

Elements of Programming in EViews

B. Essama-Nssah
Poverty Reduction Group (PRMPR)
The World Bank
Washington, D.C.
Module 1

Topics

- Basics
- Program Variables and Arguments
 - Control variables
 - String variables
 - Replacement variables
 - Program arguments

Topics

- Control of Execution
 - Mode
 - If statements
 - Loops
- Subroutines
 - Global
 - Local

Basics

- A program is a text file with extension **.PRG**, containing a list of EViews commands (each line corresponds to a single command).
- Purpose of Program:
 - Automate repetitive tasks
 - Create a record of a project

Basics

- Content of a program comprises two types of commands:
 - Those designed for the command line interface.
 - Program control statements.

Basics

- Command line interface
 - Object Declarations
 - Object Commands
 - Object Assignment Statements
 - Auxiliary Commands
 - Unrelated to a particular object
 - Act on it in a way that is independent of type or content of object: e.g. **STORE** or **FETCH**

Basics

- Creating a Program
 - Open a program window with a statement like the following:
 - PROGRAM MINIPAMS
 - Enter text for each command and terminate line by hitting ENTER
 - Autowrap feature for lines longer than window
 - Possible to use underscore continuation character as last character of broken line.

Basics

- Use **SAVE** command to save program on disk.
- Use **OPEN** command to load a previously saved program:
 - **OPEN MINIPAMS.PRG**
- Use the **RUN** command to execute the program.
 - **RUN MINIPAMS**
- By default program will stop running as soon as an error is encountered.
- Use **STOP** sign to run only part of the program (above the stop sign).

Program Variables and Arguments

■ Control Variables

- These variables are designed to serve wherever one could use a numerical value.
- The name must start with "!" mark and may contain up to 15 alphanumerical characters
 - Examples:
 - `!i=8`
 - `!j=12`
 - `MATRIX(!i,!j) AMAT` (Declares a matrix, *AMAT*, with 8 rows and 12 columns)

Program Variables and Arguments

- Control Variables
 - No need to declare before assignment
 - Disappear after program execution
- String Variables
 - A string is text enclosed in double quotes
e.g. "private consumption"
 - String variable is a variable whose value is a string of text

Program Variables and Arguments

- String Variables
- Name begins with symbol: "%" `%`
- Assignment: `%VR="GDP"`
 - Once assigned, it can appear in any expression in lieu of the underlying string.

Program Variables and Arguments

■ Replacement Variables

- Used to refer to an underlying object
- Obtained by enclosing a string or a control variable in curly braces (“{” and “}”)
- Examples
 - EQUATION OLSEQ.LS {%VR} C {%VR}(-1)
means EQUATION OLSEQ.LS GDP C GDP(-1)
 - !K=1
 - SERIES Y{!K}=GDP means SERIES Y1=GDP

Program Variables and Arguments

■ Program Arguments

- Special string variables whose values are passed on to the program during execution.
- Named as "%0", "%1", "%2", "%3" and so on.
- Allow the user to customize program execution by changing the values of these variables as needed
- The following line comes from a program (PAMSDLR) that requires an argument
- **WFOPEN %0**

Program Variables and Arguments

■ Program Arguments

- The values of the arguments must be supplied when invoking the program.
- `RUN PAMSDLR DLRHHDATA`
- Following program, `REGPRG`, has more two arguments:
 - `EQUATION OLSEQ`
 - `SMPL 1980Q3 1994Q1`
 - `OLSEQ.LS {%0} C {%1} {%1}(-1) TIME`
 - (Call: `RUN REGPRG LGDP M1`)

Control of Execution

- There are three basic aspects in controlling the way a program runs:
 - Mode of Execution
 - Selective execution of commands
 - Repeated execution of commands under changing conditions.

