2.7 行列 論理式と論理 ver. 0.1

行列の節の最後には、行列データの扱い方でも、もっともプログラムらしいのですが、わかっていない人は全く見向きもされないようなトピックス、論理式と論理について解説します。質的な変数や文字がデータに入っている場合などの０と１の識別にはじまって、プログラムの論理的な構造の基礎になったり、プログラムに困ったときのちょっとした工夫になったり、これを熟知していると用途は多用に広がります。

まずは前回の続きから、printfmt のマスク(文字の列ならば０、数字の列ならば１)に困った時、論理式でひと工夫できることを示します。前回と基本的に同じプログラムです。

プログラム

```
new;  cls;
dataset="d:datafile0.dat";
open fh=^dataset;
print $getname(dataset)';
mask=readr(fh,1) .> 0.1;   print /rz mask;
__fmtcv={ "*.*s " 16 16 };
call printfmt(readr(fh,5),mask);
closeall fh;
```

画面表示

name	sex	listen	write	read
0	**0**	**1**	**1**	**1**
Ueda	F	85	82	95
Eda	M	91	75	83
Ota	F	89	70	79
Kida	M	80	83	70

上のプログラムでは前回、mask={ 0 0 1 1 1};として各列の変数のタイプを０(文字)と１(数字)として手動で mask という変数に代入しました。mask=vartypef(fh)'として変数タイプをプログラムで計算代入することは最近のバージョンでは可能ですが、3.2 を含めた以前のバージョンではエラーになることを述べました。そこは、GAUSS は完全なプログラム言語です。転置することなく、論理式をつかって、０と１に各列のタイプを見分けることが可能です。いろいろな論理式の作り方がありますが、太字のようにすることも一例です。ここでは、print 文で mask という変数自体も画面表示させています。そのほかの部分は、前回最後のプログラムと同じです。前回同様、GAUSS データファイル d:datafile0.dat がすでに作成されているものとします。上のプログラムは、どのようなバージョンでも動くはずですし、mask の変数をわざわざ転置する手間もありません。

ここでの論理は、readr(fh,1)でファイルハンドル fh で指定されているファイルの１行目

を読みこんだ変数（ここでは１×５の行ベクトル）を 0.1 とそれぞれの要素ごとに比較することで、もしそれぞれの値が 0.1 より大きければ True で１、そうでなければ False で０となります。そこで作られるもとの１×５の行ベクトルと同じディメンションのベクトルを mask という変数に＝の記号で代入します。わざと誤解されるように書きましたが、ここでは mask=readr(fh,1)がひとかたまりではなくて、readr(fh,1) .> 0.1 を先にドット比較で要素ごとにくらべてできた行ベクトルを＝のマークの前の mask に代入しています。いわば

mask=(readr(fh,1) .> 0.1)とでも考えるのが適当です。GAUSS では、文字は+DEN または非常に０に近い 10 のマイナス 300 乗あたりの正の数になっていることが普通です。正と言うことで、+DEN にもプラスのマークがついています。ここのデータでは文字以外はすべて正の整数であることはわかっていますから、0.1 とか１とかいうナンバーをさかいにして、それよりも大きければその論理式が True で１、そうでなければ False で０となるようにすれば、各列の文字タイプの０と１にこの場合は一致することになります。

ただ、これでは、もし数値が入っているところに０やマイナスの値、それに０に近い値があると論理式としてはうまく機能しません。そこで、論理式の and や or を使って２つ以上の論理式を結びつけて、それでもって０か１を返すようにできます。上の論理式のところを下のものに変更してみてください。

```
mask= (readr(fh,1) .> 0.00001) .or (readr(fh,1) .<= 0);
```

