Solving MPSGE Models Using GEMPACK[1]

14 July 2004

Laurent Cretegny[*], Mark Horridge, Ken Pearson
Centre of Policy Studies, Monash University

Thomas Rutherford
Department of Economics, University of Colorado

[1] A short and preliminary version of this paper dated May 2004 was presented at the GTAP conference in Washington D.C. in June 2004.

[*] This author gratefully acknowledges financial support from the Swiss National Science Foundation (post-doctoral research fellowship).

Abstract

MPSGE is a non-algebraic language for the formulation of applied general equilibrium models. When you build a model using MPSGE you do not need to specify equations, but you instead work with a tabular representation of the model. The input tables make reference to source statistics, typically drawn from a single benchmark year. MPSGE input data describing your model characterize technology, preferences, tax rates, and factor endowments. These data, together with a set of equilibrium conditions, define the equilibrium framework.

MPSGE is a framework designed primarily to permit rapid prototyping of applied models. The compact, non-algebraic format can substantially improve productivity through the automated generation of demand and supply functions. This approach reduces the scope for errors and simplifies sensitivity analysis of results with respect to model structure. To take advantage of the MPSGE framework, a user must learn the syntax and conventions of the MPSGE language.

The purpose of this paper is to introduce and document a new program (called MGE2GP) that can be used to convert MPSGE representations of models to GEMPACK. If you write down a model in MPSGE form, and create a data file for the model, you can use this program to convert your model to GEMPACK and then use GEMPACK software to solve the model. This paper contains detailed instructions for carrying out these steps.

The paper contains a self-contained introduction to the MPSGE language, illustrated via several example models. It also contains detailed hands-on instructions for solving these example models using GEMPACK. The aim is to make the paper accessible to all general equilibrium modellers. In particular, we do not assume that readers are already familiar with MPSGE or GEMPACK. Many of the example models can be solved using the Demonstration Version of GEMPACK which can be downloaded (at no cost) from the web.

1. Introduction

MPSGE[2] is a *non-algebraic* language for the formulation of applied general equilibrium models. When you build a model using MPSGE you do not need to specify equations, but you instead work with a tabular representation of the model. The input tables make reference to source statistics, typically drawn from a single benchmark year. MPSGE input data describing your model characterize technology, preferences, tax rates, and factor endowments. These data, together with a set of equilibrium conditions, define the equilibrium framework.

MPSGE is a framework designed primarily to permit *rapid prototyping* of applied models. The compact, non-algebraic format can substantially improve productivity through the automated generation of demand and supply functions. This approach reduces the scope for errors and simplifies sensitivity analysis of results with respect to model structure. To take advantage of the MPSGE framework, a user must learn the syntax and conventions of the MPSGE language.

The purpose of this paper is to introduce and document a new program (called **MGE2GP**[3]) that can be used to convert MPSGE representations of models to GEMPACK. If you write down a model in MPSGE form, and create a data file for the model, you can use this program to convert your model to GEMPACK and then use GEMPACK software to solve the model. This paper contains detailed instructions for carrying out these steps.

MPSGE was developed by Rutherford in the 1980s (Rutherford, 1985, 1987).[4] Until now MPSGE has been used only by GAMS modellers who use the MPSGE subsystem of GAMS (Brooke *et al*, 1988, 1998) to formulate the equations of their models and then solve their models using the MILES or PATH algorithms (Rutherford, 1995, 1999). Now, with the development of MGE2GP, it is possible also to solve MPSGE models using GEMPACK.

The paper contains a self-contained introduction to the MPSGE language, illustrated via several example models. It also contains detailed hands-on instructions for solving these example models using GEMPACK. The aim is to make the paper accessible to all general equilibrium modellers. In particular, we do not assume that readers are already familiar with MPSGE or GEMPACK. Most of the example models can be solved using the Demonstration Version of GEMPACK which can be downloaded (at no cost) from the web – see section 1.2 for details.

The current version of MGE2GP is **Version 1.1, July 2004**. We expect to make available new versions of this program regularly over the next year or so. Accordingly, when you want to start using the program, you should check the relevant web sites (see section 1.1) to obtain the latest version of the program and of the associated documentation.

