Chapter 3

Tutorial: A Module for Linear Regression

Chapter Table of Contents

Chapter 3

Tutorial: A Module for Linear Regression

Overview

SAS/IML software makes it possible for you to solve mathematical problems or implement new statistical techniques and algorithms. The language is patterned after linear algebra notation. For example, the least-squares formula familiar to statisticians

$$B = (X'X)^{-1}X'Y$$

can be easily translated into the Interactive Matrix Language statement

```
b=inv(x‘*x)*x‘*y;
```

This is an example of an assignment statement that uses a built-in function (INV) and operators (transpose and matrix multiplication).

If a statistical method has not been implemented directly in a SAS procedure, you may be able to program it using IML. Because the operations in IML deal with arrays of numbers rather than with one number at a time, and the most commonly used mathematical and matrix operations are built directly into the language, programs that take hundreds of lines of code in other languages often take only a few lines in IML.

Solving a System of Equations

Because IML is built around traditional matrix algebra notation, it is often possible to directly translate mathematical methods from matrix algebraic expressions into executable IML statements. For example, consider the problem of solving three simultaneous equations:

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 8 \\ 2x_1 - 2x_2 + 3x_3 &= 2 \\ 4x_1 + x_2 - 4x_3 &= 9 \end{aligned}$$

These equations can be written in matrix form as

$$\begin{bmatrix} 3 & -1 & 2 \\ 2 & -2 & 3 \\ 4 & 1 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \\ 9 \end{bmatrix}$$

and can be expressed symbolically as

$$\mathbf{A}\mathbf{x} = \mathbf{c}$$

Because $\mathbf{A}$ is nonsingular, the system has a solution given by

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{c}$$

In the following example, you solve this system of equations using an interactive session. Submit the PROC IML statement to begin the procedure. Throughout this chapter, the right angle brackets (>) indicate statements you submit; responses from IML follow:

```
proc iml;

IML Ready
```

Enter

```
reset print;
```

The PRINT option of the RESET command causes automatic printing of results. Notice that as you submit each statement, it is executed and the results are displayed. While you are learning IML or developing modules, it is a good idea to have all results printed automatically. Once you are familiar with SAS/IML software, you will not need to use automatic printing.

Next, set up the matrices $\mathbf{A}$ and $\mathbf{c}$. Both of these matrices are input as matrix literals; that is, input the row and column values as discussed in Chapter 2, “Understanding the Language.”

```
> a={3  -1  2,
>    2  -2  3,
>    4   1 -4};

            A             3 rows      3 cols    (numeric)

                             3        -1         2
                             2        -2         3
                             4         1        -4

> c={8, 2, 9};

             C             3 rows      1 col     (numeric)

                                        8
                                        2
                                        9
```

Now write the solution equation, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{c}$, as an IML statement. The appropriate statement is an assignment statement that uses a built-in function and an operator (INV is a built-in function that takes the inverse of a square matrix, and * is the operator for matrix multiplication).

```
> x=inv(a)*c;

           X             3 rows      1 col     (numeric)

                                       3
                                       5
                                       2
```

After IML executes the statement, the first row of matrix $\mathbf{X}$ contains the x_1 value for which you are solving, the second row contains the x_2 value, and the third row contains the x_3 value.

Now end the session by entering the QUIT command.

```
>   quit;

    Exiting IML
```

A Module for Linear Regression

The previous method may be more familiar to statisticians when different notation is used. A linear model is usually written

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e}$$

where $\mathbf{y}$ is the vector of responses, $\mathbf{X}$ is the design matrix, and $\mathbf{b}$ is a vector of unknown parameters estimated by minimizing the sum of squares of $\mathbf{e}$, the error or residual.

The following example illustrates the programming techniques involved in performing linear regression. It is not meant to replace regression procedures such as the REG procedure, which are more efficient for regressions and offer a multitude of diagnostic options.

Suppose that you have response data $\mathbf{y}$ measured at five values of the independent variable $\mathbf{x}$ and you want to perform a quadratic regression.

