0.2 ジュニア版　　行列について

ここでは、GAUSS で扱われる行列のしくみと簡単な計算法について、一般の数値、ベクトル、それに行列の違いが、GAUSS ではどうなるのかを初歩から説明していきます。

変数の表示、メッセージの表示

変数の内容を設定したり、それを組み合わせて計算をする前に、それらを画面上に結果表示させる方法を先に簡単に取り上げておきましょう。下のプログラムでは、a=100 を設定してから、そのａを print する（ここでは、画面表示する）こと、それとは別に、Japan という文字列（文字からできた一続きのメッセージを表示すること、そして最後に、この２つを組み合わせて、a=という文字列と変数ａの内容を同時に表示すること、以上の３つのことを１つのプログラムで行なってみましょう。

プログラム

```
new;
cls;
a=100;

print a;
print "Japan";
print "a=" a;
```

プログラム各行の説明

１行目　メモリを初期化する
２行目　スクリーン（画面）をクリアする
３行目　変数ａに 100 を代入する
４行目　空白
５行目　変数ａの内容を print という命令で画面表示する
６行目　引用符の内側の Japan という文字列を print という命令で画面表示する
７行目　引用符の内側の a=という文字列と変数ａの内容を同時に画面表示する

画面表示

```
      100.00000
Japan
a=         100.00000
```

結果は、100 という数字に０が GAUSS の標準設定での桁数いっぱいいっぱいまでたくさんついて表示されます。注意すべきは、どんなプログラムにも new;を冒頭につけること（できれば cls もつけた方がよい）、それから GAUSS には行の概念がなくて、１つ１つのプログラムの命令や変数設定の最後には必ずセミコロン「；」をつけなくてはならないということです。したがって、見やすいように（見かけ上の）４行目のように、空白にしようとダブルスペースですべてのプログラムを書こうと、プログラムの動作には何も影響を及ぼしません。ただし、７行目の”a=”と a の間にはスペースを入れることが必須です。ここにスペースがないと、「Operator missing」のエラーが表示され、プログラムはそこで停止します。なお、GAUSS 上には日本語でメッセージを書いてはいけません。プログラムは万国共通の共通言語です。英語で注釈を入れるように心がけましょう。

注意事項

☆プログラムの最初には「new;」を必ずつける
☆各変数設定および各命令の最後には１つ１つセミコロン「；」をつける
☆変数の内容の表示　　print 変数;
☆メッセージの表示　　print “文字列”;
☆GAUSS 上に日本語を書いてはいけない！

注釈の書き方

プログラムには、実行されない部分があります。それは注釈と呼ばれる部分です。実行はされませんが、読み手である人間に対して、それがどのようなプログラムであるのか、また、その行（またはそのブロック）が何を示しているのかを示す重要なコミュニケーション手段になります。これらを使いこなすこともプレゼンテーション上、重要になります。

プログラム

```
/* This program shows how to comment program. */

/*
**      Example 0.2.2
**      Comment
*/
new;
cls;
a=1/20000000;
print a;      @ GAUSS cannot display any rational fraction. @
```

画面表示

```
5.0000000e-008
```

プログラムのやろうとしていることはおおよそわかると思います。プログラムの実行とは関係のないコメントをつけてプログラムを見やすくするためには、３つのやり方があります。それは、

１）　/*　　*/

２）　/*
　　**
　　**
　　*/

３）　@　　@

オーソドックスなコメントのつけかたは、１）の/*　*/ の間にコメントを書くものです。何行かにわたるコメントまたは冒頭部での著作や使用上の注意事項をまとめて何行にもわたって記述するには、その変形である２）の最初と最後は/*　*/で、その間は２つのアステリスクで対応します。この２つのアステリスクはなくても GAUSS には行の概念はありませんから、注釈部分が何行にわたろうともどれだけの空白が入ろうと正常に機能します。もう１つのやり方は、@　@の間にコメントを書くものです。こちらは慣習上、すでに何か式や命令が書かれている行にさらに加えて、その行についてコメントをつける際に多用されます。ただし、/*　*/も@　@も、基本的には、同じものなので見やすいようにすればよいでしょう。繰り返しますが、コメントにおいても GAUSS は一切日本語不可です。

　コメントのつけ方からは離れますが、上の画面表示のところにはｅがついた指数表示で結果が表示されています。これは、一般にコンピュータプログラムでよく用いられる表現で、GAUSS においても定められた桁数の中にその数が小数で書けないほど桁数がある場合には、この表示方法に自動的に変わります。上の 5.0000000e-008 というのは、つまり、

