GEMPACK USER DOCUMENTATION

Release 7.0

GPD-2 TABLO Reference

TABLO Reference

GEMPACK Document No. GPD-2

W. J. Harrison

K. R. Pearson

Centre of Policy Studies and Impact Project
Monash University, Melbourne, Australia

Third edition
October 2000

© Copyright 1985-2000. The Impact Project and KPSOFT.

Third edition
October 2000

ISSN 1030-2514

ISBN 0 7326 1521 6

This is part of the documentation of the GEMPACK Software System for solving large economic models, developed by the IMPACT Project, Monash University, Clayton Vic 3800, 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.

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 reference to the TABLO Input language and a User's Guide to the program TABLO. It complements GEMPACK document GPD-3 *Simulation Reference: GEMSIM, TABLO-generated Programs and SAGEM* which is a guide to carrying out simulations on a model using GEMSIM, TABLO-generated programs or SAGEM. We assume that readers are familiar with the first GEMPACK document GPD-1 *An Introduction to GEMPACK*.

This document GPD-2 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),
- 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 : TABLO Reference

Audience : CGE Modellers

Identifier : GPD-2

History :	Date	Author(s)	Comment
	April 1993	Jill Harrison and Ken Pearson	First Edition (Release 5.0)
		[Title of first edition was "User's Guide to TABLO and TABLO-generated Programs"]	
	April 1994	Jill Harrison and Ken Pearson	Second Edition (Release 5.1)
		[Title of second edition was "User's Guide to TABLO, GEMSIM and TABLO-generated Programs"]	
	October 2000	Jill Harrison and Ken Pearson	Third Edition (Release 7.0)
		[Title of third edition is "TABLO Reference"]	

Table of Contents

1. INTRODUCTION 1-1

1.1 Running TABLO on an Existing Model 1-2

1.2 Adding Equations to a Model 1-2

1.3 Writing a TABLO Input File for a New Model 1-3

1.4 Modern Ways of Writing TABLO Input Files (on a PC) 1-3
1.4.1 Using TABmate 1-3
1.4.2 Using ViewHAR to Write TABLO Code for Data Manipulation 1-3

1.5 Condensing a Large Model 1-4

2. ADDITIONAL INFORMATION ABOUT RUNNING TABLO 2-5

2.1 TABLO Options 2-5
2.1.1 TABLO Input File Written on the Auxiliary Files 2-6
2.1.2 TABLO File and TABLO STI File Stored on Solution File 2-7
2.1.3 Specialised Check Options 2-7
2.1.4 Doing Condensation or Going to Code Generation 2-8
2.1.5 TABLO Code Options 2-8
2.1.6 Identifying and Correcting Syntax and Semantic Errors 2-8

2.2 TABLO Linearizes Levels Equations Automatically 2-9
2.2.1 Change or Percentage-change Associated Linear Variables 2-10
2.2.2 How Levels Statements are Converted 2-10
2.2.3 Linearizing Levels Equations 2-11
2.2.4 Linearizing a Sum 2-13
2.2.5 Algorithm Used by TABLO 2-13

2.3 More Details About Condensation and Substituting Variables 2-15
2.3.1 Looking Ahead to Substitution when Creating TABLO Input Files 2-17
2.3.2 System-initiated Formulas and Backsolves 2-18
2.3.3 Absorption 2-19

2.4 Memory Management for Fortran 77 TABLO 2-20

3. BASIC SYNTAX DESCRIPTION 3-21

3.1 SET 3-23
3.1.1 Set Unions, Intersections and Complements 3-25
3.1.2 Data-dependent Sets 3-25

3.2 SUBSET 3-26

3.3 COEFFICIENT 3-27

3.4 VARIABLE 3-28

3.5 FILE 3-30

3.6 READ 3-31

3.7 WRITE 3-32

3.8 FORMULA 3-34

3.9 EQUATION 3-35
3.9.1 "EQUATION(NONE);" Statement 3-35
3.9.2 FORMULA & EQUATION 3-36

3.10 UPDATE 3-37

3.11 ZERODIVIDE 3-38

3.12 DISPLAY 3-39

3.13 MAPPING 3-40
3.13.1 Formulas for Mappings, and Reading and Writing Mappings 3-40

3.14 ASSERTION 3-41

3.15 TRANSFER 3-42

3.16 Setting Default Values of Qualifiers 3-43

3.17 TABLO Statement Qualifiers - A Summary 3-44
3.17.1 Spaces and Qualifier Syntax 3-45

4. SYNTAX AND SEMANTIC DETAILS 4-47

4.1 General Notes on the TABLO Syntax and Semantics 4-47
4.1.1 TABLO Statements 4-47
4.1.2 Lines of the TABLO Input file 4-48
4.1.3 Upper and Lower Case 4-48
4.1.4 Comments 4-48
4.1.5 Strong Comment Markers 4-48
4.1.6 Reserved (special) Characters 4-48

4.2 User Defined Input 4-49
4.2.1 Names 4-49
4.2.2 Labelling Information (Text between hashes #) 4-50
4.2.3 Arguments – Indices, Set Elements, Index Offsets and Index Expressions 4-51

4.3 Quantifier lists 4-53

4.4 Expressions Used in Equations, Formulas and Updates 4-53
4.4.1 Operations Used in Expressions 4-53
4.4.2 Sums over Sets in Expressions 4-54
4.4.3 Brackets in Expressions 4-54
4.4.4 Functions 4-55
4.4.5 Conditional Quantifiers and SUMs 4-57
4.4.6 Conditional Expressions 4-58
4.4.7 Linear Variables in Expressions 4-59
4.4.8 Constants in Expressions 4-60
4.4.9 Indices in Expressions 4-60
4.4.10 Index-Expression Conditions 4-61

4.5 Coefficients and Levels Variables 4-63
4.5.1 Coefficents – What are they ? 4-63

4.5.2 Model Parameters 4-63
4.5.3 Integer Coefficients in Expressions and Elsewhere 4-64
4.5.4 Where Coefficients and Levels Variables Can Occur 4-65
4.5.5 Reporting Levels Values when Carrying Out Simulations 4-68
4.5.6 How Do You See the Pre-simulation and Post-simulation Levels Values? 4-69
4.5.7 Specifying Acceptable Range of Coefficients Read or Updated 4-70

4.6 Sets 4-71
4.6.1 Set Size and Set Elements 4-71
4.6.2 Not Specifying Maximum Size of SETs 4-72
4.6.3 Set Unions and Intersections 4-72
4.6.4 Set Complements 4-73
4.6.5 Sets Whose Elements Depend on Data 4-74
4.6.6 Writing the Elements of One Set 4-74
4.6.7 Writing the Elements of All (or Many) Sets 4-75
4.6.8 Empty Sets 4-75
4.6.9 Reading Set Elements from a File 4-76

4.7 Subsets 4-78

4.8 Mappings Between Sets 4-79
4.8.1 Defining Set Mapping Values 4-80
4.8.2 Checking Values of a Mapping 4-82
4.8.3 Insisting That a Set Mapping be Onto 4-82
4.8.4 Using Set Mappings 4-83
4.8.5 Set Mappings Can Only Be Used in Index Expressions 4-83
4.8.6 Two or More Set Mappings in an Index Expression 4-83
4.8.7 Set Mappings in Arguments 4-84
4.8.8 Other Semantics for Mappings 4-84
4.8.9 Writing the Values of Set Mappings 4-85
4.8.10 Reading Part of a Set Mapping BY_ELEMENTS 4-85
4.8.11 Mapping Values Can be Given by Values of Other Mapping or Index 4-85
4.8.12 Writing a Set Mapping to a Text File 4-86
4.8.13 Writing a Set Mapping as Character Data 4-87
4.8.14 Long Name when a Set Mapping is Written to a Header Array File 4-87
4.8.15 ViewHAR and Set Mappings 4-87

4.9 Files 4-88
4.9.1 Text Files 4-88

4.10 Reads, Writes and Displays 4-90
4.10.1 How Data is Associated With Coefficients 4-90
4.10.2 Partial Reads, Writes and Displays 4-90
4.10.3 FORMULA(INITIAL)s 4-91
4.10.4 Coefficient Initialisation 4-92
4.10.5 Display Files 4-93
4.10.6 Transferring Long Names when Executing Write Statements 4-93

4.11 Updates 4-95
4.11.1 Purpose of Updates 4-95
4.11.2 Which Type of Update? 4-95
4.11.3 What If An Initial Value Is Zero ? 4-96
4.11.4 UPDATE Semantics 4-96
4.11.5 Deriving Update Statements – Example 1 (Sum of Two Flows) 4-97
4.11.6 Deriving Update Statements – Example 2 (Powers of Taxes) 4-98
4.11.7 Writing Updated Values from FORMULA(INITIAL)s 4-98

4.12 Transfer Statements 4-100

4.12.1 XTRANSFER Statements on Command Files 4-101

4.13 Zerodivides 4-102

4.14 Ordering 4-104
4.14.1 Ordering of the Input Statements 4-104
4.14.2 Ordering of Variables 4-104
4.14.3 Ordering of Components of Variables 4-105
4.14.4 Ordering of the Equation Blocks 4-105
4.14.5 Ordering of the Equations Within One Equation Block 4-105
4.14.6 Ordering of Reads, Formulas, Equations and Updates 4-106

4.15 TABLO Input Files with No Equations 4-108

5. CODE OPTIONS WHEN RUNNING TABLO 5-109
5.1.1 Code Options in TABLO Affecting the Possible Actions 5-110
5.1.2 Code Options in TABLO Affecting the Amount of Memory Required 5-111
5.1.3 Other Code Options in TABLO 5-111

5.2 Compiling, Linking and Running TABLO-generated Programs 5-112

6. VERIFYING ECONOMIC MODELS 6-113

6.1 Is Balanced Data Still Balanced After Updating? 6-114
6.1.1 Balance After n-Steps 6-116

7. INTERTEMPORAL MODELS 7-117

7.1 Introduction to Intertemporal Models 7-117

7.2 Intertemporal Sets 7-117
7.2.1 Set Size and Set Elements - Fixed or Determined at Run Time 7-118

7.3 Use an INTEGER Coefficient to Count Years 7-120

7.4 Enhancements to Semantics for Intertemporal Models 7-120

7.5 Recursive Formulas over Intertemporal Sets 7-122

8. LESS OBVIOUS EXAMPLES OF THE TABLO LANGUAGE 8-123

8.1 Flexible Formula for the Size of a Set 8-123

8.2 Adding Across Time Periods in an Intertemporal Model 8-123

8.3 Conditional Functions or Equations 8-123

9. LINEARIZING LEVELS EQUATIONS 9-125

9.1 Differentiation Rules Used by TABLO 9-125

9.2 Linearizing Equations by Hand 9-126
9.2.1 General Procedure 9-126
9.2.2 Rules to Use 9-127

9.2.3 Linearizing Equations in Practice 9-127
9.2.4 Linearizing Using Standard References 9-128

9.3 Linearized EQUATIONs on Information File 9-129

10. NEW TABLO SYNTAX, QUALIFIERS 10-131

10.1 New TABLO Statements and Qualifiers for Release 7.0 10-131

10.2 New TABLO Statement Qualifiers for Release 6.0 10-131

10.3 New TABLO Statement Qualifiers for Release 5.2 10-131

10.4 New TABLO Syntax for Release 6.0 10-131

10.5 New TABLO Syntax for Release 5.2 10-131

11. REFERENCES 11-133

12. GEMPACK DOCUMENTS 12-134

13. INDEX 13-135

CHAPTER 1

1. Introduction

This document GPD-2[1] provides complete user documentation of the TABLO program and the TABLO language. TABLO is the means by which economic models are implemented within GEMPACK, as described in GEMPACK document GPD-1 *An Introduction to GEMPACK* (with which you should be familiar before reading this document GPD-2). A third document GPD-3 *Simulation Reference: GEMSIM, TABLO-generated programs and SAGEM* describes carrying out simulations on economic models after they have been implemented using TABLO.

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.

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 CODE options when running TABLO.

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 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.

Chapter 9 gives rules for linearizing levels equations, and indicates how the linearized equations are shown on the Information file. Chapter 10 lists TABLO statement qualifiers which are new in recent Releases of GEMPACK.

This document describes the version of TABLO in Release 7.0 of GEMPACK. This is version 4.0 (March 2000) of TABLO

[1] 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 Centre of Policy Studies.

1.1 Running TABLO on an Existing Model

GEMPACK document GPD-1 describes the normal ways of running TABLO, and GPD-8 provides additional examples.

* If you are working in WinGEM, see sections 2.5.5 and 2.5.6 of GPD-1.
* If you are using the Command prompt method, see sections 2.6.1 and 2.6.2 of GPD-1.
* See the examples in GPD-8 of running TABLO for the models Stylized Johansen (SJ) and Miniature Orani (MO).
* See the examples in sections 2.3 and 2.4 of GPD-8 where TABLO is run with a Stored-input file in order to condense the models GTAP and ORANIG.

Each of these examples assumes that you wish to run an existing model, or modify slightly an existing model.

If you work on a Windows PC, you will find that TABmate (see section 1.4.1 below and section 2.4 of GPD-4) provides an excellent interface for working with TABLO Input files, especially for identifying and eliminating syntax and semantic errors.

1.2 Adding Equations to a Model

Very few modellers build a model from scratch. Most models are built by modifying an existing model.

A very common task is that of adding some equations to an existing model.

In many cases, the new equations relate to the accounting structure of the model. Then these new equations are most naturally written down and thought of in the levels. In such cases, we strongly encourage you to

add the new equations as levels equations

in the TABLO Input file. We give this advice

even if the all the equations in the model being modified are linearized

in the TABLO Input file. We think that this is the easiest and most transparent and reliable way of adding such equations to a model. In our experience, adding the new equations as linearized equations is more error prone.[2]

A full discussion of this topic, together with several fully worked and detailed examples can be found in the document

Adding Accounting-Related Behaviour to a Model Implemented Using GEMPACK

written by Jill Harrison and Ken Pearson. The current version dated September 1999 is available from the FAQ (Frequently Asked Questions) page on the GEMPACK web site.

We particularly draw your attention to the implementation in SJHT3.TAB discussed in section 2.12 of that paper. Amongst other things, that section makes it clear how you connect the levels variables in the new levels equations being added to the linearized variables in the model being modified.

[2] Unless you are completely steeped in linearized TABLO Input files, it can be tricky to add the new equations correctly. The main problem is not that of linearizing the equations. Rather it is making sure that the formula part of the file is consistent with the equations. When you add levels equations, this extra complexity is not an issue.

1.3 Writing a TABLO Input File for a New Model

If you wish to develop a TABLO Input file for your own model without relying on an existing model, how do you go about it? Chapter 3 of GPD-1 describes how to build a new model using Stylized Johansen as a simple example. To summarise the steps,

1) Write down your equations in ordinary algebra. Choose whether to write a linearized, mixed or levels model. You may need to linearize your equations by hand (see section 9.2).

2) Compile a list of variables used in these equations.

3) Compile a list of data needed for the equations (levels values, parameters, other coefficients).

4) Work out what sets are involved for the equations, variables and data.

5) Type in your TABLO Input file in a text editor and keep checking the syntax using TABLO until you have removed all the syntax and semantic errors. [If you are working on a PC, we recommend that you use TABmate (see section 1.4.1 below) for these steps.] Consult chapters 3 and 4 of this document for the TABLO syntax needed to write your TABLO statements. Use other models as extended examples of how to write TABLO code.

6) If your model is small, you may not need to condense it. In this case run your model within WinGEM or at the Command prompt, compile and link the TABLO-generated program, and run the TABLO-generated program or GEMSIM. Continue on with the simulation process as described in GPD-1 and in more detail in GPD-3.

1.4 Modern Ways of Writing TABLO Input Files (on a PC)

Originally TABLO Input files were written in any text editor (such as Gemedit or vi or Edit). If you have access to a Windows PC, you may wish to use TABmate. Another way to write TABLO code quickly is to use the Code Writing facility in ViewHAR.

1.4.1 Using TABmate

If you do not have a firm preference for any particular editor, we recommend that you use the TABmate editor because it has many in-built functions to assist you including the following.

1) Coloured highlighting of TABLO syntax.

2) Easy TABLO syntax checking allowing you to correct errors in the TAB file.

3) A powerful Gloss feature which displays all parts of the TABLO code where a chosen variable or coefficient is used.

4) You can have multiple files open and cut-and-paste between them in the usual Windows way.

More details about TABmate can be found in section 2.4 of GPD-4.

1.4.2 Using ViewHAR to Write TABLO Code for Data Manipulation

If you have a Header Array data file containing full details, the Set and Element names and the Coefficients names, View HAR can be used to write some of the TABLO code. To test this, run ViewHAR and open the file SJ.DAT from the GEMPACK examples subdirectory. Select

Export | Create TABLO Code

from the main ViewHAR Menu. This will write some text on the Clipboard. Open a Windows Editor such as TABmate or Gemedit and paste this text into a new file. The following text will be created.

Example of TABLO Code Written by ViewHAR

```
Set SECT # description # (s1, s2);
Set FAC # description # (labor, capital);

Coefficient
(All,s,SECT)(All,a,SECT) DVCOMIN(s,a)
 # Intermediate inputs of commodities to industries - dollar values #;
(All,f,FAC)(All,s,SECT) DVFACIN(f,s)
 # Intermediate inputs of primary factors - dollar values #;
(All,s,SECT) DVHOUS(s) # Household use of commodities - dollar values #;

Read
 DVCOMIN from file InFile header "CINP";
 DVFACIN from file InFile header "FINP";
 DVHOUS from file InFile header "HCON";

Update
(All,s,SECT)(All,a,SECT) DVCOMIN(s,a) = 0.0;
(All,f,FAC)(All,s,SECT) DVFACIN(f,s) = 0.0;
(All,s,SECT) DVHOUS(s) = 0.0;

Formula
(All,s,SECT)(All,a,SECT) DVCOMIN(s,a) = 0.0;
(All,f,FAC)(All,s,SECT) DVFACIN(f,s) = 0.0;
(All,s,SECT) DVHOUS(s) = 0.0;

Write
 DVCOMIN to file OutFile header "CINP" longname
 "Intermediate inputs of commodities to industries - dollar values";
 DVFACIN to file OutFile header "FINP" longname
 "Intermediate inputs of primary factors - dollar values";
 DVHOUS to file OutFile header "HCON" longname
 "Household use of commodities - dollar values";
```

ViewHAR has done all the dull part of Code writing, and you can quickly edit the code by writing in appropriate filenames, formulas, updates etc to suit your model. This is very useful if you want to write a data manipulation TABLO Input file.

1.5 Condensing a Large Model

If your model is either too large to run on your computer, or too slow, you should consider condensation. See section 3.9 of GPD-1 for an introduction to condensation, and section 2.3 of this document for fine details. Basically you need to consider:

1) Are there variables you can “omit”, that is, variables that in this set of simulations will be exogenous and not shocked?

2) Which are the endogenous variables with the largest number of components? These are candidates for substituting out or backsolving.

To make a Stored-input file for condensation, run TABLO interactively and select the SIF option (see section 4.3 of GPD-1 for details). Respond to prompts and select [c] after the Check stage to go to condensation (see section 2.1.4 below). Omit one variable, substitute out one variable and backsolve for one variable. There is an example running TABLO interactively to do condensation in section 3.9.1 of GPD-1.

Complete the TABLO run by exiting from condensation and choosing the TABLO Code options that you wish to use. Details about TABLO Code options are in chapter 5.

Using this Stored-input file as a starting point, you can easily add, using your text editor, other variables to omit, substitute or backsolve. When this Stored-input file is complete, run TABLO with this Stored-input file, and continue in the usual way, either compiling, linking and running the TABLO-generated program, or running GEMSIM.

CHAPTER 2

2. Additional Information About Running TABLO

Most of the information about running TABLO is given in chapters 2 and 3 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

This section gives details about Options that are available at the TABLO Check stage[3]. Chapter 5 gives details about options available at the TABLO Code stage[4].

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 (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 2.1.6 below for more details.)
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 4 of GPD-1.

You can choose which stages you carry out using the First Stage options (F1, F2, F3) and the Last Stage options (L1, L2, L3). The default choices for these options 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.

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

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

---

[3] Unless you wish to do something special, you should choose the default options at the Check option screen in TABLO. If you wish to condense using a Stored-input file containing your condensation, you should choose STI.

[4] At the CODE options screen, often the only choice you usually need to make is between WFP and PGS. If you choose WFP, TABLO will write a TABLO-generated program. If you choose PGS, TABLO will prepare output for GEMSIM.

