Chapter 12
CONTROLLING THE EXECUTION OF A TSP PROGRAM

This chapter describes TSP statements that control the program's order of execution. They help advanced users who use TSP as a programming lanuage for solving complex and nonstandard problems. Be aware that although many of the statements resemble Fortran or other high-level language statements, they will not execute as quickly as the corresponding Fortran statements would after being compiled. The advantage of programming within TSP is convenience, but this convenience comes at some cost in execution time.

> **Note:** If you are using TSP interactively, the commands described in this chapter (except REGOPT) must be entered in COLLECT mode, so they can be executed together. See Chapter 17 for more information on interactive use.

The statements covered in this chapter allow you to control the amount of printed regression output (REGOPT), or the order of execution of the program (GO TO, IF, THEN, ELSE). We describe two kinds of loops: ordinary DO loops and the more flexible DOT loops. Finally, we discuss how you can write a TSP procedure that can be used in multiple TSP programs to perform a task that you do often.

12.1. Loops: DO

The DO statement offers a convenient method to execute a group of TSP statements several times, using a variety of parameter values or making some other change each time. All the statements between DO and ENDDO are executed repeatedly according to information provided in the DO statement. In its fullest form, DO uses a counter variable set equal to a lower limit, increments by a fixed amount each time through the loop and stops when an upper limit is reached. The statement has the form DO followed by the name of the counter variable, an = sign, the lower limit, TO, the upper limit, BY, the increment. For example,

```
DO I=1 TO 7 BY 1;
```

specifies that I is the counter, it starts at one and goes through 7 in steps of one. Shorter forms of the DO statement are available; if BY is omitted, the increment is taken to be one; for example,

```
DO I=1 TO 7;     and     DO I=1,7;
```

have the same effect as the previous example. If everything except DO is omitted, the loop is executed just once. This last type of DOloop provides a useful extension of the IF statement (see section 12.5). For example,

```
OLSQ Y C,X;
IF @DW <= 1.5; THEN; DO;
  SET RHO = 1.0-@DW/2 ;
  GENR YT = Y - RHO*Y(-1);
  GENR XT = X - RHO*X(-1);
  GENR CT = 1-RHO;
  OLSQ YT CT,XT;
ENDDO;
```

In this example, the Durbin-Watson statistic is examined and an autoregressive transformation of the data is carried out if there are signs of positive autocorrelation of the residuals.

By using DO in combination with SET and named subscripts, you can do fairly complex data transformations in TSP, but keep in mind that there are probably many other more efficient languages available for this on your computer system.

12.2. Loops over Names: DOT

The DOT statement is like a DO statement, except the index has a set of character values (names or numbers) which can be substituted into names where a dot (.) appears. For example,

```
DOT A B C ;          is equivalent to:      PRINT PA;
  PRINT P. ;                                PRINT PB;
ENDDOT ;                                    PRINT PC;
```

This makes it possible to repeat one or more statements for each of the sectors in a multisectoral body of data. To do this, the names of the series must all have the form of a generic part, common for that series across all the sectors, and a sector part, common across all the series for that sector. This is a conventional way to identify multisectoral data. For example, the generic names might be DW, (rate of change of wages), U (unemployment), and DP (rate of change of prices). The sector names might be 1 (United States), 2 (England), 3 (Sweden), and 4 (Germany). U1 would be unemployment in Sweden, and DP4 would be the rate of change of prices in Germany.

If data are set up in this way, TSP statements may be written with just the generic parts of the names, followed by a dot (.). TSP substitutes the various sector names for the dot and executes the statements repeatedly for all sectors.

The DOT statement indicates the beginning of a set of statements to be executed repeatedly in this fashion. DOT is followed by a list of the sector names. Examples:

```
   DOT 1 2 3 4;
or
   DOT 1-4;
or
   LIST SUBS 1-4;  DOT SUBS;
or
   LIST(FIRST=1,LAST=4) SUBS;  DOT SUBS;
```

The end of the set of statements is marked with an ENDDOT; . To specify a regression of DW on U and DP for each of the four countries in the above example, use

```
DOT 1-4;
  OLSQ DW. C,U.,DP.;
ENDDOT;
```

Any number of statements may appear between DOT and ENDDOT. TSP cycles through them as many times as there are sectors in the DOT statement. Any TSP statement may contain dotted series names. Series names without dots (for example C in the OLSQ statement) are used directly and are not concatenated with the sector name.

