Chapter 6
MANIPULATION AND DISPLAY OF TSP VARIABLES

In the first few chapters of this manual, you learned how to read data into TSP, transform the data, and run simple linear regressions. Because TSP is actually a programming language tailored to the needs of econometricians, it can do much more than this. This chapter introduces some convenient features of the program for printing, plotting, sorting, and manipulating the data and regression results. It also describes a few procedures that perform certain computations automatically on time series used by many researchers, such as accumulating a capital stock from an investment series, Divisia indices, and seasonal adjustment.

To describe the full power of some of these procedures, we need to introduce the concept of a TSP variable type. You have already encountered the basic TSP type:

Series -- Time-series or cross-sectional data. This variable is stored with both a frequency and starting date (or starting observation number, in the case of undated series). Observations outside the SMPL are treated as missing for the procedure being executed, even if they exist in data storage. Exceptions to this rule are lags and leads, which are obtained from outside the current SMPL. If you try to use a series whose frequency differs from the frequency specified by the current FREQ statement, TSP will give you an error message (unless you use CONVERT). Series are usually created with the READ, GENR and UNMAKE commands.

Besides the basic series variables, TSP allows the following kinds of variables:

Scalars -- These are variables that take on a single value. TSP distinguishes between two types of scalars. **Params** are estimable coefficients in a TSP equation, and are declared in a PARAM statement. **Consts** are scalars that are considered fixed or constant. They are created by SET, CONST or UNMAKE statements, or as results from an estimation procedure.

Just as GENR is used to do computations on time series, SET may be used to do computations on scalar variables and elements of matrices and series. The rules for composing SET statements are the same as those for composing GENRs, except the right-hand side must be a scalar-valued expression. Numbers or variable names in parentheses are used as subscripts in SET, while they would be interpreted as lags or leads in GENR. Such names are at most 4 characters for a single subscript, or 2 characters each for a double subscript (see Rules 4b and 4c in Appendix A). Examples:

```
SET XMEAN = @MEAN(1) ;
SET XMAT(2,3) = 1.0 ;
SET XMAT(I,JJ) = Z(I)*W(JJ) ;
SET I1 = I-1;
SET XSIM(I) = XSIM(I1)*EXP(1.0+DELTA) ;
```

Note: Subscript expressions like XSIM(I-1) are not allowed. (See the description of SET in the *Reference Manual*.) Two-dimensional matrices can have a single or double subscript (if it's a single subscript, the matrix is treated as a stacked series of columns). Scalars and subscripted vectors and matrices may be included in GENR statements also; they will be treated as though they have the same value for all observations, rather than being subject to the control of the current sample. Scalars and subscripted variables are legal syntax for command options and arguments that require a single value.

Matrices -- These include vectors as a special case and are described in more detail in Chapter 13. There are four kinds of matrices: general, symmetric, upper triangular, and diagonal. Matrices are created with the MMAKE, MFORM, READ, and MAT commands, and by many estimation procedures.

Lists -- Lists of TSP variable names. They are stored as the output of some procedures and may be created by using the LIST command. The name of a list can be used anywhere in TSP that a list of variable names can be used. Lists can be manipulated, changed, and even lagged. See the *Reference Manual* for details.

Equations -- Explicit (and often nonlinear) formulas for model estimation and simulation, usually created with FRML or IDENT statements (see Chapter 7). Equations are similar to GENR statements, except that they do not immediately compute series; they are used only when the entire model is specified and the estimation or simulation command is invoked. Equations may be printed, used for estimation and simulation, and computed by GENR or SET. Equations are also created from estimated linear models (FORM), and from operations on other equations (EQSUB and DIFFER).

6.1. Using the results of one procedure in another: COPY

Often, the computations you want to do depend on the results of a previous procedure. It can be tedious and time-consuming to run the first computations in one program and then enter those results by hand into a second program before proceeding. TSP provides a solution to this problem by automatically storing many of its most important results in data storage for use in later computations. They are given key names that always begin with the character @ so they will not be confused with the names of variables in your program.

For example, after every regression, two series are available in data storage: one called @FIT, the fitted values of the dependent variable, and one called @RES, the residuals (actual minus fitted values). These series remain in data storage until they are replaced by another regression procedure. They can be used just like any other series before they are replaced. If you wish to save them, include a GENR or COPY statement after your regression:

```
OLSQ CONS C GNP ;
GENR CONSFIT = @FIT ;     or     COPY @FIT CONSFIT;
```

Note: In multi-equation estimation with LSQ and FIML, @RES is stored as a matrix, but the column for a particular equation can be accessed easily with the UNMAKE command (see Chapter 13).

