X.8 Stata/module to GAUSS の基礎　　ver.0.1

Stata でも未装備の新しい関数を作るためにプログラムが可能です。ここではそうして作られて公開されているプログラムを GAUSS にもってくる際の文法事項を説明します。

コメント

Stata/module	GAUSS
* This is a comment.	@ This is a comment. @
/*　Comment　*/	/*　Comment　*/
*!　Fixed 04 Nov 2002	

Stata では行の先頭に＊印がある行または、GAUSS と同じように/*　*/で囲まれた部分がコメント行になり、実際の実行の際には無視されます。GAUSS では@　@で囲まれた部分もコメント行になります。ただし、＊！から始まる行は、which コマンドによって、

which プログラム名

とすることで情報が呼び出せるようにできる部分です。通常は、作成日や修正記録などの重要な情報が書かれます。

Ado Module プログラミング

Stata/module	GAUSS
program define name1,rclass	proc name1(a,b);
version 7.0	
args varname a b	
	local mm,nn;
. . . . .	. . . . .
return local MM=`変数名'	
return local NN=`変数名'	retp(mm,nn);
end	endp;

コマンドの集まりのログファイルの Ado ファイルではなくて、手続きのプログラムは、

program define 手続き名, オプション

の体裁をとり、通常は最後は end または end と exit から成ります。オプションのところには、rclass などのクラスの概念が来ます。GAUSS への移植の観点から言えば、この概念は無視できます。一方、GAUSS では procedure でプログラムされ、endp;までが１つの固まりですが、この固まりは１つである必要はなく、プログラムファイル上に複数の procedure の固まりを同時に置けますし、相互に関数として利用できます。Stata では version を指定することもでき、通常は program 文の次に置かれます。7.0 とか 6.0 に通常はなります。それがないものは、その手続きはバージョンを指定しないということになります。Stata ではインプットはワークシートの作業領域での変数名を用いてプログラムされていきます。明示的なインプット宣言は args で行ないます。GAUSS では procedure 名の隣に丸括弧の中にカンマで区切ってすべてのインプット変数を並べて明示します。この手続きを呼び出す場合には、

Stata: name1 hensu 1000 5

GAUSS: call name1(1000,5);

となります。GAUSS では変数の指定はインプットとしてはありません。Stata では args で指定された`varname', `a', `b' はすべてマクロ名と判断されます。上では、`varname', には name1 が、`a', には 1000 が、`b' には５が割り当てられて、name1 という手続きが呼び出されていることになります。GAUSS でも varname に相当するところを何かの表示や条件分岐で使う際には、通常どおりのインプット引数、例えば hensu と丸括弧にカンマで区切って proc name1(varname,a,b);と書いて、呼び出す時には、

call name1(“hensu”,1000,5);

とすることもできますが、Stata とはまた意味が違うものになります。手続きのリターンについては、Stata も GAUSS もグラフ表示だけなどのリターンを伴わない場合には省略できます。GAUSS ではリターンのない場合またはリターンが複数の場合、

proc(3)=name1(a,b);

proc(0)=name1(a,b);

などというふうに、リターンの数と冒頭の括弧の中の数字は一致する必要があります。１個のリターンの場合には１と書いても例のように書いてもどちらでもかまいません。Stata では、リターンのある場合だけ、return 文自体をリターンの個数文だけ書く必要がありあります。また、その場合、通常そのリターンに対しても逐一ローカル宣言します。GAUSS ではリターンもリターンでない変数もまとめて冒頭２文目で local 文を用いて列挙します。Stata ではローカル変数はその変数を使うごとにその都度プログラムの各行で宣言します。また、リターン文は必ずしもローカル宣言で書く必要はなく、scalar でもかまいません。ただし、リターンがある際には、通常は冒頭でカンマに続いて rclass とクラスオプションをつけます。

行について

GAUSS には行の概念がありません。すべてセミコロン；までが１つの命令です。Stata は、何も冒頭で宣言しなければ、べた書きしていきますが、

```
#delimiter ;
```

とすれば、；を最後につけるタイプに変更できます。もとの Enter を押すタイプの区切りは

```
#delimiter cr
```

とすれば戻ります。

行数

Stata/module	GAUSS
id=_n	id=seqa(1,1,rows(data))
nt=_N	nt=rows(data)

Stata では_n は data のそれぞれの行の行番号を表します。一方、大文字の_N は data のトータルの行数を表します。すなわち、data が７つ（の行）から成っているとすれば、_n は 1,2,3,4,5,6,7 になり、_N は 7,7,7,7,7,7,7 となります。

AND と OR

Stata/module	GAUSS
&	and
|	or

Stata では、＆は AND の意味、｜は OR の意味である。特に、｜は GAUSS では垂直方向のマージであるので注意してください。

行列計算

Stata/module	GAUSS
matrix `XY' = `X' * `Y'	xy=x*y;

Stata では、行列の計算をする際には、特別に、行列指定してやらなければなりません。通

常は、matrix またはその省略形 mat の後に計算式を置きます。GAUSS は、もともとがすべての変数は行列と見なされますからそのまま書いてやります。行列の操作ではなくて、通常の Stata のコマンド操作と同様に変数を作成するのならば、generate または gen で行なうか、スケラーの場合には、scalar を使います。GAUSS では、区別はありません。

