I. THE NEED

The first exposure to computers that many students get at North Carolina State University is in the intermediate Statistical Methods course. This course concentrates mainly on analysis of variance and it would seem obvious that SAS be chosen as the statistical package for that course. However, it was not SAS, but OMNITAB that was used for this course. The reason SAS was not used was that it was too easy and automatic. OMNITAB forced the students to become accustomed to the matrix formulae of least squares. SAS would have been only a black box, but OMNITAB was more like a calculator where the details of the computation had to be specified. Later, when students became practitioners, the SAS analysis procedures would be far more desirable; but for learning statistical methods, something like OMNITAB is needed.

Several writers have commented that statistical practitioners need to know two kinds of software to survive -- a statistical package for the routine work and a computational language such as OMNITAB or even FORTRAN with SSP to do the custom work. Several packages do combine these tasks to varying degrees. SJOUPAC has a matrix manipulation package; GENSTAT, PSTAT, BSP, and other statistical packages have a few general matrix operators. Computational languages are often used in place of statistical packages: OMNITAB with its statistical commands, FORTRAN and PL/1 with SSP and IMSL, APL and SPEAKEASY with their library routines.

SAS now has a MATRIX procedure. Several statistical methods courses at North Carolina State University now use it. It greatly expands the computing horizon of the SAS user: Now virtually any statistical method or numerical manipulation is available in SAS if the user is willing to specify the operations in enough detail.
II. DESIGN CONSIDERATIONS

The general goals of computer software remain power and generality, accuracy, efficiency, and ease of use. Several more specific characteristics were deemed important for the MATRIX language and its implementation.

1. Syntax: The language should be algebraic. Users should be able to write instructions in terms of algebraic formulae, not a series of command statements. The language should correspond as much as possible to conventional matrix notation. It should also be similar to the major computational languages, such as FORTRAN and PL/1, and should resemble SAS transformation statements. The language would then be easy and natural for most users.

2. Vocabulary: The operator set should be useful and manageable. The operations that are most frequently used should be instantly available. Less common operations should not be included in the language, especially if available through a short sequence of simpler operations. The operator set should be powerful yet concise. The language should be so small and consistent that experienced users should not need to use a reference card. The language should not require a nonstandard character set.

3. Allocation and Reference of Variables: Declaration, dimensioning, and allocation should be fully automatic.
   a) Referencing a matrix shall be by name or value only, never by number or location.
   b) No explicit declaration or dimensioning should be necessary.
   c) Allocation should be done as the need is encountered. Matrices could be reset later with different dimensions.
d) The size of the workspace will be flexible according to the region size of the job, i.e., there would be no fixed limits.

e) Autocompress feature to solve fragmentation problems.

f) LIST, free, and I/O to solve storage problems.

4. Data Format: There will only be one data type. The only data type will be the double-precision, real floating-point, row-major, 2-dimensional matrix. There will be no character variables, no higher dimensional structures.

With these design goals met, the MATRIX procedure should be powerful, yet very easy to learn and use.

III. LESSON ONE

Suppose that a 2 x 2 matrix called A is to be created and set to a value. We write:

\[
A = 1 2 / \\
3 4 ;
\]

This simple assignment demonstrates that all one must do to specify a matrix by its values is write out the values by row and separate each row by a slash. Let the matrix B also be given values:

\[
B = 2 1 / 1 2 ;
\]

Now that these matrix names have been set to values, they can be referred to in subsequent expressions.

To add two matrices together to form a matrix sum called X, one writes the assignment:

\[
X = A + B;
\]
Note that the matrices are conformable. If they were not, an error message would result.

To form the difference, write

\[ \text{DIFF} = A - B; \]

The matrix product is denoted by *:

\[ X = AB; \]

This is matrix multiplication, not element-by-element multiplication. The element-wise product is written with a different infix operator:

\[ X = A\#B; \]

The matrix transpose, the operator that switches rows for columns, is specified by the traditional prime or apostrophe.

\[ T = A'; \]

Operations can always be combined into complex expressions.

\[ X = AB + C \]

means multiply A by B, then add C to that product and call the result X.

