数値計算：行列 (III, 固有値-2)

関西学院大学・理工学部・情報科学科　西谷滋人

2006 年 11 月 10 日

1　累乗 (べき乗) 法により最大固有値が求まる原理

累乗 (べき乗) 法は，最大固有値とその固有ベクトルを効率的に見つける算法である．すこし，固有値について復習しておく．正方行列 A に対して，

$$A\boldsymbol{x} = \lambda\boldsymbol{x} \tag{1}$$

の解 λ を固有値，x を固有ベクトルという．λ は，

$$\det(A - \lambda E) = 0 \tag{2}$$

として求まる永年方程式の解である．

では，なぜ適当な初期ベクトル $\boldsymbol{X}_0$ から始めて，反復

$$\boldsymbol{X}_{k+1} = A\boldsymbol{X}_k \tag{3}$$

を繰り返すと，A の絶対値最大の固有値に属する固有ベクトルに近づいていくのかを見ておこう．

すべての固有値がお互いに異なる場合を考える．今，行列の固有値を絶対値の大きなもの順に並べて，$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ とし，対応する長さを 1 に規格化した固有ベクトルを $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ とする．初期ベクトルは固有ベクトルの線形結合で表わせて，

$$\boldsymbol{X}_0 = c_1\boldsymbol{x}_1 + c_2\boldsymbol{x}_2 + \cdots + c_n\boldsymbol{x}_n \tag{4}$$

となるとする．これに行列 A を N 回掛けると，

$$A^N\boldsymbol{X}_0 = c_1\lambda_1^N\boldsymbol{x}_1 + c_2\lambda_2^N\boldsymbol{x}_2 + \cdots + c_n\lambda_n^N\boldsymbol{x}_n \tag{5}$$

となる．これを変形すると，

$$A^N\boldsymbol{X}_0 = \boldsymbol{X}_N = c_1\lambda_1^N\left\{\boldsymbol{x}_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^N\boldsymbol{x}_2 + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^N\boldsymbol{x}_n\right\} \tag{6}$$

となる．$|\lambda_1| > |\lambda_i| (i \geq 2)$ だから括弧の中は $\boldsymbol{x}_1$ だけが生き残る．

こうして最大固有値に属する固有ベクトルが，反復計算を繰り返すだけで求められる．

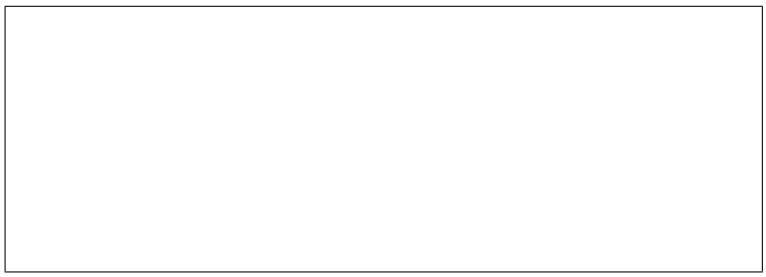
2 Jacobi回転による固有値の求め方

固有値を求める手法として，永年方程式を解くというやり方は回りくどすぎる．少し古めかしいが非対角要素を 0 にする回転行列を反復的に作用させる Jacobi(ヤコビ) 法を紹介する．現在認められている最適の方策は，ハウスホルダー (Householder) 変換で行列を単純な三重対角化行列に変形してから，反復法で解を追い込んでいくやり方である．Jacobi 法は，Householder 法ほど万能ではないが，10 次程度までの行列には今でも役に立つ．

2.1 Mapleでみる回転行列

行列の軸回転の復習をする．対称行列 B に回転行列 U を作用すると

$$B.U = \begin{pmatrix} a_{1\,1} & a_{1\,2} \\ a_{1\,2} & a_{2\,2} \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{7}$$



(8)

