3.1 プログラミング fn　　ver. 0.2

第３節では GAUSS プログラミングの基礎とその作法についてくわしく見ていきます。GAUSS では他の統計計量のパッケージソフトと違って、いくつかの procedure を作ってそれをつなぎ合わせることによって、プログラムを組み立てていく手法をとるのが正統的なやり方です。まずは、いわば procedure のミニチュア版とも言えるファンクション fn を用いた１行で表される関数の定義について見てみます。

関数定義 fn は、非常に便利な機能で、主に procedure の作法にのっとってプログラムを前方参照させることなしに、手軽にプログラムの流れの中に関数形を定義する際に使われます。たとえば、$f(x)=3x^3-4x^2+2x-7$ という関数をあらかじめ定義しておいて、f(9)を求めるやり方は次のようになります。

プログラム

```
new;  cls;
fn f(x)=3*x^3-4*x^2+2*x-7;
y=f(9);
print /rz y;
```

画面表示

```
        1874
```

上のように、fnは関数を定義する際に用いて、プログラムでそれ以降、普通の代数の計算と同じように、関数に値を代入して計算することができます。なお、関数名は何であってもかまいませんし、変数名も何であってもかまいません。上の例は、関数名をｆ、変数をｘとした場合の１変数の例です。関数f(x)をfnで定義した後、そのｆに９を代入して計算したものをｙとおいて、それを右寄せ小数点以降の連続する０を省略するフォーマットで画面表示したものです。

もちろん、関数名をｆ以外のものにしたり、２変数、３変数にすることも可能です。

プログラム

```
new;  cls;
fn f1(x,y)=x^(2/3)*y^(2/3);
a=f1(4,7);
print "        f1(4,7)=" a;
fn f2(x,y,z)=sin(x)+cos(y)+tan(z);
b=f2(0.5,0.5,0.5);
print "f2(0.5,0.5,0.5)=" b;
```

画面表示

```
      f1(4,7)=        9.2208726
f2(0.5,0.5,0.5)=      1.9033106
```

画面表示を見やすくするためにprint文に引用符でくるんで文字を表示させていますので、すこし複雑に見えますが、内容としては、第１番目のプログラムと同様に、fnで関数を定義して、その関数に数値を代入して、それを別の変数とおいた上で、画面表示させています。２変数の関数の場合には関数名のあとの括弧のなかにカンマをつけて任意の変数を入れて、その２変数を使ってイコール以降に関数形を書きます。３変数の場合も同様で、括弧のなかにカンマをつけて任意の変数を入れて、その３変数を使って、イコール以降に関数形を書きます。その利用の仕方は、１変数の場合と全く同じように、関数の中に、実際の数値を複数個入れて計算させます。

上の２つの例のように定義される関数がスケラー変数であることは、GAUSSプログラミングではまれで、むしろ、行列、少なくとも列ベクトルとして使われていることが大多数です。関数定義fnでは各変数のディメンションを明示的に書くことはできませんが、その関数にかけ合わせる変数のディメンションと代入される数値によってGAUSSは、その関数がスケラーなのか列ベクトル（または行列）なのかを決定します。まずは、代入する数値によって、同じ定義された関数が、スケラーにも列ベクトルにも（あるいは行ベクトルにも）行列にもなり得る例を示します。

プログラム

```
new;  cls;
fn f(x)=2*x^3+5*x^2;
a={1,2,3}; b={1 2 3}; c={1 2 3, 4 5 6, 7 8 9};
print /rz f(7); print /rz f(a); print; print /rz f(b); print /rz f(c);
```

画面表示

```
931


  7
 36
 99

  7        36        99

  7        36        99
208       375       612
931      1344      1863
```

上の例は、関数定義fnで$f(x)=2x^3+5x^2$と定義しておいて、右寄せ連続する小数点以下の０を省略する/rzフォーマットでｆ(x)にスケラー７、列ベクトルａ、行ベクトルｂ、そして３×３の行列ｃを代入した結果を示しています。なお、GAUSSでは、行ベクトルを表示させる場合には、標準的な行列表示のように１行送りをしませんので、print;で１行挿入してい

ます。上の画面出力結果のように、同じ関数でも、代入する数値の型にあわせて代入結果はかわってきます。

もちろん、上のように代入される数値によって関数のディメンションが決まることが優先されるのですが、関数内の変数にかかる数がスケラーかベクトルによって、行列のcomformableの計算上の制約からも関数がスケラーなのかベクトル（あるいは行列）なのかも決定されます。例えば、

```
a1=7; b1=9;
fn f1(x)=a1*x*b1;
```

