Chapter 9

General Statistics Examples

Contents

Overview

SAS/IML software has many linear operators that perform high-level operations commonly needed in applying linear algebra techniques to data analysis. The similarity of the Interactive Matrix Language notation and matrix algebra notation makes translation from algorithm to program a straightforward task. The examples in this chapter show a variety of matrix operators at work.

You can use these examples to gain insight into the more complex problems you might need to solve. Some of the examples perform the same analyses as performed by procedures in SAS/STAT software and are not meant to replace them. The examples are included as learning tools.

General Statistics Examples

Example 9.1: Correlation

The following statements show how you can define modules to compute correlation coefficients between numeric variables and standardized values for a set of data. For more efficient computations, use the built-in CORR function and the STD function.

```
proc iml;
   /* Module to compute correlations  */
start corr;
   n = nrow(x);                      /* number of observations */
   sum = x[+,] ;                        /* compute column sums */
   xpx = t(x)*x-t(sum)*sum/n;         /* compute sscp matrix   */
   s = diag(1/sqrt(vecdiag(xpx)));           /* scaling matrix */
   corr = s*xpx*s;                       /* correlation matrix */
   print "Correlation Matrix",,corr[rowname=nm colname=nm] ;
finish corr;

   /* Module to standardize data */
start std;
   mean = x[+,] /n;                           /* means for columns */
   x = x-repeat(mean,n,1);             /* center x to mean zero */
   ss = x[##,] ;                   /* sum of squares for columns */
   std = sqrt(ss/(n-1));           /* standard deviation estimate*/
   x = x*diag(1/std);                    /* scaling to std dev 1 */
   print ,"Standardized Data",,X[colname=nm] ;
finish std;

   /* Sample run */
x = { 1 2 3,
      3 2 1,
      4 2 1,
      0 4 1,
     24 1 0,
      1 3 8};
nm={age weight height};
run corr;
run std;
```

The results are shown in Output 9.1.1.

Output 9.1.1 Correlation Coefficients and Standardized Values

```
                    Correlation Matrix

                           corr
                      AGE     WEIGHT     HEIGHT

         AGE             1 -0.717102 -0.436558
         WEIGHT -0.717102          1 0.3508232
         HEIGHT -0.436558 0.3508232          1

                    Standardized Data

                            x
                  AGE     WEIGHT     HEIGHT

            -0.490116 -0.322749 0.2264554
            -0.272287 -0.322749 -0.452911
            -0.163372 -0.322749 -0.452911
             -0.59903 1.6137431 -0.452911
            2.0149206 -1.290994 -0.792594
            -0.490116 0.6454972  1.924871
```

Example 9.2: Newton's Method for Solving Nonlinear Systems of Equations

This example solves a nonlinear system of equations by Newton's method. Let the nonlinear system be represented by

$$F(\mathbf{x}) = 0$$

where $\mathbf{x}$ is a vector and F is a vector-valued, possibly nonlinear function.

In order to find $\mathbf{x}$ such that F goes to 0, an initial estimate $\mathbf{x}_0$ is chosen, and Newton's iterative method for converging to the solution is used:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}^{-1}(\mathbf{x}_n)F(\mathbf{x}_n)$$

where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix of partial derivatives of F with respect to $\mathbf{x}$. (For more efficient computations, use the built-in NLPNRA subroutine.)

For optimization problems, the same method is used, where $F(\mathbf{x})$ is the gradient of the objective function and $\mathbf{J}(\mathbf{x})$ becomes the Hessian (Newton-Raphson).

In this example, the system to be solved is

$$\begin{aligned} \mathbf{x}_1 + \mathbf{x}_2 - \mathbf{x}_1\mathbf{x}_2 + 2 &= 0 \\ \mathbf{x}_1 \exp(-\mathbf{x}_2) - 1 &= 0 \end{aligned}$$

The following statements are organized into three modules: NEWTON, FUN, and DERIV.

```
   /*     Newton's Method to Solve a Nonlinear Function      */
   /* The user must supply initial values,                   */
   /* and the FUN and DERIV functions.                       */
   /* On entry: FUN evaluates the function f in terms of x   */
   /* initial values are given to x                          */
   /* DERIV evaluates jacobian j                             */
   /* tuning variables: CONVERGE, MAXITER.                   */
   /* On exit: solution in x, function value in f close to 0 */
   /* ITER has number of iterations.                         */
proc iml;
start newton;
   run fun;          /* evaluate function at starting values */
   do iter = 1 to maxiter           /* iterate until maxiter */
   while(max(abs(f))>converge); /* iterations or convergence */
      run deriv;              /* evaluate derivatives in j */
      delta = -solve(j,f);    /* solve for correction vector */
      x = x+delta;                /* the new approximation */
      run fun;                    /* evaluate the function */
   end;
finish newton;

maxiter = 15;                  /* default maximum iterations */
converge = .000001;         /* default convergence criterion */

   /* User-supplied function evaluation */
start fun;
   x1 = x[1] ;
   x2 = x[2] ;                           /* extract the values */
   f = (x1+x2-x1*x2+2) //
       (x1*exp(-x2)-1);             /* evaluate the function */
finish fun;

   /* User-supplied derivatives of the function */
start deriv;
   /* evaluate jacobian */
   j = ((1-x2)||(1-x1) ) // (exp(-x2)||(-x1*exp(-x2)));
finish deriv;

do;
   print "Solving the system: X1+X2-X1*X2+2=0, X1*EXP(-X2)-1=0" ,;
   x={.1, -2};     /* starting values */
   run newton;
   print x f;
end;
```

The results are shown in Output 9.2.1.

Output 9.2.1 Newton's Method: Results

```
              Solving the system: X1+X2-X1*X2+2=0, X1*EXP(-X2)-1=0


                                     x         f

                             0.0977731 5.3523E-9
                             -2.325106 6.1501E-8
```

Example 9.3: Regression

This example shows a regression module that calculates statistics that are associated with a linear regression.

```
   /*         Regression Routine                */
   /* Given X and Y, this fits Y = X B + E     */
   /* by least squares.                         */
proc iml;
start reg;
   n = nrow(x);                    /* number of observations */
   k = ncol(x);                       /* number of variables */
   xpx = x`*x;                              /* crossproducts */
   xpy = x`*y;
   xpxi = inv(xpx);                 /* inverse crossproducts */
   b = xpxi*xpy;                      /* parameter estimates */
   yhat = x*b;                           /* predicted values */
   resid = y-yhat;                              /* residuals */
   sse = resid`*resid;              /* sum of squared errors */
   dfe = n-k;                     /* degrees of freedom error */
   mse = sse/dfe;                      /* mean squared error */
   rmse = sqrt(mse);              /* root mean squared error */
   covb = xpxi#mse;               /* covariance of estimates */
   stdb = sqrt(vecdiag(covb));            /* standard errors */
   t = b/stdb;                      /* ttest for estimates=0 */
   probt = 1-probf(t#t,1,dfe);   /* significance probability */
   print name b stdb t probt;
   s = diag(1/stdb);
   corrb = s*covb*s;             /* correlation of estimates */
   print ,"Covariance of Estimates", covb[r=name c=name] ,
          "Correlation of Estimates",corrb[r=name c=name] ;

   if nrow(tval)=0 then return;    /* is a t value specified? */
   projx = x*xpxi*x`;                          /* hat matrix */
   vresid = (i(n)-projx)*mse;     /* covariance of residuals */
   vpred = projx#mse;      /* covariance of predicted values  */
   h = vecdiag(projx);                /* hat leverage values */
   lowerm = yhat-tval#sqrt(h*mse); /* low. conf lim for mean */
   upperm = yhat+tval#sqrt(h*mse);    /* upper lim. for mean */
```

```
      lower = yhat-tval#sqrt(h*mse+mse); /* lower lim. for indiv*/
      upper = yhat+tval#sqrt(h*mse+mse);/* upper lim. for indiv */
      print ,,"Predicted Values, Residuals, and Limits" ,,
      y yhat resid h lowerm upperm lower upper;
   finish reg;

      /* Routine to test a linear combination of the estimates  */
      /* given L, this routine tests hypothesis that LB = 0.    */

   start test;
      dfn=nrow(L);
      Lb=L*b;
      vLb=L*xpxi*L`;
      q=Lb`*inv(vLb)*Lb /dfn;
      f=q/mse;
      prob=1-probf(f,dfn,dfe);
      print ,f dfn dfe prob;
   finish test;

     /* Run it on population of U.S. for decades beginning 1790 */

   x= { 1 1 1,
        1 2 4,
        1 3 9,
        1 4 16,
        1 5 25,
        1 6 36,
        1 7 49,
        1 8 64 };

   y= {3.929,5.308,7.239,9.638,12.866,17.069,23.191,31.443};
   name={"Intercept", "Decade", "Decade**2" };
   tval=2.57;  /* for 5 df at 0.025 level to get 95% conf. int. */
   reset fw=7;
   run reg;
   do;
      print ,"TEST Coef for Linear";
      L={0 1 0 };
      run test;
      print ,"TEST Coef for Linear,Quad";
      L={0 1 0,0 0 1};
      run test;
      print ,"TEST Linear+Quad = 0";
      L={0 1 1 };
      run test;
   end;
```

The results are shown in Output 9.3.1.

Output 9.3.1 Regression Results

```
                    name            b     stdb       t   probt

                    Intercept 5.06934 0.96559 5.24997 0.00333
                    Decade    -1.1099  0.4923 -2.2546 0.07385
                    Decade**2 0.53964  0.0534  10.106 0.00016

                             Covariance of Estimates

                                      covb
                                Intercept  Decade Decade**2

                      Intercept   0.93237 -0.4362   0.04277
                      Decade      -0.4362 0.24236   -0.0257
                      Decade**2   0.04277 -0.0257   0.00285

                             Correlation of Estimates

                                      corrb
                                Intercept  Decade Decade**2

                      Intercept         1 -0.9177    0.8295
                      Decade      -0.9177       1   -0.9762
                      Decade**2    0.8295 -0.9762         1

                     Predicted Values, Residuals, and Limits

          y    yhat   resid       h  lowerm  upperm   lower   upper

      3.929 4.49904   -0.57 0.70833 3.00202 5.99606 2.17419 6.82389
      5.308 5.00802 0.29998 0.27976 4.06721 5.94883 2.99581 7.02023
      7.239 6.59627 0.64273 0.23214 5.73926 7.45328 4.62185 8.57069
      9.638 9.26379 0.37421 0.27976 8.32298 10.2046 7.25158  11.276
     12.866 13.0106 -0.1446 0.27976 12.0698 13.9514 10.9984 15.0228
     17.069 17.8367 -0.7677 0.23214 16.9797 18.6937 15.8622 19.8111
     23.191  23.742  -0.551 0.27976 22.8012 24.6828 21.7298 25.7542
     31.443 30.7266 0.71638 0.70833 29.2296 32.2236 28.4018 33.0515

                               TEST Coef for Linear

                              f     dfn     dfe    prob

                        5.08317       1       5 0.07385

                            TEST Coef for Linear,Quad

                              f     dfn     dfe    prob

                        666.511       2       5 8.54E-7

                               TEST Linear+Quad = 0
```

Output 9.3.1 *continued*

```
                                   f      dfn      dfe     prob

                             1.67746        1        5 0.25184
```

Example 9.4: Alpha Factor Analysis

This example shows how an algorithm for computing alpha factor patterns (Kaiser and Caffrey 1965) is implemented in the SAS/IML language.