The initial version of MGE2GP (Version 1, June 2004) was aimed at pedagogical models. The current version can also handle at least one larger-scale model, namely the MINIMAL model – see section 6.

MPSGE has been in use for over a decade as GAMS subsystem – see Rutherford (1995, 1999). During this time MPSGE has been used mainly by programmers who work at the command line and who use text editors. The main disadvantage of the GAMS/MPSGE environment has been the absence of an algebraic representation of the underlying model. The program MGE2GP translates MPSGE's tabular representation of an economic model into GEMPACK equations which can subsequently be examined or edited. This approach offers a considerable improvement in transparency and flexibility vis-a-vis GAMS/MPSGE.

- GEMPACK provides a Windows environment, so this allows MPSGE users to work in a Windows environment to carry out and analyse simulations.
- When an MPSGE model has been translated to GEMPACK, the GEMPACK file provides a complete and detailed representation of the underlying model structure. A knowledgeable user can

[2] MPSGE stands for **M**athematical **P**rogramming **S**ystem for **G**eneral **E**quilibrium modeling.

[3] MGE because is begins from a model written down in **M**PS**GE** format and GP since it converts to **G**EM**P**ACK.

[4] The early versions of MPSGE only allowed scalars. Versions of MPSGE allowing vectors and matrices were introduced in 1989.

edit and modify the GEMPACK code produced from MPSGE model, thereby providing considerable flexibility and opportunity for extending the modelling framework.

Now MPSGE models can be solved either using GAMS/MPSGE or using GEMPACK. We hope that having a common starting point for models will encourage GAMS and GEMPACK users to better understand each other's work. That aspect of this project is a continuation of the "mending the family tree" theme developed in Hertel *et al* (1992).

If you are a GAMS modeller who has never worked with GEMPACK you will find detailed instructions in sections 4 and 5 for working with several models in GEMPACK. You will be interested to learn about GEMPACK's capability for producing executable image versions of individual models (see section 5.4.2). If you convert your MPSGE model to GEMPACK, you can produce a version of your model that can be freely distributed to model users. In addition, GEMPACK comes with a number of helpful Windows programs for simulation analysis of policy issues. When GEMPACK models are generated from an MPSGE model format, modellers can exploit the user-friendly features of the GEMPACK environment while also exploiting the simplicity of MPSGE for defining model structure.

This paper is intended both for GEMPACK modellers who have not worked with MPSGE and for GAMS/MPSGE modellers who have not previously worked with GEMPACK. Of course, the paper is also intended for economists have used neither MPSGE nor GEMPACK. Depending on your background, you may be able to skip or skim through certain sections of the paper.

Section 2 introduces the MPSGE language syntax from scratch. It is primarily intended for readers (GEMPACK users) who have never used GAMS/MPSGE. In section 2 we describe several example models which are used throughout this document (including in the hands-on parts in sections 4 and 5). We have chosen these particular models in order to illustrate specific features of model representation in MPSGE.

Section 3 describes the specific steps involved in converting existing MPSGE models into GEMPACK, using the program MGE2GP. This section contains some details of the equations underlying an MPSGE model and of how these equations are written in the TAB file produced by the conversion program MGE2GP.

Sections 4 and 5 contain detailed hands-on instructions for solving the example MPSGE models we supply using GEMPACK. The material in these two sections will be of particular interest to you if you have worked with GAMS/MPSGE but have never used GEMPACK. Our hope is that modellers who have never used GEMPACK before will find sufficient detail in these sections to be able to solve the example models on their own computer. If you work through these sections, you will have received a good introduction to GEMPACK.

Section 7 contains some advice about designing MPSGE models which are most suitable for conversion to GEMPACK. Unlike the previous sections which we intend to be read in sequence, this section is more of a reference section. It contains technical information which you may need to know if you wish to build your own models by first creating an MPSGE representation and then using MGE2GP to convert your models to GEMPACK in order to solve them. There are several language features in MPSGE which are as yet not supported by the conversion program MGE2GP. These restrictions are documented in section 7.13.

In section 8, we set out our plans for further work on this topic. References (including references to the complete GEMPACK user documentation) are given in section 9.

Appendix 1 (section 10) contains syntax rules for the current version of the MPSGE modelling language.

