Paper 329-2010

Rediscovering SAS/IML® Software: Modern Data Analysis for the Practicing Statistician

Rick Wicklin, SAS Institute Inc., Cary NC

ABSTRACT

For over 25 years, the SAS/IML programming language has been used by researchers and SAS® programmers who write custom algorithms and analyses. Recently, SAS/IML® Studio arrived with flashy statistical graphics, new language features, and the ability to call the open-source statistical language, R.

If you are a longtime SAS/IML programmer, it is time to take a new look at the SAS/IML language. Recent language-features and functions make it easier than ever to program modern data-analytic techniques. If you have not used SAS/IML software before, it is time to discover how its high-level matrix-vector programming language can increase your productivity.

This paper presents short programs that implement modern data analyses in SAS/IML software. These analyses include exploring outliers with regression diagnostics, robust estimation of multivariate statistics, bootstrap permutation tests, and calling functions in R that compute certain exact statistics.

INTRODUCTION

The statistical power of SAS procedures and the data manipulation capabilities of the DATA step are sufficient to serve the analytical needs of many data analysts. However, sometimes you need to implement a proprietary algorithm or an algorithm that has recently been published in a professional journal. Other times, you need to use matrix computations to combine and extend the results from procedures. In these situations, you can turn to SAS/IML software.

SAS/IML software has two components: the IML procedure and the SAS/IML Studio application. PROC IML, a SAS product for more than 25 years, is a computational procedure that implements the SAS/IML language for matrix-vector programming. This language includes standard programming features such as DO loops and IF/THEN-ELSE statements, along with a rich library of hundreds of statistical and linear algebraic functions. The syntax of the programming language is similar to the DATA step, except that the fundamental "unit" in a SAS/IML program is a matrix, whereas in the DATA step it is an observation. The high-level SAS/IML language enables you to formulate and solve statistical problems by using functions and expressions that mirror the mathematical equations that you see in textbooks and in research journals.

The SAS/IML Studio application is a more recent development. The product was initially released as SAS/IML Workshop in 2001. For SAS 9.2, the product was briefly called SAS® Stat Studio, but was renamed SAS/IML Studio in 2009 to better emphasize its relationship to the SAS/IML programming language.

SAS/IML Studio is an environment for developing SAS/IML programs and for exploring data by using high-level statistical graphics. It is available to all SAS/IML users who also license SAS/STAT software. SAS/IML Studio runs on a Windows PC and can connect to one or more SAS workspace servers. Many of the SAS/IML Studio features—including the ability to call SAS procedures from within a SAS/IML program—are described in Wicklin (2008). See *SAS/IML Studio for SAS/STAT Users* for a short introduction to programming SAS/IML Studio. For a detailed exposition, see Wicklin (2010). There is also a discussion forum about SAS/IML software available at `http://support.sas.com/forums/index.jspa`.

This paper shows you how to program classical and modern data-analytic techniques in SAS/IML Studio. The following techniques are included in this paper:

- defining a subroutine to compute robust statistics
- analyzing regression diagnostics to detect outliers and high-leverage points
- estimating robust multivariate statistics
- implementing bootstrap methods and permutation tests
- calling functions in R

A MOTIVATING EXAMPLE

For readers who are unfamiliar with the SAS/IML language, this section presents a brief example that shows why you might want to invest time in learning the language. Suppose that you have several variables with extreme outliers. For each variable, you want to standardize the variables by using robust estimates of the location and scale parameters. You decide to estimate the location parameter with a median and to estimate the scale parameter with the median absolute deviation (MAD) from the median. How can you standardize the variables? You can use SAS/IML statements for both the computation and the transformation, as shown in the following statements which assume that **x** is an $n \times p$ data matrix:

```
proc iml;
use MyLib.MyData;               /* open data set for reading */
read all var _NUM_ into x;      /* read p numeric variables  */
close MyLib.MyData;             /* close the data set        */

m = median(x);                  /* 1 x p vector of medians   */
s = mad(x);                     /* 1 x p vector of MAD       */
stdX = (x-m)/s;                 /* center and scale          */
```

The MEDIAN function computes the median of each column of the data matrix and stores the results in a $1 \times p$ row vector, **m**. The MAD function computes the MAD statistic for each column and stores the results in **s**. These estimates are used to transform each column of the data matrix. The expression **x-m** centers the data by subtracting the ith median from the ith column of **x**. The expression **(x-m)/s** divides the ith column of the centered data by the ith scale factor and saves the result in the matrix **stdX**. Notice that there is no need to loop over observations or variables; the SAS/IML language performs operations on matrices of data.

EXTENDING THE SAS/IML FUNCTION LIBRARY

The rich library of functions and the fact that you can perform matrix arithmetic in a natural and succinct way are two reasons why SAS/IML software is appealing to data analysts. Another reason is that you can create (and share) a library of SAS/IML modules that extend the built-in SAS/IML library.

For example, suppose you want to alter the previous example so that the location and scale are estimated by the 10% trimmed mean and the trimmed variance, respectively. There is no built-in function that computes the trimmed variance, but you can compute both of these quantities by defining a *module*. A module is a function or subroutine written in the SAS/IML language that you can define, store, and call from your code as if it were a built-in function.

You begin the module definition with the START statement and end the definition with the FINISH statement. To compute the 10% trimmed mean, you can sort the data and exclude the smallest and largest 10% of the data. You can then compute the mean of the trimmed values. The trimmed standard deviation is computed similarly. The following statements define a module that computes these robust estimates:

```
/* This module computes the trimmed mean and trimmed standard deviation
 * for each column of a matrix.
 * INPUT      x:  The nxp data matrix. Assume no missing values.
 *         prop:  A proportion of observations to trim, 0 < prop <0.5.
 *                The smallest d and largest d observations are trimmed
 *                from each column, where d = prop*n.
 *
 * OUTPUT  mean:  A 1 x p vector of trimmed means
 *          std:  A 1 x p vector of trimmed standard deviations
 */
start Trim(mean, std, x, prop);
   p = ncol(x);               /* number of columns                        */
   n = nrow(x);               /* num rows (assume no missing values)      */
   d = ceil(prop*n);          /* number of observations to trim           */
   mean = j(1, p);            /* allocate results matrix for mean         */
   std  = j(1, p);            /* allocate results matrix for std. dev.    */
   do i = 1 to p;
      z = x[,i];              /* copy i_th column                         */
      call sort(z,1);         /* sort data in i_th column                 */
      w = z[d+1:n-d];         /* trim d largest and d smallest values     */
      mean[i] = w[:];         /* store mean of trimmed data               */
      std[i] = sqrt(var(w)); /* store std. dev. of trimmed data          */
   end;
finish;

/* call the Trim module to compute trimmed mean and std. dev. */
run Trim(m10, s10, x, 0.1);  /* m10=trimmed mean; s10=trimmed std. dev. */
stdX = (x-m10)/s10;          /* standardize data                         */
```

In the previous program, the name of the module is Trim. The module takes two input arguments: the data matrix, **x**, and a 1×1 matrix, **prop**, that specifies the proportion of observations to trim in each tail. The module returns its computations in two output arguments. The main steps of the program are as follows:

1. The NCOL and NROW functions determine the number of columns and rows, respectively, in the data.
2. The CEIL function in Base SAS software computes how many observations will be trimmed, based on the value of **prop**.
3. The J function allocates space for the statistics prior to the main computational loop.
4. A column of the data is copied into the **z** vector and is sorted.
5. The extreme values are discarded and the central portion of the data is copied into the vector **w**. The expression within the brackets is a vector of indices. The index operator (**:**) is used to generate the elements to copy. For example, if $n = 100$ and **prop** $= 0.1$, then the expression in the brackets is equivalent to **11:90**, which is equivalent to the vector **{11 12 ... 89 90}**. The brackets are subscript operators, so **w[11:90]** means to extract the central 80% of the ordered data.
6. The subscript reduction operator (**:**) computes the mean of the trimmed data.
7. The VAR function is used to compute the standard deviation of the trimmed data. (See the Appendix for a note about the VAR function.)

After the module is defined, you can call the module to compute the trimmed mean and standard deviation. You can share the module source with your colleagues, or store a compiled form in a SAS catalog that can be accessed by everyone in your group.

In summary, the SAS/IML language is appealing because of its rich library of functions, because of the fact that you can perform matrix arithmetic in a natural and succinct way, and because you can extend the capabilities of the software by defining your own functions and subroutines.

