GPD-2

Second Edition
April 1994

User's Guide to TABLO, GEMSIM and TABLO-Generated Programs

GEMPACK Document No. 2

Copyright 1993-94 by the Impact Project and KPSOFT

ISSN 1030-2514

This is part of the documentation of the GEMPACK Software System for solving large economic models, developed by KPSOFT and the IMPACT Project, Menzies Building, Monash University, Wellington Road, Clayton, Vic. 3168, Australia.

Abstract

GEMPACK is a suite of general-purpose economic modelling software especially suitable for general and partial equilibrium models. It can handle a wide range of economic behaviour and also contains powerful capabilities for solving intertemporal models. GEMPACK provides software for calculating accurate solutions of an economic model, starting from an algebraic representation of the equations of the model. These equations can be written as levels equations, linearized equations or a mixture of these two.

The software is user-friendly, provides a range of utility programs for handling the economic data base and the results of simulations, and is fully documented from a user's point of view. GEMPACK runs on a wide variety of computers.

TABLO is the GEMPACK program which translates the algebraic specification of an economic model into a form which is suitable for carrying out simulations with the model. The output from TABLO can be either computer files used to run the GEMPACK program GEMSIM or alternatively a Fortran program, referred to as a TABLO-generated program. Either GEMSIM or the TABLO-generated program can be run to carry out simulations.

This document is a complete User's Guide to TABLO, GEMSIM and TABLO-generated programs. It complements GEMPACK document GPD-1 **An Introduction to GEMPACK** (with which readers should be familiar). This document contains

- fine print about running TABLO and its Condensation stage,
- a detailed description of the syntax and semantics of the TABLO language (as used in TABLO Input files specifying models),
- complete documentation of the different ways that GEMSIM and TABLO-generated programs can be run,
- advice about verifying that your model is actually carrying out the calculations that you intend, and
- information about implementing and solving intertemporal (that is, dynamic) models using GEMPACK.

Document Attributes

Name	:	User’s Guide to TABLO, GEMSIM and TABLO-Generated Programs.
Audience	:	CGE Modellers
Identifier	:	GPD-2

History :

Date	*Author(s)*	*Comment*
April 1993	Jill Harrison and Ken Pearson	First edition [Title of first edition was "User’s Guide to TABLO and TABLO-generated Programs"]
April 1994	Jill Harrison and Ken Pearson	Second edition

CONTENTS

CHAPTER 1 INTRODUCTION

CHAPTER 2 RUNNING TABLO

2.1 TABLO Options 2-1
2.1.1 TABLO Code Options 2-3
2.1.2 Identifying and Correcting Syntax and Semantic Errors 2-4
2.2 TABLO Linearizes Levels Equations Automatically . 2-4
2.2.1 Change or Percentage-change Associated Linear Variables 2-5
2.2.2 How Levels Statements are Converted 2-6
2.2.3 Linearizing Levels Equations 2-7
2.3 More Details About Condensation and Substituting Variables 2-10
2.3.1 Looking Ahead to Substitution when Creating TABLO Input Files 2-13
2.3.2 System-initiated Formulas and Backsolves . . . 2-14
2.3.3 Absorption 2-16
2.4 Increasing Parameter Values in TABLO 2-17

CHAPTER 3 BASIC SYNTAX DESCRIPTION

3.1 SET 3-3
3.2 SUBSET 3-5
3.3 COEFFICIENT 3-6
3.4 VARIABLE 3-7
3.5 FILE 3-8
3.6 READ 3-9
3.7 WRITE 3-10
3.8 FORMULA 3-11
3.9 EQUATION 3-12
3.9.1 "EQUATION(NONE);" Statement 3-12
3.9.2 FORMULA & EQUATION 3-13
3.10 UPDATE 3-14
3.11 ZERODIVIDE 3-15
3.12 DISPLAY 3-16
3.13 Setting Default Values of Qualifiers 3-17
3.14 TABLO Statement Qualifiers - A Summary 3-18

CHAPTER 4 SYNTAX AND SEMANTIC DETAILS

4.1 General Notes on the TABLO Syntax and Semantics . 4-1
4.1.1 Input Statements 4-1
4.1.2 Upper and Lower Case 4-2
4.1.3 Comments 4-2
4.1.4 Strong Comment Markers 4-2
4.1.5 Reserved (special) Characters 4-3
4.2 User Defined Input 4-3
4.2.1 Names 4-3
4.2.2 Labelling Information (Text between hashes #) . 4-5
4.2.3 Arguments - Indices and Set Elements 4-5
4.3 Quantifier lists 4-6
4.4 Expressions Used in Equations, Formulas and Updates 4-7
4.4.1 Operations Used in Expressions 4-7

4.4.2 Sums over Sets in Expressions 4-7
4.4.3 Brackets in Expressions 4-8
4.4.4 Functions 4-9
4.4.5 Conditional Operators 4-9
4.4.6 Conditional Expressions 4-11
4.4.7 Linear Variables in Expressions 4-12
4.4.8 Constants in Expressions 4-13
4.4.9 Indices in Expressions 4-13
4.4.10 Integer Coefficients in Expressions and Elsewhere 4-14
4.4.11 Where Coefficients and Levels Variables Can Occur 4-16
4.5 Sets and Subsets 4-19
4.5.1 Sets 4-19
4.5.2 Subsets 4-19
4.6 Files 4-20
4.6.1 Text Files 4-21
4.7 Reads, Writes and Displays 4-23
4.7.1 How Data is Associated With Coefficients . . . 4-23
4.7.2 Partial Reads, Writes and Displays 4-24
4.7.3 FORMULA(INITIAL)s 4-25
4.7.4 Coefficient Initialisation 4-25
4.7.5 Display Files 4-26
4.8 Updates 4-26
4.8.1 Purpose of Updates 4-26
4.8.2 Which Type of Update? 4-27
4.8.3 What If An Initial Value Is Zero ? 4-27
4.8.4 UPDATE Semantics 4-28
4.9 Zerodivides 4-29
4.10 Ordering 4-31
4.10.1 Ordering of the Input Statements 4-31
4.10.2 Ordering of Variables 4-31
4.10.3 Ordering of Components of Variables 4-32
4.10.4 Ordering of the Equation Blocks 4-32
4.10.5 Ordering of the Equations Within One Equation Block 4-33
4.10.6 Ordering of Reads, Formulas, Equations and Updates 4-33
4.11 Model Parameters 4-35
4.12 TABLO Input Files with No Equations 4-36

CHAPTER 5 GEMSIM AND TABLO-GENERATED PROGRAMS

5.1 Introduction 5-1
5.2 More Details About Multi-step Calculations 5-1
5.2.1 Gragg’s Method and the Midpoint Method 5-1
5.2.2 Reusing Pivots 5-3
5.2.3 Time Taken for the Different Steps 5-4
5.2.4 Total Time for Multi-step Compared to 1-step . . 5-5
5.2.5 Using Existing Files to Speed Up Step 1 5-6
5.2.6 Warnings About How Well the Equations are Solved 5-7
5.2.7 Gragg and Midpoint Are Not Suitable in Some Cases 5-7
5.2.8 Linearization Used Can Affect Accuracy 5-8
5.3 Options Available with GEMSIM and TABLO-generated Programs 5-10
5.4 Options Affecting Simulations 5-11
5.4.1 Ignoring/Keeping Zero Coefficients (Options IZ1 and KZ2) 5-11
5.4.2 No Reuse of Pivots (Option NRP) 5-13
5.4.3 No Iterative Refinement of Solutions (Option NIR) 5-13

5.4.4 No Warnings About How Well the Equations are Solved (Option NWE) 5-13
5.5 Options Affecting Extrapolation Accuracy Summaries/Files 5-13
5.5.1 Required Figures of Accuracy (Option RQF) . . 5-14
5.5.2 Selecting Variables on the Extrapolation Accuracy File 5-14
5.6 Options Affecting the Actions Carried Out . . . 5-15
5.6.1 Option NSE - Don't Save Equations File 5-16
5.6.2 Options Affecting How Writes and Displays are Carried Out 5-17
5.7 Options Affecting CPU and Activity Reporting . . 5-19
5.8 Splitting a Simulation into Several Subintervals (Option SSI) 5-20
5.9 Names of Auxiliary Files 5-22
5.9.1 TABLO-generated Programs 5-22
5.9.2 GEMSIM 5-22
5.10 Code Options When Running TABLO 5-23
5.10.1 Code Options in TABLO Affecting the Possible Actions 5-24
5.10.2 Code Options in TABLO Affecting the Amount of Memory Required 5-26
5.10.3 Other Code Options in TABLO 5-29
5.11 Compiling, Linking and Running TABLO-generated Programs 5-30
5.12 Possible Errors While Running GEMSIM or TABLO-generated Programs 5-30
5.12.1 Simulation Fails Because of a Singular LHS Matrix 5-31
5.12.2 Division by Zero Errors 5-32
5.13 Increasing Parameter Values in the Source Code . 5-33
5.13.1 Increasing Parameter Values in GEMSIM 5-33
5.13.2 Increasing Parameter Values in TABLO-generated Programs 5-34
5.14 Final and Intermediate Updated Data Files . . . 5-37
5.14.1 Default Names for Intermediate Files When Using a Command File 5-40
5.14.2 Command File Syntax for Data File Names . . . 5-40

CHAPTER 6 VERIFYING ECONOMIC MODELS

6.1 Is Balanced Data Still Balanced After Updating? . 6-3
6.1.1 Balance After n-Steps 6-5

CHAPTER 7 INTERTEMPORAL MODELS

7.1 Introduction to Intertemporal Models 7-1
7.2 Intertemporal Sets 7-1
7.2.1 Set Size - Fixed or Determined at Run Time . . . 7-2
7.3 Use an INTEGER Coefficient to Count Years 7-5

CHAPTER 8 LESS OBVIOUS EXAMPLES OF THE TABLO LANGUAGE

8.1 Adding Across Time Periods in an Intertemporal Model 8-1
8.2 Conditional Functions or Equations 8-2
8.3 Mappings Between Sets 8-3
8.4 Aggregating Data 8-4

APPENDIX A LINEARIZING LEVELS EQUATIONS

A.1 Differentiation Rules Used by TABLO A-1
A.2 Linearizing Equations by Hand A-2
A.2.1 General Procedure A-2
A.2.2 Rules to Use A-3
A.2.3 Linearizing Equations in Practice A-4
A.2.4 Linearizing Using Standard References A-5

APPENDIX B LINEARIZED EQUATIONS ON INFORMATION FILE

APPENDIX C EXAMPLES OF THE DERIVATION OF UPDATE STATEMENTS

C.1 Example 1 - A Levels Data Base Value Which is the Sum of Two Flows C-1
C.2 Example 2 - Powers of Taxes C-2

REFERENCES

INDEX

CHAPTER 1

INTRODUCTION

This document GPD-2* provides complete user documentation of TABLO, GEMSIM and TABLO-generated programs. These are the means by which economic models are implemented and solved within GEMPACK, as described in GEMPACK document GPD-1 **Introduction to GEMPACK** (with which you should be familiar before reading this document GPD-2).

We expect that this document will be used mainly as a reference document (rather than being read through in order). The fairly comprehensive **Index** should help you find the relevant part whenever you need more information about TABLO, GEMSIM or TABLO-generated programs.

Chapter 2 provides some of the fine print about running TABLO, including how it linearizes levels equations and about its Condensation stage. Chapter 3 is a full description of the syntax required in TABLO Input files while chapter 4 contains a comprehensive description of the semantics (and a few points about the syntax which may not be clear from chapter 3) for the current version of TABLO. Chapter 5 is a complete documentation of GEMSIM and TABLO-generated programs. Chapter 6 provides some assistance with the important task of verifying that your model is actually carrying out the calculations you intend. Chapter 7 provides some details about intertemporal (that is, dynamic) modelling in GEMPACK. In chapter 8 we give examples of

---

* References to GEMPACK documents identify the document by GEMPACK Document (GPD) number, rather than by author or date. References are always to the version of the document which is current at the date of issue of the cross-referencing document. The GEMPACK documents referenced are listed in a separate section at the end of the References section of this document. Comments from readers on this or any of the GEMPACK documents, either pointing out errors, inaccuracies, omissions or obscurities, or making other suggestions for improvements, will be welcomed. Please address such comments to one of the authors at the Impact Project.
The numbering of GEMPACK Documents has been re-started with Release 5 of GEMPACK, when the abbreviation "GPD" was first used. Previous editions of these documents did not have the same numbers as the current editions. Pre-Release-5 documents are numbered "GED-xx". Document GPD-2 supersedes the pre-Release 5 documents GED-20, GED-20a, GED-30 and GED-31.

ways in which the TABLO Language can be used to express relationships which at first sight are difficult to capture within the syntax and semantics of TABLO Input files.

This document describes version 3.1 (March 1994) of TABLO, the TABLO-generated programs produced by it, and version 1.2 (March 1994) of GEMSIM.

CHAPTER 2

RUNNING TABLO

Most of the information about running TABLO is given in chapters 2-5 of GPD-1. This chapter provides some additional information, namely about the TABLO Options screens (section 2.1), how TABLO linearizes levels equations (section 2.2), some of the fine print about Condensation (section 2.3) and increasing parameter values in TABLO (section 2.4).

2.1 **TABLO Options**

The TABLO program is divided into three distinct stages: CHECK, CONDENSE and CODE.

(1) In the CHECK stage, TABLO analyses the information on the TABLO Input file and points out any syntax errors (where the format expected by TABLO has not been followed) or semantic errors errors (where different parts of the input are not consistent with one another). Errors are output briefly to the screen and also to an **Information file** (usually with suffix .INF). Errors can be found by searching the Information file for %% which precedes each error message. (See section 2.1.2 below for more details.)

If you stop after the CHECK stage and no syntax or semantic errors were reported, TABLO saves the results on two binary files - a TABLO Record file usually with suffix (.TBR), and a TABLO Table file (.TBT), as described in section 3.9.4 of GPD-1.

(2) The CONDENSE stage is optional. Details of condensation are given in section 3.9 of GPD-1 and in section 2.3 below.

(3) The CODE stage either writes output for GEMSIM or writes the TABLO-generated program which corresponds to the TABLO Input file.

On starting TABLO, you make selections from the TABLO Options menu shown below. Standard Basic Options LOG, STI, SIF, ASI, BAT, BPR at the top of the screen are described in chapter 5 of GPD-1.

Default values are F1 for the First Stage and L3 for the Last Stage; these mean that TABLO starts with the CHECK stage, then, if no errors are encountered during the CHECK, carries out CONDENSE (if requested), and then goes on to the CODE stage. However later, as the program

progresses, you can choose to bypass the optional CONDENSE stage, or opt to stop after any of the three stages.

To start at the CONDENSE stage, choose F2 for the First Stage from the Options Menu. However you will need the TABLO Record file (.TBR) and Table file (.TBT) file from a previous run when the CHECK stage was carried out. Similarly you can choose F3 as the First Stage to write the CODE only if the previous stages have been completed successfully earlier. If you just wish to carry out the CHECK, you can choose option L1 (Last Stage is CHECK), while choosing L2 means that TABLO stops after CONDENSE.

Section 3.9.4 of GPD-1 describes stopping and restarting TABLO.

TABLO OPTIONS
(--> indicates those in effect)

```
   BAT Run in batch          STI Take inputs from a Stored-input file
   BPR Brief prompts         SIF Store inputs on a file
   LOG Output to log file    ASI Add to incomplete Stored-input file

   First stage               Last Stage
   -----------               ----------
-->F1 CHECK                  L1 CHECK
   F2 CONDENSATION           L2 CONDENSATION
   F3 CODE GENERATION    --> L3 CODE GENERATION

   ACD Always use Change Differentiation of levels equations
   SCO Specialized Check Options menu

 Select an option   :  <opt>  Deselect an option      : -<opt>
 Help for an option : ?<opt>  Help on all options     : ??
 Redisplay options  : /       Finish option selection Carriage return
Your selection >
```

Main TABLO Options Menu

Option ACD is discussed in sections 2.2.3 and 5.2.8 below. It affects the way TABLO linearizes any levels equations which, in turn, can affect the numerical properties of multi-step calculations.

Choosing SCO gives access to the Specialised Check Options menu given below. However these options are rarely used so TABLO uses the default values for these unless you actively choose SCO and one or more of the Specialised Check options.

```
                  Specialised Check Options
                ( --> indicates those in effect )

        Semantic Check Options
        ----------------------
        SM1 Omit semantic check
        SM2 Allow duplicate names
        SM3 Omit coefficient initialisation check
        SM4 Omit checking for warnings
        SM5 Do not display individual warnings

        Information File Options
        ------------------------
        IN1 Has the same name as the TABLO Input file
        IN2 Only show lines containing errors
        IN3 Omit the model summary in CHECK stage
        IN4 Omit summary of generated code in CODE stage

Select an option   :  <opt>  Deselect an option      : -<opt>
Help for an option : ?<opt>  Help on all options     : ??
Redisplay options  : /       Return to TABLO Options : Carriage return
Your selection >
```

Specialised Check Options Menu

After the CHECK stage is complete, if no syntax or semantic errors have been found, you are given the choice below:

```
  Do you want to see a SUMMARY of the model        [s], or
              perform CONDENSATION                 [c], or
              proceed to AUTOMATIC CODE GENERATION [a], or
              EXIT from TABLO                      [e] :
```

If you select [c], the following choice is presented. See section 3.9 of GPD-1 and section 2.3 below for a description of Condensation.

```
 --> Starting CONDENSATION

  Do you want to SUBSTITUTE a variable             [s], or
      substitute a variable and BACKSOLVE for it   [b], or
                  OMIT one or more variables       [o], or
                  ABSORB one or more variables     [a], or
                  DISPLAY the model's status       [d], or
                  EXIT from CONDENSATION           [e] :
```

If you select [e] at either of the last two choices, the TABLO Record file (.TBR) and the Table file (.TBT) are written and the program TABLO ends.

2.1.1 **TABLO Code Options**

When you proceed to Automatic Code Generation, a Code Options Menu is presented. The main choice here is whether to produce output for GEMSIM (option PGS) or to write a TABLO-generated program (option WFP). Because the effect of the other options is intimately bound up with the way GEMSIM or the resulting TABLO-generated program will run,

we postpone a discussion of these options until section 5.10 of chapter 5, which deals with GEMSIM and TABLO-generated programs.

2.1.2 **Identifying and Correcting Syntax and Semantic Errors**

If TABLO finds syntax or semantic errors during the CHECK stage, it reports them to the terminal and also, more usefully, to the Information file.

To identify these errors, look at the Information file (via a text editor, or print it out). Syntax and semantic errors are marked by two percent signs '**%%**', so you can search for them in an editor. The Information file usually shows the whole TABLO Input file (with line numbers added); lines with errors are repeated and a brief explanation is given of the reason for each error. (Also a question mark '?' in the line below the line with an error points to the part of the line where the error seems to be.)

Usually the change needed to correct the error will be clear from the explanation given. If not, you may need to consult the relevant parts of chapter 3 (for syntax errors) or chapter 4 (for semantic errors).

One syntax or semantic error may produce many more. If, for example, you incorrectly declare a COEFFICIENT A6, then every reference to A6 will produce a semantic problem ("Unknown coefficient"). In these cases, fixing the first error will remove all consequential errors.

2.2 **TABLO Linearizes Levels Equations Automatically**

During the CHECK stage, when TABLO processes a TABLO Input file containing levels EQUATIONs and levels VARIABLEs, it converts the file to a linearized file; we refer to this as the **associated linearized TABLO Input file**. Although you may not see this associated linearized file (since the conversion is done internally by TABLO), you should be aware of some of its features.

The most important feature of this conversion is that, for each levels VARIABLE, say X, in your original TABLO Input file, there is an associated linear VARIABLE whose name is that of the original levels variable with "p_" added at the start.* Also, for each levels VARIABLE in the original TABLO Input file, a COEFFICIENT with the same name as the levels VARIABLE is declared in the associated linearized TABLO Input file.

Other features of this conversion will be explained in sections 2.2.1 to 2.2.3 below.

---

* Actually this is not entirely accurate. If the levels VARIABLE is declared via a VARIABLE(LEVELS,CHANGE) statement (see section 2.2.1 below), the associated linear VARIABLE has "c_" at the start.

It is important to realise that the rest of TABLO (the last part of the CHECK and all of the CONDENSE and CODE stages) proceed

as if the associated linearized TABLO Input file were the actual TABLO Input file.

This means that

- during CHECK, warnings and error messages may refer to statements in this associated linearized file rather than in your original TABLO Input file, and
- during CONDENSE, if you wish to substitute out or omit variables, you must use the names of the associated linear VARIABLEs (those with "p_" or "c_" added) rather than those of the levels VARIABLEs.

Also, when you carry out simulations by running GEMSIM or the TABLO-generated program from your model, you must refer to variables on the associated linearized file rather than the levels variables.

During the CHECK stage, TABLO normally echos the original TABLO Input file to the Information file (and flags any errors or warnings there). When there are levels EQUATIONs in the original file, in the Information file which TABLO writes to describe this model, each levels EQUATION is followed by its associated linearized EQUATION. So, if you wish to see the associated linearized EQUATIONs you can do so by looking at the CHECK part of the Information file. In Appendix B we show part of the Information file obtained from processing the TABLO Input file SJ.TAB for the mixed version of Stylized Johansen; you can look there to see the linearized EQUATIONs associated with some of the levels EQUATIONs from this TABLO Input file, which is shown in full in section 3.3.2 of GPD-1.

2.2.1 **Change or Percentage-change Associated Linear Variables**

When you declare a levels VARIABLE in your TABLO Input file, you must also decide which form of associated linear VARIABLE you wish to go in the associated linearized TABLO Input file. If you want it to be the corresponding percentage change, you don’t need to take special action since this is usually the default. If however you wish it to be the corresponding change, you must notify TABLO of this by including the qualifier (CHANGE) in your VARIABLE statement. For example, the statement

```
VARIABLE (LEVELS,CHANGE) BT # Balance of Trade # ;
```

in a TABLO Input file will give rise to a CHANGE variable c_BT in the associated linearized TABLO Input file.

As explained in section 3.3.4 of GPD-1, there are some circumstances when a change linear variable is preferable to the percentage-change alternative. When you declare a levels VARIABLE we suggest the following guidelines.

- For a levels variable which is always positive (or always negative), direct TABLO to work with the associated percentage change as a linear VARIABLE in the associated linearized TABLO Input file.

- For a levels variable which may be positive, zero or negative, direct TABLO to work with the associated change as a linear VARIABLE in the associated linearized TABLO Input file. This can be achieved by declaring the levels VARIABLE via a VARIABLE(CHANGE) statement, as in, for example,

  ```
  VARIABLE (LEVELS,CHANGE) BT # balance of trade # ;
  ```

2.2.2 **How Levels Statements are Converted**

When you declare a levels VARIABLE or write down a levels EQUATION in a TABLO Input file, these give rise to associated statements in the associated linearized TABLO Input file created automatically by TABLO. After that, TABLO's processing proceeds **as if** you had actually written these associated statements rather than the levels statements actually written. We look at the different possibilities below.

Declaration of a Levels VARIABLE

Each declaration of a levels VARIABLE is automatically converted to three statements in the associated linearized TABLO Input file. These are

(1) the declaration of a COEFFICIENT(NON_PARAMETER) with the same name as the levels VARIABLE;

(2) the declaration of the associated linear VARIABLE - its name has "p_" or "c_" added at the start depending on whether the corresponding percentage-change or change has been indicated by a qualifier PERCENT_CHANGE or CHANGE , or by the default statement currently in force if neither of these qualifiers is present;

(3) an UPDATE statement saying how the COEFFICIENT in (1) is to be updated during a multi-step simulation.

Examples

1. The statement

```
VARIABLE (LEVELS,PERCENT_CHANGE) (all,c,COM) X(c) #label# ;
```

is converted to the 3 statements

```
COEFFICIENT (NON_PARAMETER) (all,c,COM) X(c) ;
VARIABLE (LINEAR,PERCENT_CHANGE) (all,c,COM) p_X(c) #label# ;
UPDATE (all,c,COM) X(c) = p_X(c) ;
```

2. The statement

```
VARIABLE (LEVELS,CHANGE) (all,i,IND) Y(i) #label# ;
```

is converted to the 3 statements

```
COEFFICIENT (NON_PARAMETER) (all,i,IND) Y(i) ;
VARIABLE (LINEAR,CHANGE) (all,i,IND) c_Y(i) #label# ;
UPDATE (CHANGE) (all,i,IND) Y(i) = c_Y(i) ;
```

A Levels EQUATION

Each levels EQUATION is converted immediately to an appropriate linearized EQUATION. This linearized EQUATION has the same name as that used for the levels EQUATION. Exactly how the linearization is done depends on the types of associated linear VARIABLEs.

Example

The statement

```
EQUATION (LEVELS) House (all,c,COM) DVH(c) = P(c) * QH(c) ;
```

may be converted to the linearized equation

```
EQUATION (LINEAR) House (all,c,COM) p_DVH(c) = p_P(c) + p_QH(c) ;
```

if the levels VARIABLEs DVH, P and QH have percentage-change linear variables associated with them.

We describe the different linearizations in section 2.2.3 below.

2.2.3 **Linearizing Levels Equations**

In linearizing a levels equation, there are two methods of carrying out the differentiation. We illustrate these by considering the equation

$$G(X, Y, Z) = H(X, Y, Z)$$

where G and H are non-linear functions of the levels variables X,Y,Z.

1. If we take the percentage change of each side,

$$p_G(X, Y, Z) = p_H(X, Y, Z)$$

and apply the percentage-change differentiation rules which are given in detail in section A.1 of Appendix A, the final result is a linear equation in terms of the percentage changes p_X, p_Y, p_Z of the levels variables X,Y,Z respectively,

$$R(X,Y,Z).p_X + S(X,Y,Z).p_Y + T(X,Y,Z).p_Z = 0$$

where R, S, T are functions of the levels variables X,Y,Z. If R, S, T are evaluated from the initial solution given by the (pre-simulation) database, a linear equation with constant

coefficients results.

2. The alternative is to begin by taking the differential (or change) of both sides of the levels equation giving

d_G(X, Y, Z) = d_H(X, Y, Z)

and then using the change differentiation rules in section A.1 of Appendix A. If the levels variable X has a CHANGE linear variable c_X associated with it,

replace the differential dX by c_X.

If the levels variable X has a percent-change linear variable p_X associated with it,

replace the differential dX by X/100*p_X.

The result is an equation of the form

K(X,Y,Z)*p_X + L(X,Y,Z)*p_Y + M(X,Y,Z)*p_Z = 0

if X,Y,Z all have associated percent-change linear variables, or

D(X,Y,Z)*c_X + E(X,Y,Z)*c_Y + F(X,Y,Z)*c_Z = 0

if X,Y,Z all have associated change linear variables, or some alternative if some of X,Y,Z have associated change linear variables and some have associated percent-change linear variables.

The linearization of levels equations is essentially transparent to you as a user. It happens automatically (without any intervention being required by you). In most, if not all, cases, you will not need to know how TABLO linearizes a particular equation. Accordingly you may prefer to ignore the rest of this section.

Note that, however the equation is linearized, and whether each variable has associated a change or percent-change linear variable, once the relevant functions (for example, the ones referred to above as R,S,T,K,L,M,D,E,F) are evaluated at the initial solution given by the initial database, the result is a system of linear equations

C.z=0

as indicated in section 2.5.1 of GPD-1.

To illustrate how individual levels equations are linearized, we look at two simple examples.

1. Consider the equation

Z = X*Y

where X, Y and Z are levels variables with associated percent-change linear variables p_X, p_Y and p_Z. If we use the percentage-change differentiation rules in section A.1 of Appendix A, we obtain the associated linear equation

$$p_Z = p_X + p_Y.$$

Alternatively, if we use the change differentiation rules in section A.1 of Appendix A, we obtain the associated linear equation

$$Z/100*p_Z = Y*(X/100*p_X) + X*(Y/100*p_Y)$$

2. Consider the same equation as in 1 but suppose that X,Y and Z have associated change linear variables c_X,c_Y and c_Z. Then, using the change differentiation rules in section A.1 of Appendix A, we obtain the linearization

$$c_Z = X*c_Y + Y*c_X.$$

3. Consider the equation

$$Z = X + Y$$

where X, Y and Z are levels variables with associated percent-change linear variables p_X, p_Y and p_Z. If we use the percentage-change differentiation rules in section A.1 of Appendix A, we obtain the associated linear equation

$$p_Z = X/(X+Y)*p_X + Y/(X+Y)*p_Y.$$

Alternatively, if we use the change differentiation rules in section A.1 of Appendix A, we obtain the associated linear equation*

$$Z/100*p_Z = X/100*p_X + Y/100*p_Y.$$

Algorithm Used by TABLO

At present, if allowed to operate in its default mode, TABLO uses change differentiation

* if either the left or right hand side of the equation is zero,
* if there are any change variables in the equation,
* if the operator at the highest levels is + or -, or
* if a SUM occurs anywhere in the equation .

In all other cases TABLO uses percentage-change differentiation.

However, as indicated in section 5.2.8 below, convergence of multi-step calculations may be hindered by percentage-change differentiation. In such a case you can force TABLO to use only change differentiation by selecting option

ACD Always use Change Differentiation of levels equations

from the Options menu presented at the start of TABLO. (This menu is shown in section 2.1 above.)

---

* At present, TABLO is not clever enough to cancel all "/100"s.

2.3 **More Details About Condensation and Substituting Variables**

The main ideas involved in condensing models and substituting out variables have been described in section 3.9 of GPD-1.

In planning substitutions to make with a model, the first thing to keep in mind is that, in order to use a particular equation block to substitute out a given variable, **the number of equations in the equation block must equal the number of components of the variable**. If, for example, you wish to substitute out a variable of the form x(i,j) where indices 'i' and 'j' range over sets COM and IND respectively, then you must use an equation block which has the two quantifiers (all,i,COM) and (all,j,IND) in it.

Even then, an equation containing a variable cannot always be used to substitute out that variable*. Such a substitution is only possible if, via simple rearrangements of the equation, it is possible to get an expression for EVERY component of the variable. In addition, the resulting expression must not involve the variable in question.

An example showing the sorts of rearrangements TABLO can do is given in section 3.9.1 of GPD-1. TABLO can also combine two terms in the variable being substituted out as in, for example,

```
(all,i,COM)  A8(i)*x(i) + z(i) = A9(i)*x(i) + w
```

which is rewritten as

```
(all,i,COM)  [A8(i)-A9(i)]*x(i) = w - z(i)
```

and then used to get the substituting expression

```
(all,i,COM)  x(i) = {1/[A8(i)-A9(i)]} * [w - z(i)]
```

for variable **x**.

However in the following examples, the equation cannot be used to substitute out variable **x**.

(i) Consider the equation

```
(all,i,COM)  x(i) = z(i) + x("c2")
```

in which "c2" is a particular element of the set COM. In this case the only possible substituting expression (the right-hand side) still involves variable **x**.

(ii) Consider the equation

```
x("c2") = z + w.
```

---

* Because TABLO linearizes any levels EQUATIONs before condensation takes place (as explained in section 2.2 above), in this section the term variable refers to a linear variable, that is the change or percentage change in some levels variable. Equation refers to a linearized equation.

Because the occurrence of variable **x** has an element "c2" as an argument (rather than an index such as 'i'), it is not possible to obtain from this equation an expression for ALL components of variable **x**. Hence this equation cannot be used to substitute out the variable **x** (but it could be used to substitute out variable **z**).

(iii) Consider the equation

```
(all,i,MARGCOM)  x(i) = z(i) + w
```

in which MARGCOM is a subset of the set COM over which the variable **x** is defined. Here again this equation cannot be used to obtain an expression for ALL components of variable **x**, only those components in the subset MARGCOM. Hence this equation cannot be used to substitute out the variable **x**.

(iv) Consider the equation

```
SUM(i,COM, A(i)*x(i)) = z + w.
```

Although this equation involves all components of variable **x**, it is not possible to obtain expressions for x(i) for all values of the index 'i'. Indeed, **x** has several components (one for each commodity 'i' in COM) but this is just one equation. One requirement for substitution is that the number of equations in the relevant equation block is the same as the number of components of the relevant variable.

(v) Consider the equation

```
(all,i,COM)(all,j,IND)  x(i) = z(j) + w.
```

Here there are more equations than components of variable **x**. (If COM has M elements and IND has N, then there are MN equations and only M components of **x**.) This equation block cannot be used to substitute out **x** since different values of the index 'j' lead to different expressions for each x(i).

(vi) Consider the equation

```
(all,i,COM)  x(i,i) = z(i) + w
```

where variable **x** now has two arguments, each ranging over the set COM. This equation block could not be used to substitute out **x** because it gives no expression for components x(i1,i2) of **x** in which indices 'i1' and 'i2' are different. Also the number of equations is less than the number of components of **x** in this case.

(vii) Consider the equation

```
(all,i1,COM)(all,i2,COM)  x(i1,i2) = x(i2,i1) + z(i1) + w
```

where again variable **x** has two arguments, each ranging over the set COM. This equation block could not be used to substitute out **x** since we cannot combine the two occurrences as their index patterns are different.

(viii) Consider the equation

(all,t,TIME1) x(t+1) = z(t) + w

in an intertemporal model. This equation cannot be used to substitute out variable **x** because of the offset "+1" in its argument 't+1'.

We state the full set of requirements for a given equation to be used to substitute out a nominated variable occurring in it. The reasons for these should be clear from the examples above. If, during the condensation stage of TABLO, you ask TABLO to make a substitution in which one or more of these conditions does not hold, you will get a message explaining briefly which condition fails; in this case you can still make other substitutions.

Note that the conditions for backsolving are identical to those for substitution.

The requirements for a substitution (or a backsolve) to be possible are as follows:

[1] Every argument of each occurrence of the variable in question must be an index; an element must not occur as an argument. (Examples (i) and (ii) above violate this.)

[2] Every index of the variable in question must be an equation ALL index; a SUM index must not occur. (Example (iv) above violates this.)

[3] Every equation ALL index must occur as an index in each occurrence of the variable in question. (Example (v) above violates this.)

[4] Every index of the variable in question must range over the full set as defined in the VARIABLE statement for this variable; it must not be ranging over a subset of that set. (Example (iii) above violates this.)

[5] In any one occurrence of the variable in question, all indices must be different; no index can be repeated. (Example (vi) above violates this.)

[6] In an intertemporal model, no argument of the variable in question can contain an offset. (Example (viii) above violates this.)

[7] In any two occurrences of the variable in question, the index patterns must be the same. (Example (vii) above violates this.)

Note that the order of doing substitutions can affect whether or not a particular substitution is possible. Consider, for example, the linearized TABLO Input file for Stylized Johansen shown in section 3.5.1 of GPD-1. The equation "Com_clear" can be used to substitute out variable p_XCOM. But if previously equation "Intermediate_com" has been used to substitute out variable p_XC, then equation

"Com_clear" is no longer suitable for substituting out p_XCOM since then this equation contains a term

```
SUM(j,SECT, BCOM(i,j)*p_XCOM(j) )
```

from the substitution of p_XC, as well as the original term p_XCOM(i); these two different index patterns for p_XCOM violate condition [7] above.

