2.5 行列 組込み関数(2) ver. 0.1

引き続き組込み関数について、少々高度な行列操作や複雑なことをする場合に使うものについて説明します。まず、inv と invpd についてですが、基本的に、戻り値は同じものとなります。これら逆行列を計算する組込み関数は、引数に入る行列の種類に依存します。まず、組込み関数 inv の引数には、正方行列であればどんな一般的な行列でも入ります。一方、invpd の方の引数には、正方行列でかつ対称的で positive definite な行列が入ります。これらの組込み関数は、もともとプログラムのアルゴリズム自体が異なります。アメリカの学部およびマスターレベルの教育を受けた人ならわかると思いますが、手で逆行列を求めるのにはかなりのステップを要します。行列のディメンションが大きくなると計算機と言えども、多大な時間がかかるわけです。前者の inv は一般的な正方行列を計算していくのに対して、後者の invpd は行列が対称でかつ positive definite であることを想定して計算しているので、行列のディメンションが大きくなれば、後者の invpd の方が計算スピードは上がります。ちなみに、前者の inv は内部で Crout 分解のアルゴリズムを用いているのに対して、後者の invpd は内部で Cholesky 分解のアルゴリズムを用いている違いがあります。通常の regression などでモーメント x'x をつくってそれを逆行列にする作業では、すでに x'x は対称形をしていて、しかも、たいていの場合 positive definite の性質をもちますから、invpd をそのまま使ってもさしつかえありません。

GAUSS 内部のシステムクロックを使って簡単な時間計測をしたところ、少なくともライト版が許すところの数十×数十の逆行列計算にはほとんど時間の差はありませんでした。ただし、正式版、とくにバージョン 3.2 でもっと大きなディメンションの行列の逆行列の計算をすると inv では数値演算プロセッサーのオーバーフローを起こします。invpd はメモリの許すかぎりオーバーフローとはならない結果がでました。（極端に遅いコンピューターで試しましたが、結果にはプロセッサーの構成によって変化するかもしれません。）以下が時間計測のやり方です。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(100,100);
x=x'x;
b = hsec;
y=invpd(x);                    ←ここを  y=inv(x);  に変更して計算しなおす
a = hsec;
runtime = (a-b)/100;
print runtime "sec";
```

invpd のところを inv に変更して時間を計測してみてください。ライト版ではこの程度の行列のディメンションが限界ですが、正式版では rndn の列の数を増やすことによってさらに大きな行列の逆行列がどの程度の時間を要して計算されるのかが計測できます。なお、hsec という変数は、それだけで、その時点の午前 0 時からの秒数の 100 倍が入るシステム変数です。この値を逆行列計算の前後において見れば、その実行時間がおおよそわかるはずです。（なお、この実行時間には組込み関数の計算以外の、表示したりする時間やコンピューターと GAUSS の間の受け渡しに要する時間も入っています。また inv と invpd の両方の実行時間数を１つのプログラム上に書くと、どちらかが０になってしまいます。これは、GAUSS が同じ値が入ると判断して計算をしないからです。この場合、プログラムは別々にする必要があります。）結果は、invpd の場合も inv の場合とほとんど同じか若干 invpd の方が速いだけの結果が出るはずです。ライト版が実行できる範囲内では、ほとんど差は見られないはずです。print 文では引用符内のコメントは変数の後にも置くことができますし、変数をいくつか並べた、その間にも置くことができます。変数とコメントの引用符をくっつけてしまうとエラーになるので注意してください。

ポイント　invpd(symmetric positive definite **な行列**)　inv **と同じく逆行列を計算する**

ポイント　GAUSS **の時間計測には、**hsec **という変数を使う。**

例　b=hsec;

```
(   program to be estimated in time   )
a=hsec;
time=(a-b)/100;   print time “sec”;
```

すこし行列の概念に深入りをすることにはなりますが、上の inv と invpd の計算アルゴリズムの違いにでてきた行列の分解 decomposition についての組込み関数を少し説明しましょう。GAUSS は行列のソフトですから、その分野の組込み関数は充実しています。まず Cholesky 分解について説明します。これは、行列の「平方根」のようなもので、通常は、もとの行列をＡとするとＡ＝Ｌ L’となる下三角行列 L のことを指します。GAUSS ではその変形バージョンでするＡ＝U’Uとなる上三角行列 U を計算します。なお、転置すれば両者は同じことになります。下のプログラムはその変形 Cholesky 分解を求めるものです。

プログラム