というふうに設定すると、関数ｆ１は一般的な行列の関数として定義されます。スケラーもGAUSSでは１×１の特殊な行列として扱われます。もちろん、ｘのところに列ベクトルを代入すれば、関数ｆ１は同じ関数ばかりの集まりの列ベクトルになることは、以前の例からも明らかです。ｘにスケラーを代入すれば、結果はスケラーで返されます。しかしながら、次の場合には、必ず、関数は行列として定義されます。

```
a2={9 8 2}; b2={3 9 1, 2 4 8, 5 1 2};
fn f2(x)=a2*x*b2;
```

ｘに右からかかる a2 は１×３、左からからかかる b2 は３×３ですから、ｘは３×３であるはずで、ｘに３×３の数値を与えてやると、ｆ２の返す結果は１×３の行ベクトルになるはずです。このように、GAUSS では、データを読み取る際以外には、変数の型やディメンションの宣言をしないのが普通で、プログラムする際には、変数のディメンションを考えながらアルゴリズムをユーザーサイドで組み立てなければなりません。プログラムの読み手も、もとのデータのディメンションや各行列変数の comformable な関係をたどって、どういうディメンションの計算が行なわれているのかを理解する必要があります。これは、関数定義 fn や proc に限ったことではなくて、GAUSS プログラミング全般に言えます。

ポイント　fn 関数名(x,y,z,….)=関数式;　　fn は「１行で」表現できる関数を定義する

さて、この fn は非常に便利なのですが、一体どういう場合に使えばいいのでしょうか。GAUSS のマニュアル関係には、微積関係の組込み関数を用いる場合、proc の手法でで関数形を定義して、その proc をプログラムの一番おしまいにおいて、それらの微積の組込み関数の中で前方参照させるように解説されています。また、多くの公開されているスクリプトにもその方法が踏襲されています。しかしながら、関数定義 fn の本当の使い道は、エラーや分岐を考えなくてもよい、１行で定義される関数、すなわち微積の組込み関数で使われるような関数形だけがほしいような場合を念頭においてもともと作られた機能であると言えます。マニュアルに記述が全くないために、以下のような使われ方をすることはまれですが、実際にはそう使われるべきなのです。以下、微積関係の組込み関数の使い方を関数定義 fn との関連で説明していきます。

まずは、ある座標上の数値で評価された gradient を求める方法です。まず、関数 f(x)を定義したあとに gradp という組込み関数で、その関数 f を呼び出して、評価されるべき値を代入します。ここでは、関数は単一変数の列ベクトルである必要があります。

プログラム

```
new;  cls;
fn f(x)=2*x^3+5*x^2;
x0=1;
y=gradp(&f,x0);
print y;
```

画面表示

```
   16.000000
```

上の例は、関数f(x)=$2x^3+5x^2$とfnで定義したうえで、x0=1で評価されるgradientを求めています。すなわち、f'(x)=$6x^2+10$に１を代入した値と同じものを求めていることになります。gradientを求める組込み関数gradpは、その第１要素に定義された関数名に＆のマークをつけておき、第２要素には評価すべき場所の座標を入れます。この場合、スケラー１がx0とおかれたうえで代入されています。ただし、原理的にGAUSS内部では、Exactな微分した式に値を代入しているのではなくて、微分の定義式

$$y = f'(x) = \frac{f(x+h) - f(x)}{h}, \quad as \qquad h \to 0$$

に基づいた近似値を出しています。すなわち、

プログラム

```
new;  cls;
fn f(x)=2*x^3+5*x^2;
x0=1;
h=1e-8;
y=(f(x0+h)-f(x0))/h;
print y;
```

というふうに、hのところに例えば1e-8というような十分に小さな値を入れることによって、線形近似を行なっています。これにより、この場合は誤差なく答えは16と出ます。

今度は、評価する値が１値ではなくて列ベクトルであると次のようになります。

プログラム

```
new;  cls;
fn f(x)=2*x^3+5*x^2;
x0={1,2,3};
y=gradp(&f,x0);
print y;
```

画面表示

```
      16.000000       0.00000000       0.00000000
      0.00000000       44.000000       0.00000000
      0.00000000       0.00000000       84.000001
```