If you need to see the new form of any EQUATION or UPDATE after one or more substitutions have been made, you can always do so during Condensation by selecting option 'd' ("Display model's status") and then selecting option '3' or '4' to display the current form of the EQUATION or UPDATE required.

2.3.1 **Looking Ahead to Substitution when Creating TABLO Input Files**

The way your model is defined in your TABLO Input file should take into account the substitutions you anticipate making in the model.

Example 1

For example, if SOURCE is a set with two elements "domestic" and "imported" and variable **x** is declared via

```
VARIABLE (ALL,i,COM)(ALL,s,SOURCE) x(i,s) ;
```

then the equation

```
(ALL,i,COM) x(i,"imported") = C5(i)*z(i)
```

cannot be used to substitute out variable **x** since it violates condition [1] of the substitution rules above. (As a more intuitive explanation, this equation contains no information about those components of **x** with second argument "domestic".)

Suppose you want to substitute out only part of the variable **x**, for example, **x(i,"imported")**, but not **x(i,"domestic")**. One way of achieving this is to define **x** as two separate variables, for example, replace

x(i,"domestic") by **xdom(i)**, and
x(i,"imported") by **ximp(i)**.

The original equation would be rewritten as

```
(ALL,i,COM) ximp(i) = C5(i)*z(i)
```

and could now be used to substitute out all occurrences of the variable **ximp**, leaving **xdom** untouched.

Example 2

Suppose there are two (or more) equations which, between them, give expressions for all components of some variable you wish to eliminate by substitution. For example, using the same variable **x(i,s)** as in Example 1 above, the two equations

```
(ALL,i,COM) x(i,"imported") =  C5(i)*z(i)
 (ALL,i,COM) x(i,"domestic") = F8(i)*cpi
```

between them give values for **x(i,s)** for all values of **i** and **s**. But, because neither equation does so by itself, the substitution could not be made with the equations written in this form. However, if again variable **x(i,s)** is split into the two variables **xdom** and **ximp**, both could be eliminated using these equations.

An alternative to the above, which does not involve splitting the variable **x** into two parts, is to rewrite the two equations as a single equation containing **x(i,s)** values for all **i** and **s**. This rewritten equation can then be used to eliminate variable **x**. This could be done (although somewhat artificially) by introducing a new coefficient

```
COEFFICIENT (ALL,s,SOURCE)(ALL,t,SOURCE) DELSOURCE(s,t) ;
```

and giving it values so that DELSOURCE(s,t) is one if s = t and is zero otherwise, which can be done by the formulas

```
FORMULA (ALL,s,SOURCE)(ALL,t,SOURCE) DELSOURCE(s,t)=0.0;
FORMULA (ALL,s,SOURCE) DELSOURCE(s,s) = 1.0;
```

The two equations could them be rewritten as the single equation

```
 (ALL,i,COM)(ALL,s,SOURCE) x(i,s)
   = DELSOURCE(s,"imported")*C5(i)*z(i)
      + DELSOURCE(s,"domestic")*F8(i)*cpi ,
```

which can now be used to substitute out variable **x**.

A similar example, in which two equations are combined into one, is given in section 8.2 below. The method used there could also be used in Example 2 above, when the equations there would be written using conditional expressions (signalled by "IF" - see section 4.4.5) as the single equation

```
  (all,i,COM)(all,s,SOURCE) x(i,s) =
              IF( DELSOURCE(s,"imported") = 1,  C5(i)*z(i) ) +
              IF( DELSOURCE(s,"domestic") = 1,  F8(i)*cpi )
```

2.3.2 **System-initiated Formulas and Backsolves**

During condensation, TABLO may introduce new COEFFICIENTs and FORMULAs for them if it thinks this may reduce the amount of arithmetic required when GEMSIM or the TABLO-generated program runs; we refer to these as **system-initiated coefficients and formulas**. Similarly it may convert a substitution of your variable into a backsolve (which we then call a **system-initiated backsolve**). In the case of a system-initiated backsolve, note that the values of the

relevant variable are not available on the Solution file; TABLO just chooses to backsolve in order to reduce the amount of arithmetic required in the update calculations.

You do not need to be aware of exactly when this will happen, or to be sure exactly what happens. We provide the examples below to indicate, for those readers who wish to know more about them, the procedures and the reasons for them.

Example of a System-initiated Formula

Suppose that you are substituting out a (linear) VARIABLE x(i,j) with two arguments, and suppose that the equation you are using to substitute it out has another term A(i,j)*y(i), where A(i,j) is a COEFFICIENT and y(i) is a linear VARIABLE. When TABLO is making this substitution, it replaces all occurrences of variable 'x' in all other equations. Suppose that another equation has a term

```
SUM( j, IND, B(i,j)*x(i,j) )
```

in it. When the substitution is made for x(i,j), this equation will contain a term

```
SUM( j, IND, B(i,j)*A(i,j)*y(i) )
```

which can be rewritten as

```
[SUM( j, IND, B(i,j)*A(i,j)] * y(i)
```

where the order of the SUM and product (*) have been changed. Here, if this equation is later used to make a substitution, this complicated term (the sum of the products B(i,j)*A(i,j)) may enter several other equations and have to be calculated several times. Since this calculation must be done at least once, and to forestall it being done several times, TABLO will choose to introduce a new coefficient say C00234(i) and a formula setting

```
(all,i,COM)  C00234(i) = SUM(j,IND, B(i,j)*A(i,j) )
```

[When TABLO introduces new coefficients in this way, it always gives them names Cxxxxx, such as C00234.]

Example of a System-initiated Backsolve

Suppose that you make a substitution which entails replacing all occurrences of a (linear) VARIABLE x(i) by the expression

```
A(i)*y(i) + B(i)*z(i) + C(i)*t
```

in which A(i),B(i),C(i) are COEFFICIENTs and y(i),z(i),t are linear VARIABLEs. If variable 'x' occurs in several UPDATE formulas, this expression would normally be put into each of the UPDATE formulas. This would mean that the expression has to be evaluated several times. If so, TABLO may choose to make a system-initiated backsolve for variable 'x', thus eliminating the need to calculate the above expression more than once.

2.3.3 **Absorption**

Absorbing variables is an alternative (in our view, an inferior one) to omitting them. (See section 3.9.3 of GPD-1 for details about omitting variables.) Absorption is a hangover from the days when condensation was done by modellers with pencil and lots of paper. We only allow absorption as an option in TABLO to retain upwards-compatibility from older versions of GEMPACK. We discourage new users of GEMPACK from using it and encourage users who may have used it in the past to switch to omitting the variables which they previously absorbed.

Any collection of variables that will, in the condensed model, always be exogenous and never given a (non-zero) shock, can be absorbed. TABLO prompts you for

(i) the names of the variables to absorb, and

(ii) the names of the composite variables that will replace the absorbed variables; one composite variable for each equation containing at least one absorbed variable.

The names of composite variables consist of

- a root, which is either selected by you, or defaults to the string '**COM**' (for COMposite), and
- a positive integer, the first value of which is either selected by you or receives a default value. Subsequent composite variables within the same absorption have their integer part incremented by one each time. TABLO makes sure that each composite variable name is unique across all variable names in the model.

For example, if you choose '**bnew**' as the root and 1 as the starting integer, the new composite variable names will be **bnew001**, **bnew002**, and so on (one for each equation block in which one or more of the absorbed variables occur). These composite variables inherit indices from the ALL quantifiers of the equation block in which they occur.

Note that root names chosen must be no longer than 12 characters so that the resulting composite variable names do not exceed 15 characters.

Example

Suppose the original system of equations defining a model consists of just the following three equations.

```
(ALL,i,COM)(ALL,j,IND) A(i)*x(i,j) + y(i) + B(j)*z(j) = 0,
(ALL,i,COM)              y(i) + C(i)*w(i) = 0,
(ALL,j,IND)          SUM(i,COM, D(i,j)*x(i,j) + p(i,j) ) + t(j) = 0.
```

If you choose to absorb variables **x** and **z**, using root **ncomp** and starting integer 10, then

(1) the 1st equation becomes

```
(ALL,i,COM)(ALL,j,IND) y(i) + ncomp010(i,j) = 0,
```

(2) the 2nd is unchanged, and

(3) the 3rd becomes
(ALL,j,IND) SUM(i,COM, p(i,j)) + ncomp011(j) + t(j) = 0.

The new variables are
ncomp010(i,j) (i in COM, j in IND),
ncomp011(j) (j in IND).

The new composite variables are a part of the condensed system, and TABLO automatically creates new equations to define them. These automatic definitions are equivalent to the two following **EQUATION** statements.

```
EQUATION COM-ncomp010 #ABSORPTION - Definition of ncomp010#
(ALL,i,COM)(ALL,j,IND) -ncomp010(i,j) + A(i)*x(i,j) + B(j)*z(j) = 0 ;

EQUATION COM-ncomp011 #ABSORPTION - Definition of ncomp011#
(ALL,j,IND) -ncomp011(j) + SUM(i,COM, D(i,j)*x(i,j)) = 0 ;
```

These new EQUATIONs are not in the condensed system, however.

2.4 **Increasing Parameter Values in TABLO**

When running TABLO, you may receive a message (of the kind described in section 5.5 of GPD-1) saying that you should increase the value of one of the parameters in the code.* The relevant parameter values are set not in the source code of TABLO but in one of the so-called INCLUDE files associated with TABLO; you will need to edit one of these INCLUDE files.

Note that on some machines (for example, Unix machines or VAX/VMS), you may not be able to make the changes yourself, but may have to ask your GEMPACK Manager to make them for you.

To help you find which of these INCLUDE files to edit, we list below the parameters whose values you are most likely to have to increase, grouping them to show which INCLUDE file they are in.

INCLUDE FILE	**Relevant Parameters in it**
TABLE1	DCSRIG, MMNCS, MMNDS, MMNEQ, MMNFL, MMNFM, MMNIG, MMNIN, MMNRD, MMNSS, MMNST, MMNUD, MMNVC, MMNWR
TABLE2	DCSMVC, MMCSTN, MMESTN, MMNCSF, MMNCVU, MMNZD, MMVSTN
CONDENSE	MAXABS, MMACT
TBCDDEC	MMCDLN, MMCESK, MMDCLN, MMCLSK, MMLOOP, MMNIIT
TBGSIM	MMFSK, MMFXSK, MMGSIN, MMGSOP, MMGSRC, MMGSSK
TBSTACK	MMINSK, MMLBSK, MMOPSK, MMRSSK, MMVTSK
VOSTACK	MMOESK, MMSLSK, MMVESK, MMVNSK

* If you only have executable images of the GEMPACK programs, you will not be able to reconfigure TABLO. Either you must reduce the size of the relevant part of your model, or else upgrade to a source-code licence for GEMPACK.

For example, MMNDS in TABLE1 sets a limit on the number of DISPLAY statements that can be handled.

(Don’t change the value of any of these parameters unless you receive a message from TABLO suggesting you do so.)

If necessary, consult your machine-specific documentation to find where the INCLUDE files are stored on disk (and what their suffixes are).

Once you have changed the appropriate parameter value (by editing the INCLUDE file in which its value is set), you must then rebuild all the TABLO libraries and then relink TABLO. Again, consult your machine-specific documentation to see how to do this.

CHAPTER 3

BASIC SYNTAX DESCRIPTION

When implementing an economic model within GEMPACK, you prepare a TABLO Input file specifying the theory of the model. This specification uses the statements listed below. This chapter contains a full description of the syntax of these statements.

[1] **SET**

[2] **SUBSET**

[3] **COEFFICIENT**

[4] **VARIABLE**

[5] **FILE**

[6] **READ**

[7] **WRITE**

[8] **FORMULA**

[9] **EQUATION**

[10] **UPDATE**

[11] **ZERODIVIDE**

[12] **DISPLAY**

In the following TABLO syntax specifications :

- Optional syntax is enclosed between square brackets '[ ]'.
- Each TABLO statement begins with a **keyword**. Keywords and other literals are UPPERCASE bolded, as in '**SET**'.
- A **qualifier** is a word surrounded by round brackets following the keyword. A qualifier describes various subtypes of the same keyword. A **qualifier_list** is either just a single legal qualifier or a list of legal qualifiers separated by commas as in OLD,TEXT,ROW_ORDER. If no qualifier is included, the default values are used. In some cases these default values can be reset by use of a so-called Default statement (see section 3.13 below).

- The meaning and syntax of **quantifier_list** and **expression**, which are used in several syntax descriptions, are given below in section 4.3 (’Quantifier Lists’) and section 4.4 (’Expressions used in Equations, Formulas and Updates’) respectively.

- Necessary user-defined inputs, such as names, are enclosed between angle brackets ’< >’. Details about user-defined input are given below in section 4.2 (’User Defined Input’). In particular, ’<information>’ refers to labelling information, as described in section 4.2.2.

- The breaking up of syntax descriptions over several lines is purely to present the syntax attractively. (All TABLO input is in free form, which means that blank spaces, tabs and new lines can be inserted anywhere in the input and will be ignored by TABLO.)

- Uppercase in the syntax descriptions is used purely to identify the literals. (TABLO makes no distinction between upper and lower case input, which can be mixed for all input including literals. A possible consistent use of upper and lower case is suggested later in section 4.1.2.)

As an example of the general form of a TABLO statement, the following declares a variable PCOM.

```
VARIABLE(PERCENT_CHANGE) (all,i,COM) PCOM(i) #Price of commodity i# ;
```

In this statement

- VARIABLE is the keyword,

- PERCENT_CHANGE is the qualifier,

- (all,i,COM) is the quantifier_list,

- PCOM(i) is a user-defined name with index i,

- # Price of commodity i # is labelling information about this variable, and

- the statement ends with a semicolon ’;’.

3.1 **SET**

A collection of economic objects (for example, the SET of commodities or the SET of industries).

A SET can be defined by giving its size as an integer :

```
SYNTAX | SET [(NON_INTERTEMPORAL)] <set_name> [#<information>#]     |
       |         SIZE <integer> ;                                   |
```

or by specifying its maximum size and giving its size as the value of an integer coefficient (one with no arguments) :

```
SYNTAX | SET [(NON_INTERTEMPORAL)] <set_name> [#<information>#]     |
       |    MAXIMUM SIZE <integer>  SIZE <integer_coefficient> ;    |
```

where <integer_coefficient> must have been declared as an integer coefficient which is a parameter (as described in section 3.3 below),

or, by listing all its elements :

```
SYNTAX | SET [( qualifier )] <set_name> [#<information>#]           |
       |        (<element_1, ... ,element_n>) ;                     |
```

or by specifying its maximum or actual size and saying where its elements can be read from :

```
SYNTAX | SET [(NON_INTERTEMPORAL)] <set_name> [#<information>#]     |
       |     [MAXIMUM] SIZE <integer> READ ELEMENTS FROM            |
       |       FILE <logical_name> HEADER "<header_name>" ;         |
```

Of these four above, only the third can be used for intertemporal sets.

Intertemporal sets can also be declared using the following syntax (where the SET qualifier INTERTEMPORAL is used).

```
SYNTAX | SET (INTERTEMPORAL) <set_name> [#<information>#]
       |     [MAXIMUM] SIZE <integer>
       |      ( <stem>[<initial-element>] - <stem>[<end-element>]) ;
```

In the case just above, both <initial-element> and <end-element> must each be either an integer or an expression of the form

```
<integer_coefficient> + <integer>     OR
<integer_coefficient> - <integer>
```

where <integer_coefficient> must have been declared as an integer coefficient which is a parameter (as described in section 3.3 below).

The possible SET qualifiers are **INTERTEMPORAL** and **NON_INTERTEMPORAL**, of which NON_INTERTEMPORAL is the default.

The square brackets '**[]**' in the last template above (the one for intertemporal sets) do not indicate optional syntax in this case but are used for intertemporal sets whose actual sizes are read in at run time.
See chapter 7 for information about INTERTEMPORAL models, and the definition of intertemporal sets.

Examples

```
SET COM # commodities # SIZE 10 ;
SET COM # commodities # MAXIMUM SIZE 114  SIZE NCOM ;
SET IND # industries # (wool, car, food) ;
SET (NON_INTERTEMPORAL) IND # industries # (ind1 - ind100) ;
SET IND MAXIMUM SIZE 5
      READ ELEMENTS FROM FILE params HEADER "INDE" ;
SET (INTERTEMPORAL) fwdtime
      MAXIMUM SIZE 100 (p[0] - p[NINTERVAL-1]) ;
SET (INTERTEMPORAL) endtime SIZE 1 (p[NINTERVAL]) ;
```

The fourth of these examples illustrates the way lists of element names can sometimes be abbreviated: see section 4.2.1 below for details.

3.2 **SUBSET**

A sub-collection of elements of a previously defined SET (for example, the SUBSET of agricultural commodities).

SYNTAX

SUBSET [**(BY_ELEMENTS)**] <set1_name>
IS SUBSET OF <set2_name> **;**

Alternatively the element numbers in the big set of the elements in the small set can be given via :

SYNTAX

SUBSET (BY_NUMBERS) <set1_name> **IS SUBSET OF** <set2_name>
READ ELEMENT NUMBERS FROM FILE <logical_name>
HEADER "<header_name>" **;**

The possible SUBSET qualifiers are **BY_ELEMENTS** or **BY_NUMBERS**, of which BY_ELEMENTS is the default.

Examples

```
SUBSET AG_COM IS SUBSET OF COM ;
SUBSET (BY_NUMBERS) ag_com IS SUBSET OF com
  READ ELEMENT NUMBERS FROM FILE params HEADER "AGCC" ;
```

Section 4.5 below explains when SUBSET statements are required.

3.3 **COEFFICIENT**

This is the current value of a levels variable. It can occur either as a coefficient of a variable in an EQUATION, or a value of base data, or some value derived from base data via a FORMULA (for example, a share). Coefficients represent real numbers (the default) or integers.

SYNTAX

COEFFICIENT [**(** qualifier_list **)**] [quantifier_list]
<coefficient_name>[**(**index_1, .. ,index_n**)**]
[#<information>#] **;**

The possible COEFFICIENT qualifiers are

REAL or **INTEGER** (of which REAL is the default), and

PARAMETER or **NON_PARAMETER**. The default is NON_PARAMETER for real coefficients and PARAMETER for integer coefficients. The PARAMETER/NON_PARAMETER default can be reset for real coefficients by use of a Default statement (see section 3.13).

COEFFICIENT(PARAMETER)s are constant throughout any simulation whereas COEFFICIENT(NON_PARAMETER)s may be non-constant - see section 4.11 below.

If there are *n* indices in the declaration, this defines an *n*-dimensional array. For REAL or INTEGER coefficients, *n* must be between 0 and 7 (inclusive). The number *n* is referred to as the **dimension** of the coefficient, or its number of arguments or indices.

Examples

```
COEFFICIENT    (all,i,COM) TOTSALES(i)
  # Total sales of commodities #          ;
COEFFICIENT  (all,i,COM)(all,j,IND) INTINP(i,j)  ;
COEFFICIENT (REAL)   GNP   # Gross National Product #   ;
COEFFICIENT (INTEGER)  NCOM  # Size of set COM # ;
COEFFICIENT (REAL, PARAMETER) (all,j,COM) ALPHA(j) ;
COEFFICIENT (INTEGER,NON_PARAMETER) NCOUNT ;
```

See also section 4.4.10 for a description of "Integer Coefficients in Expressions", section 4.7.1 for "How Data is Associated with Coefficients", section 4.4.9 for information about indices in coefficients and section 4.4.11 on "Where Coefficients and Levels Variables Can Occur".

3.4 **VARIABLE**

An economic variable (unknown) that occurs in one or more EQUATIONs. The set of equations is solved to find the value of percentage changes (or actual changes) in the levels variables.

```
SYNTAX | VARIABLE [( qualifier_list )] [quantifier_list]
       |          <variable_name>[(index_1, .. ,index_n)]
       |          [#<information>#] ;
```

The possible VARIABLE qualifiers are

PERCENT_CHANGE or **CHANGE** (PERCENT_CHANGE is the default), and

LINEAR or **LEVELS** (of which LINEAR is the default).

Both of these defaults can be reset by use of Default statements (see section 3.13).

LINEAR variables represent the percentage change or the actual change, depending on the PERCENT_CHANGE / CHANGE qualifier, in the corresponding levels variable.

The declaration of a LEVELS variable X results in

- a COEFFICIENT (with the same name X),
- an associated linear percentage change variable p_X (if the qualifier PERCENT_CHANGE applies) or actual change variable c_X (for qualifier CHANGE), and
- an UPDATE statement for X

in the associated linearized TABLO Input file (see section 2.2.2 above).

The number of indices must be between 0 and 7. This number is referred to as the **dimension** of the coefficient, or its number of arguments or indices.

Examples

```
VARIABLE   (all,i,COM)  p0(i)  #Basic price of commodities # ;
VARIABLE (PERCENT_CHANGE)
    (all,i,COM)(all,s,SOURCE)    xhous(i,s)
 # Household consumption of commodity i from source s #   ;
VARIABLE   phi  # exchange rate #  ;
VARIABLE (CHANGE) delB # Change in Balance of Trade # ;
VARIABLE (LEVELS, CHANGE) (all,i,SECT) X(i) ;
```

See also section 4.4.7 for use of "Linear Variables in Expressions" and section 4.4.9 for information about indices in variables. See section 4.4.11 on "Where Coefficients and Levels Variables Can Occur".

3.5 **FILE**

A file containing data (for example, base data for the model).

SYNTAX
```
FILE [( qualifier_list )] <logical_name>
     ["<actual_name>"] [#<information>#] ;
```

The possible FILE qualifiers are **OLD** or **NEW**, **HEADER** or **TEXT**, and **ROW_ORDER** or **COL_ORDER** or **SPREADSHEET, SEPARATOR =** "<character>".

(a) **OLD** or **NEW** (OLD is the default)
OLD files are used for reading data from a pre-existing file, NEW files for writing data and creating a new file. (Data cannot be written to an OLD file or read from a NEW file.)

(b) **HEADER** or **TEXT** (HEADER is the default)
Files can be GEMPACK Header Array files (as described in section 3.4.4 of GPD-1 and in GPD-3) or TEXT files. Text files are described in section 4.6.1 of this document and also in Appendix C of GPD-1.

(c) **ROW_ORDER** or **COL_ORDER** or **SPREADSHEET**, **SEPARATOR =** "<character>"
These three qualifiers are only relevant when you are writing a text file; that is, they must only be used after both of the qualifiers NEW, TEXT. The default is ROW_ORDER.

SPREADSHEET is similar to row order but there is a separator between data items. The default separator is a comma. To use a different separator,include the qualifier SEPARATOR = followed by the single-character separator surrounded by quotes.
For example, SPREADSHEET, SEPARATOR = ";" would separate values with semicolons ;
Other details about the syntax of text data files and row order, column order and spreadsheet data are given in Appendix C of GPD-1.

Examples

```
FILE io  # Input-output data # ;
FILE (OLD) params  "PAR79.DAT" # parameters # ;
FILE (NEW, TEXT) summary ;
FILE (TEXT,NEW,SPREADSHEET,SEPARATOR=";") table ;
```

See also sections 4.6 on Files and 4.6.1 on Text files, and section 3.4.4 in GPD-1.

3.6 **READ**

An instruction that the values of a given COEFFICIENT are to be read directly from a given FILE or from the terminal.

All the values can be read into a COEFFICIENT :

SYNTAX | **READ** <coefficient_name> **FROM** <location> [#<information>#] ;

or, just a part of the COEFFICIENT can be read :

SYNTAX | **READ** [quantifier_list]
| <coefficient_name>**(**argument_1, ... , argument_n**)**
| **FROM** <location> [#<information>#] ;

In the above, <location> must be

(a) **FILE** <logical_name> **HEADER "**<header_name>**"**
if the file is a Header Array file,

(b) **FILE** <logical_name>
if the file is a text file, or

(c) **TERMINAL**
if the data is to be read from the terminal.

An argument is either an index or the element of a SET, as described in section 4.2.3.

A levels variable can be given as the name of the coefficient being read. (This is because the declaration of a VARIABLE(LEVELS) produces a COEFFICIENT of the same name in the associated linearized TABLO Input file, as explained in section 2.2.2 above.)

Examples

```
READ TOTSALES FROM FILE io HEADER "TSAL" ;
READ (all,i,COM) INTINP(i,"wool") FROM FILE params ;
READ INTINP FROM TERMINAL ;
```

See also section 4.7 on "Reads, Writes and Displays" and section 4.7.2 on "Partial Reads".

At present, data cannot be read into all or part of an integer coefficient with more than 2 dimensions. (But values can be assigned via a formula.)

3.7 **WRITE**

An instruction that the values of a given COEFFICIENT are to be written to a given FILE or to the terminal.

All the values of a COEFFICIENT can be written :

SYNTAX | **WRITE** <coefficient_name> **TO** <location> [#<information>#] **;**

or, just a part of the COEFFICIENT can be written :

SYNTAX | **WRITE** [quantifier_list]
| <coefficient_name>**(**argument_1, ... , argument_n**)**
| **TO** <location> [#<information>#] **;**

In the above, <location> must be

(a) **FILE** <logical_name> **HEADER "**<header_name>**"**
[**LONGNAME** "<long_name>"]
if the file is a Header Array file,
(Here the LONGNAME "<long_name>" is optional. If omitted, the longname will be all blanks.)

(b) **FILE** <logical_name>
if the file is a text file, or

(c) **TERMINAL**
if the data is to be written to the terminal.

An argument is either an index or the element of a SET, as described in section 4.2.3.

A levels variable can be given as the name of the coefficient being written. (This is because the declaration of a VARIABLE(LEVELS) produces a COEFFICIENT of the same name in the associated linearized TABLO Input file, as explained in section 2.2.2 above.)

Examples

```
WRITE TOTSALES TO FILE io HEADER "TSAL" ;
WRITE COMPROD TO FILE basedata HEADER "COMP" LONGNAME
"Production of commodity i by industry j" ;
WRITE (all,i,COM) INTINP(i,"wool") TO FILE params ;
WRITE INTINP TO TERMINAL ;
```

See also section 4.7 on "Reads, Writes and Displays" and section 4.7.2 on "Partial Writes".

At present, data cannot be written from all or part of an integer coefficient with more than 2 dimensions.

3.8 **FORMULA**

An algebraic specification of how the values of a given COEFFICIENT are to be calculated from those of other COEFFICIENTs.

SYNTAX

FORMULA [**(** qualifier **)**] [quantifier_list]
<coefficient_name>**(**argument_1,... ,argument_n**)**=expression **;**

The possible qualifiers are **INITIAL** or **ALWAYS**. The default is ALWAYS for formulas with a real coefficient on the left-hand side and is INITIAL for formulas with an integer coefficient on the left-hand side. In the former case (real coefficient on the LHS), the default can be reset by use of a Default statement (see section 3.13).

FORMULA(INITIAL)s are only calculated during the first step in a simulation, while FORMULA(ALWAYS)s are calculated at every step.

A levels variable can be given as the <coefficient_name> being calculated on the left hand side of a FORMULA(INITIAL). However a levels variable can not be given on the left hand side of a FORMULA(ALWAYS), because a levels variable is automatically updated using its associated percentage change or change at later steps.

A FORMULA(INITIAL) produces a READ statement in the associated linearized TABLO Input file. The FORMULA is used in step 1 of a multi-step calculation while the READ is used in subsequent steps. (See section 4.7.3 below.)

An argument is either an index or the element of a SET, as described in section 4.2.3. See section 4.4 for the syntax of expressions used in FORMULAs.

Examples

```
FORMULA  (all,i,COM)  HOUSSH(i) = HOUSCONS(i)/TOTCONS ;
FORMULA  (all,i,COM) TOTSALES(i)= SUM(j,IND, INTINP(i,j) ) ;
FORMULA  NIND = 10 ;
FORMULA (INITIAL) (all,i,SECT) PCOM(i) = 1.0 ;
```

3.9 **EQUATION**

An algebraic specification of some part of the economic behaviour of the model using COEFFICIENTs and VARIABLEs.

SYNTAX

EQUATION [**(**qualifier**)**] <equation_name> [#<information>#]
[quantifier_list] expression_1 = expression_2 **;**

Either expression_1 or expression_2 can be just the single character 0 (zero).

The possible qualifiers are **LEVELS** or **LINEAR**, of which LINEAR is the default. However the default can be reset by use of a Default statement (see section 3.13).

TABLO converts an EQUATION(LEVELS) to an equivalent associated linear equation as described in sections 2.2.2 and 2.2.3.

See section 4.4 for the syntax of expressions used in EQUATIONs.

Examples

```
EQUATION  HOUSCONS  #Household consumption #
 (all,i,COM) xh(i) = SUM(s,SOURCE, A6(i)*xhous(i,s)) ;
EQUATION   BALTRADE
   -100.0 * delB + E*e - M*m = 0 ;
EQUATION(LEVELS)  eq1  0 = X1 + X2 - A3*X3 ;
```

3.9.1 **"EQUATION(NONE);" Statement**

There is a special statement

EQUATION (NONE) ;

which can be used in a TABLO Input file containing no EQUATION statements (that is, a file used for data manipulation only). This statement must be the first statement in the file. See section 4.12 below for the associated semantic details.

3.9.2 **FORMULA & EQUATION**

As a shorthand way of defining both a FORMULA and an EQUATION at the same time, you can use the ampersand **&** between the keywords FORMULA and EQUATION. The ampersand **&** indicates that there is a double statement equivalent to **both** a FORMULA and an EQUATION at the same time.

```
SYNTAX | FORMULA [(qualifier)] & EQUATION [(qualifier)]                |
       |      <equation_name> [#<information>#]   [quantifier_list]    |
       |<coefficient_name>(argument_1,...,argument_n) = expression;    |
```

This is only possible with a FORMULA(INITIAL) and an EQUATION(LEVELS). If the qualifiers are omitted, it is assumed that these are the qualifiers for this double statement, even if the defaults for the TABLO Input file are set differently.

The double statement must obey the syntax rules for both a FORMULA(INITIAL) and an EQUATION(LEVELS).

The keyword for the next statement after the double statement FORMULA & EQUATION must be included; it cannot be omitted (see section 4.1.1 below).

Examples

```
FORMULA & EQUATION Comoutput
         # Commodity outputs #      (all,i,SECT)
   XCOM(i) = XHOUS(i) + SUM(j,SECT, XCOMIN(i,j)) ;

FORMULA(INITIAL) & EQUATION(LEVELS)
    HOUSE   # Commodity i - household use #
 (all,i,SECT) XHOUS(i) = DVHOUS(i)/PCOM(i) ;
```

The second example is equivalent to

```
FORMULA(INITIAL)
 (all,i,SECT) XHOUS(i) = DVHOUS(i)/PCOM(i) ;

EQUATION(LEVELS) HOUSE   # Commodity i - household use #
 (all,i,SECT) XHOUS(i) = DVHOUS(i)/PCOM(i) ;
```

3.10 **UPDATE**

An algebraic specification of how the values of a given COEFFICIENT are to be updated after each step of a multi-step simulation.

```
SYNTAX | UPDATE [( qualifier )] [quantifier_list]                    |
       | <coefficient_name>(argument_1,...,argument_n)= expression ;|
```

The possible UPDATE qualifiers are **PRODUCT** or **CHANGE*** of which PRODUCT is the default.

The PRODUCT qualifier is used to update a coefficient which, in the levels, is a product of several quantities (one or more). In this case, the expression on the right hand side of the UPDATE equation is the product of the associated percentage change variables.

CHANGE updates are used in all other cases. The expression on the right hand side of the UPDATE equation is the change in the updated coefficient expressed in terms of other coefficients and variables.

An argument is either an index or the element of a SET, as described in section 4.2.3.

Examples

```
UPDATE (all,i,COM)  HOUSCONS(i) = pCOM(i)*xHOUS(i) ;
UPDATE (CHANGE) (all,i,COM) DELB_L(i) = delB + PCOM(i)*delX ;
UPDATE (PRODUCT) (all.i.SECT) Z(i) = x * y(i) * z ;
```

See also section 4.8 on "Updates" and sections 3.5.2 and 3.5.3 of GPD-1.

---

* The pre-Release-5 UPDATE qualifier DEFAULT was renamed PRODUCT (for Release 5) as this more accurately describes its action. Readers familiar with Release 4.2.02 of GEMPACK will have expected to see an EXPLICIT UPDATE statement. While this form of an UPDATE statement is still accepted, we recommend that you do not use statements of this kind in future and that you change any such statements in old TABLO Input files to UPDATE(CHANGE) statements, following the procedure indicated in a footnote of section 3.5.3 in GPD-1. The numerical accuracy of solutions obtained via Gragg's method or the midpoint method increases greatly if you use the UPDATE(CHANGE) form.

3.11 **ZERODIVIDE**

A specification of the default value to be used in a FORMULA when the denominator of a division operation is equal to zero.

The default can be either the value of a scalar coefficient (that is, one which is declared without any indices) :

SYNTAX | **ZERODIVIDE** [**(** qualifier **)**] **DEFAULT** <coefficient_name> **;**

or a real constant :

SYNTAX | **ZERODIVIDE** [**(** qualifier **)**] **DEFAULT** <real_constant> **;**

Alternatively, the particular default value can be turned off to indicate that this kind of division by zero is not allowed :

SYNTAX | **ZERODIVIDE** [**(** qualifier **)**] **OFF ;**

The possible ZERODIVIDE qualifiers are

ZERO_BY_ZERO or **NONZERO_BY_ZERO**, (ZERO_BY_ZERO is the default).

ZERO_BY_ZERO applies when the numerator in the division is zero (zero divided by zero) while NONZERO_BY_ZERO applies when the numerator in the division is a nonzero number (nonzero divided by zero).

Examples

```
ZERODIVIDE DEFAULT  A1 ;
ZERODIVIDE (ZERO_BY_ZERO) DEFAULT  1.0 ;
ZERODIVIDE (NONZERO_BY_ZERO) OFF ;
```

See also section 4.9 for further details.

3.12 **DISPLAY**

An instruction that the values of a given COEFFICIENT are to be displayed (that is, written to a text file called the Display file) for examination by the user. (This is not really part of the model definition, but is rather a useful way of checking that the COEFFICIENTs used in the model have been defined correctly.)

All the values of a COEFFICIENT can be displayed :

```
SYNTAX | DISPLAY <coefficient_name> [#<information>#] ;
```

or just a part of the COEFFICIENT can be displayed :

```
SYNTAX | DISPLAY [quantifier_list]
       |      <coefficient_name>(argument_1, ... , argument_n)
       |      [#<information>#]  ;
```

A levels variable can be given as the name of the coefficient being displayed. (This is because the declaration of a VARIABLE(LEVELS) produces a COEFFICIENT of the same name in the associated linearized TABLO Input file, as explained in section 2.2.2 above.)