Submit the PROC IML statement to begin the procedure.

```
> proc iml;

   IML Ready
```

Input the design matrix $\mathbf{X}$ and the data vector $\mathbf{y}$ as matrix literals.

```
> x={1 1 1,
>    1 2 4,
>    1 3 9,
>    1 4 16,
>    1 5 25};


           X           5 rows      3 cols    (numeric)

                         1         1          1
                         1         2          4
                         1         3          9
                         1         4         16
                         1         5         25


> y={1,5,9,23,36};

           Y           5 rows      1 col     (numeric)

                                   1
                                   5
                                   9
                                  23
                                  36
```

Compute the least-squares estimate of $\mathbf{b}$ using the traditional formula.

```
> b=inv(x\baccent *x)*x\baccent *y;

           B             3 rows      1 col     (numeric)

                                     2.4
                                    -3.2
                                       2
```

The predicted values are simply the $\mathbf{X}$ matrix multiplied by the parameter estimates, and the residuals are the difference between actual and predicted $\mathbf{y}$.

```
> yhat=x*b;

           YHAT          5 rows      1 col     (numeric)

                                     1.2
                                       4
                                    10.8
                                    21.6
                                    36.4

> r=y-yhat;

           R           5 rows      1 col     (numeric)
```

```
                                        -0.2
                                           1
                                        -1.8
                                         1.4
                                        -0.4
```

To calculate the estimate of the variance of the responses, calculate the sum of squared errors (SSE), its degrees of freedom (DFE), and the mean squared error (MSE). Note that in computing the degrees, you use the function NCOL to return the number of columns of $\mathbf{X}$.

```
> sse=ssq(r);

           SSE           1 row       1 col     (numeric)

                                     6.4

> dfe=nrow(x)-ncol(x);

           DFE           1 row       1 col     (numeric)

                                     2


> mse=sse/dfe;

           MSE           1 row       1 col     (numeric)

                                    3.2
```

Notice that each calculation has required one simple line of code.

Now suppose you want to solve the problem repeatedly on new data sets without reentering the code. To do this, define a module (or subroutine). Modules begin with a START statement and end with a FINISH statement, with the program statements in between. The following statements define a module named REGRESS to perform linear regression.

```
> start regress;                  /* begin module */
>    xpxi=inv(t(x)*x);         /* inverse of X'X          */
>    beta=xpxi*(t(x)*y);       /* parameter estimate      */
>    yhat=x*beta;              /* predicted values        */
>    resid=y-yhat;                 /* residuals           */
>    sse=ssq(resid);               /* SSE                 */
>    n=nrow(x);                    /* sample size         */
>    dfe=nrow(x)-ncol(x);          /* error DF            */
>    mse=sse/dfe;                  /* MSE                 */
>    cssy=ssq(y-sum(y)/n);         /* corrected total SS  */
>    rsquare=(cssy-sse)/cssy;      /* RSQUARE             */
>    print,"Regression Results",
>       sse dfe mse rsquare;
>    stdb=sqrt(vecdiag(xpxi)*mse); /* std of estimates    */
>    t=beta/stdb;                  /* parameter t-tests   */
>    prob=1-probf(t#t,1,dfe);      /* p-values            */
>    print,"Parameter Estimates",,
>       beta stdb t prob;
>    print,y yhat resid;
> finish regress;                  /* end module          */
```

Submit the module REGRESS for execution.

```
> reset noprint;
> run regress;                     /* execute module      */


                             Regression Results


                     SSE       DFE       MSE    RSQUARE
                     6.4         2       3.2  0.9923518

                            Parameter Estimates


                    BETA       STDB          T       PROB
                     2.4  3.8366652  0.6255432  0.5954801
                    -3.2  2.9237940  -1.094468  0.3879690
                       2  0.4780914  4.1833001  0.0526691

                           Y      YHAT     RESID
                           1       1.2      -0.2
                           5         4         1
                           9      10.8      -1.8
                          23      21.6       1.4
                          36      36.4      -0.4
```