まったく同じ結果が得られると思います。しかし、今度は、データの１行目のおのおのの値が「0.00001 より大きい」かまたは「０以下(０を含む)」であれば、True で１を返し、そうでなければ(この場合 10 のマイナス何百乗であらわされる文字)、False で０を返すように論理が変更になっています。ここの or というのは論理の disjunction「または」に相当する演算記号で、その前にドットをつけることにより要素対要素で計算ができます。０から１の値が数値に見込まれるならば、0.00001 のところを変更して文字と数値が区別できるように論理構造を変更する必要があります。このように複雑な大小関係も括弧つきの論理式でそれぞれを書いてやって、and や or、それに xor などの論理で結びつけることが簡単にできます。加えて、and に似ているけれども F と F を結びつけたときに T を返す eqv、単一で使う not なども通常の論理と同じように使えて、要素対要素の場合にはドットをその演算記号の前につければ、ちょっとしたプログラムの困った時のひと工夫に使えます。

上の例は、ちょっとしたお遊びですが、論理は統計計量の分析のコアの部分にも使えることを以下で示しましょう。以前、０と１でできたダミー変数のを作る際に 2 つの方法を説明しました。１つは、単位行列を reshape で引き伸ばして季節ダミーを作る方法。もう１つは、組込み関数 dummy を使って条件に見合うダミー変数を作る方法がありました。ここでは、後者の組込み関数 dummy を使わずに、同じようなことを論理式でやってしまおうというわけです。この方法は、組込み関数 dummy を使うよりも一般的に広まってい

る方法です。今度は、横方向の行ベクトルではなくて、縦方向の列ベクトルを論理式で比較して、０と１のグループに分けます。

　いま、年次データが下のようになっていて、1989 年を含めて以降と、それよりも前をダミー変数で区別してみましょう。

プログラム

```
new;  cls;
   data={1980 300 70,
      1981 330 72,
      1982 360 73,
      1983 390 75,
      1984 420 77,
      1985 450 78,
      1986 570 79,
      1987 600 80,
      1988 630 83,
      1989 640 83,
      1990 650 84,
      1991 655 84,
      1992 656 85,
      1993 655 84,
      1994 658 86,
      1995 658 85,
      1996 655 84,
      1997 659 85,
      1998 660 86,
      1999 670 87,
      2000 680 88};
   year=data[.,1];
   d= year.>=1989;
   data=data~d;
   print /rz data;
```

画面表示

1980	300	70	0
1981	330	72	0
1982	360	73	0
1983	390	75	0

1984	420	77	0
1985	450	78	0
1986	570	79	0
1987	600	80	0
1988	630	83	0
1989	640	83	1
1990	650	84	1
1991	655	84	1
1992	656	85	1
1993	655	84	1
1994	658	86	1
1995	658	85	1
1996	655	84	1
1997	659	85	1
1998	660	86	1
1999	670	87	1
2000	680	88	1

上のプログラムは、data というデータ行列変数の１行目を year としたうえで、その値が要素対要素で 1989 よりも大きいか等しければ True で１、そうでなければ False で０となるような year と同じディメンションの列ベクトルを論理式で作って、それを d という変数に代入しています。その d をもとの data という変数に水平方向にマージしたものを、あらためて data とおきなおしています。ここで気をつけなければいけないことは、d=year ではないことです。誇張して書くならば、d=(year.>1989);ということで、括弧内の論理式の評価（True か False）を行なって、year と同じディメンションの変数の中に１か０を入れた上で、それをｄに代入しているのです。

今度は、1989 年から 1995 年までだけが違っているダミー変数を作る際のプログラムはどうなるでしょうか？それは、以前やった論理の.or を使って、というふうに、ダミー変数を作成するところを変更してみてください（符号の向きをかえましたので注意のこと）。

```
d=(year.<=1988) .or (year.>=1996);
```

画面結果

1980	300	70	1
1981	330	72	1

1982	360	73	1
1983	390	75	1
1984	420	77	1
1985	450	78	1
1986	570	79	1
1987	600	80	1
1988	630	83	1
1989	640	83	0
1990	650	84	0
1991	655	84	0
1992	656	85	0
1993	655	84	0
1994	658	86	0
1995	658	85	0
1996	655	84	1
1997	659	85	1
1998	660	86	1
1999	670	87	1
2000	680	88	1

