Abstract

Despite the fact that "SAS" does not stand for you-know-what, it is true that between SAS/STAT®, SAS/ETS®, SAS/OR® and SAS/QC®, the SAS System does provide an extensive variety of statistical procedures. However, there still are occasions requiring a statistical analysis for which a SAS Proc does not exist. At that point, IML, the Interactive Matrix Language, can be invaluable. This paper will give an overview of IML drawing on examples from bootstrapping, non-parametric density estimation and regression, Bayesian estimation imposing inequality restrictions, and graphics. In the discussion that follows, all IML statements and functions will appear in bold print.

Overview of IML

IML is a programming language in which all variables are matrices, either numeric or character. As a programming language it has the usual features for looping, e.g., five flavors of DO loops, and conditional execution with IF-THEN/ELSE.

In addition, IML has over 100 of its own functions along with a similar number of Base SAS functions. A couple of IML functions useful for statistical algorithms which appear in the examples below are: root - Cholesky factor, eigval - eigenvalues for symmetric matrix.

IML also has a number of general purpose commands to control the workspace during a session and manage storage catalogs where matrices and modules are stored permanently. Finally, there are a set of commands for reading, writing and querying SAS data sets as well as accessing external files.

IML supports the concepts of the subroutine and function with the IML module. An IML module is defined as follows:

```
START module_name (optional argument list);
   . body of module
FINISH;
```

Modules are executed with either the RUN or CALL statements. The difference is the priority followed in resolving the module. If a user's module shares a name with a built-in IML function, then RUN will execute the user's module and CALL the IML function. IML statements executed outside a module are said to be executed in immediate mode. All variables defined in immediate mode are in the global symbol table. A module with no arguments has as its symbol table the global table. Variables defined in a module with arguments are in the module's local symbol table. A GLOBAL option is available on the START command to declare specific variables in a module to be treated as global. An IML module can behave like a function and return a single result by including the statement RETURN(variable_name) in the body of the module. Of course, that single result returned can contain a variety of information as an mxn matrix. Compiled modules, as well as matrices, can be saved permanently in IML storage catalogs. For example,

```
LIBNAME mylib '\sugi17';
RESET STORAGE = 'mylib.plot';
STORE MODULE = (plot text abline);
```

To retrieve the compiled modules:

```
RESET STORAGE = 'mylib.plot';
LOAD MODULE = _all_;
```

There are two shortcomings in the way modules are currently implemented in IML. First, all arguments are positional. Thus, keyword arguments, as in the macro language, are not allowed. If a module is defined with n arguments, then n arguments must be used each time the module is called. This makes it more difficult to assign default values for arguments in user-defined modules. Second, only matrices are allowed as arguments, not modules. More will be said later on these two points and specific examples will show how to circumvent some of the problems.
IML supports dynamic program execution through commands, PUSH, QUEUE, and EXECUTE. Only execute will be discussed here. Execute builds a SAS statement from its arguments, each of which is either a character string or a character variable. Up to 15 arguments can be used, but this is not a limitation because the concat function can construct a single string from several arguments. For example, the statements

```
   msg = 'This is x';
   CALL EXECUTE("print ", msg, ", \"x\")
```

will build the statement: print 'This is x'; and immediately execute it. This feature of IML is similar to the macro language building SAS code under program control and can be very powerful. An example will be given later where use of execute is necessary.

The rest of this paper will demonstrate applications of IML through specific examples in bootstrapping, density estimation, Bayesian estimation with inequality restrictions and graphics.

**Bootstrapping**

The bootstrap is a method for estimating the sampling variances and confidence intervals of statistics in which the observed data plays the role of the unknown underlying population. It is nonparametric in the sense that untested assumptions about the underlying population need not be made, and it is flexible in that the technique can be used for a wide variety of statistics. It is also computationally intensive.

