Chapter 13
MATRIX COMPUTATIONS

Chapter 6 mentioned that one of the TSP variable types available is the matrix. This chapter shows how to create matrices, combine them with other data in your TSP program, and perform computations on them. The advantage of dealing directly with matrices in TSP is that it facilitates computation of estimators not provided as "canned" procedures within TSP. We give some examples of these estimators later in this chapter.

Here is a simple application involving matrices that illustrates how they can be used:

```
OLSQ INV1 C F1 K1 ;
COPY @RES E1 ;
OLSQ INV2 C F2 K2 ;
COPY @RES E2 ;
MMAKE E E1 E2 ;
MAT RESCOV = (E'E)/@NOB ;
PRINT RESCOV ;
```

In this example, we form an estimate of a asymptotic residual covariance matrix for a two-equation model estimated by ordinary least squares. E1 and E2 contain the residual series for each equation; each can be thought of as a T (where T is the number of observations) by 1 vector. MMAKE combines the two series to make a T by 2 matrix:

```
E = [ E1 E2 ]
```

The next command, MAT, is a matrix computation command. It calculates the 2 by 2 matrix RESCOV as the cross-product matrix of the residual matrix E divided by the number of observations in the regressions. This matrix is an estimate of R, the covariance of the disturbances in the two equations. The result is displayed by PRINT; PRINT can always be used for matrices as well as for variables of any other type in TSP.

This simple example introduces combining matrix operations with other information in your TSP program. Before giving any more examples, we give an overview of the matrix facilities available in TSP: the types of matrices, creation of matrices in your program, and the matrix computation commands themselves.

13.1. Matrix formats

To economize on matrix storage and computation, TSP stores matrices of several different types in different ways, and uses only the unique and non-zero elements of a matrix to save on computations. For the most part, you need not be aware of this; the program takes care of it for you. When you load matrices, however, it is sometimes helpful to label the matrix type to save data entry. You will also notice that the format of the printed output of matrices will vary by matrix type. The matrix types recognized by TSP are the following:

1. **GENERAL** -- A general matrix may have any rectangular form (any number of rows and columns). Any matrix can be loaded or used as a general matrix, even one that has a special form. A vector has only one row or only one column. Some matrix operations allow series to be used as vectors (as long as the number of observations in the current sample conforms to other matrices in the procedure). Scalars are also treated as special cases of general matrices with number of rows and columns equal to one when they are used in matrix procedures.

2. **TRIANG** -- A triangular matrix contains meaningful elements on and above the diagonal and zeroes below the diagonal. TSP stores only the diagonal and above diagonal elements. This corresponds to an upper triangular matrix; if you wish a lower triangular matrix, use the transpose of this matrix in all your operations. When a triangular matrix is printed out, elements below the diagonal are blank.

3. **SYM** -- A symmetric matrix has the same elements above the diagonal as below the diagonal. TSP stores only the elements below the diagonal and expands the matrix before it is used. When the matrix is printed out, the elements

above the diagonal are blank.

4. **DIAG --** A diagonal matrix is any square matrix with nonzero or zero elements on the diagonal and zero elements everywhere off the diagonal. Diagonal matrices are obviously also triangular and symmetric. TSP stores only the diagonal of this type of matrix, and expands the matrix before it is used. It can be created or reformed from a vector (the diag operation from matrix algebra) using the procedure MFORM or the matrix function DIAG( ). If gamma is a vector of length 5, an example of forming the matrix diag(gamma) is:

```
    MFORM(NROW=5,TYPE=DIAG) DG=GAMMA ;
```

or

```
    MAT DG = DIAG(GAMMA) ;
```

The resulting 5 by 5 diagonal matrix DG has zeros everywhere except along the main diagonal, where DG(1,1)=GAMMA(1), DG(2,2)=GAMMA(2), and so forth.

An identity matrix may also be formed in this way, but it is more easily formed with the IDENT() function:

```
    MAT ID = IDENT(5) ;
```

13.2. Creating a matrix

There are three ways to create a matrix in a TSP program: read it in with your input data, obtain it as the result of a TSP procedure, or form it from a group of time series or other data. The next three sections describe each of these methods.