となる．回転行列を 4x4 の行列に

$$U^t B U \tag{9}$$

と作用させたときの各要素の様子を以下に示した．

```
> restart:
n:=4:
with(LinearAlgebra):
B:=Matrix(n,n,shape=symmetric,symbol=a);
```

$$B := \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{1,2} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{1,3} & a_{2,3} & a_{3,3} & a_{3,4} \\ a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} \end{bmatrix}$$

```
> U:=Matrix(n,n,[[c,-s,0,0],[s,c,0,0],[0,0,1,0],[0,0,0,1]]);
#U:=Matrix(n,n,[[c,-s],[s,c]]);
```

$$U := \begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> TT:=Transpose(U).B.U;
```

$TT :=$

$\left[(c\,a_{1,1} + s\,a_{1,2})\,c + (c\,a_{1,2} + s\,a_{2,2})\,s\,,\ -(c\,a_{1,1} + s\,a_{1,2})\,s + (c\,a_{1,2} + s\,a_{2,2})\,c\,,\right.$
$\left. c\,a_{1,3} + s\,a_{2,3}\,,\ c\,a_{1,4} + s\,a_{2,4}\right]$

$\left[(-s\,a_{1,1} + c\,a_{1,2})\,c + (-s\,a_{1,2} + c\,a_{2,2})\,s\,,\ -(-s\,a_{1,1} + c\,a_{1,2})\,s + (-s\,a_{1,2} + c\,a_{2,2})\,c\,,\right.$
$\left. -s\,a_{1,3} + c\,a_{2,3}\,,\ -s\,a_{1,4} + c\,a_{2,4}\right]$

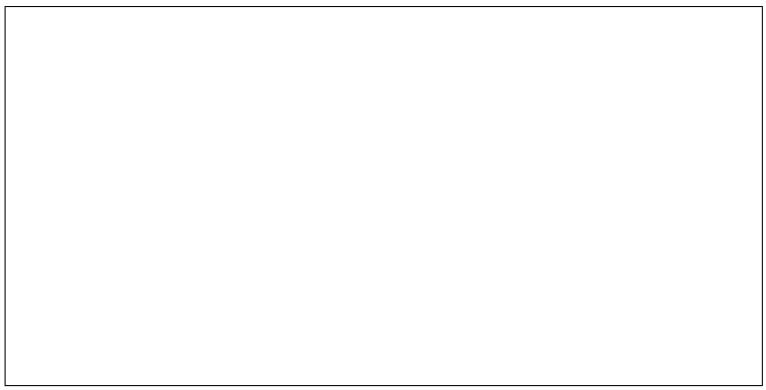
$[c\,a_{1,3} + s\,a_{2,3}\,,\ -s\,a_{1,3} + c\,a_{2,3}\,,\ a_{3,3}\,,\ a_{3,4}]$

$[c\,a_{1,4} + s\,a_{2,4}\,,\ -s\,a_{1,4} + c\,a_{2,4}\,,\ a_{3,4}\,,\ a_{4,4}]$

```
> expand(TT[1,1]);
expand(TT[2,2]);
expand(TT[1,2]);
expand(TT[2,1]);
```

$$c^2\,a_{1,1} + 2\,c\,s\,a_{1,2} + s^2\,a_{2,2}$$
$$s^2\,a_{1,1} - 2\,c\,s\,a_{1,2} + c^2\,a_{2,2}$$
$$-s\,c\,a_{1,1} - s^2\,a_{1,2} + c^2\,a_{1,2} + c\,s\,a_{2,2}$$
$$-s\,c\,a_{1,1} - s^2\,a_{1,2} + c^2\,a_{1,2} + c\,s\,a_{2,2}$$

この非対角要素を 0 にする θ は以下のように求まる.



(10)

このとき注目している $i, j = 1, 2$ 以外の要素も変化する.

```
> expand(TT[3,1]);
expand(TT[3,2]);
```

$$c\,a_{1,3} + s\,a_{2,3}$$
$$-s\,a_{1,3} + c\,a_{2,3}$$

これによって一旦 0 になった要素も値を持つが，なんども繰り返すことによって，徐々に 0 へ近づいていく．

2.2 Jacobi 法による固有値を求める C コード

以下にはヤコビ法を用いた固有値と固有ベクトルを求めるコードを示した．結果は，固有値とそれに対応する規格化された固有ベクトルが縦 (column) ベクトルで表示される．

リスト 1: ヤコビ法.