Examples

```
DISPLAY  TOTSALES  ;
DISPLAY (all,i,COM) INTINP(i,"wool")
         # Intermediate inputs of wool # ;
```

See also section 4.7 on "Reads, Writes and Displays" and section 4.7.2 for information about partial displays.

At present, integer coefficients cannot be displayed. (But you can WRITE them to the terminal or to a text file.)

3.13 **Setting Default Values of Qualifiers**

It is possible to reset the default values for some of the qualifiers in **some** of the statements described above. Although we refer to these statements as **Default statements**, note that DEFAULT is not a keyword but a qualifier which follows the keyword of the statement where the default is being reset.

SYNTAX | Keyword **(DEFAULT** = qualifier **) ;**

Keyword can be any of **COEFFICIENT**, **VARIABLE**, **FORMULA**, **EQUATION**.

For real COEFFICIENTs, the default can be set to PARAMETER or NON_PARAMETER. (This does not affect the default for integer coefficients, which is always PARAMETER.)

For VARIABLE, the default can be set to LINEAR or LEVELS, or to PERCENT_CHANGE or CHANGE.

For FORMULAs with a real coefficient on the left-hand side, the default can be set to INITIAL or ALWAYS. (This does not affect the default for formulas with an integer coefficient on the left-hand side; for these the default is INITIAL.)

For EQUATION, the default can be set to LINEAR or LEVELS.

Examples

```
EQUATION (DEFAULT = LEVELS) ;
VARIABLE (DEFAULT = CHANGE) ;
VARIABLE (DEFAULT = LEVELS) ;
FORMULA (DEFAULT = INITIAL) ;
COEFFICIENT (DEFAULT = PARAMETER) ;
```

If no Default statements are included, the following defaults apply.

Statement	**Default**
COEFFICIENT	NON_PARAMETER
VARIABLE	LINEAR and PERCENT_CHANGE
FORMULA	ALWAYS
EQUATION	LINEAR

These defaults are the ones that naturally apply in a linearized TABLO Input file so no Default statements are needed in this case.

See sections 3.3.2 and 3.3.3 of GPD-1 for the Default statements often put at the start of a mixed or levels TABLO Input file.

3.14 **TABLO Statement Qualifiers - A Summary**

This lists the different statement qualifiers currently in use. Qualifiers are put in round brackets after the key word they qualify. If there are two or more qualifiers, they can appear in any order, separated by commas, as, for example, in FILE (OLD, TEXT) ..... The defaults (which apply if no relevant qualifier is given) are indicated. However those marked with an asterisk * can be changed as explained in section 3.13 above.

Variable qualifiers

PERCENT_CHANGE or CHANGE	* PERCENT_CHANGE is the default
LINEAR or LEVELS	* LINEAR is the default

File qualifiers

HEADER or TEXT	HEADER is the default
OLD or NEW	OLD is the default
ROW_ORDER or COL_ORDER or SPREADSHEET, SEPARATOR ="<character>"	ROW_ORDER is the default Comma is the default separator

Set qualifiers

INTERTEMPORAL or NON_INTERTEMPORAL	NON_INTERTEMPORAL is the default

Subset qualifiers

BY_ELEMENTS or BY_NUMBERS	BY_ELEMENTS is the default

Update qualifiers

PRODUCT or CHANGE	PRODUCT is the default

Zerodivide qualifiers

ZERO_BY_ZERO or NONZERO_BY_ZERO	ZERO_BY_ZERO is the default

Formula qualifiers

ALWAYS or INITIAL	* ALWAYS is the default when is a real coefficient on LHS INITIAL is the default when is an integer coefficient on LHS

Coefficient qualifiers

NON_PARAMETER or PARAMETER	* NON_PARAMETER is the default for real coefficients PARAMETER is the default for integer coefficients

Equation qualifiers

LINEAR or LEVELS	* LINEAR is the default
NONE	Special statement - see sections 3.9.1 and 4.12

CHAPTER 4

SYNTAX AND SEMANTIC DETAILS

This section contains a comprehensive description of the semantics (and any points of syntax not covered in the previous chapter) for the current version of TABLO.

4.1 General Notes on the TABLO Syntax and Semantics

4.1.1 Input Statements

- A TABLO Input file describing an economic model consists of a collection of separate Input statements.

- Each input statement must usually begin with its keyword (**SET**, **SUBSET**, **COEFFICIENT**, **VARIABLE**, **FILE**, **READ**, **WRITE**, **FORMULA**, **EQUATION**, **UPDATE**, **DISPLAY** or **ZERODIVIDE**) and must end with a semicolon ';'. The keyword can be omitted if the previous statement on the file is of the same type, as in, for example, the following three VARIABLE declarations.

  ```
  VARIABLE (all,i,COM) x(i) ;
           (all,i,COM) x2(i) ;     y ;
  ```

  However, if the previous statement begins with the two keywords **FORMULA & EQUATION** (see section 3.9.2 above), the keyword must be included.

- Although a statement can inherit its keyword from the previous statement as described just above, it is very important to realise that a statement **never inherits qualifiers from the previous statement**. Thus, for example, if you define 3 linear VARIABLEs via the following statements

  ```
  VARIABLE (CHANGE) c_X ; c_Y ; c_Z ;
  ```

  note that, although the first is declared to be a CHANGE variable, the second and third (c_Y and c_Z) will be PERCENT_CHANGE variables (assuming the usual default values for qualifiers are in place). If you want to make them all CHANGE variables, you must explicitly include this qualifier for them all, even if you leave out the keyword in the declarations of the last two, as in

  ```
  VARIABLE (CHANGE) c_X ; (CHANGE) c_Y ; (CHANGE) c_Z ;
  ```

4.1.2 **Upper and Lower Case**

- Input is in free format. Multiple spaces, tabs and new lines are ignored by TABLO.
- Upper and lower case letters are not distinguishable, and can be freely intermixed. A suggested, consistent usage of different cases in linearized TABLO Input files is to use uppercase for keywords and COEFFICIENTs (base data and shares, for example) and lower case for linear VARIABLEs (changes or percentage changes).

4.1.3 **Comments**

- Comments begin and end with an exclamation mark '**!**'. Such comments, which are ignored by TABLO, can go anywhere in the file.

4.1.4 **Strong Comment Markers**

- Strong comment markers, **![[!** at the start and **!]]!** at the end, can be used to comment out sections of text which already contain ordinary comments indicated by '!' or even other strong comment markers. An example follows.

```
![[! Strong comment includes ordinary comments
     and previously active text
     ! ordinary comment ! previously active text ! old comment!
       Strong comment ends with        !]]!
```

These strong comment markers accumulate, so that one !]]! cancels out one previously active ![[! and so on, as in the next example.

```
![[! strong comment begins   ![[! and continues   !]]!
      and still continues, ending with !]]!
```

Note that the start of a strong comment should not usually be made in the middle of an existing ordinary comment, as the next example shows.

```
! ordinary comment starts ![[! strong comment - ends with !]]!
    But this text is still inside the ordinary comment which
      needs another exclamation mark to end it   !
```

4.1.5 **Reserved (special) Characters**

- There are three reserved characters, namely

  ; which terminates an input statement

  # the delimiter for labelling information

  ! the delimiter for comments

  We recommend that you do not use any of these reserved characters except for their defined function. For example, even though TABLO ignores the content of comments, you should still not include semicolons (**;**) within them.

4.2 **User Defined Input**

The syntax descriptions in chapter 3 referred to the following types of user-defined input.

4.2.1 **Names**

- All names of COEFFICIENTs, VARIABLEs, SETs, set elements, indices, EQUATIONs and logical FILEs consist of letters, digits and/or underscores '_' and/or '@'s, and **must commence with a letter**. Examples are SALES, X3, X, TOT_SALES, p_XH, c_abc, Focc, xcom, X3@Y2.

- The maximum lengths of names are as follows :

Name of object	**Maximum length**	
Header	Must be 4 characters exactly	
COEFFICIENT	12	
SET	12	
Index	12	
VARIABLE(LINEAR)	15	
VARIABLE(LEVELS)	12	
EQUATION	20	
Logical FILE	20	
Set element	12	
Intertemporal element stem	6	(see section 7.2.1)
Actual file name	40	
Real constant	20	
Integer constant	18	
Integer set size	9	

- Note that duplication of a name for two different purposes is not allowed. For example, you cannot use 'X1' to denote a coefficient and 'x1' to be a variable. (Remember that input is case-independent.)

- **Certain names** (SUM, IF, function names and operators used in conditionals) **are reserved**, which means that they cannot be used as the names of coefficients, variables, sets, set elements or files. These reserved words are listed below. [We include 'PROD' in this list (even though it is not reserved in the present version of TABLO) because we expect to use it later to introduce PRODUCTS (like SUMs) as in, for example, PROD(i,COM,P(i)).]

Reserved Words

ALL	(quantifier list)
SUM PROD	(sum and product)
IF	(conditional expressions)
LE GE LT GT EQ NE	(comparison operators)
NOT AND OR	(logical operators)
ABS MAX MIN SQRT EXP LOGE LOG10	(functions)
ID01 ID0V	(special functions)

- Actual file names can include any characters legal on your computer (except for the reserved characters ';', '#' or '!').

- **Lists of set elements** such as

ind1, ind2, ind3, (and so on up to) ind24

can be abbreviated using a dash. For example, the above could be abbreviated to

ind1 - ind24.

Such abbreviations can be mixed with other element names to give lists such as

(cattle, grain1 - grain4, services1 - services12, banking).

There are two ways of implying a list of element names. The first is illustrated above. A second has the number parts expanded with leading zeros such as

ind008 - ind112

which is an abbreviation for

ind008, ind009, ind010, ind011 (and so on up to) ind112.

In this second case, the number of digits at the end must be the same before and after the dash. For example, the following are allowed

ind01 - ind35, com0001 - com0123,

while

ind01 - ind123 is not allowed.

4.2.2 **Labelling Information (Text between hashes #)**

- Text between delimiting hashes ('#') must be contained on a single input line.
- VARIABLE and SET labelling information appears in GEMPIE Print files reporting simulation results.
- VARIABLE, EQUATION and SET labelling information appears in SUMEQ maps (see section 8.1.1 of GPD-1).
- When a COEFFICIENT is DISPLAYed, any labelling information in the DISPLAY statement is shown. If there is none, any labelling information in the statement declaring the COEFFICIENT is shown on the display file.
- At present, labelling information on FILE, READ and WRITE statements is not used.

4.2.3 **Arguments - Indices and Set Elements**

- An argument is either an index or the element of a SET.
- Arguments of variables and coefficients are separated by commas and enclosed in round brackets. They follow immediately after the variable or coefficient name. When a variable or coefficient is declared in a VARIABLE or COEFFICIENT input statement, all arguments will be (dummy) indices. In all other occurrences of arguments, they could be indices or actual elements from the relevant set. Set element names are enclosed inside double quotes. Examples of coefficients followed by arguments are X3(i,j) and INTINP(i,"wool").
- The arguments must range over the appropriate sets, in the order specified in the coefficient or variable declaration. For example, if variable x3 is declared via

  VARIABLE (ALL,i,S) (ALL,j,T) x3(i,j) ;

  then, in every occurrence of x3,

  - the first argument must either be an index ranging over the set S (or some set S1 declared, via a SUBSET declaration, to be a subset of S) or be an element of the set S (in which case S must have been declared using a form of SET syntax which lists all the set elements),

- the second argument must be either an index ranging over T (or a subset of T) or be an element of T.

o Arguments which are set element names are only allowed if the relevant set has fixed element names (that is, names appearing on the TABLO Input file in the SET statement), rather than names read at run-time.

As an example, consider the use of the element name "imported" in the formula below.

```
COEFFICIENT (all,i,SOURCES) X(i) ;
FORMULA X("imported") = TOTIMPSALES ;
```

This is fine if the set SOURCES is declared via

```
SET SOURCES (domestic, imported) ;
```

but is not allowed if SOURCES is declared via

```
SET SOURCES READ ELEMENTS FROM FILE data HEADER "SSRC" ;
```

If this restriction causes a problem, note that it should be relatively easy to get around it by defining sets with just one element. For example, in the example above, define a set IMP with just "imported" in it, declare IMP to be a subset of SOURCES and rewrite the formula using the set IMP, as shown below.

```
SET SOURCES READ ELEMENTS FROM FILE data HEADER "SSRC" ;
SET IMP (imported) ;
SUBSET IMP IS SUBSET OF SOURCES ;
FORMULA (all,s,IMP) X(s) = TOTIMPSALES ;
```

4.3 **Quantifier lists**

A quantifier list consists of one or more quantifiers, concatenated together in the input. A quantifier is of the form

QUANTIFIER SYNTAX

(ALL, <index_name>**,** <set_name> [:<condition>] **)**

Examples of Quantifier Lists

```
(ALL, i, COM)
(all,i,COM)(all,j,IND)
(all,i,COM: TPROD(i) > 0)
```

The optional condition is a logical expression which may restrict the range of the index involved. For example, the

condition "TPROD(i) > 0" in the third example above restricts the index i to range over only those commodities i (elements of the set COM) for which TPROD(i) is greater than zero (which may not be all things in COM).

Conditions are only allowed in quantifiers in FORMULA(ALWAYS) and UPDATE statements. (They are not allowed in quantifiers in READs, FORMULA(INITIAL)s, EQUATIONs, WRITEs, DISPLAYs or declarations of VARIABLEs or COEFFICIENTs.)

4.4 **Expressions Used in Equations, Formulas and Updates**

Expressions occur in equations, formulas and updates.

4.4.1 **Operations Used in Expressions**

The following operations can be used in expressions.

Addition (**+**),

Subtraction (**-**),

Multiplication (*****),

Division (**/**)

Powers (**^**).

- Note that **^** means "to the power of". For example, X^3 means X to the power of 3 (that is, X cubed) while X^Y means X to the power Y. The "Y" in Z^Y is referred to as the **exponent**.
- The order of processing operators in expressions is the usual one; that is, brackets are done first, followed by unary operators, followed by binary operators. The binary operation **^** is done before the binary operators ***** and **/** (which have the same precedence), which are done before the binary operators **+** and **-** (which are also of equal precedence). For example, **-A+B^C/D** is processed as **((-A) + ((B^C)/D))**. If in doubt, use additional brackets.
- A multiplication operation MUST be shown explicitly whenever it is implied - for example A6(i)SALES(i) is incorrect and must be written as A6(i)*SALES(i).

4.4.2 **Sums over Sets in Expressions**

Sums over sets or subsets can be used in expressions, using the following syntax :

SUM SYNTAX

SUM (<index>**,** <set_name>[:<condition>], expression**)**

If, for example the set IND has two elements "car" and "food" then

SUM(i,IND,A6(i)*SALES(i))

means

A6("car")*SALES("car") + A6("food")*SALES("food").

As for quantifiers, the optional condition is a logical expression which may restrict the number of things summed. For example, with IND as just above, if A6("car") is -1 and A6("food") is 2 then

SUM(i,IND: A6(i) > 0, A6(i)*SALES(i))

will be equal to just A6("food")*SALES("food").

- Pairs of brackets **[]** or **{}** can be used as alternatives to the pair () in SUM(), as in, for example,

SUM[i,IND, A6(i)*SALES(i)],
SUM{i,IND: A6(i) > 0, A6(i)*SALES(i) }.

4.4.3 **Brackets in Expressions**

- Pairs of brackets **()**, **[]** or **{}** can be used to express grouping in expressions. They can also be used with SUMs (see section 4.4.2), IFs (see section 4.4.6) and to surround function arguments (section 4.4.4). You can use whichever pair makes the expression most readable.

- In quantifiers (using the ALL syntax in section 4.3), round brackets **()** are required; [] and {} cannot be used.

- Keyword qualifiers must be surrounded by **round** brackets **()**, as in, for example, EQUATION (LINEAR).

- Square brackets in intertemporal sets **[ ]** indicate flexible set sizes where the set size is read in at run time - see chapter 7.

4.4.4 **Functions**

- Certain functions can be used in expressions. Those recognised at present are

ABS	ABS(x) is the absolute value of x
MAX	MAX(x1,x2,x3) is the maximum of these 3; MAX can take 2 or more arguments
MIN	MIN(x1,x2) is the minimum of these 2; MIN can take 2 or more arguments
SQRT	SQRT(x) is the square root of x
EXP	EXP(x) is E raised to the power x where E is the base of the natural logarithms
LOGE	LOGE(x) is log to the base E of x (natural log)
LOG10	LOG10(x) is log to the base 10 of x
ID01	ID01(x) is x if x is not equal to 0, or is 1 if x=0
ID0V	ID0V(x,v) is x if x is not equal to 0, or is v if x=0

  In the above, **ID01** stands for "**ID**entity function except that **0** maps to **1**". **ID0V** stands for "**ID**entity function except that **0** maps to a specified **V**alue". (The "identity" function is the one mapping x to x for all relevant x.) Note that there is a zero ’0’ in these names, not an oh ’o’.

- Function arguments must be surrounded by pairs of brackets **()**, **[]** or **{}** as in, for example, ABS[X], MIN{X1,X2+1}.

- The arguments of these functions can be expressions involving COEFFICIENTs, levels VARIABLEs and/or real numbers, but cannot include linear VARIABLEs. For example, SQRT(C1+5) is accepted if C1 is a COEFFICIENT or levels VARIABLE but not if it is a linear VARIABLE.

- Only a limited list of functions can be used in levels EQUATIONs, namely
  SQRT, EXP, LOGE, LOG10.

4.4.5 **Conditional Operators**

- Conditions can be specified to restrict SUMs and ALLs. The condition is specified after a colon ’:’ as in

  SUM(j,IND: XINP(j) > 0, XINP(j)*y(j))

  or

  (ALL,j,IND: XINP(j) > 0)

  We recommend reading the colon ’:’ as "such that" (just as in set notation in mathematics).

  The judicious use of conditionals may result in GEMSIM or

TABLO-generated programs running much more quickly. It may also help to specify complicated scenarios such as taxes applied at increasing rates depending on income (though, in such cases, care must be taken to use this only in situations where the underlying functions and their derivatives are continuous - that is, do not make discrete jumps as their inputs vary).

Examples of the use of conditional SUMs can be found in sections 8.3 and 8.4 below.

At present conditional SUMs are allowed everywhere that SUMs are allowed but conditional ALLs are allowed only in FORMULA(ALWAYS)s and UPDATEs, not in EQUATIONs, READs, WRTIEs, DISPLAYs or FORMULA(INITIAL)s.

The syntax for conditional SUMs and ALLs is as follows.

SUM(<index>, <set> : <condition> , <expression to sum>)

ALL(<index>, <set> : <condition>)

- Conditions specified in ALLs or SUMs can depend on the data of the model but not on the changes or percentage changes in the data. Conditions can involve **coefficients or levels variables (but not linear variables)** of the model. The operations in the conditions may involve comparing real numbers using the operations below. In each case there is a choice of the syntax to be used in TABLO Input files to express these: either a letter version or a symbol version is available, as indicated below.

Comparison Operator	**Letter Version**	**Symbol Version**
less than	LT	<
less than or equal to	LE	<=
greater than	GT	>
greater than or equal to	GE	>=
equals	EQ	=
not equal to	NE	<>

Note that no space is allowed between the two characters in the symbol versions <=, >=, <> of LE, GE and NE. When using the letter versions, it is often obligatory to leave a space before the operator and best to leave one after it, such as in "X(i) GT Y(j)".

The logical operators **AND**, **OR** and **NOT** can also be used. Thus compound conditions such as

[X(i) > 0] AND [Y(i) LT (Z(i) + W(i))]

are possible.
Note that the operations +,-,*,/ and ^ can be used in the numerical part of these conditions. For example,

[{X(i)+Y(i)} * Z(i) > 10.0]

The precedence rules for AND, OR and NOT are that

NOT behaves like '-' in arithmetic, while
AND and OR behave like '*' and '+' respectively.
Thus, for example,

NOT A(i) > B(i) AND C(i) < 1 OR D(i) GT 5.3

really means

[{ NOT [A(i) > B(i)] } AND {C(i) < 1}] OR {D(i) GT 5.3} .

If in doubt, we recommend using brackets liberally to make your meaning unambiguous.

4.4.6 **Conditional Expressions**

- The IF syntax shown below can be used as part of any expression, including expressions in equations and on the right hand side of formulas and updates.

IF SYNTAX

IF **(** <condition> **,** <expression> **)**

The value of the above **conditional expression** is equal to the value of <expression> if <condition> is true, otherwise the value is zero.

For example,

FORMULA (all,i,COM) A(i) = B(i) + IF(C(i) >= 0, D(i)) ;

sets

A(i) = B(i) + D(i) if C(i) is positive or zero, or
A(i) = B(i) if C(i) < 0.

For other examples, see section 8.2 and the comment after Example 2 in section 2.3.1.

- Pairs of brackets **[]** or **{}** can be used as alternatives to the pair () in IF(), as in, for example,

IF[C(i) >= 0, D(i)].

4.4.7 **Linear Variables in Expressions**

- In the following section, a **linear variable** is a variable representing the change or percentage change of a levels quantity. A linear variable can be declared using VARIABLE(LINEAR) or as the change (c_...) or percentage change (p_...) associated with a VARIABLE(LEVELS).

- Linear variables cannot occur in a formula or in the conditional part of an ALL, IF or SUM. Division by an expression involving linear variables is not allowed in an equation.

- In the following, a **linear equation** is one declared by a statement EQUATION(LINEAR) or just by EQUATION if the default for equations is not reset (as described in section 3.13). Similarly a levels equation is one declared by EQUATION(LEVELS).

  In a linear equation, coefficients (or levels variables) must multiply linear variables, and, in such a multiplication, coefficients must go on the *left* and linear variables must go on the *right*. For example, if **xind** and **xcom** are linear variables and A6, SALES are coefficients, then

  A6(i)*__xcom(i)__ + (SALES(i)/A6(i))*__xind(i)__ is correct.

  But __xcom(i)__*A6(i) is incorrect,

  and

  SUM(i,S,A6(i)*__xcom(i)__) * SALES(j) is incorrect.

  A coefficient and a linear variable cannot be added. For example,

  __xcom(i)__ + A6(i) is incorrect.

  Similarly a levels variable and a linear variable cannot be added.

- Every term in a linear equation must contain a linear variable part. For example

  A6(i)*__xcom(i)__ + SALES(i) = 0 is incorrect.

  This is not a sensible equation because the second term contains no linear variable.

- The right-hand-side of a CHANGE UPDATE can contain any legal expression involving coefficients and variables. However, the right-hand-side of a PRODUCT UPDATE statement can only contain PERCENT_CHANGE linear variables multiplied (via '*') together. Each PRODUCT update statement is translated automatically by TABLO into the appropriate algebraic expression. For example,

```
UPDATE (all,i,COM) DVHOUS(i) = p_PC(i)*p_XH(i) ;
```

(which by default is a PRODUCT update) is the same as

```
UPDATE (CHANGE) (all,i,COM) DVHOUS(i) =
            DVHOUS(i)*[ p_PC(i)/100 + p_XH(i)/100 ] ;
```

In algebraic terms, both these updates are the same as the algebraic expression:

DVHOUS(i)_Updated=DVHOUS(i)*[1 + p_PC(i)/100 + p_XH(i)/100]

- Linear variables and COEFFICIENT(NON_PARAMETER)s are not allowed in levels equations. Expressions in levels equations can only contain levels variables, parameters and constants (and of course operators).

4.4.8 **Constants in Expressions**

- As well as using coefficients and variables in expressions, you can use ordinary numbers (real or integer) written as decimals if necessary. Examples are

  16.54 -23 1 0.0

- Real numbers should not be written using exponent notation. Don’t use for example, 1.3E12.

- Especially when used as an exponent (that is, in the "B" part of an expression of the form A^B), it may be best to write integers without a decimal point.*

  For example, write A^(C-2) rather than A^(C-2.0)

4.4.9 **Indices in Expressions**

- When used as arguments of a coefficient or variable, each index must be what is called *active* - that is, be an ALL index or a SUM index still inside the scope of that ALL or SUM. No index which is still active can be re-used as an ALL or SUM index. If these rules are not followed, semantic problems will result. The examples below should make this clear.

---

* This is because, if A is negative, some Fortran compilers will evaluate A^B when B is an integer but will not evaluate it if B is the same real number.

Examples

(a) The index 'j' in A7(i,j) below is not active.

```
 FORMULA  (all,i,IND)  A6(i) = A7(i,j)  ;    !incorrect!
```

(b) The SUM index 'j' below is already active.

```
                                                    !incorrect!
 FORMULA  (all,j,IND)  A6(j) = SUM( j, IND, B7(i,j) ) ;
```

(c) The index 'j' in A8(j) below is not active because it does not fall within the scope of the SUM.

```
                                                    !incorrect!
 FORMULA   T1 = SUM( j, COM, A6(j) )  + A8(j)  ;
```

- Every index quantified at the beginning of a formula *must* be used as an index of the coefficient on the left hand side of the formula. For example,

  ```
  FORMULA (all,i,COM)(all,j,IND) A6(i) = 28 ;      !incorrect!
  ```

  is not allowed.

- The same coefficient does not usually occur on both sides of a formula. If the same coefficient **does** occur on both sides, then there must be NO overlap between its components or parts on each side. For example,

  ```
  FORMULA (all,i,COM)
    SH1(i,"domestic") = 1.0 - SH1(i,"imported") ;    !correct!
  ```

  is allowed, but

  ```
  FORMULA (all,i,COM)
    SH2("cars",i) = 1.0 - SH2(i,"shops")*A6 ;      !incorrect!
  ```

  is not allowed because the component SH2("cars","shops") of SH2 occurs on both sides. In particular, the following type of formula is NOT ALLOWED:

  ```
  FORMULA (all,i,COM) x(i) = x(i) + y(i) ;         !incorrect!
  ```

4.4.10 **Integer Coefficients in Expressions and Elsewhere**

- Integer coefficients can be used in formulas and equations much like real coefficients. When used in an equation, or in a formula whose left-hand side (LHS) is a real coefficient, integer coefficients are treated exactly like real ones.

- In formulas whose LHS coefficient is an integer coefficient, the following restrictions hold.

  * Only integer coefficients can occur on the RHS. (However, real coefficients can occur in SUM or IF conditionals.)

  * No real numbers (those with a decimal point such as 12.3, as distinct from integers such as 123) can occur on the RHS.

  * Division is not allowed.

- Integer coefficients may be involved in the equations of a model. If so, these coefficients cannot change their values between the steps of a multi-step simulation*, and so must be parameters of the model. Hence

  (i) any integer coefficients occurring in levels or linear equations must have been declared as PARAMETERs (following the syntax set out in section 3.3), and

  (ii) integer coefficients cannot be updated (that is, cannot be on the left-hand side of an UPDATE statement).

- To make it easy for you to satisfy (i) of the previous point,

  (a) the default for integer coefficients is always set to PARAMETER (rather than NON_PARAMETER). [The default statement COEFFICIENT(DEFAULT=...) only applies to real coefficients - see section 3.13.]

  (b) FORMULAs with integer coefficients on the left-hand side are by default assumed to be FORMULA(INITIAL), rather than FORMULA(ALWAYS).

- On occasions, it may be useful to have integer coefficients which are not parameters but which can change their values between steps of a multi-step calculation. (You can perhaps use such coefficients to report information such as the number of entries in an array which exceed some specified value at each step.)

  Such coefficients cannot occur in an equation (as explained above) but can be written to the terminal at each step if you select option DWS (see section 5.6.2) when you run GEMSIM or the TABLO-generated program.

  To overcome the defaults set, as described above, you will need to include an explicit NON_PARAMETER qualifier when you declare such a coefficient, and will need to include an explicit ALWAYS qualifier with any FORMULA having such a coefficient on the left-hand side.

---

* The multi-step solution methods in GEMPACK are based on the notion of small changes. Integers cannot change by small amounts and still remain integers.

- Integer coefficients used in setting sizes of sets must be parameters (see sections 3.1 and 4.5.1).

- Integer coefficients used in specifying the elements of an intertemporal set must be parameters (see section 3.1).

- If coefficients with integer values occur as exponents in an expression, there may be some advantage in declaring them as COEFFICIENT(INTEGER)s, for the reason explained in section 7.3 below.

- Using integer coefficients in formulas can be quite useful in setting ZERODIVIDE defaults. For example if IND is a set declared via

  ```
  SET IND MAXIMUM SIZE 120 SIZE NO_IND ;
  ```

  (where NO_IND is an integer coefficient), then the following sets a ZERODIVIDE default which adjusts sensibly for any set IND:

  ```
  COEFFICIENT RECIP_NIND ;
  FORMULA RECIP_NIND = 1/NO_IND ;
  ZERODIVIDE DEFAULT RECIP_NIND ;
  FORMULA (all,i,IND) X(i) = Y(i)/SUM(j,IND,Y(j)) ;
  ZERODIVIDE OFF ;
  ```

  (If all the Y(j) are zero then each X(i) will be set equal to RECIP_NIND.)

4.4.11 **Where Coefficients and Levels Variables Can Occur**

Some details of the syntax of levels variables and levels equations have been given in the preceding sections.

In the following section, a brief summary is given of different ways of representing levels quantities in the model using real COEFFICIENTs and VARIABLE(LEVELS).

In a linearized representation of a model, a COEFFICIENT represents the current value of a levels variable. This is why, as explained in section 2.2.2 above, every VARIABLE(LEVELS) statement gives rise to a COEFFICIENT with the same name in the associated linearized TABLO Input file automatically produced by TABLO. (The associated linear variable has a different name, namely one with "p_" or "c_" added according as to whether it is a percentage-change or change variable.) Thus there are three types of COEFFICIENTs.

(i) Those declared explicitly via COEFFICIENT(NON_PARAMETER) statements or just COEFFICIENT (if the default has not been reset to PARAMETER as in section 3.13). They are allowed in in most statements including FORMULA(INITIAL)s. If read or given an initial value by a FORMULA(INITIAL), they will be updated if an UPDATE statement for them is included. However they are not allowed in EQUATION(LEVEL)s because TABLO does not know what the associated percentage change or change

linear variable is.

(ii) Those arising from VARIABLE(LEVELS) declarations. These are allowed in levels EQUATIONs and FORMULA(INITIAL)s. The associated linear variable (beginning with "p_" or "c_" as appropriate) can occur in EQUATION(LINEAR)s. They can occur in most statements. Their values can be initialised via a READ or FORMULA(INITIAL). If levels variables are initialised, their values are automatically updated using the associated linear variable. Because of this, levels variables cannot occur on the left-hand side of a FORMULA(ALWAYS) or an UPDATE statement.

(iii) Parameters declared via COEFFICIENT(PARAMETER) statements. These are levels variables which are constant. Parameters can occur in levels and linear equations and in FORMULA(INITIAL)s. They are not allowed on the left-hand side of a FORMULA(ALWAYS) or an UPDATE statement since they are constant throughout a multi-step calculation.

All three types can be initialised at the **first step** of a multi-step calculation by reading from a file via a READ statement or by a formula given in a FORMULA(INITIAL) statement. COEFFICIENT(NON_PARAMETER)s of type (i) above can also be initialised at the first step (and every subsequent step) via a FORMULA(ALWAYS).

At **subsequent steps**,

(i) the values of COEFFICIENT(NON_PARAMETER)s are either given via an UPDATE statement or by a FORMULA(ALWAYS). If one of these COEFFICIENTs is read or initialised via a FORMULA(INITIAL), and if no UPDATE statement is given for it, it remains constant (that is, it is effectively a parameter).

(ii) the value of the COEFFICIENT arising from a levels variable is updated from its associated change or percentage-change variable.

(iii) a COEFFICIENT(PARAMETER) remains constant.

After the **final step**, the data files are updated to reflect the final values of any COEFFICIENTs or levels variables whose values were read initially, but final values of COEFFICIENTs or levels variables initialised via a FORMULA(INITIAL) are not shown on these updated data files.

All three types of coefficients can be used in conditions (that is, the condition of a SUM or ALL) in situations where these are allowed.

The following table summarises which quantities can occur in which types of statements, and also which ones are illegal.

Statement	**Permitted**	**Illegal**
EQUATION(LINEAR)	Linear variables Coefficients Levels variables Constants Operators	Non-parameter int coeff.
EQUATION(LEVELS)	Levels variables Parameter coeff. Constants Operators	Linear variables Non_parameter coeff.
FORMULA(ALWAYS)		
Left hand side	Non_parameter coeff.	Levels variables Parameter coeff. Linear variables
Right hand side	Coefficients Levels variables Constants Operators	Linear variables
FORMULA(INITIAL)		
Left hand side	Coefficients Levels variables	Linear variables
Right hand side	Coefficients Levels variables Constants Operators	Linear variables
UPDATE(PRODUCT)or(CHANGE)		
Left hand side	Non_parameter coeff.	Parameter coeff. Levels variables Linear variables Integer coeff.
UPDATE(PRODUCT)		
Right hand side	Linear variables Operator ’*’ for multiplication	Coefficients Levels variables Constants Other operators
UPDATE(CHANGE)		
Right hand side	Coefficients Linear variables Levels variables Constants Operators	
SUM conditions for FORMULAs, EQUATIONs ALL conditions for FORMULAs	Coefficients Levels variables Constants Comparison operators	Linear variables
SUM, ALL conditions for UPDATE(CHANGE)	as above plus Linear variables	

4.5 **Sets and Subsets**

4.5.1 **Sets**

- The size of a set can be determined at run time from the value of an integer coefficient. In a multi-step calculation, the size of each set is determined during the first step, and does not vary in subsequent steps. (For this reason, the integer coefficient determining the set size must be a parameter, constant throughout the simulation).

4.5.2 **Subsets**

TABLO checks that indices range over appropriate sets, as described in section 4.2.3 above. SUBSET statements may be necessary for this to be done correctly. Suppose, for example, that you need the formula

```
(all,i,MARGCOM) X(i) = C1(i) + Y(i) ;
```

where coefficients X, C1, Y have been declared via

```
COEFFICIENT (all,i,COM) C1(i) ;
COEFFICIENT (all,i,MARGCOM) X(i) ;   (all,i,MARGCOM) Y(i) ;
```

in which the sets COM (all commodities) and MARGCOM (the margins commodities) are defined by

```
SET COM (wool, road, rail, banking) ;
SET MARGCOM (road, rail) ;
```

In the formula above, *i* ranges only over MARGCOM but C1 has been defined to have one index which must be in COM. When MARGCOM is declared to be a subset of COM via the SUBSET statement

```
SUBSET MARGCOM IS SUBSET OF COM ;
```

TABLO can tell that *i* in MARGCOM is therefore also in COM. Otherwise TABLO will think that the index *i* in C1(i) is not in the expected set and an error will result.

- In the case of SUBSET (BY_ELEMENTS), which is the default for SUBSET, the two sets must already have been defined in separate SET statements which assign elements to the sets (either explicitly via a list or via an instruction to read the set elements from a file).

- Not all subset relations need to be declared explicitly. If you have declared set A to be a subset of set B and have declared set B to be a subset of set C, you do not need to declare that set A is a subset of set C. This will be inferred from the other two subset declarations.

