4.1 特殊関数とライブラリ 最適値問題(1) ver. 0.2

ライブラリとは、厳密には、libraryという命令で呼び出す組込み関数群をさします。ライト版も含めて標準で装備されているライブラリは、グラフを描くためのライブラリpgraphです。これも、library pgraph;で呼び出されたことを思いおこしてください。このpgraphも、内部には、ｘｙグラフやバーグラフ、ヒストグラムなどたくさんの組込み関数が含まれていました。そういう意味で、ライブラリとは、libaray命令で呼び出さなければ使えないような組込み関数の集まりとでも言えましょう。

そのGAUSSのシステムにどのようなライブラリが装備されているかは、GAUSSのフォルダー内のlibという名前の中に入っていますので確認できます（一方、組込み関数はsrcという名前の中に入っています)。フルバージョンのGAUSSをお使いでしたら、maxlikやcmlといったライブラリが見当たるはずです（ライト版の場合はpgraphと諸定義のファイルのみ)。呼び出し方は、pgraphのときと同じように、

```
library maxlik;
maxset;
```

や

```
library cml;
cmlset;
```

などとします。これらのライブラリ呼び出しによって、それぞれのライブラリの中の組込み関数群が使えるようになります。また、graphset;と同じように、maxset;やcmlset;の命令によって、そのライブラリで扱う各種設定のグローバル変数を初期化します。（もし＃のマークがついたところがあるなら、それはコンパイルする際にのみ使われるインデックスのことで、通常のGAUSSの使用では使いません。）これらのよく使われるライブラリの使用法とその応用例は後の章で説明します。

これに関連して、もう１つ言っておかなければならないことは、libraryといってもprocの集まりで、その大多数がグローバル変数を持った関数で、通常のprocの書いてある.srcファイルが存在するほかに、グローバル変数が定義されている.decファイルと外部グローバル変数が定義されている.extの２つがあります。なお、.srcファイルはGAUSSの中からは、標準ファイルであるので、拡張子の.srcは見えません。一般に、.gはユーザーが定義した関数で区別した方がよいとされます。(.srcファイルのほかに、.arcという拡張子のファイルも同様にprocの書かれたところとして認められていますが、現在では同名の圧縮ツールの拡張子がWindowsの世界には存在しますので、.arcは避けたほうが無難です。）したがって、.srcのファイルを見れば、procのコメント文で書かれた説明文とそのプログラム本体がわかります。また、グローバル変数および外部グローバルがどのようなものがあるのかについては、.decと.extのファイルを見てみればわかります。このように、GAUSSのprocソースは、ライブラリも含めてユーザーに公開されています。ユーザーは、公式マニュアル以上のほ

かに、このsrcの冒頭のコメント文の説明およびプログラム本体を見て参考にしてプログラムを組み立てるとともに、存在しない関数や有効数字が不十分な関数についてはproc形式で自作することになります。自作したproc形式の関数が多くなれば、それをライブラリとして一括してまとめた形式にして公開するなり、オプションとしてお金を取って商用化するなりすることになります。（use gpe2;として呼び出すProf.LinのGPE2パッケージは、ソースがコンパイルされてしまっていて非公開なこと、そしてlibraryとして呼び出せないことからしても、GAUSSの本体の仕様とは別物であることがおわかりになるかと思います。）

さて、第４節のメインのmaxlikとcmlのライブラリの使用方法に入る前に、本章では数式の処理関係で組込み関数で説明していなかった事項と、最大化関係の標準装備の組込み関数について説明します。

まずは線形の場合の解法について述べます。行列表示でＡｘ＝ｂという場合、Ａとｂが既知であるとするとベクトルｘは、$x = A^{-1}b$で計算できますから、GAUSSでは行列Aとベクトルbをあらかじめ設定した後、

```
x=inv(A)*b;
```

ということになります。これはクレーマーのルールを解いていることと同じで、ほかに、

```
x=b/A;
```

とすれば簡単に計算できます。GAUSSでは、Aのところにスケラー以外のものがくると割り算ではなくて解を求める計算をすることを思いおこしてください。Aのところに行列またはベクトルがきて、それでも要素対要素で割り算をしたい場合には、./としてドット計算をすることになります。行列Aとベクトルｂを設定してAx＝ｂとなるｘを求めるには、

プログラム

```
new; cls;
A={3 5 8,
   2 2 9,
   8 5 2};
b={1,
   3,
   6};
x1=inv(A)*b;
x2=b/A;
print x1;
print x2;
```