　5.0000000e-008　＝　5.0000000×10^{-8}

という意味です。ｅは「10 の指数乗」を表すという表現です。もう少しわかりやすく言うならば、10^{-8} は 1/100000000（０が８つ）ですから、

　5.0000000e-001　＝　5.0000000×10^{-1}　＝5/10　=0.5
　5.0000000e-002　＝　5.0000000×10^{-2}　＝5/100　=0.05
　5.0000000e-003　＝　5.0000000×10^{-3}　＝5/1000 =0.005
　　・・・・・・・・・・・・・・・・・・・・
　5.0000000e-008　＝　5.0000000×10^{-8}　＝5/100000000=0.00000005

ということになります。反対に、桁数におさまらない大きすぎる数に対しては、

　5.12000000e+011　＝　5.1200000×10^{11}　＝5120000000

ということになります。すなわちｅのところを「10 の何乗倍」と考えればよいでしょう。

注意事項

☆コメントは/*　*/または@　@の中に書き入れる。必ず英語で書くこと。日本語不可。

☆@のマークは@@@@@@@@@@@@@などいうふうに縦または横にコメントの枠にするなど重ねることはできない。挙動がおかしくなる原因となる。

☆桁数におさまりきれない小さすぎる数または大きすぎる数は指数表現ｅを使うことによって「10 の何乗倍」という形で表現されることもある。

☆GAUSS は計量計算の言語なので、有理数（分数表現）で表示させることは不可能。

スケラー（スカラー)、ベクトル、行列の３種類

話を簡単にするために、普通の数えられる数「実数」を考えるとすると、そのディメンションから「実数」を大別すると数学的にはスケラー、ベクトル、行列に分けられます。

プログラム

```
new;
cls;
a=100;
b={1,0,0};
c={1 0 0};
d={1 0, 0 0};
print a;
print b;
print c;
print d;
```

プログラム各行の説明

１行目　メモリを初期化する

２行目　スクリーン（画面）をクリアする

３行目　変数ａにスケラー100 を代入する

４行目　変数ｂに３×１の列ベクトルのそれぞれの要素を代入する（カンマまでが１行）

５行目　変数ｃに１×３の行ベクトルのそれぞれの要素を代入する

６行目　変数ｄに２×２の行列のそれぞれの要素を代入する

７行目　変数ａの内容を画面表示する

８行目　変数ｂの内容を画面表示する

９行目　変数ｃの内容を画面表示する

10 行目 変数ｄの内容を画面表示する

画面表示

```
      100.00000

      1.0000000
     0.00000000
     0.00000000
      1.0000000      0.00000000      0.00000000

      1.0000000      0.00000000
     0.00000000      0.00000000
```

上のように通常の１個の要素（成分）からなるスケラー（スカラー）だけでなくて、列ベクトルも行ベクトルも、それから行列も、GAUSS 上ではすべてみんな等しく扱われることがわかると思います。ほかの言語でプログラムをしたことがある人なら、行列は配列を宣言しなければいけないとか、ベクトルとスケラーの扱いはそれぞれ宣言のしかたが違って異なる扱いをすると理解しているかもしれません。しかしながら、GAUSS ではそういった変数のタイプによる宣言をする必要は全くありません。実は、スケラーも含めてすべてのタイプの変数が行列として扱われているのです。スケラーも１×１の行列として扱われています。したがって、宣言やタイプごとの使い分けはまったく必要がありません。それが GAUSS の本質なのです。

上で注意すべきは、{1,0,0} と表現される場合、通常使われる数学的意味では「行ベクトル」と横１列のものと考えられますが、GAUSS ではこれを「列ベクトル」と縦１列のものとします。カンマまでが１行であって、要素間に２つのカンマがあるので３×１の行列、すなわち列ベクトルとなります。反対にカンマをつけずに{1 0 0}と並べると GAUSS では「行ベクトル」と考えられます。なお、d={1 0, 0 0};は見やすいように

```
d={1 0,
   0 0};
```

と見た目上しばしば複数行にわたって書かれることもあります。GAUSS プログラムでは「行の概念がない」ので、１行にまとめて書こうと、実際の見え方と同じようにカンマごとに複数行にわたって書こうと、同じことなのです。GAUSS ではすべての計算は行列として処理されます。ベクトルもスケラーも「行数×列数」のどちらか一方または両方が１の特殊な場合の行列に含まれると考えることができます。

