1.3 行列データの作成と保存　　ver. 0.2

引き続き行列データを作成することについて説明していきましょう。くわしい行列の計算や変換などの作業は、あとで別のセクションをもうけて説明しますので、まずは GAUSS 特有の行列データの作成方法とその保存をこの章では扱います。

プログラム

```
new;
cls;
x = {    1990 93.1,
         1991 96.1,
         1992  97.7,
         1993  98.8,
         1994  99.3,
         1995  99.0,
         1996  99.0,
         1997  100.6,
         1998  101.3,
         1999  100.9,
         2000 100.0    };
print “data=” x;
```

行列 x の１列目は年（日本の統計の場合、年と年度は違う上、元号から西暦への変換も必要になる)、２列目は年平均消費者物価指数（ＣＰＩ)。実際の統計データの並びと同じように、縦にデータを並べる。{　} の中は、カンマ，のマークまでが GAUSS では１行と判断され、その後は次の行となる。一番上に、new;という命令を入れてみたが、これはいままでデータが入っているメモリーをリフレッシュさせるためである。小さなプログラムで小さなデータを扱う場合には特には必要ないですが、前のプログラムと同じ変数を使って、中に入るデータが違う場合など、メモリーをリフレッシュさせないと、混乱する場合がある。new; cls;の２つの命令はプログラムの最初に毎回書く癖をつけた方が無難である。なお、行列の中の空白の部分は、いくつあいていても同じ結果となる。

ポイント　行列データのプログラム内での格納の方法は、
変数名 = { **１行目，２行目，　　,最後の行**};
{　　}**で囲み、１行ごとにカンマをつける。**

ポイント　各プログラムの最初には必ず new; cls;**の２つの命令を入れること。**

ポイント　new;**の命令は変数などのメモリーをリフレッシュさせる役割がある。**

このように、データは縦に列ベクトルの集まりとして変数に格納されるということに慣れたでしょうか。数学の世界や、数学を扱うソフトウエアの世界では、あまり列ベクトルとして縦に並べたりしないのかもしれません。しかしながら、GAUSS は Econometrics のソフトウエアです。データが縦に並んでいれば、当然のことながら、それに合わせてソフトウエアのデータ変数の取り込みも縦のまま列として扱われます。

さて、こうして取り込んだデータ変数をそのまま変数として保存して、後で自由に呼び出して使えないでしょうか。アウトプットとしてファイルに出力するのではなくて、行列形式のままファイルに保存できないでしょうか。次のような命令により可能になります。

```
save d:data1.fmt=ｘ;
```

プログラム

```
new;
cls;
x = {     1990 93.1,
          1991 96.1,
          2000  97.7,
          2001  98.8,
          2002  99.3,
          2003  99.0,
          2004  99.0,
          2005  100.6,
          2006  101.3,
          2007  100.9,
          2000 100.0     };
save d:/gauss/data1.fmt=ｘ;
```

上のプログラムは、前のプログラムで行列変数ｘを画面に印刷していたものを、今度はファイル名 data1.fmt という GAUSS 専用のマトリクスファイルフォーマットに保存するものです。ｄ:はドライブ名を表し、ドライブ名の後には：のマークをつけます。/gauss/はそのファイルがｄドライブの中の gauss というフォルダーの中にあることを意味します。他の人のプログラムを見て参考にすると、スラッシュ/　/でフォルダーを表しているのではなくて、バックスラッシュや￥のマークで¥gauss¥となっているかもしれません。どちらでもかまいません。英語環境の Windows ではバックスラッシュはバックスラッシュのままですが、日本語をはじめアジアフォントの環境では￥のマークになってしまいます。したがって、この場合本当はバックスラッシュでディレクトリーを区切るのが正統なほうほう

なのですが、アジア環境ではそう表記されずに￥のマークに置き換わってしまうために、反対向きのスラッシュ／も代用として有効ですし、何ら不都合なことはありません。なおここでのファイル名の拡張子は必ず fmt としなけれないけません(GAUSS に特有の行列形式のフォーマットを表します)。変数ｘの内容を前のファイル名に代入格納するという意味でイコール＝のマークでつなげます。

ポイント　行列変数の直接保存の方法は、
save **ファイル名**.fmt = **変数名**;

ポイント　ファイル名のところには、ドライブ名ｄ：やフォルダー名/gauss/などがファイルの保存場所に応じてつける。プログラムを書く際には、バックスラッシュやそのアジアフォントの￥のマークはスラッシュ／のマークに統一した方がわかりやすい。

