3.2 プログラミング Loop ver. 0.3

プログラミングとコンピューターを操作することの根本的な違いというのは、どの計量統計ソフトを使おうと、ループや条件分岐をともなったプログラムを書くかどうかです。ここでは、プログラミングの初学者がつまずいたり、本当のことをわからないまま他人のアルゴリズムをコピー&ペーストしたりしてごまかしている事柄を詳説します。

まずは、erudite という単語を 10 回画面表示するプログラムをいろいろな方法で書いてみます。まずは、最も簡易な for ループの手法で書きます。

プログラム

```
new; cls;
for i (1,10,1);
    print “erudite”;
endfor;
```

この方法は、for から endfor の間のプログラムを、カウンタ i を 1 から 10 まで 1 ずつ増やすことによって 10 回実行させるものです。カウンタとは、１回、２回、３回...とプログラムがループの区間（ここでは for と endfor のあいだ）を通過するにつれて変化するカウントするものと考えてください。ここで少し変更を加えて、for i (0,10,1);とすると、ループを通過するあいだ、このカウンタは０から 10 まで 1 ずつ増えます。つまり、11 回ループの区間を通過しますから、erudite という文字を 11 回画面表示することになります。もし、ここで for i (10,1,-1);とすると、今度は、プログラムがループの区間を通過するあいだ、カウンタは 10 から 1 まで－1 ずつ増える、つまり 1 ずつ減ります。この場合も、10 回文字を表示することになります。現実的には、１からはじめてある数まで（例えば、考えている行列の行数まで）１ずつ増やすというのが最も多く使われます。

そうではなくて、オーソドックスに do while を使って上と同じことをさせることの方がほかのプログラム言語を知っておられる方にはポピュラーでしょうか。

プログラム

```
new; cls;
i=1;
do while i<=10;
    print "erudite";
    i=i+1;
endo;
```

この場合は、少し複雑で、do から endo の間を、 i を最初に 1 としておいたうえで、 i が 10 以下のあいだ、ループの区間を実行させます。ループの区間内の最後には、i=i+1;として毎回 i に 1 ずつ足されて増えて i に代入される行を書かなくてはなりません。また、do ループの外側で、i=1;という具合に、カウンタ i の値が何から始まるのか初期化しなければ

なりません。do while の後の i<=10 は、行列のところの論理でやりました 1 か 0 の論理判定式です。この場合、カウンタ i の初期値はループの外であらかじめ 1 とおかれていて、それがループの最後まで行くたびに 1 ずつ増えていきます。10 回ループを通過するあいだ i<=10 は True であるはずで、ここには、その間 1 が入っています。10 回目にプログラムがループの最後まで行ったとき、i=i+1;によって、 i は 11 になり、再びループの先頭にやってきた時、i<=10 は False になって、そこには、0 が入って、ループの区間から抜けます。（この場合、プログラムは終了します。）もちろん、この後にプログラムが続くのが普通です。試しに、i<=10 のところに 0 を代わりに書いてみてください。プログラムは正常に終了しますが、何も画面表示されないはずです。なぜなら、do while の後が 0 で False であるからです。その代わりに、1 を書いてプログラムを実行させてみてください。GAUSS は半永久的に無数の erudite を画面表示し続けるでしょう。これは、do while の後が 1 で True で定数でかわりようがないためです。このように、do while の後には条件文がきますが、実は 0 か 1 の論理判定式なのだということを肌で体感してください。for ループと違って、do while のあとの論理判定式を不変のままにしておいては、いつまでたってもループは機能しないのであって、そう言う意味で、プログラムする人が自前でカウンタを設定してカウンタが変化するようにする必要があります。また、カウンタ、例えば i は、GAUSS 上ではただの変数であって、もともと何も入っていません。ですから、最初にループの外で何からはじめるのかを設定しておかなければなりません。