Given a sample \( S = \{X_1, \ldots, X_n\} \), where \( X_i \) has distribution function \( F \), let \( \theta \) be a parameter of interest and \( \hat{\theta}(X) \) an estimator of \( \theta \) using the observed sample. For example, \( \theta = E(X) = \mu \) and \( \hat{\theta}(X) = \bar{x} \), the sample mean. A measure of the accuracy of \( \hat{\theta} \) is its standard error, \( \sigma(F) = \sqrt{\text{VAR}_E(\hat{\theta}(X))} \). In some cases there are formulas for \( \sigma(F) \), eg. \( \text{var(\bar{x}) = \sigma^2/n} \), but in most cases it is necessary to either make some assumptions about the underlying distribution, eg. normality, or rely on asymptotic results from maximum likelihood theory. The bootstrap provides a way to estimate \( \sigma(F) \) without making any distributional assumptions.

Letting \( F_n(x) = \) the empirical distribution function of the sample, \( F_n(x) = (\# x_i \leq x)/n \), then \( F_n(x) \) is in some sense an optimum estimator of the unknown \( F(x) \). If we could repeatedly sample from the underlying population \( \sim F \), then we could directly estimate \( \sigma(F) \). Since this is not generally feasible, the bootstrap method samples from the bootstrap distribution \( \sim F_n \), i.e., we sample with replacement from the observed data.

The bootstrap algorithm is as follows: given the sample \( S = \{X_1, \ldots, X_n\} \), \( B \) = the number of bootstrap samples drawn, and \( \hat{\theta} \) = the statistic of interest,

1. for \( b = 1 \) to \( B \)
   - draw a bootstrap sample of size \( n \), \( X^{(b)} \), by sampling with replacement from \( S \).
   - calculate the statistic of interest \( \theta^{(b)} \) from \( X^{(b)} \).
2. next \( b \)

estimate \( \sigma = \) standard error of \( \hat{\theta} \) with

\[
\sigma_s = \sqrt{\frac{1}{B} \sum_{b=1}^{B} (\theta^{(b)} - \text{avg}(\theta^{(b)}))^2}
\]

Suppose the statistic of interest is the sample correlation coefficient for two variables \( X \) and \( Y \) given the sample \( \{x_1, \ldots, x_n\} \) and \( \{y_1, \ldots, y_n\} \). Assume the data for \( X \) and \( Y \) have been read into the IML variables \( x \) and \( y \). Then the following segment of IML code would do the necessary calculations. We first define a function module to calculate the statistic.

```
start correl(x,y);
   rho = ((x - x[:])#(y - y[:]))/(sqr((ssq(x - x[:])#ssq(y - y[:]))));
   return(rho);
finish;
```

```
start bootrho(x,y,B);
   * n = sample size;
   n = nrow(x);
   * initialize vector for bootstrap values of statistic;
   rhostar = J(B,1,0);
   do i = 1 to B;
      * get indices for a bootstrap sample;
      smpl = floor(n*uniform(J(n,1,0)) + 1);
      * draw bootstrap sample;
      xboot = x[smpl];
      yboot = y[smpl];
      * calculate statistic with bootstrap sample;
      rhostar[i] = correl(xboot,yboot);
   next i;
   * compute standard error of estimate with
   * formula above;
   rho = correl(x,y);
   return(rho);
finish;
```
The basic percentile method has undergone several enhancements, each of which can be easily programmed in IML. For example, the bias-corrected percentile method, or BC method, corrects for possible bias in the percentile method under the assumption that a transformation to normality exists for the estimator \( \hat{\theta} \). Let \( m = \# \text{ of } \hat{\theta}^{(m)} \leq \hat{\theta} \), i.e., the number of bootstrap estimates less than the original estimate. In the "bootstrap world" (where the original sample becomes the population from which the bootstrap samples are drawn), \( \hat{\theta} \) is the "true value" of the parameter for which the \( \hat{\theta}^{(m)} \) are estimates. Let \( z_0 = \Phi^{-1}(m/B) \), where \( \Phi \) is the standard normal cdf and \( B = \text{total # of bootstrap samples drawn} \). Then the BC method produces a lower end point for a 100(1 - 2\( \alpha \))% confidence interval by using as the lower percentile point the value

\[
L_\alpha = \Phi(2z_0 + \Phi^{-1}(\alpha)).
\]