1.1 Downloading MGE2GP and Associated Models and Documentation

We expect to make available new versions of the program MGE2GP regularly over the next year or so. These new versions will fix bugs and will extend the MPSGE syntax supported.

Accordingly, when you want to start using the program, you should check one of the following web sites to obtain the latest version of the program and of the associated documentation.

http://www.monash.edu.au/policy/gpmge2gp.htm

http://www.cretegny.ch/mge2gp.htm

1.2 Release 8 or Later of GEMPACK is Required

In order to work with the TAB files produced by MGE2GP, you need Release 8 (October 2002) or later of GEMPACK.[5] And you may need to install a couple of bug fixes – see

http://www.monash.edu.au/policy/gpmge2gp.htm

1.3 Downloading the Demonstration Version of GEMPACK

You will need a version of GEMPACK to carry out the simulations described in sections 4 and 5.

If you do not have GEMPACK already, you can download the Demonstration Version of GEMPACK for free from

http://www.monash.edu.au/policy/gpdemo.htm

- You should also download WinGEM (the Windows version of GEMPACK) and may wish to download AnalyseGE and the GEMPACK documentation.
- Follow the instructions at *http://www.monash.edu.au/policy/gpdemo.htm* for downloading and installing the various programs and files.

Of the various example models in section 2, all except for OPEN, BOP and TARIFF can be solved using the Demonstration Version of GEMPACK.[6]

1.4 Changes from Earlier Versions of MGE2GP

You can ignore this section if you have not used earlier versions of MGE2GP.

We describe briefly in section 12 the changes from earlier versions of MGE2GP. If you are familiar with one of these earlier versions you should read the relevant parts of that section since that will probably be the most efficient way of finding out what you need to know in order to move to the current version of MGE2GP.

1.5 Feedback on This Document

We want this document to provide good documentation for MPSGE, MGE2GP and the relevant parts of GEMPACK.

We are committed to maintaining this document, and expect to modify and improve this document in the light of experience with MGE2GP and feedback from users.

- If you find errors, or have difficulty understanding part of this document, please pass on your comments to one of the authors.
- If you have suggestions for improvements, please pass them on to one of the authors.
- If you find problems with any of the files from the website, or if you have difficulties carrying out the hands-on examples described in sections 4 to 6, or have problems reproducing the results set out in those sections, please pass on the details to Ken Pearson.
- If you find bugs in MGE2GP, please report these – see section 3.3 for details.

Laurent Cretegny (laurent@cretegny.ch)
Mark Horridge (Mark.Horridge@buseco.monash.edu.au)

[5] Some of the syntax used in the TAB files written by MGE2GP (for example, the "Linear_Var=" qualifier in Variable declarations) was not available in Release 7 of GEMPACK.

[6] The TAB files for OPEN, BOP and TARIFF have more formulas than are allowed by the Demonstration Version of GEMPACK.

Ken Pearson (Ken.Pearson@buseco.monash.edu.au)
Tom Rutherford (rutherford@colorado.edu)

2. Introduction to MPSGE via Several Example Models

MPSGE is intended primarily for producing general equilibrium models in which

- there are multiple interacting agents,
- individual behaviour is based on optimization,
- agent interactions are mediated by markets and prices,
- equilibrium occurs when endogenous variables (e.g., prices and activity levels) adjust such that agents, subject to the constraints they face, cannot do better by altering their behaviour.

In an MPSGE model producers maximize profit subject to available technology, and consumers maximize welfare subject to their budget constraint. Prices adjust so that markets for goods and factors clear. In most cases this means that supply equals demand in each market.

In short, the MPSGE modelling format for Arrow-Debreu general economic equilibrium models is based on *competitive equilibria*. There are three sets of “central variables” in an MPSGE model: prices (for primary factors and produced goods), activity levels for constant-returns-to-scale production sectors, and income levels for consumption agents.

An MPSGE approach to model building reduces both algebraic tedium and the scope for programming errors. This simplification of certain steps in model development permits the modeller to pay more attention to economic interpretation and to the testing of alternative formulations.

There is a long sequence of tasks involved in constructing a general equilibrium model. This process begins with the collection of source data, typically an IO table, household survey and a set of trade statistics. The work at the initial stages of a modelling project is largely focused on the interpretation, reconciliation and balancing of the source data.