CHECK, you can choose option L1 (Last Stage is CHECK), while choosing L2 means that TABLO stops after CONDENSE.

```
                  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

     RMS Require maximum set sizes to be specified
     NTX Dont store TAB file on Auxiliary file
     NWT Add Newton-correction terms to levels equations
     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 section 2.2.3 below and section 12.6.3 in GPD-3. It affects the way TABLO linearizes any levels equations which, in turn, can affect the numerical properties of multi-step calculations.

Option RMS affects the CHECK stage of TABLO.

RMS Require Maximum Set Sizes

You should select this option if you have a Fortran 77 compiler and plan to write a TABLO-generated program at the CODE stage. When this option is selected the statement

```
SET REG # Regions #
  READ ELEMENTS FROM FILE GTAPSETS Header "H1  " ;
```

would produce a semantic error. If RMS is not selected there would still be an error later in the TABLO run, but then TABLO would not find the error until the CODE stage. See section 4.6.2 for details.

When using Newton's method to solve equations, you can use TABLO to add the *$del_newton* term to all levels equations if you choose option

NWT Add Newton-correction terms to levels equations.

See section 7.5 of GPD-3 for a description of Newton's method.

2.1.1 TABLO Input File Written on the Auxiliary Files

In Release 6.0 and later, the TABLO Input file for your model is written to the Auxiliary Table file (.AXT for a TABLO-generated program, .GST for GEMSIM) by default. It is written in binary format but there are special programs which can convert it back to a text file. You can use the program TEXTBI (see chapter 14 of GPD-4) to recover the Stored-input file from the Auxiliary Table file.

If, for reasons of confidentiality, you do not wish to send out your TABLO Input file as part of the model, you can turn off this default. At the first option screen in TABLO or at the top of your condensation Stored Input file, select the option

NTX Don't store the TAB file on Auxiliary file

then continue as usual with the TABLO Input.

2.1.2 TABLO File and TABLO STI File Stored on Solution File

When the TABLO Input file is stored on the Auxiliary Table (.AXT or .GST) file, this TABLO Input file is transferred from the Auxiliary Table file to the Solution file when you carry out a simulation using GEMSIM or a TABLO-generated program.[5] This means that you can use the program TEXTBI (see chapter 14 in GPD-4) to recover that TABLO Input file from the Solution file. This may assist in understanding simulation results.

Similarly, if you use a Stored-input file to run TABLO, this Stored-input file is transferred to the Auxiliary Table file produced by TABLO (unless TABLO option NTX is selected). This Stored-input file is also transferred to the Solution file when you run a simulation using GEMSIM or a TABLO-generated program. You can use TEXTBI (see chapter 14 of GPD-4) to recover the Stored-input file from the Solution file or from the Auxiliary Table file. [Strictly speaking, the Stored-input file put on the Auxiliary Table file is the one used for the CODE stage of TABLO. If you stopped and restarted TABLO – see section 3.9.4 of GPD-1 – condensation actions may not be on the Stored-input file put on the Auxiliary Table or Solution file.]

This means that it is usually possible to recover the TABLO file and any condensation actions from any Solution file.

Since the original data is stored on the SLC file (see section 8.4 in GPD-3), this means that you can recover everything about a simulation from the Solution and SLC files.

2.1.3 Specialised Check Options

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. You can find out more about these options from this menu. [For example, type **?SM5** to find out about option SM5.]

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

       Semantic Check Options
       ----------------------
       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

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

[5] The features described in this section were introduced with Release 7.0.

2.1.4 Doing Condensation or Going to Code Generation

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] :

  (Enter a carriage return to proceed directly
     to AUTOMATIC CODE GENERATION)
```

If you select [a] (or if you type a carriage return), you will skip condensation and go directly to the Code stage of TABLO.

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.

2.1.5 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 chapter 5.

2.1.6 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.

If you work on a Windows PC, you will find that **TABmate** (see section 1.4.1 above and section 2.4 of GPD-4) provides an excellent interface for working with TABLO Input files, especially for identifying and eliminating syntax and semantic 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.[6]

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.

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 echoes 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 section 9.3 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.

---

[6] 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.

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 9.1, 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 9.1.

   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

$$C \cdot z = 0$$

as indicated in section 2.11.1 of GPD-1.

To illustrate how individual levels equations are linearized, we look at some 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 9.1, we obtain the associated linear equation

   p_Z = p_X + p_Y.

   Alternatively, if we use the change differentiation rules in section 9.1, 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 9.1, we obtain the linearization

c_Z = X*c_Y + Y*c_X.

2.2.4 Linearizing a Sum

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. There are two ways of linearizing it.[7]

(a) Share Form. If we use the percentage-change differentiation rules in section 9.1, we obtain the associated linear equation

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

The terms (X+Y) in the denominator can be replaced by Z to give a familiar form

p_Z = [X/Z]*p_X + [Y/Z]*p_Y.

Here the expressions X/Z and Y/Z are the **shares** of X and Y in the initial values of Z.

(b) Changes Form. Alternatively, if we use the change differentiation rules in section 9.1, we obtain the associated linear equation

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

which can be simplified (by multiplying both sides by 100) to

Z*p_Z = X*p_X + Y*p_Y

Note that here the left-hand side can be interpreted as **100 times the change** in Z. Similarly the terms X*p_X and Y*p_Y on the right-hand side are just 100 times the changes in X and Y respectively. This linearized version of the sum in the original levels equation is thus easy to understand: it says that 100 times the change in Z is equal to 100 times the change in X plus 100 times the change in Y.

TABLO would produce the second (Changes form) when linearizing this equation. [See, for example, the linearized version of the equation Factor_use in section 9.3.]

2.2.5 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.

---

[7] In the ORANI book [Dixon *et al* (1982)], the shares form was used. In the ORANI-G document [Horridge *et al*(1993)]. the changes form was used. TABLO usually produces the changes form when it linearizes a sum.

However, as indicated in section 12.6.4 in GPD-3, 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.)

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[8].

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**.

a) 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**.

b) Consider the equation

   x("c2") = z + w.

   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**).

c) Consider the equation

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

---

[8] 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.

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.**

d) 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.

e) 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).

f) 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.

g) 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.

h) 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 (a) and (b) above violate this.)
2. Every index of the variable in question must be an equation ALL index; a SUM index must not occur. (Example (d) above violates this.)
3. Every equation ALL index must occur as an index in each occurrence of the variable in question. (Example (e) 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 (c) above violates this.)
5. In any one occurrence of the variable in question, all indices must be different; no index can be repeated. (Example (f) above violates this.)
6. In an intertemporal model, no argument of the variable in question can contain an offset. (Example (h) above violates this.)
7. In any two occurrences of the variable in question, the index patterns must be the same. (Example (g) 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 8.3 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 was 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 has not been supported since Release 5.2.[9]

[9] If you need to find out more about absorption, consult section 2.3.3 of the second edition of GPD-2 (April 1994).

2.4 Memory Management for Fortran 77 TABLO

This section is only applicable if you have a Source-code version of GEMPACK and a Fortran 77 compiler. It is not applicable if you have an Executable-image version of GEMPACK.

Compilers available for use with GEMPACK are either Fortran 77 (for example Lahey F77L3) or Fortran 90 (for example Lahey LF90 or LF95). Fortran 90 provides greatly improved memory management compared to Fortran 77.. The difference is in ease of use because **program parameters do not need to be adjusted to accommodate large models when using Fortran 90 GEMPACK programs.**

If you are using a Fortran 77 compiler, when running TABLO, you may receive a message (of the kind described in section 13.2.2 of GPD-3) saying that you should increase the value of one of the parameters in the code.[10]

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, MMNTR, MMNUD, MMNVC, MMNWR
TABLE2	DCSMVC, MMCSTN, MMESTN, MMNCSF, MMNCVU, MMNZD, MMVSTN
CONDENSE	MAXABS, MMACT, DVARHS
TBCDDEC	MMCDLN, MMCESK, MMCLSK, MMLOOP, MMNIIT, MMIITV, DEFMSB, DEFMLS, DEFMNZ, DEFMMS, DEFMSH, DFMCM, DFMCMC
TBGSIM	MMFSK, MMFXSK, MMGSIC, MMGSIN, MMGSOP, MMGSP, MMGSRC, MMGSSK
TBSTACK	MMCNSK, MMINSK, MMLBSK, MMOPSK, MMRSSK, MMVTSK
VOSTACK	MMOESK, MMSLSK, MMVESK, MMVNSK, MXVESK

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 remake the executable image of TABLO. Again, consult your machine-specific documentation to see how to do this.

If you are working on a Windows PC, see the relevant section of GPD-6 for details about this.

[10] 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.

CHAPTER 3

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

[13] MAPPING

[14] ASSERTION

[15] TRANSFER

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 3.16 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. Lines are limited to 80 characters.)

- 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

(1) listing all its elements :

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

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

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

Alternatively, a SET can be defined by

(3) giving its size as an integer :

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

Sets defined as in (3) above and (4) below do not have elements defined at run time. That is, no names are given to the elements of the set when GEMSIM or the TABLO-generated program runs. For this reason, we encourage you not to use these types of sets (3) and (4) in new models. Sets with named elements (such as those defined by (1) and (2) above) are much easier to work with.[11]

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

```
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).

Note that it is not necessary to specify the maximum size of sets in all cases[12] - see section 4.6.2.

---

[11] For example, when the elements of the set have names, simulation results (via GEMPIE or ViewSOL) show these element names.

[12] More precisely, the cases where you can omit the MAXIMUM SIZE are

(i) if you have a Fortran 90 compiler and are writing a TABLO-generated program, or

Of these four above, only (1) can be used for intertemporal sets.

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

```
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 (5) 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. These [] are used in the run-time names of the element (see section 4.6.1).

See chapter 7 about INTERTEMPORAL models, and the definition of intertemporal sets.

Examples

```
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]) ;

SET COM # commodities # SIZE 10 ;      !not recommended !

SET COM # commodities # MAXIMUM SIZE 114  SIZE NCOM ;
!not recommended because set elements unnamed !
```

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

The sets in the last two examples above do not have named elements. As indicated earlier, we discourage the use of such sets. We recommend that you only work with sets whose elements are named either in the TABLO Input file (as in the first example) or at run time (as in the third example above where the element names are read from a Header Array file). For example,

SET COM # commodities # (c1-c10) ;

is a preferred alternative to the second-last example above.

(ii) if you have any compiler or the Executable-image version and are writing output for GEMSIM.

3.1.1 Set Unions, Intersections and Complements

Set unions and intersections can be defined using the following syntax. (See section 4.6.3 for details.)

UNION/INTERSECTION SYNTAX

```
     SET <set_name> [#<information># ]= <setname1> UNION <setname2> ;
  SET <set_name> [#<information># ]= <setname1> INTERSECT <setname2> ;
```

See section 4.6.3 for examples of set unions and intersections.

You can also define the complement of a SET (see section 4.6.4) using:

COMPLEMENT SYNTAX

```
     SET <new_setname> = <bigset> - <smallset> ;
```

See section 4.6.4 for examples of set complements.

3.1.2 Data-dependent Sets

Sets depending on data (see section 4.6.5) can be written:

Data-dependent SET SYNTAX

```
     SET <new_set> = (All, <index>, <old_set> : <condition>) ;
```

Example

```
SET  EXPIND  # export-intensive industries #  =
                  (all,i,IND : EXP(i) > 0.2*SALES(i)) ;
```

This defines the set of export-intensive industries to consist of all industries whose exports [EXP(i)] are at least 20% of total sales [SALES(i)].

See section 4.6.5 for other examples of sets whose elements depend on data.

3.2 SUBSET

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

```
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 :

```
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 4.7 below explains when SUBSET statements are required.

3.3 COEFFICIENT

In a model, this is often the current value of a levels variable. It can occur as a coefficient of a variable in a linearized EQUATION (hence the name "COEFFICIENT"). It can contain values of base data or values derived from base data via a FORMULA (for example, totals or shares). Alternatively, it can be used to store the values of a parameter of the model. See section 4.5.1 for more details.

Coefficients represent real numbers (the default) or integers.

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

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.

The possible COEFFICIENT qualifiers are:

REAL or **INTEGER** (of which REAL is the default).

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 3.16).
> COEFFICIENT(PARAMETER)s are constant throughout any simulation whereas COEFFICIENT(NON_PARAMETER)s may be non-constant - see section 4.5.2 below.

<operator> <real number > where the operator can be GE, GT, LE, or LT

> This form of qualifier is used to declare a **specified range** within which each coefficient must stay. [For example, if X(i) must be positive, you could use the qualifier **(GT 0)** when declaring the Coefficient X.] These qualifiers request the program to check that the coefficient stays within this range at run time. [See section 4.5.7 below and also section 6.4 of GPD-3 for details.]

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 ;
COEFFICIENT(GE 20.0) (all,i,COM)  DVHOUS(i) ;
```

See also section 4.5.3 for a description of "Integer Coefficients in Expressions and Elsewhere", section 4.10.1 for "How Data is Associated With Coefficients", section 4.4.9 for information about indices in coefficients and section 4.5.4 on "Where Coefficients and Levels Variables Can Occur". See section 4.5.7 for details on "Specifying Acceptable Range of Coefficients Read or Updated".

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.

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

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.

The possible VARIABLE qualifiers are:

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

[Both of the above defaults can be reset by use of Default statements (see section 3.16).]

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

ORIG_LEVEL = <coefficient-name> or
ORIG_LEVEL = <real number>....

The ORIG_LEVEL qualifiers apply to linear variables, and are used for reporting levels values corresponding to a linear variable (see section 4.5.5 for details).

<operator> <real number > where the operator can be GE, GT, LE, or LT.

This form of qualifier can be used in when declaring a LEVELS variable. It declares a **specified range** within which the value of the Levels Variable must stay[13]. [For example, if X(i) must be positive, you could use the qualifier **(GT 0)** when declaring the Levels Variable X.] These qualifiers request the program to check that the value of the levels variable stays within this range at run time. [See section 4.5.7 and section 6.4 of GPD-3 for details.]

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).

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) ;
VARIABLE (LEVELS, GE 0) (all,i,SECT) X(i) ;

COEFFICIENT (all,i,IND) Y  ;
VARIABLE (ORIG_LEVEL = Y)  (all,i,IND) yy(i)
                  # yy is Percent change in Y # ;
```

[13] In fact the check is made on the associated Coefficient (see section 2.2.2).

```
COEFFICIENT (all,i,SECT) DVCOM(i) ;
VARIABLE (ORIG_LEVEL=DVCOM)  (All,i,SECT)  p_XCOM(i) ;
VARIABLE (ORIG_LEVEL=1)  (All,i,SECT)  p_PCOM(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 4.5.4 on "Where Coefficients and Levels Variables Can Occur". See section 4.5.5 for details of "Reporting Levels Values when Carrying Out Simulations". See section 4.5.7 for details on "Specifying Acceptable Range of Coefficients Read or Updated".

3.5 FILE

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

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

See section 4.9 below and section 4.1 of GPD-3 for the connection between the logical file names in TABLO Input files and actual data files on your computer. Although this connection can be hard-wired by including the optional "<actual_name>" in the TABLO Input file; we recommend that you do this sparingly for the reasons set out in section 4.9.

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

1. **OLD** or **NEW** or **FOR_UPDATES** (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.)

   The FILE qualifier "**FOR_UPDATES**" is provided to declare a logical file which can have updated values written to it.

2. **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 section 3.1 of GPD-4) or TEXT files. Text files are described in section 4.9.1 of this document and also in chapter 6 of GPD-4.

3. **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 chapter 6 of GPD-4.

4. **GAMS** as in FILE(NEW,GAMS). This is used when converting GEMPACK-style data to GAMS-style data. See section 16.1.2 of GPD-4 for details.

Examples

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

See also section 4.9 on Files, section 4.9.1 on Text files and section 3.4.4 in GPD-1. See section 4.11.7 about FOR_UPDATE files.

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 :

```
READ [ (BY_ELEMENTS) ] <coefficient_name>
                    FROM <location>  [#<information># ]  ;
```

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

```
READ [ (BY_ELEMENTS) ]  [quantifier_list]
 <coefficient_name>  (argument_1,… ,argument_n)
  FROM <location>  [#<information># ]  ;
```

In the above, <location> must be

1. **FILE** <logical_name> **HEADER** "<header_name>"
   if the file is a Header Array file,
2. **FILE** <logical_name>
   if the file is a text file, or
3. **TERMINAL**
   if the data is to be read from the terminal.

The qualifier **(BY_ELEMENTS)** is only allowed if this is an instruction to read character data in defining all or part of the values of a set mapping (see section 3.13 below).

An argument is either an index or the element of a SET, as described in section 4.2.3. Index offsets (see section 4.2.3) are not allowed here.

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 4.10 on "Reads, Writes and Displays" and section 4.10.2 on "Partial Reads, Writes and Displays".

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.)

See section 3.13 for details about reading mappings.

See chapter 4 of GPD-3 for information about the connection between logical files and actual files.

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 :

```
WRITE  <coefficient_name> TO <location> [#<information># ]  ;
```

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

```
WRITE [quantifier_list]
 <coefficient_name>  (argument_1,… ,argument_n)
  TO <location>  [#<information># ]  ;
```

In the above, <location> must be

1. **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 LONGNAME is omitted, the long name written on the file is determined as described in section 4.10.6.)
2. **FILE** <logical_name>
   if the file is a text file, or
3. **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. Index offsets (see section 4.2.3) are not allowed here.

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.)

There are two new qualifiers used for writing the elements of sets to text or Header array files:[14]

```
WRITE (SET) <setname> TO FILE <logical-file> [ HEADER "<header>"]  ;
```

or, to write all sets to a Header Array file (you do not specify Header names):

```
WRITE (ALLSETS) TO FILE <hafile> ;
```

[14] The possibility of writing set elements was added in Release 6.0. [In Release 5.2, WRITE(SET) was allowed only if writing to a GAMS file.]

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 ;

WRITE(SET)  COM  TO  FILE  outfile HEADER  "COMS" ;

FILE(NEW) manysets ;
WRITE (ALLSETS) to FILE  manysets ;
```

See also section 4.10 on "Reads, Writes and Displays" and section 4.10.2 on "Partial Reads, Writes and Displays".

See sections 4.6.6 and 4.6.7 for details on "Writing the Elements of One Set" and "Writing the Elements of All (or Many) Sets".

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

See section 3.13 for details about writing mappings.

See chapter 4 of GPD-3 for information about the connection between logical files and actual files.

3.8 FORMULA

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

```
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 3.16).

FORMULA(INITIAL)s are only calculated during the first step in a multi-step 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 4.10.3 below.)

Index offsets (see sections 4.2.3) are not allowed in the arguments of the left-hand side of a FORMULA(INITIAL) since they are not allowed in a READ statement (see section 4.10.3)

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

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

See section 3.13 for details about special formulas for mappings.

The qualifier "WRITE UPDATED ..." used for FORMUAL(INITIAL) has the syntax:

```
FORMULA (INITIAL,
   WRITE UPDATED VALUE TO FILE <logical_filename>
     HEADER "<headername>" LONGNAME "<words>")
[quantifier_list]
 <coefficient_name>  (argument_1,… ,argument_n) = expression ;
```

In this case, GEMSIM or the TABLO-generated program will write the updated (ie post-simulation) values of the coefficient to the specified logical file at the specified header with the specified long name. See section 4.11.7 for details.

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 ;

FORMULA (INITIAL,
    WRITE UPDATED VALUE TO FILE upd_prices
                  HEADER "ABCD" LONGNAME "<words>" )
    (all,c,COM) PHOUS(c) = 1 ;
```

3.9 EQUATION

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

```
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 3.16).

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 4.15 below for the associated semantic details (including why we introduced this statement).

If you are using a version of TABLO which has been compiled with a Fortran 90 compiler (this includes the Executable-Image version of GEMPACK), you do not need to include the statement

EQUATION (NONE) ;

in your TABLO Input file. TABLO infers this after it has done the preliminary count of the numbers of various statements (including the number of equations). [However, this statement is still accepted by Fortran 90 TABLOs.]

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.

```
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 conditions on a FORMULA(INITIAL) are quite strict - see section 3.8.

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.

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

The possible UPDATE qualifiers are **PRODUCT** or **CHANGE**[15] 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 ;
```

Point 5 in section 3.5.2 of GPD-1 includes an explanation as to why update statements are necessary for obtaining accurate solutions of the levels equations of a model.

More details can be found in section 3.5.3 of GPD-1 and section 4.11 on "Updates".

[15] 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) :

```
ZERODIVIDE [ (qualifier) ] DEFAULT <coefficient_name> ;
```

or a real constant :

```
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 :

```
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 4.13 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 :

```
DISPLAY  <coefficient_name> # <information> # ;
```

or just a part of the COEFFICIENT can be displayed :

```
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.)

DISPLAYs of both real and integer coefficients are allowed.

