Paper 268-25

Multidimensional arrays in SAS/IML® Software

Michael Friendly
York University

Abstract

This paper describes a data structure and a set of SAS/IML modules for handling multidimensional arrays in a flexible way. The data structure provides variable names and labels for the levels of the table for all dimensions, along with the table data. The processing modules ilustrate:

Generalized transpose of a multidimensional array, as in *APL*.
Applying a reduction operation over specified coordinates.
Printing a labeled multidimensional array.
Producing LaTeX and HTML tables from *n*-way arrays.
Transferring such data between SAS/IML and other software.

The usefulness of this approach is illustrated with examples from my work on graphical methods for categorical data.

1 Introduction

SAS/IML contains a rich variety of matrix operations and built-in functions for statistical computation and graphics. Other array-oriented programming languages, such as *Mathematica*, *APL* and *MATLAB*, provide some of these facilities, but the combination of high-level operations, statistical functions, and graphics in SAS/IML is hard to beat for most of the statistical graphics applications I have developed.

At least that *was* true until I began work on graphical methods for categorical data, where multidimensional arrays are the rule, rather than the exception, in contrast to quantitative data, where two-dimensional arrays (tables, SAS data sets) handle almost all of ones needs.

Table 1: Survival on the Titanic

			Class			
Gender	Age	Survived	1st	2nd	3rd	Crew
Male	Adult	Died	118	154	387	670
Female			4	13	89	3
Male	Child		0	0	35	0
Female			0	0	17	0
Male	Adult	Survived	57	14	75	192
Female			140	80	76	20
Male	Child		5	11	13	0
Female			1	13	14	0

Table 1 illustrates a four-way contingency table (4 2 2 2), containing the numbers of passengers and crew on the *Titanic*, classified by Class, Sex, Age, and Survival. How can we represent such data for analysis and graphics?

A SAS dataset, such as might be produced by `PROC FREQ`, structures the frequencies as a single variable (`COUNT`), together with factor variables (`Class Sex Age Survived`), indicating the cell of the four-way table to which the `COUNT` belongs.

```
Class     Sex      Age      Survived   COUNT

1st       Male     Adult    Died         118
2nd       Male     Adult    Died         154
3rd       Male     Adult    Died         387
Crew      Male     Adult    Died         670
1st       Female   Adult    Died           4
2nd       Female   Adult    Died          13
3rd       Female   Adult    Died          89
Crew      Female   Adult    Died           3
  ... (24 more data lines) ...
```

But in such a data set, the 4-way structure of the frequencies is implicitly contained in the order and arrangement of the factor variables. This makes it hard to do computations and graphics with multidimensional arrays.

The observations in the dataset appear in row-major order (Class varying most rapidly, Survived least rapidly). If we wished to rearrange the data so that Survived varied most rapidly, we might try

```
proc sort data=titanic;
  by class gender age survived;
```

Unfortunately, this does not work well for character variables that represent ordered factors. A factor with levels 'Low', 'Medium', 'High', for example would be ordered as 'High', 'Low', 'Medium'; even worse, the result for the `class` variable here would depend on the collating sequence of your machine!

An alternative structure consists of a set of SAS/IML variables containing the variable names (`VNAMES`), the number of levels of each variable (`LEVELS`), and the labels for those levels (`LNAMES`), as shown below. With the convention that the table entries (`COUNT` in the dataset) are arranged in row-major order (first variable varying most rapidly), these variables (plus the table) represent the same information as in the dataset, but turn out to be much more convenient in SAS/IML.

```
VNAMES       LEVELS    LNAMES
Class             4    1st       2nd       3rd       Crew
Sex               2    Male      Female
Age               2    Adult     Child
Survived          2    Died      Survived
```

(The `LEVELS` and `COUNT` variables are in the form required by the IML function `IPF` which fits loglinear models by iterative proportional fitting.)