```
#include <stdio.h>
#include <math.h>

#define M 10
void PrintMatrix(double a[M][M], int n);

int main(void){
  double a[M][M],v[M][M];
  double eps=0.0001,div,r,t,s,c,apj,aqj,aip,aiq,vip,viq;
  int i,j,n,iter,count,iterMax=1000000,p,q;

  scanf("%d",&n);
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) scanf("%lf",&a[i][j]);
  }
  PrintMatrix(a,n);
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) v[i][j]=0.;
    v[i][i]=1.;
  }

  for(iter=1;iter<=iterMax;iter++){
    count=0;
    for(p=1;p<=n-1;p++){
      for(q=p+1;q<=n;q++){
        if(fabs(a[p][q])<eps) continue;
        count++;
        div=a[p][p]-a[q][q];
        if (div != 0.0){
          r=2.0*a[p][q]/div;
          t=0.5*atan(r);
```

```
        } else {
          t=0.78539818;
        }
        s=sin(t);
        c=cos(t);
        for(j=1;j<=n;j++){
          apj=a[p][j];
          aqj=a[q][j];
          a[p][j]=apj*c+aqj*s;
          a[q][j]=−apj*s+aqj*c;
        }
        for(i=1;i<=n;i++){
          aip=a[i][p];
          aiq=a[i][q];
          a[i][p]=aip*c+aiq*s;
          a[i][q]=−aip*s+aiq*c;
          vip=v[i][p];
          viq=v[i][q];
          v[i][p]=vip*c+viq*s;
          v[i][q]=−vip*s+viq*c;
        }
        printf("p,q=%3d,%3d\n",p,q);
        PrintMatrix(a,n);
      }
    }
    if (count==0) break;
  }
  printf("Eigen values:\n");
  for(i=1;i<=n;i++) printf("%6.2f",a[i][i]);
  printf("\nEigen vectors:\n");
  PrintMatrix(v,n);

  return 0;
}

void PrintMatrix(double a[M][M], int n){
  int i,j;
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) printf("%6.2f",a[i][j]);
    printf("\n");
  }
  printf("\n");
}
```

リスト 2: ヤコビ法の計算結果.

```
[BobsNewPBG4:~/NumRecipe/chap8] bob% cat input.txt
4
5 4 1 1
4 5 1 1
```

```
1 1 4 2
1 1 2 4

BobsNewPBG4:~/NumRecipe/chap8] bob% Jacobi2<input.txt
  5.00  4.00  1.00  1.00
  4.00  5.00  1.00  1.00
  1.00  1.00  4.00  2.00
  1.00  1.00  2.00  4.00

p,q=  1,  2
  9.00 -0.00  1.41  1.41
 -0.00  1.00 -0.00 -0.00
  1.41 -0.00  4.00  2.00
  1.41 -0.00  2.00  4.00

p,q=  1,  3
  9.37 -0.00 -0.00  1.88
 -0.00  1.00  0.00 -0.00
 -0.00  0.00  3.63  1.57
  1.88 -0.00  1.57  4.00

p,q=  1,  4
  9.96 -0.00  0.47 -0.00
 -0.00  1.00  0.00  0.00
  0.47  0.00  3.63  1.50
  0.00  0.00  1.50  3.41

...<中略>...

Eigen values:
 10.00  1.00  5.00  2.00
Eigen vectors:
  0.63 -0.71 -0.32  0.00
  0.63  0.71 -0.32  0.00
  0.32  0.00  0.63 -0.71
  0.32  0.00  0.63  0.71
```

3　数値計算ライブラリーについて

一般の数値計算ライブラリーについては，時間の関係で講義ではその能力を紹介するにとどめる．演習で詳しく取り上げているので，研究や今後のために必要と思うときは，テキストを取りにおいで．

行列の計算は，数値計算の中でも特に利用する機会が多く，また，律速ルーチンとなる可能性が高い．そこで，古くから行列計算の高速ルーチンが開発されてきた．なかでも BLAS と LAPACK はフリーながら非常に高速である．

前回に示した，逆行列を求める単純な LU 分解法を C 言語でコーディングしたものと，LAPACK のルーチンを比べた場合，1000 次元の行列で計測すると