Examples

```
DISPLAY  TOTSALES  ;

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

See also section 4.10 on "Reads, Writes and Displays" and section 4.10.2 for information about partial displays.

See section 4.3 of GPD-3 for details about Displays, Display files and Command files.

3.13 MAPPING

This statement is used to define Mappings between sets.[16]

```
MAPPING [ONTO] <set_mapping>  FROM  <set1>  TO  <set2> ;
```

The optional MAPPING qualifier ONTO means that every element in the codomain (<set2>) is mapped to by at least one element in the domain set (<set1>). If the qualifier ONTO is not present, the mapping need not be *onto in the mathematical sense*. [See sections 4.8.2 and 4.8.3 for details.]

Example

```
MAPPING  Producer  from  COM  to  IND  ;
```

Further examples and details are given in section 4.8.

3.13.1 Formulas for Mappings, and Reading and Writing Mappings

Set mapping values can be read in or assigned by formulae and can be written to a file. In each case the syntax is as already described above for Reads, Writes or Formulas. Statements especially relevant to set mappings are:

```
READ (BY_ELEMENTS) [quantifier_list] <set_mapping> FROM FILE …. ;
READ <set_mapping>  FROM FILE … ;

WRITE <set_mapping> TO FILE … ;
WRITE (BY_ELEMENTS) <set_mapping> TO FILE … ;

FORMULA (BY_ELEMENTS) [quantifier_list] <set_mapping> = … ;
```

The qualifier BY_ELEMENTS means the element names are read or written or assigned by formula rather than the position number in the set.

Examples

```
READ (BY_ELEMENTS)  (all,i1,S1)  MAP1(i1)  from file … ;
FORMULA (BY_ELEMENTS)  MAP1("food") = "aggfood" ;
```

Further examples and details are given in section 4.8.

[16] Set Mappings were introduced in Release 5.2.

3.14 ASSERTION

An ASSERTION statement requests the software to check conditions that are expected to hold.[17]

ASSERTION SYNTAX

```
ASSERTION  [<qualifiers>] [ # message # ]
[quantifier_list] <condition> ;
```

Here <condition> is a logical expression, the optional message between hashes '#' is the message that will be shown (at the terminal or in a LOG file) if the condition is not satisfied, and <quantifier-list> (optional) can be one or more quantifiers. If the condition is not satisfied, the run of GEMSIM or the TABLO-generated program is terminated prematurely (that is, just after checking this assertion) and an error message is given.

Allowed qualifiers are **ALWAYS** or **INITIAL** (of which ALWAYS is the default).

With qualifier ALWAYS (which is the default), the assertion is checked at every step of a multi-step calculation. With qualifier INITIAL the assertion is checked only on the first step.

We strongly advise you to **include a message** (between #'s) with each assertion. Otherwise, if an assertion is not satisfied, you will not know which assertion is failing.

If an assertion is not satisfied, the software tells you the element names (or numbers if names are not available) each time it fails.[18] For example, if the assertion

```
ASSERTION # Check no negative DVHOUS values # (all,c,COM) DVHOUS(c) >= 0 ;
```

fails for commodity “wool” you will see the message

```
%% Assertion ‘Check no negative DVHOUS values’ does not hold .
 (quantifier number 1 is ‘wool’)
```

(and once for each other such commodity 'c' where the assertion fails) and then the message

```
Assertion 'Check no negative DVHOUS values' does not hold.
```

You can suppress the testing of assertions, or convert them to warnings, by including appropriate statements in your Command file. Details can be found in section 6.3 of GPD-3.

Examples

```
ASSERTION # Check X3 values not too large # (all,c,COM) X3(c) <= 20 ;
```

```
ASSERTION (INITIAL) # Check X values are not negative  #
     (all,c,COM) (all,i,IND) X(c,i) >= 0 ;
```

[17] Assertions were introduced in Release 5.2.

[18] This is new for Release 6.0. [Release 5.2 software did not show this.]

3.15 TRANSFER

TRANSFER statements can be used for transferring data from an old Header Array file to a new or updated header Array file.[19]

```
TRANSFER <header> FROM FILE <logical-file1> TO FILE <logical-file2> ;

TRANSFER UNREAD FROM FILE <logical-file1> TO FILE <logical-file2> ;

TRANSFER UNWRITTEN FROM FILE <logical-file1> TO FILE <logical-file2> ;

TRANSFER <header> FROM FILE <logical-file1> TO UPDATED FILE ;

TRANSFER UNREAD FROM FILE <logical-file1> TO UPDATED FILE ;
```

Here <header> is a Header Array on the file called <logical-file1> in your TABLO Input file. It will be transferred unchanged to the file called <logical-file2> or to the updated version of <logical-file1>.

Both files must be Header Array files.

If the statement is TRANSFER UNREAD…. , all Headers on file 1 which have not been read in the TABLO Input file are transferred to the new file 2 or to the updated version of file 1.

If the statement is TRANSFER UNWRITTEN…. , all Headers on file 1 which have not been written onto file 2 are transferred to the new file 2.

Note that

Transfer Unwritten from file <logical-file1> to Updated File ; ! not allowed !

is not allowed. [Use "Transfer Unread …" instead.]

Examples

```
Transfer "EXT1" from File iodata to updated file ;
Transfer Unread from File iodata to File out2 ;
```

See section 4.12 for the motivation behind these statements and for further details and examples.

[19] Transfer statements were introduced in Release 7.0.

3.16 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.

```
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, and also 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) ;
FORMULA (DEFAULT = INITIAL) ;
COEFFICIENT (DEFAULT = PARAMETER) ;
VARIABLE (DEFAULT = CHANGE) ;
VARIABLE (DEFAULT = LEVELS) ;
```

The two VARIABLE defaults can both be set simultaneously, so after the VARIABLE defaults were set as in the previous two examples, a "VARIABLE X ;" statement without qualifiers would define a LEVELS variable X with an associated CHANGE differential c_X.

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.

Note that Default statements can be put anywhere in the TABLO Input file. For example, if you have a group of levels equations followed by a group of linearized equations, you can put

`EQUATION( Default = Levels) ;` before the group of levels equations and then

`EQUATION( Default = Linear) ;` before the group of linearized equations.

3.17 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 3.16 above.

```
      Set qualifiers

INTERTEMPORAL or NON_INTERTEMPORAL   NON_INTERTEMPORAL is the default

      Subset qualifiers

BY_ELEMENTS or BY_NUMBERS            BY_ELEMENTS is the default

      Coefficient qualifiers

REAL or INTEGER                      REAL is the default

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

<operator> <real_number>           where operator is GE,GT,LE or LT
                                      (see section 4.5.7)

      Variable qualifiers

PERCENT_CHANGE or CHANGE           * PERCENT_CHANGE is the default

LINEAR or LEVELS                   * LINEAR is the default

ORIG_LEVEL = <coefficient> or <real>  (for linear variables only – see
                                              section 4.5.5)

<operator> <real_number>           where operator is GE,GT,LE or LT
                                      (see section 4.5.7)

      File qualifiers

HEADER or TEXT or GAMS               HEADER is the default

OLD or NEW or FOR_UPDATES            OLD is the default

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

      Read qualifiers

BY_ELEMENTS               (for reading character data to a set mapping)

      Write qualifiers

SET or ALLSETS                           (for writing sets)

      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

WRITE UPDATED ...                     (see section 3.8)
```

Equation qualifiers	
LINEAR or LEVELS	* LINEAR is the default
NONE	Special statement - see section 3.9.1
Update qualifiers	
PRODUCT or CHANGE	PRODUCT is the default
Zerodivide qualifiers	
ZERO_BY_ZERO or NONZERO_BY_ZERO	ZERO_BY_ZERO is the default
Mapping qualifiers	
ONTO	This is not the default
Assertion qualifiers	
ALWAYS or INITIAL	ALWAYS is the default

3.17.1 Spaces and Qualifier Syntax

In TABLO Input files, a space after the keyword before a qualifier is not necessary. For example, either of the following is allowed.

```
File  (New)  output   # Summary output # ;
File(New)  output   # Summary output # ;
```

But, in "extra" TABLO-like statements on Command files (see section 6.6 of GPD-3), at least one space is required after the keyword before the qualifier. Thus, for example,

```
Xfile  (New)  output   # Summary output # ;
```

is allowed but

```
Xfile(New)  output   # Summary output # ;
```

will result in an error.

CHAPTER 4

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 TABLO Statements

- A TABLO Input file consists of a collection of separate TABLO **Statements**.

- Each input statement must usually begin with its **keyword**

  (**SET**, **SUBSET**,**COEFFICIENT**, **VARIABLE**, **FILE**, **READ**, **WRITE**, **FORMULA**, **EQUATION**, **UPDATE**, **DISPLAY**, **ZERODIVIDE, MAPPING, ASSERTION** or **TRANSFER**)

  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 Lines of the TABLO Input file

- Input is in free format. Multiple spaces, tabs and new lines are ignored by TABLO.
- Lines are limited to 80 characters. [Any more will result in an error.]

4.1.3 Upper and Lower Case

- 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).

4.1.4 Comments

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

4.1.5 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   !
```

4.1.6 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 RANDOM NORMAL CUMNORMAL	(functions)
$POS	(special function)

- Actual file names can include any (English) characters legal on your computer (except for the reserved characters '**;**', '**#**' or '**!**'). See section 4.9 of GPD-1 for details about allowed file names on different computers. [See also section 4.9 where we recommend that you use actual file names sparingly in FILE statements.]

- **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 ('#') is **labelling information**. It must be contained on a single input line in the TABLO Input file.

- We recommend that you include labelling information wherever possible in your TABLO Input file. This labelling information is used by programs other than TABLO (see below). These labels make the output more intelligible to yourself and others using your model.
- VARIABLE and SET labelling information appears when GEMPIE and ViewSOL report simulation results.
- VARIABLE, EQUATION and SET labelling information appears in SUMEQ maps (see section 13.1.1 of GPD-4).
- 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.
- When a Coefficient is written to a Header Array file, the labelling information for the Coefficient may be used in the long name. See section 4.10.6 for details.
- FILE labelling information is used in RunGEM's Model/Data page.
- ASSERTION labelling information is used when the assertion fails to indicate which assertion has failed.
- At present, labelling information on READ or WRITE statements is not used.

4.2.3 Arguments – Indices, Set Elements, Index Offsets and Index Expressions

- An argument can be an index or the element of a SET. Set element names are enclosed inside double quotes (for example, "wool") when used as arguments.
  Examples X2(i) X2("wool")
- An argument can also be of the form **index + <integer>** or **index - <integer>**if the set in question is an intertemporal set.
  For example, "t+1" as in X(t+1) or "t-1" as in X(t-1).
  Here the "+/-<integer>" part is called the **index offset**.
- Arguments can also involve set mappings. For example, in X2(AGGCOMTOCOM(aggc)) the argument is AGGCOMTOCOM(aggc) where AGGCOMTOCOM is a mapping (from the set AGGCOM to the set COM) and aggc is an index ranging over the set AGGCOM.
- Most generally, an argument can be any **index expression**.
  - The simplest index expressions are those consisting of a single index or a single element name inside double quotes, or an index followed by an index offset[20].
    Examples. i "wool" t+1
  - A set mapping can be applied to any index expression to form another index expression. When the set involved is an intertemporal one, an index offset can be added.
    Examples. MAP1(i) MAP1(MAP2(c)) MAP1(t)+2 MAP1("t2")-3
    MAP2(MAP1(t)+2)-3
- 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 elements from the relevant set (enclosed in quotes) or indeed, any appropriate index expression. Examples of coefficients followed by arguments are
  X3(i,j) INTINP(i,"wool") X5(MAP1(i), t-2)
- 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.
- If an argument involves an index offset, usually the index in question must range over a subset of the relevant set. For example, if Coefficient X2 is declared via

  ```
  COEFFICIENT  (ALL,i,S) X2(i) ;
  ```

  then for the occurrence X2(t+1) to be allowed, the set S must be an intertemporal set and usually the index t must be ranging over a subset of S. However, this requirement can be relaxed in some circumstances (as explained in detail in section 7.4 below).

---

[20] An index offset is only allowed when the index in question is ranging over an intertemporal set.

- 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 – see section 4.6.1), 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 the above restriction about the relevant set having fixed element names 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 ;
```

- When an argument involves one or more set mappings, a little care is required to check that the argument is in the appropriate set. Details can be found in section 4.8.7.

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.)

See section 4.4.5 for more details about conditions in quantifiers.

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_name>,<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"). See section 4.4.5 for more details about conditions in SUMs.

- 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[21]

Functions

```
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
RANDOM      RANDOM(a,b) gives a random number between a and b
NORMAL      NORMAL(x) traces out a normal distribution with
               mean 0 and standard deviation 1
CUMNORMAL CUMNORMAL(x) is the probability that a normally
               distributed variable with mean 0 and standard
               deviation 1 is less than or equal to x
$POS        $POS(<index>)   position in set function
            $POS(<index>,<set2>)
            $POS(<element>,<set>)
```

Below we give details about some of these.

- 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.
- In the table above, **ID01** stands for "IDentity function except that **0** (zero) maps to **1** (one)".
  ID0V stands for "IDentity 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'.

  These functions **ID01** and **ID0V** can be used to guard against division by zero in equations or formulas. For example, consider the equation **E_p1lab_o** in the ORANI-G or ORANI-F models usually supplied with GEMPACK (see chapter 1 of GPD-8). This equation is written

  (all,i,IND) [TINY+V1LAB_O(i)]*p1lab_o(i) = sum{o,OCC, V1LAB(i,o)*p1lab(i,o) };

  Here V1LAB_O(i) = SUM(o,OCC,V1LAB(i,o) and the TINY on the left-hand side is to guard against the possibility that, for some industry i, V1LAB(i,o) is zero for occupations o and hence V1LAB_O(i) is zero. [TINY has been defined as a Coefficient and given a very small value (such as 0.00000001).] Without the TINY, the coefficient of variable p1lab_o(i) on the left-hand side

[21] Functions RANDOM, NORMAL and CUMNORMAL were introduced in Release 7.0. Their introduction was suggested by Mark Horridge.

would be zero and it would not be possible to use this equation to solve for the usually endogenous variable p1lab_o. The function ID01 could be used in place of TINY. Then the equation would read

(all,i,IND) **ID01{**V1LAB_O(i)**}** * p1lab_o(i) = sum{o,OCC, V1LAB(i,o)*p1lab(i,o) };

If V1LAB_O(i) is zero for some industry i, ID01 converts the coefficient on the left-hand side to 1, so that the equation can be used to solve for p1lab_o(i) (which is then equal to zero for this industry i). An advantage of using ID01 rather than TINY is that ID021 only changes the coefficient on the left-hand side when it would otherwise be zero, whereas the use of TINY changes the value of that coefficient in every case (though only by a very small amount).

- The function **RANDOM** can be used to generate random numbers in a specified range. Successive calls to RANDOM(a,b) with the same values of a and b produce numbers which are uniformly distributed between a and b. RANDOM(a,b) is allowed whether a <= b or a > b.

  Each time you run the program, you may wish to get a new sequence of such random numbers, or you may wish to get the same sequence each time. If you wish to get the same sequence each time, you should put the statement

  randomize = no ;

  in your Command file. [The default is to randomize each time, which corresponds to the statement "randomize = yes ;".][22]

  The RANDOM function is available in GEMPACK whenever you are using a Fortran 90 compiler, or if you are using the Lahey F77L3 compiler. However, it may not be available on some Unix machines using a Fortran 77 compiler. [See chapter 13 of GPD-3 for more about compilers.]

- The functions **NORMAL** and **CUMNORMAL** are included to give you access to the normal probability distribution.

  The **NORMAL** function defines the standard normal (bell-shaped) curve with mean 0 and standard distribution 1. Its formula is

  NORMAL(x) = [1.0/SQRT(2*PI)]*EXP(-X*X/2)

  where PI is (approximately) 3.141592.

  For any real X, the value of **CUMNORMAL(X)** is equal to the integral of the above normal curve from negative infinity to X (that is, to the area of the left tail from negative infinity up to X). CUMNORMAL(X) values range from 0 (when X is negative infinity) through 0.5 (when X=0) up to 1 (when X is plus infinity).[23]

  Suppose that you are considering a variable T which is normally distributed with mean 3 and with standard deviation 0.5. You can use the CUMNORMAL function to find the probability that T lies

---

[22] Suppose that the set COM has 100 elements c1 to c100, and consider the formula
(all,c,COM) COEF1(c) = RANDOM(0,1) ;
If you include the statement "randomize = no ;" in your Command file, and if you are using the Lahey compiler LF90, the values of COEF1("c1") and COEF1("c2") will always be 0.56569254 and 0.54562181 respectively. [Similarly for different compilers, though the actual values may be different.]
However, if you do not include the statement "randomize = no ;" in your Command file (or if you put the statement "randomize = yes ;" in), the values for COEF1("c1") and COEF1("c2") will be different each time you run the TABLO-generated program or GEMSIM. [The way the RANDOM function is calculated each time usually depends on the system time clock.]

[23] More about the normal distribution and these functions can be found in many different references including chapter 6 of Press *et al* (1986). The formula for CUMNORMAL(x) in GEMPACK uses a modification (supplied by Mark Horridge) of the approximation EFRCC to ERFC given in section 6.2 of Press *et al* (1986).

between two specified values. For example, suppose you wish to calculate the probability that T lies between 3.6 and 4.0. This is between 1.2 and 2.0 standard deviations away from the mean, so that the probability is

CUMNORMAL(2.0) – CUMNORMAL(1.2) = 0.97725 - 0.88493 = 0.09232.

- The function $POS is used to determine the position number in a set from the index or element name. The function $POS can be used in the following three ways.

  * **$POS(<index>)** indicates the position number of <index> in the set over which this index is ranging.

    For example, suppose that COM is the set (c1-c5). The formula

    ```
    FORMULA (all,c,COM) X(c) = $POS(c) ;
    ```

    puts X("c1") equal to 1, X("c2") = 2, X("c3") = 3, X("c4") = 4 and X("c5") = 5.

  * **$POS(<index>,<set2>)** indicates the position of <index> in the set <set2>. In this case, <index> must be ranging over a set which is a subset of <set2>.

    For example, consider COM as above and suppose that MARCOM is the set (c3,c4) and that MARCOM has been declared as a subset of COM. The formula

    ```
    FORMULA (all,mc,MARCOM) X(mc) = $POS(mc,COM) ;
    ```

    puts X("c3") equal to 3 and X("c4") = 4 (their position numbers in COM). But the formula

    ```
    FORMULA (all,mc,MARCOM) X(mc) = $POS(mc) ;
    ```

    puts X("c3") equal to 1 and X("c4") = 2 (their position numbers in MARCOM).

  * **$POS(<element>,<set>)** is also allowed. This gives the position number of the element <element> in the set <set>.

    For example, with the sets above, $POS("c3",MARCOM) = 1 and $POS("c3",COM) = 3.

  Example from ORANI-F

  In the TABLO Input file for ORANI-F [the file ORANIF.TAB supplied with GEMPACK and documented in Horridge, Parmenter and Pearson (1993)], the Coefficient ORD(y) for "y" in YEARS is no longer needed; every occurrence of ORD(y) could be replaced by

  $POS(y,YEARS)

  [or by $POS(y) whenever index 'y' is ranging over the set YEARS].

  * In fact the first argument of $POS can be an index expression (see section 4.8.5). For example, $POS(COMTOAGGCOM(c)) and $POS(COMTOAGGCOM(c),AGGCOM) are legal if index "c" is ranging over set COM (or over a subset of it) and COMTOAGGCOM is a mapping from the set COM to some other set.

4.4.5 Conditional Quantifiers and SUMs

- Conditions can be specified to restrict SUMs (see section 4.4.2) and ALLs (see section 4.3). 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. Conditionals 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 4.8 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, WRITEs, 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. That is, 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 are possible such as

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

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.

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 8.3 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 3.16). 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 (see section

4.11.4). 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 in 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.[24]

  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.

  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.

[24] 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.

```
                                                !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.

  However exactly the same expression as on the left-hand side is allowed on the right-hand side.[25] For example, the following type of formula is allowed.

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

4.4.10 Index-Expression Conditions

An index expression is an expression built up from indices and set MAPPINGs (see section 4.2.3). For example, COMTOAGGCOM(c) is an index expression.

- **Comparing Indices**

  Conditions for SUMs and IFs can now involve comparisons of indices and index expressions. You can also use index comparisons for conditions in ALLs in the cases where conditions on ALLs are allowed (see section 4.3).

  You can test if an index is equal to **EQ** or not equal to **NE** another index, as in the formula:
  `FORMULA (all,c,COM)(all,i,COM)  X(c,i) = IF( c EQ i , Y(i)) ;`
  The expression c EQ i is a comparison of two indices. The IF condition c EQ i is true if the element names of the set elements c and i are the same.
  This would set X(i,i) = Y(i) and X(c,i) = 0 when c is not equal to i.

  Similarly,
  `FORMULA (all,c,COM) Y(c) = SUM(i,COM : i NE c , X(c,i) ) ;`
  will, for each c in COM, sum all X(c,i) except for X(c,c).

- **Index Expression Comparison**

  Conditions of the form
  <index-expression-1> <comparison-operator> <index-expression-2>
  are allowed with <comparison-operator> replaced by **EQ** (equals) or **NE** (not equal to).

---

[25] This was not allowed before Release 5.0.

The expression "COMTOAGGCOM(c) EQ aggc" in the formula

```
AGGDHOUS(aggc) = SUM(c,COM: COMTOAGGCOM(c) EQ aggc, DHOUS(c) ) ;
```

is an **index-expression comparison.**

If the index expressions are in intertemporal sets, then <comparison-operator> can be replaced by any of:

LE (less than or equal to)
LT (less than)
GE (greater than or equal to)
GT (greater than)
EQ (equal to)
NE (not equal to).

In the general form of the condition above, the set in which <index-expression-1> lies must be either equal to, or else a SUBSET of, the set in which <index-expression-2> lies, or vice versa. [An index by itself lies in the set over which it ranges. When a set MAPPING from <set1> to <set2> is applied to an index ranging over <set1> or a subset of <set1>, the resulting expression is in the set <set2>. See section 4.8.6 for details.]

For example you cannot compare c with i when c belongs to the set of commodities COM and i belongs to the set of industries IND. But using the mapping PRODUCER defined:

```
MAPPING PRODUCER from COM to IND ;
```

you can compare the mapping of c, PRODUCER(c), with industry i as in "PRODUCER(c) EQ i" since PRODUCER(c) ranges over the set of industries IND.

When indices are ranging over intertemporal sets, **index offsets** can be used. For example,

```
MAP1(t+2)-3
```

is legal if index "t" is ranging over intertemporal set S1 and MAP1 is a MAPPING from S1 to an intertemporal set S2.

4.5 Coefficients and Levels Variables

4.5.1 Coefficents – What are they ?

In the linear equation 3x + 4y = 7, the 3 and the 4 are usually called the coefficients. This explains why the name COEFFICIENT is used in GEMPACK. In a linear EQUATION in a TABLO Input file, a typical term is of the form C*x where C is a COEFFICIENT and x is a linear VARIABLE.

However, COEFFICIENTs can occur and be used in other ways in TABLO Input files, as we now describe.

- In a **model** (that is, a TABLO Input file containing EQUATIONs), COEFFICIENTs can be used to hold base data or consequences of the base data (for example, totals or shares). They can also be used to hold the values of model parameters (such as elasticities).

  For example, in the linearized TABLO Input file SJLN.TAB for Stylized Johansen shown in section 3.5.1 of GPD-1, DVHOUS holds base data, DVCOM and BHOUS hold consequences of the base data and ALPHACOM holds the values of model parameters. Of these, BHOUS and ALPHACOM appear as coefficients in the linear EQUATIONs Price_formation and Com_clear respectively.[26]

- In a **data-manipulation TABLO Input file** (that is, a TABLO Input file containing no EQUATIONs), COEFFICIENTs can be used to hold base data or consequences of the base data.

  For example, in the data-manipulation TABLO Input file SJCHK.TAB usually distributed with GEMPACK (see section 1.1 of GPD-8), DVHOUS holds base data, DVCOM and DVCOSTS hold consequences of the base data.