With this data structure, I can enter the *Titanic* data in SAS/IML using the statements,

```
proc iml;
  levels = f4 2 2 2g;
  vnames = f'Class' 'Sex' 'Age' 'Survived'g;
  lnames = f'1st' '2nd' '3rd' 'Crew',
           'Male'  'Female' '' '',
           'Adult' 'Child'  '' '',
           'Died'  'Survived'   '' ''g;
  table = f
/* 1    2     3 Crew     Survive    Age      Sex    */
 118  154   387  670,    /* No      Adult    Male   */
   4   13    89    3,    /* No      Adult    Female */
   0    0    35    0,    /* No      Child    Male   */
   0         17    0,    /* No      Child    Female */
  57   14    75  192,    /* Yes     Adult    Male   */
 140   80    76   20     /* Yes     Adult    Female */
   5   11    13    0,    /* Yes     Child    Male   */
   1   13    14    0,    /* Yes     Child    Female */
       g;
  data='titanic';
  title='Survival on the Titanic';
```

With the processing modules for handling such arrays, the LaTeX version of Table 1 is then produced by an IML module MD2TEX as

```
run md2tex(levels, table, vnames, lnames, data, title);
```

The same table can be printed to an HTML file

```
run md2html(levels, table, vnames, lnames, data, title);
```

(or to a listing file, with an analogous `run md2print`).

More importantly, there are various operations we may wish to perform on such a multidimensional array. Two fundamental operations are

generalized transpose: For a two-way array, the transpose `t(A)` interchanges rows and columns. For an *n*-way array, generalized transpose permutes the dimensions.

generalized reduction: For a two-way array, `A[+,]` gives sums over rows, and `A[,+]` gives sums over columns. SAS/IML provides other subscript reduction operators for means (`:`), products (`#`) and some others. For an *n*-way array, we would like analogous operations.

We describe these extensions to SAS/IML below. They are the basic tools for manipulating *n*-way arrays. They also serve as the basis for graphical methods of *n*-way contingency tables as illustrated by mosaic displays and other methods described in *Visualizing Categorical Data* (Friendly, 1999b, *VCD*).

2 Generalized transpose

For a matrix $\boldsymbol{A} = \{a_{ij}\}$, the transpose operation interchanges rows and columns so that $\mathtt{t}(\boldsymbol{A}) \equiv \boldsymbol{A}^{\mathsf{T}} = \{a_{ji}\}$,

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \qquad \mathtt{t}(\boldsymbol{A}) = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Similarly, for an *n*-way array, $\boldsymbol{A} = \{a_{ijkl\cdots}\}$, the generalized transpose reorders the dimensions of the array according to a permutation of its indices. A three-way array, of size $I \times J \times K$, is turned "inside-out" by

$$\mathtt{t}_{321}(\boldsymbol{A}) = \{a_{kji}\}$$

For example, the display below shows a $2 \times 2 \times 2$ array transposed with the permutations (3,2,1) and (3,1,2).

```
 ORDER   1    2    3     -->    ORDER   3    2    1
           B1      B2                     B1      B2
         A1   A2   A1   A2              C1   C2   C1   C2
C1        1    2    3    4      A1       1    5    3    7
C2        5    6    7    8      A2       2    6    4    8

                             -->    ORDER   3    1    2
                                          A1      A2
                                        C1   C2   C1   C2
                                B1       1    5    2    6
                                B2       3    7    4    8
```

The ability to be able to perform such operations as summing or averaging over specified dimensons depends on being able to reorder the dimensions in any specified way.

2.1 Generalized transpose in SAS/IML

SAS/IML has only two-way matrices, but the functions `IPF` and MARG both represent *n*-way arrays using a vector of LEVELS to specify the table dimensions. The MARG function is designed to calculate the sums (margins) over any number of variables, specified by an array of coordinates for which sums are desired. To find the marginal totals for the variables A and B (summing over C) in the example above,

```
config = f1,
          2g;
call marg(loc,marginal,dim,table,config);
print marginal;
```

This produces:

```
MARGINAL
      6      8     10     12
```

If all coordinates are included in the configuration, the table is returned unchanged:

```
config = f1,
          2,
          3g;
call marg(loc,marginal,dim,table,config);
print marginal;
MARGINAL
    1      2      3      4      5      6      7      8
```

However, if the column in CONFIG is a permutation of the integers `1:n`,

```
config = f3,
          2,
          1g;
```

the same call to MARG gives:

```
MARGINAL
    1      5      3      7      2      6      4      8
```

These are the elements in the array transposed with the permutation `(3;2;1)`. This use of the MARG function[1] is the basis for generalized transpose, shown here as a demonstration function,

[1] The MARG function requires that the table entries be non-negative, but not necessarily integers. If the table contains negative values, it is necessary to add a quantity to all entries first, then subtract that quantity after transposing the array.

```
start margdemo(dim, tab, order);
  ord = t(order);
  lev = t(dim);
  run marg(loc,table,lev,tab,ord);
  table = row(table);
  print order, table;
  finish;

order={3 1 2};
run margdemo(dim, table, order);

order={3 2 1};
run margdemo(dim, table, order);
```