画面表示

```
 1.3195266
-1.0177515
 0.26627219
```

```
 1.3195266
-1.0177515
 0.26627219
```

というぐあいになります。ｘ＝A^{-1}ｂで計算しても、ｘ＝ｂ/Aで計算しても、結果は全く同じになることがわかるでしょう。したがって、ニュートンステップの計算(HはHessian行列で、ｇはgradientベクトル)では、

```
ｂ＝ｂ－g/H;
```

としたのは、同様にして、

```
ｂ＝ｂ－inv(H)*g;
```

としてもよいことがわかると思います。上のプログラム例では、

$$3x_1+5x_2+8x_3=1$$
$$2x_1+2x_2+9x_3=3$$
$$8x_1+5x_3+2x_2=6$$

といった一次の（線形の）計算をしていることになります。なお、行列Aに相当するものが、symmetricでpositive definiteであるならば、Ax＝ｂの計算は、

```
x=invpd(A)*b;
```

または、

```
x=solpd(b,A);
```

でも計算できます。組込み関数solpdの引数の順番は、上の一般的な行列計算の場合のb/Aの順序に同じです。上のAと同じ値を用いて、そのモーメントA'AをＡとおきなおして計算するには、行列の性質からこれらの組込み関数が使えますから、

プログラム

```
new; cls;
A={3 5 8,
   2 2 9,
   8 5 2};
b={1,
   3,
   6};
A=A'A;
print "A=" A;
x1=invpd(A)*b;
x2=solpd(b,A);
print x1;
print x2;
```

画面表示

```
A=
        77.000000       59.000000       58.000000
        59.000000       54.000000       68.000000
        58.000000       68.000000       149.00000

      -0.20443962
       0.30096985
     -0.017506390

      -0.20443962
       0.30096985
     -0.017506390
```

上の例では、行列Ａはsymmetricでかつpositive definiteではなくてはいけません。行列がsymmetricでかつpositive definiteであれば、Cholesky分解のアルゴリズムを用いるこれらの方法の方がより効率的であると言われていますが、計算速度にはさほど差がないことは以前示したとおりです。invpdおよびsolpdでは、たとえ行列Aに相当するものがsysmetricでなくても、上三角行列を見て計算してしまい答えも間違ったまま正常に返されるので、使用にあたっては注意が必要です。なお、モーメントx'xなどを扱う場合には、これらを用いても、その行列の性質から、問題はないでしょう。

二次形式(quadratic)の最大化問題を解くには、Qprogという組込み関数がGAUSSには標準で装備されています。ここで、二次形式というのは、目的関数が最大限x_1^2やx_1x_2などの都合２次で、制約関数が１次の線形である場合をさします。文法は、下の最小化問題

min 0.5 * x'Qx - x'R　　s.t.　Ax=ｂ，　Cx>=ｄ，　bnds[.,1] <= x <= bnds[.,2]

に対して、初期値ベクトルをstartとして、QProg(start,Q,ｒ,A,b,C,d,bnds)として組込み関数を用いて計算するものです。大文字の変数はは正方行列、小文字の変数は、変数の数だけの行数の列ベクトルとします。たとえば、x_1,x_2,x_3の３変数であるとすると、大文字のところは３×３の正方行列、小文字のところは３×１のベクトルになります。今、簡単化のためx_1とx_2の２変数の場合を考えてみましょう。行列とベクトルで表されているの少し複雑に感じますが、通常はｘ'Rの項はなくて、等号制約だけであるとするならば、例えばσを分散共分散、ｗをウエイトとして

$$\min \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \sigma_{ij}$$

$$s.t. \quad \sum_{i=1}^{n} w_i = 1$$

を満たすwのベクトルを求めます。上の GAUSS での設定で言うならば、Q には分散共分散行列がきて、その両側からwのベクトルをかけます。これによって行ｉと列ｊの自己同士を含むすべての組み合わせの２次項ができます。二分の１は計算上の都合からついています。３つのウエイトからなる場合、分散共分散行列は３×３のディメンションになります。等号制約のところは、A を１×３のすべてが１からなる行ベクトルとして、今求める３×１のウエイトベクトルとかけ合わせると、ちょうどウエイトの合計ができます。GAUSS では大文字も小文字も区別がありませんから、