After these data have been assembled, the modeller needs to decide on the specification of model variables and equations. The selected functional forms will have a number of parameters, and these coefficients (for both preferences and technology) must be calibrated from the base year data.

The next step is to replicate the benchmark equilibrium. This verifies internal consistency of the model equations and the derived coefficients.

Once the model is operational, the modeller then proceeds to define and solve a sequence of scenarios, produce reports in the form of tables and figures, and interpret results.

MPSGE offers substantial reductions in the work required for the middle steps in this process. GAMS/MPSGE automates the calibration of function parameters to a benchmark equilibrium while simultaneously providing an automatic specification of the equations which define an equilibrium. The novice modeller benefits from using MPSGE because it provides a clear framework for the underlying model. The expert benefits from the sparse format in which the model is portrayed and the resulting ease with which alternative models can be implemented and compared. These benefits are retained when the MPSGE model is translated to GEMPACK.

The rest of this section is an introduction to MPSGE via a sequence of relatively simple examples. Once you get some experience with general equilibrium models from the MPSGE perspective, you will be able to make sense of the advantages of this language for theoretical and/or empirical analysis.

We encourage you to download the example models (see section 1.1) and to look at the MPSGE files (they have suffix .MGE) as you read about the models in the subsections below.

2.1 An Introductory Example - TWOBYTWO.MGE

We begin with a familiar model of a two-by-two closed economy. This model, which is provided in the file `twobytwo.mge`, is a workhorse of applied micro-economics and trade theory, representing an economy with two goods (*X* and *Y*), two factors (*K* and *L*), and a single representative consumer. The goods are produced through constant returns to scale production activities which combine primary factor inputs. In the simplest case both factors are in fixed supply, so an equilibrium is characterized by equality of factor endowments and factor demands, equality of commodity production and commodity demand.   These equilibrium conditions have corresponding variables representing the commodity prices (p_x and p_y) and factor prices (w, the wage rate, and r, the rental rate).

Competitive producer behaviour assures equalization of output prices and marginal cost in equilibrium, two equations which correspond to producer output levels (x and y). Finally, an equilibrium involves budget balance which relates the value of consumer expenditure to the value of factor endowments.

Almost all applied equilibrium models begin with accounting data describing purchases and sales in one (or more) years. Technology and preferences in primitive form are only defined implicitly by these statistics. Most applied models are based, in one way or another, on input-output statistics or social accounts. The following social accounting matrix (SAM) is used for deriving coefficients of the production and utility functions in the present example:

	X	Y	C	L	K	RA
X			100			
Y			50			
C						150
L	50	20				
K	50	30				
RA				70	80	

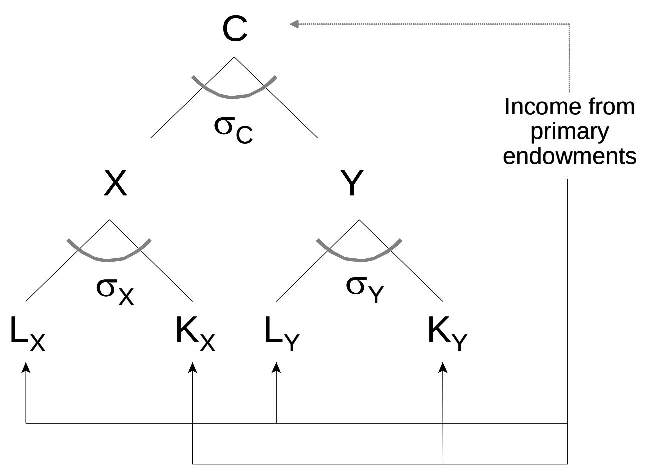
These data are presented above in a square social accounting matrix (SAM). The accounts labelled X and Y in this matrix refer to markets for final commodities. Account C corresponds to final consumption, an *activity* which transforms goods *X* and *Y* into a composite consumption good *C*. The RA account corresponds to the representative agent. This account defines both the endowments and expenditures for the model's single representative consumer.

The SAM provides a variety of benchmark value shares. For example, you can see that sector *X* constitutes two-thirds of base year GDP, and the benchmark value shares of labour and capital in the production of *X* are both one half. Sector *Y* constitutes one third of GDP, and the value shares of labour and capital in the production of *Y* are 40% and 60%, respectively. All produced output enters final demand, and the value of expenditure equals the value of factor earnings.