13.2.1. Reading matrices: READ

Matrices can be read just like series with two important differences: the current SMPL does not apply to them and options on the READ command tell the program the matrix's format. These options are the matrix type (TYPE=GENERAL, SYMMETRI, TRIANG, or DIAG), number of rows (NROWS=), number of columns (NCOL=), and whether the full matrix or just the significant elements are being loaded (FULL/NOFULL).

Here is an example of reading a general matrix:

```
READ (NROW=4,NCOL=3) COEFMAT ;
.32 0.5 1.3
.30 0.4 1.35
.25 0.61 1.1
.28 .55 1.23
;
```

The data should consist of all the elements of the first row followed by all the elements of the second row, and so on, with a ";" at the end of the last column. In this example, we entered each row of the matrix on a separate line to make it easy to read and to check, but this is not required.

The next two examples of reading a symmetric matrix produce the same result; they demonstrate the FULL option.

```
READ (NROW=3,TYPE=SYM) COVAR ;
4.6
2.3 5.1
0.9 2.1 3.9 ;

READ (NROW=3,TYPE=SYM,FULL) COVAR ;
4.6 2.3 0.9
2.3 5.1 2.1
0.9 2.1 3.9;
```

The primary use of the FULL option is when the matrix is in full format from another source, and you do not want to remove the redundant elements.

Matrices do not have to be read in free format, as they were in these examples; you can use any of the formats described for the READ command in the *Reference Manual* or in Chapters 3 and 16 of this manual.

13.2.2. Matrix results from TSP procedures: COPY

The second way to create a matrix is to obtain it from the results of a matrix procedure or a statistical procedure. For example, to use the estimated covariance matrix from an estimation procedure later in your TSP program, the command

```
COPY @VCOV Q;
```

will save the matrix for later use under the name Q.

All results which can be obtained this way are listed in the *Reference Manual* under the Output Section for each procedure. The exact dimensions and type of matrices concerned are also given. Most of the results available are printed by the estimation procedures (OLSQ, 2SLS, LIML, AR1, LSQ, FIML, etc.) and are only available for use until the next estimation procedure, since the same temporary names (beginning with @) are reused for all procedures.

Some of the standard matrix results stored after estimation procedures are:

@COEF	the vector of coefficient estimates
@SES	the vector of standard errors of the coefficients
@VCOV	the estimated variance-covariance matrix
@VCOR	the estimated correlation matrix corresponding to @VCOV (if REGOPT(CALC) VCOR; is in effect)
@LAGF	the vector of estimated lag coefficients (if a PDL variable was used)

All the results below are stored as vectors rather than scalars if a multi-equation model was estimated, so they may be manipulated as matrices:

@SSR	the sum of squared residuals
@S2	the estimated variance of the residuals
@S	the standard error of the residuals
@RSQ	the R-squared
@DW	the Durbin-Watson statistic
@YMEAN	the mean of the dependent variable
@SDEV	the standard deviation of the dependent variable

These additional results are stored as scalars for the single equation models only:

@FST	the F-statistic
@ARSQ	the adjusted R-squared or R-bar-squared
@DH	Durbin's h statistic (for a lagged dependent variable)
@DHALT	Durbin's alternate statistic (for lagged dep. variables)
@LOGL	Log of likelihood function
@AIC	Akaike Information Criterion
@SBIC	Schwarz Bayesian Information Criterion
@LMHET	Lagrange multiplier test for heteroskedasticity
@RESET2	Ramsey's RESET test of functional form
@JB	Jarque-Bera statistic

Residuals @RES and fitted values @FIT are also stored as matrices (rather than series) if a multi-equation model is being estimated. The number of rows is equal to the number of observations and the number of columns is equal to the number of equations.

Note: LSQ, GMM, FIML, etc. do not store @YMEAN, @SDEV, @FIT, @RSQ, etc. for unnormalized equations (those without explicit dependent variables).

VAR, LSQ, GMM, 3SLS, FIML, etc. also store:

@COVU	the covariance matrix of the residuals.

Most nonlinear procedures (LSQ, FIML, PROBIT, ML, etc.) store:

@IFCONV	convergence flag (1=converged, 0=not converged)
@GRAD	gradient vector w.r.t. the parameters at convergence

The procedures for simple statistics, MSD, CORR, COVA, and MOM, also store their results as matrices (see the *Reference Manual* for details).