- Data read in a SUBSET (BY_NUMBERS) statement must be integer (not real) data. Integer 1 refers to the first set element in the original SET, integer 2 refers to the second and so on. So if the data read in is 1 2 5 the subset contains the 1st, 2nd and 5th elements of the set. At run time, GEMSIM or the TABLO-generated program checks that the number of integers at this header on the data file matches the size of the small set in the SUBSET statement. (If the small set has size 4, exactly 4 integers are expected on the data file at the specified header.) It also checks that all the integers at the specified header are in the right range and are distinct.

- The two sets in a SUBSET statement must be of the same type - INTERTEMPORAL or NON_INTERTEMPORAL.

  For intertemporal sets with fixed elements, the small set cannot have "gaps" in it. For example, the following would cause an error.

```
SET (INTERTEMPORAL) time0 ( p0 - p10 ) ;
SET (INTERTEMPORAL) time3 ( p0, p2, p4, p6, p8, p10 ) ;
SUBSET time3 IS SUBSET OF time0 ;                !incorrect!
```

  [This is because arguments such as ’t+1’ must have an unambiguous meaning. In the example above, if ’t’ were in ’time3’ and t equals ’p0’, we would not know if ’t+1’ refers to ’p2’ (the next element in ’time3’) or to ’p1’ (the next element in ’time0’).]

4.6 **Files**

- A logical filename is *always* required in a FILE statement, since all references to files in READ and WRITE statements use logical names, *never* actual filenames. When no actual filename is given, GEMSIM or the TABLO-generated program will prompt for the actual filename when it is run. This has the great advantage of allowing you to use different base data without needing to rerun TABLO. If you include an actual filename in your FILE declaration then

  * it must be a valid filename on the computer you intend running GEMSIM or the TABLO-generated program on (which means it may be not a valid name on other computer systems), and

  * if you want to change to different base data, you will need to rerun TABLO to regenerate your model from scratch, after changing the actual filename (in the FILE declaration) to the name of the new base data file.

  We recommend that you do not specify actual filenames in FILE statements, unless you have good reason to do so.

- Files can be GEMPACK Header Array files (see GPD-1, section 3.4 and GPD-3) or text files (see section 3.4 and Appendix C of GPD-1 and section 4.6.1 below for more details). Real and integer data can be read from, or written to, these files by GEMSIM or TABLO-generated programs. These programs can read character data from a Header Array file as part of a SET statement which asks for the set elements to be read, but they cannot read or write other character data. (Set element names should be on the file as an array of character strings, each string being of (the same) length up to 12.)

4.6.1 **Text Files**

- Text files can be created using an editor and can be printed directly. They are especially useful for small arrays of data. GEMSIM and TABLO-generated programs can read real or integer (but not character) data from such files or write to them. In a TABLO Input file, the default file type is Header Array. Thus, whenever you want a particular file to be a text file, you must indicate this by using the 'TEXT' file qualifier when declaring the file, such as in the example below.

  ```
  FILE (TEXT) input1 ;
  FILE (TEXT, NEW) output1 ;
  ```

  Then you can read data from file 'input1' and write data to file 'output1'. (The qualifier 'NEW' is required to indicate a file to be written to.) Data on text files has no 4-character header associated with it, so the READ/WRITE instructions in the TABLO Input file do not mention a header either, as in the next two examples.*

  ```
  READ A1 FROM FILE input1 ;
  WRITE A2 TO FILE output1 ;
  ```

  Partial READs/WRITEs to/from text files are also allowed. Use the same syntax as for Header files except that 'HEADER "xxxx"' is omitted. For example,

  ```
  READ (all,i,COM) A3(i,"imp") FROM FILE input1 ;
  ```

  You can also read data from the terminal or write it to the terminal. The syntax is as shown below.

  ```
  READ A1 FROM TERMINAL ;
  WRITE A2 TO TERMINAL ;
  ```

---

* GEMSIM and TABLO-generated programs ignore any header in the "how much data" information (see Appendix C of GPD-1) at the start of each array.

- Details of the preparation of data for text files are given in Appendix C of GPD-1, with an example given in section 3.4 of GPD-1.

- When preparing text data for a partial read, such as of

  (all,i,COM)(all,j,IND) A1(i,"dom",j),

  the first line indicating how much data follows should refer to the part to be read, not the full array. In the example just above, if COM and IND have sizes 3 and 20 respectively, the first line should be

  3 20 row_order ;

  to indicate a 2-dimensional array of size 3 x 20. Then the data (3 rows each of 20 numbers) should follow, each row starting on a new line. Note also that long "rows" can be spread over more than one line. (For example, the 20 numbers here can be put say 8 on each of two lines and the 4 on a third line.)

  In preparing the "how much data" information at the start of each array, you can omit any sizes equal to 1.
  For example, when preparing data to be read (via a partial read) into the (3, 1-4, 5, 1-7) part of some coefficient, regard this as a 2-dimensional array of size 4 x 7 (not a 4-dimensional array of size 1 x 4 x 1 x 7). The only time a size equal to one need be used is for a single number, when the "how much data" line will be "1 ;" meaning a 1-dimensional array of size 1.

- Several different arrays of data can appear consecutively on a text file. GEMSIM or the TABLO-generated program reads them in order. Of course you must be careful to prepare the data file so that the order corresponds with the order or the READ statements in your TABLO Input file. Indeed, this question of order is one reason why using text files can introduce errors that would not occur if you were using Header Array files. We recommend that you never split a model whose data is read from text files into submodels using TABLO because of this.

- Two READs from the same TEXT file must not be separated by READs from other files. Otherwise GEMSIM or the TABLO-generated program would close the TEXT file when it comes to the read from the other file; then, when it re-opens the TEXT file for the next read from it, it would start reading at the beginning of the TEXT file again. (That is, it would lose its place in the TEXT file.)

- When reading data from the terminal, GEMSIM and TABLO-generated programs ask if you wish to input the data in row order (the default) or column order. You are then prompted for it one row or column at a time. [If you want to prepare a batch file for this, follow the instructions above as if preparing a text file except that the first line showing amount of data should be omitted.]

- Note that, at present, GEMSIM and TABLO-generated programs can only read real and integer data (but not character data) from text files. Set elements and subsets-by-number must be read from Header Array files.

4.7 **Reads, Writes and Displays**

4.7.1 **How Data is Associated With Coefficients**

- The order of the elements of a set (as declared in a SET declaration) determines the association of data on a Header Array file with actual parts of a coefficient. For example, if the set IND has elements "car", "wool" and "food" and the set FAC has elements "labor" and "capital", and if coefficients A3, A4, B3 and A5 are declared via

```
COEFFICIENT (ALL,i,IND)(ALL,f,FAC)  A3(i,f) ;
COEFFICIENT (ALL,i,IND)                A4(i)   ;
COEFFICIENT                               B3      ;
COEFFICIENT (ALL,i,IND)(ALL,j,IND)(ALL,f,FAC)  A5(i,j,f) ;
```

  then, for the purposes of associating them with data on a file, A3 should be thought of as a 3 x 2 matrix (where the rows are indexed by the set IND and the columns by the set FAC), A4 should be thought of as vector (that is, a one-dimensional array) of length 3, B3 as a single number and A5 as a 3 x 3 x 2 array. For

```
READ A3 FROM FILE cid HEADER "C003" ;
```

  to succeed, the data on the file at header ’C003’ must be a 3 x 2 matrix. A3("car","capital") gets the value of the entry in row 1 and column 2 of the matrix on the file. For

```
READ A4 FROM FILE cid HEADER "C103" ;
```

  to succeed, the data on the file at header ’C103’ must be exactly 3 numbers. A4("wool") gets the value of the 2nd number on the file. For

```
READ B3 FROM FILE cid HEADER "C113" ;
```

  to succeed, the data on the file at header ’C113’ must be a single number. B3 gets the value of this number on the file.

4.7.2 **Partial Reads, Writes and Displays**

- With A5 declared as above and IND having elements as above, for the partial read of the coefficient A5

```
READ (ALL,i,IND)(ALL,j,IND) A5(i,j,"capital")
  FROM FILE cid HEADER "C032" ;
```

  to succeed, the data on the file at header 'C032' must be a 3 x 3 matrix. A3("car","food","capital") gets the value of the entry in row 1 and column 3 of the matrix on the file.

- In checking that the amount of data at the appropriate place on the file is as expected, any 1's in the size are ignored. For example, if a 3 x 2 matrix of data is required, an array of size 3 x 1 x 2 on the file is considered to match this (in the obvious way).

- In READ, WRITE or DISPLAY statements which only transfer some of the values of the relevant coefficient (so that quantifiers occur explicitly),

  (a) all indices associated with the coefficient must be different. For example,

```
READ (all,i,COM) A6(i,i) FROM ......         !incorrect!
```

  is not allowed.

  (b) every index quantified must appear as an index of the coefficient. For example,

```
READ (all,i,COM)(all,j,COM) A7(i) FROM ....  !incorrect!
```

  is not allowed.

  (c) indices can range over subsets of the sets over which the coefficient is defined. For example, if coefficient A is defined by

```
COEFFICIENT (all,c,COM) A(c) ;
```

  and MARGCOM has been defined as a SUBSET of COM, the statement

```
READ (all,c,MARGCOM) A(c) ;                   !correct!
```

  is allowed.*

- Similar rules apply to partial writes and displays.

---

* This is a change from Release 5.0, where subsets were not allowed in this context.

4.7.3 **FORMULA(INITIAL)s**

- Each FORMULA(INITIAL) statement in a TABLO Input file produces an additional READ statement in the associated linearized TABLO Input file. For example, the statement

  FORMULA(INITIAL) (all,c,COM) A(c) = ... ;

  also gives rise to the statement

  READ (all,c,COM) A(c) FROM FILE ... ;

  A temporary file (the "intermediate extra data" file - see section 5.14 below) is used to hold the values of the coefficient in this case. The FORMULA is implemented during step 1 of a multi-step calculation while the READ is applied during subsequent steps.

  Because of this, the restrictions in section 4.7.2 above for partial reads apply also to FORMULA(INITIAL)s. That is, the quantifiers and LHS coefficient occurrence of a FORMULA(INITIAL) must satisfy the rules for partial READs. For example,

  FORMULA(INITIAL) (all,i,COM) A6(i,i) = ... ; !incorrect!

  would produce a semantic error.

4.7.4 **Coefficient Initialisation**

Part of the semantic check is to ensure that all coefficients have their values initialised (that is, assigned) via reads and/or formulas. You should note that the check done by TABLO is not complete, as explained below.

Consider, for example, a coefficient A6 declared via

```
COEFFICIENT (ALL,i,COM)(ALL,j,IND) A6(i,j) ;
```

TABLO can tell that a read such as

```
READ A6 FROM FILE cid HEADER "ABCD" ;
```

initialises all parts of A6 (that is, initialises A6(i,j) for all relevant i and j), as does a formula such as

```
FORMULA (ALL,i,COM)(ALL,j,IND) A6(i,j) = A4(i) + A5(j) ;
```

provided A4 and A5 have been fully initialised. TABLO can also tell that a partial read such as

```
READ (ALL,i,COM) A6(i,"sheep") FROM FILE cid HEADER "ABCD" ;
```

or a formula such as

```
FORMULA (ALL,i,COM) A6(i,"sheep") = A4(i) ;
```

only initialises some parts of A6 (since A6(i,j) for j different from "sheep" is not affected).

At present TABLO gives no warning about possibly uninitialised coefficients if two or more partial initialisations are made. You should note that, depending on the actual details, there may still be parts of the coefficient not initialised. If, for example, the set IND in the examples above has just two elements "sheep" and "cars" then the two reads

```
READ (ALL,i,COM) A6(i,"sheep") FROM FILE cid HEADER "ABC1" ;
READ (ALL,i,COM) A6(i,"cars")  FROM FILE cid HEADER "ABC2" ;
```

would fully initialise A6, but the two reads

```
READ (ALL,i,COM) A6(i,"sheep") FROM FILE cid HEADER "ABC1" ;
READ         A6("tin","cars")  FROM FILE cid HEADER "ABC2" ;
```

would leave some parts of A6 uninitialised (if COM has elements different from "tin").

4.7.5 **Display Files**

All DISPLAYs are written to a single text file called the **display file**. If you are running GEMSIM or the TABLO-generated program interactively, you will be prompted for the name of this file. If you are using a Command file, you specify the name of the display file in your Command file via a statement "display file = ... ;".

4.8 **Updates**

Update statements have been introduced in sections 3.5.2 and 3.5.3 of GPD-1. Here we give extra information about them.

4.8.1 **Purpose of Updates**

- The purpose of the update statements is to give instructions for calculating the new values for all data (that is, coefficients whose values are read) as a result of the changes in the variables of the model in one step of a multi-step simulation. In the expression on the **right-hand** side of an update statement, the values of any coefficients are those before the current step of the multi-step simulation and the values of any linear variables on the right-hand side are the changes or percentage changes as a result of the current step of the multi-step simulation. The values of the coefficient on the **left-hand side** are the new values (that is, those after the current step) of this coefficient.

- A coefficient is said to be updated if there is an UPDATE statement having this coefficient on the left-hand side of its update formula.

- Only coefficients of type NON_PARAMETER can be updated. COEFFICIENT(PARAMETER)s remain constant throughout the simulation and so cannot be updated. Levels variables do not need UPDATE statements since they are automatically updated by the associated percentage change or change variable - see section 2.2.2.

4.8.2 **Which Type of Update?**

The three kinds of UPDATE statements are shown in section 3.5.3 of GPD-1.

Note that a CHANGE UPDATE has the form

UPDATE (CHANGE) X = *<expression for change in X>* ;

The usual way of deriving the expression for the change in X is to use the Change differentiation rules in section A.1 of Appendix A.

Sometimes it may be convenient to think of the above slightly differently, namely as

UPDATE (CHANGE)
X = X*[*<expression for percent-change in X>*/100];

In either case the expressions used for the change or percent-change should be **linear in the linear VARIABLEs of the model**.

There is a further discussion of updates in section 6.1 below. Appendix C of this document and section 3.5.3 of GPD-1 contain examples of the derivation of update statements.

- The values of coefficients occurring on the RHS of an UPDATE are their values AFTER ALL formulas have been calculated.

- The values of linear variables on the RHS of an UPDATE are the values for the current step of the multi-step simulation (whether the variable is exogenous or endogenous).

4.8.3 **What If An Initial Value Is Zero ?**

In some cases you will specifically want to allow for the possibility that the updated value may be nonzero even if the initial value is zero. Examples are export subsidies or import duties which may change from zero before a simulation to nonzero after. In such cases **don't use** an update statement of the form

UPDATE (CHANGE) X = X*[*<expression for percent-change in X>*/100] ;

since the percent-change in X will be undefined if X is zero. Rather, use an update statement of the form

UPDATE (CHANGE) X = *<expression for change in X>* ;

4.8.4 **UPDATE Semantics**

- Only COEFFICIENTs whose values have been read or have been assigned (at step 1) via a FORMULA(INITIAL) can be updated. In fact only these coefficients need to be updated. These coefficients must be of type NON_PARAMETER.

- An updated coefficient must not appear on the left-hand side of any FORMULA(ALWAYS). (This is so that its values are not altered from those given as a result of statements in the current step of the multi-step simulation.)

- If a particular coefficient does not change its values as a result of any simulation (which means that this coefficient is effectively a parameter of the model), no update formula need be given for this coefficient. TABLO assumes that the values of a coefficient do not change if no update formula for it is given. Coefficients defined as PARAMETERS are not allowed to be updated.

- If only some of the values of a coefficient change as a result of a simulation but other parts of it cannot change, you only need to include an UPDATE statement to update the parts that change. TABLO infers that the other parts do not change.* For example,

  UPDATE (all,i,COM) X(i,"VIC") = p0(i) * x1(i,"VIC") ;

  will change the "VIC" part of coefficient *X* and leave all other parts unchanged.

- Integer coefficients cannot be updated.

- In a PRODUCT update statement, the right-hand side must be of the form

  v1 * v2 * ... * vn

  where each *vi* is a percentage change linear variable (not a change variable), and * is multiplication. A special case (see case 1. in section 3.5.3 of GPD-1) is where there is only one percentage change variable on the right-hand side, say 'v1'.

---

* This is different from Release 4.2.02 in which you also had to include an UPDATE statement saying that the unchanging parts did not change.

- If reads are made into two different coefficients from the same header of the same Header Array file, neither of these coefficients can be updated.

- A levels VARIABLE must not appear on the left-hand side of an UPDATE statement. (However these variables are automatically updated using their associated linear VARIABLE, as explained in section 2.2.2.)

4.9 **Zerodivides**

- Division by zero is never allowed in EQUATIONs or UPDATEs. ZERODIVIDE statements only affect calculations carried out in FORMULAs.

- Often, because the equations solved by TABLO are linearized, some of the coefficients are shares (or proportions) of various aggregates. Naturally, these shares should sum to unity. In some specific instances, however, the aggregate may be zero and the shares of this aggregate will amount to a zero proportion of a zero sum. So that simulations turn out as expected, it may nevertheless be important that the share values used *do* still add to one. Consider, for example,

```
FORMULA (ALL,i,COM)(ALL,j,IND) S(i,j) = A(i,j) / SUM(k,IND,A(i,k)) ;
```

Here S(i,j) is the share of A(i,j) in the sum across all industries k of A(i,k), and we would expect that, for all commodities i,

SUM(k,IND,S(i,k))

equals one. If, for commodity "boots", A("boots",j) is zero for all industries j, then each S("boots",j) would be calculated as zero divided by zero. TABLO allows zero divided by zero in formulas, and uses a default value of zero for their results. However, then the shares S("boots",j) would not add to one over all industries (indeed, SUM(j,IND,S("boots",j)) would be zero). This can be rectified using a ZERODIVIDE instruction which changes the default value that is used whenever a division of zero by zero is encountered during formula calculations. In the above example, by changing the default to 1/NIND (where NIND is the number of industries), the shares S("boots",j) can be made to sum, over all industries, to one. This can be done by the TABLO input shown below.

```
COEFFICIENT NIND #number of industries# ;
COEFFICIENT RECIP_NIND #reciprocal of the number of industries# ;
FORMULA NIND = 114 ;
FORMULA RECIP_NIND = 1/NIND ;
ZERODIVIDE DEFAULT     RECIP_NIND ;
FORMULA (ALL,i,COM)(ALL,j,IND) S(i,j) = A(i,j)/SUM(k,IND,A(i,k)) ;
```

- Two types of division by zero are distinguished, division of zero by zero and division of a nonzero number by zero. Different default values can be set for these two cases and one can be allowed while the other is not. The second of these two cases is controlled by statements beginning **ZERODIVIDE (NONZERO_BY_ZERO)** while the first of these is controlled by statements beginning **ZERODIVIDE (ZERO_BY_ZERO)** (where the qualifier **(ZERO_BY_ZERO)** can be omitted since it is the default).

- You should note that once one of these two zerodivide default values has been set to a particular value by a ZERODIVIDE instruction, that value will be used as the zerodivide default for all calculations involving formulas that appear *after* that ZERODIVIDE statement in the TABLO Input file. This default will remain in effect until the zerodivide default is changed by another ZERODIVIDE statement of the same type or until it is turned off by a statement of the form

  ZERODIVIDE [(<qualifier>)] **OFF ;**

  When either type of zero divide is turned off, if that type of division is encountered in a formula, GEMSIM or the TABLO-generated program will stop running with an error message which indicates where the division has occurred. (See section 5.12.2 below for details.)

- There is the convention that, at the start of each TABLO Input file, division of zero by zero is allowed and the default result is zero, while division of a nonzero number by zero is not allowed. This is as if there were the following two statements at the start of every TABLO Input file.

  ZERODIVIDE DEFAULT 0.0 ;
  ZERODIVIDE (NONZERO_BY_ZERO) OFF ;

- We recommend that you turn off ZERODIVIDE defaults in all places except those where your knowledge of the data leads you to expect division by zero. In this way, you will not get unintended results as a result of ZERODIVIDE default values operating.

- Note that, after any formula in which ZERODIVIDE default values have been used, GEMSIM and TABLO-generated programs usually reports during step 1 of a multi-step calculation the number of occurrences and the default value used. Separate reports are given for zero-divided-by-zero and nonzero-divided-by-zero. Round brackets () enclose the former, while angle brackets <> enclose the latter. When such division occurs, we suggest that you check the data and formulas to make sure you understand why it is happening and that the default value begin given is acceptable. (If you don’t want your TABLO-generated program to be able to report these, you can select option NRZ in the Code stage of TABLO, as explained in section 5.10.1 below.)

4.10 **Ordering**

4.10.1 **Ordering of the Input Statements**

There is no fixed order for the appearance of the different types of input statements on the TABLO Input file. The only requirement is that COEFFICIENTs, VARIABLEs, SETs, set elements, SUBSETs and FILEs be declared in their own input statement before they are used in other input statements. (For example, you must declare a coefficient in a COEFFICIENT statement before you use its name in an EQUATION statement.)

A suggested order of the various parts of your model is :

- Declare sets and subsets.
- Declare files.
- Declare variables.
- Declare coefficients that are read in or are used in several formulas and/or equations. If preparing for multi-step simulations, insert update statements for those coefficients that must be updated after their declarations.
- Follow with read instructions, formulas, equations, displays and declarations of less frequently used coefficients, keeping related coefficients close together.

4.10.2 **Ordering of Variables**

(This refers to the order that the *variables* appear in the Equations file which usually determines their order on a Print file showing results of a simulation.)

o The order of variables is determined by the order in which the variables are declared in the TABLO Input file. This is one reason why it is a good idea to declare all variables in a bunch. Give careful thought to this order. It determines the order in which the variables are presented in SAGEM if you choose to respond to prompts in order to specify the exogenous variables in a simulation, for example, and usually determines the order of the variables on GEMPIE Print files showing results of simulations. Of course, it is easy to change this order in the TABLO Input file by a fairly simple edit.

o Note that the order of the endogenous variables on a GEMPIE Print file can be changed by selecting the option

RPO Choose row print order

as explained in more detail in section 7.6.2 of GPD-1.

4.10.3 **Ordering of Components of Variables**

(This refers to the order, within a given variable, that the components of that variable appear in the tableau, which determines their order in the Equations file and on a GEMPIE Print file.)

- The order of indices in the variable declaration determines the component order, under the rule that the first index varies fastest, followed, in order, by each subsequent index.

  For example, if the set IND has elements "car" and "food" and the set FAC has elements "labor" and "capital" and variable x3 is declared via

  ```
  VARIABLE  (ALL,i,IND)(ALL,f,FAC)  x3(i,f)  ;
  ```

  then x3 has four components. The first is x3("car","labor"), the second is x3("food","labor") (because i varies faster than f), then x3("car","capital") and, finally, x3("food","capital").

  Other examples are given in section 6.1.3 of GPD-1.

- The order of the quantifiers in the declaration has no bearing on the order of the components. Thus, if x3 above had been declared via

  ```
  VARIABLE  (ALL,f,FAC)(ALL,i,IND)  x3(i,f)  ;
  ```

  the order of the components would be the same as described above.

- The order of the components of a variable is important because that order is used on the Print files giving the results of simulations. Accordingly, you should consider the order of the indices carefully before writing down the equations. (If you have declared x3 as above, you may need to do time-consuming editing of the TABLO Input file if you decide later to reverse the order of the arguments of x3 because, in different occurrences of x3, different indices may be used or some of the arguments may be element names.)

4.10.4 **Ordering of the Equation Blocks**

- The order of the equation blocks in the tableau is determined by the order in which they are declared on the TABLO Input file.

4.10.5 **Ordering of the Equations Within One Equation Block**

- Note that, unlike the order of components of a variable, the order of equations in a block is not very important. It does not affect the results of simulations. You only need to know it if you are checking the entries of the Equations file.

- The order of equations in a block is determined by the order of the quantifiers in the EQUATION statement. The index in the first quantifier varies fastest, followed, in order, by each subsequent index. For example, if the set IND has elements "car" and "food" and the set FAC has elements "labor" and "capital" and equation EX1 is defined via

```
EQUATION EX1 #Example equation#
         (ALL,i,IND)(ALL,f,FAC)  x3(i,f) - y(i) = 0 ;
```

then there are four EX1 equations, corresponding to the elements of the sets IND and FAC as follows :

	IND value	FAC value
First EX1 equation -	"car"	"labor"
Second EX1 equation -	"food"	"labor"
Third EX1 equation -	"car"	"capital"
Fourth EX1 equation -	"food"	"capital"

4.10.6 **Ordering of Reads, Formulas, Equations and Updates**

- There is a similarity between reads and formulas, in that both assign values to some or all of the parts of a coefficient. The order of reads and formulas can affect the final composition of the model significantly. Formulas are calculated and reads are performed in GEMSIM or the TABLO-generated code in the same order in which they were listed in the TABLO Input file. For example, suppose IND has elements "car", "wool" and "food" and that coefficient A6 is declared via

```
COEFFICIENT (ALL,i,IND) A6(i) ;
```

Then, after

```
FORMULA (ALL,i,IND) A6(i) = 3.0 ;
FORMULA          A6("car")   = 1.0 ;
```

A6("car") will equal 1.0, while after

```
FORMULA          A6("car")   = 1.0 ;
FORMULA (ALL,i,IND) A6(i) = 3.0 ;
```

A6("car") will equal 3.0.

- As indicated at the start of section 4.2 of GPD-1 and in Figure 4.2 there, the reads and formulas are all done before the equations are calculated. Thus, even though the formulas and reads may be intermingled with the equations and updates in the TABLO Input file, the values of coefficients appearing

  (i) anywhere in every equation, or

  (ii) on the right-hand side (only) of every update

  are the values these coefficients have been given BY THE END of the TABLO Input file - that is, AFTER ALL FORMULAs AND READs HAVE BEEN PROCESSED. For example, given the TABLO input

```
FORMULA (ALL,i,IND) A6(i) = 3.0 ;
EQUATION  EQ1 (ALL,i,IND) A6(i)*x3(i) + x4(i) = 0 ;
FORMULA          A6("car")   = 1.0 ;
```

  the value of A6("car") used in the equation is 1.0, not 3.0 as it would be if its value "before" the equation were used. Therefore, you should regard equations as always appearing AFTER all reads and formulas on the TABLO Input file, no matter where you actually placed them.** Similarly you should think of the updates as appearing after all the equations, reads and formulas.

- As an example of how you can use ordering to define complicated expressions, consider the following formula :

** The reason for this somewhat confusing point is that, when the same coefficient is used in two different equations, it is important that the values of the coefficient in each equation are identical. Otherwise the equations would be virtually impossible to understand. This is especially clear when you consider what happens when an equation is used to substitute out a variable. Consider, for example the two equations

A6(i)*x3(i) + x4(i) = 0 (1)

and

A7(i)*x4(i) + A6(i)*x5(i) = 0 (2)

We may wish to use (1) to eliminate variable x4 (replacing x4(i) by - A6(i)*x3(i)) so that equation (2) becomes

-A7(i)*A6(i)*x3(i) + A6(i)*x5(i) = 0 (3)

Imagine what a mess we would get into if the A6 in (1) has different values from the A6 in (2). We would have a very difficult time interpreting (3).

$$ETA(i1,i2) = \begin{cases} -1.0 + S(i1), & \text{if } i1 = i2, \\ S(i1), & \text{otherwise.} \end{cases}$$

At present, TABLO does not accept conditionals involving indices themselves (such as i1 = i2) in formulas. But, making use of the order in which formulas are treated, this can be handled, as shown below.

```
FORMULA (ALL,i1,COM)(ALL,i2,COM)  ETA(i1,i2) = S(i1) ;
FORMULA (ALL,i1,COM)              ETA(i1,i1) = -1.0 + S(i1) ;
```

The first formula gets the values of ETA(i1,i2) correct when i1 is different from i2, while the second puts in the correct values when i1 = i2. Note that the order of these formulas is vital: if they were reversed, the result would not be as required.

A different method of implementing conditional expressions such as this one is shown in section 8.2 below.

- The order of updates of any one coefficient can also affect the final updated values of that coefficient. For example, reversing the order of the following two updates would clearly change the result.

```
UPDATE (CHANGE) (all,i,COM)(all,s,STATES) X(i,s) = 0 ;
UPDATE (all,i,COM) X(i,"VIC") = p0(i) * x1(i,"VIC") ;
```

But the order of updating DIFFERENT coefficients has no effect on the final result since the values of coefficients on the right-hand side of update statements are always their values at the start of the current step (not their updated values).

4.11 **Model Parameters**

In GEMPACK, a **parameter** of a model is a coefficient whose values do not change between the steps of a multi-step calculation; such coefficients can (and often should) be declared as COEFFICIENT(PARAMETER)s.

Non-parameter coefficients are used to carry the current values of levels variables. Their values can change between the steps of a multi-step calculation.

A different use of "parameter" in GEMPACK refers to the parameter values in the source code of GEMPACK programs. See section 5.5 of GPD-1 and sections 2.4 and 5.13 of this document for further details of this usage.

4.12 **TABLO Input Files with No Equations**

TABLO Input files which contain no EQUATION statements can be used for data manipulation (see, for example, section 3.4.3 of GPD-1).

In such a TABLO Input file, there is no distinction between "parameter" (see section 4.11 above) and "non-parameter" since no multi-step calculation will be carried out. Similarly, there is no distinction between FORMULA(INITIAL) and FORMULA(ALWAYS) in this case.

We suggest that you begin such a TABLO Input file with the special statement

EQUATION (NONE) ;

This statement, which must be the first statement, tells TABLO that the TABLO Input file being processed contains no EQUATION statements. When this is the first statement,

- TABLO ignores any qualifiers (PARAMETER) or (NON_PARAMETER) in COEFFICIENT statements,
- TABLO ignores any qualifiers (ALWAYS) or (INITIAL) in FORMULA statements,
- there must be no EQUATION, VARIABLE or UPDATE statements (since VARIABLEs can only be used in EQUATIONs and UPDATEs are only relevant to multi-step calculations),
- any COEFFICIENTs declared are treated as PARAMETERs for the purpose of the semantic rules, and
- any FORMULAs are treated as FORMULA(ALWAYS)s for the purposes of the semantic rules. (For example, conditional qualifiers are allowed in all FORMULAs.)

The main reason for introducing this "EQUATION(NONE);" statement was to facilitate data manipulation files containing formulas involving integer coefficients and conditional qualifiers, such as

```
COEFFICIENT (INTEGER) (all,i1,S)(all,i2,S2) INT1(i1,i2) ;
FORMULA (all,i1,S1)(all,i2,S2: C1(i1,i2) NE 0)
                INT1(i1,i2) = ... ;
```

If an "EQUATION(NONE);" statement is not at the start of this file, the FORMULA above would be treated as a FORMULA(INITIAL) [see section 4.4.10 above] and so would lead to a semantic error since conditional qualifiers are not allowed in FORMULA(INITIAL)s [see section 4.4.5]. Having an "EQUATION(NONE);" statement at the start of the file saves you having to include an explicit (ALWAYS) qualifier in the FORMULA above and hence an explicit (NON_PARAMETER) qualifier in the COEFFICIENT statement above.

We apologise for the rather ugly form of the "EQUATION(NONE);" statement; our only excuse is that we didn’t want to introduce another keyword for this case.

CHAPTER 5

GEMSIM AND TABLO-GENERATED PROGRAMS

5.1 Introduction

An overview of GEMSIM and TABLO-generated programs and the actions they may be capable of carrying out is given in chapter 4 of GPD-1. You should be familiar with that chapter before reading this one. In this chapter we provide more detailed information about running GEMSIM and TABLO-generated programs.

In section 5.2 we discuss the procedure for calculating multi-step solutions in more detail. GEMSIM and TABLO-generated programs have a large number of options which affect the actions they carry out and how they carry out these actions; these options are described in detail in sections 5.3 to 5.9 below. The Code options in TABLO itself affect how TABLO-generated programs are written and how these programs and/or GEMSIM carry out their actions; these options are described in section 5.10 below. Section 5.11 says a little about compiling and linking TABLO-generated programs. Sections 5.12 and 5.13 contain information about the sorts of errors that you may encounter when running GEMSIM or TABLO-generated programs, and what you can do about them. Section 5.14 discusses final and intermediate updated data files.

5.2 More Details About Multi-step Calculations

An introduction to the procedure by which multi-step solutions are calculated is given in sections 2.5, 4.2, 7.4 and 7.5 of GPD-1 (with which you should be familiar before reading this section). This section discusses various aspects of multistep calculations. We begin by giving (in section 5.2.1) an intuitive idea of Gragg' method, and the midpoint method, and then discuss how long these calculations may take (sections 5.2.3, 5.2.4 and 5.2.5). Occasionally indications of possible numerical problems with the solution of a system of linear equations may be received; this is discussed in section 5.2.6 below. We conclude this section with discussion in sections 5.2.7 and 5.2.8 about possible convergence problems you may encounter.

5.2.1 Gragg's Method and the Midpoint Method

Gragg's method and the midpoint method are the same except that Gragg's method does one extra pass at the end, as explained later.

The graphical motivation for Euler’s method is shown in Figure 2.5.3 in section 2.5.3 of GPD-1. At step 1, Gragg and midpoint are identical to Euler’s method, following a tangent along the curve from the initial solution. At other steps, all three methods take their direction from the tangent at the current point. The difference is that Euler’s method follows this tangent from the current point while Gragg’s method and the midpoint method follow this direction but

start from the previous point.

The difference can be seen at step 2, as illustrated in Figure 5.2.1. Each method gets to point B after step 1. At step 2, Euler’s method follows line BC (parallel to the curve at point B) and reaches point C after step 2, while Gragg and midpoint follow line AD (also parallel to the tangent to the curve at point B) to reach point D after step 2. You can see that D is significantly closer to the curve than B, which is why, usually, Gragg and midpoint produce more accurate results than Euler. You can experiment with other curves to convince yourself that this is generally the case.