少しわかりづらいですが、簡単に言えば（厳密ではありませんが)、1988 を含めてそれ以下の場合 True で１、それから、1996 を含めてそれ以上の場合も True で１、それ以外のすべて、この場合 1989 から 1995 までを０とする縦１列の列ベクトルを、ここでは変数ｄに代入しているのです。

上の変数タイプのマスクの例、それから年次でダミー変数を作る例の場合、少し厳密な説明を省いて直感的な説明をしましたので、後半では、GAUSS における論理式と論理について詳しく見ていきたいと思います。まずは、論理表の作成から

プログラム

```
new; cls;
a={1 1 0 0};
b={1 0 1 0};
x1=.not a;
x2=.not b;
x3=a .and b;
x4=a .or b;
x5=a .xor b;
```

```
x6=a .eqv b;
print "(True:1    False:0)";
print "-------------------------------------------------------------------------------";
print /rz "    a    " a;
print /rz "    b    " b;
print "-------------------------------------------------------------------------------";
print /rz "not a" x1; print /rz "not b" x2;
print "-------------------------------------------------------------------------------";
print /rz " and " x3;print /rz "    or " x4; print /rz " xor " x5; print /rz " eqv " x6;
print "-------------------------------------------------------------------------------";
```

画面表示

```
(True:1   False:0)
--------------------------------------------------------------------------------
  a                1                1                0                0
  b                1                0                1                0
--------------------------------------------------------------------------------
not a              0                0                1                1
not b              0                1                0                1
--------------------------------------------------------------------------------
 and               1                0                0                0
  or               1                1                1                0
 xor               0                1                1                0
 eqv               1                0                0                1
--------------------------------------------------------------------------------
```

上のプログラムは論理表を、Trueを１、Falseを０として作成したものです。行組みが一致しない場合には、引用符の中の空白の個数で１と０がそろうように調節をしてください。

上のような論理はわかってしまえば簡単ですが、GAUSS上では、以下の２つの約束事があります。１つ目は、Falseの０を比べるのであって、それ以外の数は１であっても２であっても、はたまた９であっても、論理計算では、全てまとめてNot Falseとして１と認識されます。もう１つは、複素数の場合、は上の表にある論理計算ではnotだけが可能で整数部分だけが比較されます。そのほかの論理はeqvも含めてすべてエラーになります。

まず、０であることを比べるのであって、それ以外の論理計算は何であっても１とされることを上のプログラムのｂのところの数を少し変更することによって見てみましょう。上のプログラムのうち、ｂのところの数値だけを次のように変更してみてください。

```
a={1 1 0 0};
b={3 5 2 0};
```

画面表示

(True:1　False:0)

```
--------------------------------------------------------------------------------
  a                1                1                0                0
  b                3                5                2                0
--------------------------------------------------------------------------------
not a              0                0                1                1
not b              0                0                0                1
--------------------------------------------------------------------------------
 and               1                1                0                0
  or               1                1                1                0
 xor               0                0                1                0
 eqv               1                1                0                1
--------------------------------------------------------------------------------
```

上の結果でわかるように、０以外の数字は１であろうと３であろうと５であろうと、すべて非零として扱われて論理計算が行なわれます。ですから、GAUSSでは、Falseが０で、Trueは非零と厳密には考えるようにしてください。

ポイント　論理においてGAUSSは、０と非零を比較して論理計算を行なう。

　次に、２番目の注意点の複素数の場合を比べる場合について例をあげて説明しておきましょう。まず、複素数を人工的につくるには、その整数部分とそうでない部分を別々に行列で作成しておき、それらをcomplexという組込み関数に代入してやってつなげば、複素数が返されます。以下がその例で、論理計算をやっています。