Other results you might want from a regression are: the estimated coefficients, stored as a vector called @COEF; the sum of squared residuals, stored as a scalar called @SSR; the standard error of the residuals, stored as a scalar called @S; and the variance covariance of the estimates, stored as a symmetric matrix called @VCOV. Almost all regression results are available in this form. See the *Reference Manual* for lists of the results available under each procedure.

Procedures other than the estimation procedures also store some of their results in data storage. For example, the seasonal adjustment procedure SAMA stores the seasonal factors under the name @SFAC; and the MSD procedure stores the variable means, standard deviations, etc., in vectors named @MEAN, @STDDEV, etc.

COPY works with all TSP variable types except lists; use the LIST command to copy a list. For example:

```
LIST  NEWLIST  OLDLIST;
```

places the list defined by OLDLIST in storage under the name NEWLIST.

6.2. Printing series and other variables: PRINT, WRITE

WRITE and PRINT are synonymous. They can write any type of TSP variable; you can include different variable types in the same WRITE statement and each will be written using an appropriate format. The number of variables in one WRITE statement is limited only by the amount of working space and space available for the command line.

Here is an example showing the output format for the main TSP variable types. We write the items B1 (a scalar), CONSEQ (an equation), @VCOV (a symmetric matrix), and @RES (a series):

WRITE B1 CONSEQ @VCOV @RES ;   ? The output from this example is shown in **Example 6.1**.

When planning the format of your output, an important consideration is readability. Accordingly, when all the items

```
          B1 =     0.63392
          EQUATION: CONSEQ

               FRML CONSEQ CONS = B0+ B1* GNP

                                                @VCOV

                          1             2
                  1     87.14140
                  2    -0.095976    0.00011333

                           @RES
          1949          26.73818
          1950          17.70645
          1951          -5.35210
          1952         -10.69831
          1953         -12.16861
          1954          -0.33389
          1955          -2.18795
          1956           0.13716
          1957           0.86674
          1958           6.05423
          1959           2.62692
          1960           3.73067
          1961           1.20317
          1962          -5.86250
          1963          -7.39437
          1964          -7.79663
          1965         -11.04351
          1966         -17.97245
          1967         -17.79807
          1968         -15.55392
          1969         -10.66973
          1970           5.04898
          1971           7.63681
          1972           8.41954
          1973           2.22633
          1974          10.11634
          1975          32.32052
```

Example 6.1: WRITE Output

are series, they are produced in table format with as many series per line as will conveniently fit on the page width specified for the current job. The first column of each table will contain the date or observation ID to label the observations. For items that are not series, each will be shown in a separate table. This gives you flexibility in arranging the tables in your output.

If any of the series or other variables have missing values or unprintable values, the number(s) in question will show a period (.) instead of a value.

6.3. Graphic displays of data

Frequently the best way to begin the analysis of a new set of data is by graphing it, either series by series or in more complex ways. TSP provides several routines for the graphical analysis of data: PLOT for one or more time series (assuming that the X axis is time), GRAPH for two variable plots (Y vs. X), and HIST for frequency distributions.

In most personal computer versions of TSP, the plots produced by GRAPH and PLOT are displayed on the screen using high resolution graphics, and can be printed with software supplied with the program. On unix computers, the graphics are produced using only printer characters and have far lower resolution.

6.3.1. Plotting time series: PLOT, PLOTS, NOPLOT

PLOT produces a plot of one or more time series against time or against the observation number if the series is undated. For character plots, the user specifies the series to plot and the character to use for each series in the plot. PLOT is followed by a series name, the character to use in plotting the series, possibly a second series name and a second character, and so on. Up to nine series may be plotted. The characters may be anything but $. ; ' " , . For graphics plots, the character is not necessary. For example:

```
PLOT GNP,*,CONS,X ;          ? character-based plot
PLOT GNP CONS ;              ? graphics plot
```

Various parameters that control the appearance of the plot may be specified in an options list in parentheses following the word PLOT. These options are shown in the *Reference Manual*, although you may not need them if you like the appearance of the plot with the default settings. The default has a box around it but no vertical lines within the box, the maximum and minimum value are labeled, and the observations ID and series value are printed down the right and left side respectively.