REGRESSION DIAGNOSTIC PLOTS IN SAS/IML STUDIO

SAS/IML Studio provides the computational power of the SAS/IML language, the capability to call any SAS procedure, and high-level statistical graphics that can assist you in building statistical models and investigating outliers. This section describes how you can use SAS/IML Studio to create interactive regression diagnostic plots.

THE U.S. DOMESTIC AIRLINE DATA

The data set used in this section is a subset of data from the ASA Data Expo 2009 (American Statistical Association 2009). See Wicklin (2009) for an account of how SAS/IML Studio was used to analyze aspects of the data. This paper uses two subsets of the data:

ORD_AA is a simple random sample of 1,000 American Airline flights in 2007 that originated at the Chicago O'Hare International Airport (ORD) and terminated at one of fifteen major airports in the continental U.S. The data set includes the following variables:

- **ArrDelay** is the number of minutes that the flight was delayed. A negative number means that the flight arrived ahead of schedule.
- **CRSDepTime** is the scheduled departure time according to the computer reservation system (CRS).
- **logArrDelay** is a transformation of the ArrDelay variable: logArrDelay $=$ log(ArrDelay $+ 30$).
- **LateAircraftDelay** is one of several causes of delay. This variable contains the number of minutes that a flight was delayed because the aircraft that was needed for the flight was late arriving from another airport.

mvts contains delay data that consist of 365 observations and 21 variables. The Date variable records the day in 2007 for which the data were recorded. The other 20 variables contain the mean delays, in minutes, experienced by an airline carrier for that day's flights. The airline variables in the data are named by using two-digit airline carrier codes, as listed in Table 1.

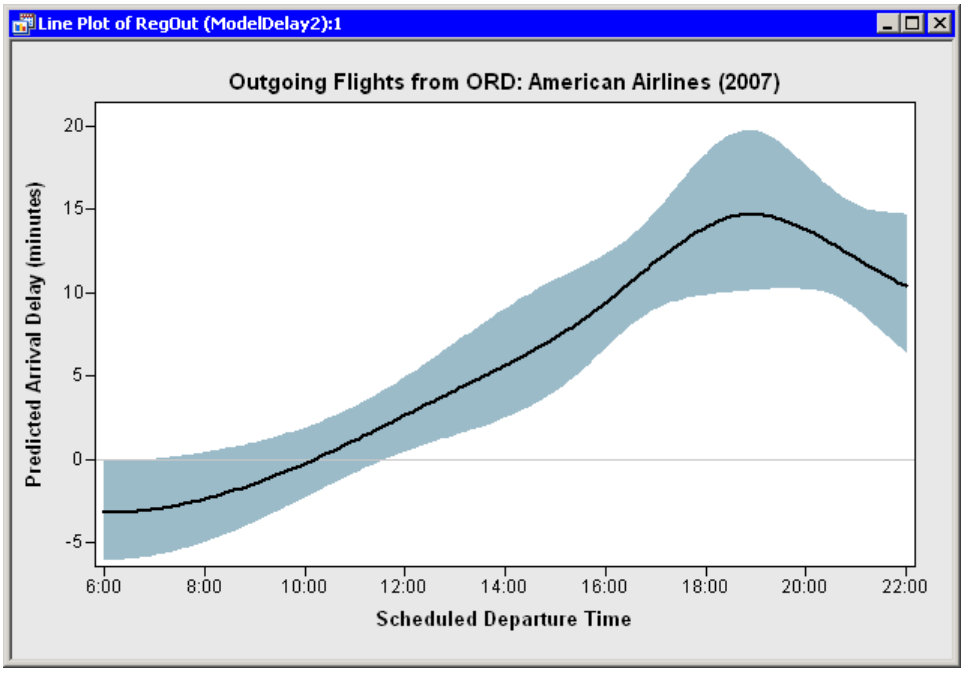
Table 1 Major Airline Carrier Codes

Code	Airline	Code	Airline	Code	Airline
AA	American Airlines	AQ	Aloha Airlines	AS	Alaska Airlines
B6	JetBlue Airways	CO	Continental Airlines	DL	Delta Air Lines
EV	Atlantic Southeast Airlines	F9	Frontier Airlines	FL	AirTran Airways
HA	Hawaiian Airlines	MQ	American Eagle Airlines	NW	Northwest Airlines
OH	Comair	OO	SkyWest Airlines	PE	Pinnacle Airlines
UA	United Airlines	US	US Airways	WN	Southwest Airlines
XE	ExpressJet Airlines	YV	Mesa Airlines		

SMOOTHING A SCATTER PLOT

Of interest to travelers is the *delay profile* of a particular carrier for an airport. The delay profile is the mean delay experienced by flights from that airport plotted against the scheduled departure time. The delay profile for the American Airlines flights in the ORD_AA data set are shown in Figure 1. The figure is created by using SAS/IML Studio, as are all figures in this paper.

Figure 1 A Delay Profile



The delay profile shows that, on average, American Airline flights (from ORD) that leave in the early morning hours arrive at their destination as scheduled or perhaps even a little ahead of schedule. However, the expected delay increases as the day progresses. For flights that are scheduled to depart at 6:00 p.m., the expected delay is 15 minutes. Later in the evening, the expected delay decreases. The band in the figure indicates a 95% confidence limit for the mean delay. For example, the expected delay for flights scheduled to depart at 6:00 p.m. is between 10 and 20 minutes, with high confidence.

You can obtain a delay profile by smoothing a scatter plot of ArrDelay versus the scheduled departure time for each flight. (Actually, because the distribution of the ArrDelay variable has a long tail, it is useful to transform the ArrDelay variable by a logarithmic transformation such as $\text{logArrDelay} = \log(\text{ArrDelay} + 30)$.) There are many ways to create a graph like Figure 1. For example, you can use nonparametric regression techniques such as a loess smoother or penalized B-splines, as implemented in the LOESS and TRANSREG procedures, respectively. These nonparametric techniques produce a delay profile that looks similar to Figure 1. You can also use variable selection techniques, as implemented in the GLMSELECT procedure, to select the most important variables from a variety of B-spline basis functions.

You can use the following steps to write a program that creates Figure 1.

1 Use the SUBMIT and ENDSUBMIT statements to call the GLMSELECT and REG procedures. The GLMSELECT procedure determines a small set of B-spline basis functions that best predict the observed delays (Cohen 2009). The REG procedure then uses these selected variables to run a classical ordinary least squares regression. The predicted values and confidence limits for the mean predictions are written to an output data set.

```
/* SAS/IML Studio program to create regression diagnostic plots */
submit;
proc glmselect data=sgf10.ORD_AA
               outdesign(addinputvars)=design;  /* output the design matrix */
   effect spl = spline(CRSDepTime / split
          knotmethod=multiscale(endscale=2));
   model logArrDelay = spl / selection=LASSO;   /* use LASSO method         */
quit;

proc reg data=design;                           /* read from design matrix  */
   model logArrDelay = spl_:;                   /* use selected splines     */
   output out=RegOut P=P UCLM=UCLM LCLM=LCLM    /* predictions and CLM       */
                  RSTUDENT=ExtStudR             /* studentized residuals    */
                  COOKD=CookD H=Leverage;       /* influence diagnostics    */
quit;
endsubmit;
```

2 Read the results into SAS/IML vectors. Invert the log transformation so that predicted values are in the original scale.

```
use RegOut;
read all var {P UCLM LCLM} into logFit;
read all var {CRSDepTime};
close RegOut;

fit = exp(logFit) - 30;
```

3 Use the DataObject class to create an in-memory version of the data. *SAS/IML Studio for SAS/STAT Users* describes the use of the DataObject class and various plot classes.

```
declare DataObject dobj;
dobj = DataObject.CreateFromServerDataSet("work.RegOut");

/* add transformed predicted values to the data object */
dobj.AddVars({"Pred" "ObsNum"},
             {"Predicted Arrival Delay (minutes)" "Observation Number"},
             fit[,1] || T(1:nrow(fit)));
```