```
new;   cls;
rndseed 10000;
x=rndn(100,5);
x=x'x;
print "X" x;
print "cholesky decomposition to U" chol(x);   print "U'U" chol(x)'chol(x);
```

画面結果

X

84.112852	11.825896	-13.202456	14.535958	-13.011281
11.825896	82.444635	-8.8096043	5.7067460	-2.1801626
-13.202456	-8.8096043	83.413777	1.2143666	6.0869662
14.535958	5.7067460	1.2143666	107.71692	-12.082809
-13.011281	-2.1801626	6.0869662	-12.082809	101.26730

cholesky decomposition to U

9.1713059	1.2894451	-1.4395394	1.5849387	-1.4186945
0.00000000	8.9878788	-0.77364164	0.40755494	-0.039034112
0.00000000	0.00000000	8.9857099	0.42414565	0.44676506
0.00000000	0.00000000	0.00000000	10.240063	-0.97732308
0.00000000	0.00000000	0.00000000	0.00000000	9.9044600

U'U

84.112852	11.825896	-13.202456	14.535958	-13.011281
11.825896	82.444635	-8.8096043	5.7067460	-2.1801626
-13.202456	-8.8096043	83.413777	1.2143666	6.0869662
14.535958	5.7067460	1.2143666	107.71692	-12.082809
-13.011281	-2.1801626	6.0869662	-12.082809	101.26730

上の結果では、まず最初の X は乱数によって作ったデータ x のモーメント x'x をつくって対称行列化したものです。たいていのケースは positive definite になっているはずです。組込み関数 Chol の中には対称な positive definite 行列が入ります。さおうでなければエラーを返してプログラムは止まります。２番目の行列は chol で分解した結果です。Cholesky 分解の変形バージョンの上三角行列 U となっています。U の対角成分より下はすべて０が入っています。３番目の行列はその U を用いて U'U を計算したものです。当然のことながら、X そのものになるはずです。ゆえに、これが Cholesky 分解の行列の「平方根」と言われる由縁です。これをルーチンに使っている invpd も同様に対称な positive definite 行列しか入らないのもこのためです。なお、chol、invpd、それに行列／行列などに関連した計算には、通常の計算精度よりも高い精度が用いられます。他のプログラムとの比較から８bit からなる標準の倍数精度に戻したい場合には、

```
prcsn 64;
```

という文を上の関数計算などの前に置くことが必要になります。なお、デフォルトは 10bit の 19 桁である prcsn 80;となっています。これは、精度を鈍らせるためのみに使います。

さらに、ＬＵ分解およびその仲間の Crout 分解のプログラムを示しておきましょう。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(5,5);
print "X" x;
{L,U}=lu(x);

print "LU decomposition" L;  print U;
print "LU" L*U;
```

画面結果

X

-1.2355640	0.45194399	1.1973594	-0.33016131	-1.9308480
0.43909975	-1.4560906	-0.49176805	-0.92582422	-0.20847823
1.4762333	0.83263409	-0.71098512	0.87568040	-0.73166686
-0.50424713	0.15933167	-1.1514522	-0.76360988	0.16403621
-0.31437855	-0.062086665	0.60564869	0.24693176	-0.43010455

LU decomposition

-0.83697064	-0.67429573	-0.28166425	1.0000000	0.00000000
0.29744603	1.0000000	0.00000000	0.00000000	0.00000000
1.0000000	0.00000000	0.00000000	0.00000000	0.00000000
-0.34157684	-0.26044846	1.0000000	0.00000000	0.00000000
-0.21295993	-0.067633595	-0.29665221	-0.20118823	1.0000000

1.4762333	0.83263409	-0.71098512	0.87568040	-0.73166686
0.00000000	-1.7037543	-0.28028835	-1.1862919	0.0091531760
0.00000000	0.00000000	-1.4673089	-0.77346563	-0.083500316
0.00000000	0.00000000	0.00000000	-0.61501170	-2.5605788
0.00000000	0.00000000	0.00000000	0.00000000	-1.1252301

