応用数理実験

固有値問題 (2) 実対称行列の三重対角化

桂田 祐史

2002 年 7 月 9 日

1 今回の課題

次の二つの課題のうち、いずれか一つを選択する。

昨年度までは C 言語 (または Fortran で) プログラムを書く人が多かったが、Octave を使っても構わない。

1.1 実対称行列の三重対角化

(1) Householder 変換または Lanczos 法により実対称行列を三重対角行列に相似変換するプログラムを作成せよ (相似変換のための行列は求めなくとも構わない)。

(2) 実対称三重対角行列に対して (その特性を十分生かして)、冪(べき)乗法、逆反復法により、絶対値が最大の固有値と絶対値最小の固有値を求めるプログラムを作成し、(1) で作ったプログラムと結合せよ。

(3) (二分法はまだ説明していない。次週説明する。) 三重対角行列に対して、二分法により固有値を求めるプログラムを作成し、実験せよ。

三重対角化のプログラムのテスト用のデータとしては、例えば

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

を用いよ[1]。これを講義で説明した Householder 法[2]で三重対角化すると

$$P^{-1}AP = \begin{pmatrix} 1.0000000000 & -1.7320508076 & 0 & 0 \\ -1.7320508076 & 7.6666666667 & 1.2472191289 & 0 \\ 0 & 1.2472191289 & 0.9761904762 & -0.1237179148 \\ 0 & 0 & -0.1237179148 & 0.3571428571 \end{pmatrix}$$

[1] この行列も前回と同様、一松信著「数値解析」朝倉書店 (1982) から採った。

[2] Lanczos 法を用いた場合も、相似変換のための直交行列の第 1 列ベクトルを $\mathbf{e}_1 = {}^t(1, 0, \cdots, 0)$ に取ると、(符号の違いを無視すれば) 同じ結果になるはずである。

となる。さらに固有値は

$$\begin{aligned}
\lambda_1 &= 8.290859369381589606621222410409759227\cdots,\\
\lambda_2 &= 1,\\
\lambda_3 &= 0.426022047760461836484914938273277876 12\cdots,\\
\lambda_4 &= 0.283118582857948556893862651316962896 25\cdots,
\end{aligned}$$

となる。

プログラムの正しさに自信が持てたら大きな行列で実験してみること。

1.2 QR 法の数値実験

Octave を使って QR 法の実験を行なう。QR 分解をする関数は `qr()` である。

例えば、行列 `a` に対して、QR 変換を 10 回施す (そして中間結果を表示する) には以下のようにすれば良い。

```
for i=1:10
  [q r]=qr(a);
  a=r*q
end
```

- 行列 A の構造 (実対称性や帯行列であることなど) は QR 変換で保存されるかどうか、数値計算の結果を観察して推測せよ (もちろん証明できれば良い)。
- 対角成分を固有値の近似値として採用するとして、収束の速さを調べよ。