In a model, COEFFICIENTs which are not parameters usually represent the **current value of levels variables**. For example, in Stylized Johansen,

* DVHOUS, DVCOM and BHOUS all represent the current values of what could be thought of as levels variables (namely the dollar values of household consumption, total production of commodities and the share of households in the total demand for commodities).
* In the first step of a multi-step calculation, their values are as in, or as derived from, the base data.
* In subsequent steps, their values change. For example, in the second step, DVHOUS holds the values of household consumption as updated after the changes occurring in the first step and DVCOM and BHOUS values are also consequences of the data updated after the first step. Similarly for other steps.
* Although they are not explicitly included as VARIABLEs in SJLN.TAB, the percentage changes in DVHOUS, DVCOM and BHOUS could be added as linear variables. [The percentage changes in DVHOUS and DVCOM are reported from simulations based on the mixed TABLO Input file SJ.TAB – see section 3.3.2 of GPD-1.]

4.5.2 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 (for example, elasticities). Such coefficients can (and often should) be declared as COEFFICIENT(PARAMETER)s.

---

[26] Here the word “coefficient” is used with the usual, non-technical meaning alluded to in the first sentence of this subsection.

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. [See section 4.5.2.]

A different use of "parameter" in GEMPACK refers to the parameter values in the source code of GEMPACK programs. See sections 13.2.2 and 13.3 of GPD-3 and section 2.4 of this document for further details of this usage.

4.5.3 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 conditions.)
    - ◊ No real numbers (those with a decimal point such as 12.3, as distinct from integers such as 123) can occur on the RHS (except inside SUM or IF conditions).
    - ◊ Division is not allowed (except inside SUM or IF conditions).

  [Inside SUM or IF conditions, arithmetic is done as if all coefficients were real coefficients.]
- 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,[27] and so must be parameters of the model. Hence
    1. 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
    2. 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 (1) 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 3.16.]
    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 6.1.9 of GPD-3) 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.
- Integer coefficients used in setting sizes of sets must be parameters (see sections 3.1 and 4.6.1).

---

[27] 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 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.)

4.5.4 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.

1. Those declared explicitly via COEFFICIENT(NON_PARAMETER) statements or just COEFFICIENT (if the default has not been reset to PARAMETER as in section 3.16). They are allowed 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.
2. 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.
3. 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 (1) above can also be initialised at the first step (and every subsequent step) via a FORMULA(ALWAYS).

At **subsequent steps**,

* 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).
* the value of the COEFFICIENT arising from a levels variable is updated from its associated change or percentage-change variable.
* 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 usually shown on these updated data files. [However, values initialised via a FORMULA(INITIAL) will be shown on the relevant updated data file if the qualifier "WRITE UPDATED VALUE TO … " is included in the FORMULA statement – see section 3.8.]

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	

4.5.5 Reporting Levels Values when Carrying Out Simulations

It is possible to ask GEMSIM or a TABLO-generated program to calculate and report pre- simulation and post-simulation levels values as well as the usual percentage change results.[28] To do this, you may need to add appropriate information in your TABLO Input file to tell TABLO what are the pre-simulation levels values of selected linear variables.

For example, in the linearized TABLO Input file SJLN.TAB for the Stylized Johansen model (see section 3.5.1 of GPD-1), we have included the qualifier "(ORIG_LEVEL=DVCOM)" when declaring the variable p_XCOM, as in

```
Variable (ORIG_LEVEL=DVCOM)  (All,i,SECT)  p_XCOM(i) ;
```

[Here DVCOM(i) is a dollar value for each commodity i. But its original (that is, pre-simulation) value can be thought of as a quantity, where one unit of volume is whatever amount can be purchased for one dollar at pre-simulation prices.]

Also we have included the qualifier (ORIG_LEVEL=1) when declaring the variable p_PCOM, as in

```
Variable  (ORIG_LEVEL=1)  (All,i,SECT)  p_PCOM(i) ;
```

to indicate that the original levels value of the commodity prices are being taken as 1.

When you run a simulation, GEMSIM or the TABLO-generated program will report the pre-simulation, post-simulation levels values of XCOM and PCOM, also the change in them, as well as the percentage change p_XCOM in XCOM and p_PCOM in PCOM, for each commodity.

The syntax for these variable qualifiers is

```
(ORIG_LEVEL=<coefficient-name>)
```

or

```
(ORIG_LEVEL=<real-number>)
```

Qualifiers "ORIG_LEVEL=" are only needed when you define linear variables (percentage changes or changes).

In a levels or mixed model, you do not need such qualifiers when declaring levels variables (since the correspondence is clear). [When you declare a levels variable X, TABLO does not need to be told that the pre-simulation values of the associated linear variable p_X or c_X are given by the coefficient X.] Thus, for example, no ORIG_LEVEL statements are needed in the mixed TABLO Input file SJ.TAB for Stylized Johansen (see section 3.3.2 of GPD-1) in order to get levels values reported when you carry out a simulation.

Note that no indices need to be shown (nor will TABLO allow them to be shown) in a "ORIG_LEVEL=" qualifier. Simply specify the name of the coefficient. TABLO automatically checks that the coefficient and variable have the same numbers of indices and that these indices range over exactly the same sets; a semantic error will be reported otherwise.

The coefficient referred to in a qualifier

```
(ORIG_LEVEL=<coefficient-name>)
```

must be a real coefficient (not an integer one).

To make it easier to add "ORIG_LEVEL=" qualifiers to existing TABLO Input files, we have made an exception to the usual rule that everything must be defined before it is referenced. In the qualifier

```
(ORIG_LEVEL=<coefficient-name>)
```

---

[28] This is true even if your TABLO Input file has only linear variables and linearized equations. Reporting levels results was new in Release 6.0.

the Coefficient in <coefficient-name> is allowed to be declared later in the TABLO Input file. For example, in SJLN.TAB (see section 3.5.1 of GPD-1), it is correct to include the qualifier "(ORIG_LEVEL=DVHOUS)" in the declaration of variable p_XCOM even though coefficient DVHOUS is not declared until several lines later.

Note that the values of <coefficient-name> must, of course, be available (via a Read or Formula). The values needed are just the pre-simulation values - the values are not needed (for this purpose, at least) at subsequent steps. Thus, if you are adding Formulas just to give the values in question (and not also for other purposes), you can make them Formula(Initial)s. For example, in SJLN.TAB it is natural to include the qualifier (ORIG_LEVEL=Y) in the declaration of linear variable p_Y. Coefficient Y (value of household expenditure) was not available in the Release 5.2 version of SJLN.TAB. In the Release 6.0 version we added it via

```
Coefficient  Y  # Total nominal household expenditure # ;
Formula (Initial)  Y = SUM(i, SECT, DVHOUS(i) ) ;
```

[Had we not done so, we could not have included the (ORIG_LEVEL=Y) qualifier when declaring variable p_Y.]

4.5.6 How Do You See the Pre-simulation and Post-simulation Levels Values?

When you run GEMPIE and select the totals solution, you will see the levels values reported unless you select GEMPIE option "**NLV**" (No levels values)[29]. Then, for each component of each variable whose pre-simulation levels values are know (e.g., via an ORIG_VALUE= qualifier), you will see 4 numbers underneath each other, as in, for example,

3.000 (the percent change)

500.0 (the pre-simulation levels value)

515.0 (the post-simulation levels value)

15.0 (the change from pre-simulation to post-simulation)

The linearized result (percentage-change or change) is always first, then below it are the pre-simulation, post-simulation and change results.

Note that levels values are not necessarily available for all variables. [In a linearized TABLO Input file, they are only available for those variables for which you add an "ORIG_LEVEL=" qualifier when you declare the variable.]

If you don't want to include these levels results on a Solution file, you can add the statement

```
levels results = no ;
```

to your Command file.

The current version of ViewSOL can also show any levels results available on the Solution file. These are shown as separate solutions. The abbreviations “Pre”, “Post” and “Chng” are placed before the name of the Solution file (for example, “Pre sjlb”) to indicate the different sorts of results. Note that you can suppress levels results via the Options menu item under the File menu of ViewSOL.

SLTOHT processes levels results if present if you specify the SLTOHT option **SHL** - Show levels results, if available. The default is not to show levels results so that the new SLTOHT is still compatible with old Stored Input files or batch programs written for Release 5.2.

[29] Levels results are not shown if you select GEMPIE option **SNA** (Single solution not across the page) since levels results are only shown when the results go across the page.

4.5.7 Specifying Acceptable Range of Coefficients Read or Updated

Qualifiers such as (GE 0) can be used to specify acceptable ranges of values for coefficients and levels variables (see sections 3.3 and 3.4 above). These can be used to report errors if a simulation takes the values out of the acceptable range. They can also be used in conjunction with user-specified accuracy. See section 6.4 of GPD-3 for details.

4.6 Sets

4.6.1 Set Size and Set Elements

The size of a set, and its elements, can be specified in the TABLO Input file, or may be determined at run time. For example, the set

```
SET  COM  (c1-c10) ;
```

has **fixed size** 10 (that is, the size is specified in the TABLO Input file) whereas the set

```
SET COM  Read Elements from File Setinfo Header "COM" ;
```

has its size determined at run time. In the first of these examples, the element names are fixed in the TABLO Input file whereas in the second example, the element names are defined at run time (when the Setinfo file is read).

In a multi-step calculation, the size and elements of each set are determined during the first step, and do not vary in subsequent steps.[30]

Sets may have named elements. If so, these element names may be **fixed** or only **defined at run time**.

- Element names are said to be **fixed** if they are listed in the TABLO when the set is defined using style (1) in section 3.1.
- Element names are **defined at run time** if the names are not known until they are read when GEMSIM or the TABLO-generated program runs. This is the case with set declarations of types (2) and (5) in section 3.1.
- Sets defined using styles (3) and (4) in section 3.1 do not have named elements. These sets are less useful in models than those with named elements, and accordingly we recommend that you do not use sets without named elements in the future (though we continue to support them for existing models).

If element names are read at run time as in, for example,

```
SET COM  Read Elements from File Setinfo Header "COM" ;
```

[see set declaration numbered (2) in section 3.1], the data at the relevant position of the relevant file must be strings of length no more than 12. [Shorter lengths are ok.] See section 4.6.9 for more details about reading element names for a Header Array file. This is because element names are limited to at most 12 characters (see section 4.2.1 and 4.6.9).

> For example, if you are preparing a header to contain the names of the 20 commodities in your model, the data at the relevant header should be 20 strings of length 12 (or it could be 20 strings of length 8 if all element names were this short).

The elements of intertemporal sets defined using style (5) in section 3.1 are based on the intertemporal element stem. For example, if the set fwdtime is defined via

```
SET (INTERTEMPORAL) fwdtime
     MAXIMUM SIZE 100 (p[0] - p[NINTERVAL-1]) ;
```

and if coefficient NINTERVAL has the value 6 at run time, then the elements of fwdtime are p[0], p[1], p[2], p[3], p[4] and p[5].
See section 7.2.1 for more details about intertemporal sets and elements.

Sets defined as Unions, Intersections (see section 4.6.3), Complements (see section 4.6.4) or depending on data (see section 4.6.5) may have fixed elements or run-time elements. For details, see the relevant sections below.

Note that only fixed element names can appear as arguments in the TABLO Input file (see section 4.2.3).

---

[30] For this reason, if an integer coefficient determines the set size as in (4) in section 3.1, the integer coefficient must be a parameter, that is, must be constant throughout the simulation.

4.6.2 Not Specifying Maximum Size of SETs

In many cases, in TABLO Input files, it may not be necessary to specify the MAXIMUM SIZE of SETs.[31]

For example, in GTAP the statement

```
SET REG # Regions # MAXIMUM SIZE 10
  READ ELEMENTS FROM FILE GTAPSETS Header "H1  " ;
```

can be replaced by

```
SET REG # Regions #
  READ ELEMENTS FROM FILE GTAPSETS Header "H1  " ;
```

This means it is no longer necessary to have different TABLO Input files for different sized versions of a model. [Another example is with the Monash Model; now the 32-sector aggregation used in the Monash Model Course can have the same TABLO Input file as the 113-sector used by CoPS.]

More precisely, the cases where you can omit the MAXIMUM SIZE are

(i) if you have a Fortran 90 compiler and are writing a TABLO-generated program, or

(ii) if you have any compiler and are writing output for GEMSIM.

In these cases, any MAXIMUM SIZEs in the TABLO Input file are ignored by TABLO. [For example, you may say in the TABLO Input file that a set IND has maximum size 10 and then read in, quite happily, a set of size 20 when you run GEMSIM or the Fortran 90 TABLO-generated program.]

However, if you have a Fortran 77 compiler and are writing a TABLO-generated program, you still need to specify MAXIMUM SIZEs of all sets. TABLO checks this when you ask to write a TABLO-generated program in this case.

Because of this, there is the option **RMS** [Require Maximum Set Sizes] for the CHECK stage of TABLO. You should select this option if you have a Fortran 77 compiler and plan to write a TABLO-generated program at the CODE stage. When this option is selected the statement

```
SET REG # Regions #
  READ ELEMENTS FROM FILE GTAPSETS Header "H1  " ;
```

(see above) would produce a semantic error. If RMS is not selected there would still be an error later in the TABLO run, because then TABLO would not find the error until the CODE stage.

4.6.3 Set Unions and Intersections

Unions and intersections of sets are allowed in TABLO Input files.[32] For example, in GTAP, once the sets ENDW_COMM (endowment commodities) and TRAD_COMM (tradeable commodities) are defined, the set DEMD_COMM (demanded commodities) can be defined via the statement

```
Set DEMD_COMM # Demanded Commodities # = ENDW_COMM union TRAD_COMM ;
```

The syntax for set union and intersection is

```
SET <set-name> [#<label-information>#] = <set-name1> UNION <set-name2>;
SET <set-name> [#<label-information>#] = <set-name1> INTERSECT <set-name2>;
```

(see section 3.1.1). Here both <set-name1> and <set-name2> must be previously defined sets whose elements are either fixed or known at run-time (see section 4.6.1).

The elements of the **UNION** are those in <set-name1> followed by those in <set-name2> which are not in <set-name2>.

---

[31] This was new for Release 6.0. Prior to that, MAXIMUM SIZE was always required.

[32] They were introduced in Release 6.0.

The elements of the **INTERSECTion** are those in <set-name1> which are also in <set-name2>.

For example, if

SET1 has elements (e1, e2, e3, e4) and
SET2 has elements (e2, e6, e1), and if

```
SET3 = SET1 UNION SET2 ;
SET4 = SET1 INTERSECT SET2 ;
SET5 = SET2 UNION SET1 ;
```

then

SET3 has elements (e1, e2, e3, e4, e6),
SET4 has elements (e1, e2), and
SET5 has elements (e2, e6, e1, e3, e4).

Note that, although the elements of SET3 and SET5 are the same, their order is different. The order is, of course, important in associating data with indices.

Note also that, with SET1 and SET2 as above, the elements of
SET1 INTERSECT SET2
and
SET2 INTERSECT SET1
are in a different order.

Thus, as far as TABLO is concerned, the order of the sets in a UNION or INTERSECT statement can affect the sets so defined.

Intertemporal sets are not allowed in Set Union or Intersection statements.

Unions and intersections can be used in **XSET extra statements** in Command files (see section 6.6 in GPD-3).

We are grateful to Robert McDougall for suggesting that TABLO allow explicit unions and intersections.

4.6.4 Set Complements

The syntax for set complements is

```
SET  <new-setname>  =  <bigset>  -  <smallset>  ;
```

(see section 3.1.1).[33] Here <bigset> and <smallset> must have already been defined in earlier SET statements, and <smallset> must have been declared as a SUBSET of <bigset>.

This statement means that the elements of <new-setname> are all the elements in <bigset> which are not in <smallset>.

For example, if COM is the set of all commodities and MARCOM is the set of all margins commodities, then the statement

```
SET NONMARCOM = COM - MARCOM ;
```

says that NONMARCOM consists of all commodities which are not in MARCOM. [It is no longer necessary to list or read the elements of NONMARCOM once those of COM and MARCOM have been specified.] .

As well as defining a new set, a set complement statement automatically generates the obvious **SUBSET** statement, namely

```
SUBSET <new-setname> IS SUBSET of <bigset> ;
```

[In the example above, NONMARCOM is automatically a SUBSET of COM.]

---

[33] Set complements were introduced in Release 5.2.

If both <smallset> and <bigset> have **fixed elements** (that is, those defined explicitly in the TABLO Input file), then the complement set <new-setname> has fixed elements.

For example, with the statements

```
SET COM (c1,c2,c3,c4,c5) ;
SET MARCOM (c3, c4) ;
SUBSET MARCOM is subset of COM ;
SET NONMARCOM = COM - MARCOM ;
```

the set NONMARCOM has fixed elements c1,c2,c5.

If <bigset> has run-time elements (for example, read from a file), then <new-setname> also inherits the appropriate run-time elements.

If <smallset> and <bigset> have fixed sizes, so does <new-setname>. Otherwise <new-setname> has MAXIMUM SIZE the same as that for <bigset>.

Intertemporal sets are not allowed in set complement statements. That is, <bigset> and <smallset> are not allowed to be intertemporal sets and the set qualifier (INTERTEMPORAL) is not allowed in a set complement statement.

Set complements can be used in **XSET extra statements** in Command files (see section 6.6 in GPD-3).

4.6.5 Sets Whose Elements Depend on Data

Sets can be defined in TABLO Input files in ways which depend on data, as in for example,

```
SET SPCOM = (all,c,COM: TOTX(c) > TOTY(c)/5 ) ;
```

which says that the set SPCOM consists of all commodities c in COM for which TOTX(c) is greater than one-fifth of TOTY(c). (This set is defined at the first stage of a simulation using initial values of TOTX and TOTY read in or calculated by initial formulas. It will not change at subsequent steps for updated values of TOTX and TOTY).

The required syntax is

```
SET  <new-set>  =  (All, <index> , <old-set> : <condition>  ) ;
```

which says that <new-set> consists of the elements of <old-set> which satisfy the given condition. (See section 3.1.2.)

Such a set definition automatically implies the obvious SUBSET statement (namely that <new-set> is a subset of <old-set>).

If the old set <old-set> has element names defined, then names for the elements of the new set <new-set> are inherited at run time.

Only one quantifier (ALL,<index>,<set>) is allowed. The condition follows the colon ':' in this quantifier. [The condition can contain operators like AND, OR.]

Note that the resulting set can be empty (since empty sets are allowed – see section 4.6.8).

At present, set statements of the kind described in this section which depend on data cannot be included as "extra" statements in Command files (see section 6.6 in GPD-3).

4.6.6 Writing the Elements of One Set

The syntax of the TABLO statement to do this is

```
WRITE (SET) <setname> to FILE <logical-file>
        [HEADER "<header>"]   [LONGNAME "<longname>"]  ;
```

(see section 3.7). Here the HEADER part is required if <logical-file> has been previously declared to be a new Header Array file. The HEADER part is not allowed if <logical-file> has been previously

declared to be a new text file. In the former case (Header Array file case), the output is written as strings of length 12 (the length of set element names – see section 4.2.1). In the second case (Text file case) the output is written using the standard GEMPACK syntax for character strings (see chapter 6 in GPD-4).[34]

When writing to a Header Array file, if the long name is not specified in the WRITE(SET) statement via the optional LONGNAME part, the long name written is of the form

Set <setname> <labelling information>

This ensures that the file produced will be of maximum use with the 'ds' option in MODHAR (see section 3.6.2 in GPD-4).

Note that this syntax (without a HEADER part) was allowed in Release 5.2 if <logical-file> had been declared to be a new GAMS file (see chapter 16 of GPD-4). This functionality remains unaltered.

Note that this type of statement can be added to your Command file as an "extra" statement.

We are grateful to Mark Horridge for suggesting this feature.

4.6.7 Writing the Elements of All (or Many) Sets

The syntax is

```
WRITE (ALLSETS) to file <hafile> ;
```

(see section 3.7).[35] If you are running a Fortran 90 version of GEMSIM or a TABLO-generated program, this will write the elements all sets whose elements are specified. If you are running a Fortran 77 version of these programs, it will only write out the elements of those sets which have been defined earlier in the TABLO Input file (whose elements have been specified).

The file in question must be (the logical name of) a Header Array file. No other writes are allowed to this same file (in order for us to be able to ensure relatively easily that the resulting file will not have duplicate headers).

If the elements of a set have been read from a Header Array file, they will be written to the same header (where possible); otherwise the program makes up a header. Do not include a header name in the WRITE(ALLSETS) statement.

The long name written with the elements of each set is of the form

```
Set <setname> <labelling information>
```

This ensures that the file produced will be of maximum use with the '**ds**' option in MOHDAR (see section 3.6.2 in GPD-4).

Note that this statement can be added to your Command file as an "extra" statement. For example, the following statements in a Command file will write out all (or most) sets to the file somesets.har.

```
xfile (new) manysets ;
file manysets = somesets.har ;
xwrite (allsets) to file manysets ;
```

4.6.8 Empty Sets

TABLO, TABLO-generated programs and GEMSIM allow empty sets. For example, your model may contain a set of sluggish endowment commodities which is a subset of the set of all the endowment

[34] The possibility of writing set elements to all sorts of files was added in Release 6.0. [In Release 5.2, WRITE(SET) was allowed only if writing to a GAMS file.]

[35] This was introduced in Release 6.0.

commodities. If so, it may be convenient to have no endowment commodities sluggish for some simulations, which means that the set of sluggish endowment commodities is empty.[36]

In particular, empty sets can be used in SUBSET and Set Complement, Set Union and Set Intersection statements.

You can define an empty set in various ways.

1. The simplest is to use the "listing elements" style [see (1) in section 3.1], indicating no elements between the brackets as in, for example,[37]
```
SET  EMPTY1   # Empty set #   ( ) ;
```
2. Alternatively you can read the elements from a file. If there are zero strings at this header in the file, the resulting set will be empty. For example,
```
SET EMPTY2  Read Elements from File Setinfo  Header  "EMP2" ;
```
3. You could also use a set complement as in
```
SET EMPTY3 = IND – IND ;
```
4. You could use a condition as in for example (provided COEF1 has been defined and assigned values all less than 5000)
```
SET EMPTY4 = (All, i, IND : COEF1(i) > 5000) ;
```

4.6.9 Reading Set Elements from a File

Element names can be read in at run time when GEMSIM or a TABLO-generated program runs.