An operator precedence table defines which operators are performed first. The order of operation can always be directed by using parentheses around expressions that are to be evaluated first.

<table>
<thead>
<tr>
<th>Operator Precedence Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ - ' (,<em>,</em>)             (highest)</td>
</tr>
<tr>
<td># # # ** &lt; &gt; &gt; &lt; #/ @ @</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>+ -</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;= = ↑ =</td>
</tr>
<tr>
<td>&amp;</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
There are almost one hundred operations. Most of these operations are functions. Functions are written with the name of the function followed by a parenthesized list of arguments, each argument separated by commas. The matrix inverse of $A$ is found by:

$$Z = \text{INV}(A);$$

With this function we can write a one-line solution to the least squares problem with the familiar looking formula:

$$\beta = \text{INV}(X'X) \times (X'Y);$$

The MATRIX procedure is particularly adept at non-arithmetic manipulations such as subsetting, combining and inserting. A subscript operator is useful for forming submatrices from specified rows and columns of a matrix. Horizontal and partial concatenation operators are useful in combining several matrices into one large matrix.

A series of command statements is also available. These commands are used mainly to control the flow of execution. The GOTO and LINK statements behave exactly as they do in regular SAS program statements. There is also a simple DO group facility modeled after the FL/I statement (but not iterative yet). The IF statement has been expanded to allow nested IF's, DO clauses, and ELSE statements.

For a more detailed description of the procedure, please consult the SAS User's Guide.

IV. THE FUTURE

The MATRIX procedure is still fairly limited in its capabilities. Some limitations keep the procedure simple and easy to use. However, there are a number of limitations serious enough to greatly frustrate the user. The problems and absent features include:
1. The ability to recognize missing values.
2. The saving and retrieving of whole workspaces full of matrices.
3. Iterative DO loops.
4. True interactive response.
5. Row and column names and other formatting features.
6. User defined functions and subroutines either in the matrix language or in FORTRAN on the library.
7. Virtual feature: shuttling matrices to disk when more memory is needed.
8. Higher dimensional matrices, complex and other data types.
9. Character, complex and other data types.

Of this list, I would look forward to providing the first three features soon and the middle three features eventually. The last three features are not planned at this time.

I think that the MATRIX procedure and other products like it have a great future. Perhaps some future version of SAS will be written in a dialect of EL/l that includes most of the matrix operators.

The matrix language implemented with this procedure can also be a valuable descriptive tool. It not only embraces the mathematical notation of matrix algebra, but also the control notation of computer languages. Users may consider writing statistical routines in the matrix language and then invoking them with the macro facility from a library. It may also be useful to publish source listings of statistical routines written in the matrix language; They would be a concise, yet complete description of the method as well as an immediately useful program.
V. EXAMPLES

These examples are pasted up from the printout of computer runs at TUCC. The printed output from the MATRIX procedure actually appears separately from the log of the source statements.

1. A complete list of the operators as of January 26, 1976.

2. The CORR and STANDARD procedures are matched with their matrix equivalents. Note the use of the J function to create a vector of 1's to multiply into a matrix to obtain row or column sums.

3. Solving a Nonlinear System.
   Note the use of the concatenation operator to compose matrices of functions.

4. Alpha Factor Analysis.
   This was transcribed directly from the journal article to the MATRIX statements in only a few minutes.

5. All possible regressions.
   See Technometrics 4 (1968), pp 769-779 (Schatzoff et al).
   This example works very much like the RSQUARE procedure in SAS. The sequential and reversibility properties of the sweep are used to the greatest advantage. All $2^P$ models are examined in only $2^P$ sweeps.

6. Categorical Example.
   The weighted least squares approach to categorical analysis involves only a few matrix multiplications.

