2.2 行列 データの加工　　ver. 0.1

セクション１のＩ/O のところでもファイルの読込みの際に少しふれましたが、ここではあらためて行列としてデータを見たてて、それを加工変形する手法を説明します。まず、実際のデータは縦に並んでいて、それぞれの列が１つ１つのデータのシリーズになっていることがほとんどです。それを１つの行列と考える習慣をつけてください。ここではその読込まれた行列を自由に切りはりして、統計計量分析にかける下準備をします。

基本的な考え方としては、まず行列データは**変数名[行数，列数]**という体裁をとります。例えば、a[30,5]というふうな行列変数で表されます。行列名がａで、その行列変数は 30 行５列で成り立っているという意味です。しかしながら、実際のデータには分析に不要なシリーズがあるのが常ですから、その部分をカットします。例えば、１行目にはそれぞれのデータシリーズのラベルが書いてあるとすると、その部分の行をまずカットしなければなりません。それには、a[2:30,5]としてやれば、行数のうち、２行目から３０行目が行列変数ａの中味になります。このことは、すでに 1.2 でやりました。日本では、Excel 形式のファイルが官公庁では多いですから、実際には、エクセルに読込んだ後、スペース形式のテキストファイルで保存する形となります。データが始まるまで６行ラベルなどの説明があれば、７行目から始めるということで、a[7:30,5]というふうに、７行目から何行目までというふうに決めなくてはなりません。実際には、ファイルの加工の段階で上を何行か切ってしまえば、この作業は省略できます。それよりも問題は、ファイルが実際に長くて、何行かわからない場合には、自分でラベル部分を切り取ってしまって、データのみにした後で、一体データが何個あるのかを GAUSS に数えさせて、行列の行数を決めた方が無難です。一部繰り返しになりますが、そこから今回は始めましょう。

d:ドライブに datafile2.txt というファイルがあるとします。これを読み込んで見ましょう。このファイルは、すでにラベルは切り取られていて、データのみになっています。

プログラム

```
new;  cls;
load data[ ]=d:datafile2.txt;
n1=rows(data);
n=n1/5;
load data[n,5]=d:datafile2.txt;
print data;
print "n=" n;
```

画面結果

```
18.000000      8.0000000      307.00000      130.00000      3504.0000
15.000000      8.0000000      350.00000      165.00000      3693.0000
```

```
        ..........................................................................
        ..........................................................................
         31.000000          4.0000000          119.00000          82.000000          2720.0000
n=          392.00000
```

2.2 でとりあげたように、データの配列を指定しないで[　]の中を空でファイルを取りこむように設定すると、ファイルの中の数値はすべて１列の列（縦）ベクトルとして取りこまれます。その行列の行数をカウントすれば、データ全体の個数がわかるはずです。組込み関数 rows を用いて、仮に１列になっている行列変数 data の行数を計算して、それを n1 と定義代入しています。それを実際のデータの列数５で割れば、実際のデータの行数ｎが求まるわけです。そのｎをあらためて行列変数 data の配列の行数として data[n,5]として再び正式にファイルを data という変数に読込んでいるのです。なお、[ ]は[]とくっついていてもかまいません。最初の変数 data はその次の data の読込みによって上書きされて新しい２番目のものが data という変数の中味になっています。１度でうまくデータの取りこみが成功するとは限りませんが、元データを何度か切り貼りして、なんとかこの方法でたいていは読込めます。データの場所が d:ドライブの階層深くにある場合には、ファイル名の部分を、d:/gauss/data/datafile2.txt　という具合にスラッシュ“ / ”で場所を表します。一部の書物やマニュアルではバックスラッシュや円のマークになっていますが、スラッシュと同じ意味です。スラッシュにすれば、どのような言語環境でもスラッシュとして表示されます。「あの円のマーク￥は何だ？このマニュアルはバックスラッシュなのに。」ということにはならないでしょう。

　さて、実際のデータの行には、分析したくないグループやミッシングデータが含まれていることがあります。それらを除いたものをセレクトする、または、それらをデリートする方法を説明しましょう。意外にこの作業は重要であり、かつ、データの性質を考えて慎重にしなければならないところでもあります。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2.txt;
data=selif(data,data[.,2] .>4);
n=rows(data);
print data;
print "n=" n;
```

画面結果

