

Chapter 8

General Statistics Examples

Chapter Table of Contents

OVERVIEW	123
GENERAL STATISTICS EXAMPLES	123
Example 8.1 Correlation	123
Example 8.2 Newton's Method for Solving Nonlinear Systems of Equations	124
Example 8.3 Regression	126
Example 8.4 Alpha Factor Analysis	128
Example 8.5 Categorical Linear Models	130
Example 8.6 Regression of Subsets of Variables	134
Example 8.7 Response Surface Methodology	142
Example 8.8 Logistic and Probit Regression for Binary Response Models	144
Example 8.9 Linear Programming	149
Example 8.10 Quadratic Programming	153
Example 8.11 Regression Quantiles	155
Example 8.12 Simulations of a Univariate ARMA Process	159
Example 8.13 Parameter Estimation for a Regression Model with ARMA Errors	161
Example 8.14 Iterative Proportional Fitting	167
Example 8.15 Full-Screen Nonlinear Regression	168

Chapter 8

General Statistics Examples

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 may 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 8.1. Correlation

This example defines modules to compute correlation coefficients between numeric variables and standardized values for a set of data.

```
      /* 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;
x
      /* 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 as follows.

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 8.2. 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 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

$$\begin{aligned}x_1 + x_2 - x_1 x_2 + 2 &= 0 \\x_1 \exp(-x_2) - 1 &= 0\end{aligned}$$

The code is 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.                         */

start newton;
  run fun;          /* evaluate function at starting values */
  do iter=1 to maxiter /* iterate until maxiter iterations */
  while(max(abs(f))>converge); /* 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 as follows.

```

The SAS System

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

                X          F

0.09777731  5.3523E-9
-2.325106  6.1501E-8

```

Example 8.3. Regression

This example shows a regression module that calculates statistics not calculated by the two previous examples:

```

/*          Regression Routine          */
/* Given X, and Y, this fits Y = X B + E */
/* by least squares.                   */

start reg;
  n=nrow(x);          /* number of observations */
  k=ncol(x);         /* number of variables */
  xpx=x`*x;          /* cross-products */
  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 limit for mean */
  lower=yhat-tval#sqrt(h*mse+mse); /* lower limit for indiv */
  upper=yhat+tval#sqrt(h*mse+mse); /* upper limit 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 as follows.

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

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

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

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

F	DFN	DFE	PROB
5.08317	1	5	0.07385

F	DFN	DFE	PROB
666.511	2	5	8.54E-7

F	DFN	DFE	PROB
1.67746	1	5	0.25184

Example 8.4. Alpha Factor Analysis

This example shows how an algorithm for computing alpha factor patterns (Kaiser and Caffrey 1965) is transcribed into IML code.

For later reference, you could store the following ALPHA subroutine in an IML 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      */
/*                                               */

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, */
/* 2nd 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 as follows.

```

M
5.937855
2.0621956
0.1390178
0.0821054
0.018097
-0.047487
-0.09148
-0.100304

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

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 8.5. Categorical Linear Models

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

$$\mathbf{K} \log \pi = \mathbf{X}\beta + e$$

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 β . 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 (refer to Cox 1970).

```

/* 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 */
/* */

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 results are as follows.

```

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

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

```

```

                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

                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

                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
    
```

```

                                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

                                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 8.6. Regression of Subsets of Variables

The following example performs regression with variable selection similar to some of the features in the REG procedure.

```

/*      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 */

/* Save a copy of crossproducts matrix */
csave=(xpx || xpy) // (xpy` || ypy);
finish initial;

/* 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 forward;

/* 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 backward;

/* 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 stepwise;

/* 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 backstep;

/* Search all possible models */
start all;

/* Use method of schatzoff et al. for search technique */
betak=repeat(0,k,k); /* rec. ests. 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/(1|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];
/* Mallows CP plot */
cp=msek#(n-ik`-1)/min(msek)-(n-2#ik`);
cp=ik`||cp;
cpname={"nparm" "cp"};
/* output cp out=cp colname=cpname; */
finish all;

/* 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 ztrail;

/* 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-1-1;
    sse=c[k1,k1];
    mse=sse/dfe;
    rsq=1-sse/ypy;
finish swp;

/* 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//(1||mse||rsq||(c [ik,k1]#in)');
finish bpr;

/*          Stepwise Methods          */
/* After a run 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;         /* forward selection method */
    run backward;        /* backward elimination method */
    run stepwise;        /* stepwise method        */
finish seq;