7. Drazin Inverse.
   Both methods were transcribed from an algorithm in the notes for a book on generalized inverses.
LESS THAN

ELEMENTWISE MAXIMUM

LESS THAN OR EQUAL TO

SUBSCRIPT SELECTS ROWS AND COLUMNS

MATRIX ADDITION

OR

HORIZONTAL AUGMENTATION

AND

MATRIX MULTIPLICATION

MATRIX POWER

NOT

NOT EQUAL TO

MATRiX SUBTRACTION

PREFIX NEGATIVE

VERTICAL AUGMENTATION

GREATER THAN

ELEMENTWISE MINIMUM

GREATER THAN OR EQUAL TO

CREATE AN INDEX VECTOR

ELEMENTWISE MULTIPLICATION

DIVISION

ELEMENTWISE POWER

DIRECT (KRONECKER) AND SCALAR PRODUCT

HORIZONTAL DIRECT PRODUCT

TRANSPOSE

EQUAL TO

ELEMENTWISE ABSOLUTE VALUE

TEST IF ALL ELEMENTS NONZERO

ANY TEST IF ANY ELEMENTS NONZERO

BLOCK

BLOCK DIAGONAL MATRIX

DESIGN

MAKE DESIGN MATRIX FROM VECTOR OF LEVELS

DET

DETERMINANT

DIAG

OBTAIN A DIAGONAL MATRIX

DO

DO GROUP (LIKE PLI)

ECHelon

ROW ECHelon NORMAL FORM

EiGEN

FIND EIGENVALUES AND EIGENVECTORS

EiGval

FIND ONLY EIGENVALUES

EiGvec

FIND ONLY EIGENVECTORS

ELSE

ELSE CLAUSE (LIKE PLI)

END

END OF DO GROUP (LIKE PLI)

EXP

ELEMENTWISE EXPONENTIAL

FETCh

INPUT A MATRIX FROM SAS DATASET

FRee

DEALLOCATE A MATRIX

FUZZ

FUZZ TO AN INTEGER

GIiNV

MOORE-PENROSE G - INVERSE

GOTO

GOTO THE STATEMENT LABELED

HALF

CHOLESKl DECOMPOSITION

IF

IDENTITY OF A GIVEN SIZE

THEN

CONDITIONAL

INT

INTEGER VALUE

INv

MATRIX INVERSE

J

CREATE MATRIX OF IDENTICAL VALUES

LINCk

LINK TO A SUBROUTINE

LiST

LIST ATTRIBUTES OF ALL DEFINED MATRICES

LOG

ELEMENTWISE LOG

MAX

MAXIMUM OF ALL ELEMENTS

MiN

MINIMUM OF ALL ELEMENTS

MOD

REMAINDERING

NCOL

NUMBER OF COLUMNS IN A MATRIX

NOTE

PRINT INFORMATION ON OUTPUT

NRow

NUMBER OF ROWS IN A MATRIX

OUTPUT

OUTPUT A MATRIX TO A SAS DATASET

PRiNT

PRINT THE MATRIX

RANk

CONVERT TO RANKS

RECIp

ELEMENTWISE RECIPROCAL

RETURN

RETURN FROM LINK, OR STOP

ROWMAX

MAXIMUM OF EACH ROW

ROWMIN

MINIMUM OF EACH ROW

ROWESSQ

SUM OF SQUARES ACROSS ROWS

ROwSUM

SUM ACROSS ROWS

SHAPE

REVERSE OF VEC

SOLVE

SOLVE A NONSINGULAR LINEAR SYSTEM

SOLVIT

SOLVE WITH ITERATIVE IMPROVEMENT

SORT

ELEMENTWISE SORT

SQR

SUM OF SQUARES OF ALL ELEMENTS

STOP

STOP AND RETURN TO SAS

SUM

SUM OF ALL ELEMENTS

SVD

SINGULAR VALUE DECOMPOSITION

SHEEP

Sweep certain pivots

TOEPLITZ

TOEPLITZ A VECTOR INTO A MATRIX

TRACE

SUM OF THE DIAGONAL ELEMENTS

VECC

STACK UP THE ROWS INTO A COLUMN VECTOR

VECDiAg

CONVERT THE DIAGONAL TO A VECTOR

XmLiT

EXTENDED PRECISION MULTIPLY
**Correlation Example:**

**Title:**

**Data:** Input X Y Z; Cards;

Note: Data set WORK.DATAR has 6 observations and 3 variables.

