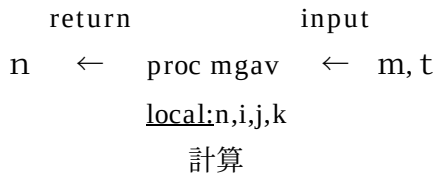
3.5 プログラミング proc(n) ver. 0.2

前回のprocのところで扱ったプログラムのモジュール化は、リターンが１変数（行列としての１変数）のもっとも使われるケースですが特殊なケースでした。移動幾何平均のprocであるmgav(m,t)について、その仕組みを図解すると下の図のようになります。



この場合、return戻り値はnだけの１つです。GAUSSでは、この１個のreturnがprocの外部のプログラムで用いられたり、ほかのprocのinputに引き渡されたりします。そういう意味で、しばしばプログラムでは

```
proc(1)=mgav(m,t);
    local n,i,j,k;
    ..............
    retp(n);
endp;
```

のように書かれることもあります。この(１)というのはreturnの数が１個という意味です。なお、returnが２個以上の関数は、そのretrunの数だけの数字をここに書きます。もし、retrunがない場合には(０)を書きます。それぞれのreturnの数のケースについて例をあげて説明していきましょう。

リターンが０個の場合

ここでいうretrunの数が０個というのは、動作をしないまたは画面表示をしないという意味ではなくて、procの外部に受け渡す変数がないという意味です。主に、画面表示やフォーマットに関することだけをすることに使われます。条件分岐ifのところでとりあげたｔ値による棄却決定のプログラムをprocedureに変更した上で、この０リターンのケースとして扱ってみましょう。

プロシジャー

```
proc(0)=tdecision(t,df,pp);
    local p;
    p=2*cdftc(t,df);
    if p<=pp;
        print /rz "Reject H0: beta=0 at" pp*100 "% level";
```

```
        else;
            print /rz "Do NOT reject H0 at" pp*100 "% level";
        endif;
        print "|t|<" p;
    endp;
```

上のような、どんなプログラムにもあとで繰り返し利用できるprocedureが完成しました。見てわかるように、retp文がありません。外部にreturnとして受け渡す変数がまったくないということをこのことは意味します。なお、このプログラムだけでは動きません。procの外部で引数のインプットに何か数値を与えなければなりません。一番簡単な方法は、おのおのの引数に直接数値を代入して、

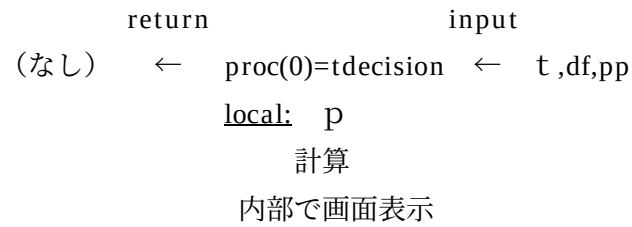
```
call tdecision(1.94,4,0.1);
```

とそのprocedureの外部に書いてやれば、以前と同じように次のように画面表示します。

画面表示

```
Do NOT reject H0 at                    10 % level
|t|<          0.12437736
```

このように、proc(0)のリターンの数が０個というのは、print文などでいくつの行にわたって出力されようとリターンの数には関係がまったくありません。このretp文がないことを、上の場合について図で表現すると以下のようになります。



ただし、この０個のリターンのprocの呼び出し方は、call文で呼び出すか、またはそのままその命令文を書くかのいずれかでなければなりません。１個のリターンと同じように何か別の変数に代入したり、print文の内容にしてはいけません。なぜなら、その関数自体にリターンがないと設定したからです。

リターンが１個の場合

これは、前回扱ったrndx2、mav、mgavの３つともにあてはまります。冒頭に書きましたように、いずれの関数もproc(1)=という形で書きなおしても同じことになります。以前やったように、d:ドライブのdatafile1.txtというデータファイルを読みこんで、OLS estimator

