3.4 プログラミング proc ver. 0.2

長い長い前置きをへて、いよいよGAUSSの本来のプログラム方法であるprocedureの扱い方に入ります。Fortranなどのサブルーチンなどとは違って、それ自身でプログラムのブロックを形成していきます。もし、procをほとんど使わないでプログラムをしている方がいるならば、その人は本当にGAUSSのことを自身で知らないか、きっちりと操作方法を教わっていないことを意味します。このproc役割は、プログラムそれぞれををブロック化するとともに、共通するアルゴリズムを後で利用したり他の人と共有したりすることです。またGAUSSにはない組込み関数をその都度自前で作成するというのもその役割の１つです。

微分積分関係の組込み関数を扱うには、まず関数形を定義しなければなりませんでした。この節の最初でfnという一行関数の定義について詳しくみました。そこで扱ったfnでの関数定義は、実はprocの特殊な例です。procでも同じことができます。関数$f(x_1,x_2)=3x_1^3x_2^2$のgradientとHessianを座標(1,2)について求める同じプログラムを今度はprocで書いてみます。以前のfnで書くプログラムは、

```
new; cls;
fn f(x)=3*x[1]^3*x[2]^2;
x0={1,2};
format /rd 16,3;
print gradp(&f,x0);
print hessp(&f,x0);
```

というものでした。この関数定義fnの部分を、procで本格的に書いてみます。

プログラム

```
new; cls;
proc f(x);
    retp( 3*x[1]^3*x[2]^2 );
endp;
x0={1,2};
format /rd 16,3;
print gradp(&f,x0);
print hessp(&f,x0);
```

基本的に、通常のGAUSSの設定では、プログラム内の前方参照が可能で、慣習的にプログラム内では、procは末尾にまとめて書くというスタイルをとります。したがって、

プログラム

```
new; cls;
x0={1,2};
format /rd 16,3;
```

```
print gradp(&f,x0);
print hessp(&f,x0);
proc f(x);
    retp( 3*x[1]^3*x[2]^2 );
endp;
```

というのが理想的な書き方です。この場合には、関数定義というよりも、f(x)という組込み関数を自前で作るという色彩をおびてきます。上の太字のところの３行が、procを作る際の最小単位の文法を表しています。まず、procの後に、好きな名前のprocedure名をつけ、そのあとの丸括弧の中に、その引数を書きます。これらの引数をもとに、procedureの内部の計算をすることになります。いわば、括弧の中の引数は、インプット変数ということができます。この場合、ｘだけですが、列ベクトルを表しています。最後のendpはprocedureの終わりという意味で、必ずつけなくてはいけません。proc文からここまでが、procedureの内部にあたることをGAUSSに伝えています。procedureの内容は、複数行あることが普通ですが、ここでは、retp(変数名または計算式)によって、１行だけになっています。いわば、print文で　print 2*x^3;などと別の変数に答えを置換えて、その変数を画面表示するのを１ステップとばして、計算式の内容自体をprint文で画面表示させているのと原理は同じです。１行だけで済むことを３行も最低書かなくてはいけないので、proc文で数式を定義するよりはfnで１行関数の定義で済ました方が簡便であると言えます。

　次に、以前グラフィックのところでやったのと同様のχ^2分布にしたがう乱数を発生させるプログラムをprocedureで書いてみましょう。自由度５、1000個の１列のデータ乱数を作成する部分のプログラムは、χ^2分布の定義「N(0,1)にしたがう互いに独立なdf個の確率変数の２乗和の分布」を利用して、

```
rndseed 1000;
df=5;
x=rndn(1000,df);
x=sumc((x.*x)');
```

でしたから、これを一般化してｒ個自由度dfのχ^2分布にしたがう乱数を発生させるには、さらにシードの値を加えてやってプロシジャー内部で計算するようにします。ただし、fnの書き換えのプログラムのように、retpの括弧の中に計算式をまとめて書くのは一般的には困難なことが多いので、そこで、procの内部だけに通用して計算が終わると消えてなくなっていまうテンポラリーな変数である、ローカル変数を定義して、procの内部をもっと複数行にわたる長いものにできます。（ただし、関数自体の戻り値にローカル変数を使っても差し支えありません。）ただし、乱数はシードを設定しても２回目にでてくるときには１回目のものとは違いますから、ただ１回だけ発生させて変数において、これと同じものを要素対要素でかけなければいけません。よく犯す誤りの典型です。