For convenience, the PLOT options that you specify are retained in the next PLOT(s) until overridden either explicitly or by including the option RESTORE in the list. RESTORE resets the options at their default values.

Here are some additional examples of the PLOT command.

```
    PLOT(MIN=500,MAX=1500,LINES=(1000)) GNP G GNPS H CONS C CONSS D ;
    PLOT(MIN=-25.,MAX=25.,BMEAN,HEADER,VALUES,BAND=STANDARD,INTEGER) RESID * ;
or
    PLOT GNP GNPS CONS CONSS ;
    PLOT RESID ;
```

There is also a command called PLOTS which "turns on" plots of residuals for all further estimation commands such as OLSQ, 2SLS, LIML, AR1, LSQ, FIML. The NOPLOT command "turns off" the residual plots. These automatic plots are useful for checking the fit of the model.

Example 6.2 shows a plot produced by the DOS/Windows version of TSP (using the Almon(1965) data on expenditures and appropriations).

6.3.2. Graphs or scatter plots: GRAPH

GRAPH produces a scatter plot of one series against another. After the word GRAPH, list first the name of the series you want on the vertical axis and then the name of the series you want on the horizontal axis. Example:

```
GRAPH CONS,YD ;
```

When two or more observations correspond to the same point on the graph, the count of observations overlapping will appear at that point on the graph (10 to 35 observations are denoted by the letters A through Z). A list of such tied observations is printed after the plot.

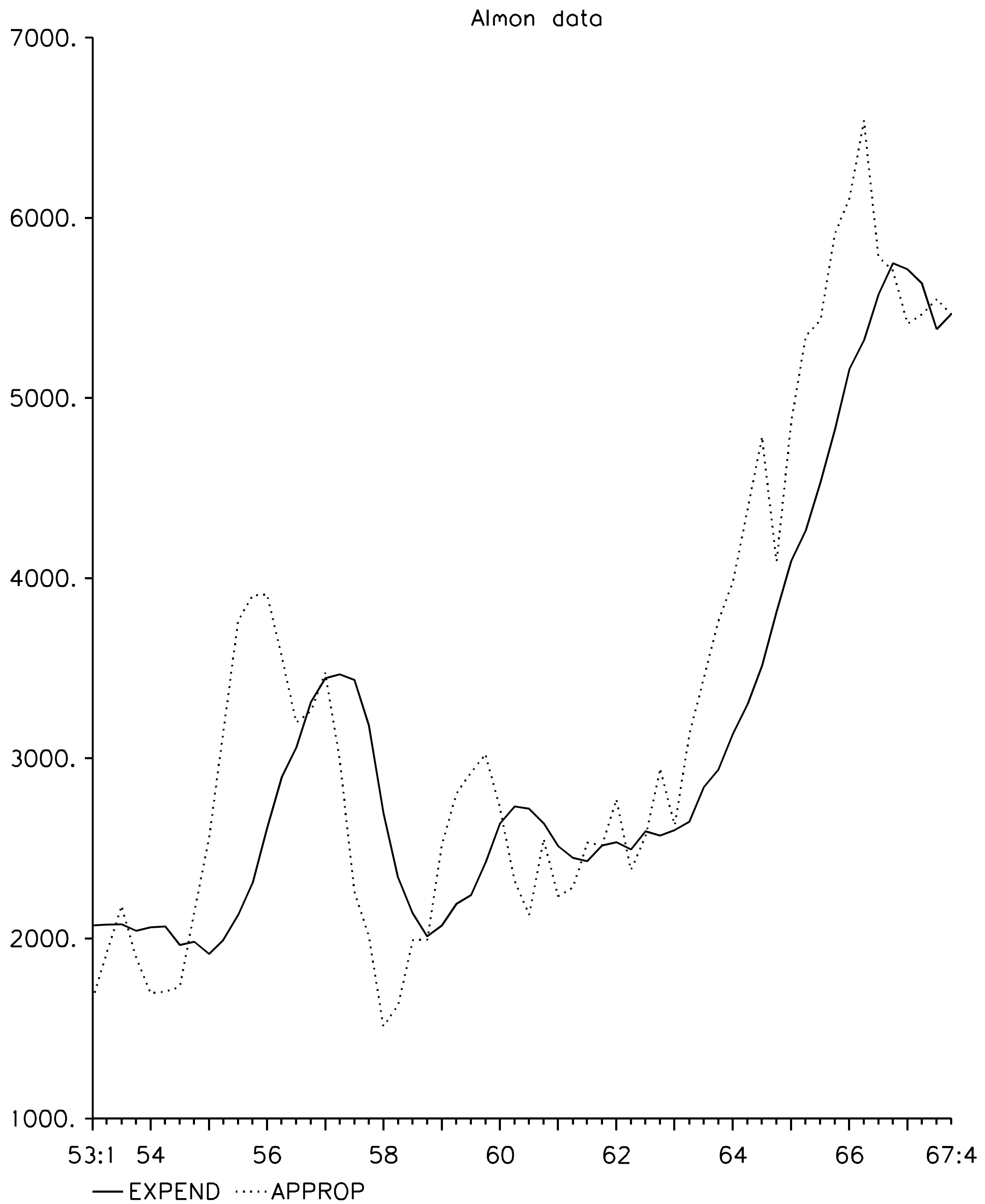
As in the case of PLOT, in personal computer versions, GRAPH also produces high resolution graphics rather than printer character plots. In this case the x-axis series is listed first, and additional y axis series may be included:

```
OLSQ CONS C YD ;
GRAPH YD CONS @FIT ;
```