であるｂを求めるプログラムをリターン１個のprocedure形式になおしてみましょう。行列方式でOLSを求めるには、

$$b = (X'X)^{-1}X'Y$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ \cdot \\ \cdot \\ y_{27} \\ y_{28} \end{pmatrix}, X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & x_{27} \\ 1 & x_{28} \end{pmatrix}$$

の関係を用います。ｙ切片がある場合には、ｘのデータの１列目をすべて１に設定する必要があります。ｘが多変数になっても同様にして切片と係数の推定ができます。なお、この場合できるｂは( )$^{-1}$の部分が２×２、そのあととあわせて(２×２)×(２×28)×(28×１)ですから、２×１になるはずです。ｂの第１行目は切片の推定値、その２行目はXの係数の推定値になるはずです。

プロシジャー

```
proc(1)=lse(y,x);
    local b;
    b=inv(x'x)*x'y;
    retp(b);
endp;
```

同様に、

```
proc lse(y,x);
    local b;
    b=inv(x'x)*x'y;
    retp(b);
endp;
```

と書いても同じことになります。上ののプログラムはprocedure本体だけですから、これの引数インプットに数値または行列を与えてやった上で、戻り値を操作するなり画面表示するなりしないと何も動作はしません。いま、d:ドライブのdatafile1.txtというデータを読み込んで、その２行目から29行目までのデータを用いて以前やったように、regressionをしてみます。データの１列目をｙ、２列目をｘとして、さらにｘには切片に対応する１からなる列ベクトルを水平方向につけて扱います。

プログラム

```
new; cls;
load data[29,2]=d:datafile1.txt;
y=data[2:29,1]; x=ones(28,1)~data[2:29,2];
bhat=lse(y,x);
print "coefficients:" bhat';

proc(1)=lse(y,x);
    local b;
    b=inv(x'x)*x'y;
    retp(b);
endp;
```

画面表示

```
coefficients:          77.795198          52.009829
```

上のプログラムでは、後半のprocedure lse(y,x)を呼び出すのに、ｘとｙに数値を代入したあとの関数lse(y,x)の戻り値の値をbhatとおいて、それをcoefficinets:というメッセージつきで画面表示させています。戻り値は、procedureの中ではｂという変数で扱われて、それが戻り値としてリターンされますが、このように名前の違う変数で受けることも可能です。ただし、ｂというローカル変数として宣言した値を直接扱ったり画面表示させるためのprint文の内容にはなることはできません。すなわち、

プログラム（間違った例）

```
new; cls;
load data[29,2]=d:datafile1.txt;
y=data[2:29,1]; x=ones(28,1)~data[2:29,2];
print b;
……….
```

ただし、b=lse(y,x);と関数の戻り値を一度ｂという変数に受けて代入すれば、print b;で画面表示できます。という意味で、ローカル変数で宣言された変数は、それがリターンであろうとなかろうと、外部から直接扱うことはできません。ここのところは、誤解のないようにしてください。もっとも、直接関数をそこに置くことによって、lse(y,x);とすれば、その戻り値、すなわちここではｂの内容２行１列で画面表示されます。print lse(y,x);とした場合と同じ結果が得られます。このように、リターンが１個の場合の呼び出し方は、何かの変数に代入すること、または、そのステップを省略して画面表示させるためprint文の内容にすることができます。もしcall文でリターンが０個のように呼び出すと、リターンの変数の引渡しはされません。procの内部に外部への変数引渡しとは別に画面表示やその他のシステム制御に関する命令があれば、その部分だけを実行します。なお、関数そのものだけを書いてやると、その関数の引数の内容が画面表示されます。call文の役割としては、proc