Note: The data statement used 0.07 seconds and 90K.

```sas
PROC MATRIX; FETCH X; NOBS=NROW(X); NVAR=NCOL(X);
SUM=J(1,NOBS)*X; XPX=X'*X; * Sums and Crossproducts;
XPX=XPX-SUM*SUM/NOBS; * Corrected for the mean;
S=diag(1/SUM); * Scaling matrix;
CORR=S*XPX*S; PRINT CORR;
N*VAR=NCOL(X); *
SUMS AND CROSSPRODUCTS; * CORRECTED FOR THE MEAN;
SCALING MATRIX; * CORRELATION MATRIX;
```

Note: The procedure MATRIX used 0.22 seconds and 200K and printed page 20.

**Proc Corr:**

Note: The procedure CORR used 0.18 seconds and 200K and printed page 21.

**Run:** * Standardizing Example:

```sas
PROC MATRIX; FETCH X; NOBS=NROW(X); NVAR=NCOL(X);
MEAN=J(1,NOBS)*X/NOBS; * Find the means;
X=X-J(NOBS,1).MEAN; * Center at zero;
SS=J(1,NOBS)*(X*X,; * Sums of squares;
STD=SORT(SS./(NOBS-1); * Standard deviations;
X=X*diag(1/STD); * Standardize;
PRINT X;
```

Note: The procedure MATRIX used 0.21 seconds and 200K and printed page 22.

**Proc Standard m=0 s=1:**

Note: Data set WORK.DATAR has 6 observations and 3 variables.

Note: The procedure STANDARD used 0.09 seconds and 200K.

**Proc Print:**

Note: The procedure PRINT used 0.12 seconds and 90K and printed page 23.

**Note:** SAS Project

**Institute of Statistics**

**N.C. State University**

**Raleigh, N.C. 27607**

<table>
<thead>
<tr>
<th>CORR</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>1</td>
<td>-0.717102</td>
<td>-0.436558</td>
</tr>
<tr>
<td>ROW2</td>
<td>-0.717102</td>
<td>1</td>
<td>-0.436558</td>
</tr>
<tr>
<td>ROW3</td>
<td>-0.436558</td>
<td>0.350823</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>-0.490116</td>
<td>-0.322749</td>
<td>0.226655</td>
</tr>
<tr>
<td>ROW2</td>
<td>-0.272287</td>
<td>-0.322749</td>
<td>-0.452911</td>
</tr>
<tr>
<td>ROW3</td>
<td>-0.163372</td>
<td>-0.322749</td>
<td>-0.452911</td>
</tr>
<tr>
<td>ROW4</td>
<td>-0.59903</td>
<td>1.61374</td>
<td>-0.452911</td>
</tr>
<tr>
<td>ROW5</td>
<td>2.01492</td>
<td>-1.29099</td>
<td>-0.792594</td>
</tr>
<tr>
<td>ROW6</td>
<td>-0.490116</td>
<td>0.645497</td>
<td>1.92487</td>
</tr>
</tbody>
</table>
PROC MATRIX;

*-------- SOLVE A NONLINEAR SYSTEM BY NEWTON'S METHOD --------*;

NOTE SOLVE: X1+X2-X1*X2+2=0, X1*EXP(-X2)-1=0;
X = .1 / -2;

L: X1=X(1.*); X2=X(2.*);

F= (X1+X2-X1*X2+2)*/
    (X1*EXP(-X2)-1);  

IF MAX(ABS(F))<.000001 THEN GOTO X; * CONVERGENCE? ;

J=[ (1-X2) | (1-X1) ]//
    [ EXP(-X2) | (-X1*EXP(-X2)) ]; * EVALUATE JACOBIAN;

DELTA=-SOLVE(J,F);

X=X+DELTA;

GOTO L;

* SOLVE FOR CORRECTION VECTOR;
* THE NEW APPROXIMATION;
* ITERATE;

X: PRINT X F;

NOTE: THE PROCEDURE MATRIX USED 0.28 SECONDS AND 200K AND PRINTED PAGE 5.

