3.3 プログラミング If　　ver. 0.1

ループとならんで、プログラミングの機関車の両輪となるのは、条件分岐 if とそれにまつわる命令群です。単に場合分けをしたり、H0 仮定が棄却されるかされないかの判断をしたり、変数にその変域以外の数を代入した場合にエラーログを出させたり、break や continue の命令とセットで Loop から抜けたり先頭に戻ったり、これを自由に扱えるようになれば、自由自在に、論理をともなった真のプログラミングができるようになります。

まずは GAUSS での If 条件分岐の基本的な使い方と独自のきまりについて説明します。最も重要な決まりごとは、たとえ If が１行の文でそこで終わっていようとなかろうと、必ず endif;文が必要なことです。前章で説明しましたように、if や for などの文の最後も；のマークをつけて、そこまでが１命令であるとすることとあわせて、これらの事項は、他言語のプログラミングを少し知っている方には注意を必要とします。

一枝分岐

プログラム

```
new; cls;
x=2.56;
mu=2;
sigma2=0.5;
if sigma2<=0;
    "Variance must be a positive number.";
    end;
endif;
p=cdfn((x-mu)/sqrt(sigma2));
print p;
```

画面表示

```
 0.78580788
```

上のプログラムは、標準化されていない正規分布$N(\mu, \sigma^2)$の場合の$-\infty$からｘまでの確率ｐを求めるプログラムです。前章のstandard normalの$N(0,1)$のテーブルを作成する際に用いた組込み関数cdfn(x)を用いて分布をμ方向に並行移動させて、σ^2で縦方向につぶしています（この場合はのばしています）。ただ、実際σ^2は必ず正の数であるはずですから、プログラム的には、もっと大きなプログラムで変数を受け渡したり、procedureとして他の人も使えるようにしたりするには、このことも考えたプログラムである必要があります。そこで、０か負の数の場合には、Variance must be a positive number.というメッセージを画面表示して、そこでプログラムを終了させてしまうことが必要です。そうでない場合は、endif;行よりも下のプログラムを実行していきます。if文とendif文は、いつもどんなときに

も組で使います。他の言語のようにif文を書いてgotoで飛ばしたら、あとは知らないというプログラム体系にはなってはいないので、そこのところはifがきたらendifでしめるという確認をいつも怠らないようにしてください。繰り返しになりますが、if文にも、行末を示す；のマークが必要です。次の条件文内部の命令とあわせて１行にまとめて書いてしまう場合にも、条件文に１つめの；のマークをつけ、条件文内部の命令のあとにも；のマークをもう１つつけることが必要です。エラーログの書き方には、独自の決まった作法があるのですが、とりあえずは一般的なprint文で出力する方法を示すことにとどめておきます。

プログラムを終了させるend;文について少し述べたいと思います。本書では、end文をプログラムの最後に置くことを奨励していません。公式マニュアルにあるように、つけてもいいものであって、つけることが必須ではありません。それは、本書の主眼であるprocを作成してつなげていく手法には障害になるからです。procを用いないプログラムでは、end;をプログラムの末尾に書いて、開いているファイルをすべて閉じて正常終了をプログラマーサイドでいちいち指定することもよいかもしれません。しかしながら、プログラムの後半にprocをいくつか置いて、前方参照の形でプログラムを書いていく際には、procの終わりにend;文があっても困りますし、procの出てくる前の本プログラムの末尾にend;文があっても同様に困ります。そういう観点から本書では、上の例のように、プログラムの流れを強制的に止めて終了させないかぎりは、end文は使わないことにします。またGAUSSのプログラムを書く人にはそうしてもらいたいと思います。（通常の短いプログラムでもend文なしでも問題なく動きます。また公式マニュアルも必ずしも必要とは書いてはいません。）

