精度保証付き数値計算
【演習 5】

600P0328 竹内智彦

平成13年 2月 5日

問題 5：Jampack,Jama,Java LAPACK の 3 つのパッケージのうち、少なくとも 1 つより、連立一次方程式及び固有値問題を解いて簡単な評価をせよ。

私は Jama,Java LAPACK を使用してみた。実験環境は、Pentium 166MHz,Memory 48MB,Linux JDK 1.3 である。

1 インストール及び使用感

Jama は JAR ファイルがあり、それをダウンロードし、環境変数の CLASSPATH を編集すれば使用できる。
Java LAPACK は、よく web ページが見られなくなるが、それを除けば tgz ファイルを伸長し、その中にある classes.zip の PATH を CLASSPATH に追加すれば使用できる。

両方ともソースコードが公開されているため、引数の数や方がわからないときは、ソースコードをみて使用した。

特に再コンパイルすることなく使用でき、また、一つの環境変数の変更のみで使用でき、Java の良さを体験することが出来た。

2 連立一次方程式

2.1 Jama

Jama を使ったソースは以下のとおりである。

```
import java.util.*;

import Jama.*;

public class RepoJama{

   public static void main(String[] args){
```

```
        int n = 500;
        Matrix A,x,b;
        Date start,end;

        A = Matrix.random(n,n);
        b = Matrix.random(n,1);

        start = new Date();
        x = A.solve(b);
        end = new Date();
        System.out.println((end.getTime() - start.getTime()) / 1000.0 +" seconds");
    }
}
```

実行結果は以下の通りである。

```
[tomohiko@cj3059872-a REPO5]$ time java RepoJama
10.71 sec
13.51user 0.72system 0:16.06elapsed 88%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (5177major+3592minor)pagefaults 0swaps
```

計算時間は、10.71 秒であった。オブジェクトの生成などで、3 秒ほどかかっていると考えられる。

岩村氏のレポートに、Jama は octave と比べて決して遅れをとっているわけではなさそうだという記述があったので、私の環境でも試してみた。使用した octave は特に最適化はされていない。

以下が octave のソースである。

```
[tomohiko@cj3059872-a tomohiko]$ octave
GNU Octave, version 2.0.16 (i386-redhat-linux-gnu).
Copyright (C) 1996, 1997, 1998, 1999, 2000 John W. Eaton.
This is free software with ABSOLUTELY NO WARRANTY.
For details, type 'warranty'.

octave:1> n = 500
n = 500
octave:2> A = rand(n,n);
octave:3> b = rand(n,1);
octave:4> tic(); x = A \ b ; toc()
ans = 6.3467
octave:5> tic(); x = A \ b ; toc()
ans = 6.2444
```

octave の実行結果は 6 秒ほどであった。これをみても、Jama は octave と比べても決して遅いとはいえないだろう。

2.2 Java LAPACK

Java LAPACK を用いたソースは以下の通り。

```
import org.netlib.util.*;
import java.util.*;
import org.netlib.lapack.*;

public class Repo5 {
  public static void main(String [] args) {

      int n = 500;
      double[] da = new double[n*n];
      double[] db = new double[n];

      int nrhs = 1;
      int lda = n;
      int ldb = n;
      int off_set_a=0;
      int off_set_b=0;
      int off_set_ipiv=0;
      int[] ipiv = new int[n] ;
      int i,l;

      Date start,end;

      for(i=0;i<n;i++){
          for(l=0;l<n;l++){
              da[(i*n)+l] = Math.random() * 100;
          }
          db[i] = Math.random() * 100;
      }

      intW info = new intW(0);

      start = new Date();
      Dgesv.dgesv(n,nrhs,da,off_set_a,lda,ipiv,off_set_ipiv,db,
                  off_set_b,ldb,info);
      end = new Date();
      System.out.println( (end.getTime() - start.getTime())/1000.0 + "seconds.");

  }
}
```

実行結果は以下の通りである。

```
[tomohiko@cj3059872-a REPO5]$ time java Repo5
26.094seconds.
27.58user 0.61system 0:29.87elapsed 94%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (4403major+3072minor)pagefaults 0swaps
```

計算時間は、26.094 秒であった。Jama より 2 倍以上遅い結果がでた。また、次元数を変化させても、2 倍以上の差がついていた。

この原因を探るためには、使用コンピュータを変えてみたり、JIT を変えてみてまず使用環境によるものなのか、パッケージの実装によるものなのかをはっきりさせる必要があるだろう。

3 固有値問題

3.1 Jama

Jama を使用したソースは以下の通りである。

```
import java.util.*;

import Jama.*;

public class RepoEigJama{
public static void main(String[] args)
    {

        double[][] d = {{1.0,2.0,3.0},{2.0,3.0,-4.0},{3.0,-4.0,5.0}};
        Matrix A;
        EigenvalueDecomposition e;
        A = new Matrix(d);
        e = A.eig();
        System.out.println("V =");
        e.getV().print(8,8);
        System.out.println("D =");
        e.getD().print(8,8);
    }
}
```

実行結果は以下の通り。

```
V =

 0.65212944 -0.73480651 -0.18651164
 -0.55465417 -0.63016687 0.54336770
 -0.51680358 -0.25089662 -0.81851386

D =
```

```
 -3.07851408 0.00000000 0.00000000
 0.00000000 3.73952880 0.00000000
 0.00000000 0.00000000 8.33898528
```

3.2 Java LAPACK

Java LAPACK を使用したソースは以下の通り。

```
import org.netlib.util.*;
import java.util.*;
import org.netlib.lapack.*;

public class RepoEig {
  public static void main(String [] args) {

      int n = 3;
      double[] da = new double[n*n];
      double[] w = new double[n];
      double[] work = new double[3*n];
      String jobs = "V";
      String uplo = "U";
      int lda = n;
      int lwork =3*n;
      int off_set_a = 0;
      int off_set_w = 0;
      int off_set_work = 0;
      intW info = new intW(0);
      int i,l;

      da[0]=1.0;da[1]=2.0;da[2]=3.0;
      da[3]=2.0;da[4]=3.0;da[5]=-4.0;
      da[6]=3.0;da[7]=-4.0;da[8]=5.0;

      Dsyev.dsyev(jobs,uplo,n,da,off_set_a,n,w,off_set_w,
                  work,off_set_work,lwork,info);

      System.out.println("固有値");
      for(i=0;i<w.length;i++){
          System.out.print( w[i] + "  " );
      }
      System.out.println("");

      System.out.println("固有ベクトル");
      for(l=0;l<n;l++){
```

```
            for(i=0;i<n;i++){
                System.out.print( da[(i*n)+l] + "  " );
            }
            System.out.println("");
        }

    }
}
```

実行結果は以下の通り

固有値
-3.0785140762667664 3.7395288001734963 8.338985276093268
固有ベクトル
-0.6521294385337699 0.7348065074634955 0.18651164035236512
0.5546541742011382 0.6301668703570372 -0.543367704731953
0.516803582068549 0.2508966163456909 0.8185138639433048

このように、固有値が実数ならば、Jama, Java LAPACK とも正負は逆になっているものの同じ解が得られた。しかし、固有値が複素数の場合を計算させると、全く違う値となった。

おそらく複素数を扱うクラスが別にあるのだと考えられる。