produces a conventional graph of the actual and fitted values of CONS against the right-hand-side variable YD.

6.3.3. Plotting histograms: HIST

HIST produces histograms (bar charts or frequency distributions) of series. It is convenient for obtaining a rough pictureof the univariate distribution of your data. For example,



Example 6.2: Times Series Plot using Graphics Version

```
HIST X;
```

produces a bar chart with ten bars, indicating the number of observations distributed in sections along the range of the series X. Several options are available to control the number of bars, the range of the series, and the labeling and placement of the bars. These options described in the *Reference Manual*. For example,

```
HIST(DISCRETE,WIDTH=1)  MERGYR;
```

produces a histogram with one line per horizontal bar of the variable MERGYR. Because the option DISCRETE was set, TSP creates a different bar for each unique value of MERGYR, on the assumption that it takes on only a small number of discrete values.

6.4. Sorting data: SORT

You can sort any individual series in ascending or descending order with the SORT command:

```
SORT X ;     or   SORT(REVERSE) X ;
```

You can also sort some or all of your series using a single series to determine the sort order followed by the list of the series to be sorted. For example, a set of panel data for ten countries, in which each country has three years of data (year 1 for all the countries, followed by year 2 for all the countries, etc.), could be reordered so that all the years for each country are adjacent:

```
KEY = COUNTRY*10+YEAR ;
SORT (ALL) KEY ;
```

The original order of the data was

```
COUNTRY        YEAR         KEY
      1           1          11
      2           1          21
      .           .           .
     10           1         101
      1           2          12
      2           2          22
      .           .           .
     10           2         102
      1           3          13
      2           3          23
      .           .           .
     10           3         103
```

and the new order will be

```
      1           1          11
      1           2          12
      1           3          13
      2           1          21
      2           2          22
      2           3          23
      .           .           .
     10           1         101
     10           2         102
     10           3         103
```

6.5. Dummy and trend variables: DUMMY, TREND

Dummy variables are series that are either zero or one in a given observation. They are typically used to measure things that are on or off (true or false). For example, the dependent variable of a Probit model is a dummy variable (see Section 9.2), and the constant term C in a regression is a dummy variable that is always "on". Dummy variables can be created in several ways. They can be LOADed directly, in which case a special convention for repeated values may be useful:

```
    SMPL 1,20;  READ D10; 10*1 10*0;
```

They can also be created in pieces under different SMPL statements. For example (also see TREND below):

```
    SMPL 1,10; D10=1;  SMPL 11,20; D10=0;
```

Dummy variables can also be based on the values of other series. We have already seen examples of this in Section 3.5.1, where logical (true/false) expressions are used in a GENR command:

```
    XPOS = X > 0 ;   or   GENR X1 = (X = 1);
```

Often the series in question takes on several distinct values or categories, and we would like to create a set of dummy variables (one for each category in the original series). This is easily done with the DUMMY command. For example, if the series EDUC takes the values 1, 2, and 3,

```
    DUMMY EDUC;
```

would create the dummy variables EDUC1, EDUC2, and EDUC3. This would be equivalent to the commands:

```
    EDUC1 = (EDUC=1);  EDUC2 = (EDUC=2);  EDUC3 = (EDUC=3);
```

DUMMY can easily create seasonal dummy variables. In fact, this is the default if the DUMMY; command is given (with no argument), and FREQ Q or M is in effect. The series Q1-Q4 or M1-M12 are created. Other DUMMY options are available to exclude the last dummy variable, and to save the names of the new variables in a TSP list. See the *Reference Manual* for details.