LU

-1.2355640	0.45194399	1.1973594	-0.33016131	-1.9308480
0.43909975	-1.4560906	-0.49176805	-0.92582422	-0.20847823
1.4762333	0.83263409	-0.71098512	0.87568040	-0.73166686
-0.50424713	0.15933167	-1.1514522	-0.76360988	0.16403621
-0.31437855	-0.062086665	0.60564869	0.24693176	-0.43010455

今度の X は対称である必要はないので、まず一般的な正方行列を乱数で直接生成しています。次の組込み関数 lu は行列のＬＵ分解を計算するものです。L が最初の行列で、U がそ

の次の行列になっています。ただし、L の方は、まだ行を入れかえる前の形になります。１列目に１がある行を１行目に、２列目に１がある行を２行目に、３列目に１があるものを３行目に、４列目に１がある行を４行目に、そして５列目に１がある行を５行目にすると上三角行列ができます。この操作によってできた L と lu によってできた U との組合せは下三角行列の対角成分がすべて１になる Doolittle 分解になります。最後に L と U をかけ合わせると X そのまのになることを確認しています。ＬＵ分解はＡｘ＝ｂのときにＬＵｘ＝ｂとすることにより、Ｌｙ＝ｂとＵｘ＝ｙと２つの問題に分割して考えることが可能になるため、ともに三角行列がかかっているだけで計算が楽になる特質があります。なお、プログラムのように、組込み関数 lu は２つの戻り値を返す関数なので、のちにプログラムのところで扱うように｛変数,変数｝という形をとります。もし、{L,U}＝のところを単変数ｙ＝などとおくと、GAUSS はエラーを返して止まります。現在のところは、２変数以上を返す組込み関数または proc は、１変数とおけないとおぼえておいてください。なお、print 文で、print lu(x);とすることには何ら問題はありません。

次に、inv のアルゴリズムにも使われている Crout 分解のプログラムについてみていきましょう。これは、Doolittle 分解で１が U の方の対角成分についていたのに対して、その１が今度は L の方につく分解です。こちらは、U と L とをまとめて１つの行列として計算して出す組込み関数 crout が用意されています。通常の意味での Crout 分解にするには、少しプログラムを書き加えなければなりません。引き続き一般的な正方行列を扱います。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(5,5);
print "X" x;
print "LU decomposition" crout(x);
print "L" lowmat(crout(x));
print "U" upmat1(crout(x));
print "LU" lowmat(crout(x))*upmat1(crout(x));
```

画面結果

```
X
     -1.2355640      0.45194399       1.1973594     -0.33016131      -1.9308480
      0.43909975     -1.4560906     -0.49176805     -0.92582422     -0.20847823
       1.4762333      0.83263409     -0.71098512      0.87568040     -0.73166686
     -0.50424713      0.15933167      -1.1514522     -0.76360988      0.16403621
     -0.31437855     -0.062086665      0.60564869      0.24693176     -0.43010455
LU decomposition
     -1.2355640     -0.36577952     -0.96907923      0.26721507       1.5627261
```