You can store the following ALPHA subroutine in a catalog and load it when needed.

```
/*                Alpha Factor Analysis                      */
/*  Ref: Kaiser et al., 1965 Psychometrika, pp. 12-13        */
/*  r correlation matrix (n.s.) already set up               */
/*  p number of variables                                    */
/*  q number of factors                                      */
/*  h communalities                                          */
/*  m eigenvalues                                            */
/*  e eigenvectors                                           */
/*  f factor pattern                                         */
/*  (IQ,H2,HI,G,MM) temporary use. freed up                  */
/*                                                           */
proc iml;
start alpha;
   p = ncol(r);
   q = 0;
   h = 0;                                      /* initialize */
   h2 = i(p)-diag(1/vecdiag(inv(r)));                /* smcs */
   do while(max(abs(h-h2))>.001); /* iterate until converges */
      h = h2;
      hi = diag(sqrt(1/vecdiag(h)));
      g = hi*(r-i(p))*hi+i(p);
      call eigen(m,e,g);         /* get eigenvalues and vecs */
      if q=0 then do;
         q = sum(m>1);                  /* number of factors */
         iq = 1:q;
      end;                                   /* index vector */
      mm = diag(sqrt(m[iq,]));           /* collapse eigvals */
      e = e[,iq] ;                       /* collapse eigvecs */
      h2 = h*diag((e*mm) [,##]);        /* new communalities */
   end;
   hi = sqrt(h);
   h = vecdiag(h2);
   f = hi*e*mm;                         /* resulting pattern */
   free iq h2 hi g mm;                   /* free temporaries */
finish;

  /* Correlation Matrix from Harmon, Modern Factor Analysis, */
  /* Second edition, page 124, "Eight Physical Variables"    */

r={1.000  .846 .805 .859 .473 .398 .301 .382 ,
    .846 1.000 .881 .826 .376 .326 .277 .415 ,
    .805 .881 1.000 .801 .380 .319 .237 .345 ,
    .859 .826 .801 1.000 .436 .329 .327 .365 ,
    .473 .376 .380 .436 1.000 .762 .730 .629 ,
    .398 .326 .319 .329 .762 1.000 .583 .577 ,
    .301 .277 .237 .327 .730 .583 1.000 .539 ,
    .382 .415 .345 .365 .629 .577 .539 1.000};
nm = {Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8};
run alpha;
print ,"EIGENVALUES" , m;
print ,"COMMUNALITIES" , h[rowname=nm];
print ,"FACTOR PATTERN", f[rowname=nm];
```

The results are shown in Output 9.4.1.

Output 9.4.1 Alpha Factor Analysis: Results

```
 EIGENVALUES

      m

  5.937855
 2.0621956
 0.1390178
 0.0821054
  0.018097
 -0.047487
  -0.09148
 -0.100304

COMMUNALITIES

      h

VAR1 0.8381205
VAR2 0.8905717
VAR3   0.81893
VAR4 0.8067292
VAR5 0.8802149
VAR6 0.6391977
VAR7 0.5821583
VAR8 0.4998126

FACTOR PATTERN
```

Output 9.4.1 *continued*

```
                             f

              VAR1  0.813386 -0.420147
              VAR2 0.8028363  -0.49601
              VAR3 0.7579087 -0.494474
              VAR4 0.7874461 -0.432039
              VAR5 0.8051439 0.4816205
              VAR6 0.6804127 0.4198051
              VAR7  0.620623 0.4438303
              VAR8 0.6449419 0.2895902
```

Example 9.5: Categorical Linear Models

This example fits a linear model to a function of the response probabilities

$$\mathbf{K} \log \pi = \mathbf{X}_{\text{¸}} + e$$

where $\mathbf{K}$ is a matrix that compares each response category to the last. Data are from Kastenbaum and Lamphiear (1959). First, the Grizzle-Starmer-Koch (1969) approach is used to obtain generalized least squares estimates of ¸. These form the initial values for the Newton-Raphson solution for the maximum likelihood estimates. The CATMOD procedure can also be used to analyze these binary data (see Cox (1970)). Here is the program.

```
   /* Categorical Linear Models                         */
   /* by Least Squares and Maximum Likelihood           */
   /*  CATLIN                                           */
   /*  Input:                                           */
   /*     n the s by p matrix of response counts        */
   /*     x the s by r design matrix                    */

proc iml ;
start catlin;

   /*---find dimensions---*/
   s = nrow(n);                       /* number of populations */
   r = ncol(n);                         /* number of responses */
   q = r-1;                        /* number of function values */
   d = ncol(x);                  /* number of design parameters */
   qd = q*d;                      /* total number of parameters */

   /*---get probability estimates---*/
   rown = n[,+];                                   /* row totals */
   pr = n/(rown*repeat(1,1,r));       /* probability estimates */
   p = shape(pr[,1:q] ,0,1);      /* cut and shaped to vector */
   print "INITIAL PROBABILITY ESTIMATES" ,pr;

      /*   estimate by the GSK method   */

      /* function of probabilities */
   f = log(p)-log(pr[,r])@repeat(1,q,1);

      /* inverse covariance of f */
   si = (diag(p)-p*p`)#(diag(rown)@repeat(1,q,q));
   z = x@i(q);                    /* expanded design matrix */
   h = z`*si*z;                     /* crossproducts matrix */
   g = z`*si*f;                             /* cross with f */
   beta = solve(h,g);             /* least squares solution */
   stderr = sqrt(vecdiag(inv(h)));        /* standard errors */
   run prob;
   print ,"GSK ESTIMATES" , beta stderr ,pi;

      /*   iterations for ML solution   */
   crit = 1;
   do it = 1 to 8 while(crit>.0005);/* iterate until converge*/

      /* block diagonal weighting  */
      si = (diag(pi)-pi*pi`)#(diag(rown)@repeat(1,q,q));
      g = z`*(rown@repeat(1,q,1)#(p-pi));        /* gradient */
      h = z`*si*z;                                /* hessian */
      delta = solve(h,g);            /* solve for correction */
      beta = beta+delta;             /* apply the correction */
      run prob;                     /* compute prob estimates */
      crit = max(abs(delta));       /* convergence criterion */
   end;
   stderr = sqrt(vecdiag(inv(h)));        /* standard errors */
   print , "ML Estimates", beta stderr, pi;
   print , "Iterations" it "Criterion" crit;
finish catlin;

   /*   subroutine to compute new prob estimates @ parameters  */
start prob;
   la = exp(x*shape(beta,0,q));
   pi = la/((1+la[,+] )*repeat(1,1,q));
   pi = shape(pi,0,1);
finish prob;

   /*---prepare frequency data and design matrix---*/
n= { 58 11 05,
     75 19 07,
     49 14 10,
     58 17 08,
     33 18 15,
     45 22 10,
     15 13 15,
     39 22 18,
     04 12 17,
     05 15 08};     /* frequency counts*/

x= { 1 1 1 0 0 0,
     1 -1 1 0 0 0,
     1 1 0 1 0 0,
     1 -1 0 1 0 0,
     1 1 0 0 1 0,
     1 -1 0 0 1 0,
     1 1 0 0 0 1,
     1 -1 0 0 0 1,
     1 1 -1 -1 -1 -1,
     1 -1 -1 -1 -1 -1};    /* design matrix*/

run catlin;
```

The maximum likelihood estimates are shown in Output 9.5.1.

Output 9.5.1 Maximum Likelihood Estimates

```
                  INITIAL PROBABILITY ESTIMATES

                               pr

                  0.7837838 0.1486486 0.0675676
                  0.7425743 0.1881188 0.0693069
                  0.6712329 0.1917808 0.1369863
                  0.6987952 0.2048193 0.0963855
                        0.5 0.2727273 0.2272727
                  0.5844156 0.2857143 0.1298701
                  0.3488372 0.3023256 0.3488372
                  0.4936709  0.278481 0.2278481
                  0.1212121 0.3636364 0.5151515
                  0.1785714 0.5357143 0.2857143

                         GSK ESTIMATES

                             beta    stderr

                       0.9454429 0.1290925
                       0.4003259 0.1284867
                       -0.277777 0.1164699
                       -0.278472 0.1255916
                       1.4146936  0.267351
                        0.474136  0.294943
                       0.8464701 0.2362639
                       0.1526095 0.2633051
                       0.1952395 0.2214436
                       0.0723489 0.2366597
                       -0.514488 0.2171995
                       -0.400831 0.2285779
```

Output 9.5.1 *continued*

```
                              pi

                          0.7402867
                          0.1674472
                          0.7704057
                          0.1745023
                          0.6624811
                          0.1917744
                          0.7061615
                          0.2047033
                           0.516981
                          0.2648871
                          0.5697446
                          0.2923278
                          0.3988695
                          0.2589096
                          0.4667924
                          0.3034204
                          0.1320359
                          0.3958019
                          0.1651907
                          0.4958784

                         ML Estimates

                          beta    stderr

                     0.9533597 0.1286179
                     0.4069338 0.1284592
                     -0.279081 0.1156222
                     -0.280699 0.1252816
                     1.4423195 0.2669357
                     0.4993123 0.2943437
                     0.8411595 0.2363089
                     0.1485875 0.2635159
                     0.1883383 0.2202755
                     0.0667313  0.236031
                     -0.527163  0.216581
                     -0.414965 0.2299618
```

Output 9.5.1 *continued*

```
                                    pi

                                 0.7431759
                                 0.1673155
                                 0.7723266
                                 0.1744421
                                 0.6627266
                                 0.1916645
                                 0.7062766
                                 0.2049216
                                 0.5170782
                                 0.2646857
                                 0.5697771
                                  0.292607
                                 0.3984205
                                 0.2576653
                                 0.4666825
                                 0.3027898
                                 0.1323243
                                 0.3963114
                                  0.165475
                                 0.4972044

                                    it                 crit

                      Iterations         3 Criterion 0.0004092
```