の内容だけを実行させるかわりに、戻り値を表示したくない場合によく用いられます。

ポイント　call プロシジャー名;　　procの内容だけを実行して戻り値を表示しない

リターンの値が１個の場合について、callでprocを呼ぶと、その内容だけを実行して戻り値を表示しない例を示してみましょう。係数βの推定とは別に、proc内部でresidual plotをする部分をlse(y,x)に付け加えてみます。ここでresidual残差には以下の関係を用います。この残差は、Yと同じディメンションの28×１の列ベクトルになるはずです。

$$\hat{u} = Y - \hat{Y} = Y - Xb$$

ここで、$b = (X'X)^{-1}X'Y$

プロシジャー

```
proc(1)=lse(y,x);
    local b,uhat,xaxis;
    b=inv(x'x)*x'y;
    uhat=y-x*b;
    xaxis=seqa(1,1,rows(x));
    uhat=uhat~zeros(rows(x),1);
    library pgraph;
        graphset;
        xy(xaxis,uhat);
    retp(b);
endp;
```

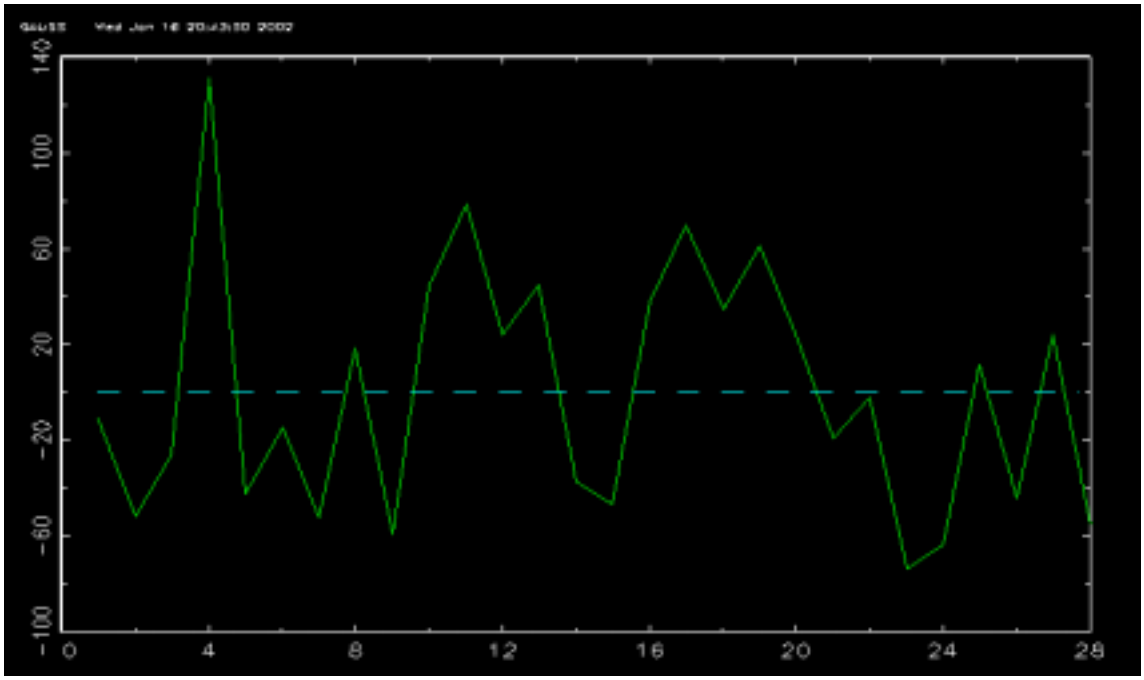
プログラム

```
new; cls;
load data[29,2]=d:datafile1.txt;
y=data[2:29,1]; x=ones(28,1)~data[2:29,2];
call lse(y,x);

proc(1)=lse(y,x);
    local b,uhat,xaxis;
    b=inv(x'x)*x'y;
    uhat=y-x*b;
    xaxis=seqa(1,1,rows(x));
    uhat=uhat~zeros(rows(x),1);
    library pgraph;
```

```
        graphset;
        xy(xaxis,uhat);
    retp(b);
endp;
```