プロシジャー

```
proc rndx2(r,df,seed);
    local x,x2,ss;
    rndseed seed;
    x=rndn(r,df);
    x2=x.*x;
    ss=sumc(x2');
    retp(ss);
endp;
```

できあがったprocは後方に置いて、その前方でその関数のパラメーターに数値を設定してやって、その関数自体をprint文の中に書いて直接画面表示させるか、もしくは、下のように何かの変数（ここでは仮にrx2）においたものを間接的に画面表示させます。

プログラム

```
new; cls;
r=1000;
df=5;
seed=1000;
rx2=rndx2(r,df,seed);
print rx2;

proc rndx2(r,df,seed);
    local x,x2,ss;
    rndseed seed;
    x=rndn(r,df);
    x2=x.*x;
    ss=sumc(x2');
    retp(ss);
endp;
```

上のプログラムでは、関数rndx2の内部だけで使われるローカル変数としてx、x2とssをまず宣言して、それぞれに途中計算の値を代入していき、最後にssの値をretpの丸括弧の中に書いて、関数rndx2の戻り値にしています。このように、retpは、procedureのreturnの略で、その丸括弧の中に、本当は、関数全体の戻り値になるべき変数を入れるのですが、一行で表現されるような簡単な場合には、retpの丸括弧の中に直接計算式を書くこともあるのです。その場合には、ローカル変数はありませんから、local文は不用になっているのです。なお、ローカル変数といっても、引数以外の変数であれば、retpの中の値もローカル変数に含まれます。なお、公開されるprocにはエラー制御をつけることがGAUSSでの作法になっていますが、自分でプログラム内で前方参照して使うかぎりは、その構造や引数の変域が

既知なわけですから、エラー制御の部分は省略してもかまいません。エラー制御については、procを一通り説明してから、その作法について章をもうけて説明することにします。これで、関数内で自由に乱数シードを設定できるようになりました。proc文からendp文までのブロックを、以後自由に他のプログラムにカット＆ペーストして使えます。

もう少しプログラミングの文脈に戻って、procでできることを説明していきましょう。少し複雑になりますが、日本における天下の悪弊EXCELを使うことを回避するために、移動平均を行列ごとやってしまうプログラムをします。直感でもわかるように、表計算ソフトを使わなければ、移動平均はループでまわして計算しなければなりません。しかしながら、GAUSSでは変数はすべて行列（ベクトル）として扱われるため、行ごとループがまわせて、列のことはワイルドカードとして手軽に扱えて、放っておけます。

以下のプログラムは、20行２列のデータを前後５期の移動平均のデータに変換するプログラムです。ｔは５としてありますが、他とはまだリンクさせていません。

プログラム

```
new; cls;
m=seqa(1,1,20)~seqa(2,2,20); t=5;
n=zeros(20,2);
i=3;
do while i<=18;
    j=0;
    do while j<=4;
        n[i,.]=n[i,.]+m[i-2+j,.];
        j=j+1;
    endo;
    i=i+1;
endo;
n=n/5;
print n;
```

このプログラムは、ｍという数列からなる２列のデータを作成しておいて、それを前後５期の移動平均でデータを変換して、ｎという新たな行列データを作るものです。ループは、２重になっていますが、列についてはワイルドカードのドットで扱ってそのままにしています。インデックスiは外側のループで行列の行数番目を表し、インデックスjは内側のループで５つの値を足し合わせていくときの次の行数を１ずつ増やすときの数に使います。複雑に見えますが、行番号は、ｍおよびｎにおいて共通で、そのi番目の行にｊを使って１つずつずらした行の値を５つ足し合わせています。具体的には、ｎ行列を初期化して零行列としたものに、もとのm行列の２つ前の行から順にj=0,1,2,3,4と５つの行を足し合わせて、そのｎのi行目のところに入れています。この作業を外側のループでiを

i=3,4,5,...16,17,18までずらしていって、ｎに前後５期の合計の値が入るようにしています。最後に、ループの外で、行列の値を５で割って、５期の移動平均を求めています。プログラムのコツは、もとの行列もできる行列もディメンションを同じにして、i番目の行と呼んでいることです。ｊには、５期分の合計をまわす役割を持たせています。

これでは、前後２期ずつ０が入っていますので、さらに以下のルーチンを加えます。

```
k=1;
do while k<=2;
    n[k,.]=miss(zeros(1,2),0);
    n[21-k,.]=miss(zeros(1,2),0);
    k=k+1;
endo;
```