These produce the elements in TABLE ordered as shown in the transpose examples at the start of this section:

```
ORDER    3    1    2
TABLE    1    5    2    6    3    7    4    8

ORDER    3    2    1
TABLE    1    5    3    7    2    6    4    8
```

The ordinary matrix transpose is reflexive, so that $(\boldsymbol{A}^{\mathsf{T}})^{\mathsf{T}} = \boldsymbol{A}$, but the general rule is that a transposed array is restored by a second transpose using the *anti-ranks* (inverse permutation) of the original permutation vector:

```
run transpos(dim, table, vnames, lnames, order);
r = rank(row(order));
anti = r;
anti[,r]=1:ncol(row(order)); *-- anti-ranks;
run transpos(dim, table, vnames, lnames, anti);
```

For a three-way array,

$$t_{321}\,[t_{321}\,(\boldsymbol{A})] = \boldsymbol{A}$$

but

$$t_{312}\,[t_{231}\,(\boldsymbol{A})] = \boldsymbol{A}$$

The data structure for *n*-way arrays also comprises the vectors of LEVELS and variable names (VNAMES) and the character matrix (LNAMES) whose rows give the factor level names. To complete the transpose for this data structure, we must also reorder these arrays using the same permutation vector. The module TRANSPOS, listed in the Appendix, carries out this operation. As well, it allows the permutation of the variables to be expressed as a permutation of the integers or of the variable names.

2.2 Reordering levels along any dimension

For two-way arrays, we can reorder the rows or columns using the subscript ([,]) operator by specifying a permutation of the row or column indices within the subscript brackets. For example, with a 3 × 4 matrix A, the expression A[{2 3 1},] puts the first row in last position, and A[,4:1] reverses the order of the columns.

Using generalized transpose we can also rearrange an *n*-way array along any dimension by: (a) transposing so that the dimension to be reordered is in first position (warying most rapidly), (b) reshaping the result to a matrix with that dimension as its columns, (c) re-ordering the columns as desired, (d) transposing that array so that the dimensions are back in the original order.

Why should we want to do this? When the *n*-way array is a table of means for an *n*-way-factorial design, (and the factors are not intrinsically ordered), ordering each factor according to the main-effect means gives a more coherent table in which it is easier to see the trends over each factor. When the array is an *n*-way contingency table of frequencies, ordering each factor according to the principal correspondence analysis dimension gives a table in which the patterns of association among variables are more apparent (Friendly, 1992a, 1994). The general principle is that for unordered factor, order the levels *according to the effects you'd like to see* (Friendly, 1999c).

For example, here is a three-way (3 × 2 × 4) table with variables ordered C, B, A. To help see the effect of reordering, each entry contains the level subscripts of A, B, C as its integer part, and its ordinal position in the table as fractional part:

```
   B                      1                      2
   C       1       2      3       1       2      3
A
A1    111.01  112.02  113.03  121.04  122.05  123.06
A2    211.07  212.08  213.09  221.10  222.11  223.12
A3    311.13  312.14  313.15  321.16  322.17  323.18
A4    411.19  412.20  413.21  421.22  422.23  423.24
```

If SAS/IML allowed three-way arrays, we could reverse the order of levels along the first dimension (variable C) by the expression TABLE[3:1,,]. Using the module DIMORDER, this is accomplished by

```
run dimorder(dim, lnames, vnames, table, 1, dim[1]:1);
run mdprint(dim, table, vnames, lnames);
```

where the last two arguments to dimorder give the dimension to be reordered, and a permutation of its indices. This gives

```
   B                      1                      2
   C       3       2      1       3       2      1
A
A1    113.03  112.02  111.01  123.06  122.05  121.04
A2    213.09  212.08  211.07  223.12  222.11  221.10
A3    313.15  312.14  311.13  323.18  322.17  321.16
A4    413.21  412.20  411.19  423.24  422.23  421.22
```

Note that as with the generalized transpose operation, the elements in LNAMES are rearranged in corresponding order.

2.3 Applications

These operations were developed out of need to rearrange *n*-way contingency tables in a flexible way so as to reveal the patterns of association between variables in graphic displays.

Association-ordered displays

For example, Table 2 shows data on the relation between hair color and eye color among 592 subjects (students in a statistics course) collected by Snee (1974).