The file from which they are read must be a Header Array file.

The type of TABLO statement used is given in set declaration numbered (2) in section 3.1.

For example,

```
SET COM  Read Elements from File Setinfo Header "COM" ;
```

The data at the relevant position of the relevant file must be strings of length no more than 12. [Shorter lengths are ok.] This is because element names are limited to at most 12 characters

For example, if you are preparing header `"COM"` to contain the names of the 20 commodities in your model, the data at the relevant header should be 20 strings of length 12 (or it could be 20 strings of length 8 if all element names were this short).

TABLO Style of Set Element Names

In TABLO Input files, set element names follow the general rule for names given in section 4.2.1, so that

- names of set elements are limited to at most 12 characters,
- names of set elements must commence with a letter
- allowed characters in names consist of letters, digits and/or underscores '_' and/or '@'s,
- must not contain blank characters.

We refer to this style of element names as **TABLO style.**

Flexible Style of Set Element Names

When you are reading element names from a Header Array file, you can also use **Flexible style** for element names. In flexible style the restrictions are relaxed to be

- names of set elements are limited to at most 12 characters, and

[36] Empty sets were new for Release 6.0.

[37] This is new for Release 7.0.

- must not contain blank characters.

There is a new statement for Command files in GEMSIM or TABLO-generated programs[38]:

```
set elements read style = TABLO | flexible ;
```

where `TABLO` is the default.

In order to use this flexible style of set element names you must include in your Command file the statement

```
set elements read style = flexible ;
```

We have included the flexible style of set element names for backwards compatibly with Release 6.0 of GEMPACK. However we encourage you to use the stricter form of TABLO style in new models.

[38] This statement is new for Release 7.0.

4.7 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.
- If two sets have elements defined (either fixed or at run time), a SUBSET(BY_NUMBERS) statement between them is not allowed since the subset mapping can be inferred (at least at run time) from the names of the elements in the sets. [Use a SUBSET(BY_ELEMENTS) statement instead; of course, the qualifier BY_ELEMENTS can be omitted since it is the default – see section 3.2 .]
- 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').]

4.8 Mappings Between Sets

As set out in section 3.13, mappings between sets can be defined via statements following the syntax[39]

```
MAPPING [(ONTO)] <set-mapping>  FROM  <set1>  TO  <set2>  ;
```

Here <set1> and <set2> must have already been declared as sets. The mapping called <set-mapping> is a function which takes an element of set <set1> as input and produces as output an element of set <set2>. The optional qualifier **(ONTO)** requests the software to check that every element of set2 is mapped to by at least one element of set1 – see section 4.8.3 for details.

Sometimes we call these **mappings** and sometimes **set mappings**. We sometimes refer to the set <set1> as the **domain** of the mapping and to the set <set2> as the **codomain** of the mapping.

Example 1 - Unique Production

Suppose each commodity in the set COM is produced by only one industry in the set IND. We can declare a mapping called PRODUCER from COM to IND as follows.

```
MAPPING PRODUCER from COM to IND ;
```

Then PRODUCER("Com1") would be an element of set IND, namely the industry in IND which produced "Com1". This is an example of a one-to-one mapping if each industry produces only one commodity.

Below we give an example of a many-to-one mapping. You can not use as a mapping a many-to-many relation or a one-to-many relation where one element of the first set corresponds to more than one element of the second set. (In the example above this would be where a commodity is produced by more than one industry.)

Example 2 - Aggregating Data

Imagine a data manipulation TABLO Input file which is aggregating some data set with commodities in set COM to an aggregated set of AGGCOM commodities.[40] The following statement could be useful.

```
MAPPING COMTOAGGCOM from COM to AGGCOM ;
```

With this, for each "c" in COM, COMTOAGGCOM(c) is the aggregated commodity to which "c" is mapped. The formula for aggregating domestic household consumption could then be written as follows.

```
COEFFICIENT (all,aggc,AGGCOM) AGGDHOUS(aggc) ;
COEFFICIENT (all,c,COM) DHOUS(c) ;
FORMULA (all,aggc,AGGCOM)
AGGDHOUS(aggc) = SUM(c,COM: COMTOAGGCOM(c) EQ aggc, DHOUS(c) );
```

[This uses an "index expression comparison" (see section 4.4.10) to compare COMTOAGGCOM(c) with "aggc".]

Example 3 - Aggregating Simulation Results

With COM, AGGCOM and the mapping COMTOAGGCOM as in Example 2 above, it is a simple matter to report simulation results for the aggregated commodities. For example, consider the variable **xhous(c)** which reports the percentage change in household consumption of commodity c in COM. Suppose that you wished to aggregate these (and possibly other similar results) to report the percentage change in household consumption of each of the aggregated commodities. Then you might use the mapping COMTOAGGCOM and the TABLO statements:

---

[39] Set mappings were introduced in Release 5.2.

[40] As a very simple example, perhaps COM has the 4 elements wool, wheat, banking, hotels and AGGCOM has the two aggregated commodities agriculture (the aggregation of wool and wheat) and services (the aggregation of banking and hotels).

```
VARIABLE (all,c,COM) xhous(c) ;
VARIABLE (all,aggc,AGGCOM) agg_xhous(aggc) ;
EQUATION E_agg_xhous (all,aggc,AGGCOM)
SUM(c,COM: COMTOAGGCOM(c) EQ aggc, DVHOUS(c)) * agg_xhous(aggc)
= SUM(c,COM: COMTOAGGCOM(c) EQ aggc, DHOUS(c) * xhous(c) );
```

Here the DVHOUS(c) (as in Example 2 above) are suitable weights to use to aggregate the xhous results.[41]

The procedure in Example 3 is perfectly general and can be used with any model to report aggregated results (calculated while the rest of the model solves); this is more reliable and accurate than trying to aggregate percentage changes after the simulation has been run. [The only slightly tricky thing is sometimes finding suitable weights to use.]

Of course MAPPINGs must be defined. That is, the TABLO Input file must give a way of specifying <set-mapping>(s) for all s in <set1>. And the values of <set-mapping>(s) must be in the set <set2> for each s in <set1>.[42]

4.8.1 Defining Set Mapping Values

Values for set mappings can be established in any of the following ways.

- **by reading the element NAMEs in <set2> where each element of <set1> is mapped**.
  An example is

```
READ (BY_ELEMENTS) COMTOAGGCOM from File setinfo ;
```

  The qualifier "(BY_ELEMENTS)" is required. The file *setinfo* can be a text file or a Header Array file (in which case a HEADER must also be specified). The data on the file at the relevant place must be CHARACTER data giving, for each c in COM, an element name in AGGCOM. Since element names are limited to at most 12 characters (see section 4.2.1), the data on the file should be strings of length no more than 12. [Shorter lengths are ok.]

- **by formula(s) assigning the element NAMEs** in <set2> to which the different elements of <set1> are mapped. An example is

```
FORMULA COMTOAGGCOM("C10TCF") = "AC5Manuf" ;
```

  where the RHS is an element of AGGCOM. This uses element NAMEs on the RHS.
  Another example of this type is

```
FORMULA (all,c1,COM1) COMTOAGGCOM(c1) = "AC5Manuf" ;
```

  where COM1 is a subset of COM.

- **by reading the element NUMBERs in <set2> each element of <set1> is mapped**.
  An example is[43]

---

[41] Using the commodities in the previous footnote, note that the equation says, for example, that
[DVHOUS("wool")+DVHOUS("wheat")]*agg_xhous("agriculture") =
DVHOUS("wool")*xhous("wool") + DVHOUS("wheat")*xhous("wheat").
You can see that agg_xhous("agriculture") is a suitably weighted average of xhous("wool") and xhous("wheat").

[42] Set mappings do not need to be "onto" in the mathematical sense. Consider a mapping from SET1 to SET2; it need not be the case that every element in SET2 is mapped to by some element in SET1. [Of course, for data aggregation, you may well want this.] You can ask the software to check that a mapping is onto – see section 4.8.3.

[43] The Release 5.2 documentation about set mappings inadvertently omitted information about reading (and writing - see later in this section) set mappings as integers.

```
READ COMTOAGGCOM from File setinfo ;
```

The file *setinfo* can be a text file or a Header Array file (in which case a HEADER must also be specified). The data on the file at the relevant place must be INTEGER data giving, for each c in COM, an element number in AGGCOM. It is also possible to read as integers just some of the values of a mapping, as in the example

```
READ (all,c1,COM1) COMTOAGGCOM(c1) from File setinfo ;
```

where the set COM1 is a subset of the domain COM of the mapping COMTOAGGCOM. [Note that the qualifier "(BY_ELEMENTS)" is not present when integer data is read; this qualifier indicates character data.]

- **by formula(s) assigning the element NUMBERs** in <set2> to which each element of <set1> is mapped. An example is

```
FORMULA  COMTOAGGCOM("C10TCF") = 5 ;
```

where the RHS is the element number in AGGCOM. This uses element NUMBERs on the RHS. [Indeed, in this case, the RHS can be any expression allowed on the RHS of FORMULAs whose LHS is an Integer Coefficient - see section 4.5.3 for details. For example, the RHS could be ICOEF+2 in the example above if ICOEF had been declared as an Integer Coefficient.]

- **by formula(s) of the type** FORMULA(BY_ELEMENTS) assigning the element NAMEs as in

```
Formula(By_Elements) (All,m,MAR) MAR2AGGMAR(m)=COM2AGGCOM(m) ;
```

Here the RHS is another mapping, or it could be an index. See section 4.8.11 for more details.

Example - Aggregating Data or Results

Consider the following statements

```
SET  COM (wool, wheat, banking, hotels) ;
SET  AGGCOM (agriculture, services) ;
MAPPING COMTOAGGCOM from COM to AGGCOM ;
```

The mapping COMTOAGGCOM could be defined in one of the following ways.

(1) Via the following formulas with element NAMEs on the RHS

```
Formula  COMTOAGGCOM("wool") = "agriculture" ;
Formula  COMTOAGGCOM("wheat") = "agriculture" ;
Formula  COMTOAGGCOM("banking") = "services" ;
Formula  COMTOAGGCOM("hotels") = "services" ;
```

(2) Via the following formulas with element NUMBERs on the RHS

```
Formula  COMTOAGGCOM("wool") = 1 ;
Formula  COMTOAGGCOM("wheat") = 1 ;
Formula  COMTOAGGCOM("banking") = 2 ;
Formula  COMTOAGGCOM("hotels") = 2 ;
```

(3) Via the following file and read statements

```
File (text)  setinfo ;
Read (BY_ELEMENTS) COMTOAGGCOM from file setinfo ;
```

In this case, the text file SETINFO should have the following array on it.

```
4 strings length 12 ;
agriculture
agriculture
```

```
services
services
```

(4) Via the following file and read statements[44]

```
File (text)  setinfo ;
Read COMTOAGGCOM from file setinfo ;
```

In this case, the text file SETINFO should have the following array on it.

```
4 integer ;
1  1  2  2
```

(5) Via a FORMULA(BY_ELEMENTS)

```
Formula(By_Elements) (All,m,MAR) MAR2AGGMAR(m)=COM2AGGCOM(m) ;
```

See full details in section 4.8.11 below.

Note the general form of defining statements for COMTOAGGCOM:
COMTOAGGCOM("element from COM") is equal to either an "element of set AGGCOM" or else to an element number of set AGGCOM.

4.8.2 Checking Values of a Mapping

When you assign values to a set mapping, GEMSIM and TABLO-generated programs check that the values are sensible. For example, with COM, AGGCOM and COMTOAGGCOM as above, the formulas

```
COMTOAGGCOM("wool") = "steel" ;
COMTOAGGCOM("wool") = 15 ;
```

would lead to errors at run time since the "steel" is not an element of AGGCOM and AGGCOM does not have 15 elements.

The values assigned to a set mapping are not checked after each assignment.[45] Rather they are checked just before the set mapping is used (in a formula) or at the end of the preliminary pass (once all sets and set mappings have been set up).

The values of all set mappings are checked, even if the set mapping is not used in any formula, equation or update.[46]

4.8.3 Insisting That a Set Mapping be Onto

As far as GEMPACK is concerned, set mappings do not need to be "**onto**" in the mathematical sense. That is, for a mapping from SET1 to SET2; it need not be the case that every element in SET2 is mapped to by some element in SET1.[47] For example, if mapping MAP3 maps the set EXPIND of

---

[44] In examples (3) and (4) here, a Header Array file could be used instead of a text file.

[45] This is to allow several different formulas or reads to set up the values of each set mapping, if necessary.

[46] This is a change in Release 7.0. Previously the values of unused set mappings were not checked. If this causes you problems, please contact the code developers and ask how to use debug option 74 in TABLO.

[47] A mapping from SET1 to SET2 is defined to be **onto** in the mathematical sense if every element of the codomain set SET2 is mapped to by at least one element of the domain set SET1. For example, suppose MAP3 maps COM1=(c1,c3,c4) to COM2=(c6,c7).
If MAP3("c1")=MAP3("c3")=MAP3("c4")="c6" then MAP3 is not onto since nothing maps to c7.
If MAP3("c1")=MAP3("c4")="c6" and MAP3("c3")="c7" then MAP3 is onto.

export industries to the set IND of all industries, there may be industries in IND which are not mapped to by anything in EXPIND.

Of course, for data aggregation, you may well want the mapping to be onto.

If you use the qualifier **(ONTO)** when defining the mapping, the software will check that the every element of the codomain is mapped to by at least one element of the domain.[48] If you use this qualifier when declaring the mapping and the mapping is not onto, the program will stop with an error.

The check that the mapping is **onto** is done at the same time as the check that the values are in range (see section 4.8.2).

4.8.4 Using Set Mappings

Set MAPPINGs can be used in arguments of VARIABLEs and COEFFICIENTs.

For example, continuing on the aggregation example, we could write

```
COEFFICIENT (all,c,COM) SHARE(c)
# Share of com c in aggregate com # ;
FORMULA (all,c,COM)
SHARE(c)= DHOUS(c) / AGGDHOUS( COMTOAGGCOM(c)) ;
```

We can write `AGGDHOUS(COMTOAGGCOM(c))`
since AGGDHOUS requires one argument which is in the set AGGCOM and "COMTOAGGCOM(c)" is in AGGCOM since "c" is in COM.

But it would be an error to write `DHOUS(COMTOAGGCOM(c)) !Wrong !`
because DHOUS requires one argument which is in the set COM yet COMTOAGGCOM(c) is an element of the set AGGCOM (so is not in the set COM since AGGCOM is not a subset of COM).

4.8.5 Set Mappings Can Only Be Used in Index Expressions

The term “index expression” was defined in section 4.2.3. An index expression is either an index or an element or an expression involving indices, element, index offsets and set mappings. Index expressions can be used as the arguments of coefficients or variables (see section 4.2.3), or in index-expression comparisons (see section 4.4.10).

Set MAPPINGs can **ONLY be used in index expressions** - that is, in arguments of VARIABLEs or COEFFICIENTs as in

```
AGGDHOUS(COMTOAGGCOM(c))
```

or in index-expression-comparisons such as

```
COMTOAGGCOM(c) EQ aggc
```

in the aggregation example given earlier.

4.8.6 Two or More Set Mappings in an Index Expression

Set mappings can be applied to expressions which themselves involve one or more set mappings. In mathematical parlance, this is **composition** of functions.

For example, suppose that MAP1 and MAP2 are set mappings.

```
MAPPING MAP1 FROM S1 TO S2 ;

MAPPING MAP2 FROM S3 TO S4 ;
```

Since MAP1 maps set S1 into set S2 and MAP2 maps set S3 into set S4, then the expressions

---

[48] This was introduced with Release 7.0.

```
MAP1(MAP2(i))
MAP1(MAP2("element"))
```

are legal provided that

- index i is ranging over set S3 or over a subset of S3,
- "element" is an element of set S3,
- S4 is the same as S1 or else S4 is a subset of S1.

[For example, then MAP2(i) is in the set S4 because "i" is in S3 and so MAP2(i) is also in the set S1 since S4=S1 or S4 is a subset of S1; hence MAP1(MAP2(i)) makes sense since the argument "MAP2(i)" of MAP1 is in the set S1, as required.] The resulting expressions are thought of as being in set S2 (since this is where the outer mapping MAP1 maps things).

If intertemporal sets and index offsets are involved, similar rules apply. Basically the index offset does not affect the set in which the expression is thought to be. For example, if set S1 is an intertemporal set, then the expression MAP1(MAP2(i) + 2) is legal precisely when MAP1(MAP2(i)) is. This is because MAP2(i)+2 lies in the same set as MAP2(i).

4.8.7 Set Mappings in Arguments

When an argument of a coefficient or variable involves a set mapping, the argument must be in the appropriate set.

For example, supppose that MAP1 and MAP2 are as in the section above and consider Coefficients X4 and X5 is defined by

```
COEFFICIENT  (All,c,S2)  X4(c) ;
COEFFICIENT  (All,c,S5)  X5(c) ;
```

Suppose that i is an index ranging over the set S3 or a subset of S3. Recall from the section above that the index expression MAP1(MAP2(i)) is thought of as being in the set S2 (the codomain of MAP1). Hence

- the expression X4(MAP1(MAP2(i)) is allowed since the argument MAP1(MAP2(i)) is known to be in set S2 which is where the argument of X4 must be.
- the expression X4(MAP1(MAP2(i)) is only allowed if the set S2 (the codomain of MAP1) is a subset of the set S5 since the argument of X4 must be in set S5.

4.8.8 Other Semantics for Mappings

- Several READs and/or FORMULAs can be used to set up the values of each set mapping. But, once a set mapping is used (for example, in a Formula or in a Write, or in an index expression in a condition as in section 4.4.10), its values cannot be changed (that is, it cannot appear in a READ statement or on the LHS of a Formula).
- Set mappings are not allowed in index expressions on the LHS of a formula or of an update. For example, the following is not allowed.

  ```
  FORMULA (all,c,COM) C1(COMTOAGGCOM(c)) = C2(c) ;  !Wrong !
  ```

  [If several different "c"s in COM were mapped to the same place in AGGCOM, the results would be ambiguous since it would depend on the order of processing the elements "c" in COM.]
- Set mappings are not allowed in the declaration of coefficients or variables. Nor are they allowed in displays.

4.8.9 Writing the Values of Set Mappings

Although Set Mappings are not the same as Integer Coefficients, they can be written as if they were. For example, the statement

```
WRITE COMTOAGGCOM to terminal ;
```

is allowed. The values of the set mapping COMTOAGGCOM will be written as integers (which will show the position number in AGGCOM of COMTOAGGCOM(c) for each c in COM).[49]

4.8.10 Reading Part of a Set Mapping BY_ELEMENTS

It is possible to read just some of the values of a set mapping by elements.[50] That is, statements of the form

```
READ (BY_ELEMENTS)  (all,i1,S1)  MAP1(i1)  from file … ;
```

are allowed. [Above, if MAP1 is a mapping from set S2 to set S3, then S1 must be a subset of S2.]

For the above to be valid, the data on the file (at the relevant header if it is a Header Array file, or in the relevant position if it is a text file) must be character data. [It cannot be integer data as would be required if the qualifier (BY_ELEMENTS) were omitted.]

4.8.11 Mapping Values Can be Given by Values of Other Mapping or Index[51]

Mapping Example 1

An example will make this clear. Suppose that you are aggregating some data. You have defined the set COM of commodities and the set AGGCOM of aggregated commodities, and have defined a set mapping COM2AGGCOM from COM to AGGCOM. [For each c in COM, COM2AGGCOM(c) is the aggregated commodity c belongs to.] Suppose also that MAR is the set of margins commodities (MAR is a subset of COM) and that AGGMAR is the set of aggregated margins commodities. You want to define a mapping MAR2AGGMAR from MAR to AGGMAR. Indeed, for each m in MAR, the value of MAR2AGGMAR(m) should just be the commodity in AGGCOM to which m maps under the mapping COM2AGGCOM already set up. So you want a formula of the form

```
FORMULA  (All,m,MAR)  MAR2AGGMAR(m) = COM2AGGCOM(m) ;
```

This sort of formula is allowed provided you add the qualifier (BY_ELEMENTS). That is, you define the mapping and give it values via the 2 statements

```
MAPPING  MAR2AGGMAR  from  MAR  to  AGGMAR ;
FORMULA (BY_ELEMENTS)  (All,m,MAR)
              MAR2AGGMAR(m) = COM2AGGCOM(m) ;
```

To make this concrete, suppose for simplicity that COM is the set (c1-c10), that MAR is the subset (c6-c9), that AGGCOM is the set (agc1-aggc5) and that COM2AGGCOM maps each of c1,c2,c3 to aggcom1, each of c4,c5 to agcom2, each of c6,c7 to aggcom3, each of c8,c9 to aggcom4 and c10 to aggcom5. Suppose also that AGGMAR is the set (aggomc3,aggcom4).

Then COM2AGGCOM("c6") is equal to "aggcom3" and so the formula above will set MAR2AGGMAR("c6") equal to "aggcom3" as you would expect. Similarly for the other elements of MAR. You can see that MAR2AGGMAR will map c5 and c6 to aggcom3 and each of c7,c8 to aggcom4.

---

[49] The Release 5.2 documentation about set mappings inadvertently omitted information about reading and writing set mappings as integers.

[50] This was introduced with Release 7.0.

[51] This was introduced with Release 7.0.

Note that, in order to understand the formula above for MAR2AGGMAR, you must think of the element names, which is why the qualifier (BY_ELEMENTS) is required.