This quantity is calculated in IML as:

\[
z_0 = \text{probit}(m/B); \\
\text{lpctile} = \text{probnorm}(2*z_0 + \text{probit}(\alpha));
\]

Note that when \( m/B = \frac{1}{2} \), then \( z_0 = 0 \) and \( \text{lpctile} = \alpha \); the same result as with the percentile method. The paper by Efron and Tibshirani [1986] discusses various bootstrap confidence interval techniques.

### Density Estimation

Given a sample \( S = \{X_1, ..., X_n\} \) from a population with unknown distribution \( F(X) \), the problem of nonparametric density estimation is to estimate the density \( f(x) = dF(x)/dx \) without making parametric assumptions about \( F \), e.g., that \( X \sim N(\mu, \sigma^2) \). Several density estimation techniques are available. The kernel method will be discussed here.

A function \( K(u) \) is called a kernel function if it satisfies the following conditions:

\[
K(u) \geq 0, \\
\int K(u) \, du = 1, \\
\text{and } K(u) = K(-u),
\]

i.e., \( K(u) \) is a symmetric density function. The role of \( K(u) \) is that of a weight function. To estimate \( f(x) \), the estimated value of \( \hat{f}(x) \) should depend on the proximity of \( x \) to the points in the observed sample, \( \{x_1, ..., x_n\} \). The closer \( x \) is to points in the sample, the greater should be
There are about a half dozen popular kernel functions, although it can be shown that it doesn't really matter which is used. The kernel function shown below is Tukey's Biweight kernel, also known as the quartic kernel. It is defined as \( K(u) = \frac{15}{16}(1-u^2)^2 \), for \(-1 \leq u \leq 1\).

A more critical choice than the kernel function is the parameter \( h \), known as the bandwidth. It controls the spread of the weight associated with the kernel function.

Define the function \( K_h(u) = \frac{K(u/h)}{h} \). Note that the support of \( K_h \) is \([-h, h]\) and \( \int K_h(u)du = 1 \).

The kernel density estimator is defined as:

\[
\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x-x_i)
\]

\[
= \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)
\]

The size of \( h \) determines the degree of 'smoothness' of the estimator. The larger the value of \( h \), the smoother the density estimator. The two extremes of \( h = 0 \) and \( h = \infty \) correspond to the estimates of interpolating the data and the sample mean, \( \bar{x} \), respectively.

A common criterion for selecting \( h \) is to minimize the mean square error of \( \hat{f}_h(x) \) as an estimate of \( f(x) \):

\[
\text{MSE}(\hat{f}_h) = \int (f(x) - \hat{f}_h(x))^2 \, dx = \text{MISE}(\hat{f}_h).
\]

It can be shown that

\[
\text{MISE}(\hat{f}_h) = \frac{1}{(nh)} \|K\|_2^2 + h^4/4 \|\mu_2(K)\|^2 \|f''\|_2^2 + o(1/nh) + o(h^4), \quad n \to \infty, \quad h \to 0
\]

where \( \|g\|_2 = L_2\)-norm of \( g = (\int g(x)^2 \, dx) \) and \( \mu_2(K) = \text{second central moment of } K \). The first term, \( 1/(nh) \|K\|_2^2 \), is associated with variance. So minimizing variance leads to \( h \to \infty \). The term \( h^4/4 \|\mu_2(K)\|^2 \) comes from the bias and calls for \( h \to 0 \) to minimize bias. Thus, the trade-off between bias and variance is reflected in a compromise choice for the optimal \( h \). The asymptotic MISE, A-MISE, drops the higher order terms listed above and leads to the optimal bandwidth, \( h_0 \):

\[
h_0 = \left( \frac{1}{n} \right)^{1/2} \left( \frac{1}{\text{MISE}(\hat{f}_h)} \right)^{1/2}
\]

The only fly in the ointment is that solving for \( h_0 \) requires knowledge of the unknown density function \( f \) in the term \( \|f''\|_2^2 \).