ここで、i=i+1;とループの内部の一番最後の行に書くのは、いささか初学者にとっては見なれないわかりずらい表現かもしれません。ループの外で i は 1 になっていて、これが論理判定式が True であると判断されるとループに入ってきます。そして、1 回目、ループの最後まで到達して 1 + 1 ですから i に 2 が代入されることになります。イコールのマークの後ろにあるものを前に代入するというイコール＝のマークの意味をしっかりと理解しないとわかりづらいかもしれません。また、イコールのマークの後ろの i と、代入されるべき前の i は違う数が入っているのだということを理解することは、プログラムを組んでいく上で、非常に重要なことです。下に示すように、ループのそれぞれの回にカウンタの数が 1 ずつ増えて、その増えた数が i に代入され、その数が、また新たなループの回に使われます。

i =1

[ループ 1 回目]
1 ＜＝10 は True　ループを続ける
画面表示
i = 2

[ループ２回目]
２＜＝10 は True　ループを続ける
画面表示
ｉ＝３

[ループ 3 回目]
3＜＝10 は True　ループを続ける
画面表示
ｉ＝４
．
．
．
[ループ 9 回目]
9＜＝10 は True　ループを続ける
画面表示
ｉ＝10

[ループ 10 回目]
10＜＝10 は True　ループを続ける
画面表示
ｉ＝11

[ループ 11 回目]
11＜＝10 は False　ループから抜ける（ループ内はもう実行しない）

１つことわっておきたいことは、for や do while などの文のあとに；のマークがついていることで、GAUSS では if 文も含めて、たとえそれが条件文であろうとなかろうと１文として扱われて、その最後には必ず；のマークが要求されます。他の言語プログラムを知っている方は、特にこの点に気をとめて、注意してください。

次に、カウンタの数自身がプログラムの計算に使われるよい例として、１から 10 までの数を足しあわせて合計を求めるプログラムをいろいろな書き方で示してみます。ただし、これを求めるには、プログラムすることなしに次のように１行で書けます。

プログラム

```
new; cls;
print sumc(seqa(1,1,10));
```

これは、１から１刻みで増加する 10 個のシークエンスを列ベクトルで返す seqa を用いて、

その返り値の列の値を sumc で合計して、それを直接 print 文で画面表示させています。

カウンタを自前で設定することの必要ない for ループでこれを計算させると以下のようになります。カウンタには i を、合計を代入していく変数には sum を使うこととします。

プログラム

```
sum=0;
for i(1,10,1);
    sum=sum+i;
endfor;
print sum;
```

ループの内容は１行です。変数 sum にカウンタ i をループがまわってくるたびに足しあわせていきます。ここでは、ループの外で、変数 sum を０と初期化しておきます。（何も入っていない sum とカウンタ i を足し合わせることはできませんから、この作業は面倒でも必要になります。）この for ループはカウンタに i を用いて、for 文から endofor までの間のプログラムへ、ループがまわるたびに１から 10 まで１ずつ増加させます。計 10 回ループがまわることになります。第 1 回目はカウンタ i には１が入っていますから sum は０+１で１になります。第 2 回目はカウンタ i には２が入っていますから sum は、前のループで計算した合計１を用いて、これとカウンタ数２を加えて、１+２で３となります。以下同様にして、10 回ループをまわすと、結果的に、１+２+３+４+５+６+７+８+９+１０を計算することになります。最後に、10 回目のループが終了すると endfor の次の命令にいき、10 回ループをまわしたあとの合計 sum を画面表示させます。当然ながら、答えは 55 です。

同様のことを今度は、do ループでやってみます。上の for ループではカウンタ i の操作は必要なかったのですが、今度は、ループ外での初期化（０に設定）と、ループ内最終行で毎回１増加の、２つのことを自前でやってやらなければなりません。

プログラム

```
sum=0;
i=1;
do while i<=10;
    sum=sum+i;
    i=i+1;
endo;
print sum;
```

do while文は、そのあとに論理判別式をともなって、そこの値がTrueで１であり続けるかぎりループをまわす働きがあります。以前説明したのと同様、１回目のループではカウンタ１は10以下ですからTrueで１が返されて、ループ内のプログラムが実行されます。ループ内の最終行 i=i+1;でカウンタが１増加します。２回目もカウンタ２は10以下ですから