RUN;

SOLVE: X1+X2-X1*X2+2=0, X1*EXP(-X2)-1=0

X

COL1
ROW1 0.0977731
ROW2 2.32511

F

COL1
ROW1 5352266E-15
ROW2 6150146E-14

---
DATA PHYSB TYPE=corr OF=304;
* SEE PP. 124-125 OF HARMAN: MODERN FACTOR ANALYSIS, 2ND ED;
INPUT NAME $ 1-8 TYPE $ 73-80
data 9-16 var 17-24 var 25-32 var 33-40
var 41-48 var 49-56 var 57-64 var 65-72;
LABEL
VAR1=HEIGHT VAR2=ARM SPAN VAR3=LENGTH OF FOREARM VAR4=LENGTH OF LOWER LEG
VAR5=WEIGHT VAR6=BICEPSHANTERIC DIAMETER VAR7=CHEST Girth VAR8=CHEST WIDTH;
CARDS;
NOTE: DATA SET WORK PHYSB HAS 8 OBSERVATIONS AND 10 VARIABLES.
NOTE: THE DATA STATEMENT USED 0.11 SECONDS AND 96K.

PROC MATRIX;
FETCH R DATA=PHYSB; PRINT R; LINK ALPHA;
NOTE EIGENVALUES; PRINT H; NOTE COMMUNALITIES; PRINT H;
NOTE FACTOR PATTERN; PRINT F; STOP;
DATA FROM HARMAN'S EIGHT PHYSICAL VARIABLES;

*------------------------- ALPHA FACTOR ANALYSIS -------------------------*
REF: KAISER ET AL., 1965 PSYCHOMETRIKA, PP. 12-13
R CORRELATION MATRIX (N*S) ALREADY SET UP
P NUMBER OF VARIABLES
Q NUMBER OF FACTORS
H COMMUNALITIES
M EIGENVALUES
E EIGENVECTORS
F FACTOR PATTERN
(10*H2;HI;G;MM) TEMPORARY USE. FREE UP

*----------------------------------*---------------------------;
ALPHA=P*NCOL(R); Q=0;
H2=I(P)-DIA(RECI(P;VECDIAG(INV(R)))); * SMCS;
LOOP;
H=H2; HI=DIAG(SQT(RECI(P;VECDIAG(H)));)
G=HI*(I(R)-I(P))*HI+I(P); * GET EIGENVALUES AND VEC;
IF Q=0 THEN DO; Q=SUM(M>1); * NUMBER OF FACTORS;
I=1:Q; END;
MM=DIAG(SQT(HI(Q))); * INDEX VECTOR;
E=E*I:10; * COLLAPSE EIGVECS;
H2=H*DIAG(FROWSQ(E+MM)); * NEW COMMUNALITIES;
IF MAX(ABS(H-H2)>0.01 THEN GOTO LOOP; * CHECK CONVERGENCE;
H=SQRT(H); H=VECDIAG(HZ); * RESULTING PATTERN;
FREE EQ H2 HI G MM; * FREE TEMPORARIES;
RETURN;

NOTE: THE PROCEDURE MATRIX USED 0.72 SECONDS AND 200K AND PRINTED PAGE 8.

