятся к единственному решению СЛАУ при любом начальном приближении $x^{(0)}$, если какая-либо норма матрицы α эквивалентной системы меньше единицы: $\|\alpha\| < 1$.

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

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

using namespace std;

const double eps=0.0001;

int main() {
  int N, it;
  double max, S, maxA, max1;
  cin >> N;
  vector <vector <double>> A (N, vector <double>(N,0));
  vector <vector <double>> A1 (N, vector <double>(N,0));
  vector <double> b (N);
  vector <double> x (N,0);
  vector <double> y (N,0);
  vector <double> e (N,0);

  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      cin >> A[i][j];
    }
    cin >> b[i];
  }

  A1=A;

  for (int i=0; i<N; i++) {
    if (A1[i][i]!=0) {
      b[i]/=A1[i][i];
      for (int j=0; j<N; j++) {
        if (i!=j)
          A1[i][j]/=A1[i][i];
      }
      A1[i][i]=0;
    }
  }

  maxA=0;
  for (int i=0; i<N; i++) {
    S=0;
    for (int j=0; j<N; j++) {
      S+=abs(A1[i][j]);
    }
```

```
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;
```

```
    for (int j=0; j<N; j++) {
      S+=abs(C[i][j]);
    }

    if (S>maxC) {
      maxC=S;
    }
  }

  it=1;

  cout << endl;

  while ((max>eps)||(it==1)) {
    cout << "-----------------------------------\n";
    cout << it << "\n-----------------------------------\nx=( ";
    for (int j=0; j<N; j++) {
      x[j]=b[j];
      if (it!=1) {
        for (int k=0; k<N; k++) {
          x[j]-=A1[j][k]*x[k];
        }
      }
    }

    max=0;
    for (int j=0; j<N; j++) {
      e[j]=abs(x[j]-y[j]);
      if (e[j]>max)
        max=e[j];
      max1=max*maxC/(1-maxA);
      y[j]=x[j];
      cout << x[j] << " ";
    }
    cout << ")\n" << "eps=" << max1 << endl;
    it++;
  }

  cout << endl;
}
```

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_{nxn} $(A = A^T)$ и решает полную проблему собственных значений и собственных векторов таких матриц.

Сначала выбираем максимальный по модулю элемент $a_{ij}^{(k)}$ в наддиагональной части матрицы A. Затем находим матрицу вращения H, соответствующую этому элементу:

$$
\begin{array}{c}
\begin{matrix} & & & i & & & j & & \end{matrix} \\
\begin{pmatrix}
1 & & & \vdots & & & \vdots & & \\
& \ddots & & \vdots & & & \vdots & 0 & \\
& & 1 & \vdots & & & \vdots & & \\
\cdots\cdots & & & \cos\varphi^{(k)} & \cdots\cdots & & -\sin\varphi^{(k)} & \cdots\cdots & \\
& & & \vdots & 1 & & \vdots & & \\
& & & \vdots & & \ddots & \vdots & & \\
& & & \vdots & & & 1 & \vdots & \\
\cdots\cdots & & & \sin\varphi^{(k)} & \cdots\cdots & & \cos\varphi^{(k)} & \cdots\cdots & \\
& & & \vdots & & & \vdots & 1 & \\
& 0 & & \vdots & & & \vdots & & \ddots \\
& & & \vdots & & & \vdots & & & 1
\end{pmatrix}
\begin{matrix} \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \end{matrix}
\end{array}
$$

В матрице вращения на пересечении i-ой строки и j-го столбца находится элемент $h_{ij}^{(k)} = -\sin\varphi^{(k)}$, на пересечении j-ой строки и i-го стоблца $h_{ji}^{(k)} = \sin\varphi^{(k)}$, диагональные элементы h_{ii} и h_{jj} равны $\cos\varphi^{(k)}$, где $\varphi^{(k)}$ - угол вращения. Остальные диагональные элементы $h_{mm}^{(k)} = 1$, $m = 1, .., n$, $m \neq i$, $m \neq j$, остальные элементы в матрице

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

$$\varphi^{(k)} = \frac{1}{2} \operatorname{arctg} \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}}.$$

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:

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

Если $t(A^{(k+1)}) > \varepsilon$, то итерационный процесс продолжается.Когда итерационный процесс останавливается, в качестве искомых 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 $H = H^{(0)} H^{(1)} ... H^{(k)}$.

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

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

using namespace std;

const double eps=0.0001;

int main() {
  int N, maxi, maxj;
  double max, phi, sl, t;
  cin >> N;
  vector <vector <double>> A (N, vector <double>(N,0));
  vector <vector <double>> H (N, vector <double>(N,0));
  vector <vector <double>> Hp (N, vector <double>(N,0));
  vector <vector <double>> Htemp (N, vector <double>(N,0));
  vector <vector <double>> Ht (N, vector <double>(N,0));
  vector <vector <double>> HtA (N, vector <double>(N,0));
  vector <vector <double>> HtAH (N, vector <double>(N,0));

  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      cin >> A[i][j];
    }
  }

  max=abs(A[0][N-1]);
  maxi=0;
  maxj=N-1;
  t=0;

  for (int i=0; i<N; i++) {
    for (int j=i+1; j<N; j++) {
      if (abs(A[i][j])>max) {
        max=abs(A[i][j]);
        maxi=i;
        maxj=j;
      }
      t+=pow(A[i][j],2);
    }
  }
  t=pow(t,0.5);

  for (int i=0; i<N; i++) {
    Hp[i][i]=1;
  }

```

```
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   sl=0;
135   cout << "\n Eigenvalues:\n";
136   for (int i=0; i<N; i++) {
137     sl+=A[i][i];
138     cout << A[i][i] << " ";
139   }
140
141   cout << "\n\n Track " << sl << "\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 $Q = H_1 H_2 ... H_{n-1}$. 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.

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

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

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

using namespace std;

const double eps=0.0001;

int sign(double a) {
  if (a>0) return 1;
  if (a<0) return -1;
  return 0;
}

void print(vector <vector<double>> v) {
  for (int i=0; i<v.size(); i++) {
    for (int j=0; j<v.size(); j++) {
      cout << v[i][j] << "\t\t";
    }
    cout << endl;
  }
}

int main() {
  int N;
  cin >> N;
  vector <vector <double>> A (N, vector <double>(N,0));
  vector <vector <double>> newA (N, vector <double>(N,0));
  vector <vector <double>> H (N, vector <double>(N,0));
  vector <vector <double>> Q (N, vector <double>(N,0));
  vector <vector <double>> newQ (N, vector <double>(N,0));
  vector <vector <double>> R (N, vector <double>(N,0));
  vector <vector <double>> vvt (N, vector <double>(N,0));
  vector <double> v (N,0);

  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      cin >> A[i][j];
    }
  }

  for (int i=0; i<N; i++) {
    Q[i][i]=1;
  }

  double max=0;
  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.