```
        ..........................................................................
        22.000000          6.0000000          232.00000          112.00000          2835.0000
n=          189.00000
```

このプログラムは、組込み関数 selif を用いて、行列変数 data の第２列の値が４よりも大きいものを選んで残そうというものです。selif は select if の略で引数には、カンマの前に操作しようとする行列変数名、カンマの後にどの行をセレクトの対象にするかが示されスペースを入れて、ドットというワイルドカードの数字が、４よりも大きい場合ということで、.>4 となっています。この部分は、selif の反対の役割を果たす組込み関数 delif を用いて、data=delif(data,data[.,2] .<=4);記述しても結果は同じになります。こちらは delete if の略です。適宜使い分けてください。なお、２つ以上の条件を and でつないでいくには、data=selif(data,(data[.,2] .>4) .and (data[.,2] .<=7));という具合に２番目の引数のところの条件を条件ごとに括弧につつんで、.and でつなげば２つの条件を満たすデータがセレクトされて残されることになります。and の前にドットをつけることと、最後の括弧が全体の組込み関数の括弧も忘れないように注意が必要です。delif の場合も同様です。

ポイント　selif(**行列変数**,**行列変数の列配列**　.**論理式**)　　**論理を満たす列をセレクト**
　　delif(**行列変数**,**行列変数の列配列**　.**論理式**)　　**論理を満たす列をデリート**

もっとより一般的な必要に応じてデータをカットしなければならないのは、ミッシングデータのケースです。GAUSS では他の一般的な統計計量ソフトと同様に missing data はドット．で示されます。これを上の selif で取り除いてみましょう。同じ 392 行５列のデータですが、最後のあたりの行のうち３行のデータの一部を改ざんしてドット（ミッシングヴァリュー）に置き換えたファイル datafile2m.txt を読込んで、そのドットのついた行３行を除いてみます。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2m.txt;
data=selif(data,(data[.,.] .>=0) .or (data[.,.] .<0));
n=rows(data);
print data;
print "n=" n;
```

画面結果

```
……
……
38.000000      6.0000000      262.00000      85.000000      3015.0000
26.000000      4.0000000      156.00000      92.000000      2585.0000
22.000000      6.0000000      232.00000      112.00000      2835.0000
36.000000      4.0000000      135.00000      84.000000      2370.0000
27.000000      4.0000000      151.00000      90.000000      2950.0000
```

```
        44.000000       4.0000000       97.000000       52.000000       2130.0000
        32.000000       4.0000000       135.00000       84.000000       2295.0000
        31.000000       4.0000000       119.00000       82.000000       2720.0000
n=          389.00000
```

うまく取り除けたでしょうか。selif を用いて、論理的には、０以上（０を含む）の数の行「または」０より小さい数の行の条件を満たす行列変数 data をセレクトして残しているのです。今回は、and ではなくて or を使いました。or の前のドットを忘れないで下さい。このテクニックには STATA などの統計ソフトで、drop 命令を使ってミッシングデータをドロップする手法と同じです。ほかにも論理的に、もっといいやり方があるかもしれません。マニュアルには論理式ということで、＞の代わりに gt が greater than の略で使われ、＜の代わりに lt が less than の略で使われていますが、＞、＜、＞＝、＜＝の記号は自由に使えますので、置き換えて考えてください。また、selif と delif の引数のカンマ以降は、別の変数、例えばｅとかとおいて、その前で e=という形で論理式を定義することもできますが、あまり実用的とは言えません。少々長くなっても、引数の中に書いてしまう方が見やすく移植しやすいプログラムと言えるでしょう。

ポイント　selif(data,(data[.,.] .>=0) .or (data[.,.] .<0))　　**ドットの削除の手法**
ポイント　selif,delif **の論理式は(　) .and (　)または(　) .or (　)でつなげていく**

　次に、データをソートするやり方を紹介しよう。何種類かコマンドはあるが、代表的なものは sortc である。たいていのことは、これ１種類で可能なので、ここではそれだけを取り上げておこう。**sortc(行列変数,ソートする列番号)**とすると、小さいものから大きいものに列番号を比較して行ごとに置き換わる。ただし、ミッシングデータのドット.は一番小さなものとされる。引き続き datafile2.txt を利用して見ていきましょう。
プログラム

```
new;  cls;
load data[392,5]=d:datafile2m.txt;
data=sortc(data,4);
print data;
```

画面表示は省略しますが、４列目を比較して、まず最初に４列目がミッシングデータとなっているもの、それから４列目が小さいものから大きいものへと順に並べられている。当然のことながら、それぞれの行は１つの組としてほかの列も４列目のソートに応じて移動しています。大きいものから小さいものへと並べるには、rev という組込み関数を使って上のプログラムを書き直します。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2m.txt;
data=rev(sortc(data,4));
print data;
```

こうすると、４列目のソートでできた小さいものから大きいものへと並んでいるものを逆さまにひっくり返して、大きいものから小さいものへと順に並び替えることができます。sort の後ろについているｃは column の略です。列についてソートするという意味です。また rev は reverse の意味です。

この組込み関数を５つ５列分書いたからといって、１列目が小さいものから大きいものへと並んでいて、その中で、１列目が同じ数どうしで、２列目も小さいものから大きいものへと並んでいて、最後の列までそうなっている状態はできません。ソートの都度、それぞれの列を基準に並び替えられてしまいます。そんな時を想定して、sortmc が用意されています。Sort の後ろの mc は multiple column についてのソートの略です。書き方は、

sortmc(行列変数,列ベクトル)　ただし列ベクトルは別に作成されていて、ソートされる列の番号がカンマつきでソートの順番に列ベクトルに入ります。直接代入はできません。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2m.txt;
a={1,2,3,4,5};
data=sortmc(data,a);
print data;
```