At this point, you still have all of the matrices defined if you want to continue calculations. Suppose that you want to correlate the estimates. First, calculate the covariance estimate of the estimates; then, scale the covariance into a correlation matrix with values of 1 on the diagonal.

```
> reset print;             /* turn on auto printing   */
> covb=xpxi*mse;           /* covariance of estimates */

           COVB          3 rows      3 cols    (numeric)

                     14.72     -10.56        1.6
                    -10.56  8.5485714  -1.371429
                       1.6  -1.371429  0.2285714

> s=1/sqrt(vecdiag(covb));

           S             3 rows      1 col     (numeric)
```

```
                                   0.260643
                                  0.3420214
                                  2.0916501
```

```
   > corrb=diag(s)*covb*diag(s);   /* correlation of estimates */

              CORRB         3 rows      3 cols    (numeric)

                                  1  -0.941376  0.8722784
                          -0.941376          1  -0.981105
                          0.8722784  -0.981105          1
```

Your module REGRESS remains available to do another regression, in this case, an orthogonalized version of the last polynomial example. In general, the columns of $\mathbf{X}$ will not be orthogonal. You can use the ORPOL function to generate orthogonal polynomials for the regression. Using them provides greater computing accuracy and reduced computing times. When using orthogonal polynomial regression, you expect the statistics of fit to be the same and the estimates to be more stable and uncorrelated.

To perform an orthogonal regression on the data, you must first create a vector containing the values of the independent variable x, which is the second column of the design matrix $\mathbf{X}$. Then, use the ORPOL function to generate orthogonal second degree polynomials.

```
   > x1={1,2,3,4,5};               /* second column of X */

              X1            5 rows      1 col     (numeric)

                                        1
                                        2
                                        3
                                        4
                                        5


   > x=orpol(x1,2);    /* generates orthogonal polynomials */

              X             5 rows      3 cols    (numeric)

                          0.4472136  -0.632456  0.5345225
                          0.4472136  -0.316228  -0.267261
                          0.4472136          0  -0.534522
                          0.4472136  0.3162278  -0.267261
                          0.4472136  0.6324555  0.5345225

   > reset noprint;           /* turns off auto printing   */
   > run regress;             /* run REGRESS */
```

```
                           Regression Results


                      SSE       DFE       MSE   RSQUARE
                      6.4         2       3.2 0.9923518




                           Parameter Estimates


                     BETA      STDB         T      PROB
                33.093806 1.7888544      18.5 0.0029091
                27.828043 1.7888544 15.556349 0.0041068
                7.4833148 1.7888544 4.1833001 0.0526691

                              Y      YHAT     RESID
                              1       1.2      -0.2
                              5         4         1
                              9      10.8      -1.8
                             23      21.6       1.4
                             36      36.4      -0.4

  > reset print;
  > covb=xpxi*mse;

              COVB          3 rows      3 cols    (numeric)

                             3.2 -2.73E-17 4.693E-16
                       -2.73E-17       3.2 -2.18E-15
                       4.693E-16 -2.18E-15       3.2


  > s=1/sqrt(vecdiag(covb));

              S             3 rows      1 col     (numeric)

                                  0.559017
                                  0.559017
                                  0.559017


  > corrb=diag(s)*covb*diag(s);

              CORRB         3 rows      3 cols    (numeric)

                               1  -8.54E-18  1.467E-16
                       -8.54E-18          1   -6.8E-16
                       1.467E-16   -6.8E-16          1
```

Note that the values on the off-diagonal are displayed in scientific notation; the values are close to zero but not exactly zero because of the imprecision of floating-point arithmetic. To clean up the appearance of the correlation matrix, use the FUZZ option.

```
   > reset fuzz;
   > corrb=diag(s)*covb*diag(s);

               CORRB          3 rows      3 cols    (numeric)

                                 1         0         0
                                 0         1         0
                                 0         0         1
```

Plotting Regression Results

You can create some simple plots by using the PGRAF subroutine. The PGRAF subroutine produces scatter plots suitable for printing on a line printer. If you want to produce better quality graphics using color, you can use the graphics capabilities of IML (see Chapter 12, “Graphics Examples,” for more information).