Dots may appear any place within a variable name, including the first character, although names that contain only numbers and a dot (.) should not be used since they will be confused with real numbers. For example, .2 is illegal. A name that consists of only a dot (.) *is* allowed, however. Here is an example where the mean is removed from each of several variables:

```
DOT S E K ;
  MSD (NOPRINT) . ;                    ? note the use of a   .   all by itself
  . = . - @MEAN ;
ENDDOT ;
?    The variables S, E, and K are now centered around zero (each of them has had its mean subtracted).
```

Double dots (two DOT loops that are nested) are also possible and can be a powerful technique. The next example evaluates a set of equations for the variables R1-R4, S1-S4 and Q1-Q4. It stores the results in variables with the tag FIT:

```
DOT S R Q ;
 DOT 1 2 3 4 ;
  GENR EQ.. .FIT. ;
 ENDDOT ;
ENDDOT ;
```

The new variables are SFIT1-SFIT4, RFIT1-RFIT4, and QFIT1-QFIT4; note that the dots are evaluated in order within a name.

The DOT command has several options which allow the best features of DO and DOT loops to be combined, and give more control over single-dotted names in nested DOT loops; examples are given in the *Reference Manual*.

12.3. User Procedures: PROC

Another useful way to group TSP statements together for repetitive use is the user procedure. The group is given a name, then the name can be used anywhere in the TSP program to stand for the whole group of statements.

The beginning of a procedure is marked by PROC followed by the name of the procedure. All the statements following PROC are incorporated in the procedure, up to ENDPROC (or ENDP), which marks the end. Example:

```
PROC REGAFT;
 ACTFIT @LHV,@FIT;
 COVA @LHV,@FIT;
ENDP;
```

REGAFT; could be invoked after a regression to carry out the ACTFIT comparison and print the COVA results. For example,

```
OLSQ CONS,C,GNP;
REGAFT;
```

executes the REGAFT procedure for this regression. The effect is as if the two statements in the body of REGAFT had been placed after OLSQ instead.

User procedures may have arguments. These are variable names defined in the PROC statement which stand for the actual names used when the procedure is invoked. Any number of arguments may be listed after the name of the procedure in the PROC statement. When the procedure is invoked elsewhere in the TSP program, the same number of actual variables must be listed after the procedure name. A TSP list variable may be a procedure argument. This provides a way to pass a variable number of series to a procedure for processing. The LENGTH command may be used inside the procedure to count the number of items in the list.

Each time the procedure is invoked, TSP sets up a correspondence between the argument names used in the procedure definition and the variable names specified in the invocation.

```
PROC PCNTCH X,Y;
 GENR Y=100*(X-X(-1))/X(-1);
ENDP ;
```

PROC PCNTCH computes Y as the percent rate of change of X. For example, to compute the percent change of GNP and call it GNPPCH, specify:

```
PCNTCH GNP,GNPPCH;
```

The next example is an alternative to the first example (REGAFT), and shows the use of a list as an argument:

```
PROC REGF VARS;
 OLSQ VARS;
 ACTFIT @LHV,@FIT;
 COVA @LHV,@FIT;
ENDP REGF;

LIST V1 CONS C GNP;                    ? using PROC REGF
REGF V1;
```

The next procedure computes a moving average of X length LEN over the current SMPL and returns them in XMA:

```
PROC MA X,LEN,XMA ;
 LOCAL LAST LAG ;
 XMA=X ;
 SET LAST=1-LEN ;
 DO LAG=LAST TO -1 ;
  XMA = XMA+X(LAG) ;
 ENDDO ;
 XMA=XMA/LEN ;
ENDPROC ;
```

Now MA R,2,RMA; is equivalent to RMA = (R+R(-1))/2; . Note the use of the LOCAL statement to name some variables that are only allocated temporary storage during the execution of procedure MA. This can be convenient if variables named LAST or LAG are in your main TSP program, or if you wish to build a library of procedures and don't want conflicts among their variable names.