13.2.3. Making a matrix from series, scalars, or other matrices: MMAKE, UNMAKE

The third way to create a matrix is from a set of time series or scalars. MMAKE makes each series into a column of the matrix. The name of the matrix is followed by the names of the series that go into the matrix. The current SMPL controls the selection of observations from the series. For example, if these series were loaded,

```
SMPL 1,3;
LOAD X; 1,2,3;
LOAD Y; 4,5,6;
LOAD Z; 7,8,9;
```

then the program could have

```
MMAKE M X,Y,Z;
```

which would create the matrix M shown below:

$$M = \begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$$

To stack the three series into a 9x1 vector, you could use MFORM to reshape M:

```
MFORM(NROW=9,NCOL=1) S=M ;
```

Alternatively, you could use the VERT option on MMAKE:

```
MMAKE(VERT) S X,Y,Z ;
```

The inverse of MMAKE is UNMAKE. To store the columns of a matrix as individual series, use UNMAKE followed by the name of a matrix and then the names for the new series. The operation must be conformable; that is, the number of rows in the matrix must be equal to the number of observations in the *current* SMPL and the number of columns must be equal to the number of series.

Below is an example that undoes the previous MMAKE:

```
UNMAKE M,A,B,Q;
```

After execution, A,B, and Q are time series that are identical to the original X, Y and Z.

To create a stacked series T:

```
SMPL 1, 9 ;
UNMAKE S T ;
```

MMAKE and UNMAKE also operate on scalar variables; in this case the first argument is a vector. This is useful for operations on starting values or parameter estimates. For example,

```
OLSQ Y C X1-X10;
PARAM B0-B10;
UNMAKE @COEF B0-B10;
```

places the OLS estimates of the coefficients into the scalars B0, B1, B2, ... for further use.

MMAKE can be used to concatenate matrices. For example, suppose that you have matrices Z1 (3 by 2), ZIT (2 by 3), and Z2 (3 by 3). The command

```
MMAKE NEWZ Z1 Z2 ;
```

makes the 3 by 5 matrix

NEWZ = [Z1 Z2]

The command

```
MMAKE(VERT) NEWZT ZIT Z2 ;
```

makes the 5 by 3 matrix

$$\text{NEWZT} = \begin{bmatrix} \text{ZIT} \\ \text{Z2} \end{bmatrix}$$

13.2.4. Making a matrix from other matrices: MFORM

We have already given a few examples of the use of MFORM to copy or reformat matrices. It can also be used to change the type of matrix (from symmetric to general, for example):

```
MFORM(TYPE=GEN) YMAT ;
```

The above example takes YMAT, a symmetric matrix stored as a lower triangle, and fills out the elements above the diagonal in the appropriate way, storing the matrix under the same name. To change its name to GMAT at the same time, use

```
MFORM(TYPE=GEN) GMAT = YMAT ;
```

Note: The above statement is equivalent to:

```
MAT YMAT = GEN (YMAT) ;
MAT GMAT = GEN (YMAT) ;
```

MFORM can also be used to create block diagonal matrices from other matrices; for example, the command

```
MFORM(BLOCK) NEWMAT = OLDMAT1 OLDMAT2 OLDMAT3 ;
```

forms the matrix NEWMAT from three matrices (all of which are normally square matrices, but need not be) using the block diagonal format. If OLDMAT1 is 2 by 2, OLDMAT2 is 3 by 3, and OLDMAT3 is 4 by 4, the new matrix NEWMAT will be a 9 by 9 general matrix, with zeroes in the appropriate locations.

13.3. Matrix algebra: MAT

In TSP you can perform algebraic operations with matrices just the way you can with series, using the same kind of easily specified formulas or equations. For example, the command

```
MAT B = (X'X)"X'Y ;
```

computes a vector of regression coefficients and stores them in B, given a matrix of data for the regressands in X and a vector of data for the regressor in Y. Note the use of the operator ' (apostrophe) to specify the transpose of X, and the operator " (quotation mark) to specify the operation of matrix inversion. Of course, you are unlikely to use this particular matrix equation, since TSP performs this operation automatically when you specify an OLSQ command, which uses a more accurate algorithm.

In the next few sections, we describe in more detail how to perform calculations using matrices. Matrix operations basically fall into three classes: ordinary operators (multiplication, addition, etc.), functions of matrices that yield scalars or matrices as output, and matrix procedures which have two or more arguments. Only the latter cannot be included in matrix equations specified in the MAT command. We begin with the operations and functions that can be used with a MAT statement, and then describe the stand-alone procedures.