これにより、データは１列目がソートされ、１列目が同じ数の中で、２列目がソートされ…というふうに最後の５列目までソートされます。なお、ソートの順番を３列目、２列目、そして５列目としたい場合には、a={3,2,5}と設定します。変数ａの名前は任意です。
これらソートの組込み関数は、クイックソートのアルゴリズムで書かれています。

ソートの技法は、計量経済系の人はあまり使わない傾向がありますが、データをまず「見る」という作業から始めるのには重要な手順です。それをたとえ、回帰分析などに使わなくても、別に行なってデータを見ることが大切です。レフリーが元データを投稿者に提出させて、それをソートで並べなおすと、ミッシングデータをもうすでに切り取った後であったり、理論も何もなくやみくもに０のデータを含む行を削除した後であったり、そういった無茶苦茶なデータ操作を見つけることもソートでは可能です。また、０と１だけのダミー変数だと思っていたものが、実はソートするとフラッグとして－１もあったりして、それを含んでダミー変数として分析してしまっていたという誤りも見つけることができま

す。偉い先生やそのお弟子さんということで無茶苦茶な「神の手」が許されるデータ「捏造」を発見するのにも、ソートの操作は、原始的ですが重要なツールです。

ポイント　sortc(行列変数,ソートする列番号)
例　sort(x,1)　ｘという行列変数を１列目について昇順にソート
ポイント　rev(sortc(行列変数,ソートする列番号))
例　rev(sort(x,2))　ｘという行列変数を２列目について降順にソート
ポイント　sortmc(行列変数,列番号の入った列変数)
例　a={3,4}; data=sort(data,a);
data という行列変数を３列目で昇順でソートした後、その中の同じ数値を、今度は４列目で昇順でソートしたものを data に代入格納。

さて、データというものは、毎年毎期アップデートされるもので、同じ種類のデータをさらにつけ足したい場合もあるでしょう。そういう際の方法を示しておきます。それは、行列を縦方向にマージする方法を応用します。同じ列数の行列同士を縦方向につぎたす場合には GAUSS では縦棒｜のマークを使います。**上の行列｜下の行列**と書いて、分数の上下関係のように上と下をくっつけてしまいます。今度は datafile2.txt の方を使います。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2.txt;
upd={44.2 5.0 90.0 66.0 2100.0,
     38.9 6.0 85.0 59.0 1985.0,
     92.4 7.0 92.0 88.0 2085.0};
data=data|upd;
print data;
```

画面表示

```
……
……
28.000000      4.0000000      120.00000      79.000000      2625.0000
31.000000      4.0000000      119.00000      82.000000      2720.0000
44.200000      5.0000000      90.000000      66.000000      2100.0000
38.900000      6.0000000      85.000000      59.000000      1985.0000
92.400000      7.0000000      92.000000      88.000000      2085.0000
```

最後の３列が元々のデータに縦方向につぎ合わさったことがわかると思います。この際、注意してもらいたいのは、GAUSS は自動的に列数の調整はしませんので、事前に上にくる

行列変数と下にくる行列変数の列数が一致しているか確認が自前で必要になります。また、変数の名前は組込み関数の名前や命令の名前などの予約された語句は使用できません。例えば、upd の変数名を new にすべて変更してプログラムを動かそうとするとエラーが出ます。英語の命令にありそうな名前は、こまめにマニュアルやその索引などを見て、存在しないかどうかのチェックが必要です。予約語と同じ変数名はつけることができません。

その次に今度は、列ごとのデータを取り出すことを考えましょう。５列目をｙとおき、１列目から４列目までをｘの行列とするとします。プログラムは以下のようになります。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2.txt;
y=data[.,5];
x=data[.,1:4];
print y; print x;
```

画面表示は縦に長くなりますから省略しますが、ｙは 392 行１列として、ｘは 392 行４列として表示されたと思います。data[.,5]の[　]は配列を表す括弧で行列のディメンションを決定するものであって、実際の行列のデータが入るわけではありません。最初のドットは行部分はワイルドカードですべてという意味。カンマの後の５は５列目という意味です。したがって、行列変数 data の５列目のすべての行をｙに代入するというのが、その全体の意味です。同様に、ｘには、行列変数 data の１列目から４列目までのすべての行が代入されるという意味になります。：のマークを使って、何列目かた何列目までということを表現できます。データの先端部分の行を切り取るために、data=data[2:392,5]というふうに、行について：のマークを使って表現できましたが、この：のマークは列の方にも使えます。もし、ｘに１列目と３列目と４列目だけを代入したいのなら、そこだけ書きなおして、