4 Create a line plot. Use the SAS/IML Studio drawing subsystem to draw the confidence band.

```
declare LinePlot line;
line = LinePlot.Create(dobj, "CRSDepTime", "Pred");
line.SetLineWidth(2);

line.DrawUseDataCoordinates();
line.DrawSetRegion(PLOTBACKGROUND);
LightBlue = 0x9BBBC9;               /* RGB value: (155, 187, 201) */
line.DrawSetBrushColor(LightBlue);
line.DrawSetPenColor(LightBlue);
N = nrow(CRSDepTime);
LCLM = fit[,2]; UCLM = fit[,3];
/* draw CLM band. This assumes the data are sorted by CRSDepTime. */
line.DrawPolygon(CRSDepTime//CRSDepTime[N:1], LCLM//UCLM[N:1], true );
```

REGRESSION DIAGNOSTIC PLOTS

The advantage of calling modeling procedures from SAS/IML Studio is that you can use the dynamically linked graphics in SAS/IML Studio to explore observations that have large influence on the model, either because they are outliers or because they are points of high leverage. The REG procedure in the previous program outputs a studentized residual, the leverage statistic, and Cook's D statistic for each observation. You can create graphs of these statistics and link them to variables that might explain outliers in the model, as shown in the following statements:

```
/* create residual plot: studentized residuals versus leverage */
declare ScatterPlot resid;
resid = ScatterPlot.Create(dobj, "Leverage", "ExtStudR");

/* create Cook's D plot */
declare ScatterPlot cook;
cook = ScatterPlot.Create(dobj, "ObsNum", "CookD");

/* create histogram of LateAircraftDelay variable */
declare Histogram hLate;
hLate = Histogram.Create(dobj, "LateAircraftDelay");
hLate.ReBin(0,15);                                  /* set bin width to 15 minutes */
hLate.SetAxisViewRange(XAXIS, 60, 270);             /* zoom graph to show tail     */
hLate.SetAxisViewRange(YAXIS, 0, 10);               /* of distribution             */
hLate.SetAxisTickUnit(YAXIS, 5);
```

Figure 2 shows the result of the complete program. In the figure, observations with large values of Cook's D statistic are highlighted. (Recall that Cook's D statistic indicates how deleting any one observation changes the parameter estimates.) You can highlight observations in SAS/IML Studio by dragging out a selection rectangle as shown in the lower-left graph. Because the graphs are dynamically linked, you can see these same observations highlighted in other graphs and data tables.

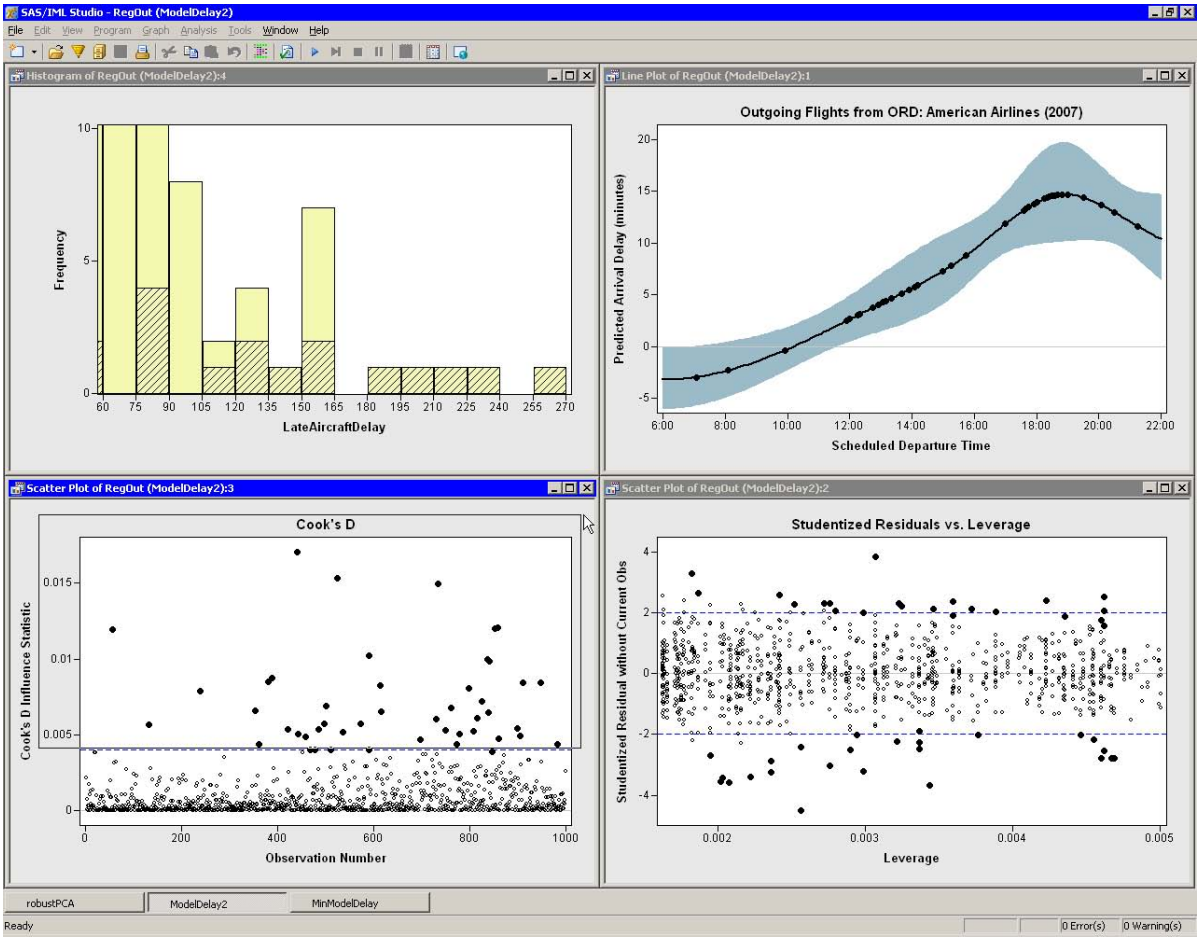
For example, the lower right plot in Figure 2 is a graph of externally studentized residuals versus the leverage statistic. The observations with large values of Cook's D statistic are highlighted, which enables you to see that the highly influential observations are outliers for the model. About half are flights whose arrival was delayed much more than predicted by the model; the other half arrived much earlier than predicted by the model. (The observation with the largest Cook's D statistic is a flight to Seattle that arrived 30 minutes ahead of schedule!) By looking at the upper-left graph, you can see that many of the influential observations are associated with late aircraft delays.

If you highlight only observations with large positive residuals (not shown), the upper-left graph in the figure shows that many of the flights with long delays were caused by a late aircraft delay. That is, the flight from ORD was delayed because a plane that arrived from some other location was late. This is a characteristic of hub airports such as ORD, and one of the reasons that the delay profile increases during the day: a flight that experiences a delay early in the day "passes on" that delay to other flights at other airports that use the same plane.

This brief example demonstrates several key advantages associated with using SAS/IML Studio for data analysis:

- You can call any SAS procedure from your SAS/IML program.
- You can read the output into SAS/IML vectors and matrices and perform additional computations or analyses.
- You can create dynamically linked statistical graphs that enable you to gain insight into the data that would be difficult to gain from static ODS graphics.

Figure 2 Observations with Large Values of Cook's D Statistic



ROBUST MULTIVARIATE OUTLIER DETECTION

The detection of outliers in multivariate data is both important and difficult. Outliers in low-dimensional data can often be identified by plotting the data, but in higher dimensions the number of possible projections makes graphical detection of outliers problematic. A numerical technique for detecting multivariate outliers is to estimate the central location of the data or the underlying distribution, and then calculate the distance from the center to each observation. An outlier is then defined as an observation that is farther than a certain distance from the center (Barnett and Lewis 1994).

The classical approach to identifying outliers is to use the Mahalanobis distance, which is essentially a scale-invariant metric that takes into account correlations between variables. If $\boldsymbol{S}$ is an estimate for the variance-covariance matrix of the underlying distribution, then the Mahalanobis squared distance (MSD) between a point $\boldsymbol{x}$ and a location $\boldsymbol{y}$ is given by

$$d^2_{\boldsymbol{S}}(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})'\boldsymbol{S}^{-1}(\boldsymbol{x}-\boldsymbol{y})$$

In particular, if the center of the data is estimated by the multivariate sample mean of the data, $\bar{\boldsymbol{x}}$, then the MSD for observation $\boldsymbol{x}_i$ is $d^2_{\boldsymbol{S}}(\boldsymbol{x}_i,\bar{\boldsymbol{x}})$. For multivariate normal data in p variables, it is well known that the MSD is distributed as χ^2_p (Hardin and Rocke 2005). Consequently, it is common to consider an observation to be an outlier if its MSD exceeds the 0.975 quantile of the χ^2_p distribution: $d^2_{\boldsymbol{S}}(\boldsymbol{x}_i,\bar{\boldsymbol{x}}) > \chi^2_{p,0.975}$.