13.3.1. MAT command and matrix operations

TSP provides a variety of operations on matrices, including multiplication, inversion, factorization, calculation of eigenvectors, and eigenvalues, and so forth. All these operations may be specified in matrix equations preceded by the word MAT; these equations are just like the variable transformations performed by GENR, except for two things: they do not operate under control of the current SMPL and the result from MAT equations is stored as a matrix. The MAT procedure checks the matrices for conformability of the operations and gives an error message if the operation specified is not possible. Often printing the matrices in question will reveal why the operation cannot be performed.

Appendix A gives a complete list of TSP operators and what they mean. In this section we list these operators and how they are interpreted in the MAT command. All the ordinary operators and functions used in TSP equations can also be used in the MAT command. They operate on an element-by-element basis (and hence require conforming matrices if they are binary operators). There is one important exception to this, the multiply operator * . For simplicity, this operator denotes the usual matrix multiplication, and element-by-element multiplication (the Hadamard product) is denoted by the operator % .

In the descriptions of the matrix operators that follow, we use the following symbols to denote the inputs and outputs of operations:

s	=	scalar or subscript variable.
i	=	integer scalar.
m	=	any matrix (if scalar, treated as 1 by 1 matrix).
qm	=	square matrix, N by N.
sm	=	symmetric matrix, assumed positive semi-definite.
dm	=	diagonal matrix, assumed positive semi-definite.
tm	=	upper-triangular matrix, assumed positive semi-definite.
v	=	column vector, N by 1.

Here are the symbolic operators understood by the MAT command (in addition to the ordinary operators used also in GENR). Remember that the operands *must* be conformable for the operations that you request; TSP will check the dimensions for you and refuse to perform the computation if this condition is violated.

*m = m*m*	matrix product

*m = m*s*	scalar multiplication (or s*m)
m = m'	matrix transpose
m = m'm	matrix transpose with implied matrix product
m = qm"	matrix inverse (also causes @DET, the determinant of qm, to be stored)
m = qm"m	matrix inverse with implied matrix product
m = m#m	Kronecker product (⊗)
m = m%m	Hadamard product (element by element)

When TSP processes a MAT command, it recognizes several operations where great savings of computation time can be made by eliminating duplicate calculations. These situations include, but are not limited to, the cross-product operation (which generates a symmetric matrix) and the calculation of a quadratic form (the expression A*B*A'). This occurs even when the arguments to these expressions are complicated expressions themselves. Thus, you should be careful to express any such complex arguments in the same way whenever they appear in the matrix expression. For example, the following matrix equation computes the well-known test for a set of linear restrictions on estimated regression coefficients (H_0: Rb = b_0):

```
MAT CHI = (R*B-B0)'(sigsq*(R*(X'X)"R'))"(R*B-B0) ;
```

Both quadratic forms in this expression will be recognized and computed as such, and the expression R*B-B0 will be computed only once.

13.3.2. Matrix functions with scalar output

The following functions take matrices as their input and produce scalars as output:

s = DET*(qm)*	determinant (truncated to zero when <1.E-37)
s = LOGDET*(qm)*	log of (positive) determinant, no truncation
s = TR*(qm)*	trace (sum of diagonal elements)
s = MIN*(m)*	element with minimum value
s = MAX*(m)*	element with maximum value
s = SUM*(m)*	sum of elements
i = NROW*(m)*	number of rows
i = NCOL*(m)*	number of columns
i = RANK*(m)*	rank (number of linearly independent columns or rows)

These functions may be used anywhere in a MAT statement where scalars are allowed, keeping in mind that a scalar is also a 1 by 1 matrix.