プログラム

```
new;  cls;
load data[392,5]=d:datafile2.txt;
y=data[.,5];
x=data[.,1 3 4];
print y; print x;
```

というふうに、配列の[　]の括弧の中のカンマ以下の列部分のインデックスにスペースをあけて１　３　４というふうに書きます。それぞれの数字の間にカンマはつけてはいけません。カンマはあくまで、行と列のインデックスを分けるところにだけ使います。

上の方法が、一番ストレートなやり方ですが、列ごとに名前をつけることもできます。これは、列ごとに変数を変換する際に有効です。例えば、ある列だけをパーセンテージ表

示を 100 で割って割合に変換するとか、ある行だけをポンド表示をキログラム表示になおすとか、いったん各変数に切り離しておいてまたくっつけたりするときによく使われます。変換しないときには、上の方法を使って、簡略に加工するのがベストです。ここでは、１列目をｘ１、２列目をｘ２、３列目をｘ３、４列目をｘ４とおいて、ｘ３とｘ４を 100 でそれぞれ割ったものをあらためてｘ３、ｘ４としましょう。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2.txt;
y=data[.,5];
x1=data[.,1];
x2=data[.,2];
x3=data[.,3]/100;
x4=data[.,4]/100;
print y; print x1; print x2; print x3; print x4;
```

ｘ１からｘ４までいずれの場合も行のインデックスはドットでワイルドカード指定で、列の方は、ｘ１には１ということで１列目が、ｘ２には２で２列目が、ｘ３には３で３列目が、ｘ４には４で４列目が入り、ｘ３とｘ４には 100 で割った数値が代入されています。このように、配列の括弧[　]つきの行列変数に四則演算や組込み関数の演算をすることも可能です。いちいち置き換えをする必要はありません、配列つきで計算式が組めます。

最後に、ｘ１、ｘ２、ｘ３、ｘ４の４つを今度は横にもう一度くっつける方法を説明しましょう。行列変数を縦にくっつけるには｜のマークが用いられましたが、行列変数を横方向に水平につなぎあわせるには～のマークが使われます。行列変数を縦にくっつけるには｜のマークが用いられましたが、行列変数を横方向に水平につなぎあわせるには～のマークが使われます。

プログラム

```
new;  cls;
load data[392,5]=d:datafile2.txt;
y=data[.,5];
x1=data[.,1];
x2=data[.,2];
x3=data[.,3]/100;
x4=data[.,4]/100;
x=x1~x2~x3~x4;
print y; print x;
```

このように、行列を横方向に水平につなぎ合わせるときには～のマークを使います。なおこの場合、当然のことながら、つなぎ合わせる行列間で行数は同じである必要があります。一部の統計計量ソフトのように、行数が違っても自動的につながるという作業は GAUSS ではできません。エラーになります。したがって、事前に行数が同じかどうか確認が必要になります。

ポイント　上下方向には｜で、水平方向には～で行列どうしをくっつけることができる。

　復習になりますが、GAUSS ではスケラーは１×１行列として断りがなければ扱われることを以前説明しましたが、それを思い起こして、ミニ行列を作成して、ここでの知識を深めましょう。

プログラム

```
new;  cls;
data=1~2 | 3~4;
print "data=" data;
print "1st col" data[.,1];
print "2nd col" data[.,2];
print "1st row" data[1,.];
print "2nd row" data[2,.];
```

画面表示

```
data=
        1.0000000        2.0000000
        3.0000000        4.0000000
1st col
        1.0000000
        3.0000000
2nd col
        2.0000000
        4.0000000
1st row         1.0000000         2.0000000
2nd row         3.0000000         4.0000000
```

まず、１と２を水平方向につなぎあわせ、３と４を水平方向につなぎ合わせたものと、２つを上下方向につぎたす。どちらを先にやるかという括弧をつけて計算をわかりやすくしてもかまいませんが、つなぎ合わせるには列数が一致していなければなんらないために、先に水平方向のつなぎあわせが２組行なわれて、最後に上下につぎたしが行なわれます。

さらに、行インデックスのワイルドカードのドットを用いて、１ですから１列目を表示し、次は２ですから２列目を表示しています。同様に、列インデックスのワイルドカードのドットを用いて１ですから１行目が表示され、次は２ですから２行目が表示されています。
以上が、人工的に作成されたデータではない生のデータを加工する際の主なテクニックでした。これらの加工されたデータは GAUSS 上に自動的に残るわけではないですし、また GAUSS 上に Excel や STATA のようなデータワークプレースが装備されているわけでもないので、加工済みのデータは、行列として fmt 形式で GAUSS の行列ファイルとして保存しておく必要があります。