グラフ表示



上のように、call文で１リターンのprocを呼び出しているので、その戻り値はプログラム上では何もせずに、procの内部のグラフを描くところが実行されています。太字でしめされている前のプログラムにさらに付け加えられた部分では、まず、uhat=y-x*b;でデータｙとその推定値との差、すなわちresidual残差を求めています。これは、xとｙの行数と同じ28行１列の列ベクトルです。これをｘｙグラフにプロットするわけですが、その準備に、ｘ軸として１から28までの数列xaxisを用意します。一方、ｙ軸にはこの残差uhatにすべてが０ばかりのベクトルを水平方向にマージしたものをuhatとしてやります。グラフを描く時の手順であるlibrary pgraph;でpgraphグラフライブラリを呼んでやって、graphset;でそのグローバル変数を初期化しています。ｘｙグラフの関数に、準備したxaxisとuhatを射れてやれば、実際の残差のプロットを結んだ線と０の基準線が描けます。これは、ｘｙグラフのグローバル変数のデフォルトがプロットを結ぶように設定されているためです。そのほかのプログラムの部分は、以前の係数βだけを求めるprocと同じ作りになっています。ただし、localのあとのローカル変数の宣言では新たに加えた変数uhatとxaxisを書かないと、Undefined symbolとエラーが出てしまいますので注意が必要です。

リターンが２個の場合

例えば、リターンが２個のprocの作り方の要領は、proc(2)＝としてやって、retp文の丸括弧の中にカンマで区切った２つの変数を入れるだけで、そのほかの部分はprocの組み立て方とまったく同じです。引き続きregressionのデータとプログラムを用いて、それを拡張していきます。まずは、リターンが２個の場合、係数βのほかに、推定式全体のstandarnd error（なお、これは係数それぞれのseとは別物ですのでご注意を）を求めてあわせて表示してみます。

プロシジャー

```
proc(2)=lse(y,x);
    local b,uhat,n,k,s2hat,shat;
    b=inv(x'x)*x'y;
    uhat=y-x*b;
    n=rows(x);
    k=cols(x);
    s2hat=uhat'uhat/(n-k);
    shat=sqrt(s2hat);
    retp(b,shat);
endp;
```

プログラム

```
new; cls;
load data[29,2]=d:datafile1.txt;
y=data[2:29,1]; x=ones(28,1)~data[2:29,2];
{b,shat}=lse(y,x);
print "coefficients:" b';
print "Std error of est:       " shat;

proc(2)=lse(y,x);
    local b,uhat,n,k,s2hat,shat;
    b=inv(x'x)*x'y;
    uhat=y-x*b;
    n=rows(x);
    k=cols(x);
    s2hat=uhat'uhat/(n-k);
    shat=sqrt(s2hat);
    retp(b,shat);
endp;
```

画面表示

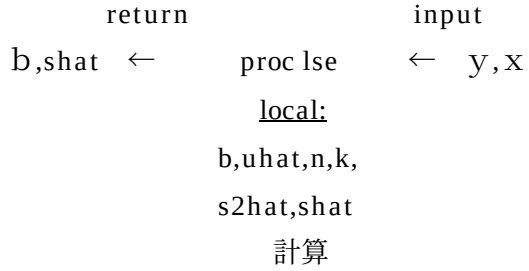
coefficients: 77.795198 52.009829

Std error of est: 52.331001

上のプログラムで、proc部分では、これまでのプログラムに加えて、データ行列ｘの行数をｎ、その列数をｋとおいて、s2hatをuhat'uhat/(n-k)で求めます。ここでuhat'uhatの部分は、(１×28)×(28×１)ですから１×１のスケラーになっています。これをスケラーであるn-kで割っています。ちょうどこれは行列表示の