/*          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 rstpulse runpulse maxpulse};
reset fw=8 linesize=90;
run seq;

```

The results are as follows.

BB									
NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE	
1	26.63425	0.092777	-0.31136	0	0	0	0	0	0
2	25.82619	0.150635	-0.37042	-0.15823	0	0	0	0	0
1	28.58034	0.026488	0	-0.1041	0	0	0	0	0
2	7.755636	0.744935	0	-0.02548	-3.2886	0	0	0	0
3	7.226318	0.770831	-0.17388	-0.05444	-3.14039	0	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0	0
1	7.533843	0.74338	0	0	-3.31056	0	0	0	0
2	7.798261	0.743533	0	0	-3.28661	-0.00968	0	0	0
3	7.336089	0.767349	-0.16755	0	-3.07925	-0.04549	0	0	0
4	7.366649	0.775033	-0.19603	-0.05915	-2.9889	-0.05326	0	0	0
3	8.037314	0.745111	0	-0.02569	-3.26268	-0.01041	0	0	0
2	24.91487	0.180607	0	-0.09305	0	-0.27474	0	0	0
3	20.28031	0.356847	-0.44698	-0.15647	0	-0.32186	0	0	0
2	21.27632	0.30027	-0.38882	0	0	-0.32286	0	0	0
1	24.67582	0.159485	0	0	0	-0.27921	0	0	0
2	23.26003	0.235031	0	0	0	-0.20684	-0.15262	0	0
3	16.81799	0.466648	-0.52338	0	0	-0.22524	-0.23769	0	0
4	16.26146	0.503398	-0.56317	-0.12697	0	-0.22981	-0.2246	0	0
3	23.81815	0.244651	0	-0.06381	0	-0.20843	-0.14279	0	0
4	7.785151	0.762252	0	-0.01231	-3.16759	0.016669	-0.0749	0	0
5	6.213174	0.817556	-0.28528	-0.05184	-2.70392	-0.02711	-0.12628	0	0
4	6.166944	0.81167	-0.26213	0	-2.77733	-0.01981	-0.12874	0	0
3	7.507972	0.761898	0	0	-3.17665	0.017616	-0.07658	0	0
2	7.254263	0.761424	0	0	-3.14019	0	-0.07351	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0	0
4	6.009033	0.816493	-0.27642	-0.04932	-2.77237	0	-0.12932	0	0
3	7.510162	0.761829	0	-0.01315	-3.13261	0	-0.07189	0	0
2	25.333	0.166855	0	-0.05987	0	0	-0.19797	0	0
3	18.63184	0.409126	-0.54408	-0.12049	0	0	-0.28248	0	0
2	18.97378	0.375995	-0.50665	0	0	0	-0.29382	0	0
1	24.70817	0.158383	0	0	0	0	-0.2068	0	0
2	21.60626	0.289419	0	0	0	0	-0.6818	0.571538	0
3	18.21725	0.422273	-0.4214	0	0	0	-0.57966	0.361557	0
4	17.29877	0.47172	-0.45243	-0.14944	0	0	-0.61723	0.426862	0
3	21.41763	0.320779	0	-0.11815	0	0	-0.71745	0.635395	0
4	6.030105	0.815849	0	-0.05159	-2.9255	0	-0.39529	0.38537	0
5	5.176338	0.848002	-0.21962	-0.0723	-2.68252	0	-0.3734	0.304908	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513	0
3	5.991568	0.809988	0	0	-2.97019	0	-0.37511	0.354219	0
4	6.208523	0.8104	0	0	-3.00426	0.016412	-0.37778	0.353998	0
5	5.549941	0.837031	-0.20154	0	-2.7386	-0.01208	-0.34562	0.269064	0
6	5.368247	0.848672	-0.22697	-0.07418	-2.62865	-0.02153	-0.36963	0.303217	0
5	6.263348	0.816083	0	-0.05091	-2.95182	0.01239	-0.39704	0.384793	0
4	20.11235	0.385797	0	-0.1194	0	-0.19092	-0.64584	0.609632	0
5	15.1864	0.554066	-0.47923	-0.1527	0	-0.21555	-0.53045	0.385424	0
4	16.29247	0.502451	-0.44717	0	0	-0.21266	-0.49323	0.319267	0
3	20.37729	0.353772	0	0	0	-0.18993	-0.61019	0.545236	0
2	25.11456	0.174039	0	0	0	-0.25219	0	-0.07364	0
3	19.2347	0.390007	-0.52736	0	0	-0.26492	0	-0.20024	0
4	18.80875	0.425607	-0.55881	-0.12604	0	-0.27056	0	-0.17799	0
3	25.59719	0.188232	0	-0.07874	0	-0.25524	0	-0.05502	0
4	8.311496	0.746179	0	-0.02053	-3.25232	-0.00393	0	-0.02064	0
5	7.19584	0.788701	-0.25795	-0.04936	-2.86147	-0.04121	0	-0.08153	0
4	7.091611	0.783432	-0.23928	0	-2.92597	-0.0339	0	-0.08777	0