この場合には、以前と同じ関数f(x)を使って、評価するポイントを列ベクトル{1,2,3}にしてgradientを求めています。関数f(x)に３×１の値を入れて評価するのですから、関数f(x)も３×１の列ベクトルになります。以前と同様、組込み関数gradpでは、ファイル名に&のマークをつけて呼び出し、評価すべき列ベクトルx0を代入しています。なお、評価する値は、必ず列ベクトルまたはスケラーである必要があります。カンマなしの行ベクトル、または２列以上の行列は不可です。画面結果でわかるように、diagonalは列ベクトルf(x)の１行目から３行目まで同じ形の関数であるため、関数を微分したそれぞれに１、２、３を順番に代入した形になっています。ただし、off diagonalはすべて０になっています。ちょうど変数ｘの第１要素がｘ、第２要素がｙ、第３要素がｚという具合になっていて、偏微分の評価値が０になっているのです。上の84.000001は、明らかに、本来は84ですが、gradpの線形近似のアルゴリズムにより計算の小数以下が若干上方にぶれる傾向があります。したがって、実際にGAUSSの微積で出た結果には注意が必要ですし、小数以下３～４桁より下の値には信頼性がありません。ある桁数を決めて、ラウンドするか、フォーマットで小数の下の方の値は表示されないようにする必要があります。

次に、同じ関数f(x)で、Hessianを求めてみましょう。やり方はgradpとほぼ同じです。

プログラム

```
new; cls;
fn f(x)=2*x^3+5*x^2;
x0={1,2,3};
y=hessp(&f,x0);
print y;
```

画面表示

```
       21.999611    -0.00038754908    -0.00025836606
-0.00038754908          34.000263          0.00000000
-0.00025836606          0.00000000          46.000354
```

上の結果は明らかに誤りです。組込み関数gradpと同様、小数点以下３～４桁より小さい桁の精度は保証されていません。上のdiagonalは整数であるはずですから、右寄せ十進法表示、16文字スペース分確保、小数点以下３桁のフォーマットである、

```
format /rd 16,3;
```

という行をprint文の前に書き入れれば、おおよそのところの結果がでます。

画面表示

```
          22.000              0.000              0.000
```

0.000 34.000 0.000
0.000 0.000 46.000

上の結果のoff diagonalはすべて０、diagonalは整数になっています。このようにGAUSSでは、微分にかかわる計算には、最大級の慎重さを要します。必要であれば、数学系のソフトでgradientやHessianを別途求めてから、GAUSS上に代入するのも手かもしれません。組込み関数hesspでは、gradpと同様に、＆マークつきの関数名を呼び出して、列ベクトルの評価座標を与えます。上の例で言えば、x0に相当する部分は、必ず列ベクトル（またはスケラーであることが必要です。結果は、対称行列になります。

ポイント　gradp(＆関数名,評価列ベクトル)　　gradientの行列を返す
ポイント　hessp(＆関数名,評価列ベクトル)　　Hessianの行列を返す
gradp、hesspともに小数点以下３～４桁よりも小さい桁数は保証されず
関数名はfnまたはprocであらかじめ定義する必要があり

より一般的なoff diagonalも非零になる例は、以下のようなプログラムになります。結果が対称行列になっていることがよくわかると思います。

プログラム

```
new; cls;
a={1 2 3};
fn f(x)=exp(a*x);
x0={3,2,1};
y=hessp(&f,x0);
format /rd 16,3;   print y;
```

画面表示

22026.854 44053.620 66080.562
44053.620 88107.629 132160.860
66080.562 132160.860 198240.993

計量の計算では、行列のままで計算するので用が足りますが、経済数学として利用するのには、x[1]やx[2]といったディメンションの簡略形をx_1やx_2の代わりに使うことをよくします。この方法を用いたやり方で$f(x_1,x_2)=3x_1^3x_2^2$のgradientとHessianを座標(1,2)について求めてみましょう。以下がプログラムです。小数点第３位よりも小さい桁は有効ではありませんので、format文で見た目上まるめます。

プログラム

```
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);
```

画面表示

```
        36.000          12.000

        72.000          36.000
        36.0             6.000
```

上の最初の１行２列の結果は、それぞれgradientである$f_1(1,2)$、$f_2(1,2)$の計算結果です。その次の２×２の行列の結果は、それぞれHessianである$f_{11}(1,2)$、$f_{12}(1,2)$、そして次の行に移って$f_{21}(1,2)$、$f_{22}(1,2)$の計算結果です。