Trueで１が返されて、ループ内のプログラムが実行されます。ループ内最終行でカウンタが１増加します。同様に、このループを続けていき、11回目カウンタは11で10以下ではないですからFalseで０が返され、ループ内を実行せずに、ループから抜けます。ループ内では、sum=sum+i;の計算が初期値sum＝０およびｉ＝１から10回ループをまわしますので、

```
sum     sum  i
  1←     0+1
  3←     1+2
  6←     3+3
 10←     6+4
 15←    10+5
 21←    15+6
 28←    21+7
 36←    28+8
 45←    36+9
 55←    45+10
```

というぐあいに、ポインタが１ずつ増えて、10回ループがまわります。よくわかったでしょうか。

　次に、ポインタがループ内で計算ではなくて、行列のディメンションに使われる例を挙げておきましょう。まずは、簡単なものから。今10×１の列ベクトルにループを10回まわして１行目から順に10行目まで７という数字で埋めてみましょう。

プログラム

```
new; cls;
m=zeros(10,1);
for i(1,10,1);
    m[i,1]=7;
endfor;
print m;
```

上のプログラムでは、ループに入る前にあらかじめ、10×１の列ベクトルmを初期化してその要素すべてが０にしてあります。この初期化の作業が、スケラーの時と同様に必ず必要です。試しに、m=zeros(10,1);を取り除いてプログラムを実行させてみてください。プログラムはエラーを出して止まってしまいます。ですから、ループの内部で使われるポインタ以外の変数は必ずあらかじめ何かのスケラーなり行列なりに定義されて、その要素にはなにかの数字が入っている必要があります。ここでは、forループを用いて、１から10まで１ずつポインタを増加させて10回ループをまわします。GAUSSでは、[　　]の中にはディメンションの数字が入る決まりになっています。一方、{　　} の中には行列の実際の要素

が、（　　）の中には組込み関数の引数（変数）が入ります。例えば、m[1,1]とは行列（またはベクトル）の第１行１列目をディメンションを指します。１回目のループでは、このm[1,1]に７が代入されます。２回目には、m[2,1]に７が代入され、３回目には、m[3,1]に７が代入されます。以下同様なことを10回繰りかえし、m[10,1]に７が代入されて、ループを終了します。この10回の作業によって、最初に10×１の列ベクトルのすべての要素が０であったものが、すべて７に置き換わります。そして、すべて要素の置き換わった列ベクトルmを画面表示させています。

同様のことをdoループでやってみます。以前と同様doループの時には、ループの外でカウンタを初期化して、ループの内の最終行でカウンタが変化するように自前でプログラムが必要になります。カウンタは、この場合、０ではなく１に初期化して定義しておかないと、０から10まで11回ループをまわしてしまうことになりますので注意してください。

プログラム

```
new; cls;
m=zeros(10,1);
i=1;
do while i<=10;
    m[i,1]=7;
    i=i+1;
endo;
print m;
```

ここで、forループの時との違いは、太字のように、カウンタｉをループの外で１に初期化しておいて、ループの中で、i=i+1;によってカウンタを毎回１ずつ増加させなければなりません。そのほかは、forループのプログラムとほぼ同じ構造になっています。

もう少し実践的にするため、人工的に５×５のCovVar Matrixを作ってみます。今off diagonalは全て０で、diagonalはm[i,i]=i+u ここでu~N(0,1)にしたがうとしましょう。

プログラム

```
new; cls;
m=zeros(5,5);
for i(1,5,1);
    m[i,i]=i+rndn(1,1);
endfor;
print m;
```

画面表示

```
 1.1698302  0.00000000  0.00000000  0.00000000  0.00000000
0.00000000   3.4372584  0.00000000  0.00000000  0.00000000
```

0.00000000	0.00000000	1.5792176	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000	2.4875016	0.00000000
0.00000000	0.00000000	0.00000000	0.00000000	4.1195825