Example 9.6: Regression of Subsets of Variables

This example performs regression with variable selection. Some of the methods used in this example are also used in the REG procedure. Here is the program.

```
proc iml;

 /*-------Initialization-------------------------------*
 | c,csave the crossproducts matrix                    |
 | n       number of observations                      |
 | k       total number of variables to consider       |
 | l       number of variables currently in model      |
 | in      a 0-1 vector of whether variable is in      |
 | b       print collects results (L MSE RSQ BETAS )   |
 *----------------------------------------------------*/
start initial;
  n=nrow(x); k=ncol(x); k1=k+1; ik=1:k;
  bnames={nparm mse rsquare} ||varnames;

  /*---correct by mean, adjust out intercept parameter---*/
  y=y-y[+,]/n;                    /* correct y by mean */
  x=x-repeat(x[+,]/n,n,1);        /* correct x by mean */
  xpy=x`*y;                       /* crossproducts     */
  ypy=y`*y;
  xpx=x`*x;
  free x y;                      /* no longer need the data*/
  csave=(xpx || xpy) //
        (xpy`|| ypy);            /* save copy of crossproducts*/
finish;


 /*-----forward method-----------------------------------------*/
start forward;
  print  "FORWARD SELECTION METHOD";
  free bprint;
  c=csave; in=repeat(0,k,1); L=0;      /* no variables are in */
  dfe=n-1; mse=ypy/dfe;
  sprob=0;

  do while(sprob<.15 & l<k);
     indx=loc(^in);           /* where are the variables not in?*/
     cd=vecdiag(c)[indx,];    /* xpx diagonals                 */
     cb=c[indx,k1];           /* adjusted xpy                  */
     tsqr=cb#cb/(cd#mse);     /* squares of t tests            */
     imax=tsqr[<:>,];         /* location of maximum in indx   */
     sprob=(1-probt(sqrt(tsqr[imax,]),dfe))*2;
     if sprob<.15 then do;    /* if t-test significant         */
        ii=indx[,imax];       /* pick most significant         */
        run swp;              /* routine to sweep              */
        run bpr;              /* routine to collect results    */
     end;
  end;

  print  bprint[colname=bnames] ;
  finish;



 /*-----backward method----------------------------------------*/
start backward;
  print  "BACKWARD ELIMINATION ";
  free bprint;
  c=csave; in=repeat(0,k,1);
  ii=1:k; run swp; run bpr;       /* start with all variables in*/
  sprob=1;

  do while(sprob>.15 & L>0);
     indx=loc(in);                /* where are the variables in? */
     cd=vecdiag(c)[indx,];        /* xpx diagonals               */
     cb=c[indx,k1];               /* bvalues                     */
     tsqr=cb#cb/(cd#mse);         /* squares of t tests          */
     imin=tsqr[>:<,];             /* location of minimum in indx */
     sprob=(1-probt(sqrt(tsqr[imin,]),dfe))*2;
     if sprob>.15 then do;        /* if t-test nonsignificant    */
        ii=indx[,imin];           /* pick least significant      */
        run swp;                  /* routine to sweep in variable*/
        run bpr;                  /* routine to collect results  */
```

```
      end;
   end;

   print  bprint[colname=bnames] ;
   finish;



 /*-----stepwise method----------------------------------------*/
start stepwise;
   print "STEPWISE METHOD";
   free bprint;
   c=csave; in=repeat(0,k,1); L=0;
   dfe=n-1; mse=ypy/dfe;
   sprob=0;

   do while(sprob<.15 & L<k);
      indx=loc(^in);           /* where are the variables not in?*/
      nindx=loc(in);           /* where are the variables in?    */
      cd=vecdiag(c)[indx,];    /* xpx diagonals                 */
      cb=c[indx,k1];           /* adjusted xpy                  */
      tsqr=cb#cb/cd/mse;       /* squares of t tests            */
      imax=tsqr[<:>,];         /* location of maximum in indx   */
      sprob=(1-probt(sqrt(tsqr[imax,]),dfe))*2;
      if sprob<.15 then do;    /* if t-test significant         */
         ii=indx[,imax];       /* find index into c             */
         run swp;              /* routine to sweep              */
         run backstep;         /* check if remove any terms     */
         run bpr;              /* routine to collect results    */
      end;
   end;

   print  bprint[colname=bnames] ;
   finish;


 /*----routine to backwards-eliminate for stepwise--*/
start backstep;
   if nrow(nindx)=0 then return;
   bprob=1;
   do while(bprob>.15 & L<k);
      cd=vecdiag(c)[nindx,];      /* xpx diagonals          */
      cb=c[nindx,k1];             /* bvalues                */
      tsqr=cb#cb/(cd#mse);        /* squares of t tests     */
      imin=tsqr[>:<,];            /* location of minimum in nindx*/
      bprob=(1-probt(sqrt(tsqr[imin,]),dfe))*2;
      if bprob>.15 then do;
         ii=nindx[,imin];
         run swp;
         run bpr;
      end;
   end;
finish;
```

```
 /*-----search all possible models----------------------------*/
start all;
     /*---use method of schatzoff et al. for search technique--*/
  betak=repeat(0,k,k); /* record estimates for best l-param model*/
  msek=repeat(1e50,k,1);/* record best mse per # parms           */
  rsqk=repeat(0,k,1);  /* record best rsquare                   */
  ink=repeat(0,k,k);   /* record best set per # parms           */
  limit=2##k-1;        /* number of models to examine           */

c=csave; in=repeat(0,k,1);/* start out with no variables in model*/

     do kk=1 to limit;
        run ztrail;              /* find which one to sweep    */
        run swp;                 /* sweep it in                */
        bb=bb//(L||mse||rsq||(c[ik,k1]#in)`);
        if mse<msek[L,] then do; /* was this best for L parms? */
           msek[L,]=mse;         /* record mse                 */
           rsqk[L,]=rsq;         /* record rsquare             */
           ink[,L]=in;           /* record which parms in model*/
           betak[L,]=(c[ik,k1]#in)`;/* record estimates        */
        end;
     end;

     print "ALL POSSIBLE MODELS IN SEARCH ORDER";
     print bb[colname=bnames]; free bb;

     bprint=ik`||msek||rsqk||betak;
     print "THE BEST MODEL FOR EACH NUMBER OF PARAMETERS";
     print bprint[colname=bnames];
finish;


 /*-subroutine to find number of trailing zeros in binary number*/
 /* on entry: kk is the number to examine                       */
 /* on exit:  ii has the result                                 */
 /*-------------------------------------------------------------*/
start ztrail;
    ii=1; zz=kk;
    do while(mod(zz,2)=0); ii=ii+1; zz=zz/2; end;
finish;


 /*-----subroutine to sweep in a pivot-------------------------*/
 /* on entry: ii has the position(s) to pivot                   */
 /* on exit:  in, L, dfe, mse, rsq recalculated                 */
 /*-------------------------------------------------------------*/
start swp;
     if abs(c[ii,ii])<1e-9 then do; print  "failure", c;stop;end;
     c=sweep(c,ii);
     in[ii,]=^in[ii,];
     L=sum(in); dfe=n-1-L;
     sse=c[k1,k1];
     mse=sse/dfe;
     rsq=1-sse/ypy;
finish;


 /*-----subroutine to collect bprint results--------------------*/
 /* on entry: L,mse,rsq, and c set up to collect                */
 /* on exit:  bprint has another row                            */
 /*-------------------------------------------------------------*/
start bpr;
   bprint=bprint//(L||mse||rsq||(c[ik,k1]#in)`);
finish;



 /*--------------stepwise methods---------------------*/
 /* after a call to the initial routine, which sets up*/
 /* the data, four different routines can be called   */
 /* to do four different model-selection methods.     */
 /*---------------------------------------------------*/
start seq;
  run initial;          /* initialization             */
  run all;              /* all possible models        */
  run forward;          /* foreward selection method  */
  run backward;         /* backward elimination method*/
  run stepwise;         /* stepwise method            */
finish;


 /*------------------------data on physical fitness--------------*
| These measurements were made on men involved in a physical     |
| fitness course at N.C.State Univ.  The variables are age(years)|
| weight(kg), oxygen uptake rate(ml per kg body weight per       |
| minute), time to run 1.5 miles(minutes), heart rate while      |
| resting, heart rate while running (same time oxygen rate       |
| measured), and maximum heart rate recorded while running.      |
| Certain values of maxpulse were modified for consistency.      |
|   Data courtesy DR. A.C. Linnerud                              |
*---------------------------------------------------------------*/
data =
   { 44 89.47  44.609 11.37 62 178 182  ,
     40 75.07  45.313 10.07 62 185 185  ,
     44 85.84  54.297  8.65 45 156 168  ,
     42 68.15  59.571  8.17 40 166 172  ,
     38 89.02  49.874  9.22 55 178 180  ,
     47 77.45  44.811 11.63 58 176 176  ,
     40 75.98  45.681 11.95 70 176 180  ,
     43 81.19  49.091 10.85 64 162 170  ,
     44 81.42  39.442 13.08 63 174 176  ,
     38 81.87  60.055  8.63 48 170 186  ,
     44 73.03  50.541 10.13 45 168 168  ,
     45 87.66  37.388 14.03 56 186 192  ,
     45 66.45  44.754 11.12 51 176 176  ,
     47 79.15  47.273 10.60 47 162 164  ,
     54 83.12  51.855 10.33 50 166 170  ,
     49 81.42  49.156  8.95 44 180 185  ,
     51 69.63  40.836 10.95 57 168 172  ,
     51 77.91  46.672 10.00 48 162 168  ,
     48 91.63  46.774 10.25 48 162 164  ,
     49 73.37  50.388 10.08 67 168 168  ,
     57 73.37  39.407 12.63 58 174 176  ,
     54 79.38  46.080 11.17 62 156 165  ,
     52 76.32  45.441  9.63 48 164 166  ,
     50 70.87  54.625  8.92 48 146 155  ,
     51 67.25  45.118 11.08 48 172 172  ,
     54 91.63  39.203 12.88 44 168 172  ,
     51 73.71  45.790 10.47 59 186 188  ,
     57 59.08  50.545  9.93 49 148 155  ,
     49 76.32  48.673  9.40 56 186 188  ,
     48 61.24  47.920 11.50 52 170 176  ,
     52 82.78  47.467 10.50 53 170 172  };

x=data[,{1 2 4 5 6 7 }];
y=data[,3];
free data;
varnames={age weight runtime rstpuls runpuls maxpuls};
reset fw=6 linesize=87;
run seq;
```

The results are shown in Output 9.6.1.

Output 9.6.1 Model Selection: Results

```
                      ALL POSSIBLE MODELS IN SEARCH ORDER


                                         bb
      NPARM    MSE RSQUARE    AGE WEIGHT RUNTIME RSTPULS RUNPULS MAXPULS

          1 26.634  0.0928 -0.311      0       0       0       0       0
          2 25.826  0.1506  -0.37 -0.158       0       0       0       0
          1  28.58  0.0265      0 -0.104       0       0       0       0
          2 7.7556  0.7449      0 -0.025  -3.289       0       0       0
          3 7.2263  0.7708 -0.174 -0.054   -3.14       0       0       0
          2 7.1684  0.7642  -0.15      0  -3.204       0       0       0
          1 7.5338  0.7434      0      0  -3.311       0       0       0
          2 7.7983  0.7435      0      0  -3.287   -0.01       0       0
          3 7.3361  0.7673 -0.168      0  -3.079  -0.045       0       0
          4 7.3666   0.775 -0.196 -0.059  -2.989  -0.053       0       0
          3 8.0373  0.7451      0 -0.026  -3.263   -0.01       0       0
          2 24.915  0.1806      0 -0.093       0  -0.275       0       0
          3  20.28  0.3568 -0.447 -0.156       0  -0.322       0       0
          2 21.276  0.3003 -0.389      0       0  -0.323       0       0
          1 24.676  0.1595      0      0       0  -0.279       0       0

                             ...<rows skipped>...


                   THE BEST MODEL FOR EACH NUMBER OF PARAMETERS
```

Output 9.6.1 *continued*

```
                                  bprint
      NPARM     MSE RSQUARE    AGE WEIGHT RUNTIME RSTPULS RUNPULS MAXPULS

          1 7.5338  0.7434       0      0  -3.311       0       0       0
          2 7.1684  0.7642  -0.15      0  -3.204       0       0       0
          3 5.9567  0.8111 -0.256      0  -2.825       0  -0.131       0
          4 5.3435  0.8368 -0.198      0  -2.768       0  -0.348  0.2705
          5 5.1763   0.848  -0.22 -0.072  -2.683       0  -0.373  0.3049
          6 5.3682  0.8487 -0.227 -0.074  -2.629  -0.022   -0.37  0.3032

                        FORWARD SELECTION METHOD

                                  bprint
      NPARM     MSE RSQUARE    AGE WEIGHT RUNTIME RSTPULS RUNPULS MAXPULS

          1 7.5338  0.7434       0      0  -3.311       0       0       0
          2 7.1684  0.7642  -0.15      0  -3.204       0       0       0
          3 5.9567  0.8111 -0.256      0  -2.825       0  -0.131       0
          4 5.3435  0.8368 -0.198      0  -2.768       0  -0.348  0.2705

                          BACKWARD ELIMINATION

                                  bprint
      NPARM     MSE RSQUARE    AGE WEIGHT RUNTIME RSTPULS RUNPULS MAXPULS

          6 5.3682  0.8487 -0.227 -0.074  -2.629  -0.022   -0.37  0.3032
          5 5.1763   0.848  -0.22 -0.072  -2.683       0  -0.373  0.3049
          4 5.3435  0.8368 -0.198      0  -2.768       0  -0.348  0.2705

                             STEPWISE METHOD

                                  bprint
      NPARM     MSE RSQUARE    AGE WEIGHT RUNTIME RSTPULS RUNPULS MAXPULS

          1 7.5338  0.7434       0      0  -3.311       0       0       0
          2 7.1684  0.7642  -0.15      0  -3.204       0       0       0
          3 5.9567  0.8111 -0.256      0  -2.825       0  -0.131       0
          4 5.3435  0.8368 -0.198      0  -2.768       0  -0.348  0.2705
```