Among data-based techniques to estimate \( h_0 \) are several based on cross-validation. The text by Härdle [1989] contains algorithms in pseudo-code and S for some of these techniques. These algorithms are easily converted to IML.

To write an IML function, kde, that returns a matrix of ordered pairs \((x_1, \hat{f}_h(x_1))\), where \( x_1, j = 1, \ldots, n \), \( n \) grid points at which to plot the kernel density estimate, the required inputs would be:

\[
x = \text{vector of sample values}
\]

\[
h = \text{bandwidth}
\]

\[
kfun = \text{name of kernel function}
\]

\[
ngrid = \text{number of grid points for which the estimate is evaluated for later graphing.}
\]

A typical call of kde would be:

\[
\text{points} = \text{kde}(x,1.1,'triwgt',50);
\]

Suppose the collection of valid kernel functions were defined by the character matrix

\[
kernels = \{'uni','triangle','epanich','quartic','\triwgt','cosinus','gauss'\};
\]
then we could use the IML function for set intersection, xsect, to see if kfun is valid:

\[
\text{choice} = \text{xsect(upcase(kfun),upcase(kernels))};
\]

if nrow(choice) ^=1 then ..... invalid value.

The other point to make is how the parameter kfun is used to actually evaluate the previously defined kernel function whose name is the value of the character variable kfun. For example, suppose that kfun = 'triwgt' and fh[j] represents the value of the kernel density estimate at the jth grid point. Then the following statement appears in a loop within kde:

\[
fh[j] = fh[j] + \text{apply}(\text{kfun}, \text{dij});
\]

The IML function apply evaluates kfun(dij), i.e., the first argument of apply is the name of a previously existing module and the other arguments of apply are arguments of that module.

The graph below shows the result of applying kde to generated data consisting of a mixture of N(-1,1) and N(2,1) distributions, with mixture rates of .7 and .3 respectively. The specific call was:

\[
\text{denplot = kde(sample,0.5,'gauss',50)};
\]

The graph was produced by the module plot, which is discussed below:

\[
\text{call plot(denplot)};
\]

Bayesian Estimation and Monte Carlo Integration

It is sometimes required, when estimating the regression model \( y = X\beta + \epsilon \), that the estimate \( \hat{\beta} \) satisfy certain inequality restrictions. Following the classical frequentist approach, a constrained least squares estimator could be obtained by solving a quadratic programming problem. A drawback to this estimator is that its distributional properties are not known, making inference difficult.

An alternative approach is to treat \( \beta \) as a random variable and apply Bayesian techniques. The article by Griffiths [1988] is a good intuitive discussion of applying Bayesian methods to impose inequality restrictions on regression models. A simple example is given there of a consumption function, \( y_i = \beta x_i + \epsilon_i \), where \( y \) is consumption and \( x \) is income. The parameter \( \beta \) is called the long-run marginal propensity to consume (or mpc), and economic theory says that \( \beta \) should be between 0 and 1. A more complicated model, yet similar in principle, is treated by Chalfant, Gray and White [1991]. Their model will be discussed further below.

Let \( p(\beta) \) be the prior probability distribution density of \( \beta \) and \( f(y/\beta) \) the likelihood function given the sample \( y \). Then by Bayes' Theorem,

\[
p(\beta/y) = \frac{f(y/\beta)p(\beta)}{f(y)}
\]

or \( p(\beta/y) \propto f(y/\beta)p(\beta) \), where \( f(y) \) is the marginal density of \( y \) and \( p(\beta/y) \) is the posterior density of \( \beta \). The uncertainty in \( \beta \) before the sample is drawn is summarized by \( p(\beta) \), and \( p(\beta/y) \) represents the uncertainty in \( \beta \) after the sample is drawn. A point estimate of \( \beta \) can be obtained by defining a quadratic loss function and finding that value of \( \beta \) that minimizes expected loss (under the posterior distribution). This optimal estimate is the mean of the posterior distribution.