上の結果は、その時々の乱数の値によって異なってきます。まず、ループの外側で５×５の要素がすべて０の行列を初期化して定義しておきます。ループは10回ではなく５回まわしますから、１から５まで１ずつ増加させるforループということでfor i(1,5,1);としています。for文からendforまでの内容を５回まわして、そのつどカウンタを１から５まで１つずつ増加させることになります。ループの中では、１回目のループでは、m[1,1]のディメンションのところに１+rndn(1,1)の値が入ります。組込み関数rndnは、ここでは１×１のつまりスケラーのstandard normalにしたがう乱数を生成します。２回目のループでは、m[2,2]のところに２+rndn(1,1)の値が入ります。以下、５回これを繰り返して、m[5,5]に５+rndn(1,1)の値を入れると、ループを終了します。できあがった行列mは、そのdiagonalの要素はすべて上の操作によって置き換わった値に置き換わっていて、残りのoff diagonalはすべて初期値の０のままになっています。最後に、このdiagonalだけが置き換わった行列mを画面表示しています。ディメンションの行と列を表す数の両方にカウンタの数を利用してループをまわしているのに加えて、代入する数の方もカウンタの数を利用しているので、少し論理的にあとを追うのがつらいかもしれません。このように、カウンタの数は、イコールのマークの左側のディメンションにも、その右側の代入する数の方にも自由に使えます。コツをよくつかんでほしいと思います。

同じことをdoループで書いてみましょう。カウンタｉをループの外で１に定義して、そのカウンタが５以下のあいだループをまわします。ループの内部の最終行にはカウンタを１ずつ増す行が必要です。

プログラム

```
new; cls;
m=zeros(5,5);
i=1;
do while i<=5;
    m[i,i]=i+rndn(1,1);
    i=i+1;
endo;
print m;
```

乱数の値がその時々で変ってくるので、結果は若干違うと思われますが、diagonalになにか数があって、off diagonalにはすべて０がきているはずです。プログラムの構造は、カウンタの設定と増加行の追加以外は、ほぼforループと同じです。

ここまでは十分に理解できたでしょうか？ここまでのことをプログラムがループの前後をどのように進んでいくかを十分にわかれば、次の二重ループも比較的抵抗なく体感でき

るでしょう。上の例と同じく５×５の行列を考えます。しかしながら、今度はその行をカウンタｉで、その列を別のカウンタｊで、ループでまわして、m[i,j]のところに定数７の値を代入するプログラムを作ってみます。

プログラム

```
new; cls;
m=zeros(5,5);
for i(1,5,1);
    for j(1,5,1);
        m[i,j]=7;
    endfor;
endfor;
print m;
```

画面表示

```
 7.0000000   7.0000000   7.0000000   7.0000000   7.0000000
 7.0000000   7.0000000   7.0000000   7.0000000   7.0000000
 7.0000000   7.0000000   7.0000000   7.0000000   7.0000000
 7.0000000   7.0000000   7.0000000   7.0000000   7.0000000
 7.0000000   7.0000000   7.0000000   7.0000000   7.0000000
```

上のプログラムでは、５×５の零行列のそれぞれの要素を、１行１列目を７に、１行２列目を７に、１行３列目を７に、１行４列目を７に、１行５列目を７に置き換えたあと、２行１列目を７に、２行２列目を７に、２行３列目を７に、２行４列目を７に、２行５列目を７にして、３行目、４行目、５行目の５行５列目を７に、逐次置き換えを行なっています。外側のfor iと外側のendforは互いに対応していて、外側のfor jと外側のendoforは互いに対応しています。読みやすいプログラムにするため、ループの内容はいくつか字数を下げて書くのが慣例で、内側のループはさらに字数を下げます。二重ループのアルゴリズムは以下のようになります。

ｉ＝１のとき[外側ループ１回目]
　　ｊ＝１のとき[内側ループ１回目]
　　m[1,1]＝７
　　ｊのカウンタ１増加

　　ｊ＝２のとき[内側ループ２回目]
　　m[1,2]＝７
　　ｊのカウンタ１増加

　　ｊ＝３のとき[内側ループ３回目]