RUN;

FACTOR PATTERN

<table>
<thead>
<tr>
<th></th>
<th>COL1</th>
<th>COL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.81336</td>
<td>0.420147</td>
</tr>
<tr>
<td>2</td>
<td>-0.802636</td>
<td>0.49601</td>
</tr>
<tr>
<td>3</td>
<td>-0.8757909</td>
<td>0.494474</td>
</tr>
<tr>
<td>4</td>
<td>-0.8787446</td>
<td>0.432039</td>
</tr>
<tr>
<td>5</td>
<td>-0.805144</td>
<td>-0.48162</td>
</tr>
<tr>
<td>6</td>
<td>-0.600413</td>
<td>-0.419805</td>
</tr>
<tr>
<td>7</td>
<td>-0.620623</td>
<td>-0.44383</td>
</tr>
<tr>
<td>8</td>
<td>-0.644942</td>
<td>-0.28959</td>
</tr>
</tbody>
</table>
DATA: RETAIN; * GENERATE THE DATA:
LOOP: N=SUM(N+1); X1=NORMAL(0); X2=UNIFORM(0); X3=NORMAL(O)*5;
X4=NORMAL(0); X5=NORMAL(O)*5;
Y=X1+X2+X3+X4+X5+NORMAL(0);
OUTPUT;
IF N<50 THEN GOTO LOOP;
DROP N;

NOTE: DATA SET WORK\*DATA2 HAS 50 OBSERVATIONS AND 6 VARIABLES.
NOTE: THE DATA STATEMENT USED 0.15 SECONDS AND 96K.

PROC MATRIX:
* THIS DEMONSTRATES THE METHOD OF SCHATZOF ET AL. FOR COMPARING
* ALL POSSIBLE MODELS, IN THIS CASE COMPARING MSE;

* GET THE DATA:
FETCH XY; N=NROW(XY); P=NCOL(XY)-1; PL=P+1; IP=1:P;
X=XY(*,IP); Y=XY(*,PI);

* FORM THE CROSS PRODUCTS AND SET UP OTHER MATRICES:
A = (X'X) || (X'Y) || (Y'X) || (Y'Y);
IDF=J(P+1,0); * TO KEEP TRACK OF WHATS SWEPT;
MSEP=J(P+1,50); * RECORD BEST MSE PER # PARNs;
BESTP=J(P+1,0); * RECORD BEST SET PER # PARNs;
K=0;

* LOOP THRU BY METHOD OF SCHATZOF ET AL.:
LOOP: K=K+1; LINK ZTRAIL: IF NZ>P THEN GOTO DONE;
IF ABS(A(NZ,NZ))<1E-9 THEN; NOTE FAILURE; PRINT A; STOP; END;
A=SWEEP(A,NZ);
NPARM=SUM(IDF); ESS=A(P1,PI); MSE=ESS/(N-NPARM);
IF MSE<MSEP(NPARM,*); THEN DO;
MSEP(NPARM,*)=MSE; BESTP(*,NPARM)=IDF;
END;
GOTO LOOP;

ZTRAIL: Z=K; NZ=0; * FIND # TRAILING ZEROES;
ZL: NZ=NZ+1; LZ=Z; Z=INT(Z*.5); IF 2*Z=LZ THEN GOTO ZL;
RETURN;

DONE: NOTE MSEs FOR BEST K-PARAMETER MODELS; PRINT MSEP;
NOTE BEST K-PARAMETER MODELS; PRINT BESTP;
OMS=(1:PI)||MSEP; OUTPUT OMS OUT=BESTMS;

NOTE: DATA SET WORK\*BESTMS HAS 5 OBSERVATIONS AND 3 VARIABLES.
NOTE: THE PROCEDURE MATRIX USED 0.64 SECONDS AND 200K AND PRINTED PAGE 15.