For the simple consumption function example, assuming \( \epsilon \sim \text{N}(0,1) \), the likelihood will have the form

\[
f(y/\beta) = \exp\left(-\frac{1}{2} \sum_{t=1}^{n} (y_t - x_t\beta)^2\right)
\]

Writing \( y_t - x_t\beta = y_t - x_t\hat{\beta} + x_t\hat{\beta} - x_t\beta \), expanding and ignoring terms not involving \( \beta \), we get

\[
f(y/\beta) = \exp\left(-\frac{1}{2} \sum_{t=1}^{n} x_t^2 (\beta - \hat{\beta})^2\right)
\]

where \( \hat{\beta} \) is the OLS estimate of \( \beta \). The posterior density for the mpc is then

\[
p(\beta/y) \propto p(\beta) \exp\left(-\frac{1}{2} \sum_{t=1}^{n} x_t^2 (\beta - \hat{\beta})^2\right)
\]

The specific form of the posterior depends on the prior, \( p(\beta) \). The prior could be categorized as follows:
informative: e.g., $\beta \sim N(0.85, 0.004)$, which is roughly equivalent to saying that $P(0.75 < \beta < 0.95) = 0.9$.

non-informative: e.g., $p(\beta) = 1$ for $-\infty < \beta < \infty$. This is an improper density.

inequality restriction: e.g., $p(\beta) = 1$ for $0 < \beta \leq 1$, uniform density.

For the MCMC, a prior of type (3) seems appropriate. The posterior becomes a truncated normal distribution in this case. In general, priors reflecting inequality restrictions lead to truncated distributions for the posterior. The next step is to find the mean of the posterior distribution. We can also ask such questions as:

What is the probability that $\beta \leq 0.95$?

In general, these integrals cannot be evaluated in closed form and must be approximated using Monte Carlo integration. The principle behind Monte Carlo integration is as follows: To evaluate the integral $\int f(x)g(x) \, dx$, where $f(x)$ is a (possibly multivariate) density function, we note that $\int f(x)g(x) \, dx = E_g[f(x)]$. If we can make $N$ draws from the $f$-distribution, then

$$\frac{1}{N} \sum_{i=1}^{N} h(x_i) = \int h(x) f(x) \, dx$$

To implement this procedure, it must be possible to make random draws from the $f$-distribution. When this is not possible, draws are made from another density, $g(x)$, called an importance density. This works because of the simple identity

$$\int f(x)g(x) \, dx = \int \{h(x)f(x)/g(x)\}g(x) \, dx = E_g[h(x)f(x)/g(x)]$$

where $E_g$ indicates the expectation taken with respect to the $g$-distribution.

The system of equations is estimated using Zellner's Seemingly Unrelated Regression (SUR) technique, where the error vector $\varepsilon$ is assumed to be distributed as a multivariate normal, $\varepsilon \sim N(0, \Sigma)$. A normal likelihood, diffuse prior on $\Sigma$, and inequality prior on $\theta$ results in a (marginal) posterior density for $\theta$ of the form:

$$p(\theta|y) \propto p(\theta) |A|^{-T/2}$$

where $T$ = the number of observations and $A = (a_i)$,

$$a_i = E_\theta e_i' e_i$$

and $E_\theta$ is the Tx1 vector of residuals for the $i$th equation in the system. Note that $p(\theta|y)$ is a truncated distribution since $p(\theta) = 0$ for $\theta \notin D$. Since $p(\theta|y)$ is not a "familiar" density, a random sample cannot be drawn from the posterior distribution and a truncated multivariate t-distribution is used as an importance density.