These data represent a *balanced equilibrium dataset.* The underlying balance of the base year accounts implies internal consistency of the social accounts. Coefficients in the matrix are payments from the column account (production sector, market or consumer) to the row account. When the row sum of an account equals column sum of the same account, the account is in balance.[7]

The social accounts are essential inputs to a model formulation, but they do not by themselves completely characterize a general equilibrium framework. A model formulation relies on a variety of assumptions regarding preferences, technology and behaviour. For the present model, these structural assumptions might be conveyed diagrammatically as follows:

[7] Consider, for example, the X market. The row account indicates that 100 units of good *X* enter into final consumption (column C). The X column has entries in the L and C accounts representing a payment of 50 to both factors in the production sector for good *X*.



The model data in the SAM only convey *local* information about technology and preferences. When we sketch the above diagram in which we indicate elasticities of substitution in the various sectors, we have *calibrated* the model to the reference point. If you were to build a model based on this data using an algebraic modelling language such as GEMPACK or GAMS, you would need to derive a number of function coefficients based on equations derived from the associated demand functions. When you build a model from this data using MPSGE, the source data itself enters directly into the MPSGE tables. Here is the MPSGE model corresponding to the input data shown in these social accounts and to the nesting diagram above:

```
$model:twobytwo

$sectors:
  x                 ! production index for sector X
  y                 ! production index for sector Y
  c                 ! consumption index

$commodities:
  px                ! price index for commodity X
  py                ! price index for commodity Y
  pc                ! consumer price index
  pl                ! price index for primary factor L
  pk                ! price index for primary factor K

$consumers:
  ra                ! income for representative agent RA

$prod:x             s:1
  o:px              q:100
  i:pl              q:50
  i:pk              q:50

$prod:y             s:1
  o:py              q:50
  i:pl              q:20
  i:pk              q:30

$prod:c             s:1
  o:pc              q:150
  i:px              q:100
  i:py              q:50

$demand:ra
  d:pc
  e:pl              q:70
  e:pk              q:80
```

The MPSGE file starts off with variable declarations. In the `$commodities:`, `$sectors:` and `$consumers:` sections of the file, variable names are introduced and classified according as to how they are to be interpreted in the model.

- Each `$sector:` in the model has an associated `$prod:` block in which the production inputs and outputs are described. Each `$prod:` block indicates the inputs (the `i:` lines) and the outputs (the `o:` lines) in the production process.
- Each `$consumer:` in the model has an associated `$demand:` block in which endowments and preferences are described. The endowments are in the `e:` lines in these blocks. The income of consumers comes from the value of the endowments and from taxes (which we introduce later, in section 2.3). Normally endowments are exogenous (though they can be made endogenous as you will see in section 2.4). The single `d:` line in each `$demand:` block indicates the commodity which the consumer demands.
- The `$commodities:` appear in `i:` and `o:` fields at the start of lines in `$prod:` blocks and they appear in `e:` and `d:` fields at the start of lines in `$demand:` blocks. These `$commodities:` are really the market prices (that is, the non-user-specific prices) of the commodities.
- All the necessary variables for solving the model are declared in the `$sectors:`, `$commodities:` and `$consumers:` blocks. These are called **central variables** of the model, and they correspond to *activity levels*, *prices* and *income levels* respectively.

Numerical values from the social account matrix can appear verbatim in the MGE file. The *quantity* (`q:`) fields contain these values – when benchmark prices are normalized to unity, benchmark values and quantities are identical.

These data in the model are organized by account as in the SAM. For example, the *X* account in the social accounts is represented by the `$prod:x` block in the MGE file. The first record in this block describes an *output* (`o:`) of 100 units from this sector to the market for *X*. The next two records describes *inputs* (`i:`) of labour and capital. Every number from the social accounts appears in at least one location in the MGE file. Some elements appear twice – for example, the output of *X* in the `$prod:x` block appears as a input in the `$prod:c` block.