```
      0.43909975      -1.2954769      0.051136072       0.80523108       0.69061117
       1.4762333       1.3726100       0.64941208      -0.96096094      -6.1387140
      -0.50424713     -0.025111603      -1.6388235      -2.1834922       4.1634635
      -0.31437855      -0.17707990       0.31004614       0.77147091      -1.1252301
L
      -1.2355640       0.00000000       0.00000000       0.00000000       0.00000000
       0.43909975      -1.2954769       0.00000000       0.00000000       0.00000000
       1.4762333       1.3726100       0.64941208       0.00000000       0.00000000
      -0.50424713     -0.025111603      -1.6388235      -2.1834922       0.00000000
      -0.31437855      -0.17707990       0.31004614       0.77147091      -1.1252301
U
       1.0000000      -0.36577952      -0.96907923       0.26721507       1.5627261
       0.00000000       1.0000000       0.051136072       0.80523108       0.69061117
       0.00000000       0.00000000       1.0000000      -0.96096094      -6.1387140
       0.00000000       0.00000000       0.00000000       1.0000000       4.1634635
       0.00000000       0.00000000       0.00000000       0.00000000       1.0000000
LU
      -1.2355640       0.45194399       1.1973594      -0.33016131      -1.9308480
       0.43909975      -1.4560906      -0.49176805      -0.92582422      -0.20847823
       1.4762333       0.83263409      -0.71098512       0.87568040      -0.73166686
      -0.50424713       0.15933167      -1.1514522      -0.76360988       0.16403621
      -0.31437855      -0.062086665       0.60564869       0.24693176      -0.43010455
```

うえの最初の行列Ｘは一般的な正方行列。次のcroutによって分解された行列は、ＵとＬとがミックスされた形で計算されます。この場合の戻り値は１つだけです。GAUSSの組込み関数lowmatとupmat1を用いてcrout(x)でできたミックスされた行列を分けます。lowmatは行列の下三角行列を求める組込み関数。対角成分より上にはすべて０が入ります。一方、upmatは行列の上三角行列を求めるもので、対角成分より下にはすべて０が入ります。これをupmat1という組込み関数にかえると、upmatでできる上三角行列のうち対角成分をすべて１にかえる働きをします。（同様にlowmat1は下三角行列のうち対角成分をすべて１にします。）最後に、ＬとＵをかけ合わせたものを計算しています。最後の行列は、もとの行列Ｘに等しくなるはずです。

ポイント　chol(対称なpositive definite行列Ａ)　Ａ＝Ｕ'ＵとなるCholesky分解Ｕを求める

ポイント　chol、invpdなどの計算精度を通常倍精度に戻すには　prcsn 64;　と設定

ポイント	{L,U} ＝ lu(正方行列A)	A＝LUとなるLU分解LとUを求める
ポイント	crout(正方行列A)	A＝LUとなるCrout分解LとUを求める
ポイント	lowmat(行列)　upmat(行列)	行列の下三角行列、上三角行列を求める
ポイント	lowmat1(行列)　upmat1(行列)	行列の下三角行列、上三角行列を求めて対角成分をすべて１にする。

なお、GAUSSにはsingular行列に対してGeneralized Sweep inverseとMoore-Penrose pseudo-inverseの逆行列の計算関数が用意されている。どちらもY=組込み関数(X)の時Moore-Penrose条件：

1) X*Y*X = X　2) Y*X*Y = Y　3) X*Y は対称行列　4) Y*X は対称行列

を満たしている。Xの行列のところにNon-singularな通常の行列がくれば、invの計算と全く同じ答えが返されます。以下は、Non-singularな行列の場合、invもinvswpもpinvも全く同じ答えを返す例です。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(5,5);
print "inv" inv(x);
print "invswp" invswp(x);
print "pinv" pinv(x);
```

画面表示

```
inv
      0.36142273       0.17027668       0.11661124      -0.96331630      -2.2708215
      0.53473588      -0.48664197      -0.030923939     -0.41023090      -2.2685325
      0.47487806      -0.097739537     -0.35768047      -1.1113635       -1.8998676
     -0.88156881      -0.12297869       0.37681893       0.84933636       3.7000997
     -0.17879742      -0.26244903      -0.36809714      -0.31399778      -0.88870713
invswp
      0.36142273       0.17027668       0.11661124      -0.96331630      -2.2708215
      0.53473588      -0.48664197      -0.030923939     -0.41023090      -2.2685325
      0.47487806      -0.097739537     -0.35768047      -1.1113635       -1.8998676
     -0.88156881      -0.12297869       0.37681893       0.84933636       3.7000997
     -0.17879742      -0.26244903      -0.36809714      -0.31399778      -0.88870713
pinv
      0.36142273       0.17027668       0.11661124      -0.96331630      -2.2708215
      0.53473588      -0.48664197      -0.030923939     -0.41023090      -2.2685325
```

```
 0.47487806     -0.097739537     -0.35768047      -1.1113635      -1.8998676
-0.88156881      -0.12297869      0.37681893      0.84933636       3.7000997
-0.17879742      -0.26244903     -0.36809714     -0.31399778     -0.88870713
```

GAUSSでは特別なエラー操作をしないかぎり、引数に入る行列の型が違ったり、許されない性質の行列が入っただけで、そこですべてのプログラムは終了します。それ以降の計算は行なわれません。注意してください。Singularのケースもその一例です。ただし、上の２つの特殊な逆行列を求める計算を通常のregressionに用いるのは不適切です。行列がsingularになってエラーが発生した場合には、例えば、ダミー変数の場合には１つ数を減らすか、または切片をなくした形でregressionし直す必要があります。

ポイント　invswp(正方行列)　Generalized Sweep inverseをsingularな行列に行なう
ポイント　pinv(正方行列)　Moore-Penrose pseudo-inverseをsingularな行列に行なう