Q＝分散共分散行列

A={ 1 1 1 }

b=1

としてやって、上のような２次形式の条件付最小化がなされるような３×１のｘベクトル（ここではwのこと）を求めます。その際、R は求めるｘベクトルと同じディメンションの零ベクトルである必要があります。不等号制約の係数および境界条件の係数は使わないので、すべてを０とします。いま、簡単化のため、分散共分散行列を対角成分がすべて１でその他の成分が０の単位行列とします（分散がすべて１、共分散がすべて０）。

プログラム

```
new;
cls;
Q={1 0 0,
   0 1 0,
   0 0 1};
R={0,0,0};
A={1 1 1};
b=1;
C=0;
d=0;
bnds=0;
start={1,1,1};
{ x,u1,u2,u3,u4,ret } = QProg( start,q,r,a,b,c,d,bnds );
print x;
```

画面表示

```
    0.33333333
```

0.33333333

0.33333333

なお、このQprogではx_1 x_2の条件付の最小なども最大２次と考えて計算できないこともありませんが、その場合にはそれぞれの変数独立の２次項がないのであまり適切ではなくて、その場合には次章で取り上げる一般的な次数の条件付最小問題を解くsqpで解きます。

ここでProf.Nerloveの例題にも取り上げられているHouthakker, H. S., The Capacity Method of Quadratic Programming, Econometrica, 28: 62 - 87 (1960)についてtypoを直して上の例題と関連付けて説明してみましょう。

Max [$18x_1+16x_2+22x_3+20x_4$

$-3x_1^2-x_1 x_2-8 x_1 x_3-5 x_2^2-x_2 x_3-4 x_2 x_4-8.5 x_3^2-3 x_3 x_4-5.5 x_4^2$]

s.t. $x_1>=0$, $x_2>=0$, $x_3>=0$, $x_4>=0$,

$5x_1+10x_3 <= 2$

$4x_2+ 5x_4 <= 3$

$x_1+x_2+x_3+x_4 <= 1.6667$

Ｑは最小化問題ではなくて、最大化問題であるから、すべての符号を逆転させてプラスにして、対角要素は２倍した値を、それ以外の要素はそのままの値を入れて対称行列になるようにする。同様にｒにはマイナスをつけるから式自体のマイナスと相殺されてすべての要素はプラスの符号になる。等号つき不等号の制約式は全部で７本。変数は４つなので、Ｃは７×４の行列、ｄは７×１の定数項の列ベクトルになる。上の４つの制約式は符号の向きが＞＝で一致しているが、後半の３つの制約式は符号の向きが逆なので、Ｃとｄのその部分の要素については符号を両方とも逆転させる。等号の制約式と境界制約はないので、Ａ、ｂ、bndsはそれぞれ０とする。なお、Prof.Nerloveの例題にはtypoが存在するので、ここでは、直してわかりやすい形に書きあらためています。

プログラム（Prof.Nerlove's basic26を書き改めたもの）

```
new; cls;
Q = { 6    1    8    0,
      1   10    1    4,
      8    1   17    3,
      0    4    3   11 };
r = { 18,
      16,
      22,
      20 };
A= 0;          b = 0;
C = { 1    0    0    0,
```

```
          0   1   0   0,
          0   0   1   0,
          0   0   0   1,
          -5   0   –10   0,
          0   -4  0   -5,
        -1   -1   -1   -1 };
    d = {0,
          0,
          0,
          0,
          -2,
          -3,
   -1.6667 };
    bnds = 0;

    start={0,0,0,0};
    {x, u1,u2, u3, u4, ret } = QProg( start, Q, r, A, b, C, d, bnds );
    print x;
```

画面結果

```
     0.40000000
     0.23308271
 2.0391589e-033
     0.41353383
```

ここで、３つ目のｘ$_3$に相当するものは、０が答えのはずで、近似で０になっている。このように、最大２次までの最小化問題と最大化問題はQprogを用いて計算できる。GAUSSは計量計算のソフトなので数値ベースの計算ではあるが、このQuadratic Programmingの関数Qprogは、その行列のパラメーターの設定の仕方が理解できれば、初等経済分析にも十分に利用できる機能である。（なお、説明をあえてとばしたu1,u2,u3,u4,retのリターン値は、それぞれ、等号制約のラグランジアンベクトル、不等号制約のラグランジアンベクトル、lower boundのラグランジアンベクトル、upper boundのラグランジアンベクトル、そしてretは０であれば計算成功、それ以外であれば不成功のリターンコードである。）不成功の場合は、たとえばiterationの最大値の値を大きくしてやる方法と、不等号の制約の代わりに等号制約を使ってやるなどの方法が考えられる。ちなみに、iterationのデフォルト値は、

```
    _qprog_maxit = 1000;
```