A problem with the classical approach is that the sample mean and covariance estimates are not robust to outliers. That is, the presence of outliers is masked by the very estimates used to locate them (Hawkins, Bradu, and Kass 1984). A modern method for overcoming this difficulty is to estimate the center and shape of the data with robust statistics. The basic idea is to compute a robust estimate ($\boldsymbol{x}_R$) of the location and a robust estimate ($\boldsymbol{S}_R$) of the shape, and then use these estimates in the Mahalanobis distance formula. The robust squared distance from observation $\boldsymbol{x}_i$ to the center of the data is therefore defined as $d^2_{\boldsymbol{S}_R}(\boldsymbol{x}_i,\boldsymbol{x}_R)$. A popular algorithm that provides a robust estimate for the location and shape of the data is Rousseeuw's minimum covariance determinant (MCD) (Rousseeuw 1985). It is a topic of current research to determine the distribution of robust distances, but many practicing statisticians use $\chi^2_{p,0.975}$ as a cutoff value (Hardin and Rocke 2005).

The SAS/IML library of built-in functions contains several methods for estimating robust measures of univariate and multivariate data, including the MCD function which implements Rousseeuw's MCD algorithm. As an example, the following program looks at outliers in the 20-dimensional space of the variables in the mvts data set. The data are mean delays for each day in 2007 for 20 airline carriers. A visualization of the complete data set as a multivariate time series is shown in Figure 13.

```
/* compute robust estimates and print dates with top robust distances */
proc iml;

Carrier = {AQ HA WN F9 DL FL PE B6 CO OO XE AS YV US UA MQ OH NW AA EV};
use sgf10.mvts;                /* mean delays for each day in 2007     */
read all var Carrier into x;   /* data for each of 20 airline carriers */
read all var {Date};           /* read date as an ID variable          */
close sgf10.mvts;

N = nrow(x);                   /* 365 observations                     */
optn = j(8,1,.);               /* use default options for MCD          */
optn[4]= floor(0.75*N);        /* use 75% of data for robust estimates */

call MCD(sc,est,dist,optn,x);  /* compute robust estimates             */

RobustDist = T(dist[2,]);      /* robust distance returned             */
m = Date || RobustDist;        /* sort dates by robust distances       */
call sort(m, 2, 2);            /* in decreasing order                  */

Top10Dates = m[1:10,1];        /* print top 10 outlying days           */
RD = m[1:10,2];
print Top10Dates[format=DATE7.] RD[label="Robust Distance" format=5.2];
```

After reading the data into a SAS/IML matrix, **x**, the program sets up an option vector that is used to control the MCD algorithm. For this example, the default values are used except for the fourth option, which controls the size of the subset of observations used in the MCD algorithm. The program calls the MCD subroutine, which returns estimates of location and scale in the **est** matrix. The subroutine also returns a matrix, **dist**, which contains Mahalanobis distances, robust distances, and an indicator variable that has the value 1 if the robust squared distance of the corresponding observation is less than $\chi^2_{20,0.975}$.

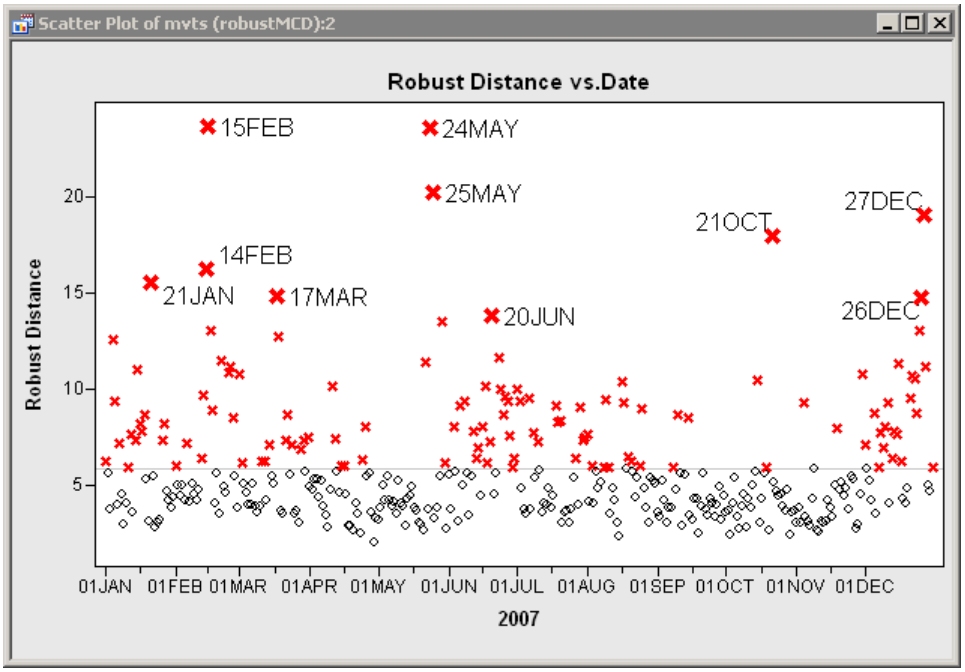
Figure 3 shows the dates for which airline delays were most unlike the delays for a "typical" day, where "typical" means a robust estimate of the location. Two days on the list are 14FEB07 and 15FEB07, which correspond to the "Valentine's Day Blizzard," a major winter storm that affected the entire eastern half of North America and paralyzed transportation in all forms. Also shown are dates within a few days of Christmas, a season in which airports are traditionally very busy and delays are commonplace. These outliers are examples of true multivariate outliers: the robust distance is large because of unusual values in many variables.

Figure 3 Dates in 2007 with the Largest Robust Distance

Top10Dates	Robust Distance
15FEB07	23.64
24MAY07	23.53
25MAY07	20.20
27DEC07	19.03
21OCT07	17.93
14FEB07	16.24
21JAN07	15.49
17MAR07	14.81
26DEC07	14.76
20JUN07	13.76

By using SAS/IML Studio graphics, you can create a graph of the robust distance versus the date, as shown in Figure 4. The dates with the 10 largest robust distances are labeled. Distances greater that the cutoff value are marked with an '×'. You can see that most dates in the spring and autumn have small robust distances, indicating that most days during these time periods do not experience atypical delays. In contrast, during the summer of 2007, there were frequently days with large robust distances, which indicates that one or more airlines experienced long delays.

Figure 4 Robust Distance versus Date



ROBUST PRINCIPAL COMPONENT ANALYSIS

Several outliers in the previous section are hard to comprehend until you explore the data by using the interactive graphics in SAS/IML Studio. The dates 24MAY07, 25MAY07, and 21OCT07 are dates on which a single carrier, Aloha Airlines (AQ), had a mean delay of 55 minutes or more. This is a *huge* delay for this otherwise consistently on-time airline: in 2007, the median delay for AQ was -3 minutes (yes, most flights arrive ahead of schedule) and estimates of the scale parameter are on the order of 3–5 minutes. Consequently, these dates are outliers because they are huge outliers in the scale of a single variable, not because of some massive nationwide delay that affects many airlines.

This highlights a potential problem for the data analyst: If an observation has a large robust distance, how can you determine the reason for the large value? Is it because many variables are moderately unusual, or because a single variable is extremely unusual?

One way to answer this question is to use a principal component analysis (PCA) to decompose the 20-dimensional space into factors that are more easily understood or interpreted. The principal components are the linear combinations of the variables that best explain the variation in the data. The classical PCA algorithm uses classical estimates of the multivariate location and shape parameters. However, when your data contain outliers, the outliers adversely affect the computation of the principal components.