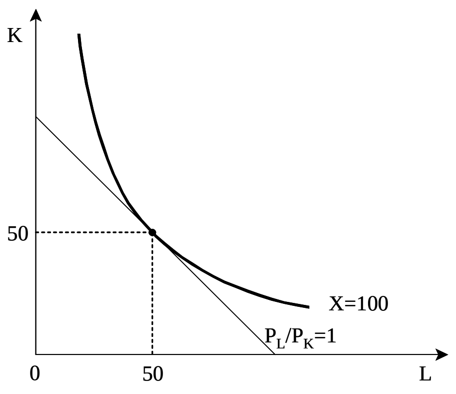
Preferences of a consumer in an MPSGE model are described by the single commodity that a consumer demands. Each consumer demands exactly one commodity, and the composition of that commodity describes the structure of consumer preference. In this example the `$prod:c` block in the MPSGE model describes the structure of final demand, representing column C in the SAM. The output of the `c` activity, commodity with price `pc`, is a composite good, the price of which represents the consumer price index.

The benchmark data do not uniquely determine the model structure. Elasticities of substitution appear in the `s:` fields in the first record of each `$prod:` block. For example, `s:1` in the `$prod:x` line indicates that the elasticity of substitution between the inputs of labor and capital is 1 in the production of `x`.

For completeness, we show in Appendix 2 (section 11) the mathematical structure of this model.

2.1.1 Calibration of TWOBYTWO

An important step involved in setting up a numerical model is the *calibration* of function parameters to benchmark data. This involves determining values of the parameters of production and utility functions which are consistent with the actual data in the reference equilibrium. It is quite easy to see how this works from a geometric perspective. Consider, for example, the production of good `X` in the 2x2 model. The combination of labour and capital to produce output is assumed to be cost-minimizing. This implies tangency of the X=100 *isoquant line* with the *isocost curve* in the following diagram:



The isoquant in this diagram is the solid curve, representing combinations of `K` and `L` which can produce 100 units of `X` output. The isocost line in this diagram is the straight line, representing combinations of `K` and `L` which have the same value as the benchmark quantity. The *calibration* of the function involves working out the equation of the isoquant from a knowledge of the benchmark point. The equation of the isoquant depends on the data at the benchmark point and on the elasticity of substitution (specified in the `s:1` field in the `$prod:x` block).

The same calibration is performed for all three of the production functions appearing in the model. Reference levels of inputs and outputs together with benchmark elasticities of substitution (the `s:` field following the `$prod:` declaration) imply that the production functions for *X*, *Y* and *C* are given by:

$$f_x(k, l) = 2\, k^{1/2}\, l^{1/2} \,,$$

$$f_y(k,l) = 1.96\;\; k^{3/5}\, l^{2/5}$$

and

$$U(c_x,c_y) = 1.89\;\; c_x^{2/3}\, c_y^{1/3}.$$

2.1.2 Converting TWOBYTWO to GEMPACK

If you are keen to see how this TWOBYTWO model converts to GEMPACK and how to carry out simulations with it using GEMPACK, you can jump ahead to section 4. If so, we suggest that you then come back to section 2.2 below, where we introduce the MPSGE representation of some other models.

2.2 Three Simple Models

The TWOBYTWO model is non-typical in two respects. First all the benchmark data are shown explicitly in the MGE file. Secondly, each commodity and sector is shown with its own account.

More typically,

- you can use set notation when convenient, as in cases where there are many goods, factors, countries or consumers.
- the benchmark data can be shown in the MPSGE file using symbols for vectors and matrices. The actual benchmark data can be read from external files (or can be specified somewhere else in the program).

Combining these two features, data can be specified in arrays or tables, and read into the computation program in a straightforward way. We are going to use this kind of notation from now on.

In this section we introduce three simple models to show you how to represent economic features in MPSGE. The first example, `sjmge.mge`, brings in intermediate demands. We then move on a joint production model, `joint.mge`. Finally, we give you an idea about how to represent international trade flows in a small open economy, `open.mge`.

2.2.1 Intermediate Demand – SJMGE.MGE

Any real economies use intermediate goods in the production process. We use the Stylized Johansen[8] model, `sjmge.mge`, to illustrate how to represent this feature in MPSGE. Consider first the SAM.