プログラム

```
new; cls;
a={1 1 0 0};
b={1 0 1 0};
x=complex(a,b);
y=complex(a,-b);
format /rzc 3,2;   print " x       " x;   print " y       " y;
print "-----------------------------------------------";
```

```
z1=.not x;
z2=.not y;
format /rz 10,2;  print " not x" z1;  print " not y" z2;
z3=x .eqv y;
print z3;
```

画面表示

```
x        1 +   1i,  1          0 +   1i,  0
 y        1 -   1i,  1           0 -   1i,  0
-------------------------------------------------
 not x          0          0          1          1
 not y          0          0          1          1

error G0043 : Not implemented for complex matrices
```

上のように、複素数を含む変数を論理で比較する場合、notは使えますが、それ以外のeqvを含むすべての論理演算はできません。エラーを返して、そこでプログラムは止まってしまいます。違った方法で比較する必要があります。上のプログラムで、/rzcのｃは、文字スペース間を１スペースとするのではなくて、かわりにカンマをつけるフォーマットを設定しています。複素数など隣の数と紛らわしい場合に、使用されるフォーマットです。なお、上のようなand、or、xor、eqvではなくて、その代わりに、/=、==、＞、＜などの大小関係を比較する演算はすべての場合に可能です。（ただし複素数には大小関係はないですから、＞と＜それに＝と組み合わせた論理式は無効になり、すべてFalseになります。） 以下がその例です。

プログラム

```
new; cls;
a={1 1 0 0};
b={1 0 1 0};
x=complex(a,b);
y=complex(a,-b);
format /rzc 3,2;
print " x       " x;
print " y       " y;
print "-------------------------------------------------";
format /rz 10,2;
z3=x .== y;
z4=x ./= y;
```

```
z5=x .>  y;
z6=x .<  y;
print "  ==  " z3;
print "  /=  " z4;
print "  >  " z5;
print "  <  " z6;
print "-----------------------------------------------";
```

画面表示

```
 x        1 +   1i,  1          0 +   1i,  0
 y        1 -   1i,  1          0 -   1i,  0
-------------------------------------------------
  ==           0          1          0          1
  /=           1          0          1          0
  >            0          0          0          0
  <            0          0          0          0
-------------------------------------------------
```

上のように、＞と＜は大小関係を比べられないので意味をなしませんが、eqvではなくて==の記号を使うことによって、複素数の比較の問題は解決します。シングルの＝の記号ではないことに注意してください。シングルの＝の記号は、後ろのものを前の変数に代入する意味に使われるのに対して、ダブルの＝＝は、２つのものが同じかどうか比べて、同じであれば１を返し、違えば０を返す論理式記号です。違うものを比較する場合には、/=の記号を使います。これは、等しくないという意味です。

ポイント　複素数の論理比較では、and,or,xor,eqvは使用できない。代わりに、==や/=などや大小関係>,<などを使う。なお、notは使用可。

以下のプログラムは、すべての大小関係および等号関係を比較する論理式の例です。１から９までの数からなる行ベクトルと５という数字を比べることによって、行ベクトルと同じディメンションの０と１からなる論理結果をそれぞれの記号の論理式について表にしてあります。以前と同様、各文字のスペースや区切りの長さは各自で調節してください。

プログラム

```
new; cls;
a={1 2 3 4 5 6 7 8 9};
x1=a.==5;
x2=a./=5;
```

```
x3=a.>5;
x4=a.>=5;
x5=a.<5;
x6=a.<=5;
format /rz 4,2;
print " a " a;
print "-----------------------------------------------";
print "==5" x1; print "/=5" x2; print " >5" x3;
print ">=5" x4; print " <5" x5; print "<=5" x6;
print "-----------------------------------------------";
```

画面表示

```
 a    1    2    3    4    5    6    7    8    9
-----------------------------------------------
==5   0    0    0    0    1    0    0    0    0
/=5   1    1    1    1    0    1    1    1    1
 >5   0    0    0    0    0    1    1    1    1
>=5   0    0    0    0    1    1    1    1    1
 <5   1    1    1    1    0    0    0    0    0
<=5   1    1    1    1    1    0    0    0    0
-----------------------------------------------
```