You can overcome this problem by using robust estimates of the location and scale in place of the classical estimates. This is shown in the following statements, which continue the program in the previous section:

```
/* compute PCA of robust covariance matrix */
p = ncol(x);                        /* 20 variables                          */
RobustCov = est[3:2+p, ];           /* robust estimate of shape parameters   */
call eigen(eVal, eVec, RobustCov); /* PCA = eigenvectors of RobustCov       */
```

The EIGEN subroutine computes the 20 eigenvalues and the associated eigenvectors of the robust covariance matrix. The eigenvalues are contained in the **eVal** vector. The ith principal component is contained in the ith column of **eVec**. Recall that you can use a PCA as a dimension reduction technique: the first few principal components often explain the bulk of the variance in the data. You can use the CUSUM function to compute the cumulative proportion of the variance that is explained by the principal components, as shown in the following statement and as shown graphically in Figure 5:

```
VarExplained = cusum(eval)/sum(eval);   /* proportion of variance explained */
```

Figure 5 Proportion of Variance Explained by Principal Components

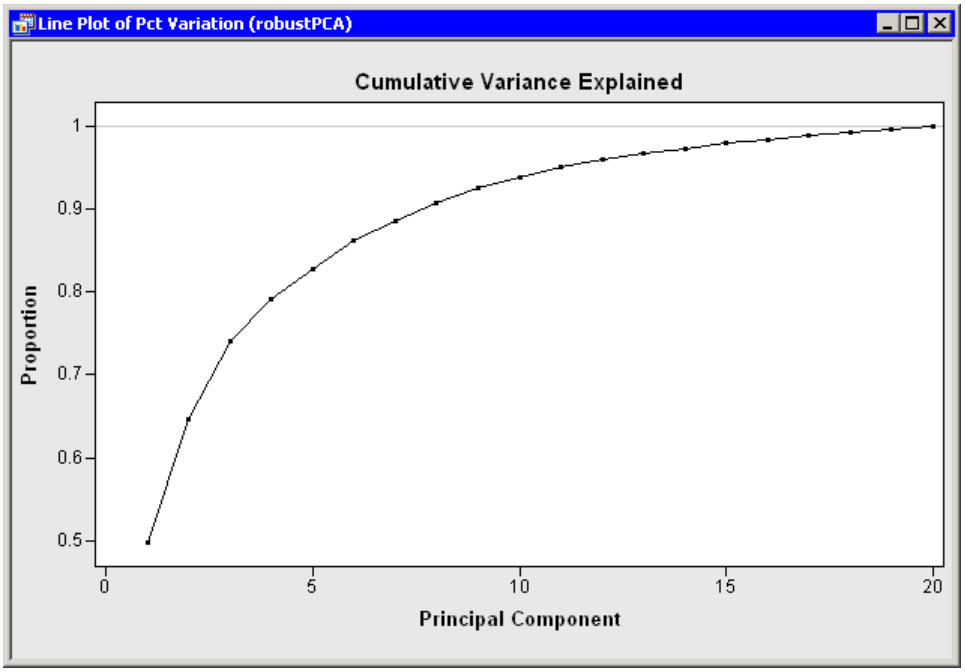


Figure 5 shows that about 75% of the variance is explained by using the first three principal components. You can visualize the data in this low-dimensional subspace by projecting the data onto the first few principal components, as shown in the following statements:

```
/* project data onto the first few principal components */
NumPC = 3;                          /* keep 3 principal components          */
RobustLoc = est[1, ];               /* robust estimate of location          */
c = (x-RobustLoc);                  /* center data around location estimate */
scores = c*eVec[,1:NumPC];          /* project centered data onto the PCs   */
```

Voilà! The robust principal component analysis is complete! The scores for the first two principal components are shown in Figure 6. For these data, the first principal component is a weighted average of the mean delays experienced by airlines that primarily fly routes in the continental U.S. Notice that some of the dates labeled in Figure 6 also appear in Figure 4; these dates have large robust distances in addition to large scores for the first principal component. In contrast, there are some dates in Figure 4 (for example, 27DEC07) that are not extreme outliers in Figure 6, and the converse is also true. For example, 19JUL07 does not appear in Figure 4 but has the third highest score for the first principal component.

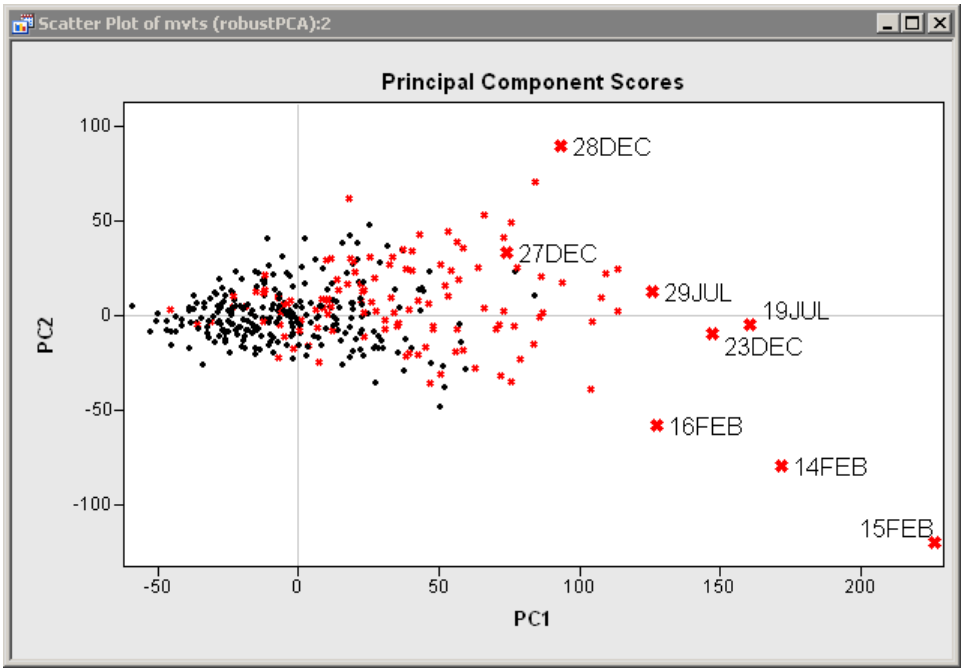
As mentioned previously, some dates have large robust distances because they are outliers along a certain dimension, such as the direction of the AQ variable. However, the AQ variable is essentially uncorrelated to the first principal component, so the projection of, say, 25MAY07, onto the plane of the first two principal components results in a score that is close to the origin.

To summarize, data analysts can use SAS/IML Studio for a robust principal component analysis, a modern statistical analysis that combines robust statistics with a classical dimension reduction technique. The software helps the analysts in the following ways:

1. You can use SAS/IML subroutines to compute robust statistics such as robust estimates of location and scale and robust Mahalanobis distances.
2. You can use SAS/IML subroutines to carry out the matrix manipulations that are required for multivariate analyses such as computing principal components and scores.
3. You can use SAS/IML Studio to create high-level statistical graphics. These graphics are dynamically linked, so that you can understand, for example, the relationship between robust distances and scores for the first principal component.

You can also use the robust covariance matrix to carry out other multivariate analyses. For an overview of robust multivariate analyses, see Barnett and Lewis (1994, p. 282).

Figure 6 Projection of Data onto the Subspace Formed by the First Two Principal Components



BOOTSTRAP PERMUTATION TESTS

According to the annual Airline Quality Report (Bowen and Headley 2008), Aloha Airlines (AQ) and Hawaiian Airlines (HA) were the top airlines for on-time performance in 2007. They also fly similar routes and experience similar weather conditions. Therefore, it is interesting to compare their performance. Is there evidence that one airline is on-time more often than the other?