と設定されていて、このグローバル変数の値を変更とiterationの数が変わる。リターンコードが０ではなくて１の場合には、この値を大きくしてやる。

線形の連立方程式は行列計算で解けるが、非線型の連立方程式の場合はGAUSSでは、特殊なeqSolveという特殊な組込み関数を使うことになる。今、次のような非線型の連立方程式を考えます。

$3x_1^3+2x_2^2+5x_3=2$

$-x_1^3-3x_2^2+x_3=-2$

$3x_1^3+2x_2^2-4x_3=0$

これをprocではなくて簡単な方のfnの関数の１行定義でf(x)と表現してから、eqSolveで呼び出して計算させてみます。

プログラム

```
new; cls;
eqsolveset;
fn f(x)=3*x[1]^3+2*x[2]^2+5*x[3]-2|-x[1]^3-3*x[2]^2+x[3]+2|
                                                3*x[1]^3+2*x[2]^2-4*x[3];
x0={0,0,0};
{ x, retcode }  = eqSolve(&f,x0);
```

上のプログラムでは、Variable not initializedとエラーメッセージを出してうまくいきません。これはスタートベクトルx0が零ベクトルであるからであって、適当な別のもの、たとえば、{1,1,1}とでもすると答えは出て、リターンコードは１を返します。なお、f(x)は、x[1]、x[2]、x[3]といったインデックスを使って、＝０という形に変換した後、｜の記号で３つの式を垂直方向にマージしてあります。なお、eqSolveでは、この関数f(x)をそれを指し示すポインタの記号＆を用いて&fと表現して呼び出しています。

プログラム

```
new; cls;
eqsolveset;
fn f(x)=3*x[1]^3+2*x[2]^2+5*x[3]-2|-x[1]^3-3*x[2]^2+x[3]+2|
                                                3*x[1]^3+2*x[2]^2-4*x[3];
x0={1,1,1};
{ x, retcode }  = eqSolve(&f,x0);
```

画面表示

```
||F(X)|| at final solution:                                    0.56476122
-------------------------------------------------------------------------------
Termination Code = 1:

Norm of the scaled function value is less than __Tol;
```

```
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
VARIABLE             START                ROOTS              F(ROOTS)
--------------------------------------------------------------------------------
X1                 1.00000           -0.63327829       -7.6769884e-006
X2                 1.00000            0.90851353        2.5590134e-006
X3                 1.00000            0.22222222       -7.6769884e-006
--------------------------------------------------------------------------------
```

リターンコードが１であれば計算は成功していて、解はROOTSのところの３つの数が上からX_1, X_2, X_3ということになります。なお、eqsolveset;の行があるのは、この関数がライブラリのように数多くのグローバル変数をもっていて、その設定値を計算するたびに初期化していることを意味します。擬似ライブラリといっても過言ではないでしょう。

以前述べましたように、gradientなどの計算は、非常に有効桁数が限られたもので、手で計算したほうが正確です。そこで、ここでは、eqSolveのグローバル変数の１つとして、もとの式のgradientを手で計算して設定する方法があります。

プログラム

```
new; cls;
eqsolveset;
fn f(x)=3*x[1]^3+2*x[2]^2+5*x[3]-2|-x[1]^3-3*x[2]^2+x[3]+2|
                                                3*x[1]^3+2*x[2]^2-4*x[3];
fn g(x)=(9*x[1]^2)~(4*x[2])~5|(-3*x[1]^2)~(-6*x[2])~1|(9*x[1]^2)~(4*x[2])~-4;
_eqs_JacobianProc = &g;
x0={1,1,1};
{ x, retcode }  = eqSolve(&f,x0);
```

上のプログラムではJacobianジャコビアンを手計算で１つ１つ計算し、９個の要素を水平方向と垂直方向を組み合わせて３×３の行列にしてそれをfnの１行定義でg(x)としています。ここで気をつけたいことは、水平方向～と垂直方向｜のマージを使う際には、それぞれの項に演算記号を含んでいてはマージしてくれません。そこで１つ１つの要素を括弧で包んでマージの作業を行っています。それぞれをg11=9*x[1]^2; g12=4*x[2];...などとおいてから、マージすることも可能です。また、f(x)と同様にprocで関数を定義もできますが、無駄な変数が副産物として出てきますから、手軽に使えるｆｎの定義を用いたわけです。なお、上の例では、結果は以前のものとほとんどかわりはありません。