m[1,3]＝7
ｊのカウンタ１増加

ｊ＝４のとき[内側ループ４回目]
m[1,4]＝7
ｊのカウンタ１増加

ｊ＝５のとき[内側ループ５回目]
m[1,5]＝7
ｊのカウンタ１増加

ｊ＝６のとき　6<=5はFalse ｊループを終了
ｉのカウンタ１増加

ｉ＝２のとき[外側ループ２回目]
ｊ＝１のとき[内側ループ１回目]
m[2,1]＝7
ｊのカウンタ１増加

ｊ＝２のとき[内側ループ２回目]
m[2,2]＝7
ｊのカウンタ１増加

ｊ＝３のとき[内側ループ３回目]
m[2,3]＝7
ｊのカウンタ１増加

ｊ＝４のとき[内側ループ４回目]
m[2,4]＝7
ｊのカウンタ１増加

ｊ＝５のとき[内側ループ５回目]
m[2,5]＝7
ｊのカウンタ１増加

ｊ＝６のとき　6<=5はFalse ｊループを終了

ｉ＝３のとき[外側ループ３回目]
　　ｊ＝１のとき[内側ループ１回目]
　　m[3,1]＝7
　　ｊのカウンタ１増加
　　ｉのカウンタ１増加
　　.
　　.
　　.
　（　中略　）
　　.
　　.
　　.
ｉ＝５のとき[外側ループ５回目]
　　ｊ＝１のとき[内側ループ１回目]
　　m[5,1]＝7
　　ｊのカウンタ１増加

　　ｊ＝２のとき[内側ループ２回目]
　　m[5,2]＝7
　　ｊのカウンタ１増加

　　ｊ＝３のとき[内側ループ３回目]
　　m[5,3]＝7
　　ｊのカウンタ１増加

　　ｊ＝４のとき[内側ループ４回目]
　　m[5,4]＝7
　　ｊのカウンタ１増加

　　ｊ＝５のとき[内側ループ５回目]
　　m[5,5]＝7
　　ｊのカウンタ１増加

　　ｊ＝６のとき　6<=5はFalse　ｊループを終了
　　ｉのカウンタ１増加

ｉ＝６のとき　６<=５はFalse　ｉループを終了

というような、内側のループを５回ずつまわして外側のループを５回まわすアルゴリズムになっています。同様のプログラムは、doループの二重ループでも実現できます。ただし、ｊ＝１の初期化の場所に十分に注意を払って下さい。ｉ＝１のとなりにｊ＝１を書いてはいけません。アルゴリズムを理解していないカット＆ペースト初学者が犯す典型的な誤りになってしまいます。

プログラム（正しい）

```
m=zeros(5,5);
i=1;
do while i<=5;
    j=1;
    do while j<=5;
        m[i,j]=7;
        j=j+1;
    endo;
    i=i+1;
endo;
print m;
```

プログラム（誤り）

```
m=zeros(5,5);
i=1; j=1;
do while i<=5;
    do while j<=5;
        m[i,j]=7;
        j=j+1;
    endo;
    i=i+1;
endo;
print m;
```

なぜｊの初期化も外側のループの外で行なってしまうと、意図した置き換えができないかを詳説することによって、ループがどうまわっているのかを理解する手助けにしてくださ

い。上の誤りのプログラムでは、結果は次のようになります。
画面表示

7.0000000	7.0000000	7.0000000	7.0000000	7.0000000
0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000	0.00000000	0.00000000

１行目しか置き換えができていません。これは、内側の列ｊについてのループが５回まわっているだけで、外側の行ｉについてのループがたった１回しかまわっていないことを示しています。アルゴリズムを追いかけると、

ｉ＝１のとき[外側ループ１回目]
　　ｊ＝１のとき[内側ループ１回目]
　　m[1,1]＝7
　　ｊのカウンタ１増加

　　ｊ＝２のとき[内側ループ２回目]
　　m[1,2]＝7
　　ｊのカウンタ１増加

　　ｊ＝３のとき[内側ループ３回目]
　　m[1,3]＝7
　　ｊのカウンタ１増加

　　ｊ＝４のとき[内側ループ４回目]
　　m[1,4]＝7
　　ｊのカウンタ１増加

　　ｊ＝５のとき[内側ループ５回目]
　　m[1,5]＝7
　　ｊのカウンタ１増加

　　ｊ＝６のとき　６<=５はFalse　ｊループを終了
　　ｉのカウンタ１増加

（これ以降、ｊは６に固定。なぜならｊがｉループの外で初期化されているため。）

ｉ＝２のとき[外側ループ２回目]