Another standard variable is the time trend, which typically equals 1, 2, 3,... etc. This is created with TREND:

```
    TREND T;
```

A trend variable is useful for estimating a deterministic trend component in a regression. It can also be useful for creating time-dependent dummy variables. The first dummy variable example above could have been created with:

```
    SMPL 1,20;  TREND T;  D10 = (T <= 10);
```

The PERIOD= option in TREND allows the series to repeat with the specified periodicity. This is useful with balanced time-series/cross-section data, to make the trend start over at 1 for each individual. See the *Reference Manual* for details.

6.6. Computation of Capital Stock: CAPITL

In working with economic time series data, we are frequently given a gross investment series from which we wish to calculate the corresponding capital stock series based on an assumption about depreciation rates. The CAPITL command enables you to perform this computation easily in TSP. For example, if you have an annual investment series, I, and a depreciation rate of 10 percent per annum, you can compute a series of beginning-of-period capital stock, K, with the following command:

```
    CAPITL I .10 K ;
```

This CAPITL assumed that the value of the capital stock at the beginning of the SMPL was zero and that you wished to build it up from that point. Options are available to alter those assumptions. For example,

```
SMPL 58:1 83:2 ;
CAPITL (BENCHVAL=145.4,BENCHOBS=70:1) QINV .025 K ;
```

uses quarterly investment figures to compute capital stock, assuming that the first quarter of 1970 has a beginning period value of 145.4. Note that since this was quarterly data we used a depreciation rate of 0.025 to correspond to ten per cent per annum.

CAPITL calculates a capital stock series from a gross investment series, using a perpetual inventory and a constant rate of depreciation. Let I be gross investment, K be the capital stock, and δ be the rate of depreciation. Then CAPITL computes

$$K_t = (1-\delta)K_{t-1} + I_{t-1}$$

or $K_t = (1-\delta)K_{t-1} + I_t$ (if the END option is used)

TSP starts from a capital stock benchmark at a specified observation. If the benchmark is in the middle of the sample as in our second example above, CAPITL also applies the reverse version of the formula,

$$K_t = (K_{t+1} - I_t)/(1-\delta)$$

or $K_t = (K_{t+1} - I_{t+1})/(1-\delta)$ (if the END option is used)

to compute values of the capital stock in periods before the benchmark.

A dynamic GENR statement is useful for simple explicit capital stock calculations. The first CAPITL example could be performed with the following statements:

```
SMPL 46,87;  K=0;                    ? initialize capital stock to zero
SMPL 47,87;  K = .9*K(-1) + I(-1);   ? compute capital stock recursively
```

6.7. Divisia Indices: DIVIND

Another commonly needed manipulation of economic time series data is the computation of aggregate price indices from several underlying series. In TSP, you can use DIVIND to compute Divisia Indices, which have several desirable properties as index numbers:

1. They are chain-linked Laspeyres Indices, that is, for each year the current prices are used as a base in estimating the rate of growth to the following year.
2. They are also chained Paasche and Fisher Indices.
3. They are symmetric in prices and quantities.
4. If you reverse time and compute the indices backwards, you obtain the same result.
5. If you form indices of subgroups of the prices and quantities and then combine them using the Divisia method, the resulting index is the same as aggregating in one step from the original series.

These properties are contained in an unpublished note by W. M. Gorman (1970). For a more accessible discussion of their properties and a list of references, see Jorgenson and Griliches (1971) or Diewert (1976).

A Divisia index of prices is obtained by cumulating (over time) a weighted sum of the rates of change of the component prices. The weights are the current shares of the component goods in the total current expenditure on all the goods in the index. If we have N goods with prices $p_1,...,p_N$ and quantities $q_1,...,q_N$, the rate of change of the logarithm of the Divisia price index P is given by

$$\log P_t - \log P_{t-1} = \Sigma\, w_{it}(\log p_{it} - \log p_{it-1})$$

The Divisia index of quantity can be obtained by applying the same strategy to quantities in place of prices, or, alternatively, by dividing total expenditure by the price index. However, the two quantity indices will not be exactly the same. If a quantity is zero or missing, that good is temporarily dropped from the index.