A mosaic display (Friendly, 1994, 1999b) attempts to show the pattern of association between variables. The rectangles in Figure 1 have areas proportional to cell frequency. Each cell is shaded according to the residual from independence, using varying shades of blue for cells with greater than expected frequencies, and shades of red for cells with less than expected frequencies. Figure 1 does not reveal the nature of the association between hair color and eye color, however, because the row and column variables were arranged in alphabetical order, as if we had sorted the dataset by

```
proc sort data=haireye;
   by hair eye;
```

Table 2: Hair-color eye-color data

Eye Color	Hair Color Black	Brown	Red	Blond	Total
Green	5	29	14	16	64
Hazel	15	54	14	10	93
Blue	20	84	17	94	215
Brown	68	119	26	7	220
Total	108	286	71	127	592

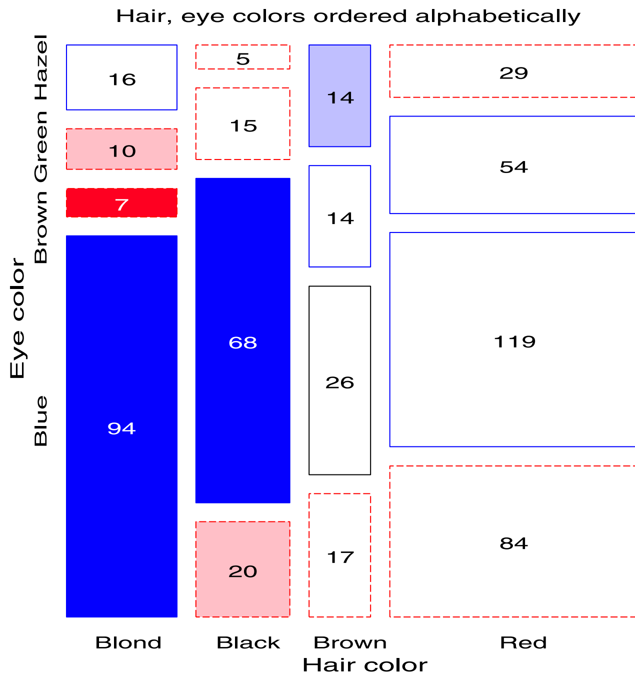


Figure 1: Mosaic display for the hair color, eye color data, ordered alphabetically

The pattern *is* revealed in Figure 2, where the hair color and eye color categories both rearranged so the the positive and negative residuals generally appear in opposite corners of the display. Here, we can see that both variables are arranged in order from dark to light, and the positive residuals in the opposite corners suggest that hair color and eye color vary together along this dark–light dimension.

In the mosaic displays application (Friendly, 1992b), the ordering of each variable is found using a correspondence analysis (CA) of the frequencies, ordering the levels by the category scores on the largest dimension.

Mosaic matrices

An extension of mosaic displays (Friendly, 1999a) provides an analog for categorical data of the scatterplot matrix used widely for quantitative data. For an *n*-way table, the mosaic matrix displays the `n(n − 1)` bivariate mosaics for all pairs of variables. The simplest case shows the bivariate marginal association between each pair, as shown in Figure 3. A further extension displays the condi-

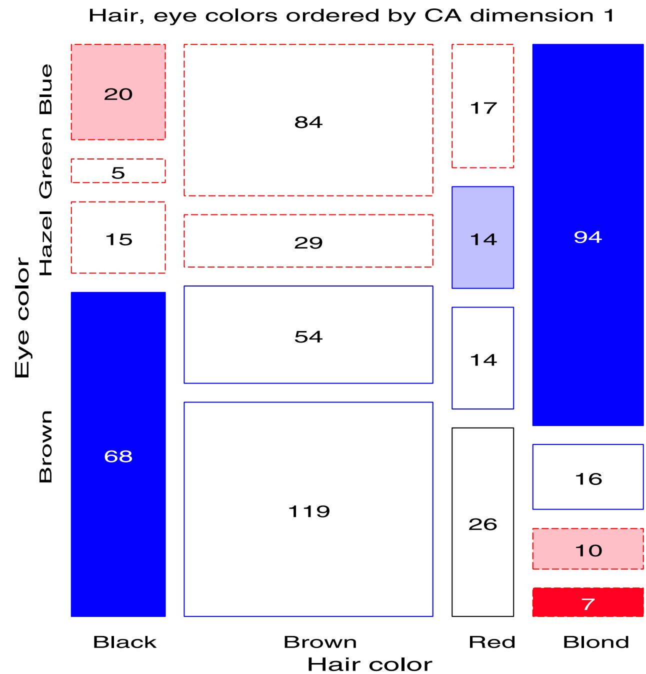


Figure 2: Hair, eye colors, ordered by CA dimension 1

tional association holding other variables fixed.