In General

Suppose that set mapping MAP1 has been declared via the statement

```
MAPPING   MAP1  from S1  to  S2 ;
```

(where S1 and S2 are sets which have already been defined). You are allowed to use the following kinds of formulas to define some or all the values of a set mapping.

```
FORMULA (BY_ELEMENTS)  (All,i1,S3)  MAP1(i1) = <index-expression> ;
FORMULA (BY_ELEMENTS)    MAP1("e1") = <index-expression> ;
```

Above S3 must be equal to S1 or else a subset of it, and "e1" must be an element of the set S1.

Above, <**index-expression**> can begin with a set mapping (as in the example above) or can be simply an index. [Index expressions are defined in section 4.2.3.]

For the first formula above to be valid,

- the set S2 which contains the values of MAP1 must have known elements (defined in the TABLO Input file or read at run time),
- the set to which the index expression belongs must also have known elements,
- for each i1 in S1, the element defined by the index expression must be an element of the set S2.

If either of the first two conditions above is false, TABLO will report an error. The third condition can only be checked when the TABLO-generated program or GEMSIM runs. If it fails, the details will be reported when that program runs.

Suppose that in the example above COM2AGGCOM("c9") were equal to "aggcom5" instead of "aggcom4". Then the formula

FORMULA (BY_ELEMENTS) (All,m,MAR)   MAR2AGGMAR(m) = COM2AGGCOM(m) ;

would produce an error when m is equal to c9 since then the RHS is equal to aggcom5 which is not an element of the set AGGMAR.

Note that no subset relationship is required between the set in which the values of <index-expression> lies and the set S2 which is to contain the values of MAP1.

Note that <index-expression> cannot be an index followed by an index offset. This is because, even if the index is ranging over an intertemporal set, the offset would take one value of the index expression out of the set over which the index is ranging.

Mapping Example 2

Suppose that a model has sets COM of commodities and IND of industries. Suppose that some industries produce several commodities and that other industries (those in the subset UPIND of IND) produce only one commodity. Suppose also that for the names of the industries in UPIND are the same as the names of the commodities they produce. Then you can define a set mapping from UPIND to COM via the statements

MAPPING   UPIND2COM from UPIND to COM ;

FORMULA (BY_ELEMENTS) (all,i,UPIND) UPIND2COM(i) = i ;

This is an example where the <index-expression> on the RHS is just an index. Note that the formula above is valid even if UPIND has not been declared to be a subset of COM.

4.8.12 Writing a Set Mapping to a Text File

When a set mapping is written to a text file via a "simple" write (that is, one with no ALL quantifiers), comments are added after each value which indicate the names of the elements in

question.[52] These comments indicate the name of the element being mapped and the name of the element to which it is mapped. For example,

```
Write COM2AGGCOM to terminal ;
```

might produce the output

```
3 integer header "Mapping COM2AGGCOM from COM(3) to AGGCOM(2)" ;

1               !  c1           maps to aggc1
1               !  c2           maps to aggc1
2               !  c3           maps to aggc2
```

4.8.13 Writing a Set Mapping as Character Data

The statement

```
WRITE (BY_ELEMENTS) <set-mapping> … ;
```

can be used to tell the software to write the values of the set mapping as character strings (rather than integers).[53] In particular, if the mapping is being written to a Header Array file, the header written is of type '1C' and contains the names of the elements to which the elements of the domain set are mapped by the mapping.

- The write qualifier BY_ELEMENTS is not allowed if there is an ALL quantifier in the statement.
- This qualifier BY_ELEMENTS is, of course, not allowed if the elements of the codomain set (the set which is being mapped to) are not known.
- This qualifier BY_ELEMENTS is not allowed in a Fortran 77 program (GEMSIM or TABLO-generated) if the mapping is being written to a Header Array file (as distinct from a text file), but is allowed in Fortran 90 programs.

4.8.14 Long Name when a Set Mapping is Written to a Header Array File

When all elements of a set mapping are written to a Header Array file via a "simple" Write (that is, no with no ALL quantifiers), if a long name is not specified in the Write statement, the software writes a special form of long name.[54] The long name is of the form

Mapping <mapping-name> from <domain-set>(<size>) to <codomain-set>(<size>)

For example, consider the mapping COM2AGGCOM which maps a set COM of size 23 to the set AGGCOM of size 14. The long name used would be

Mapping COM2AGGCOM from COM(23) to AGGCOM(14)

This form of long name is used by the aggregation functions in ViewHAR to recognise set mappings.

Note that, for simple writes of a set mapping, the convention described above replaces that described in section 4.10.6 (which still applies to other writes).

4.8.15 ViewHAR and Set Mappings

ViewHAR allows you to save set mappings as Header Array or text files, or to paste them into spreadsheets or TAB files. [It has similar capabilities for sets.]

---

[52] This was introduced with Release 7.0.

[53] This was introduced with Release 7.0.

[54] This was introduced with Release 7.0.

4.9 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 file names. When no actual file name 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 file names in FILE statements, unless you have good reason to do so.

See section 4.1 of GPD-3 to find out how to make the connection between a logical file name and the associated actual file when GEMSIM and TABLO-generated programs run.

- Files can be GEMPACK Header Array files (see section 3.4 of GPD-1 and section 3.1 of GPD-4) or text files (see section 3.4 of GPD-1, section 4.9.1 below and chapter 6 of GPD-4 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 only read character data
  - * from a Header Array file as part of a SET statement which asks for the set elements to be read (see section 3.1). [Set element names should be on the file as an array of character strings, each string being of (the same) length up to 12. This is because set element names are limited to 12 characters – see section 4.2.1.]
  - * from a Header Array or text file to define all or part of a set mapping (see sections 4.8.1 and 4.8.10). [Again the length of the string can be up to 12.]

4.9.1 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.[55]

```
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,

[55] GEMSIM and TABLO-generated programs ignore any header in the "how much data" information (see chapter 6 of GPD-4) at the start of each array.

```
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 ;
```

- Details of the preparation of data for text files are given in chapter 6 of GPD-4, with an example given in section 3.4 of GPD-1.
- When preparing text data for a partial read, such as

```
(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.
- 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 set elements and subsets-by-number from Header Array files, not from text files (see sections 3.1 and 3.2). But these programs can read the values of a set mapping from a text file (see section 4.8.1).

4.10 Reads, Writes and Displays

4.10.1 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 or text 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 while A3("food","labor") is assigned the value of the entry in row 3 and column 1 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.

- GEMSIM and TABLO-generated programs can check the order of the elements of a set, as defined in the TABLO Input file or read at run time when the set is defined, against the element names which may be stored on a Header Array file from which data is being read. This allows you to guard against errors that would occur if a data item were assigned to the wrong part of a Coefficient. You can control how much of this checking is carried out. See section 4.4 of GPD-3 for details.

4.10.2 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),

    1. all indices associated with the coefficient must be different. For example,

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

        is not allowed.

    2. 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.

    3. 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.[56]

- Similar rules apply to partial writes and displays.
- Index offsets (see section 4.2.3) are not allowed in READs, WRITEs or DISPLAYs. For example, the statement

    ```
    READ (all,t,fwdtime) X(t+1) From File (etc) ;
    ```

    is not allowed.

4.10.3 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 ... ;
    ```

---

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

A temporary file (the "intermediate extra data" file - see section 4.2.3 of GPD-3) 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 4.10.2 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.

- Index offsets (see section 4.2.3) are not allowed in the left-hand side of a FORMULA(INITIAL) since they are not allowed in a READ statement. For example, the statement

  Formula (Initial) (all,t,fwdtime) X(t+1) = Y(t) ;

  is not allowed.

4.10.4 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"). See section 6.7 of GPD-3 for an example of this. There a coefficient is written even though some of its values have not been initialised.

4.10.5 Display Files

All DISPLAYs are written to a single text file called the **Display file**. Details can be found in section 4.3 of GPD-3.

4.10.6 Transferring Long Names when Executing Write Statements

When writing an array to a Header Array file, if the long name to use is not specified in the TABLO Input file, TABLO-generated programs and GEMSIM transfer the long name (if not all blanks) from when this same data was read. The intention here is to make it easier for modellers to preserve long names on files when doing data organisation tasks. [We are grateful to Mark Horridge for suggesting this.][57]

More precisely, when these programs write an array to a Header Array file, the long name written is as set out in the rules below.

(i) If the long name is specified in the TABLO Input file, this long name is used. For example, for the WRITE statement,

```
WRITE V5BAS to file xx header "YYYY" longname "Y Matrix" ;
```

the long name is as given "Y Matrix".

(ii) Otherwise, if exactly the same part of the coefficient has been read from a Header Array file earlier in this TABLO Input file, the long name read in from the data file is used. For example, with the statements

```
READ V5BAS from file yy header "ABCD" ;
WRITE V5BAS to file  xx header "YYYY"  ;
```

the long name for the written array at header "YYYY" is the same as the long name for the array on file yy at header "ABCD".

Here the "part of the coefficient" means the sets and elements specified in the write statement. For example, with the statements

```
READ V5BAS from file yy header "ABCD" ;
WRITE (all,c,COM) V5BAS(c,"imp") to file xx header "V5BS" ;
```

the long name read at the earlier READ statement will NOT be used in the WRITE statement since, for the array V5BAS at header "ABCD", all of V5BAS is read but only the "imp" part of V5BAS is being written.

If there is no long name on the read data file(s), option (iii) is tried.

(iii) Otherwise the labelling information (given between #'s on the TABLO Input file) for the Coefficient being written is used as the basis for the long name. For example, with the statements

```
COEFFICIENT(all,c,COM)(all,s,SOURCE) V5BAS(c,s) #Other Demands# ;
WRITE V5BAS to file  xx header "YYYY"  ;
```

the labelling information "Other demands" becomes the long name for the array written to header "YYYY".

If all of the coefficient is being written (as above), the long name is just this labelling information.

---

[57] This was introduced with Release 6.0.

If only part of the coefficient is being written, the long name used is the coefficient name followed by sets or elements as arguments (saying which part is being written) followed by the labelling information for the coefficient.

For example, if the write statement is

```
WRITE (all,c,MARGCOM) V5BAS(c,"dom") to file xx header "V5BS" ;
```

the long name written will be the coefficient name V5BAS(MARGCOM,"dom") followed by the coefficient labelling information for coefficient V5BAS, so the long name is

```
      V5BAS(MARGCOM,"dom")  Other Demands
```

4.11 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.

4.11.1 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.

  These values are used to calculate the change in the values of the coefficient on the **left-hand side** of the update. These changes are added to the current values of that coefficient to give the new values for this coefficient (that is, its values at the end of the current step).
- 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 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.[58]
- 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.

4.11.2 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 9.1.

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];

---

[58] The values of other coefficients may change from step to step of a multi-step calculation. But this happens without a formal Update statement. For example, consider the coefficient DVCOM in the TABLO Input file SJLN.TAB for Stylized Johansen shown in section 3.5.1 of GPD-1. There is no update statement for DVCOM. However its values change from step to step of a multi-step simulation to reflect the changes in the values of DVCOMIN and DVHOUS, which are updated. At each step, the new values for DVCOM are calculated via the
FORMULA (All,i,SECT) DVCOM(i) = SUM(j,SECT, DVCOMIN(i,j)) + DVHOUS(i) ;
[This is a FORMULA(ALWAYS).] The values on the RHS are the current (updated) values for DVCOMIN and DVHOUS.

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. Sections 4.11.5 and 4.11.6 below, 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 in the model 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).

4.11.3 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**> ;

4.11.4 UPDATE Semantics

Some of the semantics of Update statements have been given earlier. Here we list the remaining details.

- 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 (see section 4.11.1 above).
- An updated coefficient must not appear on the left-hand side of any FORMULA(ALWAYS). [This is so that its values throughout in the current step stay equal to those calculated as a result of the update statements at the end of the previous 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 do not change, you only need to include an UPDATE statement to update the parts that change. TABLO infers that the other parts do not change.[59]

  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

[59] 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.

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'.

- 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.)

4.11.5 Deriving Update Statements – Example 1 (Sum of Two Flows)

Here we take an example in which a Coefficient represents a value on the data base which is the sum of two dollar flows. We explain how to derive the Update statement for this coefficient.

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 (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 9.2). 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].

The UPDATE statement in the TABLO Input file would be

```
UPDATE (CHANGE) VL = VL *  {S1*(p+x)+S2*(q+y)}/100  ;
```

Here the right-hand side in the value of the change in VL as a consequence of the percentage changes p,x,q and y in PL,XL,Ql and YL respectively. [See section 4.11.2 for an introduction to Change Updates.]

4.11.6 Deriving Update Statements – 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, (2)

WL = PW*PX*XL. (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 ;
```

[Section 3.5.3 of GPD-1 tells when Product Updates can be used.]

4.11.7 Writing Updated Values from FORMULA(INITIAL)s

Previously, when the initial (that is, pre-simulation) values of a coefficient were assigned via a FORMULA(INITIAL), the post-simulation values of this coefficient were not available. Now they can be written to a file by including appropriate instructions in the FORMULA(INITIAL) statement.

For example, the qualifier "WRITE UPDATED ..." below

```
FORMULA (INITIAL,
    WRITE UPDATED VALUE TO FILE upd_prices
                  HEADER "ABCD" LONGNAME "<words>" )
    (all,c,COM) PHOUS(c) = 1 ;
```

will ensure that the updated (ie post-simulation) values of PHOUS will be written to logical file "upd_prices" at the specified header with the specified long name.

In such a case, the logical file can point to either a Header Array file or to a text file. [If a text file, no header or longname is specified.]

A new FILE qualifier "**FOR_UPDATES**" is provided to declare a logical file which can have updated values written to it. For example, the declaration

```
FILE (FOR_UPDATES) io_updated  #to contain updated prices# ;
```

declares a Header Array file which can have updated values written to it. This qualifier "FOR_UPDATES" is an alternative to the current qualifiers 'OLD' and 'NEW'. [See section 3.5 for information about the FILE qualifiers OLD/NEW.]

The logical file appearing in the "WRITE UPDATED VALUE TO FILE <file>" must either have been declared using FILE qualifier "FOR_UPDATES" or have been declared as an OLD Header Array file. [An OLD text file is not allowed here.] In the case of an OLD Header Array file, the updated values from the "WRITE UPDATED VALUE TO FILE" statement will appear on the same updated data file as the updated values of coefficients originally read from this file.

Of course the HEADER and LONGNAME can only specified when the file in question is a Header Array file and, in this case, HEADER is required and LONGNAME is optional (the long name will be set all blank if LONGNAME is omitted).

When a logical file is declared with qualifier "FOR_UPDATES", nothing can be read from this file and an updated version will be created after the simulation. [So, in your Command file, a statement "updated file <logical-name> = ... ;" is required to specify the post-simulation version of this file, but a statement "file <logical-name> = ... ;" is not required since no pre-simulation version of this file is expected.] See also section 7.6.2 and 14.3 in GPD-3 on saving all FORMULA(INITIAL) updated values.

4.12 Transfer Statements

When a TABLO Input file has instructions to read data from a Header Array file, there may be extra data on the file which is not read when the TABLO-generated program or GEMSIM runs. Sometimes you may want some or all of this extra data to be transferred to the new or updated Header Array files written.[60]

The statements

TRANSFER <header> FROM FILE <logical-file1> TO FILE <logical-file2> ;

TRANSFER UNREAD FROM FILE <logical-file1> TO FILE <logical-file2> ;

TRANSFER UNWRITTEN FROM FILE <logical-file1> TO FILE <logical-file2> ;

TRANSFER <header> FROM FILE <logical-file1> TO UPDATED FILE ;

TRANSFER UNREAD FROM FILE <logical-file1> TO UPDATED FILE ;

(see section 3.15) are allowed in TABLO Input files to facilitate this.

For example, if you have data at header "HEA1" on logical file FILE1, the statement

```
TRANSFER "HEA1" FROM FILE file1 TO FILE file2 ;
```

is a request to write the data at header "HEA1" on file FILE1 to the new Header Array file FILE2. This statement will be invalid unless logical file FILE2 has been declared via a FILE (NEW) statement.

Transfer statements are similar to a READ statement followed by a WRITE statement. They also have the advantages that

- you do not need to declare a COEFFICIENT to hold the data read, and
- you can transfer character as well as real or integer data.

If some of the data on logical file FILE1 is being updated, the statement

```
TRANSFER "HEA1" FROM FILE file1 TO UPDATED FILE ;
```

will cause the data at header HEA1 on FILE1 to be transferred across to the updated version of FILE1.

The statement

```
TRANSFER UNREAD FROM FILE file1 TO FILE file2 ;
```

causes all data on file FILE1 which is not read to be transferred.

The statement

```
TRANSFER UNWRITTEN FROM FILE file1 TO FILE file2 ;
```

causes all data on file FILE1 at a header to which nothing has been written on file2 to be transferred.

The difference between TRANSFER UNREAD and TRANSFER UNWRITTEN can be seen in the following example.

```
File  file1 ;
File (new) file2 ;
! Omit declaration of sets and coefficients here !
Read Coef1 from file file1 header "C1" ;
Read Coef2 from file file1 header "C2" ;
Formula (all,c,COM) coef2(c) = coef1(c) + coef2(c) ;
Write Coef2 to file file2 header "C2" ;
```

After this the statement

```
Transfer Unread from file1 to file2 ;
```

---

[60] This was introduced in Release 7.0.

will not transfer the data at header “C1” since that has been read. But the statement

```
Transfer Unwritten from file1 to file2 ;
```

will transfer the data at header “C1” since nothing else is written to that header on file2. [Either of these statements will cause data at headers other than “C1” and “C2” on file1 to be transferred to file2.]

Of course, TRANSFER instructions must not result in duplicate headers on a Header Array file. Thus, for example, the two statements

```
WRITE COEF1 to FILE file2 HEADER “HEA3” ;
TRANSFER “HEA3” FROM FILE file4 TO FILE file2 ;
```

will result in an error. For similar reasons, a

```
TRANSFER UNREAD …..
```

statement only causes data to be transferred if that transfer will not result in duplicate headers on the output file. Note that a TRANSFER UNREAD or TRANSFER UNWRITTEN statement can result in duplicate headers, which will mean an error reported only when GEMSIM or the TABLO-generated program runs. For example, the two statements

```
Transfer Unread from file1 to file3 ;
Transfer Unread from file2 to file3 ;
```

will result in a duplicate headers error at run time if both files file1 and file2 contain data at a common header.

4.12.1 XTRANSFER Statements on Command Files

These are “extra” TRANSFER statements (see section 6.6 of GPD-3). They have the same effect as the corresponding TRANSFER statement on the TABLO Input file. The syntax is the same as for a TRANSFER statement except that the key word is “XTRANSFER” instead of “TRANSFER”.

4.13 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 = SUM(i, IND, 1) ; ! Counts number of things in IND !
FORMULA RECIP_NIND = 1/NIND ;
ZERODIVIDE DEFAULT     RECIP_NIND ;
FORMULA (ALL,c,COM)(ALL,i,IND) S(c,i) = A(c,i)/SUM(k,IND,A(c,k)) ;
```

  [Note the formula for NIND. This use of SUM saves you having to hard-wire the size of IND in your code.]

- 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 15.3 of GPD-3 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 5.1.1 below.)

4.14 Ordering

4.14.1 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 :

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

4.14.2 Ordering of Variables

[This refers to the order that the *variables* appear in the Solution file, which usually determines their order when you look at simulation results via GEMPIE or ViewSOL. It also refers to the order in which the variables occur in the columns of the Equations matrix (see section 2.11.1 of GPD-1).]

- 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 of the variables on the Solution file, which usually affects the order of the variables in simulation results. It also determines the order in which the variables occur in the columns of the Equations Matrix, which is relevant if you are using SUMEQ (see chapter 13 of GPD-4) to look at the Equations file.[61] Of course, it is easy to change this order in the TABLO Input file by a fairly simple edit.
- 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 7.5 of GPD-4.

[61] It also determines the order in which the variables are presented in GEMSIM, a TABLO-generated program or SAGEM if you choose to respond to prompts in order to specify the exogenous variables in a simulation. But you can ignore this since we recommend that you always use a Command file for simulations rather than responding to prompts which is error prone and difficult to reproduce.

4.14.3 Ordering of Components of Variables

[This refers to the order, within a given variable, that the components of that variable appear in the Solution file (and hence in simulation results) or in the columns of the tableau or Equations Matrix (see section 2.11.1 of GPD-1).]

- 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 5.3 of GPD-3.

- 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 when simulation results are reported.

  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.)

4.14.4 Ordering of the Equation Blocks

- The order of the equation blocks in the tableau or Equations Matrix (see section 2.11.1 of GPD-1) is determined by the order in which they are declared on the TABLO Input file. You may need to know this if you are using SUMEQ (see chapter 13 of GPD-4) to examine the Equations file.

4.14.5 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 (via SUMEQ, for example).

- 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"

4.14.6 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 6.2 of GPD-3 and in Figure 6.2 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

  anywhere in every equation, or
  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.

  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).

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 :

```
                    --
                   |  -1.0 + S(i1),   if i1 = i2,
ETA(i1,i2)   =     |
                   |   S(i1),          otherwise.
                    --
```

One way of achieving this is to use the fact that the formulas are carried out in the order in which they appear in the TABLO Input file. Thus the following two formulas also achieve what is wanted.

```
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 8.3 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).

4.15 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 4.5.2 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.

If you are using a version of TABLO compiled with a Fortran 77 compiler, 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[62]. 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 TABLO does not know that there are no equations in the TABLO Input file (which a Fortran 77 TABLO only knows if there is an “EQUATION (NONE) ;” statement at the start), 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.

---

[62] If you are using a version of TABLO which has been compiled with a Fortran 90 compiler (this includes the Executable-Image version of GEMPACK), you do not need to include the statement EQUATION (NONE) ; in your TABLO Input file. TABLO infers this after it has done the preliminary count of the numbers of various statements (including the number of equations). [However, this statement is still accepted by Fortran 90 TABLOs.]

CHAPTER 5

5. 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.