上のプログラムがend文が何かということを如実に示しています。もし、end文がif一枝分岐のあとになければ、プログラムの流れは、sigma2が負の数であった場合も含めて、もとのプログラムの流れであるendif;よりもあとに戻ります。このend文は、そのあとに何のプログラムがあろうとそこで強制終了するという意味なのです。したがって、この分岐は、もしif文がTrueであるならば、print文とend文を実行して強制終了して、そこでその枝は途切れます。一方、それがFalseであれば、endifのあとにプログラムの流れはいきます。同じプログラムでもend文を取り除いたならば、if文がTrueであるならば、print文を実行してから、endif文のあとのプログラムの流れにもどろうとします。しかし、sigma2がマイナスの値は計算できないわけで、GAUSSの内部でエラーを発してそこで異常終了します。また、Falseであれば、endifのあとにプログラムの流れはいきます。このように、end文はそこでプログラムの流れをたち切って、強制終了させることが本来の役目なのです。（ただし、標準ではないライブラリーの一部やGAUSSの上で動かすGPE２などのプログラムはend文をプログラム末尾に要求するケースもあるかもしれません。）

ポイント　if文には必ずendif文をともなう。またifの条件文にも；のマークは必要。

二枝分岐

プログラム

```
new; cls;
t=1.94; df=4; pp=0.1;
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;
```

画面表示

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

上のプログラムは、ｔ値と自由度dfと両側検定でのカットオフpercentage pointが与えられた時に、棄却するのかしないのかの判断をGAUSS自身にさせる簡単なプログラムです。ｔ分布のそのｔ値までのcomplementの確率を求める組込み関数cdftc(t,df)を用います。切片やそれぞれのβが有意か有意でないかは両側検定で行ないますので、cdftcで返ってきた値を２倍して、それとカットオフ%を比べています。もし、その２倍した値が0.1よりも以下であるならば、ｔ分布の両側のカットオフポイントよりも内側にある確率が10%以下ということになりますので、pp×100すなわち10% significance levelでH0: β＝０は棄却されます。そうでなければ、（すなわち、２倍した値が0.1よりも大きいならば）ｔ分布の両側のカットオフポイントよりも内側にある確率が10%よりも大きいということになりますのでpp×100すなわち10 % siginificance levelでH0:β＝０は棄却されません。この論理をすべてGAUSS自身に任せてあります。（ただし、前のプログラムのようなエラー処理は省略してあります。）文法は、if文で条件文がTrueである場合には、そのあとの命令を実行し、そうでなければelse文よりもあとの命令を実行します。こうした２つの条件分岐を経て、どちらの経路を行っても、最終的に、endif文よりもあとのプログラムの流れにもどります。

最後にプラスマイナスｔ値の内側にある確率ｐ値を画面表示させています。この値は、計量のパッケージソフトのregressionで各係数のｔ値とともに印刷されるｐ値という確率です。０に近ければ近いほど、その係数は有意であるということになります。例えば、0.09というｐ値が印刷されているのなら、0.1以下ですから両側検定の10% significance levelで棄却されます。また、0.09は0.5以下ではありませんから５%では棄却されません。これらの値は、どのソフトでも、通常すでに２倍された確率の値になっています。プログラムをしてみるということは、実際の計量経済学をより深く知るということでもあります。

三枝分岐

プログラム

```
new; cls;
t=2.8;
cp=2.34; /* Cutoff point at 10% */
if abs(cp-t)<0.1;
    print "We don't know around cutoff point. Check other criteria.";
elseif t>cp;
    print "Reject H0 at 10% level";
else;
    print "Do NOT reject H0 at 10% level";
endif;
print /lz t "        against the cutoff point of" cp;
```

画面表示

```
Reject H0 at 10% level
2.8                          against the cutoff point of 2.34
```