このルーチンは、一重ループで、１行目と下から１行目、それから２行目と下から２行目の０の値をミッシングバリューを表すドット.に変更するものです。１から２までループでまわしています。なお、ここのmissという組込み関数は、その１番目の要素に対象となる行列変数、２列目にその値が入っていればドットに置換えるべき値がきます。ここでは、１行２列のそれぞれの行の値が０ですから、その０をドットミッシングバリューに変換をさせています。

ポイント　miss(対象となる行列変数,変更されるべき値)
対象となる行列変数中、変更されるべき値であった場合ドットに置換える

以下がそれらをまとめたプログラムと画面結果です。

プログラム

```
new; cls;
m=seqa(1,1,20)~seqa(2,2,20); t=5;
n=zeros(20,2);
i=3;
do while i<=18;
    j=0;
    do while j<=4;
        n[i,.]=n[i,.]+m[i-2+j,.];
        j=j+1;
    endo;
    i=i+1;
endo;
n=n/5;
```

```
    k=1;
    do while k<=2;
        n[k,.]=miss(zeros(1,2),0);
        n[21-k,.]=miss(zeros(1,2),0);
        k=k+1;
    endo;
print n;
```

画面結果

```
                .               .
                .               .
        3.0000000       6.0000000
        4.0000000       8.0000000
        5.0000000       10.000000
        6.0000000       12.000000
        7.0000000       14.000000
        8.0000000       16.000000
        9.0000000       18.000000
        10.000000       20.000000
        11.000000       22.000000
        12.000000       24.000000
        13.000000       26.000000
        14.000000       28.000000
        15.000000       30.000000
        16.000000       32.000000
        17.000000       34.000000
        18.000000       36.000000
                .               .
                .               .
```

うまくいきましたが、これではｔ期の移動平均について一般化されたわけではありません。ｔ期（ここでは５）をｊやｋと関連づけることによって、一般的なプログラムを書いてみます。上のプログラムの太字のことろを、rows(m)、cols(m)、(t+2)/2、rows(m)-(t-1)/2、t-1、(t-1)/2、ｔ、(t-1)/2、cols(m)、rows(m)+1、cols(m)などに置き換えて一般化してみましょう。

プログラム

```
    new; cls;
```

```
m=seqa(1,1,20)~seqa(2,2,20); t=5;
n=zeros(rows(m),cols(m));
i=(t+1)/2;
do while i<=rows(m)-(t-1)/2;
    j=0;
    do while j<=t-1;
        n[i,.]=n[i,.]+m[i-(t-1)/2+j,.];
        j=j+1;
    endo;
i=i+1;
endo;
n=n/t;
k=1;
do while k<=(t-1)/2;
    n[k,.]=miss(zeros(1,cols(m)),0);
    n[rows(m)+1-k,.]=miss(zeros(1,cols(m)),0);
k=k+1;
endo;
print n;
```

さらに、procでブロック化します。いま、一般化したmav(m,t)という関数をつくってそこにmという行列データとｔ期の引数を代入して、移動平均を求めます。

プログラム

```
new; cls;
m=seqa(1,1,20)~seqa(2,2,20); t=5;
print mav(m,t);
proc mav(m,t);
    local n,i,j,k;
    n=zeros(rows(m),cols(m));
    i=(t+1)/2;
    do while i<=rows(m)-(t-1)/2;
        j=0;
        do while j<=t-1;
            n[i,.]=n[i,.]+m[i-(t-1)/2+j,.];
            j=j+1;
        endo;
        i=i+1;
```

```
    endo;
    n=n/t;
    k=1;
    do while k<=(t-1)/2;
        n[k,.]=miss(zeros(1,cols(m)),0);
        n[rows(m)+1-k,.]=miss(zeros(1,cols(m)),0);
        k=k+1;
    endo;
    retp(n);
endp;
```

上のようにproc以下endpまでをまとめれば、以後、どんな行列データ、どんな期の移動平均でもカット＆ペーストでプログラムの後方にprocedureを置いてやることで呼び出せるようになります。試しに、プログラム２行目のデータｍとｔ期を

```
m=seqa(1,1,20)~seqa(2,2,20)~seqa(10,1,20); t=9;
```