上の例では、==５のところでは、１から９までの数字のうち５だけがTrueで１になって、残りのすべてはFalseで０になっています。論理式の比較記号の左側の変数のディメンションと同じディメンションの０と１からなる変数ができあがりました。/=のところは、０と１とが逆転した変数になるはずです。＞５のところでは、６，７，８，９がそれぞれ５よりも大きいという意味でTrueですから１が入って、残りのところはすべてFalseで０が入ります。以下同様に、等号不等号関係が示されています。なお、不等号と等号の組合せの場合には、必ず不等号を先に書いて、その次に等号を書かなければなりません。

同じことですが、プログラムによっては、==,/=,>,>=,<,<=の代わりにその意味の２文字からなる略語で論理式を作っている場合がありますので、上とまったく同じ動作をするプログラムを２文字略語の大小等号不等号関係の論理式で書いてみます。

プログラム

```
new; cls;
a={1 2 3 4 5 6 7 8 9};
x1=a.eq 5;
x2=a.ne 5;
```

```
x3=a.gt 5;
x4=a.ge 5;
x5=a.lt 5;
x6=a.le 5;
format /rz 4,2;
print "  a " a;
print "-----------------------------------------------";
print "eq 5" x1; print "ne 5" x2; print "gt 5" x3;
print "ge 5" x4; print "lt 5" x5; print "le 5" x6;
print "-----------------------------------------------";
```

画面表示

```
  a    1    2    3    4    5    6    7    8    9
-------------------------------------------------
eq 5   0    0    0    0    1    0    0    0    0
ne 5   1    1    1    1    0    1    1    1    1
gt 5   0    0    0    0    0    1    1    1    1
ge 5   0    0    0    0    1    1    1    1    1
lt 5   1    1    1    1    0    0    0    0    0
le 5   1    1    1    1    1    0    0    0    0
-------------------------------------------------
```

上の場合も、前の例と同様に、要素対要素で、ドットが２文字の前に必要です。以下、略号と大小等号不等号関係の対応を示しておきます。すべて英語の意味の略です。大文字小文字は区別はありません。

ポイント

==	EQ	(equalの英語の略)
/=	NE	(not equalの英語の略)
>	GT	(greater thanの英語の略)
>=	GE	(greater than or equalの英語の略)
<	LT	(less thanの英語の略)
<=	LE	(less than or eqalの英語の略)

これらの論理計算では、ドットをともなう要素対要素の計算では、これらの記号よりも前の変数と同じディメンションの変数が一時的に作られて、Trueであれば１、Falseであれば０が該当する部分に入る。

厳密な論理計算よりも、ファジーな擬似論理計算をする関数がGAUSSには備えつけられています。サーチしていき、完全には０ではないものの、０に近い数で計算をやめたり、

off diagonalの値が計算精度の関係で完全には０にならないときに０であると判断したり実際のプログラムでは厳密な論理計算を望まない局面に遭遇する場合がありますが、このような時に使われるものがファジー論理の関数群です。上の英語の略の２文字の前にｆの文字をつけて、関数形で２変数を比べます。要素対要素の比較の場合には、さらにこの前にdotという文字がつきます。以下が、要素対要素の場合のファジー論理の例です。

プログラム

```
new; cls;
a1=10^(-13); a2=10^(-14); a3=10^(-15); a4=10^(-16); a5=10^(-17);
a=a1~a2~a3~a4~a5; b=10^(-15);
x1=dotfeq(a,b); x2=dotfne(a,b); x3=dotfgt(a,b);
x4=dotfge(a,b); x5=dotflt(a,b); x6=dotfle(a,b);
format /ro 12,4;
print "-------------------------------------------------------------------------";
print "     a    " a;
print "-------------------------------------------------------------------------";
format /rz 12,4;
print "dotfeq" x1; print "dotfne" x2; print "dotfgt" x3;
print "dotfge" x4; print "dotflt" x5; print "dotfle" x6;
print "-------------------------------------------------------------------------";
print "                                        (b=1.000e-015)";
```