ベクトルって、行列って何なんだろう？

上で取り上げたスケラー（スカラー）とベクトルと行列について、なんとなく違いがわかったでしょうか。また、GAUSS ではそれらが行列として分け隔てなく扱われるのだとい

うことをわかってもらえたでしょうか。では、ベクトルやそれが束になっているイメージの行列がなぜ必要なのでしょう。１＋２は３で、３＋３は６で、６＋４は 10 で、それをくりかえして 10 まで足し合わせると 55 になります。何もベクトルや行列にする必要なんてないじゃないかと思われるかもしれません。確かに、代数や簡単な数学ではそうかもしれません。スケラー計算でたいていのものは対応できるでしょう。しかしながら、統計学、特にその中でも計量経済学のデータは、それぞれの系列が縦に「列として」並んでいて、「列が束になって行列」になっています。例えば、次のようなラベルの部分を除いて５×３の

		1 列目	2 列目	3 列目	
	行→				
列		Y	L	K	（ ラベル行 ）
↓	１行目	100	3	4	
	２行目	102	4	4	
	３行目	110	5	3	
	４行目	112	4	5	
	５行目	115	5	5	

というような行列があったとしましょう。実際の経済や金融のデータは「縦方向にに」「列として」並んでいて、その「列が束になって行列」になっています。上のように手でいちいち勘定できる縦横のディメンションなら、代数のように１つ１つ手で計算するとか、逐次的にその作業をくりかえさせて計算するとかできるでしょう。しかし、実際のデータは縦の系列の列数は数えるだけしかないかもしれませんが、横のそれぞれの行の数はたいへんな数になるはずです。人口統計関連などのデータでは数万行がごく普通になります。そのようなケースにも、代数などと同じように逐次計算で計算しますか？答えはノーです。

１行目	100	3	4
２行目	102	4	4
３行目	110	5	3
４行目	112	4	5
５行目	115	5	5
・ ・ ・ ・ ・ ・ ・ ・			
20000 行目	985	186	192

上のような 2 万行３列の「行列データ」が与えられて利用できる状態になっている場合、私たちは代数的な逐次計算を放棄して、１つのまとまりとして「列として」、またそれが束

なった「列を水平方向に束ねた行列として」計算していくことが最も効率がよい計算方法になります。実際、Y、K、Lのそれぞれの系列について平均と分散、それに共分散を求めるにはどうしたらよいでしょうか？また、$Y=\alpha+\beta_1 K+\beta_2 L+u$ を推定するのに代数的に、初等計量計量経済学で習うであろうΣがいくつも並んだ式を計算しますか？また、その「シグマでできた公式」を一生覚えておけますか？答えはノーです。おそらく、偉い計量学者であっても、授業の前になれば予習をして思い出すかもしれませんが、そんな「代数的な」１つの要素ごとに足し合わせていくような複雑な公式は使いませんし、一生覚えておくことなどできない相談ですし、またそうすることははっきり言って無駄です。ではどうしているのか？それはすべて「列ごとに」系列として計算できるものは、列として計算してしまうのです。また、線形回帰係数をもとめるには、上のKとLの列の前に１ばかりでできた列を加えて 20000×３のディメンションのXという行列を作成したうえで、αといくつかあるβをまとめて列ベクトルのβと考えて、$\beta=(X'X)^{-1}X'Y$ という行列でできた式をまとめて計算してしまえばよいのです。研究者も実務家もみんなこの公式を覚えているのであって、初等計量経済学の教科書に載っているシグマからできた代数式を覚えているわけではありません。また、コマンドを使って自動的に線形回帰係数やそれに付随する統計量を求めてくれるパッケージソフトも、シグマの代数式を内部に使っているわけではなくて、たいていの場合、「行列でできた式」を「行列として」内部で解いているのです。ですから、私たちも「行列計算」から始めることが肝要になります。

もう一度上のデータの図を見てください。列とはなんでしょうか？それは、それぞれ異なった種類の系列のデータのそれぞれを表しています。上では、１列目がYで、２列目がL、それに３列目はKになっています。それぞれ種類の違うものです。行列とは何でしょう？それは「列が水平方向に集まったもの」と考えることができます。「行列」というのはまた５行１列目が 115 であるというように、「行」それに「列」の順でその位置を示しています。常に「行」そして「列」の順番です。それではなぜ「列」そして「行」の順番で呼ばないのでしょうか？それは欧米の書物が右側から左側に「行ごとに書く」言語であってそのおのおの行の中で何番目か、すなわち「その何列目なのか」と考えるのが普通であったからです。そのような慣習上の理由から左隅を起点に、どこの「行」そしてその何番目の「列」かという順番で位置を見つけ出します。常に行列で考えるとものごとは、簡単にすっきりとした形で計算ができ、また覚えやすいという性質をもっています。そのことは経済や金融のデータに特にあてはまります。そういった観点から、行列データを「列ごとに扱って」「すべての変数が始めから行列を仮定している」GAUSS が計量経済学の分野の計算にもっとも自然な形でマッチしている言語と言えます。私たちは、これから、GAUSS が「列の集まりとしてのデータ行列」をどのように扱うのかを見ていきます。