注意：正統的にバックスラッシュをフォルダ名の深さのレベルに用いても構わないが、バックスラッシュを用いてしまうとコンピュータのシステムやフォントの構成によっては￥のマークで表示されてしまうことがある。特に初心者を相手にしては、GAUSS のシステムのなかではバックスラッシュなのに、エディターで見たりワープロソフトにコピー＆ペーストすると￥のマークに変わってしまっていたりすると、それだけで無用な混乱を与えるであろう。したがって、最初からスラッシュ／のマークで統一したい。そうすればプログラムを読む人も、それをコピーして動かして見る人も、同じスラッシュでいつも表示される。フォルダーのレベルやファイルの位置については、プログラムの本質とは全く関係のない困難がつきまとう。指導者はくわしいファイルの位置の説明、または、より簡単なファイルの位置を示した方が、学ぶ方にはわかりやすであろう。あまりに複雑なディレクトリーをワークスペースにせざるをえないときには、ソフトウエア的にドライブ名だけの仮装ディレクトリを作ってやってそこで作業をさせるか、または、あらかじめカット＆ペーストできるコンピューター上の文章にその長々としたディレクトリー構造を記載したものを電子的に配布してそれをコピーさせてディレクトリーとすることもよい方法かもしれない。口頭で呪文のように早口でそのディレクトリーにするようにと言ったり、判別不可能な文字で黒板にディレクトリーの構造を書いても無意味であるし、教えられる側は混乱するだけである。ディレクトリーの操作と GAUSS のプログラムは根本的に何ら関係のないことである。

そこで、１度保存された行列データを次のプログラムで呼び出して、その内容を画面に表示してみましょう。

プログラム

```
new;
cls;
loadm a = d:/gauss/data1.fmt;
print “data=” a;
```

１，２行目はお決まりの命令。メモリーリフレッシュする。そして、スクリーンをクリヤーする。３行目は、先ほどの data1.fmt という gauss の行列データ形式のファイルを、変数 a に代入格納してプログラム上に行列としてロードするという意味です。ここでも、後ろのものを前に格納するということがイコール＝のマークの意味なので、前のプログラムとは逆にファイル名が後ろにきています。そして、４行目は、その a という変数を data= という文字列とともに画面に表示するということです。loadm は、以前やったアスキー（テキスト）ファイルをロードする load とは違って、gauss の fmt 行列フォーマットをロードするという意味で、matrix のmの字が load のあとについています。気をつけましょう。

今回は何も行列の計算をしていませんが、実際には１度に実行するには困難な計算をする場合に、途中結果を fmt 形式の gauss の行列データファイルに保存しておいて、また別の機会にあらためてそのファイルを呼び出して別の計算をするときに、この save と loadm の命令はよく使われます。実際には、なんてことはない簡単な作業なのですが、フォルダーのレベルが何重にもレベルが重なっていたり、￥のマークやバックスラッシュのマークが何であるのかを疑問にもつために、無用な困難さを引き起こしているだけなのです。

いままで、ファイルを保存したりプログラム上にロードしたりすることをやってきましたが、その他に、gauss のプログラムの上で、変数や行列を作成することも可能です。１列ベクトルの変数の簡単なケースについて例をあげてみましょう。

プログラム（standard normal distribution のケース）

```
new;
cls;
a = rndn(100,1);
print “data=” a;
save d:datan.fmt = a;
```

プログラム（uniform distribution のケース）

```
new;
cls;
a = rndu(100,1);
print “data=” a;
save d:datau.fmt = a;
```

１つ目のプログラムは、100 個のデータを１列のシリーズで擬似的に standard normal な分布になるように乱数を発生させるもの。new; cls;でメモリーと画面をクリヤした後に、rndn という乱数を発生させる組込み関数を用いて standard normal にふるまうデータを、まず第一要素の 100 は 100 個発生させる。第２要素の１は１列分のシリーズを発生させる。その 100×１の行列を変数 a に代入格納する。そして、その a を data=という文字列とともに画面に表示させる。そして最後に変数 a を fmt 形式の gauss データファイル datam.fmt に保存するというもの。rnd は random の略。ｎは normal の略である。

２つ目のプログラムは、100 個のデータを１列のシリーズで擬似的に uniform な分布になるように乱数を発生させるもの。rndu という乱数を発生させる組込み関数を用いて uniform にふるまうデータを、まず第一要素の 100 は 100 個発生させる。第２要素の１は１列分のシリーズを発生させる。 その 100×１の行列を変数 a に代入格納する。そして、その a を data=という文字列とともに画面に表示させる。そして最後に変数 a を fmt 形式の gauss データファイル datau.fmt に保存するというもの。rnd は random の略で、u は uniform の略である。なお、ここでのファイル名や変数は、どんな名前でもかまわない。

ポイント　標準正規分布にふるまうM個のN列の乱数を発生させるには　rndn(M,N)
ポイント　一様分布にふるまうM個のN列の乱数を発生させるには　rndu(M,N)
ポイント　rndn,rndu **ともに、**GAUSS **起動時からのクロックの値に依存している。**