画面表示

```
-------------------------------------------------------------------------
   a     1.000e-013   1.000e-014   1.000e-015   1.000e-016   1.000e-017
-------------------------------------------------------------------------
dotfeq          0            0            1            1            1
dotfne          1            1            0            0            0
dotfgt          1            1            0            0            0
dotfge          1            1            1            1            1
dotflt          0            0            0            0            0
dotfle          0            0            1            1            1
-------------------------------------------------------------------------
                                        (b=1.000e-015)
```

上のプログラムでは、10のマイナス13乗から17乗までの数を水平方向にマージして１×５の行ベクトルaをつくって、そのまんなかの値10のマイナス15乗と要素対要素のファジー比較をしています。上のように、1.000e-015と比べるのに、その許容数（許容桁数）がそれ

と同じであると論理結果は、通常の場合と異なってきます。ファジーに1.000e-015と等しいのは、厳密な比較とは異なり、1.000e-015以下の数すべてがTrueで１となります。すなわち、その許容デフォルト値10のマイナス15乗よりも小さい数はすべて論理計算がファジーになります。

もし、ファジー論理ではあるけれども、それが無視されるような小さな許容数（許容桁数）にするには、比べてやるものの最小の数と同じか、もしくは、それよりも小さい値にファジー比較の許容数（許容桁数）を設定しなおす必要があります。これには、GAUSSはグローバル変数_fcmptolが用意されていて、そのデフォルト値1e-15を違う値を代入することによって変更できるようになっています。上のプログラムのファジー論理のでてくる前に次のようなグローバル変数を置いてみてください。fcmptolとは、fuzzy comparison tolerance levelの英語の略です。

```
_fcmptol=10^(-17); または _fcmptol=1e-17;
```

そうすると、いままでファジー論理比較の最大許容数が10のマイナス15乗であったものが10のマイナス17乗に変更ができて、以下のように、通常の厳密な論理比較とまったく同じ結果が得られます。実際の数字、例えば0.001であれば、0.001を書いてもかまいませんし、10^(-3)と計算式にしてもかまいませんし、指数表示で1e-3としてもかまいません。

画面表示

a	1.000e-013	1.000e-014	1.000e-015	1.000e-016	1.000e-017
dotfeq	0	0	1	0	0
dotfne	1	1	0	1	1
dotfgt	1	1	0	0	0
dotfge	1	1	1	0	0
dotflt	0	0	0	1	1
dotfle	0	0	1	1	1

(b=1.000e-015)

これらファジー論理比較を関数として用いる際には、その比較されるもの、すなわち第１要素に入るものは、必ず、列ベクトルでなければならないというルールがあります。通常のすべての論理や論理式、論理比較では行列一般を比較できるのに対して、ファジーな論理比較では、列ベクトルしか許されていないことに注意してください。なお、dotはスケラー対スケラーの比較には不要です。Dotはドット、fはファジーの英語の略です。

ポイント　ファジーな==　feq(a,b)　要素対要素の場合　dotfeq(a,b)
ファジーな/=　fne(a,b)　dotfne(a,b)
ファジーな＞　fgt(a,b)　dotfgt(a,b)
ファジーな>=　fge(a,b)　dotfge(a,b)
ファジーな＜　flt(a,b)　dotflt(a,b)
ファジーな<=　fle(a,b)　dotfle(a,b)
許容レベルグローバル変数のデフォルトは、_fcmptol=1e-15;

また、GAUSSでは２つの集合の和集合、積集合、差集合などを得ることもできます。まずは、集合の要素が数字の場合についての例です。

プログラム

```
new; cls;
a={1,1,9,2};
b={1,0,2,0};
x1=union(a,b,1);
x2=intrsect(a,b,1);
x3=setdif(a,b,1);
x4=unique(a,1);
x5=unique(b,1);
print "-------------------------------------------------------------";
print "a              " a'; print "b              " b';
print "-------------------------------------------------------------";
print /rz "union        " x1';
print /rz "intersect " x2';
print /rz "difference" x3';
print /rz "unique a    " x4';
print /rz "unique b    " x5';
print "-------------------------------------------------------------";
print "(Both a and b are originally column vectors.)"
```