数値計算をしてみよう

ここでは簡単な「足す」「引く」「かける」「割る」といった四則演算を GAUSS の上で行なってみましょう。まずは、一番簡単な１×１のディメンションの行列、つまり、スケラーからです。以下のプログラムでは、ｘに１をｙに２をそれぞれ代入しておいて、それっらの和差積商を求めます。

プログラム

```
new;
cls;
x=1;
y=2;
print x+y;
print x-y;
print x*y;
print x/y;
```

画面表示

```
  3.0000000
 -1.0000000
  2.0000000
 0.50000000
```

計算は至って簡単でしょう。冒頭で new;でメモリを初期化して、cls;でそれまでの画面をクリアしておくことを忘れないこと、そして、それぞれの行の最後のところにセミコロン「;」を忘れさえしなければプログラムは簡単に動きます。新しいことと言えば、print の画面表示命令には変数そのものだけでなく、変数を含んだ計算式も書けるということくらいです。上のように書く場合、計算式と変数の間にはスペースは１つも入れてはいけません。スペースを入れてしまうと１つの変数のまとまりとして GAUSS が認識できなくなるためです。

次に同じことを列ベクトルについてもやっておきましょう。

プログラム

```
new;
cls;
x={1,2,3,4,5,7,9};
y={2,4,5,8,9,11,13};
print x+y;
print x-y;
print x.*y;
print x./y;
```

画面表示

3.0000000
6.0000000
8.0000000
12.000000
14.000000
18.000000
22.000000

-1.0000000
-2.0000000
-2.0000000
-4.0000000
-4.0000000
-4.0000000
-4.0000000

2.0000000
8.0000000
15.000000
32.000000
45.000000
77.000000
117.00000

0.50000000
0.50000000
0.60000000
0.50000000
0.55555556
0.63636364
0.69230769

１×１の行列であるスケラーではなくて、上の計算ではベクトルを扱っています。通常ベクトルの和差は、ディメンションが同じである限り計算できます。上では、ｘとｙのそれぞれが７×１の列ベクトルですから（カンマまでが１行であることを思い出してください）

それらの和と差は、それぞれの同じ行の要素に注目して足し算引き算するだけです。つまり、１行目に相当する１と２を足して３が和のベクトルの第 1 要素になって、次に２行目の２と４を足してその第２要素になって、以下７行目までが計算されます。引き算も同じことです。けれども掛け算はどうでしょうか？GAUSS ではすべての変数が「行列として」扱われると述べました。行列では掛け算の際に重要なきまりがあります。それは、

```
   X   ＊   Y          →       Z
7×1      7×1                  7×1？
  ↑___↑
```

行列同士の掛け算の場合、前の行列の列の数と後ろの行列の行の数が一致している必要があるということです。そうではない場合には、これら２つの行列をかけ合わせることは不可能です。上の場合、１と７は明らかに違います。したがって、行列としては掛け合わすことはできないことになります。できないなら計算しないか、または、ベクトルの和や差のように各要素ごとの計算をすると新たに約束ごとをするならば計算可能です。それが「要素対要素の積」の演算記号である「.*」です。ここでドットは「要素対要素」を意味します。同様に割り算においても、各要素ごとの割り算と新たに約束ごとをして「要素対要素の割り算」の演算記号である「./」を使います。ここでも割り算のスラッシュの前についているドットは「要素対要素」ということを意味します。こうでもしなければ、そもそも行列の割り算自体が意味をなさないか、存在しないことになります。ただし、ｘ＊ｙと違って、ｘ/ｙは計算できてしまいます。仮にプログラム上の割り算のスラッシュの記号の前についているドットを取り去って計算したとしましょう。これは一体何なんでしょうか？それは、ｘ＝βｙという定数項なしの線形回帰をやって、両辺に１/ｙをかけたもの、すなわち、ｘ/ｙ＝βに相当します。もし、ｙ/ｘとすれば通常の定数項なしの$(x'x)^{-1}x'y$と全く同じ結果が得られます。それを確かめるには、上のプログラムに加えて、

```
print x/y;
print inv(y'y)*y'x;
```