と変更してみましょう。

画面結果

```
          .           .           .
          .           .           .
          .           .           .
          .           .           .
  5.0000000   10.000000   14.000000
  6.0000000   12.000000   15.000000
  7.0000000   14.000000   16.000000
  8.0000000   16.000000   17.000000
  9.0000000   18.000000   18.000000
  10.000000   20.000000   19.000000
  11.000000   22.000000   20.000000
  12.000000   24.000000   21.000000
  13.000000   26.000000   22.000000
  14.000000   28.000000   23.000000
  15.000000   30.000000   24.000000
  16.000000   32.000000   25.000000
          .           .           .
          .           .           .
          .           .           .
          .           .           .
```

どんな呼び出し方をしようとも、どんなデータ行列であろうとも、以降この自前のmavという関数で計算できるようになります。もう表計算ソフトは不要なはずです。

なお、実際にこのprocedureを公開するには、さらに、ｔ期のｔが奇数であって、偶数になれないとか、データの中にミッシングバリューがあったときにどうするのかというったエラー処理のアルゴリズムを組み込む、コメント文で、著作権や関数の使い方の説明を明記することになります。それは省略します。

ｔ期の移動平均ができるのなら、ｔ期の移動幾何平均もできるはずです。上でやったプログラムをもとに、すこし改変作業をしてみます。行列ｎの初期化の時に零行列ではなくて、すべての要素が１の行列を使うことと、内側のｊループで足し合わせるのではなくて要素対要素でかけ合わせる.＊を用いること、そしてループの外側でｔで割るのではなくて1/t乗してやることによって、いとも簡単に移動幾何平均の関数mgav(m,t)ができます。

プログラム

```
new; cls;
m=seqa(1,1,20)~seqa(2,2,20); t=5;
print mgav(m,t);

proc mgav(m,t);
    local n,i,j,k;
    n=ones(rows(m),cols(m));
    i=(t+1)/2;
    do while i<=rows(m)-(t-1)/2;
        j=0;
        do while j<=t-1;
        n[i,.]=n[i,.].*m[i-(t-1)/2+j,.];
        j=j+1;
    endo;
    i=i+1;
endo;
n=n^(1/t);
k=1;
do while k<=(t-1)/2;
    n[k,.]=miss(zeros(1,cols(m)),0);
    n[rows(m)+1-k,.]=miss(zeros(1,cols(m)),0);
k=k+1;
endo;
retp(n);
```

```
endp;
```

画面表示

```
          .          .
          .          .
   2.6051711  5.2103422
   3.7279193  7.4558385
   4.7893890  9.5787779
   5.8273869  11.654774
   6.8534675  13.706935
   7.8725669  15.745134
   8.8871937  17.774387
   9.8987712  19.797542
   10.908172  21.816343
   11.915961  23.831921
   12.922523  25.845046
   13.928129  27.856258
   14.932974  29.865948
   15.937204  31.874409
   16.940930  33.881860
   17.944237  35.888474
          .          .
          .          .
```

ループが実際にまわっているか心配であれば、プログラムの前半の空いたスペースに、

```
print (1*2*3*4*5)^(1/5);
print (2*3*4*5*6)^(1/5);
print (16*17*18*19*20)^(1/5);
```

などと埋めこんでやれば、途中の計算の験算ができます。なお、プログラムの途中でエラーを起こし、それを何回も直すことによって、特にループを含んだプログラムは完成します。その際に、

```
Index out of range
Rows don't match
```

などのエラーメッセージに直面すると思います。これらのエラーの大半は、10メートル離れた区間に、１メートルごとに電柱を立てるのに何本かかるかという問題に帰着します。答えは11本なのですが、実際のプログラムではこれを忘れて10本であるとしてしまいがち

です。注意したいポイントです。０からｊが始まっていれば、５回ループをまわして足し合わせるのには、４までの数で十分なはずです。５期の移動平均の計算には、前２つ、そのものと、後ろ２つ分のデータがいるわけで、iは３行目から始まって18行目までを考えればよいはずです。これを３行目から行の数いっぱいの20までループiについてまわしてしまうとエラーになってしまいます。そのほか、proc文にも、retp文にも、endp文にもすべてその後に；のマークが必要です。Ｃ言語のように{　　}でプログラムのモジュールの始めから終わりまでを包む代わりに、その都度、きっちりとendpで最後はしめる必要があります。if文にもdo loop文にも、それぞれendif文やendo文があるのは、このしめのためです。