画面表示

```
-------------------------------------------------------------
a                      1             1             9             2
b                      1             0             2             0
-------------------------------------------------------------
```

```
union           0       1       2       9
intersect       1       2
difference      9
unique a        1       2       9
unique b        0       1       2
-------------------------------------------------------------
(Both a and b are originally column vectors.)
```

unionは和集合、intrsectは積集合、setdifは差集合を求める関数で、それぞれの関数の第３要素の１は、フラッグといって、マスクと同じように１ならば数字、０ならば文字を入れるます。フラッグは０または１のスケラーで、０ならばＡＢＣ順に結果がソートされ、１ならば小さいものから順に結果がソートされる機能があります。なお、関数uniqueは、その１つの集合にある要素を重複なく小さいもの順またはＡＢＣ順にソートして書き出す関数です。同じものは１回しか書き出されません。なお、上のプログラムでは、aおよびｂの列ベクトルを集合計算して、それを視覚的にわかりやすいように転置して行ベクトルに変換してあります。GAUSSでは、集合計算は必ず縦方向の列ベクトルある必要がありますので注意してください。同様に、文字のデータの場合に２つの文字データ列ベクトルの集合を考えるのが以下の例です。今度は、文字ということで、フラッグは０になっています。

プログラム

```
new; cls;
a={"George","Bill","Bill","George"};
b={"George","George","Bob","George"};
x1=union(a,b,0);
x2=intrsect(a,b,0);
x3=setdif(a,b,0);
x4=unique(a,0);
x5=unique(b,0);
print "-------------------------------------------------------------";
print "a              " $a'; print "b              " $b';
print "-------------------------------------------------------------";
print /rz "union        " $x1';
print /rz "intersect " $x2';
print /rz "difference" $x3';
print /rz "unique a    " $x4';
print /rz "unique b    " $x5';
print "-------------------------------------------------------------";
```

```
        print "(Both a and b are originally column vectors.)"
```

画面表示

```
-------------------------------------------------------------
a                George          Bill          Bill        George
b                George        George           Bob        George
-------------------------------------------------------------
union              Bill           Bob        George
intersect        George
difference         Bill
unique a           Bill        George
unique b            Bob        George
-------------------------------------------------------------
(Both a and b are originally column vectors.)
```

上では、集合の要素が数字である場合とまったく同じことをその要素が文字である場合についてやっています。文字の入ったベクトルの扱いについては、以前と同じように、関数内においてはそのまま変数を代入して操作しますが、print出力の際には、変数の前に＄のマークをつけて変数であることを明示した上で出力します。和集合、積集合、差集合、それにユニークな要素を返す関数すべて、文字を扱う際には、そのフラッグは１ではなく０になり、結果はすべてアルファベット順で返されます。間違えて、フラッグの部分を１のままにすると、一応計算はされますがアルファベット順にはなりません。

ポイント　union(ａ,ｂ,フラッグ)　　集合aと集合ｂの和集合を返す
intrsect(ａ,ｂ,フラッグ)　　積集合を返す
setdif(ａ,ｂ,フラッグ)　　差集合を返す
unique(ａ,フラッグ)　　集合aの要素を重複なしで返す
フラッグ　０：文字（アルファベット順）　１：数字（小さい順）
a,bは必ず列ベクトル。２列以上の行列は不可。

以上が、行列（その狭義のベクトルを含む）を論理で比較したり論理計算をしたりする手続き、および集合の概念の計算について説明しました。GAUSSでの行列を扱うトピックスには、まだカバーされていないところもありますが、残りの部分は、次節のプログラムのところおよび、それ以降で順次必要に応じて取り上げたいと思います。