To accomplish this, the *n*-way array is transposed for each pairwise plot, putting the `col` and `row` variables first in the ordering, then calling the `MOSAIC` module to produce the plot for that panel. The separate plots are composed into one figure using `PROC GREPLAY`.

```
do row = 1 to factors;
  do col = 1 to factors;
    others = remove( (1:factors), (row||col) );
    *-- transpose table to conform to model;
    order = col||row||others;
    dm = dim;
    vn = vnames;
    ln = lnames;
    tab = table;
    run transpos(dm, tab, vn, ln, order);
    run mosaic(dm, tab, vn, ln, plots, title);
    end; /* do row */
  end; /* do col */
```

3 Generalized Reduction

The subscript operators + (sum), : (mean), # (product), <> (minimum), and so forth provide a way to apply an operation over rows (e.g., `A[+,]` – column sums) or columns (e.g., `A[:,]` – row means), or both (e.g., `A[#,#]` – product of all) for any matrix. How can we generalize this for *n*-way arrays? In *APL2*, which provides a general reduction operator for *n*-way arrays this is just `op/[config]A`, where `op` is the operation to be performed and `config` is the set of dimensions over which the reduction is carried out (e.g., `+/[1]A` sums an array over its first dimension.

As with reordering along a dimension, the key to doing this operation in SAS/IML for general arrays is to transpose the array so

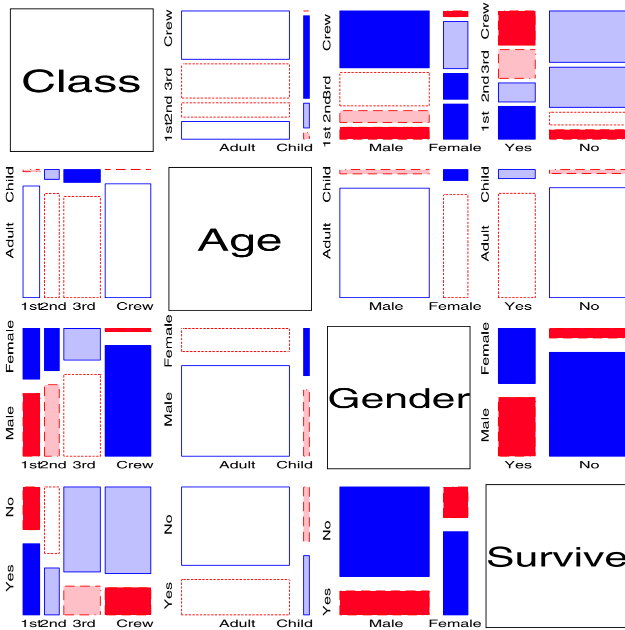


Figure 3: Mosaic matrix for Titanic data, bivariate marginal associations.

that the dimensions to be reduced appear as the rows (say) of a two-way table. If op is a character string containing the operation to be performed (e.g., +, :, #, <>, etc.), and tab is this re-shaped two-way table, then the generalized reduction may be calculated over the rows with

```
call execute( 't = tab[' + op + ',];');
```

Say config is a vector of dimension indices for which sums, means, products, etc. are desired. Then the permutation of dimensions (ord) to put the config dimensions first (as columns of a matrix) is calculated as

```
others = remove(1:nrow(dim), config);
ord = config || others ;
```

This may be transposed and re-shaped to a matrix with the config variables as its' columns with

```
run transpos(dim, tab, vnames, lnames, ord);
tab = shape(tab, 0, (dim[config,])[#]);
```

The module REDUCE carries out this generalized reduction, and is called as

```
run reduce(dim, table, vnames, lnames, config, op, expand);
```

If the last argument (expand) is positive, the result is expanded to match the size and shape of the input table argument. This is convenient for calculating residuals from a generalized linear model.