このeqSolveの関数には、数多くのグローバル変数があって、計算時に細かく設定できるようになっていますが、上のJacobianジャコビアンの手計算代入以外のグローバル変数設定の例を少し示しておきましょう。今、下の非線形の連立方程式を考えます。

$x_1^3 + x_2^2 = 2$

$\exp(x_1 - 1) + x_2^3 = 2$

プログラム

```
new; cls;
eqsolveset;
fn f(x)=x[1]^3+x[2]^2-2|exp(x[1]-1)+x[2]^3-2;
_eqs_MaxIters=10;
_eqs_IterInfo=1;
__Tol=1e-3;
x0={0,0};
{ x, retcode }  = eqSolve(&f,x0);
```

画面表示

```
--------------------------------------------------------------------------------
                         ROOTS                                F(ROOTS)
--------------------------------------------------------------------------------

Iteration #1
X1              0.00000004          F1(X)          -0.77929688
X2              1.10485435          F2(X)          -0.28342139


Iteration #2
X1             -0.27402802          F1(X)          -0.72070057
X2              1.14012130          F2(X)          -0.23828034


Iteration #3
X1              0.66787976          F1(X)          -0.53854814
X2              1.07867287          F2(X)          -0.02752514


Iteration #4
X1              0.81913321          F1(X)          -0.34864903
X2              1.04963309          F2(X)          -0.00904162
```

```
Iteration #5
X1                  1.05037642              F1(X)           0.14686811
X2                  0.99398072              F2(X)           0.03371752


Iteration #6
X1                  1.00266940              F1(X)           0.00709408
X2                  0.99953214              F2(X)           0.00127004


Iteration #7
X1                  1.00000802              F1(X)           0.00002150
X2                  0.99999873              F2(X)           0.00000420


================================================================================

||F(X)|| at final solution:                                    0.70710917
--------------------------------------------------------------------------------
Termination Code = 1:

Norm of the scaled function value is less than __Tol;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
VARIABLE             START                  ROOTS                F(ROOTS)
--------------------------------------------------------------------------------
X1                 0.00000                  1.000008          2.1504537e-005
X2                 0.00000                0.99999873          4.2006902e-006
--------------------------------------------------------------------------------
```

まず、前の３変数３方程式の例と同じく、＝０の形に直したものを、fnで関数を定義して｜のしるしで垂直方向にマージしてf(x)を作ります。この関数ｆをポインタのしるし＆をつけて、スタートベクトルとともに、eqSolveに呼び出すわけです。その前に、グローバル変数の設定をデフォルトから任意の値にかえます。特にリターンコードが１以外の場合には、Jacobianの手計算代入とともに、この作業が必要になります。まず、iterationの数は

```
_eqs_MaxIters=100;
```

でデフォルト100に規定されています。もし、リターンコード４、Iteration limit exceeded とされれば、この値をさらに大きくする必要があります。このiterationの逐次の様子を表示するのには、

```
_eqs_IterInfo=0;
```

とデフォルトで０となっているものをそれ以外の数、たとえば１にすると、画面表示で長々と示されているように、すべてのiterationの様子が示されます。また、通常の許容桁数を変更しようと思えば、

```
__Tol=1e-5;
```

の小数点第５位のところの桁数を複雑な式の場合にはJacobianの数量的計算方法では精度が落ちますから、これよりも少し大きな値、たとえば1e-3にすればよいでしょうし、Jacobianが手計算で与えられている場合には、この数をもっと小さな値、たとえば、1e-7とかのようにすればよいでしょう。これらの作業や、スタートベクトルを違う数にすることによって、正常に答えが出ることも多々あります。なお、__は２つの連続したアンダーラインです。上の例で言えば、__Tol=1e-3;に設定することによって、iteration6のところでF(ROOTS)の値が絶対値で0.001以上であったものが、iteration7では、それ以下になってそこで計算は終了しています。

繰り返しますが、これらはすべて組込み関数であって、ライブラリではありません。しかしながら、ライブラリに近いような役割や体裁をとっているものではあります。