The following two questions can now be addressed:

What is the probability of $\theta$ satisfying the inequality restrictions? i.e., evaluate the integral

$$P(\theta \in D) = \int_{\theta \in D} \frac{f(\theta|y)}{g(\theta|y)} \, d\theta$$

assuming a diffuse prior on $\theta$.
(2) What is the mean of the posterior distribution? i.e., evaluate the integral
\[ E(\theta/data) = \int f(\theta) \frac{f(x/\theta)}{g(\theta/x)} \cdot g(\theta/x) \, d\theta \]
\[ = \int_{\theta \in D} f(\theta) \frac{f(x/\theta)}{g(\theta/x)} \cdot g(\theta/x) \, d\theta \]
The last equation follows since \( p(\theta) \) is just the indicator function for the set \( D \).

An outline of the algorithm to answer these questions, along with the accompanying IML code, is given next.

Step 1: Estimate the unrestricted model by iterated SUR to obtain the maximum likelihood estimates \( \hat{\theta} \) and \( V(\hat{\theta}) \). Since there is no need to reinvent the wheel in IML, this can be done with Proc Model in SAS/ETS software.

Step 2: Construct the multivariate t distribution function to serve as the importance density. In the following IML code, the SAS data set 'ets.theta' contains the output from Proc Model.

The first observation is \( \hat{\theta} \) and the remaining observations are the rows of \( V(\hat{\theta}) \). Since \( \hat{\theta} \) has 15 elements in this example, 'ets.theta' has 16 observations. The first section of code brings the output from Proc Model into IML.

\* read estimates saved from Proc Model;
libname ets 'mydir';
* 1st read theta and transpose;
use mydir.thetavar{p1 p2 ... p15};
read point 1 into theta;
theta = theta';
* next read covariance matrix;
read point (2:16) into cov;
* find Cholesky factor of cov;
H = root(cov);
* save matrices in storage catalog;
reset storage = 'mylib.unconstr';
store theta H;

Now that we have a matrix \( H \) such that \( V(\hat{\theta}) = H \cdot H' \), we can construct the multivariate t with \( \lambda \) degrees of freedom. The motivation for using the antithetic draw below and "getting two for the price of one" is explained in Geweke [1988].

\* set degrees of freedom for this problem;
df = 4;
* 1st draw multivariate normal, size of theta;
nparams = nrow(theta);
norm1 = normal(J(nparams,1,0));
* next draw multivariate normal, size of df;
norm2 = normal(J(df,1,0));
* construct chi-square using norm2;
chisqr = (norm2*norm2);
* now put pieces together;
add = H*norm1/(sqrt(chisqr/df));
* get random draw from importance distn;
theta1 = theta + add;
* get antithetic draw;
theta2 = theta - add;

The above statements would appear in a loop where a total of \( N \) draws would be made.

Step 3. With each replication in the above loop, check if the random draws \( \theta_1 \) and \( \theta_2 \) satisfy the inequality restrictions. Using the *successful* draws, say there are \( n \) out of a total of \( N \), we can estimate the posterior mean and probability of the restrictions holding by:

\[ \overline{\theta} = \frac{\sum_{i=1}^{n} f(\theta_i/x)}{\sum_{i=1}^{n} g(\theta_i/x)} \]
\[ P(\theta \in D) = \frac{\sum_{i=1}^{n} f(\theta_i/x)}{\sum_{i=1}^{n} g(\theta_i/x)} \]

The denominators in the above expressions are normalizing factors to account for the fact that we did not carry along constants with the various density functions.