	S1	S2	LAB	CAP	C	RA
	sect		fac			
S1	4	2			2	
S2	2	6			4	
good	comin				hous	
LAB	1	3				
CAP	1	1				
fac	facin					
C						6
RA			4	2		
			Xfac			

You see immediately that the data may be grouped into different submatrices. These submatrices allow us to write down the MPSGE production and demand blocks in a more compact way, using vector and matrix notation. The full **`sjmge.mge`** file is shown below.

```
$model:sj

$sectors:
 xcom(sect)      ! production
 w               ! welfare

$commodities:
 pc(sect)        ! price of commodity sect
 pf(fac)         ! price of factor fac
 pw              ! consumer price index

$consumers:
 y               ! total nominal household expenditure

$prod:xcom(sect)  s:1.0
 o:pc(sect)      q:(sum(good, comin(good,sect)) + sum(fac, facin(fac,sect)))
 i:pc(good)      q:comin(good,sect)
 i:pf(fac)       q:facin(fac,sect)

$prod:w           s:1.0
 o:pw            q:(sum(good, hous(good)))
 i:pc(good)      q:hous(good)

$demand:y
 d:pw
 e:pf(fac)       q:endow(fac)
```

[8] See chapter 3 of Dixon *et al* (1992) and chapter 3 of GPD-1.

Below we ask you to look in detail at selected parts of the file.

In this model, the production sector for output is as follows:

```
$prod:xcom(sect)    s:1.0
  o:pc(sect)   q:(sum(good, comin(good,sect)) + sum(fac, facin(fac,sect)))
  i:pc(good)        q:comin(good,sect)
  i:pf(fac)         q:facin(fac,sect)
```

Each sector `sect` produces a commodity `pc(sect)` using primary factors, `pf(fac)`, and intermediate goods, `pc(good)`. You can see from the `s:` field that the technology used to combine these production factors is based on a Cobb-Douglas production function, which is characterized by an elasticity of substitution equal to one.

You can think of the vector notation as a shorthand for what could be spelled out by listing `$prod:` blocks separately and then listing the inputs and outputs separately (as was done in TWOBYTWO.MGE).

For example, assume that the set `good` has the two elements `s1` and `s2`. Then the `$prod:` block above is a shorthand for the following two `$prod:` blocks. The first is obtained by replacing `sect` everywhere by "`s1`". The second by "`s2`".[9]

```
$prod:xcom("s1")    s:1.0
  o:pc("s1")  q:(sum(good, comin(good,"s1")) + sum(fac, facin(fac,"s1")))
  i:pc(good)        q:comin(good,"s1")
  i:pf(fac)         q:facin(fac,"s1")

$prod:xcom("s2")    s:1.0
  o:pc("s2")    q:(sum(good, comin(good,"s2")) + sum(fac, facin(fac,"s2")))
  i:pc(good)        q:comin(good,"s1")
  i:pf(fac)         q:facin(fac,"s1")
```

Suppose that the set `fac` contains the two elements `labour` and `capital`. Then the `i:` line in the `$prod:xcom("s1")` block above is a shorthand for the two `i:` lines :

```
  i:pf("labour")    q:facin("labour","s1")
  i:pf("capital")   q:facin("capital","s1")
```

That is, there are two lots of factor inputs into the production of `xcom("s1")`, namely `facin("labour","s1")` of labour and `facin("capital","s1")` of capital.

Suppose that the set `good` has the same 2 elements `s1` and `s2` as are in the set `sect`. Then, first `i:` line in the `$prod:xcom("s1")` block above is a compact way of writing what could be expressed via the following two `i:` lines :

```
  i:pc("s1")        q:comin("s1","s1")
  i:pc("s2")        q:comin("s2","s1")
```

There are two inputs of commodities into the production of `xcom("s1")`, namely `comin("s1","s1")` of commodity `pc("s1")` and `comin("s2","s1")` of commodity `pc("s2")`.

In GAMS/MPSGE the symbols `comin`, `qfacin` above are called *parameters*. In GEMPACK they are called *Coefficients*. In this paper we refer to them using either of these terms.

You can notice that the `q:` field on the `o:pc(sect)` line is an expression, which represents the sum over all production factors used in each sector.