ｔ分布やＦ分布、χ^2分布などは、ハッキリとGAUSSで有意水準に対するパーセンテージを計算できるが、計量経済を進んでいくと、偉い大家がモンテカルロシュミレーションでそれぞれの有意水準に対しての統計量のカットオフポイントを示しているにすぎない場合によく直面する。その場合には、書物に載っている数値と自分の計算した統計量を比べて判別することになる。とはいっても、このカットオフポイントも数十年前のカードリードの時代の計算かもしれないので注意が必要で、特に、カットオフポイントの周辺の値は決めつけない方が賢明であろう。そういった時には、頭の中の論理と同様、三つ枝分岐をさせる。３つの範疇を作る。この場合、カットオフポイントの周辺0.1よりも近い距離にある統計値の場合、カットオフポイントよりも大きい棄却領域、それにそれ以外のすべての３つが考えられる。厳密には、１番目の領域と２番目の領域はカットオフポイントの周辺で重なっている。しかしながら、GAUSSはプログラムを上から順番に実行していくので、重なっていても一向に差し支えはない。二枝の場合には、if文else文で分岐させて、最後にendifでしめていたが、今度は、if文とelse文の間に新たにelseif文が入る。もし、４分岐以上させたいのなら、このelseif文を追加する分岐文だけ書けばよいわけです。すべての分岐は同等ですが、GAUSSはプログラムを上から実行していくので、一番上の分岐の条件がTrueであれば、そのあとから最初のelseif文直前までの命令（複数行でもよい）を実行したあと、以降の分岐は飛ばされて、endif文のあとからプログラムの流れは再開します。したがって、全ての分岐が正常に動くかどうかは、プログラムする側が異なるデータを代入してみたり、極端な値を入れるいじわるテストをする必要が別途必要です。上のプログラムの場合、tに2.8という統計量を与えてやり、あらかじめシュミレーションで求められて

いる10%のカットオフポイントcpの2.34と比べてH0が棄却されるかされないのかを自動判別させています。３つの範疇のうち上から順番にTrueかどうかを判別していき、Trueであれば、その時点で、それ以降の分岐には入りません。endif文よりもあとの、プログラムの流れに戻って、print文でその統計値とカットオフポイントを画面表示させています。

ポイント　一枝分岐

```
if 条件文；
    命令文；
endif;
```

ポイント　二枝分岐

```
if 条件文；
    命令文；
else;
    命令文；
endif;
```

ポイント　三枝以上の分岐

```
if 条件文；
    命令文；
elseif;
    命令文；
    ……
elseif;
    命令文；
else;
    命令文；
endif;
```

ポイント　命令文のところは複数あってもかまわない。

if文を二重構造にすることも可能ですし、並列でいくつも並べていくことも可能です。そのどの場合も、はじめにTrueになった経路以外は、プログラムは実行して動くかどうかの確認はなされませんから、すべての場合について変数に入る値を変えてやって、プログラムのすべての箇所が実際に正常に動くのかどうかをプログラムをする側で確かめなければなりません。特に、GAUSSはエラーを起こすとそれ以降のプログラムを実行しないで異常終了しますから、エラーの場合もちゃんとそこで止まるか、または、それ以外の計算も実行されるようなエラー処理を自前でする必要があります。

次に、if分岐が実際に0(False)とそれ以外の数(True)の論理によって動いていることを

実際に示してみましょう。If文の条件のところにくるものは、大小関係や＝＝関係などの判別式ではなくとも、変数自体もくることができます。

プログラム

```
new; cls;
data={1 "Tom",
      0 "Nancy",
      2 "Shawn",
      1 "Todd",
      0 "Lisa"};
i=1;
do while i<=rows(data);
   if data[i,1];
      print $data[i,2];
   else;
      print $data[i,2] "must be female.";
   endif;
i=i+1;
endo;
```

画面表示

```
  Tom
Nancy must be female.
Shawn
 Todd
 Lisa must be female.
```

上のプログラムは、dataという行列変数の１列目には、男か女を区別するフラグが入っていて、２列目には実際の名前が入っているものとします。明らかに男ならフラグは１、女なら０、どちらでもありうるなら２としています。これらのフラグをとよりに、データを選り分けようというものです。全体の構造は、do whileからendoまでのループを行数ｉにもとづいて１からdataの行数だけまわしています。そのループのなかで、変数dataの１列目の数が０以外（True）であるならば、その行の変数dataの２列目の名前だけを表示させます。そうでなければ、すなわち、変数dataの１列目の数が０であるならば、その行の変数dataの２列目の名前とそのあとにコメントとしてmust be female.と画面表示させています。画面結果のように、亀井静香、工藤静香のしずかのように、Shawnは女性名にも男性名にもなりますから２と分類されていますが、論理判別では、０以外の数として、男性である１とともにif分岐のはじめのところでTrueとして扱われています。このように、ifの直