Control of Execution

- Mode Statement

- Purpose: to change the mode of execution of the program from within.
- Options include:
 - `MODE QUIET`
 - `MODE VERBOSE`
 - `MODE VER4`
- Multiple settings can be invoked in a single line:
 - `MODE QUIET VER4`

Control of Execution

■ IF Statements

- Used to execute commands only if some condition is met.
- Start with key word "IF", followed by statement of condition and the word "THEN"
- List commands to be executed if condition is true
- End with the word "ENDIF"

Control of Execution

- IF Statements

```
IF !DEFLATE THEN
    SERIES RGDP=GDP/!DEFLATE
ENDIF
```

- Use "ELSE" to include commands to execute when condition is false

```
IF !DEFLATE THEN
    SERIES RGDP=GDP/!DEFLATE
ELSE
    SERIES RGDP=GDP
ENDIF
```

Control of Execution

- Nested IF Statements

```
IF !EPS=0 THEN
    EDE=@SUM(YHW)
ELSE
    IF !EPS=1 THEN
        EDE=EXP(@SUM(LGYW))
    ELSE
        EDE=(@SUM(YHWT))^(1/(1-!EPS))
    ENDIF
ENDIF
ENDIF
```

Control of Execution

■ Loops

- The FOR loop allows one to repeat a set of commands for different values of a control or string variable.
- Start with the word "FOR"
- End with "NEXT"
- Example:

```
FOR !K=1 TO !M  
    BETA(!K)=OSLEQ.C(!K)  
NEXT
```

Control of Execution

- In the above example, the **STEP** increase is implicitly set to 1. Here is an explicit treatment:

```
FOR !NU=1 TO !AVMAX STEP 1/!B
    SERIES QANU=QI^(!NU)
NEXT
```

- Examples of FOR loop with string variable

```
FOR %ST EQCN EQI EQR
    ISLM.MERGE {%ST}
NEXT
```

Control of Execution

```
FOR %S1 %S2 MU MU POP POP
  %S3=%S1 +@STR(!T)
  SCALAR {%S3}={%S2}
  STORE {%S3}
NEXT
```

- Nested FOR Loops

```
FOR !I=1 TO 3 STEP 2
  FOR !J=1 TO 3 STEP 2
    VECTOR(3) PM{!I}L{!J}
  NEXT
NEXT
```

Control of Execution

- The WHILE Loop
 - Analogous to an IF statement to the extent that it allows repeated execution of a set of commands while one or more conditions are satisfied.
- Provides greater flexibility than the FOR loop in the specification of required conditions
 - It starts with "WHILE" and ends with "WEND"
 - Example:

```
!K=1
WHILE !K<=10
    VECTOR VEC{!K}
    !K=!K +2
WEND
```

Subroutines

- Provide a more general approach to reusing commands and using arguments.
- A collection of commands that can be used repeatedly with minor modifications and without duplicating the commands in question.
- Start with "**SUBROUTINE**", end with "**ENDSUB**"
- Any number of commands can appear in between.

Subroutines

- Example

```
SUBROUTINE LOGSUB  
    SERIES Y=LOG(X)  
ENDSUB
```

- May include arguments listed within parentheses after name of subroutine, and separated by commas.
- Each argument is specified by object type and by a name.

Subroutines

- Example of Subroutine with arguments:

```
SUBROUTINE POWER(SERIES V,SERIES Y,  
SCALAR P)
```

```
V=Y^P
```

```
ENDSUB
```

- Subroutine definitions must be placed at the beginning of the program that will call them.
- Use **INCLUDE** command if subroutines are in separate files

Subroutines

- Use `CALL` statement to execute the commands in the subroutine. `CALL` should be followed by name of subroutine and the list of argument values, if any, enclosed in parentheses and separated by commas.
- Examples
 - `CALL LOGSUB`
 - `CALL POWER(GDP2, GDP, 2)`
(GDP2=SQUARE OF GDP)

Subroutines

- Global versus Local Objects
 - **Global**: either object exists in workfile at the time the subroutine is called or it is created in the workfile by the subroutine.
 - **Local**: Object is meaningful only within subroutine. It disappears once the subroutine has run.

Subroutines

- Global Subroutine
 - Default category
 - Has access to and can alter global objects.
 - Creates objects that are global.
 - If a global object and an argument have the same name the argument has precedence.

Subroutines

- Local Subroutine

- Key word **LOCAL** must be included in definition

```
SUBROUTINE LOCAL OLS(SERIES Y, SERIES RES,  
SCALAR SSR)  
EQUATION OLSEQ.LS Y C Y(-1)  
OLSEQ.MAKERESID RES  
SSR =OLSEQ.@SSR  
ENSUB
```

OLSEQ is a local object and will vanish after the subroutine is done running.

Subroutines

- Local Subroutine
 - May not use or update global objects directly from within the subroutine.
 - However global objects corresponding to arguments may be used and updated by referring to the arguments. Such objects must be created outside the local subroutine and passed on to the subroutine as arguments.



The End.