ｊ＝６のとき　６<=５はFalse **何もしないで**ｊループを終了

ｉのカウンタ１増加

ｉ＝３のとき[外側ループ３回目]

ｊ＝６のとき　６<=５はFalse **何もしないで**ｊループを終了

ｉのカウンタ１増加

ｉ＝４のとき[外側ループ４回目]

ｊ＝６のとき　６<=５はFalse **何もしないで**ｊループを終了

ｉのカウンタ１増加

ｉ＝５のとき[外側ループ５回目]

ｊ＝６のとき　６<=５はFalse **何もしないで**ｊループを終了

ｉのカウンタ１増加

ｉ＝６のとき　６<=５はFalse　ｉループを終了

というふうに、外側のループが１回目にまわったときには内側のループが５回まわるのだが、そのあとにはカウンタｊが初期化されないため（１に戻らないため）６にとどまったままで、内側のｊループがFalseで何もしないですべて終わってしまうため、残りの外側のループすべてが何もしないままで終わってしまう。初学者も名高い学者も注意したい。

このほか、do whileとは全く逆の論理判別式がくるdo untilがある。カウンタの操作は全く同じだが、論理的には、untilのあとに、not(論理判別式)がくる。例えば、

```
do while i<=10;
do until i>10;
```

はそれぞれ置き換えが可能である。また、

```
do while not(i>10);
do until not(i<=10);
```

はその上のものとすべて置き換えが可能な、まったく同じループ文になる。このことを１から10までの数の合計を求めるプログラムに戻って示しておこう。

プログラム

```
new; cls;
sum=0;
i=1;
do while i<=10;
```

```
    sum=sum+i;
    i=i+1;
endo;
print sum;

sum=0;
i=1;
do until i>10;
    sum=sum+i;
    i=i+1;
endo;
print sum;

sum=0;
i=1;
do while (not i>10);
    sum=sum+i;
    i=i+1;
endo;
print sum;

sum=0;
i=1;
do until (not i<=10);
    sum=sum+i;
    i=i+1;
endo;
print sum;
```

画面表示

```
55.000000
55.000000
55.000000
55.000000
```

上のように、すべてのループは、カウンタｉが１から10まで10回まわって、同じ結果55になります。このことは、whileのあとがTrue(1)であるあいだループがまわること、untilのあとがTrue(1)になるまでループがまわることの２つの意味からすれば当然と言えます。

このように、do whileループとdo untilループは真偽の裏表の関係になっていますから、使用する際には、どちらが読み手にわかりやすいかを考えてプログラムの文脈の中で適宜使い分ければよいでしょう。

このほか、goto文とif endif文を使ってFortranのようにループを強引にまわすこともできますが、GAUSSプログラムでは正統的なforループとdoループを使えば十分でしょう。

ポイント　forループ　カウンタは自動的に管理される

```
for カウンタ変数名 (初期値,最終値,増分);
実行文;
endfor;
```

ポイント　do whileループ　カウンタはユーザーサイドで自前で管理する
論理判別式が真のあいだループをまわす

```
i=1;
do while 論理判別式;
実行文;
i=i+1;
endo;
```

ポイント　do untilループ　カウンタはユーザーサイドで自前で管理する
論理判別式が偽のあいだ真になるまでループをまわす

```
i=1;
do until 論理判別式;
実行文;
i=i+1;
endo;
```

最後に、ループの一番のメリットは、自分自身のTableを自由に作成して印刷して持てるということでしょう。テーブルの載っている書物や統計表の編著者や発行者の許可を得る必要なくなります。自分で必要な桁数作れます。まずは、アメリカのある大学の統計学Ｆ教授が財布に入れていつも持ち歩いているというN(0,1)分布のテーブルを画面表示してみます。standard normal の－∞からｘまでの確率を返す組込み関数cdfn(x)を使って、二重ループで30行10列の0.00から2.99までのテーブルを作成してみます。