A classical way to answer this question is to perform a matched-pair t test for the AQ and HA variables. A matched-pair t test tests the null hypothesis that the mean of the difference between paired values is zero, versus the alternative hypothesis that the mean is not zero. The following statements implement a matched-pair t test for these data:

```
/* classical matched-pair t test */
proc iml;
use sgf10.mvts;                      /* read data for mean delays       */
read all var {HA AQ};                /* for the AQ and HA airlines      */
close sgf10.mvts;

x = HA - AQ;                         /* form vector of differences      */
obsDiff = x[:];                      /* observed mean of differences    */
sd = sqrt(var(x));                   /* sample standard deviation       */
n = nrow(x);                         /* number of observations (=365)   */
stdErr = sd / sqrt(n);               /* sample standard error of mean   */
t = (obsDiff-0) / stdErr;            /* sample t statistic (H0: mu0=0)  */
pval = 1 - cdf("T", abs(t), n-1);    /* one-sided p-value               */
pval = 2*pval;                       /* two-sided p-value (by symmetry) */
print obsDiff t pval;
```

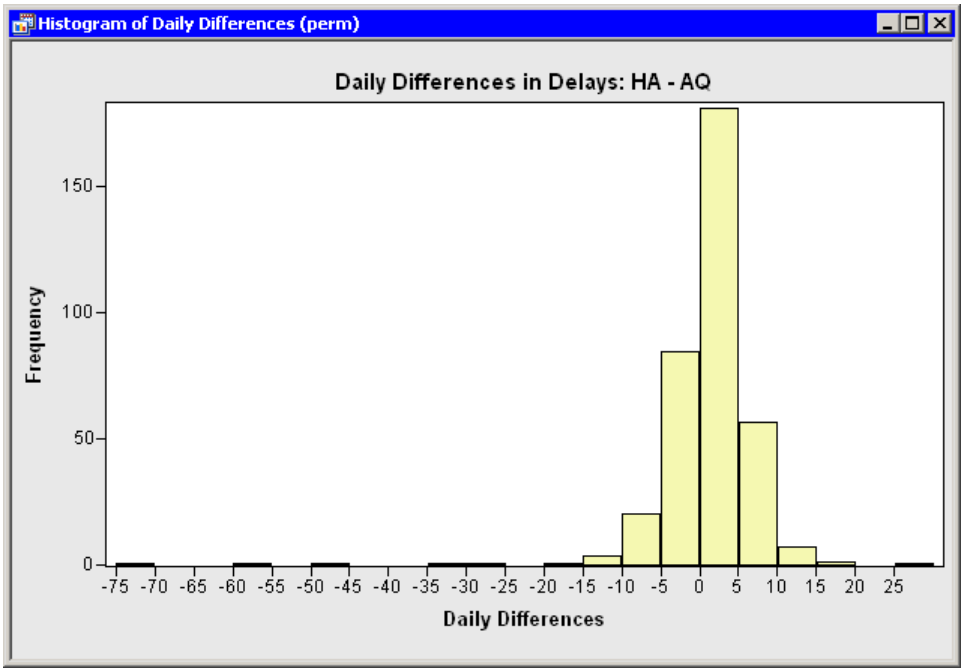
After reading the data into two vectors, **HA** and **AQ**, the program computes the observed mean difference between the delays experienced by the two airlines. This mean difference is 0.89 minutes, as shown in Figure 7. The t statistic for the difference is computed in the usual way: $t_{\boldsymbol{x}} = (\bar{x} - \mu_0)/(s/\sqrt{n})$ where $\bar{x}$ is the sample mean of $\boldsymbol{x}$, μ_0 is the population mean under the null hypothesis, s is the sample standard deviation, and n is the number of observations. The CDF function computes the probability under the null hypothesis of a value more extreme than the test statistic, **t**. For these data, the two-sided p-value is 0.0265. This seems to be a clear indication that you can reject the null hypothesis at the 95% confidence level; the means of the delays for the two airlines do not appear to be equal.

Figure 7 Results of a Classical Matched-Pair t Test

obsDiff	t	pval
0.8915595	2.2275654	0.0265203

However, if you examine the sample distribution of the differences between the airlines (see Figure 8), then it is apparent that the distribution is skewed and has outliers. Consequently, you might wonder whether the classical t test does an adequate job of testing whether the mean of the differences is zero.

Figure 8 Distribution of the Paired Differences between Delays for AQ and HA



A modern method for testing hypotheses and finding confidence intervals for parameters is the bootstrap method. Introductory references for bootstrap techniques include Efron and Tibshirani (1993) and Davison and Hinkley (1997). A previous article (Wicklin 2008) describes a different implementation of a bootstrap method in SAS/IML software.

The bootstrap technique that is useful for these data is the bootstrap permutation test for comparing the means of paired data. The basic bootstrap algorithm is as follows: Given matched pairs of data $(\boldsymbol{x}, \boldsymbol{y}) = ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$, the statistic of interest is the mean difference $s(\boldsymbol{x}, \boldsymbol{y}) = \overline{\boldsymbol{x} - \boldsymbol{y}}$. The bootstrap method resamples from the data many times in a way that is consistent with the null hypothesis. For the ith resample, $(\boldsymbol{x}^{*i}, \boldsymbol{y}^{*i})$, you compute the statistic $s(\boldsymbol{x}^{*i}, \boldsymbol{y}^{*i})$. The union of the statistics forms the *bootstrap distribution* for s. The bootstrap distribution is an estimate for the sampling distribution of the statistic. As such, you can use it to estimate the standard error of s, confidence intervals for s, and other quantities.

In this case, the null hypothesis is that the samples in the $\boldsymbol{x}$ vector come from the same underlying distribution as the samples in the $\boldsymbol{y}$ vector. If this is the case, for a given row you can swap the order of x_i and y_i and the resulting statistic for the permuted data is equivalent to the statistic evaluated on the original data. Therefore, a resampling method that is consistent with the null hypothesis is to randomly choose (with probability $1/2$) whether to permute the order of x_i and y_i for each row. The following statements demonstrate a single resampling step for the airline data:

```
/* permute HA/AQ values for random rows */
call randseed(12345);          /* initialize seed for random numbers     */
y = HA || AQ;                  /* concatenate values into 365 x 2 matrix */
u = j(nrow(y),1);              /* allocate vector to hold random numbers */

call randgen(u, "Uniform");    /* fill u with uniform variates in (0, 1) */
z = y;                         /* copy original data                     */
rows = loc(u>0.5);             /* locate rows for which u[i] > 0.5       */
z[rows,] = z[rows, {2 1}];     /* swap values of these rows              */
x = z[,1] - z[,2];             /* difference of permuted data            */
s = x[:];                      /* mean of difference                     */
print s;
```

The first three steps are preliminary to the actual resampling. A stream of pseudorandom numbers is initialized by the RANDSEED call, the data pairs are concatenated into a two-column matrix, and a vector, `u`, is allocated to hold pseudorandom numbers. The next steps carry out the resampling. The RANDGEN call fills the `u` vector with pseudorandom uniform numbers in the range $(0, 1)$. The matrix `z` is initialized as a copy of `y`. The LOC function finds the

indices of **u** for which the HA and AQ values will be swapped, and the HA and AQ values for those rows are interchanged. The **x** vector is computed as the difference between the columns of **z**, and the scalar quantity **s** is the mean difference. The value of **s** is shown in Figure 9. It is the value that is obtained by evaluating the statistic on a single bootstrap resample.

Figure 9 Mean Difference for a Single Bootstrap Resample

s
0.2831301

The next statements repeat the resampling many times to generate the bootstrap sampling distribution:

```
/* bootstrap permutation test for matched pairs */
B = 1000;                       /* number of bootstrap resamples          */
s = j(B,1);                     /* allocate vector to hold bootstrap dist */
do i = 1 to B;
   call randgen(u, "Uniform");
   z = y;                       /* copy original data                     */
   rows = loc(u>0.5);           /* locate rows for which u[i] > 0.5       */
   z[rows,] = z[rows,{2 1}];    /* swap values of these rows              */
   x = z[,1] - z[,2];           /* difference of permuted data            */
   s[i] = x[:];                 /* mean of difference                     */
end;
```