```
               TABLO PORTABLE
                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)
                                    W77 Write fortran 77 TG-program
      ACC All comment lines in      NMS Don't allow multi-step
             code                         simulations
                                    CIN Code file name same as
                                          Information file name
                                    CDM Change one or more default
                                          maximum values in the code
                                    NRZ No run-time reports re use of
                                           ZERODIVIDE default values
                                    NXS No "extra" statements allowed

  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 sections 2.5 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.[63]

The option **W77** allows users to write a Fortran 77 program instead of a Fortran 90 TABLO-generated program. Details are given in 13.1.6 in GPD-3.

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 5.1.1 below).

---

[63] 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.)

5.1.1 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, ASSERTIONs, TRANSFERs, range checks and EQUATIONs plus, if there are any UPDATEs, carrying out multi-step simulations (see section 6.1 of GPD-3). [Though, on any run of the program, you can limit those actions actually carried out – see section 6.1.7 of GPD-3.]

You can write output for GEMSIM or a TABLO-generated program which is capable of carrying out less than the full range of actions 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 or the other actions listed in section 6.1.3 of GPD-3.)

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 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.[64]

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 4.13 above. If you select option

NRZ No run-time reports re use of ZERODIVIDE default values

the program cannot report these.

Normally "extra" TABLO-like statements are allowed in Command files. If you select option

NXS No "extra" statements allowed

---

[64] 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.

extra statements (see section 6.6 of GPD-3) will not be allowed on the Command file when you run GEMSIM or the TABLO-generated program.

When writing a Fortran 77 TABLO-generated program, TABLO has to make intelligent guesses as to how much memory to allocate for various parts of its processing. If you are using a Fortran 77 compiler such as the Lahey compiler F77L3, sometimes the memory allocated initially needs to be increased (see section 13.3 of GPD-3). 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 13.3 of GPD-3).

If you are using a Fortran 90 compiler (and writing a Fortran 90 TABLO-generated program) the program will allocate the memory automatically. In this case the option CDM has no effect.

The two options **SPL** and **SMD** which were provided only to maintain compatability with earlier releases of GEMPACK were removed in Release 5.2.

5.1.2 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.

This subsection only applies to Fortran 77 TABLO-generated programs. [If you are writing a Fortran 90 TABLO-generated program, which you would normally do if you have a Fortran 90 compiler, the options in this section are not relevant.]

Up to, and including Release 6.0, we provided options which tell TABLO to write code which uses less memory. These are the options

LMC Low memory for coefficients

ECS Equations and coefficients share memory

UCS Updated coefficients share memory between themselves

PCS Low memory for previous step coefficients

For Release 7.0 (or later), these options are no longer supported. [They are not very important now that memory is relatively plentiful and inexpensive. We judge that the cost of continuing to support them was no longer justified.] If this causes you a problem, please email Ken.Pearson@buseco.monash.edu.au and ask for advice.[65]

5.1.3 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

CIN Code file name same as Information file name

---

[65] More details about these options can be found in section 5.10.2 of the Release 5.1 edition of GPD-2 "User's Guide to TABLO, GEMSIM and TABLO-generated Programs", GPD-2, Second edition, April 1994. Note also that options SCS, LIR and LRP were removed in Release 5.2.

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.[66]

5.2 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 2.1.1 and 2.6.1 of GPD-1. As stated in section 2.6.1.2 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. If you are working on a Windows PC, details can be found in GPD-6.

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 section 2.8.1 of GPD-1 and chapter 2 of GPD-3 for details about Command files for running TABLO-generated programs.

[66] Option OCS was removed in Release 7.0.

CHAPTER 6

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, or, equivalently, put xwrite and xdisplay statements (see section 6.6 in GPD-3) in your Command 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, or xwrite and xdisplay statements in your Command 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 sections 6.1.9 and 14.4 of GPD-3); but this should be used sparingly since it may produce vast amounts of output.

Examine the Equations file

The program SUMEQ (see chapter 13 of GPD-4) 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 13.1.1 of GPD-4, 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 13.1.2 of GPD-4.

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

- that it is balanced in various ways (that is, that various accounting identities hold),
- 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

$$SUM(i,S, V_i) = SUM(j,T, W_j) \qquad (1)$$

where $V_i = P_i . X_i$ and $W_j = Q_j . Y_j$ are flows (dollar values) each a price (P_i or Q_j) times a quantity (X_i or Y_j).

Suppose (as usual) that the database contains the dollar values V_i and W_j.

Now

$v_i^* = p_i^* + x_i^*$ and so $v_i = p_i + x_i$. Similarly, $w_j = q_j + y_j$.

The updated values are $V_i' = V_i(1+p_i+x_i)$ and $W_j' = W_j(1+q_j+y_j)$.

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

$$SUM(i,S,[P_i.X_i/C](p_i + x_i)) = SUM(j,T, [Q_j.Y_j/C](q_j + y_j)) \qquad (2)$$

where C is the total of either side of (1) above.

It follows easily that

$$SUM(i,S, V_i') = SUM(i,S, V_i(1+p_i + x_i))$$
$$= SUM(i,S, V_i) + SUM(i,S, V_i(p_i + x_i))$$

$= SUM(j,T, W_j) + SUM(j,T, W_j(q_j + y_j))$ by (2)

$= SUM(j,T, W_j')$

and so the updated data is balanced.

Example (ii) - Income tax

E = household expenditure

WH = factor income (wage rate * 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.

Suppose that the data base contains

E = expenditure and B = WH = pretax income.

Then

C = WH(1-T) = tax paid

b = w + h

C = WH(1-T) = tax paid

c = [WH/C](w+h) - [WHT/C](w+h+t)
= [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)
= C + B(w+h) - (B-C)(w+h+t)
= C(1+w+h+t) – Bt
= C(1+e) - Bt by (3) above.

Thus B' - C' = B(1+w+h+t) - C(1+e)
= B(1+e) - C(1+e) by (3) above
= (B-C)(1+e) = E(1+e) = E'

so the updated data is balanced.

[Note however that

T' = 1 - C'/B' = (B' - C')/B
= {E(1+e)}/{B(1+w+h)}
= {E(1+w+h+t)}/{B(1+w+h)}
= T(1+w+h+t)/(1+w+h)
= T[1+t/(1+w+h)]

which is not exactly equal to T(1+t).]

Implementation B.

Suppose now that the data base contains

E = expenditure
and
B = WH = pretax income.

Then 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

- your balance conditions are **linear** in values held in the data base, and
- the expressions in the update statements are **linear** in **percentage-change** model variables,

you can be sure that

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

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. We are grateful to Peter Wilcoxen for encouraging and assisting us to implement this method for solving intertemporal model in GEMPACK.

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 chapter 1 of GPD-8). 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,t,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 and Set Elements - Fixed or Determined at Run Time

This section complements section 4.6.1 which deals with these topics for all sets.

- 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**.

For example, if the coefficient NINTERVAL has the value 5 at run time, then the elements of the set alltimex in the example above would be
time[0], time[1], time[2], time[3], time[4] and time[5]
since the intertemporal element stem is “time”.

- 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, p6, 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 when GEMPIE or ViewSOL reports 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 chapter 1 of GPD-8).

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) ;
```

7.4 Enhancements to Semantics for Intertemporal Models

Ths isection describes enhancements made in Release 6.0 (October 1998).

Up till (and including) Release 5.2 (September 1996), there have been some limitations in the semantics of expressions allowed in Formulas, Equations and Updates.

Example 1

```
Set (Intertemporal) S1_6 Size 6 ( p[1] - p[6] ) ;
Set (Intertemporal) S2_4 Size 3 ( p[2] - p[4] ) ;
Coefficient (All,s,S1_6) C1(s) ;
Coefficient (All,s,S1_6) C2(s) ;
Read C2 from Terminal ;
Formula (All,s,S2_4) C1(s) = C2(s+1) ;
```

In this example, the last statement (the Formula) would produce a semantic error indicating that 's' on the RHS is expected to range over a subset of the set S1_6. In the example above this could be easily fixed by adding the statement

```
Subset S2_4 is subset of S1_6 ;
```

Example 2

```
Set (Intertemporal) S1_6 Size 6 ( p[1] - p[6] ) ;
Set (Intertemporal) S2_10 Size 9 ( p[2] - p[10] ) ;
Coefficient (All,s,S1_6) C1(s) ;
Coefficient (All,s,S2_10) C2(s) ;
Read C2 from Terminal ;
Formula (All,s,S1_6) C1(s) = C2(s+1) ;
```

With Release 5.2 this would generate a similar semantic error, namely that index 's' is expected to range over a subset of the set S0_6 (which it does not).

But "clearly" the last Formula in example 2 makes perfect sense (except to Release 5.2 TABLO) since 's' runs from p[1] to p[6] and so "s+1" runs from p[2] to p[7] all of which are in the domain S2_10 of coefficient C2.

In this second example, there is no way to "fix" the Formula by adding a Subset statement. Indeed S1_6 is not a subset of S2_10. [What is really happening is that adding "+1" to the elements of S1_6 always gives an element of S2_10.]

The enhancements (which are available in Release 6.0 or later) are to allow Formulas such as the last one in Example 2.

The enhancements may also allow you to reduce the number of Subset statements. The purpose of Subset statements it to facilitate index checking (see section 4.7). When indices in Formulas, Equations and Updates range over Intertemporal sets which have intertemporally-defined elements, associated Subset statements (to facilitate index checking as set out in section 4.7) are now not required.

TABLO tries to work out if the arguments (with or without an index offset) are valid. If it is unable to, it gives instructions to the TABLO-generated program or to GEMSIM to check at run-time that indices stay in range (taking into account offsets, if they are present). [See section 7.2.1 for information about intertemporal sets which have their elements defined intertemporally (as distinct from those intertemporal sets which have fixed elements).]

Of course the two sets in question must have the same intertemporal element stem (see section 7.2.1): for example, the formula in Example 1 would generate a semantic error if the set S2_4 were defined by

```
Set (Intertemporal) S2_4 Size 3 ( t[2] - t[4] ) ;
```

since its elements have stem 't' and those in S1_6 have stem 'p'.

However Subset statements are still required if the intertemporal sets in question have fixed elements. Thus, if the Set declarations in Example 1 were changed to

```
Set (Intertemporal) S1_6 Size 6 ( p1 - p6 ) ;
Set (Intertemporal) S2_4 Size 3 ( p2 - p4 ) ;
```

the formula in Example 1 would not be allowed (unless S2_4 was declared as a subset of S1_6). If the Set declarations in Example 2 were changed to

```
Set (Intertemporal) S1_6 Size 6 ( p1 - p6 ) ;
Set (Intertemporal) S2_10 Size 9 ( p2 - p10 ) ;
```

the formula in Example 2 would not be allowed and this could not be "fixed" by a subset statement since set S1_6 is not a subset of S2_10.

Subset statements are still required in conjunction with Reads, Writes and Displays, even when the sets involved are Intertemporal sets which have intertemporally-defined elements. Thus, the following statement

```
Write (all,s,S2_4) C2(s) to terminal ;
```

added at the end of Example 1 above will still produce a semantic error unless C2_4 is declared to be a Subset of C0_6. [Maybe future enhancements to TABLO will automatically generate the required Subset statement in cases such as this.]

Because Subsets are required for Reads and because Formula(Initial)s generate a Read statement (for steps 2,3... of a multi-step calculation), Subsets are still required for Formula(Initial)s.

Even when the sets in question have intertemporal elements, and subset statements are not required to facilitate Equations etc, subset statements may still be a good idea if you want to refer to different sets on Command files. For example, if you are working with Example 1 above modified so that the Coefficients there are Linear Variables (and the Formula there is an Equation) then you could say

```
exogenous C1(S2_4) ;
```

would only be allowed if S2_4 had been declared as a subset of S1_6 in the TABLO Input file.

7.5 Recursive Formulas over Intertemporal Sets

This section documents the implementation of formulas with an ALL index ranging over an intertemporal set (and points out a related bug which is now fixed). We are grateful to Robert McDougall for pointing out the bug and the gap in the documentation.

Consider an intertemporal set TIME with elements t1 to t10 and consider subsets of this, TIME1 with elements t1 to t9 and TIME2 with elements t2 to t10. Consider two coefficients C1 and D each with one argument ranging over the set TIME. Suppose also that values have already been assigned to D(t) for all t in the set TIME.

(a) Consider the formula

```
(all,t,TIME1)  C1(t+1) = C1(t) + D(t) ;
```

Note that, in TABLO-generated programs, loops over intertemporal sets are always carried out in order going from the first to the last element of the set. [That is, the formulas are carried out **forwards** in time.] Hence the formula above is carried out as 9 separate formulas (corresponding to the 9 elements of the set TIME1). First the formula is carried out for t=t1 so that C1("t2") is set equal to C1("t1")+D("t1"). Then it is carried out for t=t2 [which assigns a new value to C1("t3")] and finally for t=t9 (which assigns a new value to C1("t10")].

Provided that the value of C1("t1") is set before hand, this will calculate in turn the values of C1 at points t2, t3, …, t10 in the usual backwards-looking way often needed in an intertemporal model.

Note also that the most recent values are always used on the RHS. Thus, for example, when calculating the value of C1("t3"), it uses the value of C1("t2") which was calculated in the previous instance of the formula (that is, the formula with t=t1).

(b) Consider now the formula

```
(all,t,TIME2)  C1(t) = C1(t+1) + D(t) ;     ! not valid !
```

The intention of this is presumably to set the values of C1 working **backwards** in time. That is, provided that the value of C1("t10") is set before hand, this would calculate the values of C1("t9") from the values of C1("t10") and D("t9"), then the value of C1("t8") from this new value of C1("t9") and the value of D("t8"), and so on. For this to work out as described, TABLO-generated programs would need to run the loop over t backwards. They do not do this (as indicated above). Accordingly this formula would not be evaluated as intended.

It is a bug in Release 6.0 of GEMPACK that the formula above did not generate an error when TABLO processes it. This bug was fixed in Release 6.0-001 (March 1999).

(c) In formulas (whether the ALL qualifiers range over intertemporal or non-intertemporal sets), TABLO-generated programs always use the most recent values on the RHS. Consider the formula

```
(all,t,MIDTIME) C1(t) = [C1(t-1) + C1(t+1)] / 2  ;  ! wrong !
```

where MIDTIME is the set with elements t2 to t9. The intention of this formula is presumably to replace the values of C1(t) by the average of the previous and subsequent C1 values. This would not be achieved with the present implementation of formulas since, when t=t3, the value of C1("t2") used on the RHS would be the most recent one - that is, the one calculated by the above formula with t=t2.

At present, the simplest way of implementing the intention stated above would be to create a copy (say C2) of the values in coefficient C1 and then apply the formula above with C2 on the RHS. That is, replace the formula above by the formulas

(all,t,TIME) C2(t) = C1(t) ;

(all,t,MIDTIME) C1(t) = [C2(t-1) + C2(t+1)] ;

CHAPTER 8

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 Flexible Formula for the Size of a Set

Suppose that you need to know the size of a set in your TABLO Input file. In many cases, you may be using the same TABLO Input file with different aggregations of the model so do not want to hard-wire in the size. There is a simple way of calculating the size using SUM in a formula.

Suppose the set is called SET1. Then the following formula will calculate the size of the set.

```
Coefficient   SIZE_SET1   # Size of SET1 # ;
Formula   SIZE_SET1  =  SUM( s, SET1,  1) ;
```

[This is because the SUM adds one for every different element of SET1. Thus if SET1 has 100 elements, SIZE_SET1 will be set equal to 100.]

See section 4.13 for a practical use of this sort of formula in connection with ZeroDivide statements.

8.2 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.

8.3 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

```
                              [  G(i) + T(i)   if i is in SECT1,
  (all,i,SECT)  F(i) =        [
                              [  W(i) + V(i)   if i is in SECT2,
```

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 4.14.6 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.

CHAPTER 9

9. Linearizing Levels Equations

In this chapter we state in section 9.1 the differentiation rules used by TABLO when it differentiates levels equations in TABLO Input files. We also discuss in section 9.2 how you might go about linearizing equations by hand if you want to put linearized equations directly into a TABLO Input file.

9.1 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 9.2 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 or C*pC = A*pA + 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.

See section 2.2.4 for more about the different ways of linearizing a sum.

9.2 Linearizing Equations by Hand

We introduce the general procedure used in section 9.2.1. In section 9.2.2, we write down a list of helpful rules you can use and give examples of their use in section 9.2.3. In section 9.2.4 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.

9.2.1 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 , f and f 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 of (1).

$$f_P.P.p_P/100 + f_Q.Q.p_Q/100 + f_R.R.p_R/100 + .. = 0 \qquad (6)$$

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 9.2.2 below) for linearizing a product such as (7).

9.2.2 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 or
R*p_R = P*p_P + Q*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 9.2.2: 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. [See also section 2.2.4.]

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 9.2.1 above.

9.2.3 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.

9.2.4 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.

9.3 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. Details about the way in which TABLO linearizes levels equations can be found in section 2.2.

Suppose you are using AnalyseGE (see section 2.6 of GPD-4) to work with a model whose TABLO Input file contains levels equations. Then you can ask AnalyseGE to copy the linearized equation associated with any levels equation to the AnalyseGE form – this is one of the popup menu options when you right click on a levels equation.[67]

```
130   FORMULA & EQUATION Comin
131      # Intermediate input of commodity i to industry j #
132   (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) ;  !

133
134   FORMULA & EQUATION Facin
135      # Factor input f to industry j #
136   (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) ;  !

137
138   FORMULA & EQUATION House
139      # Household demand for commodity i #
140   (all,i,SECT) XH(i) = DVHOUS(i) / PC(i) ;
   !  EQUATION(LINEAR) House
      (ALL,i,SECT) p_XH(i) = p_DVHOUS(i) - p_PC(i) ;  !

141
142   FORMULA & EQUATION Com_clear ! (E3.1.6) in DPPW !
143      # Commodity market clearing #
144   (all,i,SECT) XCOM(i) = XH(i) + SUM(j,SECT,XC(i,j)) ;
   !  EQUATION(LINEAR) Com_clear
      (ALL,i,SECT) XCOM(i) * p_XCOM(i) = XH(i) * p_XH(i) +
          SUM(j,SECT,XC(i,j) * p_XC(i,j)) ;  !
```

[67] AnalyseGE extracts this linearized equation from the Information file.

CHAPTER 10

10. New TABLO Syntax, Qualifiers

In this chapter we show all the TABLO qualifiers and statements which are new for Release 7.0 or were new for Release 6.0 or 5.2.

10.1 New TABLO Statements and Qualifiers for Release 7.0

```
TRANSFER <header> FROM FILE <logical-file1> TO FILE <logical-file2> ;
TRANSFER UNREAD FROM FILE <logical-file1> TO FILE <logical-file2> ;
TRANSFER UNWRITTEN FROM FILE <logical-file1> TO FILE <logical-file2> ;
TRANSFER <header> FROM FILE <logical-file1> TO UPDATED FILE ;
TRANSFER UNREAD FROM FILE <logical-file1> TO UPDATED FILE ;
      see sections 3.15 and 4.12 about Transfer statements

FORMULA(BY_ELEMENTS)       see section 4.8.11

New functions  RANDOM, NORMAL  and  CUMNORMAL  (see section 4.4.4)
```

10.2 New TABLO Statement Qualifiers for Release 6.0

```
Variable (ORIG_LEVEL = <coefficient-name>)...
     ! New variable qualifier      (see section 3.4)

Coefficient (<operator> <real number >)...
       where operator can be GE, GT, LE, or LT
       A new Coefficient or Levels Variable qualifier   .
       (see section 3.3)
       EXAMPLE COEFFICIENT(GE 20.0) (all,i,COM)  DVHOUS(i) ;

Write (SET) <setname> to file <logical-file> [ HEADER "<header>"]  ;
        This can now be used to write to a Header Array file or
        a text file not just a GAMS file - see section 4.6.6

Write (ALLSETS) to file <hafile> ;
        Do not specify a Header
        New Write qualifier - see section 4.6.7
```

10.3 New TABLO Statement Qualifiers for Release 5.2

```
FOR_UPDATES      new FILE qualifier (see section 3.5)
WRITE UPDATED VALUE TO ...     FORMULA qualifier (see section 4.11.7)
FILE (NEW,GAMS)      see section 3.5
WRITE (SET)          see section 4.6.6
```

10.4 New TABLO Syntax for Release 6.0

```
Union and Intersection of Sets (section 4.6.3)
SET <setname> [#<label-information>#] = <setname1> UNION <setname2> ;
SET <setname> [#<label-information>#] = <setname1> INTERSECT <setname2>;
```

10.5 New TABLO Syntax for Release 5.2

Set complements (section 4.6.4)

```
      SET  <new-setname>  =  <bigset> -  <smallset> ;
      Example. SET NONMARCOM = COM - MARCOM ;
```

Sets depending on data (section 4.6.5)

```
SET  <new-set>  = (All, <index>, <old-set>: <condition> ) ;
Example. SET SPCOM = (all,c,COM: TOTX(c) > TOTY(c)/5 ) ;
```

Set Mappings (section 4.8)

```
MAPPING  <set-mapping>  FROM  <set1>  TO  <set2>  ;
Example. MAPPING COMTOAGGCOM from COM to AGGCOM ;
```

```
READ (BY_ELEMENTS) <set-mapping> FROM FILE … ;
! expects character data !
READ <set-mapping> FROM FILE … ;
! expects integer data - positions in the set
    see section 4.8.1 !
Example.
  READ (BY_ELEMENTS) COMTOAGGCOM from file setinfo header “CTAC” ;
  READ COMTOAGGCOM from file textinfo ; ! textinfo is a text file
```

```
WRITE <set-mapping> TO FILE … ;
! integer output (numbers) !
Example.  WRITE COMTOAGGACOM to file textoutput ;
  ! could be to HA file or terminal !
```

Assertions (section 3.14)