3	8.033673	0.745227	0	0	-3.26805	-0.00193	0	-0.02526	0
2	7.746932	0.745221	0	0	-3.27232	0	0	-0.02561	0
3	6.882626	0.78173	-0.22923	0	-3.01222	0	0	-0.09094	0
4	7.00018	0.786224	-0.24436	-0.04525	-2.97011	0	0	-0.08585	0
3	8.00441	0.746155	0	-0.02027	-3.26114	0	0	-0.02139	0
2	28.35356	0.067516	0	-0.07074	0	0	0	-0.12159	0
3	22.38148	0.290212	-0.54076	-0.11605	0	0	0	-0.24445	0
2	22.50135	0.259982	-0.5121	0	0	0	0	-0.2637	0
1	27.71259	0.056046	0	0	0	0	0	-0.13762	0

BPRINT									
NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE	
1	7.533843	0.74338	0	0	-3.31056	0	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513	0
5	5.176338	0.848002	-0.21962	-0.0723	-2.68252	0	-0.3734	0.304908	0
6	5.368247	0.848672	-0.22697	-0.07418	-2.62865	-0.02153	-0.36963	0.303217	0

BPRINT								
NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
1	7.533843	0.74338	0	0	-3.31056	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513

BPRINT								
NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
6	5.368247	0.848672	-0.22697	-0.07418	-2.62865	-0.02153	-0.36963	0.303217
5	5.176338	0.848002	-0.21962	-0.0723	-2.68252	0	-0.3734	0.304908
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513

BPRINT								
NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
1	7.533843	0.74338	0	0	-3.31056	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513

THE BEST MODEL FOR EACH NUMBER OF PARAMETERS

NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
1	7.533843	0.74338	0	0	-3.31056	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513
5	5.176338	0.848002	-0.21962	-0.0723	-2.68252	0	-0.3734	0.304908
6	5.368247	0.848672	-0.22697	-0.07418	-2.62865	-0.02153	-0.36963	0.303217

FORWARD SELECTION METHOD

NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
1	7.533843	0.74338	0	0	-3.31056	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513

BACKWARD ELIMINATION

NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
6	5.368247	0.848672	-0.22697	-0.07418	-2.62865	-0.02153	-0.36963	0.303217
5	5.176338	0.848002	-0.21962	-0.0723	-2.68252	0	-0.3734	0.304908
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513

STEPWISE METHOD

NPARM	MSE	RSQUARE	AGE	WEIGHT	RUNTIME	RSTPULSE	RUNPULSE	MAXPULSE
1	7.533843	0.74338	0	0	-3.31056	0	0	0
2	7.168422	0.764247	-0.15037	0	-3.20395	0	0	0
3	5.956692	0.811094	-0.2564	0	-2.82538	0	-0.13091	0
4	5.343462	0.836818	-0.19773	0	-2.76758	0	-0.34811	0.270513

Example 8.7. 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 may optimize a response. First, the regression is performed for two variables according to the model

$$y = c + b_1x_1 + b_2x_2 + a_{11}x_1^2 + a_{12}x_1x_2 + a_{22}x_2^2 + e.$$

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, then solves for the optimal  */
/* response, prints the optimal factors and response, and */
/* then 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);
  x=j(n,1,1)||d;
  do i=1 to k;
    do j=1 to i;
      x=x||d[,i] #d[,j];
    end;
  end;
  beta=solve(x`*x,x`*y);
  print "Parameter Estimates" , beta;
  c=beta[1];
  b=beta[2:(k+1)];
  a=j(k,k,0);
  L=k+1;
  do i=1 to k;
    do j=1 to i;
      L=L+1;
      a[i,j]=beta [L,];
    end;
  end;
  a=(a+a`)*.5;