Step 4. The next task with each draw of \( \theta \) is to evaluate the posterior and the importance densities at the given value of \( \theta \). Assuming the Txq matrix errs (\( T = \) number of observations and \( q = \) number of equations estimated) has already been defined to be the errors for the system of equations in the model after the drawn \( \hat{\theta} \) is substituted into the equations, we have:

\* define matrix A used for posterior;
A = errs*errs;
* marginal posterior pdf for theta;
post = 1/(det(A))**(nobs/2);
* multivariate t, importance pdf;
lt = 1/(df + add*inv(cov)*add)**((df+nparm)/2);

Step 5. The last calculation to make for each iteration is to check whether or not the draw, \( \theta \), is "successful" or not, i.e., does \( \theta \) satisfy the inequality restrictions of \( \theta \in D \). In this example
"concavity" means that the eigenvalues of a certain matrix are negative. Assuming the matrix of interest is called submat and has already been constructed, then:

* find eigenvalues of submat;
  eval = eigval(submat);
* check if all eigenvalues negative;
  concave = (eval[< ] <= .000000001);
* count successful draws;
  count = count + concave;

Two things to note about the above code. First, IML being a matrix language, there is a built-in function to find eigenvalues of (symmetric) matrices. Second, the variable concave is defined by a logical condition, eval[< ] <= .000000001. The expression eval[< ] selects the maximum element of the matrix eval. If eval[< ] is non-positive (allowing for some round-off error), then the value of concave is 1, else concave = 0. Thus the variable count is only incremented when the inequality restriction holds for the drawn i.

Graphics

The graphics commands in IML are available as long as SAS/GRAPH® is also installed. IML has 28 graphics commands or primitives. A graphics session is initiated by CALL GSTART; and ended by CALL GSTOP;. A collection of primitives such as GDRAW, GPOINT, and GXAXIS form a segment or graph. Segments can be named in the GOPEN statement and saved in a permanent SAS catalog. Segment attributes such as color, line style and font can be defined either with the GSET command or the GOPTIONS statement.

One of the "dirty details" that has to be taken care of when constructing a plot is to find "nice" numbers with which to label the axes. IML provides the function GSCALE for just this purpose. The statement

    call gscale(xscale,x,nincr);

will return the vector xscale containing the scaled minimum, maximum and grid increment for the input data matrix x. The parameter nincr defines the number of intervals desired for the axis. For example, if the data values in x ranged from 2.02371 to 10.8743, then we might get xscale[1] = 2.0 and xscale[2] = 11.0.

I recently had the opportunity to use the package S-PLUS® for DOS. S-PLUS is a function oriented language with a set of so-called high level graphics functions to originally produce a graph and low level functions to add text, lines, points, etc. to an existing graph. A large collection of graphics parameters have default values, so a graph can be produced quickly and easily by entering, for example, plot(x). If x is an nx1 vector, then the graph is a plot of x[i,1] versus i. If x is an nx2 matrix, then the graph is a plot of x[i,2] versus x[i,1].

Doing ad hoc plots in IML requires that a new program be written each time. A window and the axes have to be defined, labeling of the axes determined, etc.. In an attempt to make doing plots in IML as easy as it is in S-PLUS, I have written a collection of IML modules that emulate the kinds of S-PLUS functions mentioned above. The modules are:

par - define default values for graphics parameters.
plot - create graph which plots either y versus x or x versus observation number.
abline - plot line y = a + b*x on current graph.
text - place text on current graph.
points - plot points on current graph.
hpgl - copy current graph to a file in HPGL format.
prtpar - print current values of parameters.
lsfit - function to regress y on x and return estimated intercept and slope.

A short description of each module follows.

1. start par;
   device='pmvga'; * default graphics device;
   cplot='black'; * default color for plots;
   type='p'; * plot type, 'p' = points
   'l' = lines;
   * margins as % of axes, bot left top right;
   map = (.30 .20 .18 .12);
   . (similar assignment statements)

   finish;

   Note that all the variables defined in par are global, and so are available for reference in other modules.