Example 9.7: Response Surface Methodology

A regression model with a complete quadratic set of regressions across several factors can be processed to yield the estimated critical values that can optimize a response. First, the regression is performed for two variables according to the model

$$y = c + b_1 x_1 + b_2 x_2 + a_{11} x_1^2 + a_{12} x_1 x_2 + a_{22} x_2^2 + e$$

The estimates are then divided into a vector of linear coefficients (estimates) **b** and a matrix of quadratic coefficients **A**. The solution for critical values is

$$\mathbf{x} = -\frac{1}{2}\mathbf{A}^{-1}\mathbf{b}$$

The following program creates a module to perform quadratic response surface regression.

```
/*          Quadratic Response Surface Regression            */
/* This matrix routine reads in the factor variables and     */
/* the response, forms the quadratic regression model and    */
/* estimates the parameters, and then solves for the optimal */
/* response, prints the optimal factors and response, and    */
/* displays the eigenvalues and eigenvectors of the          */
/* matrix of quadratic parameter estimates to determine if   */
/* the solution is a maximum or minimum, or saddlepoint, and */
/* which direction has the steepest and gentlest slopes.     */
/*                                                           */
/* Given that d contains the factor variables,               */
/* and y contains the response.                              */
/*                                                           */

start rsm;
   n=nrow(d);
   k=ncol(d);                                  /* dimensions */
   x=j(n,1,1)||d;                    /* set up design matrix */
   do i=1 to k;
      do j=1 to i;
         x=x||d[,i] #d[,j];
      end;
   end;
   beta=solve(x`*x,x`*y);       /* solve parameter estimates */
   print "Parameter Estimates" , beta;
   c=beta[1];                           /* intercept estimate */
   b=beta[2:(k+1)];                       /* linear estimates */
   a=j(k,k,0);
   L=k+1;                     /* form quadratics into matrix */
   do i=1 to k;
      do j=1 to i;
         L=L+1;
         a[i,j]=beta [L,];
      end;
   end;
   a=(a+a`)*.5;                                /* symmetrize */
   xx=-.5*solve(a,b);            /* solve for critical value */
   print , "Critical Factor Values" , xx;
      /* Compute response at critical value */
   yopt=c + b`*xx + xx`*a*xx;
   print , "Response at Critical Value" yopt;
   call eigen(eval,evec,a);
   print , "Eigenvalues and Eigenvectors", eval, evec;
   if min(eval)>0 then print , "Solution Was a Minimum";
   if max(eval)<0 then print , "Solution Was a Maximum";
finish rsm;
```

```
/* Sample Problem with Two Factors */
d={-1 -1, -1  0, -1  1,
    0 -1,  0  0,  0  1,
    1 -1,  1  0,  1  1};
y={ 71.7, 75.2, 76.3, 79.2, 81.5, 80.2, 80.1, 79.1, 75.8};
run rsm;
```

Running the module with the sample data produces the results shown in Output 9.7.1:

Output 9.7.1 Response Surface Regression: Results

```
                    Parameter Estimates


                           beta

                        81.222222
                        1.9666667
                        0.2166667
                        -3.933333
                           -2.225
                        -1.383333

                  Critical Factor Values

                            xx

                        0.2949376
                        -0.158881

                                           yopt

            Response at Critical Value 81.495032

               Eigenvalues and Eigenvectors

                           eval

                         -0.96621
                        -4.350457

                           evec

                   -0.351076 0.9363469
                   0.9363469 0.3510761

                  Solution Was a Maximum
```

Example 9.8: Logistic and Probit Regression for Binary Response Models

A binary response Y is fit to a linear model according to

$$\begin{aligned} \Pr(Y = 1) &= F(\mathbf{X}\beta) \\ \Pr(Y = 0) &= 1 - F(\mathbf{X}\beta) \end{aligned}$$

where F is some smooth probability distribution function. The normal and logistic distribution functions are supported. The method is maximum likelihood via iteratively reweighted least squares (described by Charnes, Frome, and Yu (1976); Jennrich and Moore (1975); and Nelder and Wedderburn (1972)). The row scaling is done by the derivative of the distribution (density). The weighting is done by $w/p(1-p)$, where w has the counts or other weights. The following program calculates logistic and probit regression for binary response models.

```
/* routine for estimating binary response models              */
/* y is the binary response, x are regressors,                */
/* wgt are count weights,                                     */
/* model is choice of logit probit,                           */
/* parm has the names of the parameters                       */

proc iml ;

start binest;
   b=repeat(0,ncol(x),1);
   oldb=b+1;                              /* starting values */
   do iter=1 to 20 while(max(abs(b-oldb))>1e-8);
      oldb=b;
      z=x*b;
      run f;
      loglik=sum(((y=1)#log(p) + (y=0)#log(1-p))#wgt);
      btransp=b`;
      print iter loglik btransp;
      w=wgt/(p#(1-p));
      xx=f#x;
      xpxi=inv(xx`*(w#xx));
      b=b + xpxi*(xx`*(w#(y-p)));
   end;
   p0=sum((y=1)#wgt)/sum(wgt);           /* average response */
   loglik0=sum(((y=1)#log(p0) + (y=0)#log(1-p0))#wgt);
   chisq=(2#(loglik-loglik0));
   df=ncol(x)-1;
   prob=1-probchi(chisq,df);

   print ,
      'Likelihood Ratio, Intercept-only Model' chisq df prob,;

   stderr=sqrt(vecdiag(xpxi));
   tratio=b/stderr;
   print parm b stderr tratio,,;
finish;
```

```
   /*---routine to yield distribution function and density---*/
start f;
   if model='LOGIT' then
   do;
      p=1/(1+exp(-z));
      f=p#p#exp(-z);
   end;
   if model='PROBIT' then
   do;
      p=probnorm(z);
      f=exp(-z#z/2)/sqrt(8*atan(1));
   end;
finish;

     /* Ingot data from COX (1970, pp. 67-68)*/
data={ 7 1.0 0 10, 14 1.0 0 31, 27 1.0 1 56, 51 1.0 3 13,
       7 1.7 0 17, 14 1.7 0 43, 27 1.7 4 44, 51 1.7 0 1,
       7 2.2 0 7, 14 2.2 2 33, 27 2.2 0 21, 51 2.2 0 1,
       7 2.8 0 12, 14 2.8 0 31, 27 2.8 1 22,
       7 4.0 0 9, 14 4.0 0 19, 27 4.0 1 16, 51 4.0 0 1};
nready=data[,3];
ntotal=data[,4];
n=nrow(data);
x=repeat(1,n,1)||(data[,{1 2}]);    /* intercept, heat, soak */
x=x//x;                                          /* regressors */
y=repeat(1,n,1)//repeat(0,n,1);           /* binary response */
wgt=nready//(ntotal-nready);                   /* row weights */
parm={intercept, heat, soak};         /* names of regressors */

model={logit};
run binest;                               /* run logit model */

model={probit};
run binest;                              /* run probit model */
```

The results are shown in Output 9.8.1.

Output 9.8.1 Logistic and Probit Regression: Results

```
                      iter    loglik   btransp

                         1  -268.248         0         0         0

                      iter    loglik   btransp

                         2 -76.29481 -2.159406 0.0138784 0.0037327

                      iter    loglik   btransp

                         3 -53.38033  -3.53344 0.0363154 0.0119734
```

Output 9.8.1 *continued*

```
                    iter    loglik   btransp

                       4 -48.34609 -4.748899 0.0640013 0.0299201

                    iter    loglik   btransp

                       5 -47.69191 -5.413817 0.0790272   0.04982

                    iter    loglik   btransp

                       6 -47.67283 -5.553931 0.0819276 0.0564395

                    iter    loglik   btransp

                       7 -47.67281  -5.55916 0.0820307 0.0567708

                    iter    loglik   btransp

                       8 -47.67281 -5.559166 0.0820308 0.0567713

                                                chisq        df      prob

        Likelihood Ratio, Intercept-only Model  11.64282         2 0.0029634

                    parm              b    stderr    tratio

                    INTERCEPT -5.559166 1.1196947 -4.964895
                    HEAT      0.0820308 0.0237345 3.4561866
                    SOAK      0.0567713 0.3312131 0.1714042

                    iter    loglik   btransp

                       1  -268.248         0         0         0

                    iter    loglik   btransp

                       2 -71.71043 -1.353207  0.008697 0.0023391

                    iter    loglik   btransp

                       3 -51.64122 -2.053504 0.0202739 0.0073888

                    iter    loglik   btransp

                       4 -47.88947 -2.581302  0.032626  0.018503

                    iter    loglik   btransp

                       5 -47.48924 -2.838938 0.0387625 0.0309099

                    iter    loglik   btransp

                       6 -47.47997 -2.890129 0.0398894 0.0356507
```

Output 9.8.1 *continued*

```
                    iter    loglik   btransp

                       7 -47.47995  -2.89327 0.0399529 0.0362166

                    iter    loglik   btransp

                       8 -47.47995 -2.893408 0.0399553 0.0362518

                    iter    loglik   btransp

                       9 -47.47995 -2.893415 0.0399554 0.0362537

                    iter    loglik   btransp

                      10 -47.47995 -2.893415 0.0399555 0.0362538

                    iter    loglik   btransp

                      11 -47.47995 -2.893415 0.0399555 0.0362538

                                                  chisq        df      prob

          Likelihood Ratio, Intercept-only Model 12.028543         2 0.0024436

                    parm              b    stderr    tratio

                    INTERCEPT -2.893415 0.5006009 -5.779884
                    HEAT      0.0399555 0.0118466 3.3727357
                    SOAK      0.0362538 0.1467431 0.2470561

                parm              b    stderr    tratio

                INTERCEPT -2.893415 0.5006009 -5.779884
                HEAT      0.0399555 0.0118466 3.3727357
                SOAK      0.0362538 0.1467431 0.2470561
```

Example 9.9: Linear Programming

The two-phase method for linear programming can be used to solve the problem

$$\max \mathbf{c}'\mathbf{x}$$
$$\text{st. } \mathbf{Ax} \leq, =, \geq \mathbf{b}$$
$$\mathbf{x} \geq 0$$

A SAS/IML routine that solves this problem follows. The approach appends slack, surplus, and artificial variables to the model where needed. It then solves phase 1 to find a primal feasible solution. If a primal feasible solution exists and is found, the routine then goes on to phase 2 to find an optimal solution, if one exists. The routine is general enough to handle minimizations as well as maximizations.

```
/*  Subroutine to solve Linear Programs                      */
/*  names:    names of the decision variables                */
/*  obj:      coefficients of the objective function         */
/*  maxormin: the value 'MAX' or 'MIN', upper or lowercase   */
/*  coef:     coefficients of the constraints                */
/*  rel:      character array of values: '<=' or '>=' or '=' */
/*  rhs:      right-hand side of constraints                 */
/*  activity: returns the optimal value of decision variables*/
/*                                                           */
*ods trace output;
proc iml;
start linprog( names, obj, maxormin, coef, rel, rhs, activity);

   bound=1.0e10;
   m=nrow(coef);
   n=ncol(coef);

      /* Convert to maximization */
   if upcase(maxormin)='MIN' then o=-1;
   else o=1;

      /* Build logical variables */
   rev=(rhs<0);
   adj=(-1*rev)+^ rev;
   ge =(( rel = '>=' ) & ^rev) | (( rel = '<=' ) & rev);
   eq=(rel='=');
   if max(ge)=1 then
   do;
      sr=I(m);
      logicals=-sr[,loc(ge)]||I(m);
      artobj=repeat(0,1,ncol(logicals)-m)||(eq+ge)`;
   end;
   else do;
      logicals=I(m);
      artobj=eq`;
   end;
   nl=ncol(logicals);
   nv=n+nl+2;


      /* Build coef matrix */
   *ods trace output;
   proc iml;
   a=((o*obj)||repeat(0,1,nl)||{ -1 0 })//
     (repeat(0,1,n)||-artobj||{ 0 -1 })//
     ((adj#coef)||logicals||repeat(0,m,2));

      /* rhs, lower bounds, and basis */
   b={0,0}//(adj#rhs);
   L=repeat(0,1,nv-2)||-bound||-bound;
   basis=nv-(0:nv-1);
```

```
      /* Phase 1 - primal feasibility */
   call lp(rc,x,y,a,b,nv,,l,basis);
   print ( { ' ',
           '**********Primal infeasible problem************',
           ' ',
           '*********Numerically unstable problem**********',
           '*********Singular basis encountered************',
           '*******Solution is numerically unstable********',
           '***Subroutine could not obtain enough memory***',
           '**********Number of iterations exceeded********'
           }[rc+1]);
   if x[nv] ^=0 then
   do;
      print '**********Primal infeasible problem************';
      stop;
   end;
   if rc>0 then stop;

      /* phase 2 - dual feasibility */
   u=repeat(.,1,nv-2)||{ . 0 };
   L=repeat(0,1,nv-2)||-bound||0;
   call lp(rc,x,y,a,b,nv-1,u,l,basis);

      /* Report the solution */
   print ( { '*************Solution is optimal***************',
             '*********Numerically unstable problem**********',
             '**************Unbounded problem****************',
             '*******Solution is numerically unstable********',
             '*********Singular basis encountered************',
             '*******Solution is numerically unstable********',
             '***Subroutine could not obtain enough memory***',
             '**********Number of iterations exceeded********'
            }[rc+1]);
   value=o*x  [nv-1];
   print ,'Objective Value ' value;
   activity= x [1:n] ;
   print ,'Decision Variables ' activity[r=names];
   lhs=coef*x[1:n];
   dual=y[3:m+2];
   print ,'Constraints ' lhs rel rhs dual,
          '***********************************************';

finish;
```

Consider the following product mix example (Hadley 1962). A shop with three machines, A, B, and C, turns out products 1, 2, 3, and 4. Each product must be processed on each of the three machines (for example, lathes, drills, and milling machines). The following table shows the number of hours required by each product on each machine:

	Product			
Machine	**1**	**2**	**3**	**4**
A	1.5	1	2.4	1
B	1	5	1	3.5
C	1.5	3	3.5	1

The weekly time available on each of the machines is 2000, 8000, and 5000 hours, respectively. The products contribute 5.24, 7.30, 8.34, and 4.18 to profit, respectively. What mixture of products can be manufactured that maximizes profit? You can solve the problem as follows:

```
names={'product 1' 'product 2' 'product 3' 'product 4'};
profit={ 5.24 7.30 8.34 4.18};
tech={ 1.5 1 2.4 1 ,
       1 5 1 3.5 ,
       1.5 3 3.5 1 };
time={ 2000, 8000, 5000};
rel={ '<=', '<=', '<=' };
run linprog(names,profit,'max',tech,rel,time,products);
```

The results from this example are shown in Output 9.9.1.

