# X.1 RATS to GAUSS の基礎　　ver. 0.1

この X 節では、計量統計で使われる各種プログラムソフトから GAUSS へのコンバージョンの基礎的な Tips を解説します。もちろん、各言語には独自の記述法やファイル操作の作法があるので完全に一致させることは不可能かつ非効率です。しかしながら、数種類の言語に無秩序にリソースが分散して、互いに互いを知ることができないのは不幸なことであるばかりか、間違いは永遠に間違いで残り続け、移植を独自開発であると主張する輩が存続し続けることになります。ここでは、別の言語から GAUSS への方向だけの、GAUSS の土俵に立った立場からの、最低限知っておくべき事項を解説をしていきます。

画面表示

| RATS | GAUSS |
|---|---|
| display a | print a; |
| display 'OK' | print "OK"; |

RATS は GAUSS と同じ表示を丸括弧なしに直接設定するタイプで、display ということばを使います。RATS は 1 行ごとに命令がなされますから命令を区切るセミコロン;はつけません。GAUSS には行の概念はありませんから、命令ごとにセミコロン;をつけることが必須になります。RATS では文字列はシングルの引用符 ' ' に包むのに対して、GAUSS ではダブルの引用符 " " に包みます。

コメント行

| RATS | GAUSS |
|---|---|
| * This is a comment. | /* This is a comment. */ |
| | @ This is a comment. @ |

RATS では最初に＊のマークのついた行はコメント行になります。一方、GAUSS では、行の概念はなく/* から*/まで（または@から@まで）の部分がコメントの部分になり、その部分は実行されません。

OLS

| RATS | GAUSS |
|---|---|
| linreg y | { vnam,m,b,stb,vc,stderr,sigma,cx,rsq,resid,dwstat } |
| # constant x1 x2 | =ols(0,y,x1~x2); |

RATS では、linreg の後に従属変数をもってきます。別の行に、＃のあとに独立変数をいくつか並べます。定数項がある場合には、constant と書きます。RATS では、＃のマークは、そのほかに、制限のテストや各種設定をする際に用います。一方、GAUSS は、通常は行列計算で係数や t 値などすべての統計値を計算します。ごくまれに、ols という組込み関数を用いて計算することがありますが極めて少数派です。ちなみにプログラムは、

```
x=ones(rows(y),1)~x1~x2;
n=rows(x); k=cols(x);
b=inv(x'x)*x'y;
e=y-x*b;
s2hat=e'e/(n-k);
varb=s2hat*inv(x'x);
se=sqrt(diag(varb));
t=b./se;
```

などというように、GAUSS では行列でその都度計算していくのが一般的です。

数値定義と計算処理

| RATS | GAUSS |
|---|---|
| set trend=t | t=seqa(1,1,rows(data)); |
| set dy=y-y{1} | dy=y[2:n-1]-y[1:n-1]; |
| set rss=%resids**2 | e=…; rss=e^2; |
| compute lm=%nobs * %rsquared | r2=…; lm=rows(data)*r2; |

RATS では、数値を設定したり簡単な計算を変数に代入する際には、set 何々として変数と数値または予約語をつなぎます。複雑な計算には、compute 何々とします。それに対して GAUSS では、すべての計算が行列として処理されるため、数値設定であろうと計算であろうとただ単に「変数＝」の形に形にするだけです。RATS の方は、あらかじめ予約された変数や機能に基づいて数値を設定したり計算をしたりすることになります。上の例では、 t

はトレンドを表し、１からデータの個数分だけ順に数が割り振られて、その列を trend という変数に代入設定しています。次の y{1}という部分は RATS 特有の表現で、ｙという変数がデータの１つの系列であるとすると、{ }の中の数だけのラグと言う意味です。RSS を求めるところでは、最後の LM 統計量を計算するところの%nobs や%rsquqred とともに、あらかじめ linreg という OLS が行なわれたとすると、その結果の予約語をもとに計算をしています。RATS では、%のマークのついた変数は OLS などの推定結果の予約語、%のマークがついていて後に括弧がついているのは組込み関数です。RATS はパッケージとして推定計算を一度行なった結果の予約語をこのように使ってさらに細かい計算を補助的に行なうことをよくします。一方、GAUSS はトレンド１つ作成するのにも、データの行数から計算してｎを計算して、それに基づいて数列を作成しなくてはなりません。ラグは、GAUSS でも lag1 や lagn という組込み関数を用いて、lag1(y)や３期のラグなら lagn(y,3)とできます。ただし、GAUSS はパッケージソフトではないので、計算のときにラグにともなって生じた欠落値をそのまま放置されるので、trimir(z,1,0)などとしなくてはなりません(z という行列の上から１行、下から０行カットするという意味)。このようなプログラム方もありますが、実際的には、組込み関数に頼らずに列をずらして y[2:n-1]-y[1:n-1]などとすることが汎用性があり一般的です。また、この方が最初から行数も揃っています。当然のことながらｎはあらかじめ計算しておく必要があります。RATS ではべき乗計算では＊＊を演算子にする一方、GAUSS では ^ を演算子にする違いがあります。GAUSS ではすべての命令や式ごとにセミコロン；が必要です。なお、GAUSS には OLS コマンドが特設されていて、そのリターン値として{ vnam,m,b,stb,vc,stderr,sigma,cx,rsq,resid,dwstat }などがありますが、本書ではパッケージ計算をしない GAUSS のプログラム法なのでおすすめできません。ご自分ですべての統計値を途中計算も含めて計算してください。