13.3.3. Matrix functions with matrix output

The following functions are matrix-to-matrix; that is, they take a matrix, perform some computation on it, and produce another matrix as output:

tm	=	CHOL*(sm)*	Choleski factorization (matrix square root)
sm	=	YINV*(sm)*	Positive semi-definite inverse using CHOL()
dm	=	IDENT*(i)*	Creates an identity matrix of order *i*
v	=	EIGVAL*(qm)*	Computes the vector of eigenvalues of *qm.* If *qm* is not symmetric positive semi-definite, the imaginary parts of the eigenvalues are stored as @EIGVALI
qm	=	EIGVEC*(qm)*	Computes the matrix of eigenvectors (columns). If *qm* is not symmetric positive semi-definite, the imaginary parts of the eigenvectors are stored as @EIGVECI. @EIGVAL and @EIGVALI will also be stored automatically. EIGVAL and the corresponding EIGVEC are sorted high to low (by the name of the the eigenvalue if complex).
v	=	VEC*(m)*	Creates a vector of all the elements of *m*, column by column
v	=	VECH*(m)*	Creates a vector of all the unique elements of *m* column by column: *qm*: N*N elements

			sm, tm: N*(N+1)/2 elements
			dm: N elements
dm	=	DIAG*(m)*	Creates a diagonal matrix from a matrix
			qm, sm, tm: take the diagonal from input matrix
			v: convert the vector to a diagonal matrix
			s: illegal, use s*IDENT(*i*) to create a diagonal matrix with s on the diagonal
sm	=	SYM*(qm)*	Creates a symmetric matrix from a square matrix (the upper triangular elements are ignored)
m	=	GEN*(qm)*	Creates a general matrix from a symmetric or diagonal matrix

These functions can also be used anywhere in a MAT equation.

13.3.4. Matrix procedures: ORTHON, YLDFAC

Several matrix procedures have more than one output and are less convenient for use in MAT. These procedures are YLDFAC for performing an LDL decomposition (factorization of a semi-definite matrix), and ORTHON for matrix orthonormalization. You can use them like an ordinary TSP command with arguments; the input arguments are the names of matrices and the output arguments will be stored as matrices.

There are two procedures available for factorization (generalized square root) of symmetric matrices: the CHOL function performs a Choleski decomposition, and YLDFAC performs an LDL decomposition.

```
MAT S = CHOL(A) ;
```

finds an upper triangular matrix S such that A = S'S. If A is a symmetric positive semi-definite matrix, there must exist such an S. The number of nonzero elements on the diagonals of S will be equal to the rank of A. **Note:** $A^{-1} = S^{-1}(S')^{-1}$.

```
YLDFAC A D U ;
```

finds a diagonal matrix D and an upper triangular matrix U such that A = U'DU. This decomposition always exists if A is symmetric; the diagonal elements of D are functions of the eigenvalues (characteristic roots) of A. The number of nonzero diagonals of D is the rank of A, and the sign of the diagonals indicate whether the matrix A is positive definite (all positive), negative definite, or indefinite.

If DHALF is a diagonal matrix of whose elements are the square roots of the corresponding elements of D, then the relationship between S and U is S = DHALF*U. This makes it obvious that S is only defined when the elements of D are non-negative.

Orthonormalization is a transformation of a data matrix so that its columns are orthogonal and have a unit norm. This transformation is used by TSP's regression calculation to improve the accuracy of the results. The appropriate transformation is obtained by forming the cross product of the data matrix, X'X, factoring it to obtain the triangular matrix S, inverting S and using it to transform the original data:

$$Z = X S^{-1}$$

Since X'X = S'S, the resulting matrix Z will have the property

$$Z'Z = S^{-1\prime} X'X S^{-1} = I,$$

that is, the columns of Z will be orthonormal.

The form of the command is

```
ORTHON X S Z ;
```

13.4. Examples using matrix operations

The best way to learn how to use TSP matrix operations is to look at examples, as it is not always obvious how to do what you want. This section presents some solutions to common computation problems not yet implemented in TSP procedures. They include performing a Hausman specification test, computing the prediction error for a simple linear model, and computation of a ridge estimator for the linear regression model.

13.4.1. A Hausman specification test

A Hausman test compares two sets of estimates of the same parameters using the same data: one obtained using an efficient estimation technique assuming the specification is correct, and another obtained by an estimation method that is consistent even under a set of alternate hypotheses about the specification. The test is computed by differencing the two sets of parameter estimates and standardizing the vector of differences by the difference in the covariance matrices of the two sets of estimates. The quadratic form computed this way is asymptotically chi-squared with degrees of freedom equal to the number of parameters being tested (with certain caveats that we will mention later).