Output 9.9.1 Product Mix: Optimal Solution

```
*************Solution is optimal***************

                               value

          Objective Value  12737.059

                        activity

    Decision Variables  product 1 294.11765
                        product 2      1500
                        product 3         0
                        product 4 58.823529

                   lhs rel       rhs      dual

Constraints       2000 <=       2000 1.9535294
                  8000 <=       8000 0.2423529
                  5000 <=       5000 1.3782353

***********************************************
```

The following example shows how to find the minimum cost flow through a network by using linear programming. The arcs are defined by an array of tuples; each tuple names a new arc. The elements in the arc tuples give the names of the tail and head nodes that define the arc. The following data are needed: arcs, cost for a unit of flow across the arcs, nodes, and supply and demand at each node.

The following program generates the node-arc incidence matrix and calls the linear program routine for solution:

```
arcs={ 'ab' 'bd' 'ad' 'bc' 'ce' 'de' 'ae' };
   cost={ 1 2 4 3 3 2 9 };
   nodes={ 'a', 'b', 'c', 'd', 'e'};
   supdem={ 2, 0, 0, -1, -1 };
   rel=repeat('=',nrow(nodes),1);
   inode=substr(arcs,1,1);
   onode=substr(arcs,2,1);
   free n_a_i n_a_o;
   do i=1 to ncol(arcs);
      n_a_i=n_a_i || (inode[i]=nodes);
      n_a_o=n_a_o || (onode[i]=nodes);
   end;
   n_a=n_a_i - n_a_o;
   run linprog(arcs,cost,'min',n_a,rel,supdem,x);
```

The solution is shown in Output 9.9.2.

Output 9.9.2 Minimum Cost Flow: Optimal Solution

```
                   *************Solution is optimal***************

                                                value

                             Objective Value        8

                                         activity

                        Decision Variables  ab              2
                                            bd              2
                                            ad              0
                                            bc              0
                                            ce              0
                                            de              1
                                            ae              0

                                       lhs rel        rhs      dual

                   Constraints           2 =            2      -2.5
                                         0 =            0      -1.5
                                         0 =            0      -0.5
                                        -1 =           -1      -0.5
                                        -1 =           -1      -2.5

                   ***********************************************
```

Example 9.10: Quadratic Programming

The quadratic program

$$\begin{aligned}&\min \mathbf{c}'\mathbf{x} + \mathbf{x}'\mathbf{H}\mathbf{x}/2\\ &\text{st. } \mathbf{G}\mathbf{x} \leq, =, \geq \mathbf{b}\\ &\mathbf{x} \geq 0\end{aligned}$$

can be solved by solving an equivalent linear complementarity problem when **H** is positive semidefinite. The approach is outlined in the discussion of the LCP subroutine.

The following routine solves the quadratic problem.

```
      /*              Routine to solve quadratic programs           */
      /* names: the names of the decision variables                 */
      /* c:  vector of linear coefficients of the objective function */
      /* H:  matrix of quadratic terms in the objective function     */
      /* G:  matrix of constraint coefficients                      */
      /* rel: character array of values: '<=' or '>=' or '='        */
      /* b:   right-hand side of constraints                        */
      /* activity: returns the optimal value of decision variables   */

   start qp( names, c, H, G, rel, b, activity);
      if min(eigval(h))<0 then
      do;
         print
               'ERROR: The minimum eigenvalue of the H matrix is negative. ';
         print '       Thus it is not positive semidefinite.               ';
         print '       QP is terminating with this error.                  ';
         stop;
      end;
      nr=nrow(G);
      nc=ncol(G);

         /* Put in canonical form */
      rev=(rel='<=');
      adj=(-1 * rev) + ^rev;
      g=adj# G; b = adj # b;
      eq=( rel = '='  );
      if max(eq)=1 then
      do;
         g=g // -(diag(eq)*G)[loc(eq),];
         b=b // -(diag(eq)*b)[loc(eq)];
      end;
      m=(h || -g`) //(g || j(nrow(g),nrow(g),0));
      q=c // -b;

         /* Solve the problem */
      call lcp(rc,w,z,M,q);
         /* Report the solution */
      reset noname;
      print ( { '*************Solution is optimal***************',
                '*********No solution possible******************',
                ' ',
                ' ',
                ' ',
                '**********Solution is numerically unstable*****',
                '***********Not enough memory*******************',
                '**********Number of iterations exceeded********'}[rc+1]);
      reset name;
      activity=z[1:nc];
      objval=c`*activity + activity`*H*activity/2;
      print ,'Objective Value ' objval,
             'Decision Variables ' activity[r=names],
             '**********************************************';

      finish qp;
```

As an example, consider the following problem in portfolio selection. Models used in selecting investment portfolios include assessment of the proposed portfolio's expected gain and its associated risk. One such model seeks to minimize the variance of the portfolio subject to a minimum expected gain. This can be modeled as a quadratic program in which the decision variables are the proportions to invest in each of the possible securities. The quadratic component of the objective function is the covariance of gain between the securities; the first constraint is a proportionality constraint; and the second constraint gives the minimum acceptable expected gain.

The following data are used to illustrate the model and its solution:

```
c = { 0, 0, 0, 0 };
h = { 1003.1 4.3 6.3 5.9 ,
         4.3 2.2 2.1 3.9 ,
         6.3 2.1 3.5 4.8 ,
         5.9 3.9 4.8 10 };
g = {  1   1   1   1   ,
      .17 .11 .10 .18 };
b = { 1 , .10 };
rel = { '=', '>='};
names = {'ibm', 'dec', 'dg', 'prime' };
run qp(names,c,h,g,rel,b,activity);
```

The results in Output 9.10.1 show that the minimum variance portfolio achieving the 0.10 expected gain is composed of DEC and DG stock in proportions of 0.933 and 0.067.

Output 9.10.1 Portfolio Selection: Optimal Solution

```
                    *************Solution is optimal***************


                                                 objval

                             Objective Value  1.0966667
```

Output 9.10.1 *continued*

```
                                   activity

               Decision Variables  ibm             0
                                   dec     0.9333333
                                   dg      0.0666667
                                   prime           0

            ************************************************
```

Example 9.11: Regression Quantiles

The technique of estimating parameters in linear models by using the notion of regression quantiles is a generalization of the LAE or LAV least absolute value estimation technique. For a given quantile q, the estimate $\mathbf{b}^*$ of ˛ in the model

$$\mathbf{Y} = \mathbf{X}˛ + ›$$

is the value of b that minimizes

$$\sum_{t \in T} q|y_t - x_t b| - \sum_{t \in S} (1-q)|y_t - x_t b|$$

where $T = \{t | y_t \geq x_t b\}$ and $S = \{t | y_t \leq x_t\}$. For $q = 0.5$, the solution $\mathbf{b}^*$ is identical to the estimates produced by the LAE. The following routine finds this estimate by using linear programming.

```
/* Routine to find regression quantiles                    */
/* yname:    name of dependent variable                    */
/* y:        dependent variable                            */
/* xname:    names of independent variables                */
/* X:        independent variables                         */
/* b:        estimates                                     */
/* predict:  predicted values                              */
/* error:    difference of y and predicted.                */
/* q:        quantile                                      */
/*                                                         */
/* notes:    This subroutine finds the estimates b         */
/* that minimize                                           */
/*                                                         */
/*           q * (y - Xb) * e  + (1-q) * (y - Xb) * ^e     */
/*                                                         */
/* where e = ( Xb <= y ).                                  */
/*                                                         */
/* This subroutine follows the approach given in:          */
/*                                                         */
/* Koenker, R. and G. Bassett (1978). Regression           */
/* quantiles. Econometrica. Vol. 46. No. 1. 33-50.         */
/*                                                         */
/* Basssett, G. and R. Koenker (1982). An empirical        */
/* quantile function for linear models with iid errors.    */
/* JASA. Vol. 77. No. 378. 407-415.                        */
/*                                                         */
/* When q = .5 this is equivalent to minimizing the sum    */
/* of the absolute deviations, which is also known as      */
/* L1 regression.  Note that for L1 regression, a faster   */
/* and more accurate algorithm is available in the SAS/IML */
/* routine LAV, which is based on the approach given in:   */
/*                                                         */
/* Madsen, K. and Nielsen, H. B. (1993). A finite          */
/* smoothing algorithm for linear L1 estimation.           */
/* SIAM J. Optimization, Vol. 3. 223-235.                  */
/*---------------------------------------------------------*/
start rq( yname, y, xname, X, b, predict, error, q);
   bound=1.0e10;
   coef = X`;
   m = nrow(coef);
   n = ncol(coef);


/*-----------------build rhs and bounds--------------------*/
   e  = repeat(1,1,n)`;
   r  =  {0 0} || ((1-q)*coef*e)`;
   sign = repeat(1,1,m);

   do i=1 to m;
      if r[2+i] < 0 then do;
         sign[i]   = -1;
         r[2+i]    = -r[2+i];
        coef[i,]  = -coef[i,];
      end;
   end;

l  = repeat(0,1,n) || repeat(0,1,m)  || -bound || -bound ;
u  = repeat(1,1,n) || repeat(.,1,m)  ||   { . . }  ;

/*-------------build coefficient matrix and basis----------*/
 a  = (        y`     || repeat(0,1,m)  ||   { -1 0 }   ) //
      ( repeat(0,1,n) || repeat(-1,1,m) ||   { 0 -1 }   ) //
      (      coef     ||     I(m)       || repeat(0,m,2)) ;
 basis = n+m+2 - (0:n+m+1);

/*----------------find a feasible solution-----------------*/
call lp(rc,p,d,a,r,,u,l,basis);

/*----------------find the optimal solution----------------*/
l  = repeat(0,1,n) || repeat(0,1,m)  || -bound || {0} ;
u  = repeat(1,1,n) || repeat(0,1,m)  ||   { . 0 }  ;
call lp(rc,p,d,a,r,n+m+1,u,l,basis);

/*---------------- report the solution-----------------------*/
variable = xname`; b=d[3:m+2];
do i=1 to m;
   b[i] =  b[i] * sign[i];
```

```
end;
predict = X*b;
error = y - predict;
wsum  = sum ( choose(error<0 , (q-1)*error , q*error) );

print ,,'Regression Quantile Estimation' ,
        'Dependent Variable: ' yname ,
        'Regression Quantile: ' q ,
        'Number of Observations: ' n ,
        'Sum of Weighted Absolute Errors: ' wsum ,
         variable b,
         X y predict error;
finish rq;
```

The following example uses data on the United States population from 1790 to 1970:

```
z = { 3.929 1790 ,
      5.308 1800 ,
      7.239 1810 ,
      9.638 1820 ,
     12.866 1830 ,
     17.069 1840 ,
     23.191 1850 ,
     31.443 1860 ,
     39.818 1870 ,
     50.155 1880 ,
     62.947 1890 ,
     75.994 1900 ,
     91.972 1910 ,
    105.710 1920 ,
    122.775 1930 ,
    131.669 1940 ,
    151.325 1950 ,
    179.323 1960 ,
    203.211 1970 };

y=z[,1];
x=repeat(1,19,1)||z[,2]||z[,2]##2;
run rq('pop',y,{'intercpt' 'year' 'yearsq'},x,b1,pred,resid,.5);
```

The results are shown in Output 9.11.1.