ローカル変数設定

Stata/module	GAUSS
local i = 1	i=1;
local i 1	
local a `b'	a=b;
local a=`b'	

Stata では、変数の設定（上ではローカル変数の場合）には、イコールサインで後ろのものを前に定義する方法だけでなく、イコールサインを省略する方法もあります。ローカル変数ではなく、グローバル変数の場合は、local のところを global に変更します（GAUSS では、グローバル変数は意味が違っていて、ライブラリのパラメータをグローバルと呼ぶ)。

If 文

Stata/module	GAUSS
if `変数' {	if 変数;
式	式;
}	
else {	else;
式	式;
}	endif;

Stata でも GAUSS でも if 文の構造は同じですが、Stata ではある変数が真である場合に、それに続く計算を行なうのであれば、{ }で囲まれた部分を計算します。そうでない場合には、else に続く{ }で囲まれた部分を計算します。一方、GAUSS では、{ }括弧はデータの代入に用いられて、プログラムの固まりの区切りとしては用いられません。そのかわりに、endif;で必ず止めます。Else 文がある場合には、else にも単独でセミコロンをつけて、それよりも以前が if 文で真である場合の計算すべてにあたり、そうでない場合は、

else;よりも後 endif;よりも前の部分すべてが計算されます。変数の真偽ではなくて、式の評価や同一であるかの評価をする場合には、Stata では上の`変数'の部分に式の等号不等号などの評価式（またはその組み合わせ）をおいて

if (式の評価) {

というふうな形になります。丸括弧はつけない場合もあります。GAUSS では if の後に式を書くだけです。セミコロンまでが条件式と見なされます。＝＝でもって同一であるか判断するところは Stata も GAUSS も共通です。

While ループ

Stata/module	GAUSS
local i = 1	i=1;
while `i' <= `a' {	do while i<=a;
式	式
local i = `i' + 1	i=i+1;
}	endo;

何かａの値が与えられていて、i=1 からカウンタを１ずつ増してそこまでループを回すには、上のようになります。Stata ではカウンタにもその都度ローカル宣言をしなければなりませんが、GAUSS の場合もカウンタで使う変数も含めて冒頭で local 文で一括宣言します。また、Stata では、**`1'**の中味が空ではないうちはループを回す意味に、

```
while "`1'" ~= "" {
                式
                macro shift
                }
```

とする方法もあります。～は Stata では NOT の意味で、～＝で GAUSS の／＝に相当します。GAUSS では、記号～は水平方向のマージを表します。

上の例では、Stata によって自動的に作成されるローカルマクロの**`1'**を文字列として中味が空であるかどうかを比べています。もし、**`1'**と空の””を比べたなら意味はありません。GAUSS においても” “の中は文字列として扱いますがこういった用法はありません。また、よく出てくる**`1'**や**`2'**といった数字からなるローカルマクロは、Stata においてコマンドを使ったときに順に自動的に割り当てられるものです。もし最初に、何でも言いのですが、例えば、　list a b c

とすればローカルマクロ１には a が、２にはｂが、３にはｃが自動的に入っていきます。したがって、Stata の動きがよくわからないで自分でプログラムする際には、数字からなる

変数を使うべきではありません。また、前述のローカル宣言をするマクロは、scalar 宣言をする数値よりも 16 桁に対して 12 桁と計算精度が落ちます。したがって、このことをよくわかっていない公開されているプログラムを移植して GAUSS と比べても、結果は微妙に異なるものになる可能性があります。

組み込み関数

Stata/module	GAUSS
ln(`x')	ln(x)
inv(`x')	inv(x)
exp(`x')	exp(x)
sum(`e')	sumc(e)

Stata のコマンドをそのまま module のプログラムの中でも使えるのですが、計算上の約束は、GAUSS とそのほとんどで共通しており、違うものでも容易に類推できます。

画面表示

Stata/module	GAUSS
display “This is a comment.”	print “This is a comment”;
display "x = `x' "	print “x=” x;

Stata では display またはその省略系の di が画面表示のコマンドです。ただし、変数マクロとコメントの両方を表示する場合には、文字列の” “の中に変数マクロを入れることができます。GAUSS では引用符の中のコメントと変数は別に列挙します。

Stata における特殊なプログラミングルール

1） Stata の通常の操作コマンドをそのまま利用できるが、その場合と同様、di や su や loc などというふうに、先頭の 2 または 3 文字の省略形が使われる傾向にある。

2） `X'と”`X'”それから X 自身の区別をつけなくてはいけない。それは、`1'と”`1'”それから 1 自身の区別に典型的に表れてくる。

3） オプションで指定すれば、クラスの概念がある。したがって、r や e などの丸括弧のところを何かの組み込み関数であるととらえてはいけない。

Reference

Info about the Stata Training Movies

http://www.louisville.edu/~wrrisi01/Stata/Movies/index.html

QuickTimeMovie での Stata のコマンドについての講習については上記を DSL 以上の回線で見るとよくわかる。日本のユーザー研究者は直接人から Stata を習ったことがないので、誤解や教えている人の独断でゆがめられているが、一度すべてを見ることを推奨する。

だだし、ソースの coding についての詳しい説明は、残念ながら Material は Internet 上にも現実空間にも極めて乏しい。Bocode 等の例を細かく見ていくほかはない。