```
>  1000 [dim]     2.5200 [sec] #BOB
>  1000 [dim]     0.4700 [sec] #LAPACK
```

となった．用いた PC は MacBook(2GHz Interl Core Duo) であるが，この計算での 0.47 秒は 1.4GFLOP に相当する．ライブラリーは世界中の計算機屋さんがよってたかって検証しているので，バグがほとんど無く，また，高速である．初学者はライブラリーを使うべきである．ただし，下のサンプルプログラムの行列生成の違いのように，ブラックボックス化すると思わぬ間違い（ここでは Fortran と C との行列の並び方の違いが原因) をしでかすことがあるので，プログラムに組み込む前に必ず小さい次元 (サンプルコード) で検証しておくこと．

ちょっと長いが時間があればフォローせよ．

リスト 3: 西谷製 lazy 逆行列計算プログラム．

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//#undef PRINT
//#define PRINT

void printMatrix(double *a, double *b, long n);
int MatrixInverse(double *a, double *b, long n);

int main(void){
  clock_t start, end;
  int i,j;
  long n;
  double *a,*b;

  scanf("%ld",&n);

  a=(double *)malloc(n*n*sizeof(double));
  b=(double *)malloc(n*sizeof(double));

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[i*n+j]= 2*(double) random() / RAND_MAX - 1.0;
    }
  }
  for (i=0;i<n;i++){
    b[i]= 2*(double) random() / RAND_MAX - 1.0;
  }
  printMatrix(a,b,n);
```

```
  start = clock();
  MatrixInverse(a,b,n);
  end = clock();
  printf("%5d [dim] %10.4f [sec] #BOB\n",
         n,(double)(end-start)/CLOCKS_PER_SEC);
  printMatrix(a,b,n);

  free(a);
  free(b);
  return 0;
}

int MatrixInverse(double *a, double *b, long n){
  double *x;
  double pvt=0.00005,am;
  int i,j,k;

  x=(double *)malloc(n*sizeof(double));

  for(i=0;i<n-1;i++){
    if(fabs(a[i*n+i])<pvt){
      printf("Pivot %3d=%10.5f is too small.\n",i,a[i*n+i]);
      return 1;
    }
    for(j=i+1;j<n;j++){
      am=a[j*n+i]/a[i*n+i];
      for(k=0;k<n;k++) a[j*n+k]-=am*a[i*n+k];
      b[j]-=am*b[i];
    }
  }
  //Backward substitution
  for(j=n-1;j>=0;j--){
    x[j]=b[j];
    for(k=j+1;k<n;k++){
      x[j]-=a[j*n+k]*x[k];
    }
    b[j]=x[j]/=a[j*n+j];
  }
  free(x);
  return 0;
}

void printMatrix(double *a, double *b, long n){
  int i,j;
#ifdef PRINT
  printf("\n");
  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      printf("%10.5f",a[i*n+j]);
    }
    printf(":%10.5f",b[i]);
    printf("\n");
  }
```

```
  printf("\n");
#endif
  return;
}
```

リスト 4: LAPACK 謹製 smart 逆行列計算プログラム.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <vecLib/vecLib.h>

void printMatrix(double *a, double *b, long n);

int main(void){
  clock_t start, end;
  int i,j;
  double *a,*b;
  long n,nrhs=1, lda,ldb, info, *ipiv;

  scanf("%ld",&n);

  a=(double *)malloc(n*n*sizeof(double));
  b=(double *)malloc(n*sizeof(double));
  lda=ldb=n;
  ipiv=(long *)malloc(n*sizeof(long));

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[j*n+i]= 2*(double) random() / RAND_MAX - 1.0;
    }
  }

  for (i=0;i<n;i++){
    b[i]= 2*(double) random() / RAND_MAX - 1.0;
  }
  printMatrix(a,b,n);

  start = clock();
  dgesv_(&n, &nrhs, a, &lda, ipiv, b, &ldb, &info);
  end = clock();
  printf("%5d [dim] %10.4f [sec] #LAPACK\n",
         n, (double)(end-start)/CLOCKS_PER_SEC);
  printMatrix(a,b,n);

  free(a);
  free(b);
  free(ipiv);

  return 0;
}
```

PrintMatrix は bob と同じなので省略．OSX では

```
gcc -O3 -UPRINT lapack.c -llapack -lblas
```

とすればコンパイルできる．linux では LAPACK, BLAS がインストールされていれば，

```
#include <vecLib/vecLib.h>
```

をコメントアウトして，

```
gcc -O3 -DPRINT lapack.c -L/usr/local/lib64 -llapack -lblas -lg2c
```

などとすればコンパイルできるはず．

課題

1. 4x4 の行列を適当に作り，Maple で固有値を求めよ．求め方はマニュアルを参照せよ．
2. Jacobi 法によって固有値を求めよ．
3. LAPACK に含まれている dseyv 関数を用いて固有値を求めよ．(演習で詳しく取り上げている．研究や今後のために必要と思うときは，テキストを取りにおいで)