MSEs FOR BEST K-PARAMETER MODELS

<table>
<thead>
<tr>
<th>MSE</th>
<th>COL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>2.99327</td>
</tr>
<tr>
<td>ROW2</td>
<td>1.94036</td>
</tr>
<tr>
<td>ROW3</td>
<td>1.38041</td>
</tr>
<tr>
<td>ROW4</td>
<td>1.04742</td>
</tr>
<tr>
<td>ROW5</td>
<td>0.650595</td>
</tr>
</tbody>
</table>

BEST K-PARAMETER MODELS

<table>
<thead>
<tr>
<th>BESTP</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
<th>COL5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ROW2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ROW3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ROW4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ROW5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
PROC MATRIX;

*-- EXAMPLE FROM GRIZZLE, STAHRER, KOCH BIOMETRICS 25,P493 ----;

FREQ= 1520 266 124 66
  234 1512 432 78
  117 362 1772 265
  36 82 177 492 ;

A= 0 1 1 -1 0 0 -1 0 0 0 -1 0 0 0 /
C -1 0 0 1 0 1 1 0 -1 0 0 0 -1 0 /
0 0 -1 0 0 0 -1 0 1 1 0 0 -1 0 ; * CONTRAST MATRIX

P=PECIP(SUM(FREQ));   * TOTAL FREQUENCY;
P=RNC FREQ * FN;      * ESTIMATE THE PROBABILITIES;

V=RN*D(T(DIAG(P)-P^P));  * VARIANCE OF P;
S=A*V*A^A;      * VARIANCE OF A^P;

CHISO=P*A^A*INV(S)*A^P;     * QUADRATIC IN A^P DISTRIBUTION; CHI-Square;

NOTE: CHI-SQUARE FOR THE CATEGORICAL PROBLEM PRINT CHISO;


RUN;

CHI-SQUARE FOR THE CATEGORICAL PROBLEM

CHISO          COLI
ROW1          11.9757
PROC MATRIX;
A= 10. -8 6. -3/
 12 -10 8. -4/
 1. -1 1. 0/
 -2 -2 -2 2;
LINK DRAZIN;
RETURN;

* THIS ROUTINE COMPUTES THE DRAZIN INVERSE:
DRAZIN: N=NROW(A); LASTS=I(N); J=0;
SLOOP: J=J+1; AS=A*LASTS;
  B=RECIP(J)*TRACE(AS);
  S=AS+B*I(N);
  IF MAX(ABS(B))>E-6 THEN DO: U=J; BU=B; SU=LASTS; END;
  IF MAX(ABS(S))>E-6 THEN DO: LASTS=S; GOTO SLOOP; END;
L=N-U;
SU=SU**((L+1)/)
DINV=-RECIP(BU**(L+1))* (A**L) * SU;
NOTE DRAZIN INVERSE; PRINT DINV;
RETURN;

THE PROCEDURE MATRIX USED 0.30 SECONDS AND 200K AND PRINTED PAGE 7.

RUN;

(algorithms from C. Meyer et al., book in preparation)
### ORIGINAL MATRIX AND ITS DRAZIN INVERSE

<table>
<thead>
<tr>
<th>Row</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>10</td>
<td>-8</td>
<td>6</td>
<td>-3</td>
</tr>
<tr>
<td>ROW2</td>
<td>12</td>
<td>-10</td>
<td>8</td>
<td>-4</td>
</tr>
<tr>
<td>ROW3</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ROW4</td>
<td>-2</td>
<td>2</td>
<td>-2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>2</td>
<td>-1.5</td>
<td>1</td>
<td>-0.5</td>
</tr>
<tr>
<td>ROW2</td>
<td>2</td>
<td>-1.5</td>
<td>1</td>
<td>-0.5</td>
</tr>
<tr>
<td>ROW3</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>ROW4</td>
<td>-2</td>
<td>2</td>
<td>-2</td>
<td>2</td>
</tr>
</tbody>
</table>

### DRAZIN INVERSE

<table>
<thead>
<tr>
<th>Row</th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW1</td>
<td>2</td>
<td>-1.5</td>
<td>1</td>
<td>-0.5</td>
</tr>
<tr>
<td>ROW2</td>
<td>2</td>
<td>-1.5</td>
<td>1</td>
<td>-0.5</td>
</tr>
<tr>
<td>ROW3</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>ROW4</td>
<td>-2</td>
<td>2</td>
<td>-2</td>
<td>2</td>
</tr>
</tbody>
</table>