後やelseifの直後には、大小関係や論理を表す判別式だけではなく、ただ１つだけ変数を書くこともできます。もし、そこに変数名だけがあるプログラムを見かけたならば、それは、その変数が０以外の場合、直後の命令を実行し、そうでなくて０の場合にはelse直後の命令を実行していくのだと理解してください。見なれていないと何のことだということになります。また、elseの方が０で、ifの直後が０以外のすべての数であるということも誤解があるところなので、しっかりとおさえておいてください。けして１だけがtrueを表す数ではないということをこの例で学んでください。

BreakとContinue

forループまたはdoループにおいて、if条件分岐と組み合わせて、ループから抜けたりループの最初に戻ったりすることが可能です。Goto文で抜ける方法もありますが、特に、ループの外の特定の場所に移動しないかぎり、breakで抜けるのが正統的なやり方です。
いま、d:ドライブの一番上の階層に、datafile4.txtというデータファイルがあるものとします。[W.Greene, "Econometric Analysis 2nd Edition," Table 21.12]これを読みこんでLog Likelihood Functionが

```
logl(b)=sumc(-exp(x*b)+ y .* (x*b) - ln(y!))
```

と与えられている時のLogLikelihood Functionを最大にするベクトルβを求めてみます。
１列目の１から20までのインデックスナンバーと13列目のデータは取り除いて、12列目を従属変数ｙに、独立変数側には、切片に１の列ベクトル、２列目から10列目までの質的変数はそのまま使って、11列目の量的変数は、計算上Singularを避けるために、1000分の１に縮小させています。このスケールの値は任意ですが、0.01または0.001がsingularにならずに有効なようです。このとき、β={1,1,1,1,1,1,1,1,1,1,0}からニュートンステップで$-H(\beta)^{-1}\, g(\beta)$をβに再帰的に加えていくことににより、Likelihood Functionが最大になるβを求めます。（最後のβを１にするとsingularになりますので０から始めて下さい。）
そこでdoループである回数まわしてiterationをするのですが（これを仮に最大100回としましょう)、gradientとHessianのところで注意したように、10のマイナス３乗から４乗よりも小さい桁は、これらの計算では有効ではありませんから、gradientのすべての要素が0.001よりも小さくなった場合にこれを０と判断して、ループから抜けるためにbreak命令を置いています。なお、if文を使ったならばどのような場合にもendif文は必須です。お忘れなく。Hessianのことを考えていないので厳密ではありませんが、gradientが零ベクトルになったあたりで計算をやめて最大値に対応する値としています。実際に、Log Likelihoodの値が頂上あたりであろうことが、そのiterationの値からわかるようにてします。
プログラム

```
new; cls;
load data[20,13]="d:datafile4.txt";
y=data[.,12];
```

```
x=ones(20,1)~data[.,2:10]~(0.001*data[.,11]);
fn logl(b)=sumc(-exp(x*b)+ y .* (x*b) - ln(y!));
beta=ones(10,1)|0;
format /rz 6,4;
print "beta at start = "; print beta';
print "Gradient at start = "; print gradp(&logl,beta);
print "iterations";
i=1;
do while i<=100;
    iter = i;
    format /rz 6,6; print logl(beta);
    gvector = gradp(&logl,beta);
    if abs(gvector) < 0.001;
        break;
    endif;
    hmatrix = hessp(&logl,beta);
    beta = beta - gvector'/hmatrix;
i=i+1;
endo;
format /rz 6,4; print "# of iterations = " iter;
print "logLikelihood at the maximum = " logl(beta);
print "beta around max =      ";print beta';
print "Gradient around max";   format /rd 5,5; print gvector;
```

画面表示

```
beta at start =
     1       1       1       1       1       1       1       1       1       1       0
Gradient at start =
-184.7 -47.34  63.66 -75.34 -65.34 -60.34 -70.43 -35.43 -13.43 -65.43  926.1
iterations
-250.973
-99.7044
-54.3302
-47.3996
-46.8131
-46.7992
-46.7991
```

```
# of iterations =      7
logLikelihood at the maximum =  -46.8
beta around max =
-0.3955  4.075  3.643   2.18  3.282  3.556 -2.558 -2.156 -1.148 -1.644 0.1426
Gradient around max
-0.00004  0.00000 -0.00001 -0.00002  0.00000  0.00000 0.00000 -0.00001 -0.00002 -0.00001 -0.00002
```