1.3 ちょっと自分でやってみました

```
Octave メモ

# 『三重対角行列の QR 変換は三重対角ではないが、Hessenberg 形である。』
# つまり『Hessenberg の QR 変換は Hessenberg』ということかな？
# 三重対角行列のサンプルを作るには diag() が利用できる。
n=4; d=rand(n,1); u=rand(n-1,1); l=rand(n-1,1);
a=diag(d,0)+diag(u,1)+diag(l,-1)
for i=1:100
  [q r]=qr(a);
  a=r*q
end

# ちなみに関数にしておくと、
# function a = randtrid(n)
#   a=diag(rand(n,1),0)+diag(rand(n-1,1),1)+diag(rand(n-1,1),-1);
# endfunction

# 『実対称行列の QR 変換は実対称行列である。』
# 実対称行列のサンプルを作るには、三角部分の切り出し tril(), triu() の
# 利用も考えられるが、良く知られた命題
```

```
# 『任意の行列 A について、行列の「対称部分」 (A+A^T)/2 は対称行列になる。』
# を使うのが簡単だろう。
n=4;a=rand(n,n);a=(a+a')/2
for i=1:100
  [q r]=qr(a);
  a=r*q
end

# 『実対称三重対角行列の QR 変換は三重対角行列である。』
# (∵対称な Hessenberg 行列は三重対角であるから)
n=4; d=rand(n,1); u=rand(n-1,1); a=diag(d,0)+diag(u,1)+diag(u,-1)
for i=1:100
  [q r]=qr(a);
  a=r*q
end

# 最終的な結果と、eig() を使って得た近似固有値との比較
# a を変換してしまう前に近似固有値を計算して保存しておく必要がある。
n=4; d=rand(n,1); u=rand(n-1,1); a=diag(d,0)+diag(u,1)+diag(u,-1)
e=eig(a);
for i=1:100
  [q r]=qr(a);
  a=r*q
end
a
e

# 毎回、対角成分を eig() で計算した近似固有値と並べて見る
# 遅くなるので、あまりループは使いたくないが
#   d=zeros(n,1);
#   for j=1:n
#     d(j)=a(j,j)
#   end
# で対角成分を収めたベクトル d が得られる。
# e と d を並べて表示するには、
#   [e d]
# とするのが簡単。比較するためには、両方とも大きさの順に並べておくのがよい。
# それには sort() を使えばよい。
n=4;a=rand(n,n);a=(a+a')/2
e=sort(eig(a));
for i=1:100
  [q r]=qr(a);
  a=r*q
  d=zeros(n,1);
  for j=1:n
    d(j)=a(j,j);
  end
  d=sort(d);
  format long
  [e d]
  norm(d-e)
  format short
end

# 収束の様子を対角線の下側の成分が小さくなることで確認してみよう。
# 上三角部分 (対角線込み) を取り出す triu() を用いて
n=4;a=rand(n,n);a=(a+a')/2
e=sort(eig(a));
```

```
for i=1:100
  [q r]=qr(a);
  a=r*q
  norm(a-triu(a))
end
# とするのが考えられる。
# 元の行列が対称の場合は、非対称部分の大きさを見るのも良いだろう。
# 対角部分の切り出しは、a .* eye(n,n) で出来るので、d-a のノルムを見てみる。
n=4;a=rand(n,n);a=(a+a')/2
e=sort(eig(a));
for i=1:100
  [q r]=qr(a);
  a=r*q
  d=a .* eye(n,n);
  norm(d-a)
end

# 結果が大きいとき、それを読むために、less のようなページャーが呼び出
# されて、読み終わった後、画面から消えてしまうが、それをファイルに残し
# たい場合は、(END) が出ているときに、s と打ってから LOG ファイルの名前
# を入力する。

#
function ret = rand_hessenberg(n)
  ret = triu(rand(n,n))w + diag(rand(n-1,1),-1);
endfunction
#
function QR = qr_iteration(a,n)
  QR = a;
  for i=1:n
    [q r]=qr(QR);
    QR=r*q;
  end
endfunction
#
a=rand_hessenberg(10)
a=qr_iteration(a,1)
# 後は繰り返し、というのもいいかも
----------------------------------------------------------------
function ret = rand_h(n)
  ret = triu(rand(n,n)) + diag(rand(n-1,1),-1);
endfunction
function ret = rand_t(n)
  ret=diag(rand(n,1),0)+diag(rand(n-1,1),1)+diag(rand(n-1,1),-1);
endfunction
function ret = rand_st(n)
  u=rand(n-1,1);
  ret=diag(rand(n,1),0)+diag(u,1)+diag(u,-1);
endfunction
function ret = rand_s(n)
  a=rand(n,n);
  ret = (a+a')/2;
endfunction
function QR = qr_iteration(a,n)
  QR = a;
  for i=1:n
    [q r]=qr(QR);
    QR=r*q;
```

```
  end
endfunction
function n = norm_1(a)
  n = norm(a-triu(a));
endfunction
#
## はて、対称でない場合、固有値に虚数が出るが、
## QR して、どうして虚数が出てくるのかな？
```

2 参考 — 三重対角行列に対する Gauss の消去法

2.1 C 言語版

trid-lu.c, trid-lu.h (とそれらの “添字が 1 から始まる” バージョン trid-lu1.c, trid-lu1.h) は、http://www.math.meiji.ac.jp/%7Emk/program/ に置いてある。