2. start plot(data) global(device,cplot,type,...);
   call execute('goptions device=', device,
   'cback=',cplot,';');
* check form of incoming data to plot;
if ncol(data) = 1 then
  do;
    x = 1:nrow(data);
    y = data;
    xlab = 'Index';
  end;
else if ncol(data) = 2 then
  do;
    x = data[,1];
    y = data[,2];
  end;
else
  do;
    print '***Data has wrong # of columns***';
    return;
  end;
* start graphics session & open segment;
call gstart;
call gopen;
* set up window & plot area;
call gscale(xs, x, nincr);
call gscale(ys, y, nincr);
xlength = xs[2] - xs[1];
ylength = ys[2] - ys[1];
* leave room in margins for plot;
bot = map(1)*ylength;
left = map(2)*xlength;
top = map(3)*ylength;
right = map(4)*xlength;
x1 = xs[1] - left;
x2 = xs[2] + right;
y1 = ys[1] - bot;
y2 = ys[2] + top;
* define window in user coordinates;
window = (x1 | y1) // (x2 | y2);
call gwindow(window);
---REST OF MODULE SETS UP AXES, LABELS, TITLE, PLOT, ETC.---
* now show plot;
call gshow;
finish;

Note that all the default parameter values are available to plot via the global option on the start command. Also, it is necessary to use "call execute" for the goptions statement in order to delay execution of goptions until the module plot is executed. Otherwise, the goptions statement is executed during the compile phase of the module.

3. start abline(ab) global(ltype, cplot, xscale);
   * extract intercept & slope;
   a = ab[1];
   b = ab[2];
   * find 2 points in graph window to use in drawing the line;
   x = xs[1] // xs[2];
   y = a + b*x;
   * now draw the line on the current graph;
call gdraw(x, y, ltype, cplot);
call gshow;
finish;

Note that the variable xscale was put in the global symbol table when it was calculated in plot so that it would be available to other modules, such as abline, without having to pass it as a parameter.

4. start text(x, y, string) global(htext, font, ctext);
   * write 'string' at point (x, y);
call gscript(x, y, string, htext, font, ctext);
call gshow;
finish;

Note that the IML module gscript accepts many parameters. By defining default values for parameters such as height of the text (htext), color of the text (ctext), and the text font (font) in the module par, default values can be set for these parameters once, and then it is not necessary to pass along so many parameters each time text is to be written to the current graph.

5. start lsfit(x, y);
   * center the data;
xbar = x[:];
ybar = y[:];
xd = x - xbar;
yd = y - ybar;
* calculate OLS intercept & slope;
b = (yd#xd) / ssq(xd);
a = ybar - b*xbar;
* set up 2 element vector of estimates;
ab = a || b;
return(ab);
finish;

Note that lsfit is a function module that returns the OLS estimates from regressing y on x. Also notice how easy it is to do the calculation.
\[ \beta = \frac{\sum_{i=1}^{n} (y_i - \bar{y}) (x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \]

So to fit a regression through the scatterplot and draw it on the current graph, we issue the command: `call abline(lsfit(x,y));`

The plot below was created with the following statements:

```plaintext
x = 10*uniform(25,1,0);
y = 1 + 2*x + 1.5*normal(25,1,1));
x[1] = 9; y[1] = 4;
title = 'Plot with Outlier';
call plot(x,y);
call abline(lsfit(x,y));
call text(5,3.5,'Outlier->');
```

For more examples using IML graphics, see the paper by Hallahan [1991] in the SUGI 16 Proceedings.

**Conclusion**

In summary, I find IML to be a convenient programming language for those analyses not having their own Proc. In addition to the examples presented in this paper, I have also used IML for Box-Cox transformations, stochastic parameter regression models, maximum likelihood estimation, and stochastic dynamic programming for rational expectations models. The paper by Leyden [1991] contains further applications of IML.

**References**


SAS, SAS/ETS, SAS/GRAPH, SAS/IML, SAS/OR, SAS/QC and SAS/STAT are registered trademarks of SAS Institute Inc., Cary NC, USA
S-PLUS is a registered trademark of StatSci, Seattle, WA, USA