Figure 5.2.1 goes here

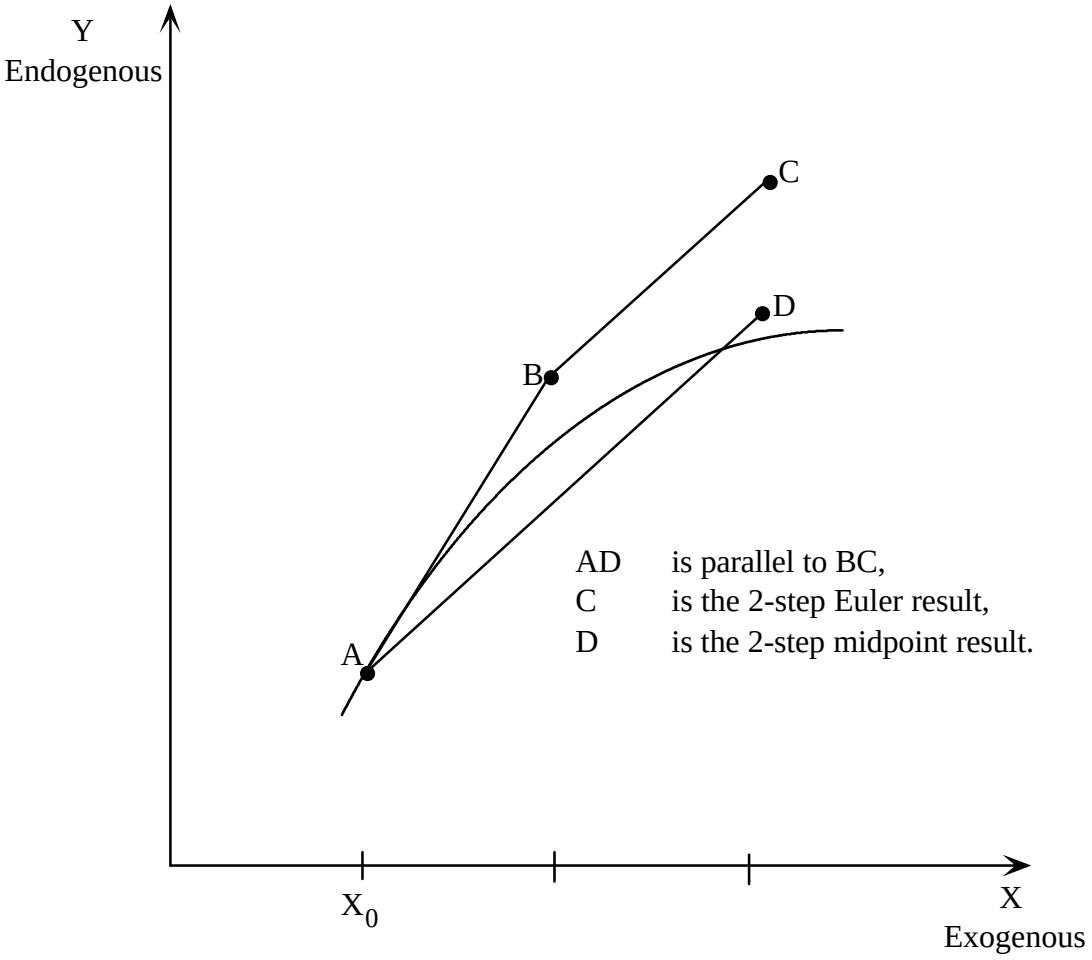

Figure 5.2.1: Midpoint Method Compared to Euler's Method

Gragg's method and the midpoint method are the same except that Gragg does one more pass. If you select N steps, midpoint does N passes while Gragg does N+1. In the final (N+1)st pass, the Gragg calculation actually starts near the final point and takes the exogenous variable past its end point. This is done to obtain a correction to the result after N passes; the corrected result is usually more accurate than the one after N passes. More details about the theory behind these methods can be found in Pearson (1991) and the references therein. (Gragg's method is sometimes called the **modified midpoint method**.)

Our experience is that, provided your simulation is not too nonlinear, Gragg or midpoint will converge significantly faster than Euler and that the extra accuracy Gragg usually gets from the extra pass it does compared to the midpoint method is usually well worth the extra time taken. However, we have found that in some highly nonlinear simulations, Gragg and midpoint diverge rapidly. If this happens, the Extrapolation Accuracy Summary will tell you clearly, and you should then try Euler's method (though that may not converge either in these cases). More information about this issue is given in sections 5.2.7 and 5.2.8 below.

Note that, if you are using Gragg's method or the midpoint method and are extrapolating from two or three separate calculations, all step numbers must be even or all must be odd. (Thus extrapolating from 2,3,4-step Gragg is not allowed.) We recommend 2,4,6-step as the simplest Gragg or midpoint. (A 1-step Gragg or midpoint is really Euler, so doesn't make much sense.)

Because Gragg's method takes the exogenous variables past their simulation end-point on the final pass, it may not be suitable for some simulations of models in which it is not sensible to increase the exogenous variables in this manner. In such a case we recommend the midpoint method.

Using any of the three methods, if a percentage change variable is shocked by more than -100 per cent, this may leave its levels value dangerously close to zero after some step in the simulation. In this case GEMSIM or the TABLO-generated program will tell you that you must change the shock or the number of steps.

5.2.2 **Reusing Pivots**

For large models, the time taken for the LU decomposition (see section 7.5 of GPD-1) of the matrix** A(K) is usually significant. (Indeed, the time taken for it may exceed the time taken for all other parts of the current step.) In this connection you should be aware that, in steps 2,3 and so on of a multi-step simulation, the Harwell sparse matrix routines can calculate the LU decomposition significantly quicker if they can make use of the position of the so-called **pivots** calculated when LU decomposing the corresponding matrix A(1) in step 1. This is called **reusing pivots**. If successful,

---

** This is the matrix called A(K) in section 4.2 of GPD-1. This matrix changes from step to step as the data is progressively updated.

the time taken for the LU decomposition in steps 2,3 etc is often less than half that for step 1.*

5.2.3 **Time Taken for the Different Steps**

You will notice that steps 2,3 etc take essentially the same time as each other. However the time taken for step 1 is usually different from the time for steps 2,3 etc, as explained below.

Steps 2,3 etc Are Usually Quicker Than Step 1

Usually the time taken for steps 2,3 etc is less than that for step 1 for four reasons.

- Reusing pivots (see above) usually makes a considerable reduction in the time taken.
- DISPLAYs and WRITEs are usually only done during step 1.
- READs are not usually necessary at steps 2,3 etc since the updated values are usually in memory.
- By step 2, the software knows the closure. It then does not need to calculate any submatrices corresponding to exogenous variables which are not shocked.

If Extrapolating, Step 1 is Very Quick for Solutions Two and Three

If you are doing two or three multi-step simulations and extrapolating from them, you will notice that relatively little time is taken for step 1 of the second and third solutions. There are two reasons.

- The formulas and equations do not need to be recalculated. This is because the starting point for step 1 is the initial data base. During step 1 of the first solution, GEMSIM or the TABLO-generated program has stored the equations based on this data on the Equations file and has stored on the Base Coefficient Values file the values of all coefficients (calculated from this base data) needed for backsolving and updating. Thus, in step 1 of the second and third solutions, GEMSIM or the TABLO-generated program does not need to calculate any equations or formulas; instead it reads the values it needs from the Equations and Base Coefficient Values files.
- The LU decomposition does not need to be recalculated, and

---

* The Harwell routine MA28A always does the LU decomposition at step 1. The Harwell routine MA28B is the one invoked at steps 2,3 etc; it attempts to use the pivot positions as found by MA28A. If it is unsuccessful (which can happen for a variety of reasons), MA28A is called to calculate the LU decomposition from scratch.

the solutions for the endogenous variables for this step have already been calculated during step one of the first multi-step calculation.

Thus the time taken for step 1 of the second and third solutions is essentially that for for doing the backsolves (if any) and the updates.

5.2.4 **Total Time for Multi-step Compared to 1-step**

If you have carried out a 1-step simulation including updates, you can form a rough estimate of the time that will be required to calculate one or more multi-step solutions.

The total time taken for a single N-step simulation should be no more than

- N times that for a 1-step if you are using Euler's method or the midpoint method;
- (N+1) times that for a 1-step if you are using Gragg's method. (This is because an N-step Gragg actually makes (N+1) passes.)

In fact the time for an N-step should be less than these estimates since steps 2,3 etc usually take less time than step 1 (as explained in section 5.2.3 above).

If you extrapolate on the basis of two or more multi-step solutions, you can pretty well ignore the time taken for step 1 of the second and third solutions (as explained in section 5.2.3 above). Also the time taken for the extrapolation (after the 2 or 3 solutions have been calculated) is usually negligible compared to the rest. Thus,

- the time taken for simulation plus extrapolation from N1 and N2-step solutions is roughly the time for an N1-step solution plus time for an (N2-1)-step solution.
- the time taken for simulation plus extrapolation from N1,N2 and N3-step solutions is roughly the sum of the times for separate N1, (N2-1) and (N3-1) step solutions.

Thus, for example, to a first order of approximation, extrapolating on the basis of 2,4,6-step Euler solutions is likely to take no more than 2+3+5=10 times as long as a Johansen simulation (including update) while for 2,4,6-step Gragg, the time is likely to be no more than for 2+4+6=12 times that for the Johansen.

You can get even better estimates on the basis of a single 2-step simulation. Suppose step 1 takes T1 seconds and step 2 takes T2 seconds. Then, since the times for each of steps 2,3 etc is essentially the same,

- the time taken for a single N-step Euler or midpoint solution will be approximately
  T1 + (N-1).T2 seconds.

- the time taken for a single N-step (but N+1 pass) Gragg solution will be approximately
  T1 + N.T2 seconds.

5.2.5 **Using Existing Files to Speed Up Step 1**

When you run GEMSIM or a TABLO-generated program, the Equations file produced contains all coefficients of the equations as evaluated from the starting data, while the Base Coefficient Values file contains the values of all coefficients needed for the backsolves and updates, also based on the starting data. If you run a second simulation with GEMSIM or the same TABLO-generated program and are starting from the same data, you can speed up the calculation of step 1 by telling the program to

use the existing Equations and Base Coefficient Values files.

This will reduce the time for step 1 since the formulas and equations don't need to be recalculated.

Of course, if you change the theory (that is, the TABLO Input file) or the data, the existing Equations and Base Coefficient Values files will still reflect the old theory and base data and so it would not be appropriate to reuse them.

If you have already carried out a simulation based on the same theory, data and closure, you can eliminate the need to compute the LU decomposition at step 1 by telling GEMSIM or the TABLO-generated program to

use the relevant LU file

provided you saved it on the previous run. This further speeds up step 1.

Note that

- Equations and Base Coefficient Values files depend on the theory (that is, the TABLO Input file) and the base data. You should only use existing Equations and Base Coefficient Values files which are based on the same theory and base data as you intend to use for the current simulation.

- LU files depend on the theory, base data and closure. You should only use an existing LU file if it is obtained using the same theory, base data and closure you intend to use for the current simulation.

The Equations file produced by GEMSIM or a TABLO-generated program is a suitable starting point for Johansen simulations via SAGEM. Also an LU file created by SAGEM can be used with GEMSIM or a TABLO-generated program (or vice versa) provided the theory, data and closure are the same.

5.2.6 **Warnings About How Well the Equations are Solved**

After solving a system of equations

$$Ay = b$$

(as in equation (3) of section 2.5.2 in GPD-1), GEMSIM or the TABLO-generated program usually calculates Ay (using the solution values for y) and compares it entry by entry with the RHS vector b. If these two appear to differ significantly, the program reports the equation numbers where these differences occur.

Also, when doing iterative refinement, the program warns if it appears to make a relatively large correction to one or more of the solution values. This can be an indication that the numerical solution is unreliable. (The LHS matrix is called **ill-conditioned** in this case.)

You should be aware that, in general, it is impossible to accurately distinguish numerically between a singular Equations Matrix (that is, one that is not invertible) and one that is nearly singular. (This is because of the limited precision of computers and the rounding errors that necessarily occur when doing large numerical computations.) In modelling, singular matrices are usually an indication that an invalid closure is being used, though occasionally they can be a consequence of zeros in the data base.

If GEMSIM or your TABLO-generated program gives either of the warnings described above (LHS and RHS differ, or possibly ill-conditioned matrix warning), you should check your closure and the alleged solution carefully since they may be unreliable (or even meaningless).

5.2.7 **Gragg and Midpoint Are Not Suitable in Some Cases**

Gragg’s method or the midpoint method converge much more quickly than Euler’s method for many simulations. (That is, they produce much more accurate results for the same number of steps.)

However these methods are known to be unsuitable for some simulations because the different multi-step simulations (for example, 6-step, 8-step and 10-step) may oscillate rather than converging monotonically as we would like. In such cases, it is known that these oscillations may become worse when the number of steps in increased. You can obtain information about oscillations from the Extrapolation Accuracy file and Summary.

It is virtually impossible to tell in advance which simulations will not converge with Gragg or the midpoint methods. Changing closure and especially shocks can and will change the behaviour. The practical moral seems to be the following.

> **If you see increasingly large oscillations in your results when using Gragg or the midpoint method, switch to Euler’s method.**

Euler’s method does not suffer from the problem of increasingly large

oscillations.*

A specific example in which there are increasingly large oscillations is given in section 5.2.8 below.

5.2.8 **Linearization Used Can Affect Accuracy**

We begin this subsection with an example, and conclude with some general comments.

Example

Consider the following levels TABLO Input file for the equation D=P*Q (Dollar value equals Price times Quantity).

```
VARIABLE (LEVELS) D  # Dollar value # ;
VARIABLE (LEVELS) P  # Price # ;  VARIABLE (LEVELS) Q  # Quantity # ;
FORMULA (INITIAL)  P = 1 ;    FORMULA (INITIAL)  Q = 1 ;
FORMULA & EQUATION  Value     D = P*Q ;
```

Example TABLO Input file

We suppose, for the purposes of this example, that Q is a stock whose levels value can be either positive or negative.

Consider the simulation in which P and Q are taken from their initial levels values of 1 and 1 to 1.1 and -0.18 respectively. This involves shocking the associated linearized variables p_P and p_Q by 10 and -118 per cent respectively. The new (post-simulation) levels value of D should be D = 1.1*(-0.18) = -0.198 so that the exact simulation result is p_D = -119.8 (that is, a 119.8 per cent decrease in D).

You can, of course, run TABLO and then GEMSIM to carry out this simulation using different step numbers and solution methods. You can also see the effect of two different linearizations by

(a) running TABLO with the default CHECK options, or
(b) running TABLO after selecting option ACD ("Always use Change Differentiation of levels equations") at the CHECK stage (see section 2.2.3).

For the equation D=P*Q, these produce two different linearizations, namely

(a) p_D = p_P + p_Q

(b) D*p_D = P*Q*[p_P + p_Q]

(as you can see by looking in the Information file, as described in section 2.2 above). The first of these comes from applying the Percentage Change rules for differentiating products (see section A.1

---

* Technically, Gragg's method and the midpoint method are only what is called "weakly stable". See, for example, section 6.4 of Atkinson (1989).

of Appendix A) while the second comes from applying the Change differentiation rule for products (also in section A.1 of Appendix A).

The numerical results for this simulation are surprisingly different for these two linearizations. For example, for 6,8,10-step Euler calculations, the results for p_D are

	6-step	8-step	10-step	extrap	
(a) Percent Change diff	-123.734	-119.226	-120.106	-150.510	OC? 2
(b) Change diff	-117.833	-118.325	-118.620	-119.799	XMA 6

As you can see, Change differentiation produces a very accurate extrapolated result, whereas Percentage Change differentiation produces poor results. If you experiment with different numbers of steps and/or different solution methods, you will see the same pattern. And, indeed, you will also see that the Percentage Change differentiation version is an example of the type of problem mentioned in section 5.2.7 above in which Gragg and/or midpoint produce increasingly large oscillations as the number of steps increases.

But, for this problem, the numerical results obtained from the Change Differentiation rule are increasingly better as the number of steps increases.

General Comments

Our experience to date with different models indicates that better numerical convergence is sometimes obtained when Change differentiation is used throughout. It is for this reason that we have made the **ACD** option (see section 2.2.3 above) available in TABLO.

If you are experiencing convergence problems with a model containing levels equations we recommend that you try selecting ACD to see if this helps. If your model is a linearized one, you will need to carry out the alternative linearizations by hand, which will be more difficult.

Note that it is only for some simulations that there is a significant difference between the different linearizations. With the D=P*Q example above, if you change the simulation to one in which Q is reduced only by 95 per cent (rather than 118 per cent) you will find that both the Percent Change and Change differentiations result in acceptable convergence. For example, the 6,8,10-step Euler results are shown below. (The exact result is p_D = -94.5.)

	6-step	8-step	10-step	extrap	
(a) Percent Change diff	-94.0610	-94.2100	-94.2860	-94.4887	CX 2
(b) Change diff	-92.9167	-93.3125	-93.5500	-94.4995	CX 5

We plan to research this question (how different linearizations affect results) more in the future.

5.3 **Options Available with GEMSIM and TABLO-generated Programs**

GEMSIM and TABLO-generated programs offer several different options, which can affect the way they run (for example, how they carry out a simulation).

The options screen for TABLO-generated programs is shown below.*

TABLO-GENERATED PROGRAM OPTIONS
(--> indicates those in effect)

```
BAT Run in batch            STI Take inputs from a Stored-input file
BPR Brief prompts           SIF Store inputs on a file
LOG Output to log file      ASI Add to incomplete Stored-input file
  NAX Name AXS/T files      CMF Take inputs from a Command file
EAA Echo all activity         SVX Select variables on XAC file (later)
CPU Report CPU times          RQF Required figures agreement (extrap)
NRP Don't reuse pivots      NIR Don't do solution iterative refinement
NSC Don't save BCV file -->IZ1 Ignore zero coefficients in step 1
NSE Don't save Eq file      KZ2 Keep zero coefficients in steps 2,3 etc
NEQ Do no equations         NWE Don't warn how well equations solved
NDS Do no displays          SSI Several subintervals; extrap after each
NWR Do no writes
NUD Do no final updates            Extra Options Screen
NSM Don't do simulation            --------------------
                            DTO Display, Terminal/Text write Options

Select an option   :  <opt>  Deselect an option      : -<opt>
Help for an option : ?<opt>  Help on all options     : ??
Redisplay options  : /       Finish option selection : Carriage return

Your selection >
```

Options Screen for TABLO-generated Programs

The options screen for GEMSIM is the same except that the option NAX, which is not relevant for GEMSIM, is not shown.

Note that, in both cases, option IZ1 is selected by default** (though you can, of course, turn it off).

We describe below in sections 5.4 to 5.9 how these different options operate. In section 5.4 we describe options affecting simulations, while options affecting Extrapolation Accuracy files are described in section 5.5. Section 5.6 tells you about options controlling which actions are carried out and section 5.7 describes options affecting what is reported to the terminal. In section 5.8 we indicate how you can split your simulation into several subintervals,

---

* The odd-looking indenting in this menu is an attempt to show the groups of related options. We would have preferred to have left blank lines to show these, but unfortunately this would have made the menu too long for most terminals.

** This is a change from Release 5.0. See section 5.4.1 for information about this option.

extrapolating after each one, while section 5.9 describes the remaining options.

Online Help screens can be displayed while running these programs by typing '**?**' followed by the abbreviation to the option on which help is needed. For example, to obtain help on Log files, type '**?LOG**'. To obtain help on all options, type '**??**' and scroll through the screens using a carriage-return. However most users will find that

the default options suit their needs.

For this reason, you may prefer to skip these sections 5.4 to 5.9 initially and only return to them later if and when you feel you need the more specialised information and advice given below. If so, you should probably also skip section 5.10 at this stage since it depends on a knowledge of the options for these programs.

Note that most of the options described in sections 5.4 to 5.9 below can be activated by appropriate statements in GEMPACK Command files; see Appendix A.1 of GPD-1 for the syntax of these statements.

5.4 **Options Affecting Simulations**

Several of the options available with GEMSIM and TABLO-generated programs affect the way simulation results are calculated. Some of these are aimed at possibly speeding up the calculations. We describe below how these different options operate.

5.4.1 **Ignoring/Keeping Zero Coefficients (Options IZ1 and KZ2)**

GEMPACK is able to solve large systems of linear equations efficiently by exploiting the sparsity of the LHS matrices of coefficients. The Harwell sparse matrix routines MA28A,B,C used by GEMPACK usually only need to be told about the nonzero entries in these matrices.

Consider, for example, the "Com_clear" equations for Stylized Johansen as shown in section 3.3.2 of GPD-1. The linearized version of these are

```
(all,i,SECT)  XCOM(i)/100.0*p_XCOM(i) - {XH(i)/100.0*p_XH(i)
                 + SUM(j,SECT, XC(i,j)/100.0*p_XC(i,j) ) } = 0
```

(see Appendix B). In the tableau for the Equations matrix C (as set out in Table 2.5.1 of GPD-1), notice that these two equations occupy rows 11 and 12 (see the SUMEQ map in section 8.1.1 of GPD-1). Since only variables p_XCOM, p_XH and p_XC occur in these equations, all entries in the other columns of these rows in the tableau are necessarily zero (irrespective of the data base values). The first of these two equations (the one referring to sector "s1") can only have nonzero entries in the columns corresponding to variable components

p_XCOM("s1"), p_XH("s1"), p_XC("s1","s1") and p_XC("s1","s2").

(These are columns 6, 10, 12 and 14 respectively, as shown in the

SUMEQ map in section 8.1.1 of GPD-1.) The entry in the column corresponding to p_XH("s1") is -XH("s1")/100.0 which is -2/100.0 for the usual data base (as shown in Table 2.2.1a of GPD-1). If however households consume none of commodity 1 (so that DVHOUS("s1"), and hence XH("s1)", are zero in the data base), this entry would be zero. This is what we mean by a **zero coefficient** - namely one that is zero because of the data base currently being used **but might possibly be nonzero given another data base**. (Contrast this to the coefficient of variable 'p_Y' in these equations which will always be zero, irrespective of the data base, since variable 'p_Y' does not appear in these equations.)

If you keep zero coefficients (in the sense just defined) during step 1 of a multi-step simulation, this maximizes the chance that reusing pivots (see section 5.2.2 above) will succeed in steps 2,3 etc. (Since the data base changes at these steps because of updating, a zero coefficient at step 1 may possibly become nonzero in later steps. Reuse of pivots fails if the LHS matrix at step 2 has an entry in a position for which no entry was recorded at step 1.) However keeping zero coefficients calculated at step 1 may increase significantly the time taken for step 1.

For many models, reusing pivots will succeed at all subsequent steps even if zero coefficients are not kept during step 1.

For some models and some simulations, reusing pivots fails unless zero coefficients are kept during step 1. In such cases, in a multi-step simulation with several steps, the extra time taken during step a may be more than offset by the speedup in later steps achieved by reusing pivots. (Though this would not be a consideration for a 1-step simulation and may not be for simulations with a small number of steps.)

To give you control over this, we have provided option

IZ1 Ignore zero coefficients in step 1

which tells GEMSIM and TABLO-generated programs to ignore zero coefficients at step 1. Because we have found that keeping zero coefficients in step 1 only speeds up simulations with many steps and only for some models, this option IZ1 is selected by default. If you wish to keep zero coefficients at step 1, you can turn off option IZ1 by entering '**-IZ1**' (or putting the statement "**IZ1 = no ;**" in your Command file).

At steps 2,3 etc, recording zero coefficients is usually counter-productive (the pivots the program attempts to reuse at step 3 are those from step 1, not from step 2) and so the default is not to record zero coefficients at steps 2,3 etc. However, in a few cases we have found that, paradoxically, the LU decomposition proceeds faster if zero coefficients are kept. Hence we have provided option

KZ2 Keep zero coefficients in steps 2,3 etc

which tells GEMSIM and TABLO-generated programs to record zero coefficients at steps 2,3 etc. If reuse of pivots is failing, you may like to experiment with option **KZ2** to see if it speeds up calculations; however it will usually slow things down, and will almost certainly not be an improvement if reusing pivots is

succeeding. The statement "**KZ2 = yes ;**" in a Command file has the same effect as selecting option KZ2.

5.4.2 **No Reuse of Pivots (Option NRP)**

As explained above in section 5.2.2, reusing pivots usually results in quicker execution. However occasionally it can reduce the numerical accuracy of solutions slightly (though this should not happen if iterative refinement is used). If you are in doubt about this, you can prohibit reuse of pivots by selecting option

NRP Don't reuse pivots

or putting the statement "**NRP = yes ;**" in your Command file. You may also prefer to select **NRP** if experience shows that reuse of pivots is failing with a particular model or closure; then selecting **NRP** will save the time spent attempting unsuccessfully to reuse pivots.

5.4.3 **No Iterative Refinement of Solutions (Option NIR)**

Iterative refinement usually results in more accurate numerical results. However it does take extra time (though usually only a fraction of that for the LU decomposition). The time taken can be considerable if memory limitations mean that the original LHS matrix is stored on a work file, as happens if option LIR (see section 5.10.2 below) is selected in the Code stage of TABLO.

If you feel that the time taken for iterative refinement is excessive, you can suppress it by selecting option

NIR Don't do solution iterative refinement

or putting the statement "**NIR = yes ;**" in your Command file.

5.4.4 **No Warnings About How Well the Equations are Solved (Option NWE)**

Selecting option

NWE Don't warn how well equations solved

(or putting the statement "**NWE = yes ;**" in your Command file) will suppress warnings about how well the equations are solved (namely warning if LHS differs from RHS or the matrix is possibly ill-conditioned, as described in section 5.2.6 above).

5.5 **Options Affecting Extrapolation Accuracy Summaries/Files**

If you are extrapolating from three multi-step simulations, an Extrapolation Accuracy Summary is always output to the terminal. This is the same as the summary produced at the end of the Extrapolation

Accuracy file if you request such a file. It gives information to help you decide if the solutions have been calculated sufficiently accurately for your purposes. The Extrapolation Accuracy file contains a brief annotation (for example, "EMA 6") about each solution. The Extrapolation Accuracy Summary summarises these.

Note that, if you are extrapolating from only 2 multi-step simulations, no annotations appear on the Extrapolation Accuracy file and no Extrapolation Accuracy Summary is given. This is because it is not possible to estimate the size of the errors on the basis of only 2 solutions.

5.5.1 **Required Figures of Accuracy (Option RQF)**

You can use the option

RQF Required figures agreement (extrap)

(or put a statement of the form "**RQF =** <integer> ;" in your Command file) to set the number of figures agreement you require. This is used in determining the two annotations

ERA Last two results equal to required accuracy
XRA Two extrapolations equal to required accuracy

in the Extrapolation Accuracy file and on the Extrapolation Accuracy Summary. If, for example, you select 3 figures then comment "ERA" will be given to any results whose last two multi-step solutions agree to three (or more) figures (unless they are equal to machine accuracy in which case the stronger comment "EMA" will apply).

Annotations "ERA" and "XRA" never appear unless you select option **RQF**.

5.5.2 **Selecting Variables on the Extrapolation Accuracy File**

By default, all endogenous variables plus any you elected to backsolve for appear on the Extrapolation Accuracy file (if you request such a file) and are summarised in the Extrapolation Accuracy Summary sent to the terminal. If you want to change the variables shown on the Extrapolation Accuracy file, you can do so by selecting option

SVX Select variables on XAC file (later)

if running interactively or from a Stored-input file, or by putting "XAC-retained ... ;" statements (see section A.1 of Appendix A of GPD-1) in your Command file. This enables you to select the variables to appear on the Extrapolation Accuracy file (these variables are referred to as the **XAC-retained variables**). You can select any exogenous or endogenous variables or any you elected to backsolve for to be on the Extrapolation Accuracy file. (The Extrapolation Accuracy Summary then summarises the XAC-retained variables.)

If you are running via a GEMPACK Command file, the variables selected are those in the **"XAC-retained ... ;"** statements in your Command file. (In this case option **SVX** need not be selected - indeed "SVX = yes ;" is not a legal statement in a Command file.)

If you are running interactively (or via a Stored-input file) and you select option **SVX** at the start, you choose the XAC-retained variables after the verbal description of the simulation (which makes it the last piece of user input).

5.6 **Options Affecting the Actions Carried Out**

These can be used to suppress actions that GEMSIM or the TABLO-generated program is capable of carrying out. The relevant options are

NSC	Don't save BCV file
NSE	Don't save Equations file
NEQ	Do no equations
NDS	Do no displays
NWR	Do no writes
NUD	Do no final updates
NSM	Don't do simulation

Each of these has a Command file equivalent. For example, "**NSC = yes ;**" has the same effect as selecting option NSC.

The effect of **NDS** or **NWR** is clear.

If you select **NSC**, a BCV (Base Coefficient Values) file will not be saved. However, you will need to specify a name for this file since this file is required during the calculation. (The file will be deleted at the end.)

If you select **NEQ** then no equations are calculated and so no simulation is carried out.

If you select **NSM** then the equations can be calculated (and the Equations file written) but no simulation is carried out. The Command file statement "**simulation = no ;**" is equivalent to selecting option NSM. (The statement "NSM = yes ;" is not allowed in Command files.) If you start from an existing Equations file and select option NSM, you can still set up and save a closure on an Environment file; see section 2.7.2 of GPD-1 for an example of this.

If you select **NUD** the **final** updated data files are not written (but intermediate ones are calculated since this is essential for calculating a multi-step solution).

The effect of **NSE** is described in the next subsection.

If you tell the program not to carry out some actions it is capable of (for example, by selecting option NWR when the TABLO Input file contains WRITEs), the program may not do all READs and FORMULAs in the TABLO Input file. It only does enough to calculate the values of all COEFFICIENTs required to carry out the actions requested. For example, consider the following simple TABLO Input file.

```
COEFFICIENT C1 ; C2 ; C3 ;    READ C1 FROM TERMINAL ;
FORMULA C2 = C1 + 1 ;         FORMULA C3 = C1 + 2 ;
DISPLAY C2 ;                  WRITE C3 TO TERMINAL ;
```

If you select option NDS, the FORMULA for C2 will not be calculated (since the value of C2 only needs to be calculated in order to display it). But if the FORMULA for C3 were changed to

```
FORMULA C3 = C2 + 1 ;
```

then the FORMULA for C2 would be executed since its result is required to calculate C3.

This principle is also applied to READs and FORMULAs at steps 2,3... of a multi-step calculation. If the values of some COEFFICIENTs are only needed to calculate entries of submatrices involving a variable which is exogenous and not shocked, the READs and/or FORMULAs for these COEFFICIENTs will not be carried out at steps 2,3... (but they will be at step 1 since the closure is not known at that time).

5.6.1 **Option NSE - Don't Save Equations File**

In order to save an Equations file, it is necessary to calculate all the separate entries in the Equations matrix (that is, the matrix C in equation (1) in section 2.5.1 of GPD-1). Once the matrix C has been calculated, when the closure is known, we can divide the matrix C into matrices A and D as in the equation

$$A.z1 = - D.z2$$

where z1 and z2 are respectively the vectors of endogenous and exogenous variables (see equation (2) in section 2.5.2 of GPD-1). When the shocks are known, it is easy to form up the matrix A and the vector b in the equation

$$A.z1 = b \qquad (*)$$

(this is equation (3) in section 2.5.1 of GPD-1) which must be solved at each step of a multi-step simulation.

If you do not want to save an Equations file, an alternative is to bypass the full C matrix and to head directly to the equation (*) above. This is what happens when you select option **NSE**. It is also what normally happens at steps 2,3... of a multi-step calculation.**

One advantage of heading straight for (*) is that memory is only required to store all the separate entries of A, which are the columns of C corresponding to endogenous variables. Especially if your model has a large number of exogenous variables, the memory for storing matrix A may be significantly less than the memory required to store all of C.

** TABLO-generated programs written using TABLO Code option SCS do not do this.

When you select option NSE, the calculations of the different submatrices distinguish between entries of C corresponding to exogenous and endogenous variables, and the number of nonzeros on the LHS (that is, in the matrix A) and the number contributing towards the vector b (that is, corresponding to shocked exogenous variables) are reported separately. In these calculations, entries of C corresponding to exogenous variables which are not shocked are ignored.

If you select option NSE, the model name, version and identifier are still required since they go on the Solution file (and may go on other files such as an Environment file), even though no Equations file is saved. You also need to supply a name for the BCV file instead of the name for the Equations file; if you are using a Command file, replace the statement specifying the Equations file name by the following:

```
BCV file = <file_name> ;
```

Option NSE cannot be selected if you are running a TABLO-generated program which was written using TABLO Code option SCS.

5.6.2 **Options Affecting How Writes and Displays are Carried Out**

There are several options affecting how writes to text files and displays are carried out. If you are running interactively or via a Stored-input file, you can see these options by first selecting option

DTO Display, Terminal/Text write Options

This reveals the following menu.

```
                DISPLAY/TERMINAL/TEXT FILE OPTIONS
                ( --> indicates those in effect )

      DWS Displays, terminal writes at all steps of a multi-step sim
      TWC Terminal writes in column order
      DPW Change page width of display file
      DPL Change page length of display file
      DDC Change number of figures after decimal point in displays
      DPN New page in display files only if needed
      D1C In display files, show 1-dimensional array as a column
      DOI In display files, omit "identical" row messages
      NEL No element labels on ROW_ORDER or COL_ORDER text files

Select an option   :  <opt>  Deselect an option      : -<opt>
Help for an option : ?<opt>  Help on all options     : ??
Redisplay options  : /       Return to other Options : Carriage return

 Your selection >
```

DTO Display, Terminal/Text Write Options Menu

Once you have selected any options desired from this, you return to the main options menu by entering a carriage-return.

Alternatively, if you are using a GEMPACK Command file, you can select these options directly - option **DTO** is not relevant. For example, "display width = 70 ;" sets the width of pages in the display file to 70.

Below we describe the options available.

Normally WRITEs and DISPLAYs are only done during step 1 of a multi-step simulation (when they reflect values on the original data files). Occasionally, perhaps because you are trying to identify a problem with your UPDATE statements, you may wish to see these values at all steps of the simulation. The option

DWS Do displays and terminal writes at all steps of a multi-step simulation

(or the statement "**DWS = yes ;**" in your Command file) causes all displays and writes to the terminal (but not to Header Array files or other text files) to be done at each step.

By default WRITEs to the terminal are done in row order. If you select the option

TWC Terminal writes in column order

(or put the statement "**TWC = yes ;**" in your Command file), all terminal writes will produce arrays written in column order (that is, col_order) as described in Appendix C of GPD-1.

The normal format for any DISPLAYs is to show real numbers with 6 decimal places and to put at most 58 lines per page and at most 132 or 80 (depending on the machine - PCs usually use 80) characters per line. The options

DPW Change page width of display file
DPL Change page length of display file
DDC Change number of figures after decimal point in displays

are designed to allow you to vary these. Page width can be varied between 70 and 200 characters and page length must be at least 20 lines. The number of figures after the decimal point can be between 0 and 10. The related Command file statements are

display width = <integer> ;
display length = <integer> ;
display decimals = <integer> ;

Normally in a display file, each new coefficient displayed starts on a new page. If you select the option

DPN New page in display files only if needed

(or put the statement "**DPN = yes ;**" in your Command file), new pages will be started only when the current one is close to the end.

Normally in a display file, one-dimensional arrays are displayed as a row (that is, across the page). If you select the option

D1C In display files, show 1-dim array as a column

(or put the statement "**D1C = yes ;**" in your Command file), such arrays will be displayed as a column (that is, down the page).

Normally in a display file, if one row contains the same values as the previous one, a message indicating this is shown rather than the actual values being repeated. This can apply to several rows in succession. If you select the option

DOI In display files, omit "identical" row messages

(or put the statement "**DOI = yes ;**" in your Command file), no "identical" row messages will be shown: instead all values will be shown.

Normally element name or number labels are shown (as comments) in row_order or col_order text files. If you select the option

NEL No element labels on ROW_ORDER or COL_ORDER text files

(or put the statement "**NEL = yes ;**" in your Command file), such labels will be suppressed. (See Appendix C of GPD-1 for more information about these text files.)