さて今度は積分です。fnで定義された関数の定積分を求める方法を説明します。１つ前のf(x)をそのまま使って、０から１までの区間を定積分してやりましょう。

プログラム

```
new; cls;
fn f(x)=2*x^3+5*x^2;
x1={1, 0};
format /ro 24,16;
print "----------------|123456789|12345--";
y=intsimp(&f,x1,1e-11); print "1e-11     :" y;
y=intsimp(&f,x1,1e-10); print "1e-10     :" y;
y=intsimp(&f,x1,1e-9); print "1e-9      :" y;
y=intsimp(&f,x1,1e-8); print "1e-8      :" y;
y=intsimp(&f,x1,1e-7); print "1e-7      :" y;
y=intsimp(&f,x1,1e-6); print "1e-6      :" y;
y=intsimp(&f,x1,1e-5); print "1e-5      :" y;
z=intquad1(&f,x1); print "Legendre " z;
```

画面表示

```
----------------|123456789|12345--
1e-11     :          2.166666666660309
1e-10     :          2.166666666641235
1e-9      :          2.166666666259766
1e-8      :          2.166666660156250
1e-7      :          2.166666640625000
1e-6      :          2.166666640625000
1e-5      :          2.166666640625000
```

Legendre　　2.166666666666666

上の例では、f(x)についての０から１までの区間の定積分を２つの方法で計算させています。１つは、第３要素に有効桁数を指定してシンプソン法で計算する組込み関数intsimp。もう１つは、ガウス＝ルジャンドル求積法で計算する組込み関数intquad1。どちらの関数も、その第１要素に関数名に＆のマークをつけて呼び出し、第２要素に２×１の区間を表す列ベクトルを入れます。０から１までを区間として指定するのには、{1,0} というふうに、インテグラルの上にくる区間の方をまず書いて、カンマで区切って、インテグラルの下にくる方を書きます。逆にしてしまうと、当然のことながら、マイナスの数になってしまいます。一部の書物には、intsimpの方がより精度の高い計算をするいう記述がありますが、上の例でみるように、手計算の定積分の結果は13/6ですから、ガウス＝ルジャンドル求積法の方が正確になるケースもあり得るわけです。ケースバイケースで使い分けるとよいのではないでしょうか。なお、シンプソン法は、有効桁数を標準で指定できますが、区間を表す第２要素は２×１の１組分の計算しかできません。しかしながら、ガウス＝ルジャンドル求積法の方は、２組以上の区間の定積分が可能です。グローバル変数を使えば、有効桁数の変更も可能です。以下が、その例です。

プログラム

```
new; cls;
fn f(x)=2*x^3+5*x^2;
x1={2 2.5 3, 0 0 0};
format /ro 16,12;
z=intquad1(&f,x1);
print z;
```

画面表示

```
21.3333333333
45.5729166667
85.5000000000
```

上の例は、定積分の区間が、０から１ではなくて、０から２、０から2.5、０から３の３組についてまとめて計算した結果です。このように２×１の列ベクトルの区間の組を水平方向に作っていくことによって、まとめて定積分できます。なお、この機能はシンプソン法にはありません。

上の組込み関数intquad1の１は普通の１重の定積分のことで、２重積分の場合には２が、３重積分の場合には３が、１の代わりにつきます。当然のことながら、fnで定義する関数は、２重積分の場合には２変数が、３重積分の場合には３変数が、それぞれ必要になってきます。区間の方も、２重積分の方は２組、３重積分の方は３組、それぞれ必要になってきます。以下がその簡単な場合についてです。まず、下の二重定積分を計算してみます。

$$\int_0^1 \int_0^1 xydxdy$$

プログラム

```
new; cls;
fn f(x,y)=x.*y;
x1={1,0}; y1={1,0};
a=intquad2(&f,x1,y1);
  print a;
```

画面結果

```
 0.25000000
```

二重定積分の組込み関数intquad2を用いるのには、変数を２つもつ関数を定義した上で、積分する区間も２変数分必要になります。関数の名前は何でもいいのですが、仮にｆとして、そこでの２変数をｘとｙとします。関数を定義するためにfnを使って、f(x,y)＝数式;とすれば関数は定義されます。通常の数学とここまでは同じです。ただし、intquad系の組込み関数は、積分の区間の複数の組を１度に計算するようなアルゴリズムで成り立っているため、数式部分は、同じ式の集まりの列ベクトルであることを許すような式でなければなりません。そのため、スケラー対スケラーを念頭に置いたｘ＊ｙではなくて、列ベクトルの要素対要素を念頭に置いたｘ.*ｙになっています。掛け算同様、割り算の場合も./の要素対要素の演算を用いる必要があります。そして、ｘの区間を０から１までという意味で、{１,０}と指定して（視覚的に１が上になっています。これを逆転させると積分の値はマイナスになります)、ｙの区間を０から１まで{１,０}と指定してから、inquad2で関数ｆを&のマークをつけて呼び出します。その第２要素にはｘの区間の入った変数、第３要素にはｙの区間の入った変数を入れます。