ここで、ランダムな変数を発生させる rndn と rndu はともに、GAUSS が起動されてからの内部クロックを利用して、それの値をもとにして乱数を発生させている。２度３度と run をして実行を繰りかえすとその数値はかわってくる。もちろん、万が一、偶然同じタイミングのクロックの値になると同じ値のシリーズを発生させる。実際には、３２ビットの２の３２乗の依存数をもとに乱数で発生させているので、十分に実用に耐える。

なお、乱数を発生させるための元の数になる GAUSS 内部のパラメーターである seed を起動時からのクロック時に依存させない方法も存在します。実際には、データの分布をあとでも同じように再現するために、乱数の依存数 seed を自分で任意に設定する方が、プロフェッショナルなプログラムでは、より一般的です。では、seed を使って上の２つのプログラムを書き換えてみましょう。

プログラム（standard normal distribution のケース）

```
new;
cls;
rndseed 1234;
a = rndn(100,1);
print “data=” a;
```

```
save d:datan.fmt = a;
```

プログラム（uniform distribution のケース）

```
new;
cls;
rndseed 1234;
a = rndu(100,1);
print “data=” a;
save d:datau.fmt = a;
```

まず１つ目のプログラムでは、rndseed 1234 としてやることによって seed の値を 1234 に設定しています。これにより、seed の値 1234 に依存した standard normal な分布の乱数が 100 個分、１列だけ発生されます。いつ何時だれがそのプログラムを実行しようと、同じ標準正規乱数が発生することになります。

２つ目のプログラムは、同様にして、seed の値を 1234 に設定しています。これにより seed の値 1234 に依存した uniform な分布の乱数が 100 個分、１列だけ発生されます。いつ何時だれがこのプログラムを実行しようとも、同じ一様乱数が発生します。

ここで、乱数発生のもとになる値 seed は、０より大きく、２の 32 乗よりも小さければ、どんな数でもかまいません。乱数の seed を設定する命令の後にスペースを一つ入れてからその後に任意の数字を入れることになります。例えば、1234 を設定すれば、後で GAUSS でプログラムを動かす指導者なりレフリーなり、だれでもいつ何時でも同じ結果が得られます。（２の 32 乗より小さいというのは、この乱数が GAUSS では 32 ビットの乱数発生システムでソフトウエア的に発生されるからです。）注意しなければならないのは、ここでの seed が０でもマイナスの値でもエラーになってしまうことです。

ポイント　いつでも同じ標準正規分布にふるまうM個のN列の乱数を発生させるには

rndseed **任意の数字**

rndn(M,N)

ポイント　いつでも同じ一様分布にふるまうM個のN列の乱数を発生させるには

rndseed **任意の数字**

rndu(M,N)

ポイント　**rndseed **の後の値は、０＜seed**＜２の** 32 **乗の値になければならない。**

ほかにも、βやγや poisson 分布などの代表的な分布の乱数を発生させる組込み関数が gauss には豊富に標準で装備されています。他のソフトウエアのように１つの乱数発生関数

から他の分布の乱数をプログラムを組んで発生させる必要はありません。
　ただし、ここで上のプログラムの代わりに、

```
seed=1234;
a = rndns(100,1,seed);

seed=1234;
a = rndus(100,1,seed);
```

などというふうに、乱数のもとになる seed を変数としてある値を代入し、それを第 3 要素として代入する方法も他人のプログラムを見るとよく見かけるかもしれません。rndn および rndu の後に s をつけて、第３要素に、変数として seed を代入しているのです。これらの方法は、GAUSS としてはもともとあるプログラム方法でしたが、上の２つ以外の新しくできた乱数のほとんどすべてが、rndseed などで外部的に seed を宣言するようになったので、古くからある rndns と rndus は最近のバージョン 4.0 では廃止されました。現在は動きますが、周知の期間を置いて、ゆくゆくは seed を内部的に与える方法ではプログラムはワークしなくなると思われます。
　最後に、すべての人が出くわすであろうエラーを取りあげたいと思います。それは、ファイル名の位置を表すディレクトリーの場所やフォルダーの間にスペースを入れてはいけないというものです。もし、ディレクトリーを変更するために、例えば、

```
save d:   datau.fmt = a;
```

とスペース（空白）が１文字以上入ってしまっただけで、プログラムはエラーを起こして動かなくなってしまいます。行列を{　　}の中に格納する場合や、プログラム自体をエディターに記入する場合には、基本的には、いくつスペースを入れても一向にかまいませんが、ファイル名やフォルダー名は空白を作らないように連続して記入してください。特に、プログラムを書きなおす場合によく起きるエラーが、このファイル名の位置の無用なスペースによるものです。気をつけましょう。