5.7 Options Affecting CPU and Activity Reporting

If you select option

CPU Report CPU times

(or put the statement "**CPU = yes ;**" in your Command file), GEMSIM or the TABLO-generated program will report CPU time (that is, processing time) for various parts of the code (for example, the time taken for all updates in each step). However, on some machines this may just report CPU times as zero, which means that CPU reporting is not available on this machine.

During a run of GEMSIM or a TABLO-generated program, "echoing activity" means indicating all reads, formulas, set, subsets, displays, writes, backsolves and updates as they are carried out, and reporting how many nonzeros are in each submatrix as it is calculated.

If you are carrying out a simulation, the default is to echo activity during the first step only. In this case, if you select option

EAA Echo all activity

(or put "eaa = yes ;" in your GEMPACK Command file), activity is echoed during all steps of a multi-step simulation.

If you are not carrying out a simulation, the default is to echo all activity. In this case, if you de-select EAA by responding "-eaa" (or put "eaa = no ;" in your Command file), activity is not echoed.

5.8 **Splitting a Simulation into Several Subintervals (Option SSI)**

Normally, if you want to obtain accurate solutions to the nonlinear equations of your model, you calculate 3 multi-step solutions and extrapolate on the basis of these. Each of these moves away from the exact solution curve as it goes along (though the one taking the most steps stays closer), but the final extrapolated result is hopefully almost on the curve again. This is illustrated for 1,2,4-step solutions in Figure 5.8a below, in which X on the horizontal axis is exogenous (moving from X0 to X1) and Y on the vertical axis is endogenous.

In this case, you can't tell how accurate your final solution is until you see the Extrapolation Accuracy Summary at the end. In a highly nonlinear case you may calculate 40,60,80-step solutions, perhaps taking several hours, only to find that the accuracy at the end is not what you require. Then you must start all over again, choosing more steps.

This is one of the situations in which we have found it useful to be able to split the simulation into several so-called **subintervals**. In each subinterval the exogenous variables are changed by the same (levels) amount, so that, after all is done, the total change is as required. The basic idea is to go across each subinterval with 3 multi-step calculations (usually with a small number of steps) and then to **extrapolate before starting the next subinterval**. The data base is extrapolated as well as the solution. Hopefully the solution comes back very close to the exact one at the end of each subinterval. This is illustrated in Figure 5.8b for two subintervals and 1,2,4-step calculations across each one.

For example, instead of calculating 40,60,80-step solutions, you could break the simulation into 10 subintervals and take 4,6,8-step calculations across each one. The total calculation time is approximately the same but one advantage is that you get an Extrapolation Accuracy Summary (see section 2.5.3 of GPD-1) for each subinterval. Thus, at the end of the first subinterval, you can see if the solution accuracy is what you require; if it is not, you can stop the simulation then and restart it with more subintervals or more steps across each subinterval (or both).

To do this, you can use the option

SSI Several subintervals; extrap after each

after which you will be asked for the number of subintervals. Alternatively you can use the

subintervals = <number> ;

statement in your GEMPACK Command file (see Appendix A.1 of GPD-1).

In either case, the number of steps you specify will be carried out **in each subinterval**. You should specify the shocks you intend for the whole interval; GEMSIM or the TABLO-generated program automatically breaks this up into the appropriate (smaller) shocks for each subinterval and then for each step in each subinterval.

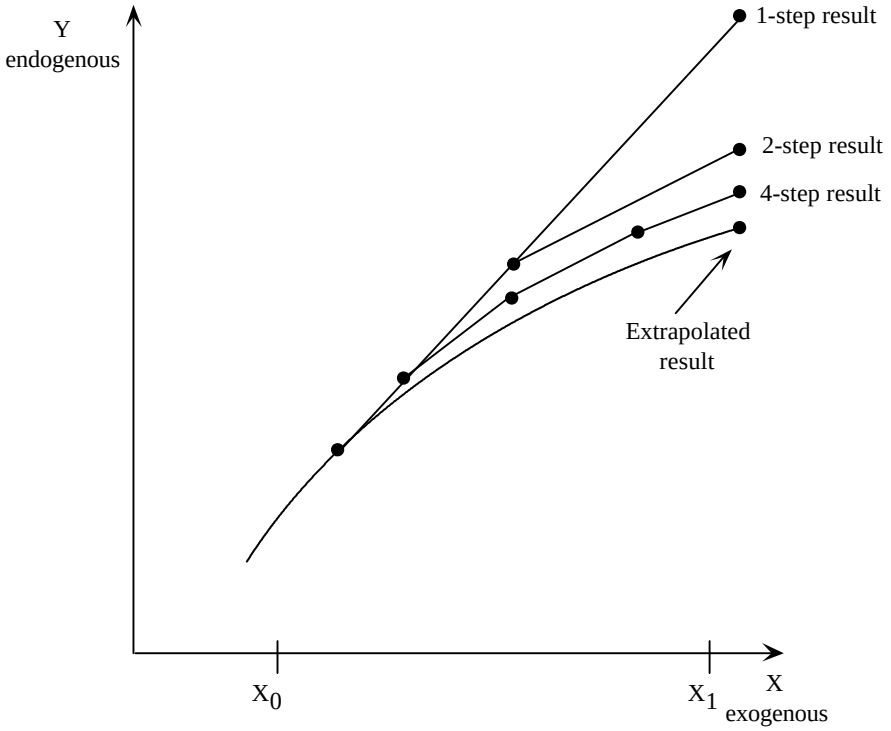

Figure 5.8a: One Subinterval

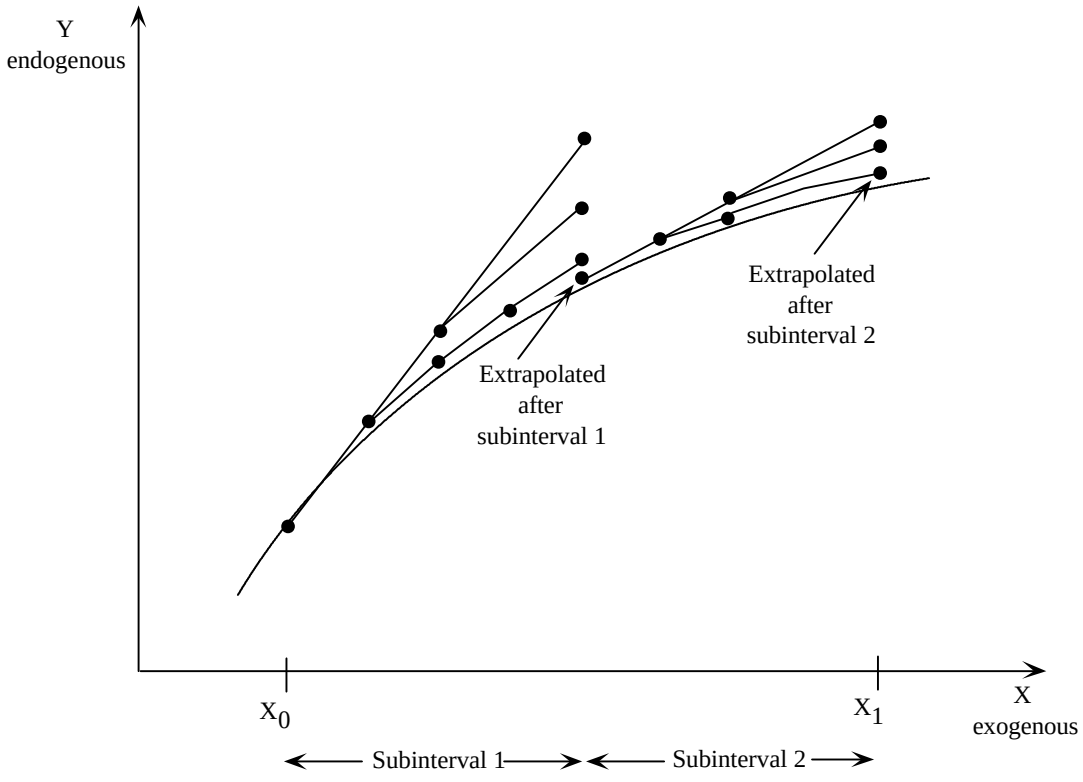

Figure 5.8b: Two Subintervals

In this case an Extrapolation Accuracy Summary is output to the terminal after each subinterval. If you are also creating an Extrapolation Accuracy file, the results for the variables are output to it for each subinterval; these results are the endogenous movements just across the current subinterval. If, for example, you have 2 subintervals and a percentage-change variable X increases by 5 per cent across the first subinterval and by 4 per cent across the second one, this means it will increase by the combination of these, namely 9.2 per cent, across the whole simulation; the Solution file will show 9.2 while the Extrapolation Accuracy file will show 5 and 4 for the two subintervals.

5.9 **Names of Auxiliary Files**

This is one of the few respects in which GEMSIM and TABLO-generated programs are different, as explained below.

5.9.1 **TABLO-generated Programs**

TABLO-generated programs must be able to access their associated Auxiliary Statement and Table files (usually with suffixes '.AXS' and '.AXT' respectively). The names of these are hard-wired in the source code of the program, and these names usually suffice. However, occasionally (perhaps when you are running the program from a file server or another directory), you may need to tell the program where to find these files. For this reason we have provided the option

NAX Name AXS,T files

If you select this option, you will be asked for the name (excluding suffix) of these files. You can include directory and path information in the name you enter. Note that the two Auxiliary files must have the same names except for their suffixes.

Note also that the statement

Auxiliary files = <file_name> ;

can be used in a Command file (see Appendix A.1 of GPD-1) to achieve the same effect.

5.9.2 **GEMSIM**

If you run GEMSIM interactively or via a Stored-input file, you give the name of the GEMSIM Auxiliary files (the GEMSIM Statement and Table files) after the first menu. If you are running from a GEMPACK Command file, you **must** give the name of these files via a statement of the form

Auxiliary files = <file_name> ;

(The relevant suffixes are added automatically by GEMSIM.)

If you want a Command file which can be used for either GEMSIM or the corresponding TABLO-generated program, you can include an Auxiliary files statement as above, provided that the <file_name> for the GEMSIM Auxiliary files for GEMSIM is the same as the name of the TABLO-generated program.

5.10 **Code Options When Running TABLO**

The Code stage of TABLO is when GEMSIM Auxiliary files or TABLO-generated programs are written. You can change these by selecting various options at this stage. (So now we switch from describing options for GEMSIM and TABLO-generated programs to options that apply for the **Code stage of TABLO itself**.)

TABLO CODE OPTIONS

CODE OPTIONS (--> indicates those in effect)

```
      NEQ Do no equations          PGS Prepare output for GEMSIM
      NDS Do no displays           WFP Write a Fortran Program
      NWR Do no writes                (i.e. a TABLO-generated program)

      ACC All comment lines in     NMS Don't allow multi-step
            code                         simulations
      OCS Omit code summary        CIN Code file name same as
                                         Information file name
        LOW MEMORY OPTIONS
      LMC .. for coefficients      CDM Change one or more default
      LIR .. for iterative               maximum values in the code
        refinement of solutions    NRZ No run-time reports re use of
      LRP .. no pivot reuse              ZERODIVIDE default values
      ECS .. eq,coeffs share
      SCS .. sim,coeffs share      SMD Code for Submatrix Data and
      UCS .. ud coeffs share             Set-up files
  --> PCS .. prev coeffs share     SPL Split code into submodels

 Select an option   :  <opt>  Deselect an option      : -<opt>
 Help for an option : ?<opt>  Help on all options     : ??
 Redisplay options  : /      Finish option selection : Carriage return
Your selection >
```

Options Menu for the Code Stage of TABLO

You have already seen the options

PGS Prepare output for GEMSIM
WFP Write a Fortran Program (i.e. a TABLO-generated program)

in section 2.2 and 2.6 of GPD-1. Which of these you select determines

whether TABLO writes output for GEMSIM (the GEMSIM Auxiliary Statement and Table files) or a TABLO-generated program.*

Of the other options, most affect only TABLO-generated programs, not GEMSIM output. The only ones affecting GEMSIM output are NEQ, NDS, NWR, NMS and DMS (see section 5.10.1 below).

5.10.1 **Code Options in TABLO Affecting the Possible Actions**

By default, GEMSIM or the TABLO-generated program written will be able to carry out all actions on the TABLO Input file. This means all WRITEs, DISPLAYs and EQUATIONs plus, if there are any UPDATEs, carrying out multi-step simulations. (Though, as we have seen in section 5.6 above, on any run of the program, you can limit those actions actually carried out.)

You can write output for GEMSIM or a TABLO-generated program which is capable of carrying out less than the full range of activities by selecting various code options when running TABLO.

If you select options

NDS Do no displays
NWR Do no writes

GEMSIM or the TABLO-generated program will not be capable of carrying out the displays and/or writes in the TABLO Input file. If you select

NEQ Do no equations

GEMSIM or the TABLO-generated program will not calculate the equations and so cannot carry out a simulation. (It will only be able to do any DISPLAYs or WRITEs.)

If you select option

NMS Don’t allow multi-step simulations

GEMSIM or the TABLO-generated program will be able to calculate the equations (and write the Equations file) but will not be capable of carrying out simulations. In the unlikely event that all coefficients

---

* One of these two options PGS, WFP will always be selected as the default. However this default varies between different copies of TABLO. (For example, in the Demonstration Version of GEMPACK, PGS is the default.) If you have a source-code version of GEMPACK, you (or your GEMPACK Manager) can change the default by altering the value of the code parameter DXGSIM in the TABLO Include file TBGSIM (see section 2.4 above) and recompiling and relinking TABLO. (The comments in Include file TBGSIM tell you how to alter DXGSIM.)

You can make your Stored-input files independent of this by always selecting the desired option explicitly, rather than relying on the default which happens to be set for the copy of TABLO you are currently running. (This does not hurt if the option you want is also the default.)

of the linear equations Cz=0 are parameters (that is, constants), you will have no UPDATE statements in your TABLO Input file but you will still need to do multi-step solutions to calculate accurate solutions to the underlying nonlinear equations of your model. In this case you are presented with option

DMS Do multi-step simulation

rather than **NMS**. If you select **DMS**, GEMSIM or the TABLO-generated program written will allow multi-step simulations.*

The other options discussed in this section affect only TABLO-generated programs; they have no effect on GEMSIM output.

Normally TABLO-generated programs report during step 1 the number of times any ZERODIVIDE or ZERODIVIDE (NONZERO_BY_ZERO) defaults have been used in each formula, as explained in section 4.9 above. If you select option

NRZ No run-time reports re use of ZERODIVIDE default values

the program cannot report these.

When writing a TABLO-generated program, TABLO has to make intelligent guesses as to how much memory to allocate for various parts of its processing. Sometimes the memory allocated initially needs to be increased (see section 5.13.2 below). If you know in advance that one or more parameters need to be increased (or decreased), you can do this via the option

CDM Change one or more default maximum values in the code

The relevant values are those of DEFMNZ, MMNZ, MMNZ1, DEFMSH, MMSHK and MMLIST in that order (see section 5.13.2 for more details about these).

Finally there are two options **SPL** and **SMD** which are provided only to maintain compatability with earlier releases of GEMPACK. They are not likely to be of use for new users of GEMPACK. If you select

SPL Split code into submodels

you can write smaller programs each capable of carrying out only some of the actions (EQUATIONs, WRITEs, DISPLAYs) on the TABLO Input file. In this case it is not possible to carry out multi-step simulations with the model. (This option is only likely to be relevant if you are

---

* You should not be tempted to use this option **DMS** in a case where UPDATE statements are necessary to get accurate solutions of the underlying nonlinear model but you have not yet got around to adding these UPDATE statements. For most (if not all) meaningful economic models, UPDATE statements will be required, and in these cases choosing **DMS** will not work. An example of a non-economic nonlinear equation for which **DMS** would work is that of the nonlinear equation Y=X^3 whose percentage-change linearisation is p_Y=3*p_X.

working on a PC with very limited memory.) If you select

SMD Code for Submatrix Data and Set-up files

the code will produce Submatrix and Set-up files rather than the Equations file (see section 8.5 of GPD-1).

5.10.2 **Code Options in TABLO Affecting the Amount of Memory Required**

This subsection applies only when you are writing a TABLO-generated program; the options here have no effect on GEMSIM output.

As with any complex computer program, TABLO-generated programs have to make a compromise between speed and memory requirements. The larger the amount of memory available, the faster the programs can run. By default, TABLO-generated programs are written for maximum speed. However it is possible to obtain exactly the same results using less memory than is required for these versions; usually the versions requiring less memory take longer to run.

At the end of its Code stage, TABLO tells you the memory required in bytes for the arrays in the program. The total memory required to run the TABLO-generated program will exceed this (since the code itself must also be in memory when the program is running); you may need to add one or two megabytes (or more for very large models) for this.

We provide options which tell TABLO to write code which uses less memory. These are the options

LMC Low memory for coefficients
LIR Low memory for iterative refinement of solutions
LRP No reuse of pivots
ECS Equations and coefficients share memory
SCS Simulation arrays and coefficients share memory
UCS Updated coefficients share memory between themselves
PCS Low memory for previous step coefficients

If you select one or more of these, data that would, by default, be in memory is held on one or more work files (on the disk). In general, the more of these options you select, the smaller will be the memory requirements of the program but the longer it will take to run. Except as indicated below, these options are independent of each other and can be selected in any combination.

Selection of appropriate options can be tricky and complicated. Most users will never need to make any of these choices. If you are not working on a PC or if you are building small models, we suggest you ignore the rest of this section. If you are working with large models and memory constraints become a problem, you should be aware of these options and should read the section below if and when it becomes necessary.

Note that selecting any of the options does not change the accuracy of the solutions calculated - it just affects the run time and memory required. You should also be aware that the low memory versions of these programs require more free disk space; the associated work files can be very large (several megabytes).

First we describe the different options. We conclude this section with advice on which to select if memory constraints are a problem.

Normally TABLO-generated programs have memory for storing the LHS matrix before doing the LU decomposition. The original LHS matrix is needed for doing iterative refinement. If you select option

LIR Low memory for iterative refinement of solutions

this LHS matrix will be stored on a work file and the associated memory (usually quite substantial) is not needed.

Normally TABLO-generated code attempts to reuse pivots from step 1 at steps 2,3 etc. This requires extra memory which will not be needed if you select option

LRP No reuse of pivots

The amount of memory released by selecting **LRP** is often substantial.

Note that selecting option **LRP** effectively means that option **LIR** (see above) is also selected. The memory saved by selecting **LRP** is usually about 1.5 times that saved by selecting **LIR** (but not **LRP**).*

Usually TABLO-generated code is allocated separate memory for the

(1) values of all coefficients (as held on the data base at the start of the current step),

(2) the Equations Matrix C in the system Cz=0 (that is, the result of calculating the equations), and

(3) the simulation data overheads (such as the shocks, the set of endogenous variables etc).

The option

ECS Equations and coefficients share memory

allows (1) and (2) to share memory by building up the equations calculation results (essentially the Equations file for the current step) on a work file (called the Equations Work file). Option

---

* The memory freed by **LIR** is that for MMNZ reals plus MMNZ integers, while for **LRP** it is that for MMNZ reals and 2.MMNZ integers. On many machines this is 8.MMNZ and 12.MMNZ bytes respectively. (Here MMNZ is the value of the code parameter of that name, as discussed in section 5.13.2 below. It is approximately the number of nonzeros in the Equations Matrix.)

SCS Simulation and coefficients share memory

allows (1) and (3) to share memory by the use of a Simulation Work file.

Usually TABLO-generated code is allocated memory to hold

(4) all updated coefficients in memory during the update stage, and

(5) all required base value coefficients in memory during the calculation of formulas, equations and backsolves.

Selecting option

UCS Updated coefficients share memory

means that work files are used for the updated coefficients (these are the so-called **intermediate updated files**), while selecting option

LMC Low memory for coefficients

means that the values of the base coefficients (that is, those in the data at the start of the current step) are held on a work file (the so-called Coefficient Work file) and only sufficient memory is allocated for those involved in any one formula, submatrix or backsolve.

It is difficult to predict how much memory is freed by selecting any of these options **ECS**, **SCS**, **UCS**, **LMC**. The Information file contains information about the first three of these; it tells how much memory would be required if you had selected any combination of them. To see the effect of **LMC** you would need to select it and then check the amount of memory required (as reported by TABLO).

Option

PCS Previous step coefficients share memory

only affects simulations carried out using Gragg's method or the midpoint method. At step number K, these methods need access to the values of the coefficients to be updated as they were at the start of the previous step (step number K-1). If **PCS** is selected (which is the default on most machines), these are held on a work file. When **PCS** is not selected, they are held in memory. TABLO reports on the Information file how much extra memory this requires.

Advice on Which of These to Select

If you are experiencing memory constraints, we suggest that you will need to experiment with the options. The options LIR, LRP and/or ECS are perhaps the first ones to try.

Since all these different options produce code which gives the same results, the main thing is to find a set of options which will let you run the program in the amount of memory available on your machine. If you are unsure that you have enough memory for the model,

perhaps select ALL low memory options and see if the resulting code will run. If that fails, select all except LMC and try again (since this may actually require less memory than if you select all including LMC). If that still fails, you will have to reduce the effective size of your model.

Remember that condensing the model (see section 3.9 of GPD-1) can help considerably in this regard.

Once you have found one set of options which allow you to solve your model, you can aim for others which still produce code requiring less memory than you have available, but which result in faster execution speeds. In this connection, note the following.

- Options **LIR**, **LRP** are pretty much independent of options **ECS**, **SCS**, **UCS** and **LMC**.
- Selecting **PCS** does not add much to the run time.
- Of the **ECS**, **SCS**, **UCS**, **LMC** group, be guided by what the Information file says about the memory required. (You will need to select **LMC** to get information about it in combination with the others.) Option **SCS** is unlikely to save much memory. Choose **LMC** as the last resort as it is likely to add most to the running time.
- Of the **LIR**, **LRP** pair, select **LIR** first and then **LRP** second.

Comparison with Release 4.2.02

Readers familiar with Release 4.2.02 of GEMPACK should note that, for a given model and condensation of it, the default options in the Code stage of TABLO produce TABLO-generated programs which require about twice (or more) as much memory as that produced by Release 4.2.02 TABLO. Selecting all of the Code options LRP, ECS, SCS and UCS will produce code requiring about as much memory as that produced by Release 4.2.02 TABLO. (But selecting option LMC as well may produce code requiring even less memory.)

5.10.3 **Other Code Options in TABLO**

This subsection applies only when you are writing a TABLO-generated program; the options here have no effect on GEMSIM output.

Option

ACC All comments lines in the code

leaves extra comment lines in the Fortran code (but otherwise make no difference). Option

OCS Omit code summary from the Information file

causes the summary of which COEFFICIENTs, FORMULAs etc are in each

submodel to be suppressed from the Information file. (This summary only occurs if you have selected option **SPL** to split into submodels.) Option

CIN Code file name same as Information file name

ensures that the name of the TABLO-generated program created is the same as that of the Information file (except for its suffix); then you are not asked for the name of the program.

5.11 **Compiling, Linking and Running TABLO-generated Programs**

"Compiling and linking" a TABLO-generated program refers to what has been called Step 1(b) in sections 2.6.1 and 2.6.3 of GPD-1. As stated in section 2.6.3 of GPD-1, the procedure for compiling and linking TABLO-generated programs varies from machine to machine, as does the syntax of the command for running the program. Consult your system-specific documentation or your GEMPACK Manager if you need more details.

When you run a TABLO-generated program to carry out a multi-step simulation, quite a lot of input is required. This can be given interactively or by preparing a Stored-input file. However we think that giving this input via a Command file is the best way; see sections 2.2.3 and 2.3.1 of GPD-1 and section A.1 of Appendix A of GPD-1 for details about Command files for running TABLO-generated programs.

5.12 **Possible Errors While Running GEMSIM or TABLO-generated Programs**

Most of the error messages reported by GEMSIM or TABLO-generated programs will be self-explanatory. Below we give information about three situations in which you may need more details in order to rectify the problem. These are when the simulation fails

- because of a singular matrix,
- because of a divide by zero error, or
- because you must increase the value of one of the PARAMETERs in the source code of the program.

The first two are discussed in sections 5.12.1 and 5.12.2 respectively. The third is discussed in section 5.13.

5.12.1 **Simulation Fails Because of a Singular LHS Matrix**

Your simulation may fail because the left-hand side matrix (that is, the matrix A in the equation* A.y=b) is singular (that is, not invertible) at some step. A singular matrix is either an indication that the closure you are using is not valid (even though it has the required number of exogenous and endogenous variables), or of zeros in your data base, or both.

The Harwell sparse matrix routines distinguish between **structurally singular** and **numerically singular** matrices, and an understanding of the difference may help you sort out the problem.

To see the difference between these, consider the (extremely uninteresting) model with just one linearized EQUATION

```
(all,i,COM)   D(i)*s(i) = t(i) ;
```

where COM is the set (c1, c2) and the two values of COEFFICIENT D are read from the data base. Consider the closure in which both components of variable t are exogenous and both components of variable s are endogenous. In this case the left-hand side matrix A will be

```
(  D("c1")      0    )
(     0     D("c2") )
```

Clearly A is invertible if and only if D("c1") and D("c2") are both nonzero.

Suppose now that D("c1") is zero and D("c2") is nonzero. Then the matrix A will be singular. Whether it is reported to be structurally or numerically singular depends on whether you are "keeping zero coefficients" in the Equations file, in the sense explained in section 5.4.1 above. If you are keeping zero coefficients, A will have two entries (that is, possibly nonzero entries), so that A has the shape

```
( *  0 )
( 0  * )
```

and A will be reported as being **numerically singular**. This means it is singular but

might be nonsingular for different values of the entries (those marked * above).

If, on the other hand, you are not keeping zero coefficients so that A has the shape

```
( 0  0 )
( 0  * )
```

---

* At each step of a multi-step simulation, a system of linear equations Ay = b must be solved, as explained in deriving equation (3) in section 2.5.2 of GPD-1. The matrix A and vector b change from step to step as the data is progressively updated, as explained in section 4.2 of GPD-1.

then A will be reported to be **structurally singular**. This means that

> **there are no values of the entries (marked * above) which could make the matrix invertible**.

Thus, if you find that the LHS matrix is structurally singular, and if you are keeping zero coefficients in the current step, this means that the matrix will never be invertible, whatever values you have on the data base, which suggests that your closure is invalid. If, on the other hand, it is numerically singular and you are keeping zero coefficients in the current step, this means that it might become invertible if you changed your data.

This is another reason for us providing options **IZ1** and **KZ2** (see section 5.4.1 above). If the LHS matrix is reported as being structurally singular at step 1 and you are using the default option IZ1, you may want to run the program again, this time deselecting option **IZ1**. If the matrix is then reported as being numerically singular, this indicates that the closure might be valid and may be failing because of the particular values in your data base. Similarly, it may occasionally help to select option **KZ2** if the singular matrix is at step 2 or more.

5.12.2 **Division by Zero Errors**

Division by zero is only allowed at certain times, namely when calculating FORMULAs provided the ZERODIVIDE defaults are set appropriately (as described in section 4.9).

GEMSIM or your TABLO-generated program may stop, reporting that division by zero was attempted when this was not allowed. If this happens during step 1 of a multi-step simulation, you will know in which FORMULA, EQUATION (indeed submatrix), backsolve or UPDATE this occurred since the program reports this just **before** it begins the associated calculation. However, if this happens at step 2 or later, you will probably not know where this happened; in this case rerun the program, this time selecting option **EAA** ("Echo all activity"), so that you can find where the error occurs.

Once you have found where the problem occurs, you can probably tell which division caused the problem. You may need to examine the values of the COEFFICIENTs in the denominator (perhaps by inserting extra DISPLAY or WRITE statements just before the place where the error occurs). This should help work out why the error occurred. Fixing the problem may involve changing the formula etc in question to allow for zeros in the data, or involve adding appropriate ZERODIVIDE statements to the TABLO Input file or it may involve changing the base data and/or closure. In sorting this out, recall that division by zero may be allowed in FORMULAs on your original TABLO Input file, but is never allowed in EQUATIONs, backsolves, UPDATEs or in FORMULAs put in by the system during condensation (see below) to simplify the subsequent arithmetic.

If you have condensed the model, identifying where the division by zero happens may be somewhat harder than described above since

- the equations, backsolves or updates being calculated may be different from the corresponding expressions in the original TABLO Input file because of substitutions made in them, or

- there may be extra coefficients, formulas and backsolves introduced by TABLO during condensation to simplify the subsequent arithmetic (see section 2.3.2 above).

In the first case above, you can see the current (that is, the post-condensation) form of the relevant equation or backsolve when running TABLO by selecting option 's' ("See a summary of the model") after Condensation and before the Code stage; after selecting 's', then select '3' or '4' to see the relevant equation or update. (Backsolves are just equations rewritten.) If the division occurs during a formula introduced by TABLO to simplify the arithmetic, you can see the formula for the relevant coefficient by looking in the Information file (the part that reports condensation actions).

5.13 **Increasing Parameter Values in the Source Code**

GEMSIM or a TABLO-generated program may stop with an error message saying that the value of a certain parameter in the code must be increased. In this section we list the different relevant parameters, explain their significance and tell you how to fix the problem.*

The information in this section is rather specialised and somewhat complicated. Most users should just register that this information is available here if and when it is needed, and then skip to the next section.

We discuss GEMSIM in section 5.13.1 and TABLO-generated programs in section 5.13.2.

5.13.1 **Increasing Parameter Values in GEMSIM**

There are many different program parameters whose values you may need to increase. These affect the maximum size model that can be handled by GEMSIM. (For example, MMNCS sets an upper bound on the number of COEFFICIENT declarations that can be handled currently. If your model has more, you will need to increase the value of MMNCS.)**

---

* Note that, in this section, we are using the word "parameter" in a technical sense relating to Fortran programs; there a "parameter" is just the symbolic name of a constant in the program. We explain how to identify the current value of these parameters later in this section.

** If you only have executable images of the GEMPACK programs, you will not be able to reconfigure GEMSIM. Either you must reduce the size of the relevant part of your model, or else upgrade to a source-code licence for GEMPACK.

All of the relevant parameter values have their values set in the two INCLUDE files GSINC and GSINC2 associated with GEMSIM.

Note that, on some machines (for example, Unix machines or VAX/VMS), you may not be able to make these changes yourself, but may have to ask your GEMPACK Manager to make them for you.

To help you find which INCLUDE file to edit, we list below the parameters whose values you are most likely to have to increase, showing which file they are defined in.

FILE	**Relevant Parameters in it**
GSINC	DCSSTN, DEQSTN, DICMEM, DPCMEM, DRCMEM, DSMRES, DSSMP, DUCMEM, DVCSTN, MCO, MMGSIN, MMGSIS, MMGSLS, MMGSOP, MMGSRC, MMGSRS, MMNCS, MMNEQ, MMNSS, MMNST, MMNVC, MMNZ, MMNZ1
GSINC2	DBVCSN, DCSRIG, DCSMVC, DCVCSN, DIUX, DRCOW, DRUX, DWA, MBNCOL, MMCM, MMCMCH, MMCNDR, MMCOMP, MMEL, MMLIST, MMNCOL, MMNFL, MMNRD, MMNROW, MMSHK

(Don’t change the value of any of these parameters unless you receive a message from GEMSIM suggesting you do so.)

If necessary, consult your machine-specific documentation to find where the source code for the INCLUDE files is stored on disk (and what their suffixes are).

Once you have changed the appropriate parameter value (by editing the INCLUDE file in which its value is set), you must then recompile and relink GEMSIM. Again, consult your machine-specific documentation to see how to do this.

5.13.2 **Increasing Parameter Values in TABLO-generated Programs**

There are six parameters whose values you may need to increase, namely MMCM, MMCMCH, MMNZ, MMNZ1, MMSHK and MMLIST. If a TABLO-generated program tells you to increase the value of another parameter, you should not do so, but rather treat it as a possible bug. Please report the circumstances to us at the Impact Project (following the procedure for internal program errors at the end of section 5.6 of GPD-1) so we can advise you.

If you need to increase **MMCM** or **MMCMCH**, follow the procedure described in section 5.5 of GPD-1. That is, edit the source code of the TABLO-generated program to change the value (the values of MMCM and MMCMCH are set very near the start of the file), and then recompile and relink the program (as described in Step 1(b) in section 2.6.3 of GPD-1). [If you have difficulty editing the TABLO-generated program, you can change the values of these parameters as follows. First edit the TABLO Include file TBCDDEC (see section 2.4 above) to change the values of the program parameters DFMCM and DFMCMC; the values of MMCM and MMCMCH are taken directly from these. Then recompile and relink TABLO, as described in section 2.4 above, and finally re-run TABLO.]

You should never have to increase parameters other than MMCM and MMCMCH in a TABLO-generated program which does data manipulation only (that is, one with no EQUATIONs).

We discuss increasing the other four parameters MMNZ, MMNZ1, MMSHK and MMLIST below. We begin with TABLO-generated programs capable of carrying out multi-step simulations in most detail. Other TABLO-generated programs are treated at the end of this section.

TABLO-generated Programs Able to Carry Out Multi-step Simulations

There are four program parameters whose values you may need to increase. These allocate the amount of memory available to the program for some part of its processing. The parameters are:

MMNZ which determines the amount of memory available for storing the condensed Equations Matrix C of the linear equations Cz=0. For the program to succeed, MMNZ must be at least as large as the number of nonzeros in C at each step, and ideally should be somewhat larger to give the Harwell routine MA28A a little "elbow room" for sorting the entries of C.

MMNZ1 which determines the amount of memory available for LU decomposing the LHS matrix A (in the equations Ay=b). For a simulation to succeed, MMNZ1 must be larger than the number of nonzeros in the LU decomposition of A. (MMNZ1 must never be less than MMNZ.)

MMSHK which determines the amount of memory available for storing the numerical shocks. For a simulation to succeed, MMSHK must be at least as large as the number of components shocked.

MMLIST which determines the amount of memory available for storing the program’s representation of the various sets of variables (such as the sets of endogenous, or shocked, or cumulatively-retained endogenous variables). The precise role of MMLIST is a little difficult to explain or understand. For a simulation to succeed, for each of these sets, MMLIST must be as large as the sum of the numbers of components of variables which have only some of their components (not zero or all) in the set.

TABLO attempts to make intelligent decisions about the values of these and could usually set values that would guarantee success in all cases. However, to do so would often take considerably more memory than would be needed in most cases. Accordingly, the approach settled on by the designers of the code is to allow user intervention (as described below) to set or influence the values of these. In this way you can intervene to ensure that appropriate values are chosen. This usually shouldn’t be necessary but may occasionally be required, especially if you are having difficulty carrying out a given simulation in the available memory.

You can influence or set these values in two ways.

(1) By selecting option **CDM** at the start of the Code stage of TABLO.