To illustrate, for the two-way (4 3) table

```
   A  a1  a2  a3  a4
B
b1     1   2   3   4
b2     5   6   7   8
b3     9  10  11  12
```

we can get row, column, and grand totals using

```
run reduce(dim, tab, vnames, lnames, 1, '+', 0);
run reduce(dim, tab, vnames, lnames, 2, '+', 0);
run reduce(dim, tab, vnames, lnames, f1 2g, '+', 0);
```

giving printed output

```
CONFIG  OP  EXPAND     TAB
     1 +         0      15    18    21    24

CONFIG  OP  EXPAND     TAB
     2 +         0      10    26    42

CONFIG  OP  EXPAND     TAB
     1 +         0      78
     2
```

Any of these may be expanded to a table of the same size and shape as the original by supplying a positive value for the last (expand) argument. For example the last example,

```
run reduce(dim, tab, vnames, lnames, 1 2, '+', 1);
```

gives

```
CONFIG  OP  EXPAND     TAB
     1 +         1      78    78    78    78
     2                  78    78    78    78
                        78    78    78    78
```

Similarly, the using the same data values re-shaped as a three-way (2 2 2) table,

```
   B      b1      b2
   A  a1  a2  a1  a2
C
c1     1   2   3   4
c2     5   6   7   8
c3     9  10  11  12
```

we can get means over each set of coordinates as follows:

```
run reduce(dim, tab, vnames, lnames, 1, ':', 1);
run reduce(dim, tab, vnames, lnames, 2, ':', 1);
run reduce(dim, tab, vnames, lnames, 3, ':', 1);
run reduce(dim, tab, vnames, lnames, f1 2g, ':', 1);
run reduce(dim, tab, vnames, lnames, f1 3g, ':', 1);
run reduce(dim, tab, vnames, lnames, f2 3g, ':', 1);
run reduce(dim, tab, vnames, lnames, f1 2 3g, ':', 1);
```

4 Printing tables: output, LaTeX & HTML

When there are more than two dimensions, printing an *n*-way table requires that some of the table variables be printed across the columns of a table, and the rest are printed down the rows.

4.1 Printed output

The printed tabular output in earlier sections illustrates the output produced by the SAS/IML module MDPRINT. This routine calculates the largest field width and number of decimals necessary to display the table entries, and also the widths required for the variable names and factor level names. (A global input variable, maxdec may be used to control the maximum number of decimals printed.

Using this information, it determines the maximum number of table variables to be allocated to the columns of the table to fit within the current linesize, taking the variables in their order in the

vnames vector. (This number may be restricted by the global input variable maxcol). The heuristic used is that printed tables should be as wide as possible to conserve space, all other things being equal. The remaining variables are allocated to the rows of the printed table. The table variables may be permuted first, of course, to achieve any desired printed output.

The MDPRINT module provides a basic format which could readily be adapted to other media.

4.2 HTML output

Printing HTML tables is easier in some ways than printed output, because the browser takes care of rendering the table with columns nicely aligned, given the HTML input properly marked-up with <TR>, <TH> and <TD> tags. In other ways it is slightly more tedious, because the HTML tags and attributes must be combined with the table data. Fortunately, the + operator performs concatenation of character values in SAS/IML, so the basic HTML to display table values can be written simply as

```
do r=1 to rows;
   do c=1 to cols;
   line = '<td align=right>'
          + right(char(tab[r,c],fw,dec)) + '</td>';
   put line;
   end;
```

(after the table has been reshaped to a matrix of size rows by cols).

My goals for HTML tables were somewhat higher, because they allow spanning rows and columns and varying fonts or background colors, which could be used to make tables far more attractive than in print form.

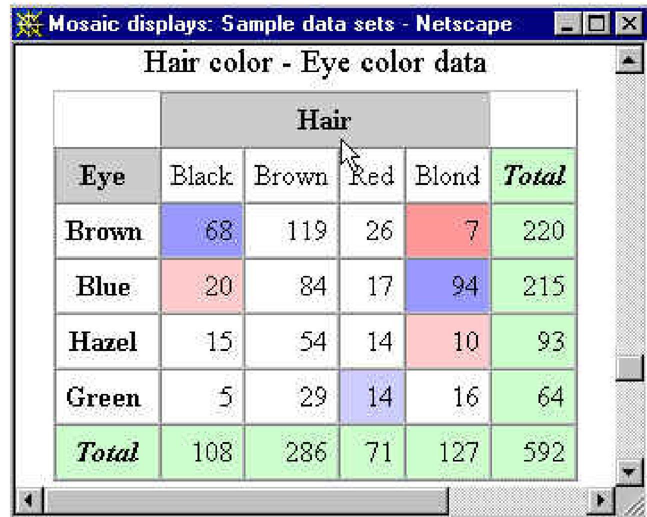
The module MD2HTML is used as shown below. It creates output to the file specified in the filename htmlout statement.

```
proc iml;
 %include 'md2html';
 filename htmlout 'mytable.html';
 *-- arguments;
 dim    = f 4 4g;
 vnames = f'Hair' 'Eye'g;
 lnames = f'Black' 'Brown' 'Red' 'Blond',
           'Brown' 'Blue' 'Hazel' 'Green'g;
 table = f68 119 26 7, 20 84 17 94, ... g;
 *-- globals;
 title= 'Mosaic displays: Sample data sets';
 run mdinit;                *-- for stand-alone doc;
 margins = 'rc+';
 run md2html(dim, table, vnames, lnames, tabid, caption);
 run mdfini;                *-- for stand-alone doc;
```