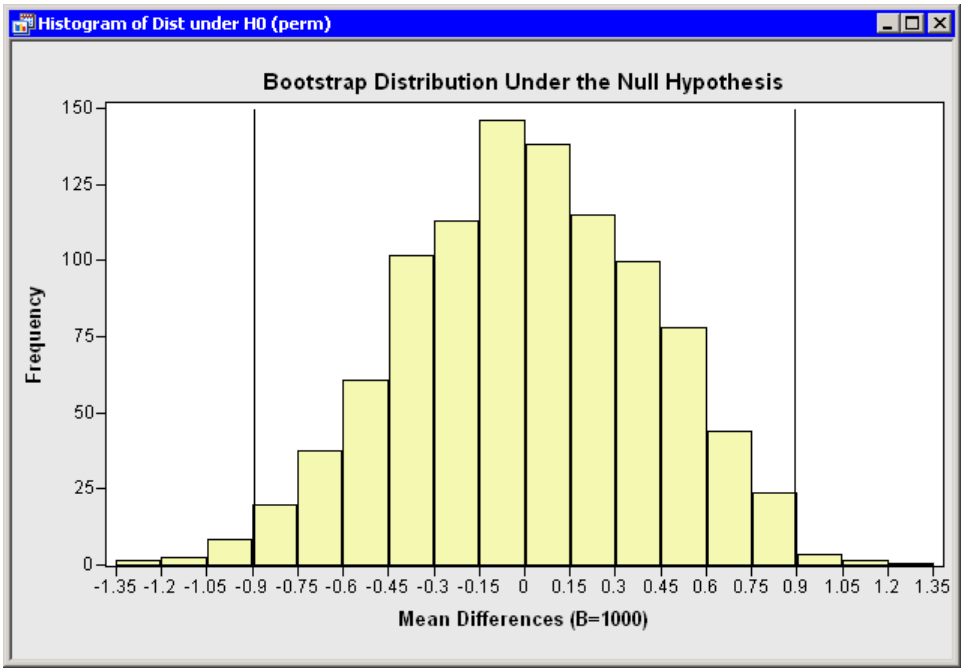
The statistic that is computed for each resample is saved in the vector **s**. That is, **s** contains the bootstrap sampling distribution, assuming the null hypothesis. The question is: how likely is it that the observed statistic (**obsDiff** $= 0.892$ in Figure 7) came from the bootstrap sampling distribution? There are two complementary ways to answer this question: graphically and through p-values. The following statements use the graphics in SAS/IML Studio to plot the bootstrap sampling distribution and to add vertical lines at plus and minus the value of the original statistic:

```
/* graph the bootstrap distribution under the null hypothesis */
declare Histogram hBoot;
hBoot = Histogram.Create("Dist under H0", diffs);

/* Add lines to indicate the observed statistic */
hBoot.DrawUseDataCoordinates();            /* plot lines in data coordinates */
hBoot.GetAxisViewRange(YAXIS, yMin, yMax); /* get min/max of vertical axis */
hBoot.DrawLine(obsDiff, 0, obsDiff, yMax); /* plot line from 0 to max */
hBoot.DrawLine(-obsDiff, 0, -obsDiff, yMax);
```

The graph is shown in Figure 10. The observed statistic is in the tail of the histogram, so it seems unlikely that the observed statistic came from the same sampling distribution as the bootstrap distribution. Notice that it only takes a few statements to create Figure 10. The histogram is created with the Histogram.Create method. The lines are drawn in the same coordinate system as the data. You can query the graph to get information such as the minimum and maximum value displayed on an axis. The values are retrieved into SAS/IML matrices, and then used in the DrawLine method to draw lines of an appropriate height.

Figure 10 Bootstrap Distribution under the Null Hypothesis



You can create a numerical estimate for the probability that the observed statistic came from the bootstrap sampling distribution: simply count how many observations in the bootstrap distribution are more extreme than the observed statistic. Because the alternate hypothesis is two-sided, you need to count observations in both tails, as shown in the following statements:

```
/* compute empirical two-sided p-value */
pval = sum(s>  abs(obsDiff)) / B +
       sum(s< -abs(obsDiff)) / B;
print pval;
```

The empirical p-value is shown in Figure 11. The p-value is small and not very different from the p-value that is computed by the classical matched-pair t test (see Figure 7). At the 95% confidence level you must reject the assumption that there is no difference between the mean delays in 2007 between Aloha Airlines and Hawaiian Airlines. The data suggest that a difference exists.

Figure 11 Small p-value Leads to Rejecting the Null Hypothesis

pval
0.024

In summary, this section demonstrates how the SAS/IML language makes it easy to implement the essential steps of bootstrap methods: generating pseudorandom numbers, resampling from the data, and computing a statistic for each resample. Furthermore, this section reaffirms how useful it is for an analyst to easily locate values within a matrix, extract values from a matrix, and assign values to a subset of a matrix. This section also presents a few statements that show how to create and annotate graphs in SAS/IML Studio.

CALLING R FUNCTIONS

SAS/IML Studio 3.2 (released in July, 2009) introduced an interface that enables you to call functions in the R statistical language from SAS/IML programs. R and SAS/IML software share many similarities. Both are matrix–vector programming languages. Both come with a large library of built-in statistical functions. Both enable users to define new functions that can be shared with colleagues. Both are used by researchers to implement statistical algorithms that appear in professional journals and books.

The following statements show how you can call R from SAS/IML Studio by adding an option to the SUBMIT statement:

```
submit / R;
print("Hello, SAS User!")          # print message from R
endsubmit;
```

Figure 12 Output from R

```
[1] "Hello, SAS User!"
```

If you use the R option in the SUBMIT statement, then all statements prior to the next ENDSUBMIT statement are sent to R for processing.

SAS users usually have their data in SAS data sets or SAS/IML matrices, so an important component of the R interface in SAS/IML Studio is the capability to exchange data between SAS software and R. The two main subroutines used to transfer data into R are the ExportDataToR subroutine and the ExportMatrixToR subroutine. For example, suppose you want to call an R function on the airline delays in the mvts data set. You can run the following statements:

```
run ExportDataSetToR("sgf10.mvts", "Airlines");        /* create an R data frame */

RCarrier = {EV AA NW OH MQ UA US YV AS XE OO CO B6 PE FL DL F9 WN HA AQ};
run ExportMatrixToR(RCarrier, "carrier");              /* copy values to an R matrix  */
```

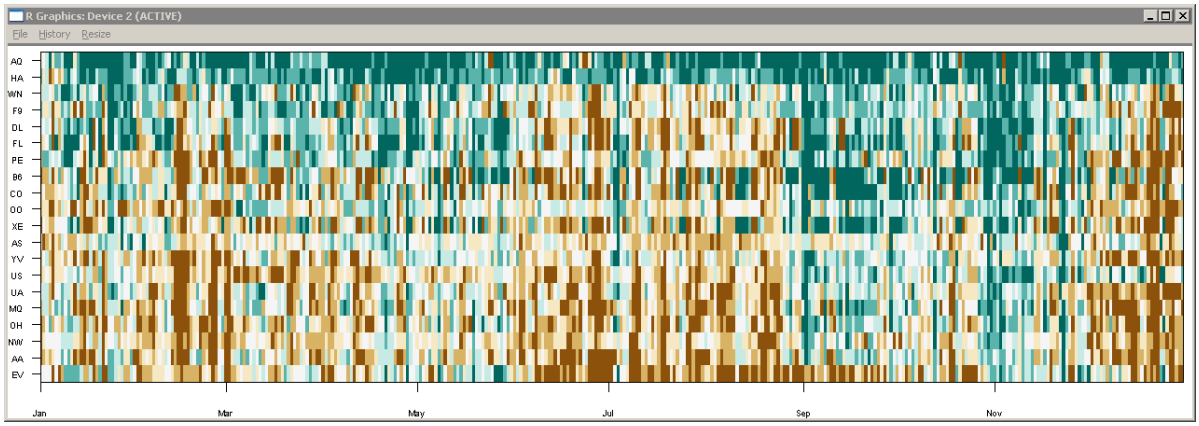
The statements create a data frame in R called Airlines, which contains a copy of the mvts data. The statements also create a 1×20 character matrix in R called **carrier**, which contains the names of the variables in a particular order.

After you have transferred the data, you can call any R function, even graphical functions. For example, suppose you want to visualize the airline delays by using the `mvtsplot` function described in Peng (2008). After downloading Peng's R code, you can load Peng's function definitions and call the `mvtsplot` function, as shown in the following statements:

```
submit / R;
source("mvtsplot.R")               # run R code that defines the mvtsplot function
mvtsplot(Airlines[,carrier],
         xtime=Airlines[,"Date"],
         norm="global", levels=7, palette="BrBG", margin=FALSE)
endsubmit;
```

The result of calling the `mvtsplot` function is an R graphic, shown in Figure 13.

Figure 13 An R Graph That Visualizes the Airline Data



CALLING AN R PACKAGE

Research articles in computational statistics (and other branches of applied statistics) are sometimes accompanied by an R *package*. A package is a user-contributed collection of functions that can be downloaded and installed as an add-in to R. (The analogous concept in SAS would be a library of macros or SAS/IML modules.) If you install an R package, you can call functions in the package to assess how an algorithm performs on your own data. This section shows how you can call functions in an R package.

Suppose you are interested in analyzing late flights from the week before Christmas Day in 2007. For each of three airline carriers, choose a simple random sample of 10 flights that originate from each of three airports. Table 2 shows

the number of flights that were more than 15 minutes late. Does the table show evidence of association between the rows (airlines) and columns (airports) for late flights?