```

```

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,vec,a);
print , "Eigenvalues and Eigenvectors", eval, vec;
if min(eval)>0 then print , "Solution Was a Minimum";
if max(eval)<0 then print , "Solution Was a Maximum";
finish rsm;

```

Running the module with the following sample data produces the results shown below.

```

/* 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;

```

```

          BETA
          81.222222
          1.9666667
          0.2166667
          -3.933333
             -2.225
          -1.383333

          XX
          0.2949376
          -0.158881

          YOPT
Response at Critical Value 81.495032

          EVAL
          -0.96621
          -4.350457

          EVEC
          -0.351076 0.9363469
          0.9363469 0.3510761

```

Example 8.8. 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 code 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 with 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 as follows.

```
ITER      LOGLIK
      1 -268.248

      BTRANSP
      0      0      0

ITER      LOGLIK
      2 -76.29481

      BTRANSP
-2.159406 0.0138784 0.0037327

ITER      LOGLIK
      3 -53.38033

      BTRANSP
-3.53344 0.0363154 0.0119734

ITER      LOGLIK
      4 -48.34609

      BTRANSP
-4.748899 0.0640013 0.0299201

ITER      LOGLIK
      5 -47.69191

      BTRANSP
-5.413817 0.0790272 0.04982

ITER      LOGLIK
      6 -47.67283

      BTRANSP
-5.553931 0.0819276 0.0564395

ITER      LOGLIK
      7 -47.67281

      BTRANSP
-5.55916 0.0820307 0.0567708
```

```

ITER      LOGLIK
      8 -47.67281

      BTRANSP
-5.559166 0.0820308 0.0567713

      CHISQ      DF      PROB
      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
      1 -268.248

      BTRANSP
      0      0      0

ITER      LOGLIK
      2 -71.71043

      BTRANSP
-1.353207 0.008697 0.0023391

ITER      LOGLIK
      3 -51.64122

      BTRANSP
-2.053504 0.0202739 0.0073888

ITER      LOGLIK
      4 -47.88947

      BTRANSP
-2.581302 0.032626 0.018503

ITER      LOGLIK
      5 -47.48924

      BTRANSP
-2.838938 0.0387625 0.0309099

```

```

ITER      LOGLIK
      6 -47.47997

```

```

BTRANSP
-2.890129 0.0398894 0.0356507

```

```

ITER      LOGLIK
      7 -47.47995

```

```

BTRANSP
-2.89327 0.0399529 0.0362166

```

```

ITER      LOGLIK
      8 -47.47995

```

```

BTRANSP
-2.893408 0.0399553 0.0362518

```

```

ITER      LOGLIK
      9 -47.47995

```

```

BTRANSP
-2.893415 0.0399554 0.0362537

```

```

ITER      LOGLIK
     10 -47.47995

```

```

BTRANSP
-2.893415 0.0399555 0.0362538

```

```

ITER      LOGLIK
     11 -47.47995

```

```

BTRANSP
-2.893415 0.0399555 0.0362538

```

```

CHISQ      DF      PROB
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

```

ITER	LOGLIK	BTRANSP		
1	-268.248	0	0	0
2	-71.71043	-1.353207	0.008697	0.0023391
3	-51.64122	-2.053504	0.0202739	0.0073888
4	-47.88947	-2.581302	0.032626	0.018503
5	-47.48924	-2.838938	0.0387625	0.0309099
6	-47.47997	-2.890129	0.0398894	0.0356507
7	-47.47995	-2.89327	0.0399529	0.0362166
8	-47.47995	-2.893408	0.0399553	0.0362518
9	-47.47995	-2.893415	0.0399554	0.0362537
10	-47.47995	-2.893415	0.0399555	0.0362538
11	-47.47995	-2.893415	0.0399555	0.0362538

Likelihood Ratio with Intercept-only Model

CHISQ	DF	PROB
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

Example 8.9. Linear Programming

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

$$\begin{aligned} &\max c'x \\ &\text{st. } Ax \leq, =, \geq b \\ &x \geq 0 \end{aligned}$$