One TSP procedure may invoke another and this may proceed to any depth. All TSP procedures are defined at the time that the program is read and printed, and before it is executed. Therefore, the definition of a procedure may appear in a program later than the first place that it is invoked. If errors occur during the execution of a PROC, the line number where each nested PROC was called is printed to aid in diagnosing the problem.

> **Note:** See the TSP examples on the TSP web site for many more complex PROC examples.

12.4. Statement Label and Go To Statement: GOTO

Any statement in TSP may be given a label, which is just a number placed before the command name. The normal order of execution of statements can be modified with a GOTO (or GO TO) statement. GOTO is followed by a statement label (number) and causes that statement to be executed next. Execution then proceeds in normal order from that statement. GOTO statements are most useful in conjunction with the IF statement. Together, they can control the execution of a program using the results of computations in the program.

> **Note**: Be careful using GOTO statements to transfer to an earlier part of the program. They can cause infinite loops.

12.5. Conditional Statements: IF, THEN, ELSE

IF evaluates a scalar expression. Usually the next statement is THEN; followed by a statement to be executed if the scalar expression is "true" (greater than 0). The statement after that may be ELSE; followed by a statement to be executed if the expression is "false" (less than or equal to 0). In other words, the scalar expression defines a logical condition; one statement is executed if the condition is "yes" and another if it is "no". Recall that TSP has relational and logical operators that create and manipulate zero-one variables. These can be tested with IF.

Examples:

```
IF .NOT. @IFCONV;  THEN;  GOTO 100;
```

Control is transferred to statement 100 if the previous estimation procedure did not converge; otherwise GOTO 100 is skipped and the next statement after it is executed.

```
IF LAMBDA>16;
  THEN; PRINT ALPHA,BETA;
  ELSE; PRINT LAMBDA;
```

The expression LAMBDA>16 has the value 1.0 if LAMBDA exceeds 16, in which case ALPHA and BETA will be printed; otherwise LAMBDA itself will be printed. LAMBDA *must* be a scalar, not a series (unless the sample has been set so that only one observation is included). Use the SELECT or GENR statements with logical expressions to perform conditional transformation on series (see Chapter 3). This is often a source of confusion for new TSP users.

Avoid using IF to define loops -- the DO statement (Section 12.1) is a better way to accomplish the same thing. If a collection of several statements is to be executed conditionally, the statements may be enclosed by DO and ENDDO statements (again, see Section 12.1). This avoids GOTO statements, which usually make a program difficult to read and are unnecessary in a well-written program.

12.6. Controlling Printed Output: REGOPT

TSP allows users to specify which results are to be printed in many procedures (mainly the estimation procedures). A complete table of results available from each procedure appears in the *Reference Manual*. Note that results can be stored without being printed under names beginning with @, like @RES, @FIT, @COEF, @SSR, etc.

REGOPT enables you to change the standard selection of results to be calculated and printed. REGOPT(NOPRINT) followed by a list of code names tells TSP to print everything but the results specified by the code names. The suppression of the results takes effect when the REGOPT(NOPRINT) statement appears, and may be turned off at any point by a REGOPT(PRINT) statement. REGOPT; by itself restores the default selection. The complete list of code names available is given in the *Reference Manual*, and a partial list is given in Section 13.2.2 of this manual.

Example:

```
REGOPT(NOPRINT) @YMEAN,@SDEV,@ARSQ,@FST;
```

causes the dependent variable mean and standard deviation, adjusted R-squared, and F-statistic to be omitted from the output in all subsequent estimation procedures.

Many estimation procedures support the SILENT option, which suppresses all the printout except for the log likelihood value, the coefficient estimates, and their standard errors. If these results are also suppressed with a REGOPT(NOPRINT) statement, no printout will occur (results will just be stored under their internal names). This feature is useful when doing many repeated estimations in a Monte-Carlo simulation experiment.