A variety of global variables control table options. For example, the the statement margins = 'rc+'; above adds row (r) and column (c) totals (+) to the table display. Other options control the style (background color or font) used for variable names, table margins and cell entries.

```
vstyle = 'bgcolor=#cccccc';  * variable names gray;
tstyle = 'bgcolor=#ccffcc';  * totals light green;
```

Figure 4 is an example of a table produced using the MD2HTML module. In this version, used to display sample datasets for the Mosaics CGI web applet (www.math.yorku.ca/SCS/Online/mosaics), each cell is colored according to the Pearson residual from a model of independence, giving a tabular analog of the mosaic display.


Mosaic displays: Sample data sets - Netscape

Hair color - Eye color data

	Hair				
Eye	Black	Brown	Red	Blond	Total
Brown	68	119	26	7	220
Blue	20	84	17	94	215
Hazel	15	54	14	10	93
Green	5	29	14	16	64
Total	108	286	71	127	592

Figure 4: Hair color, eye color data, as and HTML table. Cells colored according to residual from independence.

4.3 LaTeX output

Attractive as HTML tables may be, they are hard to include in publications. An equivalent module, MD2TEX produces *n*-way tables in LaTeX suitable for inclusion in a document. Most of the tables in *VCD* were generated automatically using MD2TEX and other tools described here.

Table 3: Survival on the Titanic. Table cells shaded according to residuals from model of mutual independence.

			Class			
Survive	Age	Sex	1st	2nd	3rd	Crew
Died	Adult	Male	118	154	387	670
		Female	4	13	89	3
	Child	Male	0	0	35	0
		Female	0	0	17	0
Survived	Adult	Male	57	14	75	192
		Female	140	80	76	20
	Child	Male	5	11	13	0
		Female	1	13	14	0

As with the HTML version, it is relatively easy to fit a loglinear model, and shade each cell according to its departure from independence, as shown in Table 3.

5 Getting data in and out of SAS/IML

Small or moderate sized contingency tables may be easily entered directly in SAS/IML, as illustrated in Section 1. For more general applications, however, it is useful to have ways to read multidimensional arrays in SAS/IML from a SAS dataset, and to produce an output dataset from such an array.

5.1 Data import

The module READTAB creates an n-way array from the dataset titanic shown earlier. The first three arguments are the name of the dataset, the name of the variable containing the table values, and a character vector containing the names of the factor variables. The corresponding table, levels, and vname arrays are returned.

```
vnames = fclass gender age survivedg;
run readtab('titanic','count', vnames, table, dim, lnames);
```

The order of the table variables is taken from the order of the observations in the dataset, rather than as listed in vnames. This is analogous to the use of the option order=data in PROC FREQ and other SAS procedures. The first variable is the one which varies most rapidly in the input dataset.

The READTAB module can also read two or more multidimensional arrays with the same structure. For example, we might have a dataset of crop yields, classified by Variety, Site, and Year, together with the residuals from some model. Multiple response variables are read when the second argument is a character vector of names:

```
resp = fyield residg;
vnames = fvariety year siteg;
run readtab('crops', resp, vnames, table, dim, lnames);
yield = table[,1];
resid = table[,2];
```

The separate responses are returned as columns of the table variable.

5.2 Data export

We can also define modules to export an n-way array to a SAS dataset, or to a text data file for processing with some other application. The module MD2DS, for example, creates a SAS dataset named by the OUT parameter. The datasetis exactly in the form shown for the Titanic data in Section 1.

```
start md2ds(dim, table, vnames, lnames, out);
 run md2var(dim, table, vnames, lnames, labels, count);
 vn = rowcat(vnames+' ');
 do i=1 to ncol(labels);
   call execute( vnames[i], '= labels[,i];');
   end;
 vars = ' var f' + vn + ' countg';
 call execute('create ', out, vars, '; append;');
 finish;
```

(The module MD2VAR ravels the frequency table into a column vector, COUNT, and creates a conforming matrix LABELS of the factor levels for each cell.)