A routine written in IML to solve 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*/
/*
/*
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 */
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:];;
dual=y[3:m+2];
print ,'Constraints ' lhs rel rhs dual,
      '*****';

finish;

```

Consider the following product mix example (Hadley 1963). 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.

Machine	Product			
	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 output from this example follows.

		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 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 defining 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.

A 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 as follows.

		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 8.10. 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 \mathbf{H} is positive semidefinite. The approach is outlined in the discussion of the LCP subroutine in Chapter 17, “Language Reference.” 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 following output shows 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.

OBJVAL	
Objective Value	1.0966667
ACTIVITY	
Decision Variables	
ibm	0
dec	0.9333333
dg	0.0666667
prime	0

Example 8.11. Regression Quantiles

The technique of parameter estimation in linear models 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 β in the model

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon$$

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 $S = \{t|y_t \leq x_t b\}$. For $q = 0.5$, the solution \mathbf{b}^* is identical to the estimates produced by the LAE. The following routine finds this estimate 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) * (Xb - y) * ^e */
```

```

/*
/* where e = ( Xb <= y ). When q = .5 this is equivalent
/* to minimizing the sum of the absolute deviations.
/*
/* This subroutine follows the approach given in:
/*
/* Koenker, R. and G. Bassett (1978)
/*
/* Bassett, G. and R. Koenker (1982).
/*
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 */
  r=repeat(0,m+2,1);
  L=repeat(q-1,1,n)||repeat(0,1,m)||-bound||-bound;
  u=repeat(q,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(q-1,1,n)||repeat(0,1,m)||-bound||{0};
  u=repeat(q,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];
  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 output is as follows.

Dependent Variable: pop			
		Q	
Regression Quantile:		0.5	
		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 allow the fit to be better at more points at the expense of allowing some points to fit worse, as the plot of the residuals against the least squares residuals:

```

/* Compare L1 residuals with least squares residuals */
/* Compute the least squares residuals */
resid2=y-x*inv(x'*x)*x'*y;

/* x axis of plot */
xx=repeat(x[,2] ,3,1);

/* y axis of plot */
yy=resid//resid2//repeat(0,19,1);

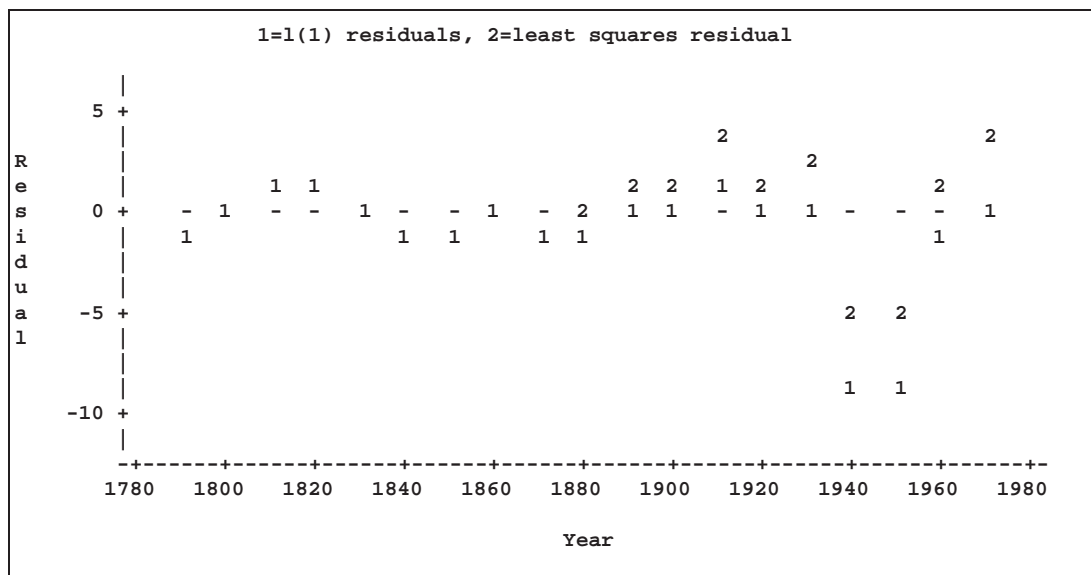
```

```

/* plot character*/
id=repeat('1',19,1)//repeat('2',19,1)//repeat('-',19,1);
call pgraf(xx||yy,id,'Year','Residual',
          '1=L(1) residuals, 2=least squares residual');

```

The output generated is as follows.



Example 8.12. 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 using the IML functions NORMAL, ARMACOV, HANKEL, PRODUCT, RATIO, TOEPLITZ, and ROOT.