いままでのLU分解やCholesky分解ではもとの行列が正方行列であることが最低条件でしたが、正方行列ではない一般的な行列の分解に移りましょう。条件数condition numberを計算する関数cond、そのもととなる特異値singular valuesを求める関数svdをまずは少し説明しましょう。これまでのように行列データは乱数でつくりますが、この場合、正方行列である必要はありませんから、３×４の行列を使うことにしましょう。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(3,4);
print "x" x;
s=svd(x);
print "singular values s" s;
print "maxc(s)/minc(s) =" maxc(s)/minc(s);
print "          cond(x) =" cond(x);
```

画面表示

```
x
      -1.2355640       0.45194399        1.1973594      -0.33016131
      -1.9308480       0.43909975       -1.4560906      -0.49176805
     -0.92582422      -0.20847823        1.4762333       0.83263409
singular values s
       2.5683043
       2.4649177
```

```
        0.81099816
maxc(s)/minc(s) =          3.1668435
          cond(x) =          3.1668435
```

まず、３×４の一般的な行列を考えるため乱数で行列を決めます。そして特異値singular valuesを求める組込み関数svdをこの行列に適用します。その戻り値は、当然のことながらもともとの行列の行数と同じ行数の列ベクトルとなります。３つの数が縦にベクトルで表示されているのが、特異値singular valuesです。１列しかありませんから、maxcとmincで特異値の最大と最小がもとまります。その最大値を最小値で割った割合が、条件数condition numberとなります。この計算した値は、組込み関数condで計算した結果とも一致しています。正方行列でない一般的な行列も含めた行列についての特異値の最大値の最小値に対する割合が条件数condition numberです。一部GAUSSの書物にある対称行列についての固有値から求める方法は、条件数の特質の１つであって、定義ではありませんので注意してください。特異値は、もともとの行列が正方行列である必要はありません。

ポイント　svd(一般行列)　一般行列の特異値を求めて列ベクトルで返す
ポイント　cond(一般行列)　一般行列の特異値の最高値と最低値の割合を計算する

このほか特異値分解singular value decompositionをする組込み関数svd1も用意されていますので説明しておきましょう。これはＸ＝ｕＳｖ'ともともとの行列Ｘを右singularベクトル行列ｕと左singularベクトル行列ｖに分解するものです。Ｓには、左上隅からの正方行列と見たてられる部分の対角成分に特異値が入り、その他の正方行列と見たてられる部分にはすべて０が入ります。そして、もともとの行列の行と列の差だけ０行列の要素が伸びて入ります。もともとの行列Ｘが３×５であれば、Ｓも同じ３×５で、その左の３×３の部分には、対角成分を特異値とする対角化された行列が入り、残りの左側の３×２の部分にはゼロ行列が入ります。この場合、ｕは３×３、ｖ'は５×４になります。対角成分は特異値の大きいものから小さなものへと順に入ります。

$$X = uSv'$$

ここで、　$u'u = I, \quad v'v = I, \quad S = [\Sigma, 0]$　または　$S = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$,

$$\Sigma = diag(s_1, s_2, ..., s_n) \quad \text{ただし、} s_1 > s_2 > .... > s_n$$

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(3,5);
```

```
print "x" x;
print "singular values" svd(x);
{u,s,v}=svd1(x);
print "u" u;
print "S" s;
print "v'" v';
```

画面表示

```
x
      -1.2355640        0.45194399         1.1973594       -0.33016131        -1.9308480
       0.43909975       -1.4560906        -0.49176805      -0.92582422       -0.20847823
       1.4762333         0.83263409       -0.71098512       0.87568040       -0.73166686
singular values
       2.7586999
       2.2444599
       1.5777679
u
      -0.93072404       -0.17298627       -0.32222433
       0.17471039        0.56372324       -0.80727467
       0.32129278       -0.80764588       -0.49444827
S
       2.7586999         0.00000000        0.00000000        0.00000000        0.00000000
       0.00000000        2.2444599         0.00000000        0.00000000        0.00000000
       0.00000000        0.00000000        1.5777679         0.00000000        0.00000000
v'
       0.61659026       -0.14771813       -0.51791155        0.15474236        0.55300835
      -0.32569415       -0.70016210        0.040043663      -0.52219019        0.35973639
      -0.43496028        0.39178339        0.22989372        0.26670704        0.73029553
      -0.56969230       -0.058721308      -0.75896381        0.26238718       -0.16471001
       0.00000000       -0.57533466        0.31825609        0.75059479       -0.065654823
```