Output 9.11.1 Regression Quantiles: Results

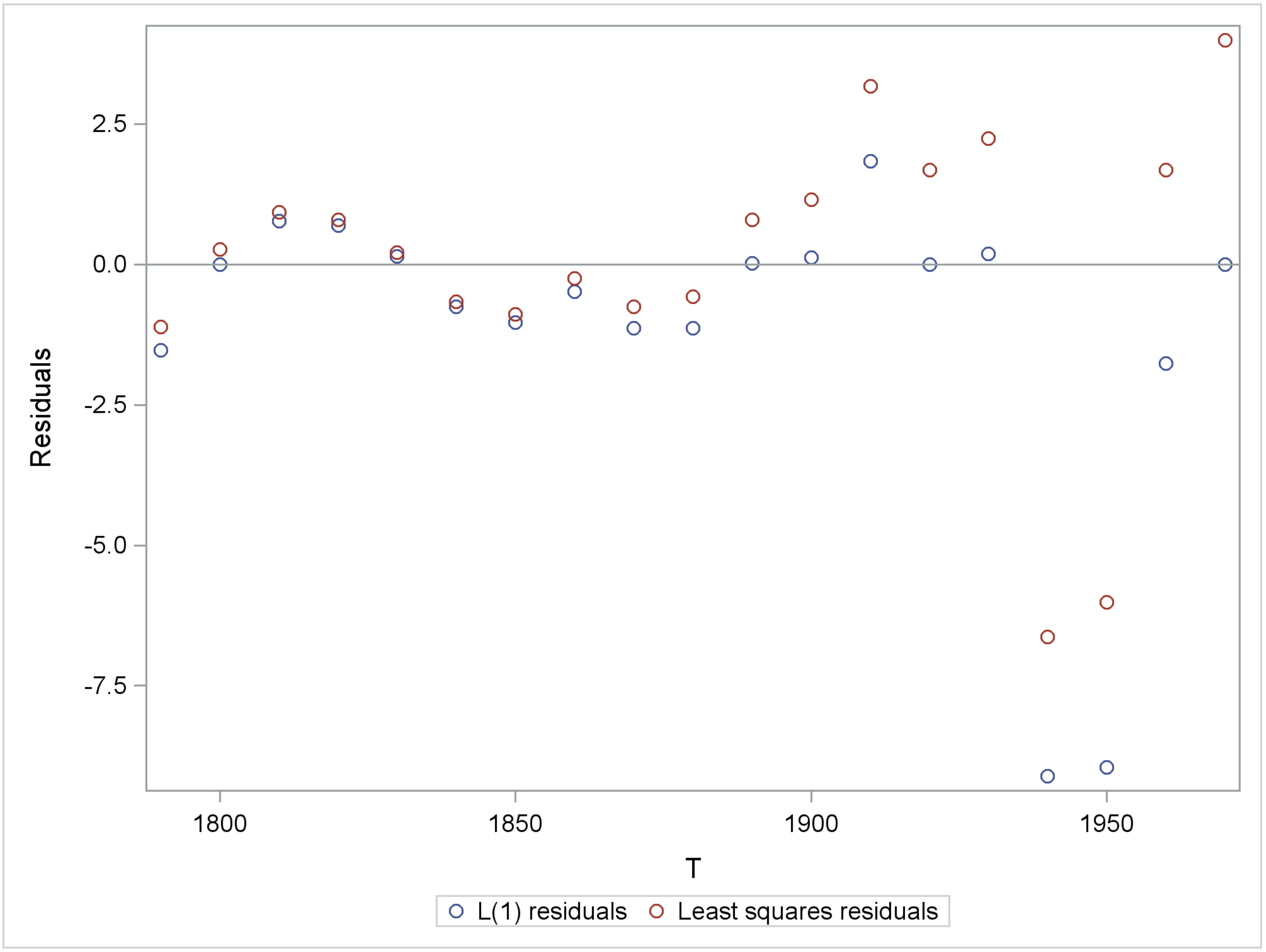
```
                         Regression Quantile Estimation


                                             yname

                           Dependent Variable:  pop

                                                    q

                         Regression Quantile:        0.5
```

Output 9.11.1 *continued*

```
                                              n

             Number of Observations:         19

                                                wsum

         Sum of Weighted Absolute Errors:  14.826429

                    variable         b

                    intercpt 21132.758
                    year     -23.52574
                    yearsq    0.006549

         X                            y   predict     error

         1      1790   3204100     3.929 5.4549176 -1.525918
         1      1800   3240000     5.308     5.308 -4.54E-12
         1      1810   3276100     7.239 6.4708902 0.7681098
         1      1820   3312400     9.638 8.9435882 0.6944118
         1      1830   3348900    12.866 12.726094 0.1399059
         1      1840   3385600    17.069 17.818408 -0.749408
         1      1850   3422500    23.191 24.220529 -1.029529
         1      1860   3459600    31.443 31.932459 -0.489459
         1      1870   3496900    39.818 40.954196 -1.136196
         1      1880   3534400    50.155 51.285741 -1.130741
         1      1890   3572100    62.947 62.927094 0.0199059
         1      1900   3610000    75.994 75.878255 0.1157451
         1      1910   3648100    91.972 90.139224 1.8327765
         1      1920   3686400    105.71    105.71 8.669E-13
         1      1930   3724900   122.775 122.59058 0.1844157
         1      1940   3763600   131.669 140.78098 -9.111976
         1      1950   3802500   151.325 160.28118 -8.956176
         1      1960   3841600   179.323 181.09118 -1.768184
         1      1970   3880900   203.211   203.211 -2.96E-12
```

The L1 norm (when $q = 0.5$) tends to cause the fit to be better at more points at the expense of causing some points to fit worse, as shown by the following plot, which compares the L1 residuals with the least squares residuals.

Output 9.11.2 L1 Residuals vs. Least Squares Residuals



When $q = 0.5$, the results of this module can be compared with the results of the LAV routine, as follows:

```
b0 = {1 1 1};               /* initial value            */
optn = j(4,1,.);            /* options vector           */

optn[1]= .;                 /* gamma default            */
optn[2]= 5;                 /* print all                */
optn[3]= 0;                 /* McKean-Schradar variance */
optn[4]= 1;                 /* convergence test         */

call LAV(rc, xr, x, y, b0, optn);
```

Example 9.12: Simulations of a Univariate ARMA Process

Simulations of time series with known ARMA structure are often needed as part of other simulations or as learning data sets for developing time series analysis skills. The following program generates a time series by using the IML functions NORMAL, ARMACOV, HANKEL, PRODUCT, RATIO, TOEPLITZ, and ROOT.

```
proc iml;
reset noname;
start armasim(y,n,phi,theta,seed);
/*-----------------------------------------------------------*/
/* IML Module: armasim                                       */
/* Purpose: Simulate n data points from ARMA process         */
/*          exact covariance method                          */
/* Arguments:                                                */
/*                                                           */
/* Input: n    : series length                               */
/*        phi  : AR coefficients                             */
/*        theta: MA coefficients                             */
/*        seed : integer seed for normal deviate generator   */
/* Output: y: realization of ARMA process                    */
/* ----------------------------------------------------------*/

   p=ncol(phi)-1;
   q=ncol(theta)-1;
   y=normal(j(1,n+q,seed));

      /* Pure MA or white noise */
   if p=0 then y=product(theta,y)[,(q+1):(n+q)];
   else do;                /* Pure AR or ARMA */

         /* Get the autocovariance function */
      call armacov(gamma,cov,ma,phi,theta,p);
      if gamma[1]<0 then
      do;
         print 'ARMA parameters not stable.';
         print 'Execution terminating.';
         stop;
      end;

         /* Form covariance matrix */
      gamma=toeplitz(gamma);

         /* Generate covariance between initial y and */
         /* initial innovations                        */
      if q>0 then
      do;
         psi=ratio(phi,theta,q);
         psi=hankel(psi[,-((-q):(-1))]);
         m=max(1,(q-p+1));
         psi=psi[-((-q):(-m)),];
         if p>q then psi=j(p-q,q,0)//psi;
         gamma=(gamma||psi)//(psi`||i(q));
      end;

         /* Use Cholesky root to get startup values */
      gamma=root(gamma);
      startup=y[,1:(p+q)]*gamma;
```

```
         e=y[,(p+q+1):(n+q)];

            /* Generate MA part */
         if q>0 then
         do;
            e=startup[,(p+1):(p+q)]||e;
            e=product(theta,e)[,(q+1):(n+q-p)];
         end;

         y=startup[,1:p];
         phi1=phi[,-(-(p+1):(-2))]`;

            /* Use difference equation to generate */
            /* remaining values                    */
         do ii=1 to n-p;
            y=y||(e[,ii]-y[,ii:(ii+p-1)]*phi1);
         end;
      end;
      y=y`;
   finish armasim; /* ARMASIM */

   run armasim(y,10,{1 -0.8},{1 0.5},1234321);
   print ,'Simulated Series:', y;
```

The results are shown in Output 9.12.1.

Output 9.12.1 Simulated Series

```
                                   Simulated Series:


                                       3.0764594
                                       1.8931735
                                       0.9527984
                                       0.0892395
                                       -1.811471
                                         -2.8063
                                        -2.52739
                                       -2.865251
                                       -1.332334
                                       0.1049046
```