To compute this test in TSP, we assume that somewhere you have obtained efficient estimates of the parameters, BEFF, and their variance, VEFF. Least squares or some other regression procedure is used to obtain consistent estimates of the parameters under a wide variety of misspecifications of the error term; these estimates are obtained immediately before you perform the test so they will be named @COEF and @VCOV. The Hausman test is computed by the following matrix operations:

```
MAT DVAR = @VCOV-VEFF ;
MAT K = RANK(DVAR) ;
MAT HTEST = (@COEF-BEFF)'YINV(DVAR)*(@COEF-BEFF) ;
CDF(CHISQ,DF=K) HTEST ;
```

The order of subtraction is important for the variance estimates, since the asymptotic variance of the efficient estimate must be less than or equal to that of the consistent estimate (in the matrix sense). However, frequently in practice, some elements of DVAR are negative on the diagonal. If this is the case, the test should be computed only for those parameters corresponding to positive diagonal elements, with a corresponding correction to the degrees of freedom. The program shown above does this automatically, since it sets the degrees of freedom equal to the rank of DVAR. The matrix inversion function YINV() will automatically perform a generalized inverse that sets the linearly dependent rows and columns of the inverse to zero.

13.4.2. Prediction error for the linear regression model

When regressors in the classical linear regression model are fixed constants and the disturbances outside the sample are assumed to be drawn from the same distribution as those within, the variance of the predictor from this model has two pieces. The first is due to the variance of the disturbance and the second to the variance of the estimate of the coefficients. If X is the matrix of in-sample regressors, X_0 is the data for the prediction sample, and s^2 is the variance of the disturbances, the equation for the prediction error variance is

$$V = s^2(I + X_0(X'X)^{-1}X_0')$$

You can use this formula to compute standard error bounds for a forecast of a linear regression model in TSP using matrix operations. Suppose that you have run an OLSQ for the historical period 1958 to 1982 and now wish to forecast through to 1990 using values of the exogenous variables previously projected. The forecast itself can be computed (without printing in this case) immediately following the regression:

```
SMPL 58,82;
OLSQ Y C X1 X2;
SMPL 83 90 ;
FORCST YPRED ;
```

Now use the following matrix commands to construct an estimate of the prediction error:

```
MMAKE X @RNMS ;                    ? Makes a matrix of the data for the forecast time period.
MAT PREDERR = SER(SQRT(@S2+VECH(DIAG(X*@VCOV*X')))) ;
PLOT(BAND=PREDERR) YPRED ;         ? Plots forecast with standard error bands for the prediction error.
```

Note the use of the SER function to convert the output of the computation into a series, and the VECH and DIAG functions to extract the diagonal elements of X*@VCOV*X'.

13.4.3. Ridge regression

Ridge regression is a Bayesian estimator of a linear regression model, with a prior of zero on all coefficients. Ridge estimators have also been viewed by some observers as a way of overcoming multicollinearity in the independent variables. For a discussion of these estimators, see Judge et al.(1980), pp. 452-501. A ridge estimator of the coefficients b in a linear regression model $y = X\beta + \epsilon$ has the form

$$b = (X'X + \gamma I)^{-1} X'y$$

where γ is a positive scalar computed in a variety of ways and I the identity matrix of order of the number of independent regressors. It can be seen easily that the effect of adding a positive constant to all the diagonal elements of X'X will be to reduce the tendency for X'X to be singular or nearly singular.

The example we give of computing a ridge estimator in TSP uses a formula for the scalar γ suggested by Sclove (1973) and described in Amemiya (1985). It is an empirical Bayesian estimator. The prior is that the coefficients are zero with a variance estimated from the data as the sums of squared of the fitted values of y divided by the trace of the X'X matrix. The γ coefficient in this case is a consistent estimate of the residual variance divided by the variance of the coefficient prior. The example below forms an estimate of γ by performing a conventional OLS regression, computing estimates of the variances, and using the matrix algebra routines to run the ridge regression.

```
?
? ORDINARY LEAST SQUARES ON ORIGINAL DATA.
?
OLSQ Y C X2 X3 X4 X5 ;
COPY @COEF ALPHA ;
?
? COMPUTE THE SCLOVE COEFFICIENT  GAMMA.
?
MMAKE X @RNMS ;
MAT GAMMA = (@SSR/@NOB)/((Y'Y-@SSR)/TR(X'X)) ;
?
? NOW FORM THE ESTIMATOR.
?
MAT AHATSTAR = (X'X+GAMMA*IDENT(@NCOEF))" X'Y ;
PRINT GAMMA ALPHA AHATSTAR ;
```