Table 2 A Sample of Late Flights

	Airports		
Carriers	**ATL**	**DFW**	**IAH**
Northwest (NW)	6	9	4
US Airways (US)	6	7	1
Mesa Airlines (YV)	7	0	1

You might call the FREQ procedure to run a chi-square test for association. You can also use Fisher's exact test to analyze the data, as shown in the following analysis:

```
data late;
input Carrier $ Airport $ Count @@;
datalines;
NW ATL 6 NW DFW 9 NW IAH 4
US ATL 6 US DFW 7 US IAH 1
YV ATL 7 YV DFW 0 YV IAH 1
;
run;

proc freq data=late;
   weight count;
   table Carrier*Airport / chisq expected;
   exact fisher;
run;
```

The output from PROC FREQ is shown in Figure 14. The output includes a warning that the (asymptotic) chi-square test might not be appropriate because most of the cells in the table have expected counts that are less than five. However, the test statistic and p-value for Fisher's exact test are shown, and this test is often used for analyzing tables that contain cells with small counts.

Figure 14 Fisher's Exact Test

```
                    The FREQ Procedure

          Statistics for Table of Carrier by Airport

Statistic                     DF       Value      Prob
------------------------------------------------------
Chi-Square                     4      8.8444    0.0651
Likelihood Ratio Chi-Square    4     11.4750    0.0217
Mantel-Haenszel Chi-Square     1      4.4667    0.0346
Phi Coefficient                       0.4645
Contingency Coefficient               0.4212
Cramer's V                            0.3284

 WARNING: 56% of the cells have expected counts less
          than 5. Chi-Square may not be a valid test.

          Fisher's Exact Test
          ----------------------------------
          Table Probability (P)    2.042E-04
          Pr <= P                     0.0438
```

But wait! There is an important fact about Table 2 that has not yet been revealed! It turns out that Mesa Airlines does not fly out of the Dallas/Fort Worth International Airport (DFW). The zero in the $(3, 2)$ cell is not a sampling zero (meaning that no late flights were observed), but is instead a *structural zero* (meaning that no flights take place).

How does this affect Fisher's exact test? Recall that Fisher's exact test enumerates all contingency tables with the observed marginal totals, and computes a p-value by counting the number of tables whose probability under the null hypothesis is more extreme than the probability of the observed table. Consequently, the set of all contingency tables that are enumerated in Fisher's exact test is too large: it contains tables for which the $(3, 2)$ cell is nonzero. This means that the p-value that is reported by Fisher's exact test is smaller than the p-value of a test that enumerates only the tables for which the $(3, 2)$ cell is constrained to be zero.

A recent article by West and Hankin (2008) describe a new variant of Fisher's exact test for two-way contingency tables in the case where some of the table entries are constrained to be zero. The authors modify Fisher's exact test by considering only tables that satisfy the structural constraints. The authors also mention that they wrote an R package called **aylmer** which contains a function named `aylmer.test` which implements the test described in the paper.

The `aylmer.test` function requires that the table be input as an $r \times c$ table of counts, where a missing value denotes a structural zero. The following statements reshape the data, transfer them to R, and call the `aylmer.test` function:

```
use late;
read all var {Count};
close late;

m = shape(Count, 3, 3);            /* reshape matrix of counts           */
m[3,2] = .;                        /* missing represents structural zero */
run ExportMatrixToR(m, "m" );      /* export data to R                   */

/* Call R library that computes exact statistics for structural zeros */
submit / R;
library(aylmer)                    # load package
aylmer.test(m)                     # run exact test
b = allboards(m);                  # get an enumeration of all possible tables
endsubmit;
```

The output from the `aylmer.test` function is shown in Figure 15. It shows a large p-value, which indicates that there is no evidence of association between airlines and airports in Table 2, given the fact that Mesa Airlines does not fly out of DFW.

Figure 15 Output from a Constrained Fisher's Exact Test

```
      Aylmer test for count data

data:  m
p-value = 0.5579
alternative hypothesis: two.sided
```

Often it is useful to transfer the results of an R analysis to a SAS data set or a SAS/IML matrix for further analysis or visualization. You can transfer data from an R matrix into a SAS/IML matrix by using the ImportMatrixFromR module. For example, in the previous program the `allboards` function returns the enumerated list of all tables with a zero in the $(3, 2)$ position and with the observed marginal totals. You can retrieve this set of enumerated tables into a SAS/IML matrix and access the ith table by extracting columns $3i - 2$, $3i - 1$, and $3i$, as shown in the following statements:

```
/* get enumeration of tables with given structural zeros and given marginals */
run ImportMatrixFromR(tbl, "b");
numTables = ncol(tbl) / 3;
print "There are " numTables "possible tables with given marginals.";
print "First table: " (tbl[,1:3])[r={NW US YV} c={ATL DFW IAH}];
```

Figure 16 Results Transfered from R

```
                    numTables

          There are        354 possible tables with given marginals.

                                        ATL       DFW       IAH

                First table:  NW          0        13         6
                              US         11         3         0
                              YV          8         .         0
```

CONCLUSION

This paper shows how you can implement modern data-analytic techniques in the SAS/IML language. You can call a rich library of built-in functions and can extend the library by defining your own modules. In addition, you can call any SAS procedure and any R function and import the results into SAS/IML matrices for additional analysis or for displaying the results with the dynamically linked graphics in SAS/IML Studio. These features make SAS/IML Studio a powerful development environment for the modern data analyst. If you have not used SAS/IML software recently, it is time find out what you are missing.

REFERENCES

American Statistical Association (2009), "Data Exposition 2009," http://stat-computing.org/dataexpo/2009/.

Barnett, V. and Lewis, T. (1994), *Outliers in Statistical Data*, Third Edition, New York: John Wiley & Sons.

Bowen, B. and Headley, D. (2008), *Airline Quality Rating 2008*, Technical report, Wichita State University.
URL http://www.aqr.aero/aqrreports/2008aqr.pdf

Cohen, R. (2009), "Applications of the GLMSELECT Procedure for Megamodel Selection," in *Proceedings of the SAS Global Forum 2009 Conference*, Cary, NC: SAS Institute Inc.

Davison, A. C. and Hinkley, D. V. (1997), *Bootstrap Methods and Their Application*, Cambridge, UK: Cambridge University Press.

Efron, B. and Tibshirani, R. (1993), *An Introduction to the Bootstrap*, New York: Chapman & Hall.

Hardin, J. and Rocke, D. M. (2005), "The Distribution of Robust Distances," *Journal of Computational and Graphical Statistics*, 14(4), 928–946.

Hawkins, D. M., Bradu, D., and Kass, G. V. (1984), "Location of Several Outliers in Multiple Regression Data Using Elemental Sets," *Technometrics*, 26, 197–208.

Peng, R. (2008), "A Method for Visualizing Multivariate Time Series Data," *Journal of Statistical Software*, 25(1), 1–17.
URL http://www.jstatsoft.org/v25/c01

Rousseeuw, P. J. (1985), "Multivariate Estimation with High Breakdown Point," in W. Grossmann, G. Pflug, I. Vincze, and W. Wertz, eds., *Mathematical Statistics and Applications*, 283–297, Dordrecht: Reidel Publishing.

West, L. and Hankin, R. (2008), "Exact Tests for Two-Way Contingency Tables with Structural Zeros," *Journal of Statistical Software*, 28(11), 1–19.
URL http://www.jstatsoft.org/v28/i11

Wicklin, R. (2008), "SAS Stat Studio: A Programming Environment for High-End Data Analysts," in *Proceedings of the SAS Global Forum 2008 Conference*, Cary, NC: SAS Institute Inc.

Wicklin, R. (2009), *An Analysis of Airline Delays with SAS/IML Studio*, Technical report, SAS Institute Inc.
URL http://support.sas.com/rnd/app/papers/papers_iml.html

Wicklin, R. (2010), *Statistical Programming with SAS/IML Software: An Introduction*, Cary, NC: SAS Press, forthcoming.

APPENDIX

The VAR function is available in SAS/IML 9.22. If you are running a version of SAS/IML software prior to 9.22, define the following module before running the examples in this paper:

```
start Var(x);
   mean = x[:,];                  /* mean of each column                    */
   c = x-mean;                    /* center the data                        */
   countn = n(x)[+,];             /* count nonmissing values in each column */
   var = c[##,] / (countn-1);     /* variance of each column                */
   return (var);
finish;
```

CONTACT INFORMATION

Rick Wicklin
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513