としてみてください。両者の答えは完全に一致するはずです。GAUSS ではこのように割り算は、割る変数がｙであるとすると、両辺に 1/ｙをかけた計算結果を得ているのと同じになります。ですから、根本的に、私たちの考える割り算とは違います。

今度は、通常の行列と呼ばれるものを扱ってみましょう。

プログラム

```
new;
cls;
x={1 2,
```

```
    3 4};
y={2 4,
    6 8};
print x+y;
print x-y;
print x*y;
print x/y;
```

画面表示

```
    3.0000000       6.0000000
    9.0000000       12.000000

   -1.0000000      -2.0000000
   -3.0000000      -4.0000000

    14.000000       20.000000
    30.000000       44.000000

   0.50000000      0.00000000
   0.00000000      0.50000000
```

上のｘもｙも同じ２×２のディメンションです。足し算と引き算は、これまでと全く同様に要素ごとの和と差をとります。掛け算はどうでしょうか？

```
   X  ＊  Y          →      Z
２×２    ２×２              ２×２
   ↑___↑
```

どちらの行列も２×２（２行２列）ですから、上の赤字の矢印のところの前の行列の列数と後ろの行列の行数が一致していて、計算が可能です。実際のところ、

$$\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}\begin{pmatrix}2 & 4\\ 6 & 8\end{pmatrix}=\begin{pmatrix}1*2+2*6 & 1*4+2*8\\ 3*2+4*6 & 3*4+4*8\end{pmatrix}=\begin{pmatrix}14 & 20\\ 30 & 44\end{pmatrix}$$

となります。上のようにＸの１行目とＹの１列目の要素をかけて足し合わせたものを １ 行１列目に、Ｘの１行目とＹの２列目の要素をかけて足し合わせたものを１行２列目に、Ｘの２行目とＹの１列目の要素をかけて足し合わせたものを２行１列目に、最後にＸの２行目とＹの２列目の要素をかけて足し合わせたものを２行２列目の答えにして、もともとの

行列のうち、前の行列の今度は外側に相当する行数と後ろの行列の列数が、新しい行列のディメンションになります。すなわち、２つの行列の配列ディメンションの内側は一致している必要があり、行列の積が計算できるかの Comformable であるかのチェックに、外側のディメンションは新しい計算された行列のディメンションになります。例えば、

```
    X   ＊   Y          →       Z
４×３      ３×２                ４×２
    ↑___↑
```

のような行と列の長さの異なる行列同士でも、その配列ディメンションの内側の数、この場合３と３が一致して comformable でありさえすれば行列の積は計算で、結果新たに計算される行列の配列ディメンションは外側の数の４×２になります。この確認は、行列の計算において極めて基本の基本になる事項であって、高度な行列計算や計量経済学の理論計算に至ってもごく一般的にチェックすべきことになります。

割り算の方はどうでしょう。上のプログラムではドットをスラッシュの前につけないスラッシュだけの演算子にしておきました。これはどういう意味でしょう？

```
b={0.5 0,
   0 0.5};
print b*y;
```

プログラムにさらに上の３行を加えてみてください（実際には、セミコロンは２つしかありませんから、GAUSS の計算上は２行ということになります）。

```
1.0000000      2.0000000
3.0000000      4.0000000
```

ともとのＸ行列と同じ内容が答えとして出てくるはずです。つまり、

Ｘ＝β Ｙ

という関係式において、両辺に 1/Ｙをかけたもの、すなわち、

Ｘ/Ｙ＝β

となる２×２のディメンションのβを求めていることになります。通常の要素対要素の割り算の結果がほしいのなら、「./」というふうに必ずドットをつけなくてはなりません。

逆行列とは

もう説明なしに行列表現の線形回帰の公式に使ってしまいましたが、逆行列とは一体何なんでしょうか？今までの説明で両辺に 1/ｙをかけるという表現は厳密にいって適切ではありませんでした。正確には、行の数と列の数が同じ正方行列同士の計算の場合、Ｉ/ｙをかけるといった方が正しいでしょう。すなわち、Ｉというのは、例えば２×２のケースで

$$I=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

というような斜めの「対角要素」（ダイアゴナル）と呼ばれるところがすべて１で、その他がすべて０であるような行の数と列の数が同じ「正方行列」（スクエア行列）のことを言います。「単位行列」（アイデンティティー行列）と呼びます。