上のプログラムは、まず３×５の一般的な行列を乱数で作り、それをＸとします。次に、その特異値を求めます。そして、３つの戻り値をとる組込み関数svd1を用いて、特異値分解singular value decompositionを行ないます。svdはその略です。その３つの戻り値をそれぞれｕ、ｓ、ｖとおいて画面表示させています。Sはうまく対角化されて、その対角成分には上で求めた特異値が入っていることがわかります。さらに左特異ベクトルｕと右特異ベクトルｖがその条件を満たすか以下の簡単なプログラムで確かめてみます。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(3,5);
{u,s,v}=svd1(x);
print "u'u" u'u;
print "v'v" v'v;
```

画面表示

```
u'u
       1.0000000  -1.1102230e-016  -5.5511151e-017
 -1.1102230e-016         1.0000000  -5.5511151e-016
 -5.5511151e-017  -5.5511151e-016         1.0000000
v'v
       1.0000000  -1.9428903e-016  -2.7755576e-016   2.0816682e-016  -1.3877788e-017
 -1.9428903e-016         1.0000000   5.5511151e-017   4.1633363e-017   4.1286419e-016
 -2.7755576e-016   5.5511151e-017         1.0000000  -8.3266727e-017  -1.7347235e-016
  2.0816682e-016   4.1633363e-017  -8.3266727e-017         1.0000000  -5.8980598e-017
 -1.3877788e-017   4.1286419e-016  -1.7347235e-016  -5.8980598e-017         1.0000000
```

上のように、完全にはＩにはなりませんが、10のマイナス16乗とか17乗とかいう値ですから、off diagonalはおよそ０にすべてなっていることがわかります。これは、小数の計算精度によるズレによるものです。なお転置行列をあとからかけても先にかけても同じです。

　そのほか、直交三角QR分解をするためのqqrをはじめとする関数群や、直交行列をもとめるorth、右零空間に対する直交基底を求めるnullがあります。

まずは、ＱＲ分解のプログラムと、その性質をチェックしてみます。

プログラム

```
new;  cls;
rndseed 10000;
x=rndn(5,4);
{q1,r}=qqr(x);
print "Q" q1; print "R" r;
print "Q'Q" q1'q1;
```

画面表示

```
Q
     -0.47938969       0.055115571       -0.48492107        0.72653710
     -0.74915476       -0.10671116        0.51836320       -0.10622759
```

```
      -0.35921296       -0.35362082       -0.57872225       -0.61306610
      -0.27585697        0.47607221        0.25759110       -0.11582491
      0.061819510       -0.79616742        0.30802360        0.26757072
R
       2.5773687       -0.78346763        0.14118275        0.37683407
      0.00000000         1.3854104       -0.041017519       -0.63081497
      0.00000000        0.00000000         -2.6134184       -0.65603871
      0.00000000        0.00000000        0.00000000       -0.59579917
Q'Q
       1.0000000    1.3877788e-017   -7.2858386e-017   -2.4286129e-017
   1.3877788e-017         1.0000000    4.1633363e-016   -1.6653345e-016
  -7.2858386e-017    4.1633363e-016         1.0000000   -8.3266727e-017
  -2.4286129e-017   -1.6653345e-016   -8.3266727e-017         1.0000000
```

ＱＲ分解は、直交行列Ｑと上三角行列Ｒに分解するもので、Ｑ’Ｑ＝Ｉが成り立ちます。
組込み関数qqrは、２つの戻り値を返しますから、{ }の印を用いて、その中に２変数を書いて操作することになります。繰り返しになりますが、２変数以上の戻り値のある組込み関数にイコールのサインをつけて１変数に代入するような通常の操作はできないことに注意してください。ここでは、最初の戻り値をq1、2番目の戻り値をrとしています。GAUSSでは大文字と小文字の区別はありませんから、大文字で戻り値を書いてももちろん結構です。それぞれの戻り値を画面出力した後、さらに、最初の戻り値である行列q1がq1’q1＝Iとなっていることを確かめるため、その値の行列を画面表示させています。ご覧のとおり対角成分はすべて１、off diagonalは10のマイナス16乗から17乗の数で、おおよそ０になっていることがわかります。これは、以前と同様、どの統計言語にも存在する小数の計算精度によるもので、避けて通ることのできないものです。気になるようでしたら、print文またはformatで、見かけ上の表示桁数をかえて０にすることも可能です。なお、組込み関数qrはRだけを返す組込み関数で、Qのところは省略されます。そのほか、QR分解の関数群は求める行列によって各種用意されています。

　組込み関数orthは、もとともと行列をXとすると、y'y＝I(Xのランク数)となる戻り値ｙを求める直交行列関数です。

プログラム