(2) By editing (via a text editor) the source code of the TABLO-generated program (after TABLO has written it). [The source of a TABLO-generated program is usually in a file with suffix '.for' or '.f'.] After editing, you will need to compile and link the program again (as described in Step 1(b) in section 2.6.3 of GPD-1).

Note that, because TABLO-generated programs for large models are very large files (they can be tens of thousand of lines long), you may not have access to a text editor capable of doing (2). But (1) should always be an option.*

If a run of a TABLO-generated program has failed because the value of one of these parameters is too small, the TABLO-generated program will usually give you information as to how large it needs to be. However, if the program tells you to increase a parameter by one, you should always increase it by more than this. Then either re-run TABLO and alter the value via Code option **CDM** (see below) or edit the file directly. In the latter case, look for a line in the source code containing the relevant value. For example, to change the value of MMNZ, search for the line containing "MMNZ=". (The value will follow the equals sign and the line should begin with the word "PARAMETER" as in

```
PARAMETER(MMNZ=12345,MMNZ1=23456)
```

which sets the values of MMNZ and MMNZ1.)

TABLO determines the values of these parameters as follows.

MMNZ TABLO is able to calculate an upper bound MAXTNZ for the number of nonzeros in the Equations Matrix C. This is usually close to the actual number of nonzeros in C if you are keeping zero coefficients (in the sense described in section 5.4.1 above), but may be a significant overestimate if you are not keeping zero coefficients.

After selecting Code option **CDM** of TABLO, you can either set the value of MMNZ directly or else you can change the value of a variable called DEFMNZ which TABLO uses to set the value of MMNZ. If the value of MMNZ is not directly set by the user, TABLO sets MMNZ equal to the minimum of 1.2 times MAXTNZ and DEFMNZ. (The extra 0.2 times MAXTNZ is for the "elbow room" referred to above.) TABLO starts with a default value of DEFMNZ. You can change the value of DEFMNZ via Code option **CDM**. Note that DEFMNZ may be set equal to zero; if so, DEFMNZ is treated as if it were

---

* Occasionally you may wish to **decrease** the value of one of the four program parameters because you need to reduce the amount of memory the program requires. If so, you should not edit the TABLO-generated program since this may not reduce the memory required (because of the way the program is written). You should always make the reduction via option CDM.

infinity so, unless you set MMNZ directly, MMNZ will be set equal to 1.2 times MAXTNZ.

MMNZ1 Under Code option **CDM**, you can select the value for MMNZ1. If you don't do so then TABLO sets MMNZ1 equal to 1.3 times MMNZ.

MMSHK This is set equal to the value of a variable called DEFMSH by TABLO. TABLO starts with a default value for DEFMSH. You can change the value of DEFMSH (and hence of MMSHK) via the **CDM** option. (However MMSHK never needs to exceed the total number of exogenous variables in your condensed system, and TABLO never sets MMSHK higher than this, even if you put DEFMSH above this value.)

Note that DEFMSH may be zero; if so, DEFMSH is treated as if it were infinity and then MMSHK is set equal to TABLO's best estimate of the number of exogenous variables. (TABLO may need to use a larger value than the actual number of exogenous variables if some of the set sizes are not known precisely until run time.)

MMLIST This is set equal to the value of a variable called DEFMLS by TABLO. TABLO starts with a default value for DEFMLS. You can change the value of DEFMLS (and hence of MMLIST) via the **CDM** option. Note that DEFMLS must never be set to zero.

The **default** values of DEFMNZ, DEFMSH and DEFMLS (see above) are stored on the TABLO Include file TBCDDEC. If you find that you frequently need to increase values via the **CDM** option, you may like to change the value of one or more of these defaults by editing this Include file and then rebuilding the executable image of TABLO (following the procedure described in section 2.4).

Other TABLO-generated Programs

For a program which creates the Equations file for your model (but cannot carry out multi-step simulations), only the value of MMNZ is relevant. The procedure for setting and/or influencing it is essentially as above for MMNZ except that, since no "elbow room" is required, 1.2 times MAXTNZ is replaced by just MAXTNZ.

For a program which creates one or more Submatrix Data files, the value of MMNZ must be as large as the number of nonzeros in any one submatrix. The procedure for setting and/or influencing its value is essentially as described above for MMNZ except that DEFMNZ is replaced by DEFMMS.

5.14 Final and Intermediate Updated Data Files

A file holding updated values of data read from the terminal is produced in a multi-step simulation whenever data is read from the terminal. So you should include a statement

updated terminal data = <file_name> ;

in your Command file precisely in this case. The file produced will be a text file.

When carrying out a multi-step calculation, GEMSIM and TABLO-generated programs sometimes need to write intermediate versions of the updated data, possibly after each step of the calculation (this applies if option UCS has been selected in the Code stage of TABLO) and/or after each subinterval (when the simulation is split into several subintervals as described in section 5.8 above). Whenever you are extrapolating, intermediate files hold the data as updated after each separate multi-step calculation. Whenever data is read initially from a text file or the terminal, or there are FORMULA(INITIAL)s, a so-called intermediate extra data file may be required to hold updated values between steps or subintervals.*

Even after you are quite experienced in carrying out multi-step simulations, it may not be easy to anticipate which of these files are needed. In this section we state the precise conditions under which intermediate updated files and intermediate extra data files are needed. We also give advice about choosing names for the final update files which will maximise the chance that GEMSIM and TABLO-generated programs can infer a sensible name for these intermediate files (if needed) and/or offer sensible defaults.

If you are running interactively, these programs will prompt you if and when these files are required. If you are taking inputs from a Stored-input file, you need to anticipate if a prompt for the name of one of these files will be given. If you are using a GEMPACK Command file (see sections 2.2.3, 2.3.1 and A.1 of GPD-1), the names of these intermediate files can often be inferred as a default from the names of other files, and, if in doubt, you can include the relevant statement in case it is needed; if not, it will be ignored.

Names for Final Update Files

We strongly suggest that you always choose names with suffixes of exactly the same length as the other standard GEMPACK file suffixes on your machine. On most machines (including DOS 386/486, Macintosh, Unix and VAX/VMS machines), this means choosing a suffix of length 3 (following the '.'); we find the suffix **.upd** ideal.

On machines where long file names are allowed (including Unix, VAX/VMS and Macintosh), it is practicable to include an indication of the various features of the simulation that affect the updated data (the closure, shocks, possibly the original data base if there are several you could have used, and the method and steps). For example, with Stylized Johansen, the updated data might be called

---

* The suffixes used on most machines for these files are
  - '.ud3', '.ud4' for intermediate data updated after each step,
  - '.ud5', '.ud6', '.ud7' for data updated after each separate multi-step calculation when you are extrapolating, and
  - '.ud8', '.ud9' for updated data after each subinterval if several are in force.

sjlb10_124e.upd

to indicate that labour has been increased by 10 percent ("lb10") and that the simulation used 1,2,4-step Euler calculations ("124e"). Even if you think the first part of the above name is a little excessive, sticking to the '.upd' suffix will help with names of intermediate updated files.

When an Intermediate Updated File is Required

An intermediate updated file is required for any existing Header Array file from which data is read if

- you are extrapolating, and/or
- there are at least 2 subintervals, and/or
- option UCS has been selected from the Code Options Menu in TABLO when the program was written and there is more than one pass in one or more of the multi-step simulations.

An intermediate updated file is never needed for a text file (however then an intermediate extra file will be required, as explained below).

When you use a GEMPACK Command file, you can always include a statement

"intermediate file <logical_name> = <file_name> ;"

even if you are not sure if it will be needed: if not, GEMSIM or your TABLO-generated code will just ignore it. However, if you follow the advice above about suffixes, you should never need one of these statements since the default chosen by the GEMSIM or your TABLO-generated program should suffice.

When an Intermediate Extra File is Required

An intermediate extra file will be required if you are reading data from a text file or the terminal or if there is a FORMULA(INITIAL); otherwise one will not be needed.

When you use a GEMPACK Command file, you can always include a statement

"intermediate extra data = <file_name> ;"

even if you are not sure if it will be needed; if not, GEMSIM or your TABLO-generated program will just ignore it. If data is not read from the terminal or a text file, GEMSIM or TABLO-generated programs will not have a default name to use so you may like to include an "intermediate extra data" statement in this case. (See section 5.14.1 below for more about this.)

Note that several intermediate updated files may be needed (possibly one for each different Header Array file from which data is read) but only ever one intermediate extra file is required.

5.14.1 **Default Names for Intermediate Files When Using a Command File**

If you are running a simulation via a GEMPACK Command file, GEMSIM and TABLO-generated programs will provide default names for intermediate updated files and the intermediate extra data file when they are needed and you didn't provide one. This avoids the program crashing just because you have not specified a name for one of these files.

These defaults are

GPXXX Intermediate extra data file
GPXXd Intermediate updated file (where 'd' is replaced by the number of the file amongst all the FILE statements, for example, GPXX2 for the second one or GPX12 for the twelfth).

The only time when it might not be safe to rely on these default names is if you have two or more programs running at the same time and in the same directory. Then they may overwrite each other's files and the results may be unpredictable unless you specify different names for these files for the different runs in progress at the same time.

5.14.2 **Command File Syntax for Data File Names**

Table 5.14 on the next page summarises the Command file syntax for all the data files.

CASE 1. Original Header Array File

INPUT : Header Array file as original data file

```
file <logical_name> = <filename.suf> ;
```

INTERMEDIATE : Intermediate Updated File

```
intermediate file <logical_name> = <filename> ;
  ! see section 5.14.1 for the default name !
```

OUTPUT : New Header Array file as updated data file

```
updated file <logical_name> = <filename.suf> ;
```

CASE 2. Original Text File (or Terminal Data or FORMULA(INITIAL))

INPUT : Text File or TERMINAL or FORMULA(INITIAL)

```
file <logical_name> =
     <filename.suf> ;
```

INTERMEDIATE : Intermediate Extra File

```
intermediate extra data = <filename> ;
! The default name is GPXXX - see section 5.14.1 !
```

OUTPUT : New TEXT file | New TEXT file | Values not written

```
updated file                updated terminal data =
<logical_name> =                 <filename.suf> ;
<filename.suf> ;
```

Table 5.14: Summary of Command File Syntax for Data Files

CHAPTER 6

VERIFYING ECONOMIC MODELS

There are, of course, errors that TABLO cannot identify. If, for example, you intend a formula

```
FORMULA (all,i,COM) A6(i) = A4(i) + A5(i) ;
```

but inadvertently type either

```
FORMULA (all,i,COM) A6(i) = A4(i) * A5(i) ;
```

or

```
FORMULA (all,i,COM) A6(i) = A3(i) + A5(i) ;
```

(where A3 is another coefficient of your model), TABLO will not know that an error has been made.

Accordingly, before you rely on your model as a research tool, you must perform cross-checks on your model, in order to verify its behaviour. Some helpful checks include

- putting DISPLAY and/or WRITE statements in your TABLO Input file,
- examining the Equations file,
- running simulations, and
- checking the updated data.

Using DISPLAY and/or WRITE statements

Put DISPLAY and/or WRITE statements in your TABLO Input file to check the values of selected coefficients.

In some cases it may be appropriate to have DISPLAYs and writes to the terminal done at all steps of a multi-step simulation. This can be done by selecting option **DWS** (see section 5.6.2); but this should be used sparingly since it may produce vast amounts of output.

Examine the Equations file

The program SUMEQ (see section 8.1 of GPD-1) can be used to do this. SUMEQ can also be used to examine a few individual entries of the tableau, one at a time. (Use the map produced by SUMEQ, as described in section 8.1.1 of GPD-1, to identify which rows and columns correspond to the equation block and variable you are interested in.) However, to display the values of many entries in one row, it is easier to use the column sum facility of SUMEQ - simply take column sums over this row. Proceed similarly to display the values of many entries in one column.

Run Simulations

For most models, there are simulations (often 1-step ones) whose results are known theoretically. Use GEMSIM, the TABLO-generated program or SAGEM to carry out these (and others whose results you believe you understand well) and check the results carefully.

Recall that the easiest way of checking homogeneity simulations is via SUMEQ, as explained in section 8.1.2 of GPD-1.

Checking the Updated Data

The UPDATE statements in your TABLO Input file control how the data is updated after each single step in a multi-step simulation. If they are incorrect, multi-step results will also be incorrect. The best way to check that the UPDATE statements are correct is to carry out thorough checks on the updated data. Because the process of updating is the same after each single step of a multi-step simulation, all this checking can be done on data updated after 1-step simulations. If this update is being done correctly, it is highly likely that data updated after any n-step simulation will also be correct.

When you assembled the original data, you presumably checked it in various ways. For example, you may have checked

1. that it is balanced in various ways (that is, that various accounting identities hold),

2. the results of simulations whose results are known theoretically; this may include certain homogeneity tests, possibly done using the program SUMEQ.

You should carry out these same tests on data updated after 1-step simulations.

You should carry out these tests on data bases updated after shocks to different sets of exogenous variables. (If a variable is exogenous, UPDATE formulas involving that variable are only checked when that variable is given a nonzero shock.) Check balance, if appropriate, and carry out simulations starting from the updated data. (Note that, provided you follow the updating strategies indicated in section 6.1 below, the updated data should be still balanced if the original data is.)

Indeed you should not attempt to carry out any multi-step simulations until you have carried out this testing of data updated after 1-step simulations. Any test that fails may indicate an error in one of your UPDATE statements.

6.1 **Is Balanced Data Still Balanced After Updating?**

The short answer is '*Yes, provided the updating is done in the right way*'. The examples below illustrate this. (Keeping balance after updating can be quite important.)

In the examples below, we use a capital letter such as *X* to denote a levels value, *x** to denote the percentage change in *X* and lower case *x* by itself to denote the fractional change (so that *x*=*x**/100). We also use *X'* to denote the updated value of *X*, so that

$$X' = X(1+x^*/100) = X(1+x).$$

We are grateful to Mark Horridge and Robert McDougall for the examples and insights presented in this section.

Example (i) - Demand equals Supply

We might have an accounting identity of the model of the form

$$\text{SUM}(i,S,\ Vi\ ) = \text{SUM}(j,T,\ Wj\ ) \qquad (1)$$

where *Vi* = *Pi* x *Xi* and *Wj* = *Qj* x *Yj* are flows (dollar values) each a price (*Pi* or *Qj*) times a quantity (*Xi* or *Yj*).

Suppose (as usual) that the database contains the dollar values *Vi* and *Wj*.

Now

*vi** = *pi** + *xi** and so *vi* = *pi* + *xi*. Similarly, *wj* = *qj* + *yj*.

The updated values are *Vi'* = *Vi*(1+*pi*+*xi*) and *Wj'* = *Wj*(1+*qj*+*yj*). Because (1) holds, the linearized form of it must be a consequence of one or more of the linearized equations of the model. Thus the simulation results will have the property that

$$\text{SUM}(i,S,[Pi.Xi/C](pi + xi)) = \text{SUM}(j,T,\ [Qj.Yj/C](qj + yj)\ ) \qquad (2)$$

where *C* is the total of either side of (1) above.

It follows easily that

$$\begin{aligned}\text{SUM}(i,S,\ Vi'\ ) &= \text{SUM}(i,S,\ Vi(1+pi + xi)\ ) \\ &= \text{SUM}(i,S,\ Vi\ ) + \text{SUM}(i,S,\ Vi(pi + xi)\ ) \\ &= \text{SUM}(j,T,\ Wj\ ) + \text{SUM}(j,T,\ Wj(qj + yj)\ ) \quad \text{by (2)} \\ &= \text{SUM}(j,T,\ Wj')\end{aligned}$$

and so the updated data is balanced.

Example (ii) - Income tax

E = household expenditure
WH = factor income (*wage rate* x *hours worked*)
T = power of income tax (e.g. 0.7 if ad valorem tax rate is 30%)

Then $E = WHT$. The linearized version is $e^* = w^* + h^* + t^*$ which, after division by 100 gives

$$e = w + h + t. \qquad (3)$$

Implementation A.

Data base contains
E = expenditure

$B = WH$ = pretax income
$b = w + h$

$C = WH(1-T)$ = tax paid
$c = [WH/C](w+h) - [WHT/C](w+h+t)$
$\quad = [B/C](w+h) - [(B-C)/C](w+h+t)$

If data is balanced, $E = B - C$.
T can be deduced from $T = 1 - C/B$.

After updating,
$E' = E(1+e)$
$B' = B(1+w+h)$
$C' = C(1+c)$
$\quad = C + B(w+h) - (B-C)(w+h+t)$
$\quad = C(1+w+h+t) - Bt$
$\quad = C(1+e) - Bt$ by (3) above.

Thus $B' - C' = B(1+w+h+t) - C(1+e)$
$\quad = B(1+e) - C(1+e)$ by (3) above
$\quad = (B-C)(1+e) = E(1+e) = E'$
so the updated data is balanced.

[Note however that
$T' = 1 - C'/B' = (B' - C')/B'$
$\quad = \{E(1+e)\}/\{B(1+w+h)\}$
$\quad = \{E(1+w+h+t)\}/\{B(1+w+h)\}$
$\quad = T(1+w+h+t)/(1+w+h)$
$\quad = T[1+t/(1+w+h)]$
which is not exactly equal to $T(1+t)$.]

Implementation B.

Data base contains
E = expenditure

$B = WH$ = pretax income
$b = w + h$

T = power of tax

If data is balanced, $E = TB$.

After updating,

$E' = E(1+e)$
$B' = B(1+w+h)$
$T' = T(1+t)$.

Note that $T'B' = T(1+t)B(1+w+h)$
$= TB(1+w+h+t) + TB(tw+th)$
$= E(1+e) + TB(tw+th)$ by (3) above
$= E'$ + second order term.

Thus updated accounts DO NOT balance here.

The moral from these examples is:

To preserve balance, the database should contain only flows and parameters (not rates etc).

Another way of looking at this is in terms of linearity. One of the properties of the procedure for linearizing levels equations is that, when the variables are interpreted as percentage-changes, any equation which is originally linear (such as Example (i) above) is solved **exactly** even in a 1-step simulation. (Only nonlinear equations such as $V=P*Q$ need multi-step simulations to solve exactly.) This means that,

provided

(i) your balance conditions are **linear** in values held in the data base, and

(ii) the expressions in the update statements are **linear** in **percentage-change** model variables,

the balance conditions must still hold after the update.

This means that you should prefer balance conditions which are **linear**. (Note that $E=TB$ in Example (ii) above is not linear.) So another way of stating the moral above is that (provided all, or most, of your variables are percentage-change variables)

Balancing conditions should be linear in data base values.

6.1.1 **Balance After n-Steps**

If your update statements are such as to guarantee that the data is balanced after a 1-step simulation, you can be almost certain that it will also be balanced after any n-step simulation (irrespective of the value of n). [This is because the updated data base after an n-step simulation is obtained by a sequence of n updates each effectively updating a data base on the basis of a 1-step simulation. The data base remains balanced after each single step of the n-step simulation.] The same is true if your solution is obtained by Richardson extrapolation. Again the resulting updated data base will be balanced if any 1-step simulation produces balanced data.

CHAPTER 7

INTERTEMPORAL MODELS

7.1 Introduction to Intertemporal Models

Intertemporal or dynamic models are those having equations linking variables at different points in time.

For example,

```
EQUATION CapAcc # Capital Accumulation # (all,t,FWDTIME)
  k(t + 1) = S1(t) * i(t) + (1.0 - S1(t)) * k(t) ;
```

Intertemporal equations may approximate differential or difference equations. Chapter 5 of Dixon et al (1992) gives a comprehensive introduction to these models. In the method described there and also in Codsi, Pearson and Wilcoxen (1992), the whole system of equations (including both non-intertemporal and intertemporal equations) is solved simultaneously. Values of all variables at all time points are found at once.

You can see examples of TABLO Input files for intertemporal models by looking at the TABLO Input files for the intertemporal models TREES, CRTS and 5SECT usually supplied with GEMPACK (see Appendix B of GPD-1). The TABLO Input file for TREES can also be found in Codsi et al (1992).

7.2 Intertemporal Sets

In an intertemporal model, there are equations (and possibly formulas) in which coefficients and/or variables have arguments of the form 't+1', 't-3'. Argument t+1 refers to the value of the variable or coefficient not at the time-point t but at the time-point t+1, which is one time interval after the current time t. The sets over which such arguments can range are called **intertemporal sets**: they are usually time-related.

- TABLO requires you to indicate when declaring each set whether it is to be used as an intertemporal set (that is, if an argument of the form 't+n' or 't-n' can be in it). Use the SET qualifier 'INTERTEMPORAL' in declaring such a set, as in the example below.

  ```
  SET (INTERTEMPORAL) alltime0  SIZE 11 ( p[0] - p[10] ) ;
  ```

  Thus there are two alternative SET qualifiers (INTERTEMPORAL)

and (NON_INTERTEMPORAL). The latter is the default and need not be included. Either the exact SIZE must be specified or a MAXIMUM SIZE specified.

- For most intertemporal models in which the number of time periods is left flexible (to be read at run-time), the declarations of the intertemporal sets required will be very similar to those in the example below.

Typical Example of Intertemporal Sets

```
COEFFICIENT (INTEGER) NINTERVAL
    ! number of time grid intervals is NINTERVAL !
    ! number of time grid points is NINTERVAL+1 ! ;
READ NINTERVAL FROM TERMINAL ;  ! or from some file !

SET (INTERTEMPORAL) alltime MAXIMUM SIZE 101
      (p[0] - p[NINTERVAL]) ;
SET (INTERTEMPORAL) fwdtime MAXIMUM SIZE 100
      (p[0] - p[NINTERVAL-1]) ;
SET (INTERTEMPORAL) endtime SIZE 1
      (p[NINTERVAL]) ;

SUBSET fwdtime IS SUBSET OF alltime ;
SUBSET endtime IS SUBSET OF alltime ;
```

The set 'alltime' is all time grid points. The set 'fwdtime' is the range of "forward-looking" equations and formulas such as

```
FORMULA (all,i,fwdtime) DT(t) = YEAR(t+1) - YEAR(t) ;
```

The set 'endtime' is for expressing terminal conditions. You may also need sets 'backtime' and 'begintime' as below for "backward-looking" conditions and initial conditions respectively.

```
SET (INTERTEMPORAL) backtime MAXIMUM SIZE 100
      (p[1] - p[NINTERVAL]) ;
SET (INTERTEMPORAL) begintime SIZE 1
      (p[0]) ;
```

7.2.1 **Set Size - Fixed or Determined at Run Time**

- Intertemporal sets can have fixed size or their sizes can be read at run-time. When of fixed size, their elements can be fixed (as for a non-intertemporal set). For example,

```
SET (INTERTEMPORAL) alltime1 ( p0-p10 ) ;
```

has the fixed elements 'p0', 'p1', ..., 'p10' (and fixed size 11).

- Especially when you are using finite difference approximations to a differential equation of the original model, you will want to be able to vary the number of time periods. Then only the start and end times are important in specifying equations and the intermediate times will never occur explicitly in equations of formulas. For this reason, intertemporal sets can also be declared in the following flexible way which leaves their size to be determined at run-time and leaves their elements unspecified except that the first and last elements have a logical form which can be used to express subsets and initial or terminal conditions. The declaration of an intertemporal set whose elements are intertemporally-defined must be of the form (1) below. [Alternatively, the word MAXIMUM can be omitted if the exact size can be inferred from the declaration as in 'endtime' below.]

  SET (INTERTEMPORAL) <set-name> MAXIMUM SIZE <integer>
  (p[<initial-element>] - p[<end-element>]) ; ! (1) !

  where <initial-element> and <end-element> are replaced by expressions of the form

  integer_coefficient +/- integer or integer

  as in the Typical Example above.

- Different "starting string"s can be used. The letter "p" in the Typical Example above could be replaced by other character strings such as

  SET (INTERTEMPORAL) alltimex MAXIMUM SIZE 101
  (time[0] - time[NINTERVAL]) ;

  Intertemporal sets declared above are said to have their **elements defined intertemporally**. The square brackets [ ] as in p[NINTERVAL] are what distinguish this type of element declaration from others. The characters before the '[' ("p" or "time" in the examples above) are called the **intertemporal element stem**.

- Such intertemporally-defined elements cannot be used explicitly in equations or formulas.
  For example, X("p[3]") and X("p[NINTERVAL]") are not allowed in a formula. For this reason a terminal condition would need to be expressed as an equation or formula ranging over (all,t,endtime).

  Thus you should distinguish between the sets 'alltime0' and 'alltime1' defined by

  SET (INTERTEMPORAL) alltime0 SIZE 11 (p[0] - p[10]) ;
  SET (INTERTEMPORAL) alltime1 (p0 - p10) ;

  In the first of these the [ ] means that the elements are intertemporally-defined (not fixed) and the dash '-' is just to indicate the first and last elements of this sets. In the second of these, 'p0 - p10' is an abbreviation for

p0,p1,p2,p3,p4,p5,6p,p7,p8,p9,p10

and the elements are fixed.

- If intertemporal sets are to be used in SUBSET statements, they must fall into one of the following three categories.

  (1) Fixed size and fixed elements (as for set 'alltime1' above),

  (2) Fixed size and intertemporally-defined elements (see 'endtime' in the Typical Example above),

  (3) Run-time size and intertemporally-defined elements (see 'alltime' in the Typical Example above)

- In a SUBSET statement

  SUBSET <set1> IS SUBSET OF <set2> ;

  the sets 'set1' and 'set2' must both be intertemporal or both non-intertemporal. If they are both intertemporal sets, this SUBSET must be BY_ELEMENTS (not BY_NUMBERS) and either both sets must have fixed elements or both must have intertemporally-defined elements. [Recall that BY_ELEMENTS is the default for SUBSET statements (see section 3.2 above).] For the sets as defined above, the following would have the obvious effect.

  SUBSET fwdtime IS SUBSET OF alltime ;

- For intertemporal sets with fixed elements, the small set cannot have "gaps" in it. For example, the following would cause an error.

  ```
  SET (INTERTEMPORAL) time0 ( p0 - p10 ) ;
  SET (INTERTEMPORAL) time3 ( p0, p2, p4, p6, p8, p10 ) ;
  SUBSET time3 IS SUBSET OF time0 ;
  !incorrect!
  ```

  This is because arguments such as 't+1' in equations must have an unambiguous meaning. In the example above, if 't' were in 'time3' and t equals 'p0', we would not know if 't+1' refers to 'p2' (the next element in 'time3') or to 'p1' (the next element in 'time0').

- Intertemporally-defined element names (such as "p[23]") are used on GEMPIE Print files when reporting simulation results.

The TABLO Input file for the TREES intertemporal model is shown in Codsi et al (1992). You can also look at the TABLO Input files for the intertemporal models TREES, CRTS and 5SECT usually supplied with GEMPACK (see Appendix B of GPD-1).

7.3 **Use an INTEGER Coefficient to Count Years**

In intertemporal models, it is often necessary to have a COEFFICIENT, perhaps called YEAR(t), which tells the date in years (relative to some base date) of time instant 't'. (For example, in a 10-interval model spanning 30 years, YEAR(t) may take the values 0,3,6,9,...,24,27,30 or possibly 1990,1993,1996,...,2014,2017,2020.) It may be best to declare this to be an INTEGER COEFFICIENT, especially if it (or quantities derived from it such as DT(t) - see below) are used in an exponent (that is, in the "B" part of an expression A^B, A raised to the power B). This is because, if A is negative, some Fortran compilers will evaluate A^B when B is an integer but will not evaluate it if B is the same real number. For example, they will evaluate (-2)^B if B is the integer 3 but not if B is the real number 3.0.

Typical statements in a TABLO Input file are as follows.

```
COEFFICIENT (INTEGER)  (all,t,alltime)  YEAR(t) ;
READ YEAR FROM FILE time ;
COEFFICIENT (INTEGER)  (all,t,fwdtime)  DT(t) ;
FORMULA  (all,t,fwdtime)    DT(t) = YEAR(t+1)-YEAR(t) ;
```

CHAPTER 8

LESS OBVIOUS EXAMPLES OF THE TABLO LANGUAGE

In this chapter we discuss some less obvious examples of the use of the TABLO language. Sometimes it is not obvious whether or not certain kinds of economic behaviour can be expressed accurately using the syntax and semantics of TABLO. These examples may help you see how to convert a wider range of behaviour into the TABLO language.

8.1 Adding Across Time Periods in an Intertemporal Model

In an intertemporal model you may have some quantity, say investment, measured for each grid interval and you may wish to accumulate it to tell how much investment has occurred since the first time instant. Typical declarations might be as follows (where we have the same time sets as described in section 7.2 above).

```
COEFFICIENT
  (all,t,fwdtime)  FWDINVEST(t)  #Investment between t and t+1# ;
COEFFICIENT
  (all,t,alltime)  TOTINVEST(t)  #Total investment from 0 to t# ;
```

Here the formula that should apply is

```
TOTINVEST(t) = SUM( u<t, FWDINVEST(u) ) ;
```

Although this is not a legal TABLO statement, it can be easily turned into one using a coefficient YEAR(t) which associates years (from some arbitrary starting date) with time instant 't' (see section 7.3 above). Then the formula for TOTINVEST can be given in the following TABLO statement.

```
FORMULA (all,t,alltime)
   TOTINVEST(t) = SUM( u,fwdtime:YEAR(u)<YEAR(t), FWDINVEST(u) ) ;
```

This conditional SUM (the condition depends on YEAR) does what is required. Notice that when 't' is the first time instant in the set alltime, the condition YEAR(u)<YEAR(t) will not be true for any time instants 'u', so the sum (value for TOTINVEST(t) for the first time instant) will be zero, as required.

8.2 **Conditional Functions or Equations**

In some cases you may have two (or more) different kinds of behaviour that may apply, the first kind applying for some sectors say and the second kind applying for the other sectors. Then you have a conditional function, that is, one whose values depend on which of the two sets the argument is in. An example would be

$$(\text{all},i,\text{SECT})\quad F(i) = \begin{cases} G(i) + T(i) & \text{if } i \text{ is in SECT1,} \\ W(i) + V(i) & \text{if } i \text{ is in SECT2,} \end{cases}$$

where SECT1 and SECT2 are subsets of SECT such that everything in SECT is either in SECT1 or SECT2 (but nothing is in both), and G(i),T(i),W(i),V(i) are coefficients (or variables) whose values are already determined.

One neat way of expressing this in the TABLO language is as follows.

```
COEFFICIENT (all,i,SECT) SUBSECT(i) ;
READ SUBSECT FROM FILE ... ;

! On your data file arrange the values of SUBSECT so that
       SUBSECT(i) = 1 if i is in SECT1
  and  SUBSECT(i) = 2 if i is in SECT2.       !

FORMULA     (all,i,SECT)
        F(i) =   IF( SUBSECT(i)=1, G(i) + T(i) ) +
                     IF( SUBSECT(i)=2, W(i) + V(i) ) ;
```

This use of two conditional expressions using "IF" (see section 4.4.6) means that the conditional function can be written in one TABLO statement. Of course, the TABLO statement could be a FORMULA (as above) or an EQUATION. In the latter case, one advantage of writing it in this way is that the equation could then be used to substitute out the variable on the left-hand side; this could not be done if the equation were expressed in two parts such as

```
EQUATION eq1 (all,i,SECT1) c_F(i) = c_G(i) + c_T(i) ;
EQUATION eq2 (all,i,SECT2) c_F(i) = c_W(i) + c_V(i) ;
```

A similar example is that of ETA(i1,i2) in section 4.10.6 above, which shows a different way of implementing a conditional formula. However the procedure there, which relies on the order in which FORMULAs are evaluated, would not work for EQUATIONs (which are essentially order-independent).

Example 2 in section 2.3.1 above shows another way of combining two equations into one.

8.3 **Mappings Between Sets**

Mappings between two sets are not obviously catered for in the TABLO language. An example from ORANI will show how they might be handled.

In ORANI, industries can produce several commodities, but most industries (the single-product ones) only produce one commodity and then the commodity they produce is not produced by any other industry. There are two sets, COM of commodities and IND of industries and neither is a subset of the other. The single-product industries form a subset SP_IND of IND and the commodities they produce form a subset SP_COM of COM. One of the market clearing equations says that supply SUPP(j) from industry 'j' in SP_IND is equal to the demand DEM(i) for the commodity 'i' it produces. That is,

```
  (all,j,SP_IND)  SUPP(j) = DEM( COMPROD(j) )
```

where COMPROD(j) is the commodity produced by industry 'j'. Here COMPROD is a function taking values in IND to values in COM, that is, a mapping between sets. Such mappings are not directly part of the TABLO language. However, they can be handled by setting up (on the data base) two functions (formally COEFFICIENTs) indicating the commodity numbers of the commodity produced by the single-product industries and the commodity numbers of the elements of COM, as shown below.**

```
COEFFICIENT (INTEGER) (all,i,COM) COMNO(i) ;
! COMNO(i) is the number in the set COM of element 'i' !
COEFFICIENT (INTEGER) (all,j,SP_IND) SPCOMNO(j) ;
! SPCOMNO(j) is the number in the set COM of the commodity produced
   by industry 'j' in SP_IND !
READ COMNO FROM FILE ... ;
READ SPCOMNO FROM FILE ... ;

VARIABLE (all,j,IND) SUPP(j) ;
VARIABLE (all,i,COM) DEM(i) ;

EQUATION   eq1   (all,j,SP_IND)
   SUPP(j) = SUM( i,SP_COM: COMNO(i) = SPCOMNO(j), DEM(i) ) ;
```

This use of a conditional SUM means that, for each 'j', the only term on the right-hand side is DEM(i) where 'i' is the unique commodity produced by industry 'j'. The condition "COMNO(i)=SPCOMNO(j)" eliminates all other commodities 'i'.

In cases like this, it is probably best to declare the coefficient number mappings to be INTEGER COEFFICIENTs as we have done above, though they could be REAL COEFFICIENTs. In the example above, the EQUATION and VARIABLEs could be either linear ones or levels.

The next section also contains an example of mappings between sets.

** We are indebted to Kevin Hanslow for the idea of using COEFFICIENTs such as COMNO and SPCOMNO to indicate commodity numbers in situations like the one described in the text.