$$\hat{\sigma}^2 = \frac{\hat{u}'\hat{u}}{n-k}$$

の計算に相当する部分です。この平方根をとって、Estimated regression standard errorを求めています。shatにその値を代入して戻り値の２つ目の値としています。なお、よくこの(n-k)の括弧を忘れてしまうことがあるので、基本的なことですが毎回確認作業をお忘れなく。GAUSSでも数学とおなじように掛け算と割り算は優先されます。前のプログラム同様、加えたローカル変数であるn,k,s2hat,shatを忘れずに宣言することと、procの後の数字がリターンが２個ということで２にすると同時に、retpの丸括弧の中にｂとshatの２つの変数を入れます。これでリターン１個から２個への変更は完了しました。プログラムでは、このprocを前方参照して、ｂとshatの２つを画面表示させています。最大の関心事は２つ以上の戻り値の場合どうするかということです。「変数＝」の形では受けられないことは想像にかたくありません。行列の関数のところで扱った２変数以上にわかれて関数の値がでてくる場合と同じように、{ 変数名A, 変数名B }=の形で受けます。すなわち、行列のところで扱った２つ以上の値がでてくる関数も実はこのprocの２以上のリターンの構造でできていたのです。３つ以上のリターンになっても、{ 変数名A, 変数名B, 変数名C ….}=の形で受けて、計算結果である戻り値をprocの外部に取り出します。リターンが１個の場合と同じく、{ }括弧内の変数名はprocから返される変数名と同じ必要は必ずしもありません。ただし、同じ数だけ変数は必要です。複数個procを連ねて変数を受け渡していく場合には同じ変数名であった方がプログラムがすっきりとして見やすくなります。なお、引数インプットとリターンとの関係は、この場合、下の図のようになります。



リターンが多数の場合

上のプログラムをすすめて、基準統計量以外の、OLSから求められる基本的な情報を得

るプロトタイプとも言えるprocを作っておきましょう。推定式のstandard errorおよび、切片と係数のほかに、それぞれのstandard errorとそれに対応するｔ値、R2を求めます。

プログラム

```
new; cls;
load data[29,2]=d:datafile1.txt;
y=data[2:29,1]; x=ones(28,1)~data[2:29,2];
{b,se,t,shat,df,r2}=lse(y,x);
print "y-intercept:" b[1] "          slope for x:" b[2];
print "          se:" se';
print "           t:" t';
print "Std of est :" shat;
print /rz "          df=" df;
print /rz "          R2=" r2;

proc(6)=lse(y,x);
    local b,uhat,n,k,df,s2hat,shat,varb,se,t,r2;
    b=inv(x'x)*x'y;
    uhat=y-x*b;
    n=rows(x);
    k=cols(x);
    df=n-k;
    s2hat=uhat'uhat/df;
    shat=sqrt(s2hat);
    varb=s2hat*inv(x'x);
    se=diag(sqrt(varb));
    t=b./se;
    r2=1-uhat'uhat/(y'(eye(n)-1/n*ones(n,1)*ones(n,1)')*y);
    retp(b,se,t,shat,df,r2);
endp;
```

画面表示

```
y-intercept:       77.795198          slope for x:       52.009829
         se:       22.037400                             3.8317273
          t:       3.5301442                             13.573468
Std of est :       52.331001
         df=              26
         R2=      0.87633124
```

上のプログラムでは、まず後半のプロシジャーのところでdfをn-kと定義してやって、n-kではなくてdfで割ることによってs2hatを計算しています。ｂについてのVarCov行列は,このs2hatと$(X'X)^{-1}$をかけ合わせることによって求めます。これは、行列表示で

$$Var(b) = \hat{\sigma}^2 (X'X)^{-1}$$

に相当するものです。計算結果は２×２のディメンションになります。その対角要素が係数（正確には切片と係数の両方）のVarianceとなっています。

$$Var(b) = \begin{pmatrix} Var(b_1) & Cov(b_1, b_2) \\ Cov(b_2, b_1) & Var(b_2) \end{pmatrix}$$