In TSP, the weights w_{it} used for calculating the index may be computed in three ways :

1. Arithmetic weights (ARITH option) are the arithmetic average of the expenditure shares ($p_{it}q_{it}/\Sigma p_{jt}q_{jt}$) in each of the two periods for which we are computing a rate of change.
2. Geometric weights (GEOM option) are the geometric average of the expenditure shares.
3. Combined weights (COMB option) combines the two weight computations given above, that is they are a geometric average of the arithmetic weight, the share this period, and the share last period.

Here is an example of a Divisia command:

```
DIVIND (WEIGHT=ARITH,TYPE=P,PNORM=67) PRICEIN,QUANTIN,PS,QS,PND,QND,PD,QD ;
```

This command computes the indices PRICEIN and QUANTIN from the three pairs of inputs S, ND, and D. Price indices are computed and the quantity indices derived (TYPE=P). The index is normalized to one in 1967.

The default values of the options are the following:

```
TYPE=Q,WEIGHT=COMB,QNORM=1,PNORM=1,NOPRINT,QVAL=1,PVAL=1
```

This means that quantity indices are to be computed using combined share weights, the first observation is to be normalized to have a quantity index of one, and the results are not to be printed. The computed indices will be stored in data storage under the names you gave the first two arguments. For further details on the options, see the *Reference Manual.*

6.8. Normalization of Series: NORMAL

NORMAL normalizes a series so that a chosen observation has an assigned value. It accomplishes this by dividing all observations of the series by the same number. After the word NORMAL, list the name of the series, the observation identifier of the base observation, and the value to be assigned. The normalized series will replace the original series. For example:

```
NORMAL CPI,75,100 ;
```

This is equivalent to the following statements:

```
SET CPI75 = CPI(75);  GENR CPI = 100*CPI/CPI75;
```

6.9. Seasonal Adjustment: SAMA

TSP provides a simple moving average method for performing the seasonal adjustment of series. If seasonally adjusted data is very important in your work, you may wish to survey the literature in this area and make use of a more sophisticated or extensive program such as X-11 from the Census Bureau. A good place to start might be with Census Bureau (1976), in the references.

There are several seasonal adjustment methods other than the "ratio to a moving average". The proper method depends on the theoretical data generation process. Alternate methods include linear or log regression on seasonal dummies and inclusion of seasonal variables in the structural model. Seasonal dummy variables can be created easily with the DUMMY command. TSP provides the SAMA command because it would be difficult to compute with simple regressions; the other methods may be just as valid and they require little programming effort.

SAMA performs seasonal adjustment on quarterly or monthly time series. For example, suppose you have quarterly data on GNP from 1948 through 1982. You could seasonally adjust this series, store and print it with the following command:

```
FREQ Q ; SMPL 48:1 82:4 ;
SAMA(PRINT) GNPQ GNPQA ;
```

For details on the computations, see the *TSP Reference Manual*.

6.10. Principal Components: PRIN

Economic time series, particularly those for aggregate data, are frequently highly collinear; there is often very little information in a fourth or fifth series after you know the first three. Factor analysis with principal components can be a useful way of examining the similarities of data series. Principal components are a set of series constructed to explain as much variance of the original series as possible. Users of this procedure should be familiar with the method and uses of principal components, described in standard texts such as Harman (1976) and Theil (1971).

Here is an example of a PRIN command in TSP:

```
PRIN(NAME=PC,NCOM=3,FRAC=.95) I TIME CONS GOVEXP EXPORTS;
```

It specifies that three principal components are to be found of the five variables I, TIME, CONS, GOVEXP, and EXPORTS. If 95% of the variance of the five variables can be explained by fewer than three components, the program will stop there. The number of principal components actually constructed in any given procedure is the minimum of the number requested, the number needed to explain FRAC of the variance, and the number of series.

In the example given, only two principal components were actually constructed: the first was highly correlated with the level variables I, CONS, and GOVEXP, which in turn were highly correlated (over .95) with TIME. The second had negligible correlation with these variables and was almost entirely related to EXPORTS.

In this example the principal components found will be stored under the names PC1, PC2, PC3, etc., for further use in the program. You may use any legal TSP name as the name for the principal components, but the names generated by adding the numbers must also be legal TSP names (that is, of length of less than 64 characters).
The default values of the PRIN options are

```
NAME=P, NCOM=number of variables, FRAC=1.00
```

This set of options is non-limiting; that is, the maximum number of components will always be constructed.