関数 trid(), trilu(), trisol()

```
 1  /* trid-lu.c -- 3 項方程式を Gauss の消去法で解く */
 2
 3  #include "trid-lu.h"
 4
 5  /* 3 項方程式（係数行列が三重対角である連立 1 次方程式のこと）Ax=b を解く
 6   *
 7   *   入力
 8   *     n: 未知数の個数
 9   *     al,ad,au: 連立 1 次方程式の係数行列
10   *       (al: 対角線の下側 i.e. 下三角部分 (lower part)
11   *        ad: 対角線       i.e. 対角部分 (diagonal part)
12   *        au: 対角線の上側 i.e. 上三角部分 (upper part)
13   *        つまり
14   *
15   *         ad[0] au[0]   0   ................. 0
16   *         al[1] ad[1] au[1]   0  ............. 0
17   *           0   al[2] ad[2] au[2]  0 ......... 0
18   *                          ...................
19   *                          al[n-2] ad[n-2] au[n-2]
20   *                            0     al[n-1] ad[n-1]
21   *
22   *        al[i] = A_{i,i-1}, ad[i] = A_{i,i}, au[i] = A_{i,i+1},
23   *        al[0], au[n-1] は意味がない)
24   *
25   *     b: 連立 1 次方程式の右辺の既知ベクトル
26   *        (添字は 0 から。i.e. b[0],b[1],...,b[n-1] にデータが入っている。)
27   *   出力
28   *     al,ad,au: 入力した係数行列を LU 分解したもの
29   *     b: 連立 1 次方程式の解
30   *   能書き
31   *     一度 call すると係数行列を LU 分解したものが返されるので、
32   *     以後は同じ係数行列に関する連立 1 次方程式を解くために、
33   *     関数 trisol() が使える。
34   *   注意
35   *     Gauss の消去法を用いているが、ピボットの選択等はしていな
```

```
 *      いので、ピボットの選択をしていないので、係数行列が正定値である
 *      などの適切な条件がない場合は結果が保証できない。
 */

void trid(int n, double *al, double *ad, double *au, double *b)
{
  trilu(n,al,ad,au);
  trisol(n,al,ad,au,b);
}

/* 三重対角行列の LU 分解 (pivoting なし) */
void trilu(int n, double *al, double *ad, double *au)
{
  int i, nm1 = n - 1;
  /* 前進消去 (forward elimination) */
  for (i = 0; i < nm1; i++) {
    al[i + 1] /= ad[i];
    ad[i + 1] -= au[i] * al[i + 1];
  }
}

/* LU 分解済みの三重対角行列を係数に持つ 3 項方程式を解く */
void trisol(int n, double *al, double *ad, double *au, double *b)
{
  int i, nm1 = n - 1;
  /* 前進消去 (forward elimination) */
  for (i = 0; i < nm1; i++) b[i + 1] -= b[i] * al[i + 1];
  /* 後退代入 (backward substitution) */
  b[nm1] /= ad[nm1];
  for (i = n - 2; i >= 0; i--) b[i] = (b[i] - au[i] * b[i + 1]) / ad[i];
}
```

関数 trid() の使用例

```
/*
 * test-lu.c --- trilu0(), trisol0() のテスト
 *   コンパイル&リンク: gcc -o test-lu test-lu.c trid-lu.c
 *   trid-lu.c, trid-lu.h の入手は以下の URL で
 *   http://www.math.meiji.ac.jp/~mk/program/linear/trid-lu.c
 *   http://www.math.meiji.ac.jp/~mk/program/linear/trid-lu.h
 */

#define NDIM 100

#include <stdio.h>
#include <math.h>
#include "trid-lu.h"

int main()
{
  int i, n, nm1;
  double al[NDIM],ad[NDIM],au[NDIM],b[NDIM],x[NDIM];
  n = 10; nm1 = n - 1;
  /* A */
  ad[0] = 2.0; au[0] = -1.0;
  for (i = 1; i < nm1; i++) {
    al[i] = -1.0; ad[i] = 2.0; au[i] = - 1.0;
  }
```

```
25    al[nm1] = - 1.0; ad[nm1] = 2.0;
26    /* x */
27    for (i = 0; i < n; i++)
28      x[i] = i;
29    /* 右辺 */
30    b[0] = ad[0] * x[0] + au[0] * x[1];
31    for (i = 1; i < nm1; i++)
32      b[i] = al[i] * x[i-1] + ad[i] * x[i] + au[i] * x[i+1];
33    b[nm1] = al[nm1] * x[nm1-1] + ad[nm1] * x[nm1];
34    /* 解く */
35    trilu(n, al, ad, au);
36    trisol(n, al, ad, au, b);
37    /* */
38    for (i = 0; i < n; i++)
39      printf("%f\n", b[i]);
40    return 0;
41  }
```