8.4 **Aggregating Data**

Suppose you have various arrays of data specified over a set COM=(c1,c2,c3,c4,c5,c6) of commodities, and you wish to aggregate all these arrays to a set AGGCOM=(aggc1,aggc2) of aggregated commodities such that original commodities c1,c2,c4,c5 contribute to aggregated commodity aggc1 and c3,c6 contribute to aggc2. This can be done quite effectively by defining mappings (formally INTEGER COEFFICIENTs) as described below; set up their values on some file (Header Array or a GEMPACK text file). Then it is easy to write a TABLO Input file to carry out the aggregation of data required.

```
COEFFICIENT (INTEGER) (all,i,COM) AGGCOMNO(i) ;
READ AGGCOMNO FROM FILE ... ;
! For example, for the above aggregation, the values read
   into AGGCOMNO would be
   AGGCOMNO("c1")=1, AGGCOMNO("c2)=1, AGGCOMNO("c3")=2,
   AGGCOMNO("c4")=1, AGGCOMNO("c5)=1, AGGCOMNO("c6")=2. !

COEFFICIENT (INTEGER) (all,aggi,AGGCOM) AGGNO(aggi) ;
! AGGNO("aggc1")=1,AGGNO("aggc2")=2 !
READ AGGNO FROM FILE ... ;

COEFFICIENT (all,i,COM) X(i)  # Data before aggregation # ;
COEFFICIENT (all,aggi,AGGCOM) AGGX(aggi)  # Data after aggregation # ;

!
  The next three statements will need to be repeated for the
   different arrays over COM which you want to aggregate.
   Each time you will probably only need to change
   the HEADER you read X from and the HEADER and LONGNAME
   indicated when writing AGGX.
!
READ X FROM FILE ... HEADER "ABCD" ;
! The next FORMULA aggregates (all,i,COM) X(i) to
    (all,aggi,AGGCOM) AGGX(aggi), as required.
  For each aggi in AGGCOM, it adds up just those X(i) for
     which i in COM corresponds to aggi  in AGGCOM. !
FORMULA (all,aggi,AGGCOM) AGGX(aggi) =
                SUM(i,COM: AGGCOMNO(i)=AGGNO(aggi), X(i) ) ;
WRITE AGGX TO FILE ... HEADER "ABCD" LONGNAME "LLLLLLLLLL" ;
```

Of course this could be extended to other sets and to coefficients with two or more arguments.

Note that the mappings used here are similar to those in section 8.3 above.

APPENDIX A

LINEARIZING LEVELS EQUATIONS

In this appendix we state in section A.1 the differentiation rules used by TABLO when it differentiates levels equations in TABLO Input files. We also discuss in section A.2 how you might go about linearizing equations by hand if you want to put linearized equations directly into a TABLO Input file.

A.1 Differentiation Rules Used by TABLO

A list of the differentiation rules used by TABLO to differentiate levels equations in TABLO Input files is given below. We include both change and percentage-change differentiation rules for each expression since, as indicated in section 2.2.3 above, TABLO sometimes uses change differentiation and sometimes uses percentage-change differentiation. We indicate the derivation of some of these in section A.2 below. In the rules below, dA denotes the differential of (or change in) A and pA denotes the percentage change in A.

Expression	Change Differentiation	Percentage-change Differentiation
C = A + B	dC = dA + dB	pC = A/(A+B)*pA + B/(A+B)*pB
C = A - B	dC = dA - dB	pC = A/(A-B)*pA - B/(A-B)*pB
C = A * B	dC=B*dA + A*dB	pC = pA + pB
C = A / B	dC=(B*dA-A*dB)/B^2	pC = pA - pB
C = A ^ B	dC = B*A^(B-1)*dA + LOGE(A)*A^B*dB	pC = B*pA + LOGE(A)*B*dB
C = F(A) [Here F is a function of one variable, and F' is its derivative]	dC = F'(A)*dA	pC= F'(A)*A/F(A)*pA
C = SUM(j,IND,A(j))	dC=SUM(j,IND,dA(j))	pC=1/SUM(k,IND,A(k))*SUM(j,IND,A(j)*pA(j))
C = PROD(j,IND,A(j))	dC = PROD(k,IND,A(k))*SUM(j,IND,1/A(j)*dA(j))	pC = SUM(j,IND,pA(j))

For each expression the algorithm keeps dividing the expression into simpler and simpler expressions till it reaches the bottom, that is the differential of a levels variable. For a levels variable Y whose associated linear variable is a CHANGE variable c_Y, the differential dY of Y is replaced by c_Y. For a levels variable X whose associated linear variable is the PERCENT_CHANGE variable p_X, the differential dX of X is replaced by

X/100*p_X

For a parameter, the differential or change is zero.

A.2 **Linearizing Equations by Hand**

We introduce the general procedure used in section A.2.1. In section A.2.2, we write down a list of helpful rules you can use and give examples of their use in section A.2.3. In section A.2.4 we list some references in which you can find linearizations of standard situations; you can often take the linearizations from these and put them into your TABLO Input files.

A.2.1 **General Procedure**

Consider a levels equation

$$f(P,Q,R,...) = 0 \qquad (1)$$

relating levels variables P,Q,R etc of the model. The standard linearization of this, which is obtained by totally differentiating both sides of (1), is

$$f_P \,.\, dP + f_Q \,.\, dQ + f_R \,.\, dR + ... = 0 \qquad (2)$$

where f_P, f_Q and f_R denote the partial derivative of f with respect to P,Q,R respectively. It is usual to think of this linearization (2) as relating small changes dP, dQ, dR, ... in the variables of the model.

If the linear variables associated with P,Q,R,... are change variables c_P, c_Q, c_R, ..., we have the linearization

$$f_P \,.\, c_P + f_Q \,.\, c_Q + f_R \,.\, c_R + ... = 0 \qquad (3)$$

If the linear variables associated with P,Q,R,... are percentage-change variables p_P, p_Q, p_R, ..., we can relate the percentage change p_V in V to the change dV in V via

$$p_V = 100.dV/V \qquad (4)$$

or, equivalently,

$$dV = V.p_V/100 \qquad (5)$$

Then, from (2), we have the linearization

$$f_P \cdot \frac{P.p_P}{100} + f_Q \cdot \frac{Q.p_Q}{100} + f_R \cdot \frac{R.p_R}{100} + \ldots = 0 \qquad (6)$$

of (1).

Of course if some associated linear variables are change variables and some are percentage-change variables, we have a linearization which includes parts of (3) and parts of (6).

Example

Suppose

$$R = CPQ \qquad (7)$$

where P,Q,R are variables and C is a parameter (a constant). Then

$$R - CPQ = 0$$

and so, following the method above, the linearization of (7) is

$$dR - C.P.dQ - C.Q.dP = 0. \qquad (8)$$

If all associated linear variables are change variables, we have the linearization

$$c_R - C.P.c_Q - C.Q.c_P = 0. \qquad (9)$$

In (9), we could replace R by CPQ, from (7), and divide both sides by CPQ and multiply by 100 to obtain

$$p_R - p_Q - p_P = 0$$

or

$$p_R = p_P + p_Q \qquad (10)$$

This is the so-called Product Rule (see section A.2.2 below) for linearizing a product such as (7).

A.2.2 **Rules to Use**

It is easy to derive other useful rules (shown below) for linearizing levels equations. The ones we list below all assume that the associated linear variables are percentage-change variables.

Rules for Linearizing a Levels Equation

Product Rule If R = CPQ then p_R = p_P + p_Q (if C is constant).

Quotient Rule If R = CP/Q then p_R = p_P - p_Q (if C is constant).

Power Rule If $R = C.P^D$ then p_R = D.p_P (if C and D are constants).

Sum Rules

(a) If R = P + Q then p_R = (P/R)*p_P + (Q/R)*p_Q.

(b) If R = P + Q then R*p_R = P*p_P + Q*p_Q.

Difference Rules

(a) If R = P - Q then p_R = (P/R)*p_P - (Q/R)*p_Q.

(b) If R = P - Q then R*p_R = P*p_P - Q*p_Q.

Table A.2.2: Rules for Linearizing Levels Equations

The two versions (a) and (b) of the Sum and Difference Rules are worth noting. The (a) versions are the ones with shares P/R,Q/R in them, while the (b) versions have no such shares. The share versions were used commonly (see, for example, Dixon et al (1982)), but the non-share (b) versions are perhaps simpler and deserve more use. We, and others, are indebted to Mark Horridge for pointing out the desirability of the (b) versions.

If some associated linear variables are change variables, it is easy to write down or derive similar rules to those in Table A.2.2 above.

Usually most equations can be linearized using the above rules; it is only occasionally necessary to linearize equations by partial differentiation following equations (2),(3) and (6) in section A.2.1 above.

A.2.3 **Linearizing Equations in Practice**

Most levels equations involving just arithmetic operations (that is, +,-,*,/,^) can be linearized easily using the rules in Table A.2.2 above (provided all associated linear variables are percentage-change variables).

Example 1

Suppose that

A = BCD.

Then, if all associated linear variables are percentage-change variables, we have

```
  p_A = %change in (BCD)
      = %change in [ (BC)*D ]
      = %change in (BC) + %change in D      [by Product Rule]
      = (p_B + p_C) + p_D                   [by Product Rule]
      = p_B + p_C + p_D.
```

Example 2

Suppose that

$$A = BC + D.$$

Then, if all associated linear variables are percentage-change variables, we have

```
p_A = %change in [ (BC) + D]
    = [BC/(BC+D)]*%change in (BC) + [D/(BC+D)]*p_D  [by Sum Rule (a)]
    = [BC/(BC+D)]*[p_B + p_C] + [D/(BC+D)]*p_D      [by Product Rule]
```

Of course, if you have any difficulty, you can always put the levels equation directly in your TABLO Input file, and let TABLO differentiate it for you.

A.2.4 **Linearizing Using Standard References**

References containing linearizations of standard situations include

Dixon et al (1992),
Dixon et al (1982),
Dixon et al (1980),
Hertel et al (1992),
Horridge et al (1993).

For example, these can be used to find linearizations of the solution to problems involving maximisation/minimisation subject to standard functions such as CES, Cresh, CET. (For example, linearizations of several problems of this kind are derived in Problem Set C in Chapter 3 of Dixon et al (1992).) In many cases, it is easy to make an appropriate modification of one of the linearizations in one of these references to include in your TABLO Input file.

APPENDIX B

LINEARIZED EQUATIONS ON INFORMATION FILE

Below we reproduce part of the Information file produced when TABLO processes the file SJ.TAB in section 3.3.2 of GPD-1. This shows the linearized EQUATIONs associated with the levels EQUATIONs.

```
117   FORMULA & EQUATION Comin
118      # Intermediate input of commodity i to industry j #
119   (all,i,SECT)(all,j,SECT) XC(i,j) = DVCOMIN(i,j) / PC(i) ;
   !  EQUATION(LINEAR) Comin
      (ALL,j,SECT) (ALL,i,SECT)
       p_XC(i,j) - {p_DVCOMIN(i,j) - [p_PC(i)]} = 0 ; !

120
121   FORMULA & EQUATION Facin
122      # Factor input f to industry j #
123   (all,f,FAC)(all,j,SECT) XF(f,j) = DVFACIN(f,j) / PF(f) ;
   !  EQUATION(LINEAR) Facin
      (ALL,j,SECT) (ALL,f,FAC)
      p_XF(f,j) - {p_DVFACIN(f,j) - [p_PF(f)]} = 0 ; !

124
125   FORMULA & EQUATION House
126      # Household demand for commodity i #
127   (all,i,SECT) XH(i) = DVHOUS(i) / PC(i) ;
   !  EQUATION(LINEAR) House
      (ALL,i,SECT) p_XH(i) - {p_DVHOUS(i) - [p_PC(i)]} = 0 ; !

128
129   FORMULA & EQUATION Com_clear ! (E3.1.6) in DPPW !
130      # Commodity market clearing #
131   (all,i,SECT) XCOM(i) = XH(i) + SUM(j,SECT,XC(i,j)) ;
   !  EQUATION(LINEAR) Com_clear
      (ALL,i,SECT) XCOM(i) / 100.0  * p_XCOM(i) -
      {XH(i) / 100.0  * p_XH(i) +
          SUM(j,SECT,XC(i,j) / 100.0  * p_XC(i,j))} = 0 ; !

132
133   FORMULA & EQUATION Factor_use ! (E3.1.7) in DPPW !
134       # Aggregate primary factor usage #
135   (all,f,FAC) XFAC(f) = SUM(j,SECT,XF(f,j)) ;
   !  EQUATION(LINEAR) Factor_use
      (ALL,f,FAC) XFAC(f) / 100.0  * p_XFAC(f) -
        {SUM(j,SECT,XF(f,j) / 100.0 * p_XF(f,j))}  = 0 ; !
```

APPENDIX C

EXAMPLES OF THE DERIVATION OF UPDATE STATEMENTS

C.1 Example 1 - A Levels Data Base Value Which is the Sum of Two Flows

Consider the data base (dollar) levels value *VL* which is read from the data base, and which is related to other levels values by

$$VL = PL*XL + QL*YL \qquad (1)$$

where *PL* and *QL* are (levels values) of prices and *XL* and *YL* are quantity volumes for goods *X* and *Y*, say. Assume that the model contains linear variables *p*, *q*, *x* and *y* which are percentage changes in *PL*, *QL*, *XL* and *YL* respectively. Furthermore, assume that the model does NOT contain any linear variable that measures either the change or percentage change in the value of *VL*.

Because of the addition in (1), a Product Update statement cannot be used (see section 3.5.3 in GPD-1), so we must use a CHANGE UPDATE statement, derived as explained below. First we need to linearize (1) above. If *v* represents the percentage change in *VL* (which is NOT a variable in the TABLO Input file of the model) then

$$v = [(PL*XL)/VL]*(p + x) + [(QL*YL)/VL]*(q + y)$$

by the usual linearization procedures (see section A.2 of Appendix A). Here the coefficients of $(p+x)$ and $(q+y)$ are the respective shares (of total dollar value) of each of the two goods *X* and *Y*. Such shares are usually defined as explicit coefficients using TABLO FORMULA statements, as in

```
FORMULA S1 = (PL*XL)/VL ;
        S2 = (QL*YL)/VL ;
```

(Alternatively S2 can be defined via FORMULA S2 = 1 - S1 ;)

Then *v*, the percentage change in *VL*, can be rewritten as

$$v = S1*(p + x) + S2*(q + y).$$

The expression for the change in *VL* is $VL*(v/100)$ which, in terms of variables in the TABLO Input file, equals

$$VL * [\{S1*(p+x)+S2*(q+y)\}/100].$$

Example 1 - A Levels Data Base Value Which is the Sum of Two Flows

The UPDATE statement in the TABLO Input file would be

```
UPDATE (CHANGE) VL = VL *  {S1*(p+x)+S2*(q+y)}/100  ;
```

C.2 **Example 2 - Powers of Taxes**

A power of a tax is an alternative way of describing tax rates. An *ad valorem* tax *T* (specified as a fraction, say 0.2 for a 20 per cent tax or -0.1 for a 10 per cent subsidy) can be related to an equivalent power of tax *PW* by the simple relationship

$$PW = 1 + T.$$

Because *PW* takes a value that is always greater than zero (provided we can exclude the possibility of a -100 per cent *ad valorem* tax rate), it can appear in the model as a percentage change variable. Consider an example in which commodity *X* has a (base) levels price *PX*, which is taxed at the **power of tax** rate of *PW*, and whose quantity volume is represented by *XL*. Suppose that the data base contains the pre- and post-tax values *VL* and *WL* respectively. Then

$$VL = PX*XL, \qquad (2)$$

$$WL = PW*PX*XL. \qquad (3)$$

Suppose the model (that is, the TABLO Input file) contains linear variables *x* and *p*, which are percentage changes of *XL* and *PX* respectively, and that the model also contains the linear variable *pw* which is the **percentage change in the value of the power of tax** *PW*. Since equation (3) expresses *WL* as the product of three levels values whose percentage change versions are variables of the model, *WL* can be updated by the Product UPDATE statement

```
UPDATE  WL = pw * p * x ;
```

and, similarly *VL* can be updated by the following Product UPDATE.

```
UPDATE  VL = p * x ;
```

REFERENCES

Atkinson, Kendall E. (1989) *An Introduction to Numerical Analysis*, second edition, Wiley, New York.

Codsi, G., K.R. Pearson and P.J. Wilcoxen (1992), 'General-Purpose Software for Intertemporal Economic Models', *Computer Science in Economics and Management* vol.5, pp.57-79. [A preliminary version was *Impact Preliminary Working Paper* No. IP-51, Melbourne (May 1991), pp.39.]

Dixon P.B., S. Bowles and D. Kendrick (1980), *Notes and Problems in Microeconomic Theory*, North-Holland, Amsterdam.

Dixon P.B., B.R. Parmenter, J. Sutton and D.P. Vincent (1982), *ORANI: A Multisectoral Model of the Australian Economy*, North-Holland, Amsterdam.

Dixon, P.B., B.R. Parmenter, A.A. Powell and P.J. Wilcoxen (1992), *Notes and Problems in Applied General Equilibrium Economics*, North-Holland, Amsterdam.

Hertel, T.W., J.M. Horridge and K.R. Pearson (1992), Mending the Family Tree: A Reconciliation of the Linearized and Levels Schools of AGE Modelling, *Economic Modelling*, vol.9, pp.385-407. [A preliminary version was *Impact Preliminary Working Paper* No. IP-54, Melbourne (June 1991), pp.45.]

Horridge, J.M., B.R. Parmenter and K.R. Pearson (1993), ORANI-F: A General Equilibrium Model of the Australian Economy, *Economic and Financial Computing*, vol.3, pp.71-140.

Pearson K.R. (1991), 'Solving Nonlinear Economic Models Accurately via a Linear Representation', *Impact Preliminary Working Paper* No. IP-55, Melbourne (July), pp.39.

*GEMPACK DOCUMENTS**

GPD-1, *An Introduction to GEMPACK*, Second edition, April 1994, pp.252+15.

GPD-3, *How to Create and Modify GEMPACK Header Array Files Using the Program MODHAR*, Third edition, April 1993, pp.27+4.

---

* The numbering of GEMPACK Documents has been re-started with Release 5 of GEMPACK, when the abbreviation "GPD" was first used. Previous editions of these documents did not have the same numbers as the current editions. Pre-Release-5 documents are numbered "GED-xx".

INDEX

Absorbed variable, 2-16
Absorption
  Example, 2-16
  Meaning of, 2-16
Accumulate, 8-1
Accuracy, 5-13
  Different methods, 5-3
  Iterative refinement, 5-13
  Required figures, 5-14
  Reuse of pivots, 5-13
  Subintervals, 5-20
  Warnings, 5-7
Actions, 5-24
  Options, 5-15
Active index, 4-13 to 4-14
Actual filename, 4-20
  Length of, 4-3
Aggregating data, 8-4
ALL
  Syntax, 4-6
ALL index, 4-13
Arguments, 2-13, 4-5
  Description, 4-5
  Indices in expressions, 4-13
Auxiliary files
  GEMSIM, 5-22
  TABLO-generated program, 5-22
Auxiliary Statement,Table files, 5-22

Backsolve
  Requirements, 2-12
  System-initiated, 2-14
Balanced equations, 6-1
Base Coefficients Values file, 5-6
Base data
  Coefficient, 3-6
  Data file, 3-8
  Different, 4-20
Brackets
  Use of, 4-8

Case, 4-2
Change
  Rules, A-1
  Variable, 3-7
Change Update, 3-14, 4-27
Check
  TABLO, 2-1
Check stage, 2-1
Checking models, 6-1
Code
  TABLO, 2-1
Code parameter
  see Parameter
Code stage, 2-1
COEFFICIENT
  Statement, 4-1
  Syntax, 3-6
Coefficient
  Active index, 4-13
  Arguments, 4-5
  Assign values, 4-33
  Associated data, 4-23
  Declaring, 4-31
  Dimension, 3-6
  Divide by zero, 4-29
  Examples, 4-5, 4-23, 4-33
  Integer, 3-6, 4-14, 7-5
  Parameter, 3-6
  Partial reads, 4-24
  Scalar, 3-15
  System-initiated, 2-14
  Types, 4-16
  Upper case, 4-2
  Use in equations, 4-12
  Values in equations, updates, 4-34
  Zero, 5-11
Coefficient initialisation, 4-25 to 4-26
Coefficient name, 4-3
  Length of, 4-3
Column order, 3-8, 4-22, 5-18
Command files, 5-11, 5-30, 5-37
Comments, 4-2 to 4-3
Comparison operators, 4-10
Compile, Link, 5-30
Components
  see Variable components
Composite variable, 2-16
  Example, 2-17
Condensation, 2-3
  Examples, 2-10
Condense
  TABLO, 2-1
Condense stage, 2-1
Conditional expressions, 2-14, 4-11

Conditional function, 8-2
Conditional operators, 4-9
Conditional SUM, 8-3
Conditions, 4-6
  In ALLs or SUMs, 4-9
Constants, 4-13
Convergence, 5-3
CPU time, 5-19

Data
  Associated with coefficients, 4-23
Data file
  Intermediate, 5-37
  Updated, 5-37
Default Options
  GEMSIM and TABLO-generated programs, 5-11
Default statements
  Meaning, 3-17
  Syntax, 3-17
Default values of qualifiers
  see Default statements
Default WFP,PGS, 5-24
Differentiation
  Rules, A-1
Dimension of coefficient, 3-6
Dimension of variable, 3-7
Disk space, 5-26
DISPLAY
  Statement, 4-1
  Syntax, 3-16
Display
  All steps, 5-18
  Checking models, 6-1
  Integer coefficients, 3-16
  Order of, 4-31
  Partial, 4-24
Display file, 4-26
Divide by zero
  Errors, 5-30
  Shares, 4-29
  ZERODIVIDE, 3-15
Division, 4-15

Echo activity, 5-19
Elimination, 2-13, 4-34
End-point
  Gragg's method, 5-3
EQUATION, 2-16
  Statement, 4-1
  Syntax, 3-12
Equation
  Active index, 4-13
  Block, 4-32, 6-2
  Example, 4-33
  Expressions in, 4-7
  Levels, 3-12
  Linear, 3-12
  Linear variables, 4-12
  Order in Input file, 4-31
  Order of, 4-32 to 4-33
  Order of indices, 4-32
  Values of coefficients, 4-34
Equation block, 2-16
Equation name, 3-12, 4-3
  Length of, 4-3
EQUATION(NONE) statement, 3-12, 4-36
Equations
  Order of, 4-33
Equations file, 5-6
  Checking, 4-33, 6-1
  SUMEQ, 6-2
  Variable order, 4-31 to 4-32
Equations Matrix, 5-16
Errors, 2-1, 5-30
  Identifying, 2-4
Euler's method, 5-1
Exponent, 4-7, 4-13
Exponential function, 4-9
Expression, 4-7
  Brackets, 4-8
  Complicated, 4-34
  Conditional, 4-11
  Constants in, 4-13
  Operators in, 4-7
  SUM, 4-8
Extrapolation
  Subintervals, 5-20
Extrapolation Accuracy file, 5-22
  Variables, 5-14
Extrapolation Accuracy Summary, 5-3, 5-13, 5-22

FILE
  Statement, 4-1
  Syntax, 3-8
File
  Actual name, 3-8
  Declaring, 4-31
  Filenames, 4-20
  Header Array, 3-8, 4-21
  Information, 2-1
  Logical name, 3-8
  LU decomposition, 5-6
  Reading, 3-9
  Text, 3-8, 4-21

  Writing, 3-10
First Stage, 2-1
FORMULA
  Statement, 4-1
  Syntax, 3-11
Formula
  Active index, 4-13
  Always, 3-11
  Assign values, 4-33
  Complicated expression, 4-34
  Divide by zero, 4-29
  Division in, 4-12
  Expressions in, 4-7
  Initial, 3-11
  Model checking, 6-1
  Order in Input file, 4-31
  Order of, 4-33
  System-initiated, 2-14
  Values in equations, 4-34
  Zero denominator, 3-15
  Zerodivide default, 4-30
FORMULA & EQUATION, 3-13
FORMULA(INITIAL)
  Restriction, 4-25
Free form, 3-2, 4-2
Functions, 4-9

GEMPACK
  On a PC, 5-26
GEMPIE Print file, 4-31
GEMSIM, 4-20, 4-33, 5-1
  Actions, 5-24
  Equations file, 5-6
GEMSIM Statement file, 5-22
GEMSIM Table file, 5-22
Gragg's method
  Graphical explanation, 5-1
  Number of passes, 5-3
  Unsuitable in some cases, 5-7

Harwell subroutine, 5-4, 5-11
Header
  Length of, 4-3
Header Array file, 3-8, 4-21, 4-23
Help
  Options, 5-11

ID01, 4-9
ID0V, 4-9
IF, 2-14, 4-11
  Syntax, 4-11
Ill-conditioned matrix, 5-7, 5-13
Inactive index, 4-14
INCLUDE file, 2-17
Increasing parameters, 2-17, 5-25
Index
  Active, 4-13
  ALL, 4-6
  Coefficient, 3-6
  Inactive, 4-13
  Range of, 4-5
  SUM, 4-8
  Variable, 3-7
Index name, 4-3
  Length of, 4-3
Index offset, 2-12
Indices
  Absorbed variable, 2-16
  As Arguments, 4-5
  Examples, 4-13
  In declarations, 4-5
  In equations and formulas, 4-13
  Maximum number
    Coefficients, 3-6
    Variables, 3-7
  Order of, 4-32 to 4-33
Information, 4-3
  Labelling, 4-5
Information file, 2-1
  Memory, 5-28
Initial FORMULA, 3-11
Initial formula
  Partial, 4-25
Input statements, 4-3, 4-5, 4-19
  General syntax, 4-1
  Order of, 4-31
Integer coefficients, 3-6, 4-14, 7-5
Integer constant, 4-13
Intermediate data file, 5-37
Intermediate extra data file, 4-25, 5-38
Intermediate updated file, 5-28
Intertemporal, 3-4
  Example sets, 7-2
Intertemporal element stem, 7-3
Intertemporal model, 2-12, 7-1
Intertemporal set
  Elements, 7-3
  Meaning, 7-1
Intertemporally defined, 7-3
Iterative refinement, 5-7, 5-13
  Low memory, 5-27

Keyword, 3-1, 4-1 to 4-2

Labelling Information, 4-5

Large models, 5-26
Last Stage, 2-1
Levels equation, 3-12
  Meaning of, 4-12
Levels variable, 3-6, 4-16
Levels variables
  Zero, 5-3
Linear equation, 3-12
  Meaning of, 4-12
Linear variable
  Meaning of, 4-12
Linearization Rules, A-1, A-3
List of TABLO statements, 3-1
Logarithmic function, 4-9
Logical filename, 4-3, 4-20
  Length of, 4-3
Logical operators, 4-10
Lower case, 3-2, 4-2
LU decomposition, 5-3
LU file, 5-6

Mapping
  Between sets, 8-3
Matrix
  Singular, 5-7
Memory, 5-26
  Option selection, 5-28
  Required, 5-28
Memory sharing, 5-27
Midpoint method
  Graphical explanation, 5-1
  Number of passes, 5-3
  Unsuitable in some cases, 5-7
Model parameter
  see Parameter
Model verification, 6-1
Modified midpoint method, 5-3
Multi-step solution, 5-1

Names
  User-defined, 4-3
Non_parameter
  Coefficient, 3-6, 4-28
Nonlinear, 5-3, 5-20
Number of steps, 5-3
Numerically singular, 5-7, 5-31

Omitting variables, 2-16
Operation, 4-7
Options
  Basic, 2-1
  GEMPIE
    RPO, 4-31

GEMSIM and TABLO-generated program, 5-11
  CPU, 5-19
  D1C, 5-18
  DDC, 5-18
  DOI, 5-19
  DPL, 5-18
  DPN, 5-18
  DPW, 5-18
  DTO, 5-17 to 5-18
  DWS, 4-15, 5-18, 6-1
  EAA, 5-19
  IZ1, 5-11, 5-32
  KZ2, 5-13, 5-32
  NDS, 5-15
  NEL, 5-19
  NEQ, 5-15
  NIR, 5-13
  NRP, 5-13
  NSC, 5-15
  NSE, 5-15 to 5-16
  NSM, 5-15
  NUD, 5-15
  NWE, 5-13
  NWR, 5-15
  RQF, 5-14
  SSI, 5-20
  SVX, 5-14
  TWC, 5-18
GEMSIM and TABLO-generated programs
  Help, 5-11
Help, 5-11
TABLO, 2-1
  ACD, 2-9, 5-8 to 5-9
  Menu, 2-2
  SCO, 2-3
TABLO Code, 2-3, 5-23
  ACC, 5-29
  CDM, 5-25, 5-35 to 5-36
  CIN, 5-30
  DMS, 5-25
  EAA, 5-32
  LIR, 5-26
  LMC, 5-26
  Low memory, 5-26
  LRP, 5-26
  NDS, 5-24
  NEQ, 5-24
  NMS, 5-24
  NRZ, 5-25
  NWR, 5-24
  OCS, 5-29
  PGS, 5-23

SCS, 5-16 to 5-17, 5-26
SMD, 5-26
SPL, 5-25
UCS, 5-26
WFP, 5-23
TABLO-generated program
Menu, 5-10
NAX, 5-22
Order of READs, FORMULAs, EQUATIONs, 4-33
Oscillating results, 5-7

Parameter
Code, 5-30
Coefficient, 3-6
Model
Definition in GEMPACK, 4-35
Parameter values
Decreasing, 5-36
Increasing, 5-34
TABLO, 2-17
TABLO-generated programs, 5-25
Partial initialisation, 4-26
Partial read, 4-25
Partial read,write, 4-24
Percent change
Variable, 3-7
Percentage change
Large negative, 5-3
Rules, A-1
Pivots, 5-4
Print file, 4-31 to 4-32
Variables, 4-31
Product Update, 3-14
Programs
SAGEM, 4-31, 6-2
SUMEQ, 6-2

Qualifier, 3-1
Default values
see Default statements
Summary, 3-18
Warning, 4-1
Quantifier, 4-6
Quantifier list, 4-6

READ
Statement, 4-1
Syntax, 3-9
Read
Data into coefficients, 4-23
Example, 4-23
Filename, 4-20
Header Array file, 4-21
Order of, 4-31, 4-33
Partial, 4-24
Values in equations, 4-34
Real coefficient, 3-6
Real constant, 3-15, 4-3, 4-13
Referenced GEMPACK documents
GPD-3, 4-21
Reserved characters, 4-3 to 4-4
Reserved words, 4-4
Reusing pivots, 5-3, 5-13, 5-27
Row order, 3-8, 4-22, 5-18

SAGEM
Equations file, 5-6
Scalar, 3-15
Semantic check, 4-25
Semantic description, 4-1
Semantic error, 2-1
Removing, 2-4
Semantic errors, 2-4
Removing, 2-4
Semantic problem, 4-13
Separator
Spreadsheet, 3-8
SET
Statement, 4-1
Syntax, 3-3
Set
Associated data, 4-23
Declaring, 4-31
Index range, 4-5
Intertemporal, 3-4
Name of, 4-3
Set element name
As argument, 4-5
Length of, 4-3
Reading, 4-21
Set elements
Arguments, 4-5
Declaring, 4-31
In subsets, 4-19
Order of, 4-23
Syntax, 3-3
Set name, 4-3
Length of, 4-3
Set size
Run-time, 4-19, 7-2
Set-up file, 5-26
Sets
Intertemporal, 4-20, 7-1
Shocks, 5-20
Simulations
Checking models, 6-1
Results, 4-31

Singular matrix, 5-30
Solution accuracy, 5-7
Solution time, 5-3 to 5-4
  Existing files, 5-6
  Subintervals, 5-20
  Total, 5-5
Sparse matrices, 5-3, 5-11
Speed of solution, 5-3 to 5-4
  Code options, 5-26
  Existing files, 5-6
  Iterative refinement, 5-13
  Subintervals, 5-20
  Total time, 5-5
  Zero coefficients, 5-13
Splitting models, 5-26
Splitting variables, 2-14
Spreadsheet, 3-8
Square root function, 4-9
Stages of TABLO
  Check, 2-1
  Code, 2-1
  Condense, 2-1
  First, 2-1
  Last, 2-1
Steps
  Number, 5-3
Steps,passes, 5-3
Strong Comment Markers, 4-2
Structurally singular, 5-31
Subintervals, 5-20
Submatrix Data file, 5-26
Submodel, 5-26
SUBSET
  Statement, 4-1
  Syntax, 3-5
Subset
  By elements, 4-19
  By numbers, 4-19
  Declaring, 4-31
  Index range, 4-5, 4-19
  Input of elements, 4-19
Substitution, 2-10, 2-13 to 2-14, 4-34
  Looking ahead to, 2-13
    Example, 2-13 to 2-14
  Order of, 2-12
  Requirements, 2-12
Suffixes
  For updated data, 5-38
SUM, 4-7
  Syntax, 4-7
SUM index, 4-13
Syntax description, 3-1 to 3-2, 4-1, 4-3
Syntax error, 2-1
Syntax errors, 2-4
  Removing, 2-4
System-initiated, 2-14

Tableau, 4-31 to 4-32, 6-2
TABLO
  Basic options, 2-1
  Parameter values, 2-17
  Running, 2-1
  Stages, 2-1
TABLO Code Options
  Menu, 2-3
TABLO Input file
  Actions, 5-24
  Data manipulation, 3-12, 4-36
  Displays,writes, 6-1
  Equation order, 4-32, 4-34
  Formula,read order, 4-33
  Index order, 4-32
  Input statements, 4-1
  No equations, 3-12, 4-36
  Statement order, 4-31
  Substitution, 2-13
  Variable order, 4-31
  Zerodivide, 4-30
TABLO input language
  Free form, 3-2, 4-2
TABLO language
  Examples, 8-1
TABLO Options
  Menu, 2-2
TABLO options
  see Options
TABLO Syntax, 3-1
TABLO-generated program options, 5-10
TABLO-generated programs, 4-20, 4-33, 5-1
  Actions, 5-24
  Advice re memory, 5-28
  Equations file, 5-6
  Parameter values, 5-25
Terminal
  Read, 3-9
  Write, 3-10
Terminal data file, 5-37
Terminal writes, 5-18
Text file, 3-8, 4-21
Time
  CPU, 5-19
Time periods
  Accumulating, 8-1

UPDATE
  Statement, 4-1
  Syntax, 3-14
Update, 4-26
  Change, 3-14
  Example, C-1
  Expressions in, 4-7
  Order of, 4-33
  Product, 3-14
  Semantics, 4-28
  Which type, 4-27
UPDATE(CHANGE), 4-27
Updated data files, 5-37
Upper case, 3-2, 4-2

VARIABLE, 2-13
  Statement, 4-1
  Syntax, 3-7
Variable
  Active index, 4-13
  Arguments, 4-5
  Change, 3-7
  Component order, 4-32
  Declaring, 4-31
  Dimension, 3-7
  Division by, 4-12
  Example, 4-32
  Examples of arguments, 4-5
  Levels, 3-7
  Linear, 3-7
  Lower case, 4-2
  Order on Print file, 4-31
  Percentage change, 3-7
  Substitution, 4-34
  Use in equations, 4-12
Variable components, 2-13
  Order of, 4-32
Variable name, 4-3
  Length of, 4-3
Variable order on Print file, 4-31
Variables
  SAGEM, 4-31

Warnings
  Ill-conditioned, 5-13
Weakly stable, 5-8
WRITE
  Statement, 4-1
  Syntax, 3-10
Write
  All steps, 5-18
  Checking models, 6-1
  Header Array file, 4-21
  Partial, 4-24

XAC-retained variables, 5-14

Zero coefficients, 5-11
ZERODIVIDE
  Statement, 4-1
  Syntax, 3-15
Zerodivide
  Default, 4-29 to 4-30
  Real constant, 4-3
  Reports, 4-30
  Use of, 4-29