データの読み込み

| RATS | GAUSS |
|---|---|
| calendar 1990 1 4 | |
| allocate 2000:4 | |
| open data d:¥datafile.txt | load data[30,3]=d:datafile.txt; |
| data(for=free,org=obs) / y x1 x2 | y=data[.,1]; x1=data[.,2]; x2=data[.,3]; |

RATS は時系列専門のパッケージソフトなのでファイルの読み込みに先だって時系列を指定しなくてはなりません。上の場合、calendar で 1990 年１期から４半期ごとに、alloacte の 2000 年第４期までを指定しています。簡略形として、cal と all と書くことがあります。特に all はすべてのという意味はありません。ファイルの読み込みは open data の後にディ

レクトリ付で指定します。バックススラッシュの記号はアジアフォントでは￥のマークをつけてディレクトリーを区切ります。そして次の行で、data の後の括弧のなかでフォーマット形式とデータが縦なのか横なのかを指定します。なお、ここでの for は format の、org は organization の略です。ここで、free はフリーフォーマット、EXCEL の場合には、xls を指定します。また、obs は observation の略で縦並びの通常のデータの並びを指します。スラッシュの後に、データの変数名を列挙します。なお、スラッシュは OLS の linreg の後に/ resids として残差を計算してその変数に蓄える機能や、通常の四則の割り算の意味もあります。一方、GAUSS では、そのような時系列の指定はありません。あくまで、行列として数学的にデータをとりこむだけです。GAUSS 同様にバックスラッシュでディレクトリーを区切ります。これはスラッシュで代用できます。GAUSS では読み込む時に必ず行列の配列を指定しなくてはなりません。この場合は 30×３としています。いったん data という行列変数に行列を読み込んでしまってから、その１列目をｙ、２列目をｘ１、３列目をｘ２と行をワイルドカードで指定して設定します。

関数の作り方

| RATS | GAUSS |
|---|---|
| procedure lmstat depvar start end | proc lmstat(depvar,x); |
| type series depvar | |
| type integer start end | |
| local real lm | local lm; |
| . . . . . | . . . . . |
| | retp(lm); |
| end procedure | endp; |

RATS では、procedure を作る際には、その後に関数名と depvar/start/end をまず列挙します。その後で、インプットについては type で series か vector か integer か real かなどを指定しなくてはなりません。ローカル宣言も変数の型を指定してから変数名を置かなければなりません。忠実に近代言語の体系を守っていると言えます。その後、操作や計算の後で end procedure で終わります。なお、呼び出す場合には、@lm y 1990:1 2000:4 などしてアットマークをつけて呼び出します。一方、GAUSS ではインプットの型の指定はなくて、すべて行列として扱われます。ただし、ローカル宣言は、型宣言なしに、必要です。操作計算のあと、retp ( );でそこからリターンを返します。最後は endp;です。

Do ループ

| RATS | GAUSS |
|---|---|
| set k=0 | k=0; |
| do i=1:10 | for i (1,10,1); |
| eval k=k+i | k=k+i; |
| end do i | endfor; |

RATS では、i=1 から 10 まで１ステップで増加するループを i=1:10 と end do i によってはさんで使います。途中の計算は、set や compute のほかに eval も使われます。GAUSS では、for ループで計算する場合には、括弧の中に１から 10 まで１ステップという意味で記述して、最後は endofor;で終わります。GAUSS のプログラミングでは、この for ループよりも下に示すように、do while ループの方が汎用性があって一般的です。

```
k=0;
i=1;
do while i<=10;
    k=k+i;
i=i+1;
endo;
```

条件分岐

| RATS | GAUSS |
|---|---|
| if (b>1) | if b>0; |
| set y=1 | y=1; |
| else | else; |
| set y=0 | y=0; |
| | endif; |

RATS では条件文のところをよく丸括弧で包みます。GAUSS でもそうしてもかまいません。条件文が何重かになる場合には{　}の括弧でどこからどこまでがそうなのかを示すこともあります。一方、GAUSS では endif;が何重かの場合には、その都度その数だけ必要です。