Example 9.13: Parameter Estimation for a Regression Model with ARMA Errors

Nonlinear estimation algorithms are required for obtaining estimates of the parameters of a regression model with innovations having an ARMA structure. The three estimation methods employed by the ARIMA procedure in SAS/ETS software are written in IML in the following program. The algorithms employed are slightly different from those used by PROC ARIMA, but the results obtained should be similar. This example combines the IML functions ARMALIK, PRODUCT, and RATIO to perform the estimation. Note the interactive nature of this example, illustrating how you can adjust the estimates when they venture outside the stationary or invertible regions.

```
/*-------------------------------------------------------------*/
/*----  Grunfeld's Investment Models Fit with ARMA Errors  ----*/
/*-------------------------------------------------------------*/
data grunfeld;
  input year gei gef gec wi wf wc;
  label gei='gross investment ge'
        gec='capital stock lagged ge'
        gef='value of outstanding shares ge lagged'
        wi ='gross investment w'
        wc ='capital stock lagged w'
        wf ='value of outstanding shares lagged w';
/*---  GE STANDS FOR GENERAL ELECTRIC AND W FOR WESTINGHOUSE  ---*/
datalines;
1935     33.1      1170.6    97.8      12.93     191.5     1.8
1936     45.0      2015.8    104.4     25.90     516.0     .8
1937     77.2      2803.3    118.0     35.05     729.0     7.4
1938     44.6      2039.7    156.2     22.89     560.4     18.1
1939     48.1      2256.2    172.6     18.84     519.9     23.5
1940     74.4      2132.2    186.6     28.57     628.5     26.5
1941     113.0     1834.1    220.9     48.51     537.1     36.2
1942     91.9      1588.0    287.8     43.34     561.2     60.8
1943     61.3      1749.4    319.9     37.02     617.2     84.4
1944     56.8      1687.2    321.3     37.81     626.7     91.2
1945     93.6      2007.7    319.6     39.27     737.2     92.4
1946     159.9     2208.3    346.0     53.46     760.5     86.0
1947     147.2     1656.7    456.4     55.56     581.4     111.1
1948     146.3     1604.4    543.4     49.56     662.3     130.6
1949     98.3      1431.8    618.3     32.04     583.8     141.8
1950     93.5      1610.5    647.4     32.24     635.2     136.7
1951     135.2     1819.4    671.3     54.38     723.8     129.7
1952     157.3     2079.7    726.1     71.78     864.1     145.5
1953     179.5     2371.6    800.3     90.08     1193.5    174.8
1954     189.6     2759.9    888.9     68.60     1188.9    213.5
;
run;

proc iml;
reset noname;
/*-----------------------------------------------------------*/
/*  name: ARMAREG Modules                                    */
/*  purpose: Perform Estimation for regression model with    */
/*  ARMA errors                                              */
/*  usage: Before invoking the command                       */
/*                                                           */
/*  run armareg;                                             */
/*                                                           */
/*  define the global parameters                             */
/*                                                           */
/*  x      - matrix of predictors.                           */
/*  y      - response vector.                                */
/*  iphi   - defines indices of nonzero AR parameters,       */
/*           omit the index 0 which corresponds to the zero  */
/*           order constant one.                             */
/*  itheta - defines indices of nonzero MA parameters,       */
/*           omit the index 0 which corresponds to the zero  */
/*           order constant one.                             */
/*  ml     - estimation option: -1 if Conditional Least      */
/*           Squares, 1 if Maximum Likelihood, otherwise     */
/*           Unconditional Least Squares.                    */
/*  delta  - step change in parameters (default 0.005).      */
/*  par    - initial values of parms. First ncol(iphi)       */
/*           values correspond to AR parms, next ncol(itheta)*/
/*           values correspond to MA parms, and remaining    */
/*           are regression coefficients.                    */
/*  init   - undefined or zero for first call to ARMAREG.    */
/*  maxit  - maximum number of iterations. No other          */
/*           convergence criterion is used. You can invoke   */
/*           ARMAREG without changing parameter values to    */
/*           continue iterations.                            */
/*  nopr   - undefined or zero implies no printing of        */
/*           intermediate results.                           */
/*                                                           */
/*  notes: Optimization using Gauss-Newton iterations        */
/*                                                           */
/*  No checking for invertibility or stationarity during     */
/*  estimation process. The parameter array par can be       */
/*  modified after running armareg to place estimates        */
/*  in the stationary and invertible regions, and then       */
/*  armareg can be run again. If a nonstationary AR operator */
/*  is employed, a PAUSE will occur after calling ARMALIK    */
/*  because of a detected singularity. Using STOP will       */
/*  permit termination of ARMAREG so that the AR             */
/*  coefficients can be modified.                            */
/*                                                           */
/*  T-ratios are only approximate and can be undependable,   */
/*  especially for small series.                             */
/*                                                           */
/*  The notation follows that of the IML function ARMALIK;   */
/*  the autoregressive and moving average coefficients have  */
/*  signs opposite those given by PROC ARIMA.                */

/* Begin ARMA estimation modules */

/* Generate residuals */
start gres;
   noise=y-x*beta;
   previous=noise[:];
   if ml=-1 then do;                        /* Conditional LS */
      noise=j(nrow(y),1,previous)//noise;
      resid=product(phi,noise`)[,nrow(y)+1:nrow(noise)];
      resid=ratio(theta,resid,ncol(resid));
      resid=resid[,1:ncol(resid)]`;
   end;
   else do;                             /* Maximum likelihood */
      free l;
```

```
         call armalik(l,resid,std,noise,phi,theta);

         /* Nonstationary condition produces PAUSE */
         if nrow(l)=0 then do;
            print ,
            'In GRES: Parameter estimates outside stationary region.';
         end;
         else do;
            temp=l[3,]/(2#nrow(resid));
            if ml=1 then resid=resid#exp(temp);
         end;
      end;
   finish gres;                           /* finish module GRES  */

   start getpar;                              /* get parameters  */
      if np=0 then phi=1;
      else do;
         temp=parm[,1:np];
         phi=1||j(1,p,0);
         phi[,iphi] =temp;
      end;
      if nq=0 then theta=1;
      else do;
         temp=parm[,np+1:np+nq];
         theta=1||j(1,q,0);
         theta[,itheta] =temp;
      end;
      beta=parm[,(np+nq+1):ncol(parm)]`;
   finish getpar;   /* finish module GETPAR  */


   /* Get SS Matrix - First Derivatives */
   start getss;
      parm=par;
      run getpar;
      run gres;
      s=resid;
      oldsse=ssq(resid);
      do k=1 to ncol(par);
         parm=par;
         parm[,k]=parm[,k]+delta;
         run getpar;
         run gres;
         s=s||((resid-s[,1])/delta);      /* append derivatives */
      end;
      ss=s`*s;
      if nopr^=0 then print ,'Gradient Matrix', ss;
      sssave=ss;
      do k=1 to 20;            /* Iterate if no reduction in SSE */
         do ii=2 to ncol(ss);
            ss[ii,ii]=(1+lambda)*ss[ii,ii];
         end;
         ss=sweep(ss,2:ncol(ss));       /* Gaussian elimination */
         delpar=ss[1,2:ncol(ss)];     /* update parm increments */
         parm=par+delpar;
         run getpar;
         run gres;
         sse=ssq(resid);
         ss=sssave;
         if sse<oldsse then do;      /* reduction, no iteration */
            lambda=max(lambda/10,1e-12);
            k=21;
         end;
         else do;                                /* no reduction */
                                 /* increase lambda and iterate */
            if nopr^=0 then print ,
               'Lambda=' lambda 'SSE=' sse 'OLDSSE=' oldsse,
               'Gradient Matrix', ss ;
            lambda=min(10*lambda,1e12);
            if k=20 then do;
               print 'In module GETSS:'
                     'No improvement in SSE after twenty iterations.';
               print ' Possible Ridge Problem. ';
               return;
            end;
         end;
      end;
      if nopr^=0 then print ,'Gradient Matrix', ss;
      finish getss;                     /* Finish module GETSS  */

   start armareg;                        /* ARMAREG main module */
      /* Initialize options and parameters */
      if nrow(delta)=0 then delta=0.005;
      if nrow(maxiter)=0 then maxiter=5;
      if nrow(nopr)=0 then nopr=0;
      if nrow(ml)=0 then ml=1;
      if nrow(init)=0 then init=0;
      if init=0 then do;
         p=max(iphi);
         q=max(itheta);
         np=ncol(iphi);
         nq=ncol(itheta);

         /* Make indices one-based */
         do k=1 to np;
            iphi[,k]=iphi[,k]+1;
         end;
         do k=1 to nq;
            itheta[,k]=itheta[,k]+1;
         end;

         /* Create row labels for Parameter estimates */
         if p>0 then parmname = concat("AR",char(1:p,2));
         if q>0 then parmname = parmname||concat("MA",char(1:q,2));
         parmname = parmname||concat("B",char(1:ncol(x),2));

         /* Create column labels for Parameter estimates */
         pname = {"Estimate" "Std. Error" "T-Ratio"};
```

```
         init=1;
      end;

      /* Generate starting values */
      if nrow(par)=0 then do;
         beta=inv(x`*x)*x`*y;
         if np+nq>0 then par=j(1,np+nq,0)||beta`;
         else par=beta`;
      end;
      print ,'Parameter Starting Values',;
      print par [colname=parmname];            /* stderr tratio */
      lambda=1e-6;                           /* Controls step size */
      do iter=1 to maxiter;            /* Do maxiter iterations */
         run getss;
         par=par+delpar;
         if nopr^=0 then do;
            print ,'Parameter Update',;
            print par [colname=parmname];      /* stderr tratio */
            print ,'Lambda=' lambda,;
         end;
      end;

      sighat=sqrt(sse/(nrow(y)-ncol(par)));
      print ,'Innovation Standard Deviation:' sighat;
      ss=sweep(ss,2:ncol(ss));          /* Gaussian elimination */
      estm=par`||(sqrt(diag(ss[2:ncol(ss),2:ncol(ss)]))
           *j(ncol(par),1,sighat));
      estm=estm||(estm[,1] /estm[,2]);
      if ml=1 then print ,'Maximum Likelihood Estimation Results',;
      else if ml=-1 then print ,
         'Conditional Least Squares Estimation Results',;
      else print ,'Unconditional Least Squares Estimation Results',;
      print estm [rowname=parmname colname=pname] ;
   finish armareg;
   /* End of ARMA Estimation modules */

   /* Begin estimation for Grunfeld's investment models */
   use grunfeld;
   read all var {gei} into y;
   read all var {gef gec} into x;
   close grunfeld;

   x=j(nrow(x),1,1)||x;
   iphi=1;
   itheta=1;
   maxiter=10;
   delta=0.0005;
   ml=-1;
   /*----  To prevent overflow, specify starting values  ----*/
   par={-0.5  0.5  -9.956306  0.0265512  0.1516939};
   run armareg;   /*----  Perform CLS estimation  ----*/
```

The results are shown in Output 9.13.1.

Output 9.13.1 Conditional Least Squares Results

```
                          Parameter Starting Values

                   AR 1      MA 1       B 1       B 2       B 3

                   -0.5       0.5 -9.956306 0.0265512 0.1516939

           In module GETSS: No improvement in SSE after twenty iterations.

                           Possible Ridge Problem.

           In module GETSS: No improvement in SSE after twenty iterations.

                           Possible Ridge Problem.

           In module GETSS: No improvement in SSE after twenty iterations.

                           Possible Ridge Problem.

                   Innovation Standard Deviation: 22.653769

                 Conditional Least Squares Estimation Results

                           Estimate Std. Error   T-Ratio

                     AR 1 -0.230905  0.3429525 -0.673287
                     MA 1   0.69639  0.2480617 2.8073252
                     B 1  -20.87774  31.241368 -0.668272
                     B 2   0.038706  0.0167503 2.3107588
                     B 3  0.1216554  0.0441722 2.7541159
```

```
/*----  With CLS estimates as starting values,  ----*/
/*----  perform ML estimation.                  ----*/
ml=1;
maxiter=10;
run armareg;
```

The results are shown in Output 9.13.2.

Output 9.13.2 Maximum Likelihood Results

```
                           Estimate Std. Error   T-Ratio

                     AR 1 -0.230905  0.3429525 -0.673287
                     MA 1   0.69639  0.2480617 2.8073252
                     B 1  -20.87774  31.241368 -0.668272
                     B 2   0.038706  0.0167503 2.3107588
                     B 3  0.1216554  0.0441722 2.7541159

                          Parameter Starting Values
```

Output 9.13.2 *continued*

```
              AR 1       MA 1        B 1        B 2        B 3

         -0.230905   0.69639 -20.87774  0.038706 0.1216554

             Innovation Standard Deviation: 23.039253

               Maximum Likelihood Estimation Results

                      Estimate Std. Error   T-Ratio

               AR 1 -0.196224  0.3510868 -0.558904
               MA 1 0.6816033  0.2712043 2.5132468
               B 1  -26.47514  33.752826 -0.784383
               B 2  0.0392213  0.0165545 2.3692242
               B 3  0.1310306  0.0425996 3.0758622
```

Example 9.14: Iterative Proportional Fitting

The classical use of iterative proportional fitting is to adjust frequencies to conform to new marginal totals. Use the IPF subroutine to perform this kind of analysis. You supply a table that contains new margins and a table that contains old frequencies. The IPF subroutine returns a table of adjusted frequencies that preserves any higher-order interactions appearing in the initial table.

The following example is a census study that estimates a population distribution according to age and marital status (Bishop, Fienberg, and Holland 1975). Estimates of the distribution are known for the previous year, but only estimates of marginal totals are known for the current year. You want to adjust the distribution of the previous year to fit the estimated marginal totals of the current year. Here is the program:

```
proc iml;

   /* Stopping criteria */
mod={0.01 15};

   /* Marital status has 3 levels. age has 8 levels. */
dim={3 8};

   /* New marginal totals for age by marital status */
table={1412 0 0 ,
       1402 0 0 ,
       1174 276 0 ,
       0 1541 0 ,
       0 1681 0 ,
       0 1532 0 ,
       0 1662 0 ,
       0 5010 2634};

   /* Marginal totals are known for both */
   /* marital status and age             */
```

```
config={1 2};

   /* Use known distribution for start-up values */
initab={1306 83 0 ,
        619 765 3 ,
        263 1194 9 ,
        173 1372 28 ,
        171 1393 51 ,
        159 1372 81 ,
        208 1350 108 ,
        1116 4100 2329};

call ipf(fit,status,dim,table,config,initab,mod);

c={' SINGLE' ' MARRIED' 'WIDOWED/DIVORCED'};
r={'15 - 19' '20 - 24' '25 - 29' '30 - 34' '35 - 39' '40 - 44'
   '45 - 49' '50 OR OVER'};
print
   'POPULATION DISTRIBUTION ACCORDING TO AGE AND MARITAL STATUS',,
   'KNOWN DISTRIBUTION (PREVIOUS YEAR)',,
   initab [colname=c rowname=r format=8.0] ,,
   'ADJUSTED ESTIMATES OF DISTRIBUTION (CURRENT YEAR)',,
   fit [colname=c rowname=r format=8.2] ;
```

The results are shown in Output 9.14.1.