したがって各係数のstandard errorを求めるには、上の計算結果のdiagonal対角成分の平方根をとることによって計算します。ｔ-ratioは係数ｂ／seの割合ですから、ｂ割るseを要素対要素の計算で求めます。そういうわけで、b./seというぐあいにドットがついています。

$$t_i = \frac{b_i}{se_i}, \qquad i = 1,2$$

これは割る方の数seがベクトルであるためです。GAUSSでは、割る数やかける数がスケラーの場合は、実際の数学と同じようにディメンションを考える必要もなく割ったりかけたり、行列のおのおのの要素対スケラーで“／”のマークで計算できます。しかしながら、割ったりかけたりする方の数がベクトルや行列の場合には、要素対要素の計算であることをしめすのにドットが必要です。（一部の言語には、さらにこれよりも厳しくスケラーの割り算にまでドットを要求するものがあったり、その反対にまったくドットを要求しない言語も統計計量のソフトにはありますので注意が必要です。GAUSSは「要素対要素」の場合のみドットが必要で、「行列対スケラー」の場合にはドットは不要です。）R2の計算では、行列表示形式では、

$$R2 = 1 - \frac{\hat{u}'\hat{u}}{y'(I_n - \frac{1}{n} ii')y}$$

を用いて計算しています。ここで

$$i = \begin{pmatrix} 1 \\ 1 \\ . \\ . \\ 1 \end{pmatrix}$$

のｎ×1の列ベクトルです。

もちろん、行列表示ではなくて、プログラムは複数行にまたがりますが、要素ごとに、

$$R2 = 1 - \frac{\sum_{t=1}^{n} \hat{u}_t^2}{\sum_{t=1}^{n} (y_t - \bar{y})^2}$$

を計算しても計算できます。同じことです。そのほかの部分は同じで、ローカル宣言のところに新たに加えたdf,varb,se,t,r2を書くことをわすれないでください。リターンとしてretpの丸括弧の中にさらにse,t,df,r2を追加して、これまでのbとshatの２つに加えて６つとなりますから、先頭のprocのところもproc(6)に変更します。

呼び出し方は、{　　} 括弧の中にリターンとすべき変数５個すべてをカンマで区切って書いてイコールサインで関数と結びつけ、その関数の引数の中にデータまたは変数を代入すれば関数を呼び出せます。{b,se,t,shat,df,r2}=lse(y,x);のようになっているはずです。そのあとで、{　　} 括弧の中そのおのおのの変数をprint文で“　　”のコメント付で画面表示させています。なお、{　　} 括弧の中にはprocのリターンの個数分変数を書かなくてはなりませんが、変数名は任意です。わかりやすいプログラムという点から慣習上、リターンと同じ変数名が使われることが多いようです。ただし、これらすべてのリターンをすべて利用したり画面表示する必要はありません。例えば、shatの画面表示行を消したり/*　　*/でくるんだりしてコメント行として一時的に実行させないようにしても、何ら支障はありません。このように、公開されているprocの一部のリターン変数だけを利用してプログラムを組み立てることも可能になってきます。

最後に上のプログラムの構造は、おわかりのように、次のようになっています。

```
        return                    input
b,shat, ←      proc lse     ←   y,x
df,se,t,r2       local:
                 b,uhat,n,k,
                 s2hat,shat,
                 varb,df,t,se,r2
                   計算
```

ポイント　proc(n)=**関数名(引数**,…);
　　local **変数**,…,**変数**;
　　計算部分
　　retp(**変数**A, **変数**B, **変数**C,….**ｎの個数だけ**);
　endp;

ポイント　{**変数**A, **変数**B, **変数**C,….n**の個数だけ**}=**関数名**( )
　　というふうに{　　}**括弧の中に変数を書いて複数リターンを呼び出す。**