Here is how you can plot the residuals against **x**. First, create a matrix containing the pairs of points by concatenating **X1** with **RESID** using the horizontal concatenation operator (||).

```
    > xy=x1||resid;

               XY             5 rows      2 cols    (numeric)

                                      1      -0.2
                                      2         1
                                      3      -1.8
                                      4       1.4
                                      5      -0.4
```

Next, use a CALL statement to call the PGRAF subroutine to produce the desired plot. The arguments to PGRAF are, in order,

- the matrix containing the pairs of points
- a plotting symbol
- a label for the X-axis
- a label for the Y-axis
- a title for the plot

```
    > call pgraf(xy,'r','x','Residuals','Plot of Residuals');

                             Plot of Residuals
   2 +
     |
     |                                                 r
R    |
e    |                     r
s    |
i    |
d  0 +
u    |       r                                                       r
a    |
l    |
s    |
     |
     |                                   r
  -2 +
     --------+------+------+------+------+------+------+------+------+--------
            1.0    1.5    2.0    2.5    3.0    3.5    4.0    4.5    5.0
                                         x
```

You can also plot the predicted values $\hat{\mathbf{y}}$ against $\mathbf{x}$. You must first create a matrix, say **XYH**, containing the points. Do this by concatenating **X1** with **YHAT**. Next, call the PGRAF subroutine to plot the points.

```
   > xyh=x1||yhat;

               XYH           5 rows      2 cols    (numeric)

                                     1       1.2
                                     2         4
                                     3      10.8
                                     4      21.6
                                     5      36.4

   > call pgraf(xyh,'*','x','Predicted','Plot of Predicted Values');
```

```
                          Plot of Predicted Values
  40 +
     |                                                               *
     |
P    |
r    |
e    |
d    |                                                 *
i 20 +
c    |
t    |
e    |                                   *
d    |
     |
     |                     *
   0 +       *
     --------+------+------+------+------+------+------+------+------+--------
            1.0    1.5    2.0    2.5    3.0    3.5    4.0    4.5    5.0
                                         x
```

You can get a more detailed plot, denoting the observed values with a “y” and the predicted values with a “p” using the following statements. Create a matrix **NEWXY** containing the pairs of points to overlay. You need to use both the horizontal concatenation operator (||) and the vertical concatenation operator (//). The NROW function returns the number of observations, that is, the number of rows of **X1**. The matrix **LABEL** contains the character label for each point, plotting a “y” for each observed point and a “p” for each predicted point.

```
> newxy=(x1//x1)||(y//yhat);

            NEWXY        10 rows      2 cols    (numeric)

                                     1         1
                                     2         5
                                     3         9
                                     4        23
                                     5        36
                                     1       1.2
                                     2         4
                                     3      10.8
                                     4      21.6
                                     5      36.4

> n=nrow(x1);


            N             1 row       1 col     (numeric)

                                       5


> label=repeat('y',n,1)//repeat('p',n,1);
```

```
               LABEL        10 rows      1 col     (character, size 1)

                                         y
                                         y
                                         y
                                         y
                                         y
                                         p
                                         p
                                         p
                                         p
                                         p

   > call pgraf(newxy,label,'x','y','Scatter Plot with Regression Line' );

                     Scatter Plot with Regression Line
y 40 +
     |                                                               y
     |
     |
     |
     |
     |                                                 y
  20 +
     |
     |
     |                                   p
     |                                   y
     |                     y
     |                     p
   0 +       y
     --------+------+------+------+------+------+------+------+------+----
            1.0    1.5    2.0    2.5    3.0    3.5    4.0    4.5    5.0
                                         x
```

As you can see, the observed and predicted values are too close together to be able to distinguish them at all values of **x**.

Summary

In this chapter, you have seen the programming techniques necessary for solving systems of equations. You have seen how to define a module for performing linear regression and obtaining covariance and correlation matrices, and how to obtain some simple diagnostic plots. Many of the ideas presented in Chapter 2, "Understanding the Language," such as the use of assignment statements, functions, CALL statements, and subscripting have been demonstrated.

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

ISBN 1–58025–553–1

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

1st printing, October 1999