```

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 output is as follows.

```

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

```

Example 8.13. 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 programmed in IML in the following example. 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 of the stationary or invertible regions.

```

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, */
/* omitting index 0 corresponding to the zero */
/* order constant one. */
/* itheta - defines indices of nonzero MA parameters, */
/* omitting index 0 corresponding 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 may be             */
/*  modified after running armareg to place estimates              */
/*  in the stationary and invertible regions, and then             */
/*  armareg may 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 may be modified.                                   */
/*          */
/*  T-ratios are only approximate and may 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') [, 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;
end;

```

```

        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);
        if sse<oldsse then
            do;    /* reduction, no iteration */
                lambda=lambda/10;
                k=21;
            end;
    end;

```

```

else do;
                                /* no reduction */
                                /* increase lambda and iterate */
    if nopr^=0 then print ,
        'Lambda=' lambda 'SSE=' sse 'OLDSSE=' oldsse,
        'Gradient Matrix', ss ;
    lambda=10*lambda;
    ss=sssava;
    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:p,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;
  if abs(par[,1] )>1 then par[,1] =-.8;
end;

sighat=sqrt(sse/(nrow(y)-ncol(par)));
print , 'Innovation Standard Deviation:' sighat;
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 */
/* See SAS/ETS User's Guide, Version 5 Edition */

/* Access Grunfeld data as produced by DATA step in */
/* SAS/ETS User's Guide Example */
use grunfeld;
read all var {gei} into y;
read all var {gef gec} into x;
x=j(nrow(x),1,1)||x;
iphi=1;
itheta=1;
maxiter=10;
delta=0.0005;
ml=-1;
run armareg; /* Perform CLS estimation */

```

The output is as follows.

```

      AR 1      MA 1      B 1      B 2      B 3
      0          0 -9.956306 0.0265512 0.1516939

```

```

      Innovation Standard Deviation: 18.639064

```

```

              Estimate Std. Error  T-Ratio
      AR 1 -0.071148  0.3248431 -0.219022
      MA 1  1.2737455  0.2319205  5.4921656
      B 1  -7.530983  20.447977  -0.3683
      B 2  0.0402554  0.0170277  2.3641054
      B 3  0.0992474  0.0354776  2.7974682

```

```

      /* Transform MA coefficient to invertible region */
      par[,2]=1/par[,2];
      maxiter=2;
      run armareg; /* Continue CLS estimation */

```

The output is as follows.

```

      AR 1      MA 1      B 1      B 2      B 3
      -0.071148 0.7850862 -7.530983 0.0402554 0.0992474

```

```

      Innovation Standard Deviation: 22.667286

```

```

              Estimate Std. Error  T-Ratio
      AR 1 -0.191675  0.3360688 -0.570345
      MA 1  0.7367182  0.2101849  3.5050966
      B 1  -19.45436  31.327362 -0.621002
      B 2   0.038099  0.0168731  2.2579666
      B 3   0.121766  0.0433174  2.8110191

```

```

      ml=1;
      maxiter=10;

      /* With CLS estimates as starting values, */
      /* perform ML estimation.                */
      run armareg;

```

The output is as follows.

AR 1	MA 1	B 1	B 2	B 3
-0.071148	1.2737455	-7.530983	0.0402554	0.0992474
Innovation Standard Deviation: 23.039253				
	Estimate	Std. Error	T-Ratio	
AR 1	-0.196298	0.3513019	-0.558773	
MA 1	1.467349	0.5851509	2.5076422	
B 1	-26.47256	33.752651	-0.784311	
B 2	0.03922	0.0165545	2.369142	
B 3	0.1310302	0.0426012	3.0757423	

Example 8.14. 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 example is a census study that estimates a population distribution according to age and marital status (Bishop, Fienberg, and Holland 1975, pp. 97-98). 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.

```
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 output is as follows.

	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

	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 8.15. 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.


```

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 pop;
run nlin;

```

A menu appears like the one below. The entry fields are shown by underscores here, but the underscores become blanks in the real session.

```

Non-Linear 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.

```

Non-Linear 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 command

```
run nlrn;
```

Here is the code 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 nlnit;
  window nlin rows=15 columns=80 color='green'
  msgline=_msg cmdndline=_cmdnd
  group=title +30 'Non-Linear 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 nlininit; /* Finish module NLINIT */

      /* Main routine */
start nlin;
      run nlininit; /* initialization routine */
      run nlrn; /* 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 nlrn;
      run nlgen; /* generate the model */
      run nlest; /* estimate the model */
finish nlrn;

      /* 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_;

```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/IML User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. 846 pp.

SAS/IML User's Guide, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1-58025-553-1

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software by the government is subject to restrictions as set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The Institute is a private company devoted to the support and further development of its software and related services.