上のプログラムは、太字の太字のあいだのループの中で、β＝β＋[-H(β)$^{-1}$ g(β)]のニュートンステップをやらせています。[　]括弧のなかは、計算では-g(β)/ H(β)のことですが、これは要素対要素の割り算ではなく、この場合βと同じディメンションの11×１の列ベクトルを計算する解の計算なので勘違いのないようにしてください。また、gradpが横方向の行ベクトルを生成するので、計算では転置させてg'としています。このステップを再帰的にやることによって、関数値を最大にするβが求まります。gradientが零ベクトルのとき、このβが関数を最大化するものであるには、厳密にはHessianがnegative definiteである必要がありますが(positive definiteであれば最小化する値)、それを確認するかわりに、iterationごとの関数値、この場合、Log Likelihoodの値を表示して、徐々に最大に近づいていることを代わりに確認しています。このプログラムは、gradientが零ベクトルになったとき、ループからbreakで抜ける仕組みになっています。プログラム上では、ifの条件文でgradientベクトルの絶対値が0.001よりも小さくなった場合**abs(gvector) < 0.001**
breakでループをぬけて、100回のループが終了する前の何回目かに、endo文の次に行きます。そのほかの部分は、fnでLog Likelihood Functionを１行で定義してやって、それを&のマーク付の変数で呼び出してやることによって、gradpとhesspで、gradientとHessianのベクトルと行列の値をそれぞれのβについてスタート値から計算させています。endo以降では、format文で適当に数値の間をせばめて、実際に行なったiterationの数、最後のmaxでのβのベクトル、そしてgradientベクトルを表示しています。当然のことながら、最後のgradientベクトルのそれぞれの値は、設定した0.001よりも絶対値で小さい値で、０に近い値になっているはずです。本格的なプログラムですが、本節でいままでやってきたことがすべて使われているプログラムであると言えます。もちろん、for文でさらに簡単に書くことも可能です。ちなみに具体的な例は示しませんが、continue文はbreak文の逆で、if分岐でそこがTrueであるとループの最初に戻る命令です。

ポイント　break;　　forループまたはdoループから抜ける
continue;　forループまたはdoループの最初に戻る

Goto文

このほか、あまりおすすめはできませんが、Fortranなどで書かれたプログラムを移植す

るのに役に立つgoto文とif文を組み合わせて使うことも可能です。（以下は、オプションとしての手段です。くれぐれも多用はしないでください。）

　上のプログラムを用いて、if～breakのところをif～gotoにおきかえてみます。

プログラム

```
new; cls;
load data[20,13]="d:datafile4.txt";
y=data[.,12];
x=ones(20,1)~data[.,2:10]~(0.001*data[.,11]);
fn logl(b)=sumc(-exp(x*b)+ y .* (x*b) - ln(y!));
beta=ones(10,1)|0;
format /rz 6,4;
print "beta at start = "; print beta';
print "Gradient at start = "; print gradp(&logl,beta);
print "iterations";
i=1;
do while i<=100;
   iter = i;
   format /rz 6,6; print logl(beta);
   gvector = gradp( &logl,beta);
   if abs(gvector) < 0.001;
      goto label;
   endif;
   hmatrix = hessp( &logl,beta);
   beta = beta - gvector'/hmatrix;
i=i+1;
endo;
label:
format /rz 6,4; print "# of iterations = " iter;
print "logLikelihood at the maximum = " logl(beta);
print "beta around max =       ";print beta';
print "Gradient around max";   format /rd 5,5; print gvector;
```

上のように、goto ラベル名;で、if条件文がTrueあれば、ラベル名:とある場所までジャンプします。なお、このラベル名は何であってもかまいませんが、：のマークが必要です。

ポイント　goto ラベル名;　ラベル名:とある場所までジャンプする