Output 9.14.1 Iterative Proportional Fitting: Results

```
              POPULATION DISTRIBUTION ACCORDING TO AGE AND MARITAL STATUS


                       KNOWN DISTRIBUTION (PREVIOUS YEAR)


                                      initab
                                SINGLE  MARRIED WIDOWED/DIVORCED

                     15 - 19        1306       83                0
                     20 - 24         619      765                3
                     25 - 29         263     1194                9
                     30 - 34         173     1372               28
                     35 - 39         171     1393               51
                     40 - 44         159     1372               81
                     45 - 49         208     1350              108
                     50 OR OVER     1116     4100             2329


                   ADJUSTED ESTIMATES OF DISTRIBUTION (CURRENT YEAR)
```

Output 9.14.1 *continued*

```
                               fit
                     SINGLE  MARRIED WIDOWED/DIVORCED

       15 - 19      1325.27    86.73             0.00
       20 - 24       615.56   783.39             3.05
       25 - 29       253.94  1187.18             8.88
       30 - 34       165.13  1348.55            27.32
       35 - 39       173.41  1454.71            52.87
       40 - 44       147.21  1308.12            76.67
       45 - 49       202.33  1352.28           107.40
       50 OR OVER   1105.16  4181.04          2357.81
```

Example 9.15: Full-Screen Nonlinear Regression

This example shows how to build a menu system that enables you to perform nonlinear regression from a menu. Six modules are stored on an IML storage disk. After you have stored them, use this example to try out the system. First, invoke IML and set up some sample data in memory, in this case the population of the U.S. from 1790 to 1970. Then invoke the module NLIN, as follows:

```
reset storage='nlin';
load module=_all_;
uspop = {3929, 5308, 7239, 9638, 12866, 17069, 23191, 31443,
         39818, 50155, 62947, 75994, 91972, 105710, 122775, 131669,
         151325, 179323, 203211}/1000;
year=do(1790,1970,10)`;
time=year-1790;
print year time uspop;
run nlin;
```

A menu similar to the following menu appears. The entry fields are shown by underscores here, but the underscores become blanks in the real session.

```
Nonlinear Regression
Response function: ______________________________________________
Predictor function: _____________________________________________

Parameter Value Derivative
: ________ ___________ __________________________________________
: ________ ___________ __________________________________________
: ________ ___________ __________________________________________
: ________ ___________ __________________________________________
: ________ ___________ __________________________________________
: ________ ___________ __________________________________________
```

Enter an exponential model and fill in the response and predictor expression fields. For each parameter, enter the name, initial value, and derivative of the predictor with respect to the parameter. Here are the populated fields:

```
Nonlinear Regression
Response function: uspop________________________________________
Predictor function: a0*exp(a1*time)______________________________

Parameter Value Derivative
: a0______ ________3.9 exp(a1*time)_______________________________
: a1______ __________0 time*a0*exp(a1*time)_______________________
: ________ ___________ ___________________________________________
: ________ ___________ ___________________________________________
: ________ ___________ ___________________________________________
: ________ ___________ ___________________________________________
```

Now press the SUBMIT key. The model compiles, the iterations start blinking on the screen, and when the model has converged, the estimates are displayed along with their standard errors, *t* test, and significance probability.

To modify and rerun the model, submit the following command:

```
run nlrun;
```

Here is the program that defines and stores the modules of the system.

```
/*          Full-Screen Nonlinear Regression                 */
/* Six modules are defined, which constitute a system for    */
/* nonlinear regression. The interesting feature of this     */
/* system is that the problem is entered in a menu, and both */
/* iterations and final results are displayed on the same    */
/* menu.                                                     */
/*                                                           */
/* Run this source to get the modules stored. Examples       */
/* of use are separate.                                      */
/*                                                           */
/* Caution: this is a demonstration system only. It does not */
/* have all the necessary safeguards in it yet to            */
/* recover from user errors or rough models.                 */
/* Algorithm:                                                */
/*    Gauss-Newton nonlinear regression with step-halving.   */
/* Notes: program variables all start with nd or _ to        */
/* minimize the problems that would occur if user variables  */
/* interfered with the program variables.                    */



/* Gauss-Newton nonlinear regression with Hartley step-halving */

/*---Routine to set up display values for new problem---*/
start nlinit;
   window nlin rows=15 columns=80 color='green'
      msgline=_msg cmndline=_cmnd
      group=title +30 'Nonlinear Regression' color='white'
      group=model / @5 'Response function:' color='white'
      +1 nddep $55. color='blue'
      / @5 'Predictor function:' color='white'
      +1 ndfun $55. color='blue'
      group=parm0 // @5 'Parameter' color='white' @15 'Value'
      @30 'Derivative'
      group=parm1 // @5 'Parameter' color='white' @15 'Value'
      group=parm2 // @5 'Parameter' color='white' @19 'Estimate'
      @33 'Std Error'
      @48 'T Ratio'
      @62 'Prob>|T|'
      group=parminit /@3 ':' color='white'
      @5 ndparm $8. color='blue'
      @15 ndbeta best12. @30 ndder $45.
      group=parmiter / @5 _parm color='white'
      @15 _beta best12. color='blue'
      group=parmest / @5 _parm color='white'
      @15 _beta best12. color='blue'
      @30 _std best12.
      @45 _t 10.4
      @60 _prob 10.4
      group=sse // @5 'Iteration =' color='white' _iter 5. color='blue'
      ' Stephalvings = ' color='white' _subit 3. color='blue'
      / @5 'Sum of Squares Error =' color='white' _sse best12.
      color='blue';
   nddep=cshape(' ',1,1,55,' ');
   ndfun=nddep;
   nd0=6;
   ndparm=repeat(' ',nd0,1);
   ndbeta=repeat(0,nd0,1);
   ndder=cshape(' ',nd0,1,55,' ');
   _msg='Enter New Nonlinear Problem';
finish nlinit;  /* Finish module NLINIT */


   /* Main routine */
start nlin;
   run nlinit;  /* initialization routine */
   run nlrun;   /* run routine */
finish nlin;

   /* Routine to show each iteration */
start nliter;
   display nlin.title noinput,
   nlin.model noinput,
   nlin.parm1 noinput,
   nlin.parmiter repeat noinput,
   nlin.sse noinput;
finish nliter;

   /* Routine for one run */
start nlrun;
   run nlgen; /* generate the model */
   run nlest; /* estimate the model */
finish nlrun;

   /* Routine to generate the model */
```

```
start nlgen;

   /* Model definition menu */
   display nlin.title, nlin.model, nlin.parm0, nlin.parminit repeat;

   /* Get number of parameters */
   t=loc(ndparm=' ');
   if nrow(t)=0 then
   do;
      print 'no parameters';
      stop;
   end;
   _k=t[1] -1;

      /* Trim extra rows, and edit '*' to '#' */
   _dep=nddep; call change(_dep,'*','#',0);
   _fun=ndfun; call change(_fun,'*','#',0);
   _parm=ndparm[1:_k,];
   _beta=ndbeta[1:_k,];
   _der=ndder [1:_k,];
   call change(_der,'*','#',0);


   /* Construct nlresid module to split up parameters and */
   /* compute model                                       */
   call queue('start nlresid;');
   do i=1 to _k;
      call queue(_parm[i] ,"=_beta[",char(i,2),"] ;");
   end;
   call queue("_y = ",_dep,";",
              "_p = ",_fun,";",
              "_r = _y-_p;",
              "_sse = ssq(_r);",
              "finish;" );

      /* Construct nlderiv function */
   call queue('start nlderiv; _x = ');
   do i=1 to _k;
      call queue("(",_der[i] ,")#repeat(1,nobs,1)||");
   end;
   call queue(" nlnothin; finish;");

      /* Pause to compile the functions */
   call queue("resume;");
   pause *;
finish nlgen;  /* Finish module NLGEN   */

   /* Routine to do estimation */
start nlest;

   /*     Modified Gauss-Newton Nonlinear Regression     */
   /* _parm has parm names                               */
   /* _beta has initial values for parameters            */
   /* _k is the number of parameters                     */
   /* after nlresid:                                       */
   /* _y has response,                                     */
   /* _p has predictor after call                          */
   /* _r has residuals                                     */
   /* _sse has sse                                         */
   /* after nlderiv                                        */
   /* _x has jacobian                                      */
   /*                                                      */

   eps=1;
   _iter = 0;
   _subit = 0;
   _error = 0;
   run nlresid;        /* f, r, and sse for initial beta */
   run nliter;                   /* print iteration zero */
   nobs = nrow(_y);
   _msg = 'Iterating';


      /* Gauss-Newton iterations */
   do _iter=1 to 30 while(eps>1e-8);
      run nlderiv;             /* subroutine for derivatives */
      _lastsse=_sse;
      _xpxi=sweep(_x`*_x);
      _delta=_xpxi*_x`*_r;              /* correction vector */
      _old = _beta;              /* save previous parameters */
      _beta=_beta+_delta;            /* apply the correction */
      run nlresid;                       /* compute residual */
      run nliter;               /* print iteration in window */
      eps=abs((_lastsse-_sse))/(_sse+1e-6);
                                    /* convergence criterion */

         /* Hartley subiterations */
      do _subit=1 to 10 while(_sse>_lastsse);
         _delta=_delta*.5;    /* halve the correction vector */
         _beta=_old+_delta;   /* apply the halved correction */
         run nlresid;                     /* find sse et al */
         run nliter;         /* print subiteration in window */
      end;
      if _subit>10 then
      do;
         _msg = "did not improve after 10 halvings";
         eps=0; /* make it fall through iter loop */
      end;
   end;

      /* print out results  */
   _msg = ' ';
   if _iter>30 then
   do;
      _error=1;
      _msg = 'convergence failed';
   end;
   _iter=_iter-1;
```

```
      _dfe = nobs-_k;
      _mse = _sse/_dfe;
      _std = sqrt(vecdiag(_xpxi)#_mse);
      _t = _beta/_std;
      _prob= 1-probf(_t#_t,1,_dfe);
      display nlin.title noinput,
      nlin.model noinput,
      nlin.parm2 noinput,
      nlin.parmest repeat noinput,
      nlin.sse noinput;
   finish nlest;                         /* Finish module NLEST */

        /* Store the modules to run later */
     reset storage='nlin';
     store module=_all_;
```

References

Bishop, Y. M. M., Fienberg, S. E., and Holland, P. W. (1975), *Discrete Multivariate Analysis: Theory and Practice*, Cambridge, MA: MIT Press.

Charnes, A., Frome, E. L., and Yu, P. L. (1976), "The Equivalence of Generalized Least Squares and Maximum Likelihood Estimation in the Exponential Family," *Journal of the American Statistical Association*, 71, 169–172.

Cox, D. R. (1970), *Analysis of Binary Data*, London: Metheun.

Grizzle, J. E., Starmer, C. F., and Koch, G. G. (1969), "Analysis of Categorical Data by Linear Models," *Biometrics*, 25, 489–504.

Hadley, G. (1962), *Linear Programming*, Reading, MA: Addison-Wesley.

Jennrich, R. I. and Moore, R. H. (1975), "Maximum Likelihood Estimation by Means of Nonlinear Least Squares," *American Statistical Association*.

Kaiser, H. F. and Caffrey, J. (1965), "Alpha Factor Analysis," *Psychometrika*, 30, 1–14.

Kastenbaum, M. A. and Lamphiear, D. E. (1959), "Calculation of Chi-Square to Test the No Three-Factor Interaction Hypothesis," *Biometrics*, 15, 107–122.

Nelder, J. A. and Wedderburn, R. W. M. (1972), "Generalized Linear Models," *Journal of the Royal Statistical Society, Series A*, 135, 370–384.