The Experimental Interactive Matrix Language
John Sall, SAS Institute Inc.

Note: This is a draft document for experimental software. This document makes no promises that the software will be released as documented here, nor does it promise any release date. As of this date, final design decisions have not been made.

The MAT procedure is a reincarnation of PROC MATRIX that is interactive. Some conventions are being changed, and many new features are being added. The new language aims to:

• compete better with APL and other interactive languages,
• be useful as a report writer,
• be useful for interactive graphics, and
• be a better implementation language for packaging statistical methods (combined with the new macro facility).

The new procedure is written in reentrant assembler language in conventions similar to those of the SAS supervisor. It will be rewritten into PL/I when portability becomes important. It is compact (less than 50K) and efficient relative to the old MATRIX procedure.

Differences Between New MAT and Old MATRIX

1. New Name
   The procedure is temporarily called MAT. We are open for suggestions for new names. A new name is necessary because MAT language is different enough to be incompatible with old MATRIX code, and MATRIX must coexist along with MAT in the library.

2. Literals
   MAT literals with more than one element must be enclosed with double-less-than double-greater-than brackets. For example:
   
   $$A=\langle\begin{array}{cc}1 & 2 \\ 3 & 4 \end{array}\rangle,$$
   
   Rows are separated by commas rather than slashes. These changes were made to correct several problems: The sign change and subtract operators can no longer be confused with signs in numeric literals (for example: Y=X**2-1;). The slash can now be used for division.

3. Character Matrices and Literals
   A matrix can now be character valued. Each element can be a string. The string can be 1-256 bytes long. Each element of a matrix is the same size. Character literals are given in double quotes, i.e., 
   
   $$A="coffee";$$
   
   Literals with several elements are enclosed by $$<>$$ like numeric matrix literals.
   
   These conventions cure the problems in old MATRIX: Single quote was used for both transpose and literals. Character valued matrices were fixed in size. They were not recognised by most operations as character-valued, i.e., the PRINT command did not know unless a character format was used.

4. Subscripts are specified differently
   They are written using $ as a postfix operator with one or two arguments:
   
   $$A=8\left\{row,column\right\};$$
   
   $$A\left\{rows,column\right\}=B;$$
   
   Either argument can be left empty to signify all rows or columns. R.H.S. subscripts can be reduction operators.
   
   The change is needed to eliminate the old ambiguity between subscripted matrices and functions with arguments. The old "rule of first use" is unworkable not that resolution is delayed until after parsing.

5. Single subscripts are allowed
   They refer to the matrix element in row-major order. For vectors with only one row or column, this accesses the expected element. For example
   
   $$A=8<1,2,3,4,5,6>;$$
   
   $$B=AS(3);$$
   
   $$C=AS(<2,3,1>);$$
   
   $$A\left\{5\right\}=10;$$
   
   is allowed

6. Certain special character operations have changed
   Division is now / rather than #/. The rarely used Horizontal Direct product operator @I has been implemented as the HDIR function now.
7. Certain commands have been dropped
The EIGEN, SVD, and GS commands have been converted into calls, for example:
CALL EIGEN(M,E,A);

8. Error diagnostics refer to the source differently
When finished, MAT will number each statement entered. The error diagnostics will (when finished) print the source line with the error diagnosed. The old line:column from the original text becomes unworkable when you work with lines that have been edited interactively inside PROC MAT.

9. Execution CPU overhead is considerably reduced compared with old MATRIX
The new interpreter is all assembler language, whereas the old one contained a lot of PL/I code. The actual worktime, however, should be the same. This means that programs with large numbers of small operations will run faster, but small programs with large operations will not.

Specifications of PROC MAT
To invoke, say:
PROC MAT; RUN;

MAT responds with a "MAT READY" message and begins recognizing statements as they are entered. There are two types of statements, those immediately executed, and those compiled to be executed later. The immediate statements are:

MOPTIONS to set various options
SHOW to show various information
LOAD to load a library module
QUIT to exit from MAT
ABEND to exit with OC1
RUN to run the compiled statements
NORUN to step over statements without executing them.

Other immediate statements relating to error recovery, and editing are planned.

The compiled statements include:

- assignment, subscripted assignment
- IF, ELSE, DO, END
- GOTO, LINK, STOP, RETURN
- PRINT, LIST, NOTE
- FREE, FETCH, OUTPUT

The control statements (IF, ELSE, DO, END, GOTO, LINK, STOP, RETURN) do not work in IMMED mode.

Primitive Graphics Routines for MAT
This section describes a preliminary set of primitive graphics routines which are accessible through the new interactive matrix procedure MAT. Other more sophisticated routines will be written later as the sublib for GWHIZ is developed. Primitive routines give the most flexibility to the programmer, but the programmer must specify every plotting item down the last detail.

It would be easy to add many more graphics operations. Please forward your comments concerning proposed enhancements and additions to SAS Institute.

Introduction
In order to use graphics, you must first load the MLIBGRL module with the command:

LOAD MLIBGRL;

In order to see the entry points that this module has, you can command: SHOW LOAD; to obtain a listing of the entry names and their attributes.

All the graphics routines are implemented as calls, since they return no values. The first graphics call must be to GSTART, since this module initializes graphics.

