# 精度保証数値計算レポート（演習４(-2?)）

**提出先：大石進一教授**
**提出期限：2000/12/4**
**提出日：2000/12/4**

**学籍番号：699P017-2**

**情報学科 大須賀研究室**

**川崎 修平**

**演習 4(-2)．Jampack/JAMA/JavaLAPACK の少なくとも１つにより、連立一次方程式及び、固有値問題を解いて、上記パッケージの簡単な評価をせよ。**

**実行環境:**

**RedHat Linux 6.2J + JDK1.3 + JAMA1.0.1**

**インストール:**

それぞれのサイトを見てみて、ドキュメントの読みやすさから **JAMA** を選びました。
インストール作業は、
**http://math.nist.gov/javanumerics/jama/**
から、**Jama-1.0.1.jar** をダウンロードして解凍するだけでした。
ドキュメントはダウンロードできませんが、上記サイトで **javadoc** 形式の **HTML** ドキュメントを参照できます。

**(1) 連立一次方程式を解く**
簡単な例ですが、

$$\begin{pmatrix} -1.0 & 2.0 & 3.0 \\ 4.0 & -5.0 & 6.0 \\ 7.0 & 8.0 & -9.0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \end{pmatrix}$$

を解く、以下のリストを作成して実行しました。

**jamatest1.java**

```
import Jama.*;

public class jamatest1 {
       public static void main(String[] args) {
              Matrix A, b, x;
              double [][] A0 = {
                     {-1.0,  2.0,  3.0},
                     { 4.0, -5.0,  6.0},
                     { 7.0,  8.0, -9.0}};
              double [][] x0 = {{1.0, 1.0, 1.0}};

              A = new Matrix(A0);
              b = (new Matrix(x0)).transpose();
              x = A.solve(b);
              x.print(12, 10);
       }
}
```

**実行結果：**

```
~/Prog/jama> java jamatest1

  0.1833333333
  0.2333333333
  0.2388888889
```

参考までに、**octave** による実行結果は以下のとおりで、同様の結果が得られました。

```
octave:1> output_precision = 10;
octave:2> [-1 2 3;4 -5 6;7 8 -9]¥[1;1;1]
ans =

   1.833333333e-01
   2.333333333e-01
   2.388888889e-01
```

### (2) 固有値問題を解く

まずは、固有値が実数となる行列でテストを行いました。

**jamatest2.java**

```
import Jama.*;

public class jamatest2 {
       public static void main(String[] args) {
              Matrix A;
              EigenvalueDecomposition eig;
              double [][] A0 = {
                     { 1.0,  2.0},
                     { 2.0,  1.0}};

              A = new Matrix(A0);
              eig = A.eig();
              eig.getV().print(12, 10);
              eig.getD().print(12, 10);
       }
}
```

**実行結果:(octave の結果も含む)**

```
octave:1> output_precision = 10;
octave:2> [V,D] = eig([1 2;2 1])
V =
   -7.071067812e-01   7.071067812e-01
    7.071067812e-01   7.071067812e-01
D =
  -1   0
   0   3

~/Prog/jama> java jamatest2
  0.7071067812      0.7071067812
 -0.7071067812      0.7071067812

  -1.0000000000     0.0000000000
   0.0000000000     3.0000000000
```

これについては、octave と同様の結果が得られました。ただし、-1,0,3 が厳密に-1,0,3 であるかどうかが判別できない点は octave と異なります。

次に、固有値が複素数となる行列に関してテストを行いました。先ほどの jamatest2.java の行列を[1 2 3;2 2 1;-1 3 5]に置き換えて実行した結果は以下のとおりです(octave の結果も併載しました)。

---

```
octave:1> [V,D] = eig([1 2 3;2 2 1;-1 3 5])
V =
  -0.10906 + 0.30344i  -0.10906 - 0.30344i   0.57615 + 0.00000i
   0.75070 + 0.00000i   0.75070 + 0.00000i   0.44644 + 0.00000i
  -0.57563 - 0.03356i  -0.57563 + 0.03356i   0.68465 + 0.00000i
D =
  0.94266 + 0.76371i  0.00000 + 0.00000i  0.00000 + 0.00000i
  0.00000 + 0.00000i  0.94266 - 0.76371i  0.00000 + 0.00000i
  0.00000 + 0.00000i  0.00000 + 0.00000i  6.11468 + 0.00000i

~/Prog/jama> java jamatest2
 -0.3887990436 -0.0626601373  0.6689964846
  0.1688676724  0.9011793724  0.5183813348
 -0.0891957808 -0.6985657172  0.7949799005

  0.9426604251  0.7637141020  0.0000000000
 -0.7637141020  0.9426604251  0.0000000000
  0.0000000000  0.0000000000  6.1146791498
```

---

この出力は、octave のものと異なります。リファレンスページによると、

If A is symmetric, then A = V*D*V' where the eigenvalue matrix D is diagonal and the eigenvector matrix V is orthogonal. I.e. A = V.times(D.times(V.transpose())) and V.times(V.transpose()) equals the identity matrix.

If A is not symmetric, then the eigenvalue matrix D is block diagonal with the real eigenvalues in 1-by-1 blocks and any complex eigenvalues, lambda + i*mu, in 2-by-2 blocks, [lambda, mu; -mu, lambda]. The columns of V represent the eigenvectors in the sense that A*V = V*D, i.e. A.times(V) equals V.times(D). The matrix V may be badly conditioned, or even singular, so the validity of the equation A = V*D*inverse(V) depends upon V.cond().

ということなので、固有ベクトルの出力については意味のない結果が返ってきてしまう仕様のようです。固有値については、

$$\begin{pmatrix} x_{1real} & x_{1imag} & & & & \\ x_{1imag} & x_{1real} & & & O & \\ & & x_{2real} & x_{2imag} & & \\ & & x_{2imag} & x_{2real} & & \\ & O & & & x_{n-1} & \\ & & & & & x_n \end{pmatrix}$$

という形式で結果が返されるということでした。また、get(Real/Imag)Eigenvalues(),と

いうメソッドによって、実部/虚部のみの配列を返すこともできます。この場合のリストと出力を以下に記載します。

---

```
public class jamatest3 {
       public static void main(String[] args) {
              Matrix A;
              EigenvalueDecomposition eig;
              double [][] A0 = {
                     { 1.0,  2.0,  3.0},
                     { 2.0,  2.0,  1.0},
                     {-1.0,  3.0,  5.0}};

              A = new Matrix(A0);
              eig = A.eig();
              double[] real = eig.getRealEigenvalues();
              double[] imag = eig.getImagEigenvalues();
              double[][] V0 = {real, imag};
              new Matrix(V0).transpose().print(12, 10);
       }
}

~/Prog/jama> java jamatest3
  0.9426604251  0.7637141020
  0.9426604251 -0.7637141020
  6.1146791498  0.0000000000
```

---

後になって気づいたことですが、オフィシャルページのトップに、

The current JAMA deals only with real matrices. We expect that future versions will also address complex matrices.

という記述がありました。そもそも現バージョンでは複素数の演算はサポートされておらず、複素数の結果は固有値の場合のみの特殊対応であったようです。

**JAMA** はパッケージとしてはドキュメントも充実していて、命令体系も **octave/matlab** と名前が一致しているものが多いので、快適に利用することができました。今回は修士論文が忙しく、速度面等での評価等を行うことができませんでしたが、今後機会があれば利用したいと感じました。