```
ASSERTION  [<qualifiers>] [# message #]
      [<quantifier-list>]<condition> ;
Example.
ASSERTION # Check no negative values #
      (all,c,COM) DVHOUS(c) >= 0 ;
```

Index expressions (section 4.4.9)

```
 <index-expression-1> <comparison-operator> <index-expression-2>
Example.
AGGDHOUS(aggc) =
             SUM(c,COM: COMTOAGGCOM(c) EQ aggc, DHOUS(c) ) ;
```

$POS function (section 4.4.4)

```
$POS(<index>)
$POS(<index>,<set2>)
$POS(<element>,<set>)
```

Writing updated values (section 4.11.7)

```
FORMULA(INITIAL, WRITE UPDATED VALUES TO FILE <filename>
HEADER "<HEADER>"  LONGNAME "<LONGNAME>") <formula> ;
Example.
FORMULA (INITIAL, WRITE UPDATED VALUE TO FILE upd_prices
 HEADER "ABCD" LONGNAME "<words>" )
    (all,c,COM) PHOUS(c) = 1 ;
```

11. REFERENCES

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. [*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 Microeconomc 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.

Harrison, Jill and Ken Pearson (1999), Adding Accounting-Related Behaviour to a Model Implemented Using GEMPACK, *available on the FAQ page of the GEMPACK web site*, September 1999.

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.

Press, W.H., B.P. Flannery, S.A. Teukolsky and W.T. Vetterling (1986), *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge.

12. GEMPACK DOCUMENTS[68]

Harrison, W.J. and K.R. Pearson (2000), *An Introduction to GEMPACK*, GEMPACK Document No. 1 **[GPD-1],** Monash University, Clayton, Fifth edition, October 2000.

Harrison, W.J. and K.R.Pearson (2000), *TABLO Reference*, GEMPACK Document No. 2 **[GPD-2]** Monash University, Clayton, Third edition, October 2000.

Harrison, W.J. and K.R.Pearson (2000), *Simulation Reference: GEMSIM, TABLO-generated Programs and SAGEM*, GEMPACK Document No. 3 **[GPD-3]**, Monash University, Clayton, October 2000.

Harrison, W.J. and K.R.Pearson (2000), *Useful GEMPACK Programs*, GEMPACK Document No. 4 **[GPD-4]** Monash University, Clayton, October 2000.

Harrison, W.J. and K.R. Pearson (2000), *Installing and Using the Source-Code Version of GEMPACK on DOS/Windows PCs with Lahey Fortran*, GEMPACK Document No. 6 **[GPD-6],** Monash University, Clayton, Tenth edition, October 2000.

Harrison, W.J. and K.R. Pearson (2000), *Installing and Using the Executable-Image Version of GEMPACK on DOS/Windows PCs*, GEMPACK Document No. 7 **[GPD-7],** Monash University, Clayton, Seventh edition, October 2000.

Harrison, W.J. and K.R. Pearson (2000), *Getting Started with GEMPACK: Hands-on Examples*, GEMPACK Document No. 8 **[GPD-8],** Monash University, Clayton, Second edition, October 2000

[68] 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".

13. INDEX

$

$POS, 4-57
  Function, 4-55

A

Absorption
  Meaning of, 2-19
Accumulate, 8-123
Actions, 5-110
Active index. *See* Index
Actual filename, 4-88
  Length of, 4-49
Adding equations, 1-2
Aggregating Data
  Mapping, 4-79
Aggregating simulation results, 4-79
ALL
  Syntax, 4-53
ALL index
  Active index, 4-60
ALLSETS
  Qualifier, 3-32
AnalyseGE, 9-129
  Looking at linearized equations, 9-129
Arguments, 4-51
  Indices in expressions, 4-60
  Involving set mappings, 4-84
ASSERTION
  Statement, 4-47
  Syntax, 3-41
Auxiliary files, 2-7

B

Backsolve
  Requirements, 2-16
  System-initiated, 2-18
Balanced equations, 6-113
Base data
  Coefficient, 3-27
  Data file, 3-30
  Different, 4-88
Brackets
  Use of, 4-54
BY_ELEMENTS, 3-31
  Mapping, 3-40

C

Case, 4-48
Change
  Rules, 9-125
  Variable, 3-28
Change Update, 3-37, 4-95
Change updates, 4-97
Character data
  For set elements, 4-71, 4-76
  For set mapping, 4-80
  Reading, 4-88
Check
  ASSERTION, 3-41
  TABLO, 2-5
Check stage, 2-5
Checking models, 6-113
Code
  TABLO, 2-5
Code parameter, 4-63
Code stage, 2-5
Codomain of mapping, 4-79
Coefficient
  Active index, 4-60
  Argument - Index Expression, 4-83
  Arguments, 4-51
  Assign values, 4-106
  Associated data, 4-90
  Declaring, 4-104
  Dimension, 3-27
  Divide by zero, 4-102
  Example argument, 4-51
  How used, 4-63
  Integer, 3-27, 4-64, 7-120
  Integer, Display, 3-39
  Meaning, 4-63
  Parameter, 3-27
  Partial reads, 4-90
  Real, 3-27
  Scalar, 3-38
  System-initiated, 2-18
  Types, 4-65
  Upper case, 4-48
COEFFICIENT
  Statement, 4-47
  Syntax, 3-27
Coefficient initialisation, 4-92
Coefficient name, 4-49
  Length of, 4-49
Coefficients
  Labelling information, 4-93
  Values in equations, updates, 4-106
Column order, 3-30, 4-89
Command files
  Extra statement, 3-45
  TABLO-like statements, 3-45
Comment
  Strong, 4-48
Comments, 4-48
Comparing indices, 4-61
Comparison operators, 4-58
Compile, Link, 5-112
Complement
  Set, 3-25, 4-73
Composition of Mappings, 4-83
Condensation, 2-8
  Examples, 2-15

Condense
TABLO, 2-5
Condense stage, 2-5
Conditional
IF, 4-58
Conditional expressions, 2-18, 4-58
Conditional function, 8-123
Conditional operators, 4-57
Conditions, 4-53
In ALLs or SUMs, 4-57
Constants, 4-60
CUMNORMAL function, 4-55, 4-56

D

Data
Associated with coefficients, 4-90
Data file
Transfer, 4-100
Default statements
Meaning, 3-43
Syntax, 3-43
Default values of qualifiers, 3-43
Default WFP, PGS, 5-109
Differentiation
Rules, 9-125
Dimension
Coefficient, 3-27
Dimension of a variable, 3-28
Display
Checking models, 6-113
Integer coefficient, 3-39
Partial, 4-90
DISPLAY
Statement, 4-47
Syntax, 3-39
Display file, 3-39, 4-93
Display files, 3-39
Divide by zero
Shares, 4-102
ZERODIVIDE, 3-38
Dividing by zero
Using ID01 to protect against, 4-55
Division, 4-64
Domain of mapping, 4-79

E

Elimination, 2-18, 4-107
Empty sets, 4-74, 4-76
Equation
Active index, 4-60
Block, 4-105, 6-113
Expressions in, 4-53
Levels, 3-35
Linear, 3-35
Linear variables, 4-59
Order of, 4-105
EQUATION
Statement, 4-47
Syntax, 3-35
Equation file
Variable order, 4-105
Equation name, 3-35, 4-49
Length of, 4-49
EQUATION(NONE) statement, 3-35, 4-108
Equations
Values of coefficients, 4-106
Equations file
SUMEQ, 6-113
Equations Matrix
Variable order, 4-104
Errors, 2-5
Identifying, 2-8
Exclamation mark, 4-48
Exponent, 4-53, 4-60
Exponential function, 4-55
Expression, 4-53
Brackets, 4-54
Complicated, 4-107
Conditional, 4-58
Constants in, 4-60
Operators in, 4-53
SUM, 4-54
Extra statement
Space before qualifier, 3-45

F

File
Actual name, 3-30
Filenames, 4-88
Header Array, 3-30, 4-88
Information, 2-5
Logical name, 3-30
Reading, 3-31
Text, 3-30, 4-88
FILE
FOR_UPDATES, 3-30, 4-98
GAMS, 3-30
Statement, 4-47
Syntax, 3-30
Writing, 3-32
First stage, 2-5
Fixed elements
Definition, 4-71
Flexible style
Element Names, 4-76
Formula
Active index, 4-60
Always, 3-34
Assign values, 4-106
By_elements, 4-86
Complicated, 4-107
Divide by zero, 4-102
Division in, 4-59
Expressions in, 4-53
For size of a set, 8-123
Initial, 3-34
Model checking, 6-113
Order, 4-107
Order of, 4-106
System-initiated, 2-18
Values in equations, 4-106
Zero denominator, 3-38
Zerodivide default, 4-102
FORMULA
BY_ELEMENTS, 3-40

Mapping, 3-40
Statement, 4-47
Syntax, 3-34
FORMULA & EQUATION, 3-36
FORMULA(INITIAL)
Restriction, 4-92
WRITE(UPDATED), 4-98
Formulas
recursive, 7-122
Fortran 77, 4-108
Fortran 90
Memory management, 2-20
Free form, 3-21, 4-48
Function
CUMNORMAL, 4-55, 4-56
NORMAL, 4-55, 4-56
RANDOM, 4-55
Functions, 4-55

G

GAMS file, 3-30
Gap in intertemporal set, 7-119
GEMPIE
Levels results, 4-69
GEMPIE Print file, 4-104
GEMSIM
Actions, 5-110

H

Header
Length of, 4-49
Header Array file, 3-30, 4-88, 4-90

I

ID01 function, 4-55
ID01 function
Use, 4-55
ID0V function, 4-55
IF
Syntax, 4-58
IF statement, 2-18
INCLUDE file, 2-20
Increasing parameters, 2-20, 5-111
Index
ALL, 4-53
Coefficient, 3-27
Inactive, 4-60
Range of, 4-51
SUM, 4-54
Variable, 3-28
Index name, 4-49
Index expression
Comparison, 4-61
Definition, 4-51
Index Expression, 4-83
Index expression comparison, 4-79
Index name
Length of, 4-49
Index offset, 2-16, 4-51, 4-91, 4-92
Indices
As Arguments, 4-51
Examples, 4-60
In declarations, 4-51
In equations and formula, 4-60
Maximum number, 3-27, 3-28
Order of, 4-105
Information, 4-48
Labelling, 4-50
Information file, 2-5
Initial formula
Partial, 4-92
Initial Formula, 3-34
Input statements, 4-48, 4-51, 4-78
General syntax, 4-47
Input Statements
Order of, 4-104
Integer
Coefficient, 3-27
Integer coefficient, 4-64
Display, 3-39
Integer Coefficient, 7-120
Integer constant, 4-60
Intermediate extra data file, 4-92
Intersection
Set, 3-25, 4-72
Intertemporal, 3-24
Example sets, 7-117
Intertemporal element stem, 7-119
Intertemporal model, 2-16
Intertemporal models, 7-117
recursive, 7-122
Semantics, 7-120
Intertemporal set, 4-51
Elements, 7-119
No gaps, 7-119
Intertemporal sets
Meaning, 7-117
Run-time elements, 4-71, 7-119
Intertemporally defined, 7-119

K

Keyword, 3-21, 4-47
Space before qualifier, 3-45

L

Labelling information
Coefficients, 4-93
Labelling Information, 4-50
Last stage, 2-5
Levels equation, 3-35
Meaning of, 4-59
Levels equations, 1-2
Levels models, 4-68
Levels results
GEMPIE, 4-69
SLTOHT, 4-69
ViewSOL, 4-69
Levels values
Reporting, 4-68
Levels variable, 3-27
Levels variables, 4-65
Pre and post-simulation, 4-68
Line length

TABLO Input file, 4-48
Linear equation, 3-35
Meaning of, 4-59
Linear Variable
Meaning of, 4-59
Linearization rules, 9-125
Linearization Rules, 9-127
Linearized versions of levels equations, 9-129
via AnalyseGE, 9-129
Linearizing a sum, 2-13
Linearizing sum
Changes form, 2-13
Share form, 2-13
List of TABLO statements, 3-21
Logarithmic function, 4-55
Logical filename, 4-88
Logical filename, 4-49
Length of, 4-49
Logical files, 3-31, 3-33
Logical operators, 4-58
Longnames
Write statements, 4-93
Lower case, 3-22, 4-48

M

Mapping, 4-79
By_elements, 3-40, 4-86
Codomain, 4-79
Domain, 4-79
Onto, 4-83
MAPPING
Statement, 4-47
Syntax, 3-40
Maximum size
Set, 4-72
Memory
Code options, 5-111
Memory Allocation
Fortran 90, 2-20
Model parameter, 4-63
Model verification, 6-113
Modifying a model, 1-2

N

Names
User-defined, 4-49
Non_parameter
Coefficient, 3-27, 4-95, 4-96
Normal distribution, 4-55, 4-56
NORMAL function, 4-55, 4-56

O

Offset. *See* Index offset
Omitting variables, 2-19
Onto Mapping, 4-83
Operation, 4-53
Options
Basic, 2-5
GEMPIE - RPO, 4-104
GEMPIE - NLV, 4-69
GEMSIM and TG program - DWS, 4-64
GEMSIM and TG program - NXS, 5-111
GEMSIM and TG programs - DWS, 6-113
SLTOHT - SHL, 4-69
TABLO, 2-5
TABLO ACD, 2-14
TABLO - NTX, 2-7
TABLO - NWT, 2-6
TABLO SCO, 2-7
TABLO Code, 2-8, 5-109
TABLO Code - ACC, 5-111
TABLO Code - CDM, 5-111
TABLO Code - CIN, 5-111
TABLO Code - DMS, 5-110
TABLO Code - ECS, 5-111
TABLO Code - LIR, 5-111
TABLO Code - LMC, 5-111
TABLO Code - Low memory, 5-111
TABLO Code - LRP, 5-111
TABLO Code - NDS, 5-110
TABLO Code - NEQ, 5-110
TABLO Code - NMS, 5-110
TABLO Code - NRZ, 5-110
TABLO Code - NWR, 5-110
TABLO Code - OCS, 5-112
TABLO Code - PCS, 5-111
TABLO Code - PGS, 5-109
TABLO Code - SCS, 5-111
TABLO Code - SMD, 5-111
TABLO Code - SPL, 5-111
TABLO Code - UCS, 5-111
TABLO Code - W77, 5-109
TABLO Code - WFP, 5-109
TABLO Menu, 2-5
ViewSOL, 4-69
Options GEMPIE - SNA, 4-69
Order of READs,FORMULAs,EQUATONs, 4-106
ORIG_LEVEL
Qualifier, 3-28
Original Levels values, 3-28

P

Parameter
Coefficient, 3-27
Model, 4-63
Parameter values
TABLO, 2-20
TABLO-generated programs, 5-111
Parameters, 4-96
Parial initialisation, 4-92
Parial read, 4-92
Partial display, 4-90
Partial read, write, 4-90
Percent Change
Variable, 3-28
Percentage change
Rules, 9-125
Position in set
Function, 4-55
Position Number
$POS, 4-57
Post-simulation coefficients, 4-68
Power of tax, 4-98
Pre-simulation coefficients, 4-68

Print file
Variables, 4-104
Product Update, 3-37
Product updates, 4-98
Programs
SAGEM, 4-104
SUMEQ, 6-113

Q

Qualifier, 3-21
Space before, 3-45
Warning, 4-47
Qualifiers
Summary, 3-44
Quantifier, 4-53
List, 4-53

R

RANDOM function, 4-55
Not available on some machines, 4-56
Random numbers, 4-55
Range of values, 3-27, 3-28
Range test, 3-27, 3-28
Read
Data into coefficients, 4-90
Filename, 4-88
Header Array file, 4-88
Order of, 4-106
Partial, 4-90
Values in equations, 4-106
READ
BY_ELEMENTS, 3-40
Mapping, 3-40
Statement, 4-47
Syntax, 3-31
Read style
Element names, 4-76
Real
Coefficient, 3-27
Real constant, 3-38, 4-49, 4-60
Recursive formulas, 7-122
Reserved characters, 4-48, 4-49
Reserved words, 4-49
Row order, 3-30, 4-89
Run-time elements, 4-71
Definition, 4-71
Intertemporal sets, 4-71, 7-119

S

Scalar, 3-38
Semantic check, 4-92
Semantic description, 4-47
Semantic errors, 2-8
Semantic Errors
Removing, 2-8
Semantic problems, 4-60
Separator
Spreadsheet, 3-30
Set
Associated data, 4-90
Complement, 4-73
Data dependent, 4-74
Declaring, 4-104
Empty, 4-74
Index Range, 4-51
Intersection, 4-72
Intertemporal, 3-24, 4-78, 7-120
mapping, 4-79
Maximum size, 4-72
Position Number, 4-57
Union, Intersection, 4-72
Writing elements, 4-74
SET
Complement, 3-25
Data dependent, 3-25
Intersection, 3-25
Statement, 4-47
Syntax, 3-23
Union, 3-25
Set element name
As Argument, 4-51
Length of, 4-49
Reading, 4-71, 4-76, 4-88
Set elements
Arguments, 4-51
Declaring, 4-104
Determined at run time, 4-52
Fixed, 4-71
Fixed names, 4-52
In subsets, 4-78
Inside quotes when arguments, 4-51
Not named, 4-71
Read Style, 4-76
Reading, 4-71, 4-76, 4-88
Run time, 4-71
Syntax, 3-23
Set mapping
in arguments, 4-84
Reading, 4-88
Semantics, 4-84
Set Mapping
Aggregating data, 4-79
Aggregating simulation results, 4-79
Set name, 4-49
Length of, 4-49
Set size
Fixed, 4-71
Formula for, 8-123
Run-time, 4-71, 7-118
Sets
Empty, 4-76
Intertemporal, 7-117
Shares, 2-13
Simulations
Results, 4-104
Size of set
Formula for, 4-102, 8-123
SLTOHT
Levels results, 4-69
Speed of solution
Code options, 5-111
Splitting variables, 2-18
Spreadsheet, 3-30
Square root function, 4-55
Stages of TABLO

Check, 2-5
Code, 2-5
Condense, 2-5
First, 2-5
Last, 2-5
Strong comment, 4-48
Subset
By elements, 4-78
By numbers, 4-78
Declaring, 4-104
Index Range, 4-51, 4-78
Input of elements, 4-78
SUBSET
Statement, 4-47
Syntax, 3-26
Substitution, 2-15, 2-18, 4-106
Looking ahead to, 2-17
Order of, 2-17
Requirements, 2-16
Sum
Linearizing, 2-13
SUM
Syntax, 4-54
SUM index, 4-60
Syntax error, 2-5
Syntax description, 3-21, 4-47, 4-49
Syntax errors, 2-8
Syntax Errors
Removing, 2-8
System-initiated, 2-18

T

Tableau, 4-105, 6-113
TABLO
Basic options, 2-5
Parameter values, 2-20
Stages, 2-5
TABLO Syntax, 3-21
TABLO Code Options
Menu, 2-8
TABLO Include files, 2-20
TABLO Input file
Actions, 5-110
Data manipulation, 3-35, 4-108
Equation order, 4-105
Formula, read order, 4-106
Index order, 4-105
Input statements, 4-47
Maximum line length, 4-48
No equations, 3-35, 4-108
Statement order, 4-104
Substitution, 2-17
Variable order, 4-104
Zerodivide, 4-102
TABLO language
Examples, 8-123
Free form, 3-21, 4-48
TABLO Options, 2-5
Menu, 2-5
TABLO style
Element Names, 4-76
TABLO-generated programs
Actions, 5-110
Parameter values, 5-111
TABLO-like statements. *See* Command files
TABmate, 1-2, 1-3, 2-9
Tax
ad valorem, 4-98
Power, 4-98
Terminal
Read, 3-31
Write, 3-32
Text file, 3-30, 4-88
TEXTBI, 2-6, 2-7
Time periods
Accumulating, 8-123
TINY
Protecting against divison by zero, 4-55
TRANSFER
Header Array, 4-100
Statement, 4-47
Syntax, 3-42
Unread, 3-42
Unwritten, 3-42

U

Union
Set, 3-25, 4-72
Update, 4-95
Change, 3-37, 4-60, 4-97
Deriving, 4-97, 4-98
Example, 4-97, 4-98
Expressions in, 4-53
Order of, 4-106
Product, 3-37, 4-60, 4-98
Semantics, 4-96
Which type, 4-95
UPDATE
Statement, 4-47
Syntax, 3-37
UPDATE(CHANGE), 4-95
Updated values
FORMULA(INITIAL), 4-98
Writing, 4-66
Upper case, 3-22, 4-48

V

Variable
Active index, 4-60
Argument - Index Expression, 4-83
Arguments, 4-51
Change, 3-28
Component order, 4-105
Dimension, 3-28
Division by, 4-59
Example argument, 4-51
Levels, 3-28
Linear, 3-28
Lower case, 4-48
Order on Print file, 4-104
Percentage Change, 3-28
Substitution, 4-106
VARIABLE
Statement, 4-47
Syntax, 3-28

Variable name, 4-49
  Length of, 4-49
Variable Order on Print file, 4-104
VARIABLE(ORIG_LEVEL=..), 4-68
Variables
  SAGEM, 4-104
  Use in equations, 4-59
ViewHAR
  Create TABLO Code, 1-3
ViewSOL
  Levels results, 4-69

W

Write
  All sets, 4-75
  Checking models, 6-113
  Header Array file, 4-88
  Partial, 4-90
  Set, 4-74
  Updated value qualifier, 3-34
WRITE
  BY_ELEMENTS, 3-40
  Mapping, 3-40
  Statement, 4-47
  Syntax, 3-32
  Updated values, 4-98
Writing SETS, 3-32
Writing updated values, 4-66

X

XSet
  Complement, 4-74
  Intersection, 4-73
  Union, 4-73
XTransfer statements, 4-101

Z

Zerodivide
  Default, 4-102
  Real constant, 4-49
  Reports, 4-103
  Use of, 4-102
ZERODIVIDE
  Statement, 4-47
  Syntax, 3-38