プログラム

```
new; cls;
x=zeros(30,10);
i=1;
do while i<=30;
    j=1;
    do while j<=10;
        k=(i-1)*0.1;
        l=(j-1)*0.01;
        x[i,j]=cdfn(k+l);
        j=j+1;
    endo;
    i=i+1;
endo;

print "                Standard Normal Distribution Table (Probability from negative infinity to x)";
print;
print "    x          .00       .01       .02       .03       .04       .05       .06       .07       .08       .09";;
xrow=seqa(0,0.1,30);
x=xrow~x;
format /rd 7,4;
print x;
```

ループは行ｉと列ｊの二重ループを用いて、内側が10回ｊをまわして、外側でそのおのおのについて30回ｉをまわしています。配列ｘ[i,j]に入れる数として行ｉから１引いたものに（０から行は始まっているため）0.1をかけて表の実際の縦軸の目盛に一致させて、同様に列ｊから１引いたものに0.01をかけて表の実際の横軸の目盛に一致させて、それぞれ、ｋとｌとおいて、その足し合わしたもの（すなわち求めたいｘに入る値）をstandard normalの組込み関数cdfnに入れて、負の無限大からその値ｘまでの確率をもとめて、表形式で画面表示させています。後半は、一番上の小数第２位のところをprint文のコメントとして適当に間隔を調整して画面表示させ、小数第１位の左端の縦の列はシークエンスで０から0.1刻みで30個2.9までを作っておいてxrowとしておいて、作成したテーブルに水平方向にマージさせています。フォーマットで、数値と数値の間をせばめ小数点以下の桁数を制御するのに、７文字スペースに小数点以下４桁まで表示する右寄せ十進法表示を採用しています。

ポイント　cdfn(x)　standard normalにおいて－∞からｘまでの確率を返す

画面表示

Standard Normal Distribution Table (Probability from negative infinity to x)

x	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0000	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1000	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2000	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3000	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4000	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5000	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6000	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7000	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8000	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9000	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0000	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1000	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2000	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3000	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4000	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5000	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6000	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7000	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8000	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9000	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0000	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1000	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2000	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3000	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4000	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5000	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6000	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7000	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8000	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9000	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986

(参考)ループを用いないプログラム法

これまで述べてきたように、２重ループで行列のディメンションを解決するのが正統かつコマンドに依存しないプログラム方法として推奨される方法です。これとは別にGAUSSの言語特性を利用したテクニカルなプログラム法もあります。以下の例は、たいへん興味深い例を示しますが、プログラムの読み手としてのものと考えて、自身の使用は控えた方が無難です。万が一、使うにしても、どういうアルゴリズムなのかコメント文等で十分に説明する必要があります。

プログラム(行および列の入れ替え)

```
new; cls;
a={1 2 3,
   4 5 6,
   7 8 9};
ind1={1,3,2}; ind2={3,2,1};
print /rz a[ind1,.];
print /rz a[ind1,ind2];
```

画面表示

```
      1        2        3        ←１行目
      7        8        9        ←３行目
      4        5        6        ←２行目

    ３列目    ２列目    １列目
    ↓        ↓        ↓
      3        2        1
      9        8        7
      6        5        4
```

上のように、もともとの行列ａの行と列をインデックス番号のところに数列を入れて入れ替えてしまうことができます。例えば、ind1が行の順番を表す数列であるとすると、もともとの行列の行の1,3,2行目の順に行列が入れ替わります。さらに、ind2が例の順番を表す数列であるとすると、3,2,1列目の順に行列が入れ替わります。

プログラム（インデックス番号の省略した入力方法）

```
new; cls;
a={0.25 0.10 0.05 0.025 0.01 0.005};
df=seqa(1,1,30)|40|60|120|1e+256;
print "                              A L P H A                              ";
```

```
print "      df          0.25        0.10        0.05        0.25        0.01        0.005";
print "-------------------------------------------------------------";;
table=cdftci(a,df);
df[34]=exp(df[34]);
format /rd 8,3; print df~table;
```