CALL GSTART;
CALL GSTART(<<0 100 0 100>>);

The only argument allowed is a vector of numbers supplying the end coordinates of the X and Y axes. The default is for the X axis to go from 0 to 100, and the Y axis to go from 0 to 100, as is specified explicitly above. Every other command refers to points in the coordinate system established by GSTART.
There are various graphics commands to generate lines, points, text, etc. However nothing is actually drawn until the GSHOW routine is called. After the graph is shown, it can be cleared by calling GCLEAR—if GCLEAR is not called, the graphs from previous drawing will keep producing on new pictures.

Example 1: Rose Curve  The equation for a rose curve is \( r = a \cos(n \theta) \) in polar form, where \( a \) is the diameter, \( n \) determines petals, and \( \theta \) goes from 0 to \( 2\pi \).

PROC MAT; RUN;
LOAD MLIBGRL;
CALL GSTART;
A=(0:360)1/2113 .1416/360;
R=8*sin(A);
R=R|R²1|R³3;
A=A|A|A;
X=R*sin(A)+50;
Y=R*cos(A)+50;
CALL GDRAW(X,Y);
CALL GSHOW;
CALL GCLEAR;
RUN;
Example 2: Pie Slice  To make the pie we started with counts, converted them to portions, cumulated them, then called the pie routine.

```plaintext
CALL GSTART(<-50 150 -50 150>);
C=<16 20 50 40 15>; N=NOCOL(C);
P=C/SUM(C);
p=P<1 1 1 1 1,
      0 1 1 1 1,
      0 0 1 1 1,
      0 0 0 0 1>;
C1=IPLS(1: (N-1))#360;
C2=P#360;
CALL GPIE(50,50,15,C1,C2);
CALL GSHOW; CALL GCLEAR;
RUN;
```

Example 3: U.S. Polygon Fill  A course set of U.S. coordinates is fed into the polygon fill routine.

```plaintext
CALL GSTART(<-50 150 -50 150>);
X=1 2 5 17 27 29 31 33 36 39 39 47 53 54 58
   62 64 65 69 66 68 74 75 78 82 79 79 77
   74 71 69 62 62 57 53 48 50 43 14 5 3 3
   35 29 21 13 8 6 8 6 1
   0 2 8 6 8 9 8 7 4
   0 4 12 16 21 32 35 38 42
   44 47 43 41 39 32 36 43
   43 41 43 45 46 51 49 46 40 55> + 30;
CALL GPOLY(X,Y);
CALL GSCRIPT(5.20,"MAP OF U.S.","COMPLEX",4);
CALL GSHOW; CALL GCLEAR;
RUN;
QUIT;
```
Example 4: 3-D Rotations  The starting and ending coordinates to a house figure are put into X1 and X2 respectively. The matrices three columns relate to the x, y, and z axes on the plot. Three rotation matrices are constructed and used to rotate the house's coordinates. These rotated coordinates are then translated to the nine places in the field.

PROC MAT; RUN;
LOAD MLIBGRL;
X1=<< 2 1 -1, -2 1 -1, 2 -1 -1, 2 -1 -1, /# BASE */
  2 1 1, -2 1 1, -2 -1 1, -2 -1 1, /# TOP FACE */
  2 1 1, -2 1 1, -2 -1 1, -2 -1 1, /# DOWN EDGES */
  2 1 1, -2 1 1, -2 -1 1, -2 -1 1, /# ROOF */
  2 0 2>>;
X2=<< 2 1 -1, -2 1 -1, 2 -1 -1, 2 1 1,
  2 1 1, -2 1 1, -2 -1 1, -2 -1 1,
  2 0 2, -2 0 2, -2 0 2, 2 0 2,
  2 0 2>>;
CALL GSTART(<<-14 14 -10 10>>);
ROLL=<< .94 .34 .00 ,
  - .34 .94 .00 ,
  .00 .00 1.00 >>;
YAW=<< .94 .00 .34 ,
  .00 1.00 .00 ,
  - .34 .00 .94 >>;
PITCH=<< 1.00 .00 .00 ,
  .00 .94 .34 ,
  .00 .34 1.00 >>;
N=NROW(X1);
H=J(N,1,6)||J(N,2,0);
V=J(N,1,0)||J(N,2,5);
MACRO _GD CALL GDRAW(X1, X2, %); %
XX1=X1-H+V; XX2=X2-H+V; _GD
X1=X1*ROLL; X2=X2*ROLL; XX1=X1+V; XX2=X2 +V; _GD
X1=X1*YAW; X2=X2*YAW; XX1=X1+H; XX2=X2 +H; _GD
X1=X1*PITCH; X2=X2*PITCH; XX1=X1+H-V; XX2=X2+H-V; _GD
X1=X1*YAW; X2=X2*YAW; XX1=X1 ; XX2=X2 ; _GD
X1=X1*ROLL; X2=X2*ROLL; XX1=X1-H ; XX2=X2-H ; _GD
X1=X1*PITCH; X2=X2*PITCH; XX1=X1-V; XX2=X2 -V; _GD
X1=X1*PITCH; X2=X2*PITCH; XX1=X1+H-V; XX2=X2+H-V; _GD
CALL GSCRIPT(-9,8.5."3 Dimensional", "TRIPEX",<<3.5 0 0 3>>);
CALL GSCRIPT(-3,8.0,"ROTATIONS", "SCRIPT",<<6 0 0 2>>);
CALL GSCRIPT(4,8.5,"Using MAT", "TITALIC",<<3 0 0 3>>);
CALL GSHOW;
CALL GCLEAR;
CALL GSTOP;
3 Dimensional \textit{ROTATIONS} Using \textit{MAT}