In *VCD* this was used for analysis of frequency data using PROC GENMOD and PROC CATMOD, using data stored in SAS/IML modules. Similar routines were used to export n-way data to Lisp-form, for analysis with *Lisp-Stat* (Tierney, 1990) and *ViSta* (Young, 1994).

A Program Listings

The TRANSPOS module

```
start transpos(dim, table, vnames, lnames, order);
   if nrow(order) =1 then order = t(order);
   if type(order)='C' then do k=1 to nrow(order);
      ord = ord // loc(upcase(order[k,])= upcase(vnames));
      end;
   else ord = order;

   *-- Dont bother if order = 1 2 3 ... ;
   if all( row(ord)=1:ncol(row(ord)) ) then return;

   if nrow(dim   ) =1 then dim   = t(dim);
   if nrow(vnames)=1 then vnames= t(vnames);
   run marg(loc,newtab,dim,table,ord);
   table = row(newtab);
   dim    = dim[ord,];
   vnames = vnames[ord,];
   lnames = lnames[ord,];
   finish;

start row (x);
   *-- function to convert a matrix into a row vector;

   if (nrow(x) = 1) then return (x);
   if (ncol(x) = 1) then return (x`);
   n = nrow(x) * ncol(x);
   return (shape(x,1,n));
   finish;
```

The REDUCE module

```
start reduce(dim,table,vnames,lnames, config, op, expand);
   if nrow(dim   ) =1 then dim   = t(dim);
   if nrow(vnames)=1 then vnames= t(vnames);
   if ncol(config)=1 then config = t(config);

   *-- order factors to put 'config' dimensions first;
   others = remove(1:nrow(dim), config);
   ord = config || others ;
   nd = dim[ord];
   nc = nrow(config);
   no = nrow(others);
   nt = nrow(table) || ncol(table);

   tab = table;
   vn = vnames;
   ln = lnames;
   dm = dim;
   *-- make the dimension(s) to be reduced as columns;
   dotrans = any( ord ^= 1:nrow(dim) );
   if dotrans then do;
      run transpos(dm, tab, vn, ln, ord);
   end;
   tab = shape(tab, 0, (dim[config,])[#]);
   if no=0 then tab=t(tab);
   call execute( 't = tab[' + op + ',];');

   if expand then do;
      tab = repeat( t, nrow(tab), 1);
      if dotrans then do;
         *-- reorder by anti-ranks of original order vector;
         r = rank(row(ord));
         neword = r;
         neword[,r]=1:ncol(row(ord));
         run transpos(dm, tab, vn, ln, neword);
         end;
      tab = shape(tab, nt[1], nt[2]);
      end;
   else do;
      vn = vnames[config];
      ln = lnames[config,];
      dm = dim[config];
      tab = t;
      end;
   finish;
```

Further information

SAS/IML programs, associated macros, and sample datasets for mosaic displays and multidimensional arrays may be found on my web site, `www.math.yorku.ca/SCS/friendly.html`.

Michael Friendly
Psychology Department, York University
4700 Keele Street
Toronto, ON, Canada M3J 1P3
`friendly@yorku.ca`

References

Friendly, M. Mosaic displays for loglinear models. In *ASA, Proceedings of the Statistical Graphics Section*, pp. 61–68, Alexandria, VA, 1992a.

Friendly, M. User's guide for MOSAICS. Technical Report 206, York University, Psychology Dept, 1992b. `http://www.math.yorku.ca/SCS/mosaics.html`.

Friendly, M. Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, 89:190–200, 1994.

Friendly, M. Extending mosaic displays: Marginal, conditional, and partial views of categorical data. *Journal of Computational and Statistical Graphics*, 8:373–395, 1999a.

Friendly, M. *Visualizing Categorical Data*. SAS Institute, Cary, NC, 1999b. In press.

Friendly, M. Visualizing categorical data. In Sirken, M., Herrmann, D., Schechter, S., Schwarz, N., Tanur, J., and Tourangeau, R., editors, *Cognition and Survey Research*, chapter 20, pp. 319–348. John Wiley and Sons, New York, 1999c.

Snee, R. D. Graphical display of two-way contingency tables. *The American Statistician*, 28:9–12, 1974.

Tierney, L. *LISP-STAT: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*. John Wiley and Sons, New York, 1990.

Young, F. W. ViSta: The visual statistics system. Technical Report RM 94-1, L.L. Thurstone Psychometric Laboratory, UNC, 1994.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

® indicates USA registration. Other brand and product names are trademarks of their respective companies.