プログラム

```
new;
cls;
x={1 2,
   3 4};
y={2 4,
   6 8};
print x/y;
print x*inv(y);


I={1 0,
   0 1};
print inv(y);
print I/y;
   print y*inv(y);
```

画面表示

```
 0.50000000          0.00000000
 0.00000000          0.50000000

 0.50000000          0.00000000
 0.00000000          0.50000000

-1.0000000           0.50000000
 0.75000000         -0.25000000

-1.0000000           0.50000000
 0.75000000         -0.25000000

  1.0000000          0.00000000
 0.00000000           1.0000000
```

つまり、上の行列の「割り算」なるものというプログラムを実行してわかるように、x/y と $x*y^{-1}$ は同じことだと言うことです（ここで、inv(y)は行列ｙのインバース、すなわち逆行列であるとします。何か逆行列を計算する関数であると仮に考えておいてください)。プログラムの後半に示したように、、このことはまた、ｙの逆行列 y^{-1} と I/y が等しいことも意味しています。最後の結果が示すように、行列Ｙの逆行列とは、

$Y*Y^{-1}=I$（単位行列）

となるような行列、すなわち、「その行列にかけ合わせて単位行列を作り出すような行列」になります。Y^{-1} のところをＺなどの他の行列と考えてみてください。「Ｙ＊Ｚ＝ＩとなるＺのことをＹの逆行列」と呼びます。

転置と行列式とは何か

前述の行列形式での線形回帰の公式でててきたプライム「'」とは何なのでしょうか？また、よく行列でててくる行列式（デターミナント）とは何なんでしょうか？

プログラム

```
new;
cls;
x={1 2,
   3 4};
print x';
print;
print det(x);
print;
print det(x');
```

プログラム各行の説明

１行目　メモリを初期化する
２行目　スクリーン（画面）をクリアする
３～４行目　変数ｘに２行２列の数値を代入する。
５行目　変数ｘの内容を転置したものを画面表示する。
６行目　改行する(print;だけで改行を行なう)。
７行目　変数ｘの行列式 det(x)を計算して、その内容を画面表示する。
８行目　改行する。
９行目　変数ｘを転置したものの行列式を計算して、その内容を画面表示する。

画面表示

```
 1.0000000     3.0000000
 2.0000000     4.0000000

-2.0000000

-2.0000000
```

上の結果からわかるように、プライム「'」を行列につけると「行と列が逆転する」というものです。前述した行列の積の計算の際に、２つの行列の内側のディメンションを一致させるためにであるとか、行で答えがほしいのに列が出てくる際に、行と列を逆転させるためであるとかのケースに用いられます。数学的には|Ｘ｜と表現される行列式デターミナントとは何なんでしょうか？説明は難しいですが、２×２の行列のケースでは複数のベクトルからなる面積のようなもの、３×３の行列のケースでは複数のベクトルからなる体積のようなものであって、マイナスの場合には、通常のベクトルのとり方が時計と反対まわりであるのに対して、ベクトルが逆転して時計回りに展開しているような状態にあります。すなわち、厳密な表現ではありませんが、その行列の反時計回りを基準にしたような各ベクトルで構成される「大きさのような目安」と考えても差し支えはないでしょう。GAUSS では det(X)によって簡単に扱えます。しかも何十×何十の行列になろうと高速に計算します。上のプログラムでは、転置した行列の行列式も元の行列の行列式も同じであることがわかると思います。この場合、マイナスがつきますが、これを２次元ですから何かよじれた面積であると考えれば、そこが転置してもベクトル関係は行と列が置き換わるだけで何も変わりませんから同じであるということです。計算としては、たすきがけをして掛け算をして、反対向きにかけたものはマイナスをつけるという計算をして出します。上の例では、１＊４－２＊３＝－２が det(Ｘ)になっています。それ以上のディメンションについても実際にどうやるのか自分でしらべてみて、GAUSS の計算と手計算が一致するのかを確かめてみてください。

注意事項

☆要素対要素の掛け算または割り算の場合には、演算記号＊または/の前にドットをつける
☆inv(X)は X の逆行列（インバース）を求める
☆X'は X の転置行列（トランスポーズ）を求める
☆det(X)は X の行列式（デターミナント）を求める
☆GAUSS において大文字小文字の区別は全くない（慣習上小文字で書いていく）

発展

⇒　本編　2.4　2.5