２組以上の定積分の組をするのも、１重積分の場合と同じです。区間の変数のところを行列代入します。１組目がｘが０から１で、ｙも０から１、２組目がｘが０から２、ｙが０から３とすれば、以下のような指定をします。

プログラム

```
new; cls;
fn f(x,y)=x.*y;
x1={1 2,0 0}; y1={1 3,0 0};
a=intquad2(&f,x1,y1);
   print a;
```

画面表示

```
 0.25000000
 9.0000000
```

次に、三重定積分も同様に計算できることを示します。

プログラム

```
new; cls;
fn f(x,y,z)=x.*y.*z;
x1={1,0}; y1={1,0}; z1={1,0};
a=intquad3(&f,x1,y1,z1);
print a;
```

画面表示

```
 0.12500000
```

上のように、３変数の関数f(x,y,z)をfnで定義して、ｘ、ｙ、ｚの３区間を決定し、組込み関数intquad3で関数を＆つきで呼び出して、３変数の区間をそれぞれ順に代入します。定義される関数の中で、列ベクトルを念頭に置いた要素対要素のドットのついた演算をするのは以前の例と同じです。

ポイント　intsimp(＆関数名,ｘの区間,有効計算精度桁数)　　シンプソン法による積分
ポイント　intquad1(＆関数名,ｘの区間)　　一重積分
ポイント　intquad2(＆関数名,ｘの区間,ｙの区間)　　二重積分
ポイント　intquad3(＆関数名,ｘの区間,ｙの区間,zの区間)　　三重積分

定積分には、このほかに、二重積分の場合に、ｙの区間にｘの関数を入れる方法に対する組込み関数intgrat2、三重積分の場合に、ｙの区間にｘの関数を、ｚの区間にｘとｙからなる関数を入れる組込み関数intgrat3が用意されています。まずは、二重積分から

プログラム

```
new; cls;
fn f(x,y)=x.*y;
fn g1(x)=x^2+x./4+3;
fn g2(x)=3;
x1={1,0}; g=&g1|&g2;
a=intgrat2(&f,x1,g);
print a;
```

画面表示

```
  1.1411458
```

上の例は、以前と同じf(x,y)＝xyを二重積分するものですが、ｘの区間は０から１、しかしながら、ｙに相当する区間は、３からg(x)=x^2+x/4+3までｘの区間にしたがって、積分するものです。したがって、関数ｇには、このg1(x)=x^2+x/4+3と定数３を垂直方向に｜のマークでマージさせた２×１のベクトルが入ります。ただし、これらは数値ではなく、定数３も含めて関数ですので、g1、g2を指すものとして＆をつけて、&g1と&g2を垂直方向にマー

ジさせたものをｇと置きます。組込み関数intgrat2には、その第１要素に関数ｆの指すもの、つまり＆ｆが入り、第２要素にはｘの区間を表す変数、第３要素にはｙの区間を表す変数が入りますが、ここでは、ｘの関数ｇが入ります。なお、ｇには＆のマークはつけません。もちろん、g(x)の区間の下限は定数である必要はなくてｘの関数でかまいません。

同様にして、三重積分の例を挙げます。

プログラム

```
new; cls;
fn f(x,y,z)=x.*y.*z;
fn g1(x)=x^2+x./4+3;
fn g2(x)=3;
fn h1(x,y)=x+y;
fn h2(x,y)=0;
x1={1,0}; g=&g1|&g2; h=&h1|&h2;
a=intgrat3(&f,x1,g,h);
print a;
```

画面表示

```
   10.372058
```

上の例は、関数f(x,y,z)=xyzに対して、二重積分までは同じで、ｚに相当する区間には、上限h1(x,y)にはｘ+ｙ、下限h2(x,y)には０として、それらを垂直方向にマージしてｈとしたのちに、組込み関数intgrat3で三重積分したものです。ｇと同様、ｈにもintgrad3の関数内では＆のマークは必要ありません。

ポイント　intgrat2(＆関数名,ｘの区間,g(x))　　ｘの区間を介した二重積分
ポイント　intgrat3(＆関数名,ｘの区間,g(x),h(x,y))　ｘの区間を介した三重積分

このように、関数定義fnを用いれば、簡単に見た目の数学と同じように１行関数が定義できて、微積関係の組込み関数が扱えるようになります。もちろん、公式マニュアルその他に書いてあるようにprocを使って関数を定義することも可能ですが、１行の関数の定義にそこまでする必要もありません。組込み関数で呼び出すだけの１行で表現できる関数には、このfnの使用を推奨します。