```
        new;  cls;
    x={1 2 3 4,
       2 4 6 8,
       9 3 1 2,
       8 7 2 5};
```

```
y=orth(x);
print y; print y'y;
print;
print "rank(x)=" rank(x);
```

画面表示

```
y
    -0.081649658       -0.42065325       -0.12800071
     -0.16329932       -0.84130649       -0.25600141
     -0.73484692        0.32395135       -0.59586536
     -0.65319726       -0.10153699        0.75034897
y'y
       1.0000000    4.1633363e-017    5.5511151e-017
  4.1633363e-017        1.0000000        0.00000000
  5.5511151e-017       0.00000000         1.0000000


rank(x)=          3.0000000
```

うえのようにあきらかに2行目が１行目の倍数になっているようなランク３の４×４の行列をＸとすると、その直交行列ｙはorth(x)として得られます。それを画面表示した上、さらに、ｙ’ｙを計算して画面表示させています。最後はもともとの行列Ｘのランクの計算で３となっています。ｙ’ｙは、４×４ではなくて、Ｘのランクの数の３×３で、対角成分がすべて１、off diagonalがおおよそ０のＩになっています。直交行列を求める関数orthには対角成分の有効桁数を決定するグローバル変数_orthtolが用意されています。デフォルトは_orthtol=1.0e-14ですが、これをより小さい桁数に精度を下げる変更することも可能です。ただし、orth関数の外部の験算をしてＩにする計算には適用されません。

今度は、右零空間に対する直交基底を求める組込み関数nullについての説明です。まずこれは右に対するものですから、もともとの行列の行数は必ず列数よりも小さくなくてはなりません。戻り値の直交基底は、もともとの行列の行数と同じ行数で、列数は、もともとの行列の(列数－行数)と同じ値になります。したがって、２×４とか３×４とか、必ず行数の方が小さくなければ、戻り値はドットのミッシングバリューで返されます。なお、

x*b＝０

b’ｂ＝I

の関係が戻り値ｂについて満たされるはずです。

プログラム

```
new;  cls;
```

```
rndseed 10000;
x=rndn(2,5);
b=null(x);
print "b" b;
print "x*b" x*b;
print "b'b" b'b;
```

画面表示

```
b
       0.45004365       -0.13543463       -0.73643409
      -0.17938377       -0.53453277       -0.26175209
       0.85195411      0.0018121253        0.20192936
   -0.0016856017        0.81528757       -0.17200408
       0.19862911       -0.17673446        0.56461474
x*b
  1.1102230e-016        0.00000000  -2.2204460e-016
  4.8572257e-017  -2.4980018e-016  -2.0816682e-016
b'b
       1.0000000  -6.9388939e-018  -4.1633363e-017
 -6.9388939e-018        1.0000000  -2.7755576e-017
 -4.1633363e-017  -2.7755576e-017        1.0000000
```

以上のように、最初の行列が null(x)によって計算された右零空間に対する直交基底で、その次に、実際にｘとｂをかけあわせて零行列になることを確認しています。最後の行列の結果は、b'b が I になることを計算で確認して画面表示しています。

そのほかに、行列のReduced Row Echelon形を計算する関数rref(x)、Schur形を計算する関数{s, z} = schur(x)、それに行列の行と列とをバランスする関数{b, z} = balance(x)などがGAUSSには用意されています。

ポイント　{u,s,v}=svd1(一般行列Ａ);　Ａ＝uSv'となる特異値分解を計算する
ポイント　{q1,r}=qqr(一般行列Ａ);　　Ａの直交三角ＱＲ分解を計算する
ポイント　orth(一般行列Ａ);　　Ａの直交行列を計算する
ポイント　null(一般行列Ａ);　　Ａの右零空間に対する直交基底を計算する