画面表示

	ALPHA					
df	0.25	0.10	0.05	0.25	0.01	0.005
1.000	1.000	3.078	6.314	12.706	31.821	63.657
2.000	0.816	1.886	2.920	4.303	6.965	9.925
3.000	0.765	1.638	2.353	3.182	4.541	5.841
4.000	0.741	1.533	2.132	2.776	3.747	4.604
5.000	0.727	1.476	2.015	2.571	3.365	4.032
6.000	0.718	1.440	1.943	2.447	3.143	3.707
7.000	0.711	1.415	1.895	2.365	2.998	3.499
8.000	0.706	1.397	1.860	2.306	2.896	3.355
9.000	0.703	1.383	1.833	2.262	2.821	3.250
10.000	0.700	1.372	1.812	2.228	2.764	3.169
11.000	0.697	1.363	1.796	2.201	2.718	3.106
12.000	0.695	1.356	1.782	2.179	2.681	3.055
13.000	0.694	1.350	1.771	2.160	2.650	3.012
14.000	0.692	1.345	1.761	2.145	2.624	2.977
15.000	0.691	1.341	1.753	2.131	2.602	2.947
16.000	0.690	1.337	1.746	2.120	2.583	2.921
17.000	0.689	1.333	1.740	2.110	2.567	2.898
18.000	0.688	1.330	1.734	2.101	2.552	2.878
19.000	0.688	1.328	1.729	2.093	2.539	2.861
20.000	0.687	1.325	1.725	2.086	2.528	2.845
21.000	0.686	1.323	1.721	2.080	2.518	2.831
22.000	0.686	1.321	1.717	2.074	2.508	2.819
23.000	0.685	1.319	1.714	2.069	2.500	2.807
24.000	0.685	1.318	1.711	2.064	2.492	2.797
25.000	0.684	1.316	1.708	2.060	2.485	2.787
26.000	0.684	1.315	1.706	2.056	2.479	2.779
27.000	0.684	1.314	1.703	2.052	2.473	2.771

28.000	0.683	1.313	1.701	2.048	2.467	2.763
29.000	0.683	1.311	1.699	2.045	2.462	2.756
30.000	0.683	1.310	1.697	2.042	2.457	2.750
40.000	0.681	1.303	1.684	2.021	2.423	2.704
60.000	0.679	1.296	1.671	2.000	2.390	2.660
120.000	0.677	1.289	1.658	1.980	2.358	2.617
+INF	0.674	1.282	1.645	1.960	2.326	2.576

上のプログラムはｔ分布の百分位点の表を作成するものです。もちろん、２重ループでプログラムするのが正統法ですが、ここでは、

```
a={0.25 0.10 0.05 0.025 0.01 0.005};
df=seqa(1,1,30)|40|60|120|1e+256;
print cdftci(a,df);
```

の３行で実質的なプログラムは足ります。ｔ分布のCDFのcomplimentのインバースマッピングを求める組込み関数cdftci(a,df)を用いて、そのaの方に上のような１行を、dfの方に同じく上のような１列を代入すると、自動的にその列はその数値、その行はその数値となる２組の数値の行列が生成されます。これは、３次元のグラフを描く時に数列を縦横に設定すると自動的に数値が埋まったのと同様なGAUSS固有の機能です。なお、プログラムのその他の部分は表を効果的に見せるためにメッセージ文を画面表示させているのと、dfの値の数列を表の変数tableに水平方向にマージしているだけです。Dfのところの数列は連続している部分はseqaで１つずつ増分させ、数値がとびとびになっている部分は垂直方向のマージである｜のしるしで直接数値をマージさせています。無限大には、無限大を直接代入しても計算はできないので、近似として、1e+256をここでは用いています。表にマージする直前にべき乗してやって、無限大の答えになるようにして、dfの34行目の1e+256と+INFとを置き換えてから画面表示させています。これらは、技術的なことですが、要するに上の３行で表の部分は簡単に作成できるということです。（だたし、この方法はすべての関数で通用するということは保証されていません。例えば、ガンマ関数系の組込み関数のχ²では２列以上いっぺんに値を指定するとエラーになります。その場合は、列だけループでまわせばよいでしょう。）