Regarding the final demand, the representative agent derives welfare `w` from consumption of the commodities. As in the earlier TWOBYTWO example, this is represented in a `$prod:` block. In vector syntax, the `$prod:` block for welfare is the following:

```
$prod:w             s:1.0
  o:pw              q:(sum(sect, hous(sect)))
  i:pc(sect)        q:hous(sect)
```

[9] Strictly speaking, you should not write these two `$prod:` blocks in an MGE file since each sector is only allowed to be defined by a single `$prod:` block (see Appendix 1). Instead you would have to define scalar sectors `xcoms1` (in place of `xcom("s1")`) and `xcoms2` (in place of `xcom("s2")`) and write the blocks as `$prod:xcoms1` and `$prod:xcoms2`.

For completeness, we show below the `$demand:` block in vector syntax for the representative agent. The consumer is endowed with primary factors, `pf(f)`, and demands the welfare composite of purchased final goods, `pw`.

```
$demand:y
  d:pw
  e:pf(fac)         q:endow(fac)
```

The benchmark data for this model are contained on a separate file (see section 5.1.1).

2.2.2 Joint Production and Nests – JOINT.MGE

We turn now to an economy whose industries produce several outputs. Building on the previous explanation for intermediate demand, joint production is easily accommodated in MPSGE. Consider the example **`joint.mge`** which is shown below.

```
$model:joint

$sectors:
  y(s)                ! production
  u                   ! utility index

$commodities:
  pd(o)               ! domestic price of commodity
  pf(f)               ! price of primary factor
  pu                  ! price index for utility

$consumers:
  ra                  ! representative agent income

$prod:y(s)            t:etrn(s)           va:esub(s)
  o:pd(o)             q:supply(s,o)
  i:pd(o)             q:interm(o,s)
  i:pf(f)             q:factor(f,s)       va:

$prod:u               s:esubc
  o:pu                q:(sum(o, demand(o)))
  i:pd(o)             q:demand(o)

$demand:ra
  d:pu
  e:pf(f)             q:endow(f)
```

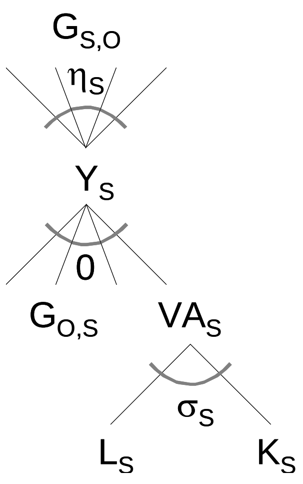
Look at the production sector for output `y(s)` in this model:

```
$prod:y(s)          t:etrn(s)          va:esub(s)
  o:pd(o)           q:supply(s,o)
  i:pd(o)           q:interm(o,s)
  i:pf(f)           q:factor(f,s)      va:
```

As in the `sjmge.mge` example, each industry `s` employs primary factors `pf(f)` and intermediate goods `pd(o)` in the production process. However these industries produce multiple outputs and use production factors in a different way, as we explain below.

As we explain below, the nesting diagram showing inputs and outputs for these sectors `y(s)` is as follows.[10]

[10] For example, L_s in the diagram corresponds to `factor(“labor”,s)` in the MGE respresentation. Similarly η_S in the figure corresponds to `etrn(s)` in the MPSGE representation.



Multiple outputs

How can you see from the above `$prod:y(s)` block that each sector (or industry) `s` produces several outputs? Look at the o:pd(o) line, which is the line that specifies the outputs of these sectors. The key is that

the index `o` in the `o:pd(o)` line is different from the index `s` in the `$prod:y(s)` field.[11]

The second clue is that both indices `s` (the sector index) and `o` (the commodity index) appear in the quantity field `q:supply(s,o)`. The symbol `supply(s,o)` indicates the output of commodity `o` by sector `s`.

In the data associated with this model, there are two sectors `sagr` (the agricultural sector) and `sind` (the industrial sector). Each sector produces the two commodities `pd("agr")` [the agricultural commodity] and `pd("ind")` [the industrial commodity]. The `supply(s,o)` matrix is[12]

supply(s,o)	o = agr	o = ind
s = sagr	90	40
S = sind	40	90

You can see that sector `sagr` produces mainly the `agr` commodity (90 units) but also some of the `ind` commodity (40 units).

The curvature of the production possibilities frontier, which is a CET (constant elasticity of transformation) function, is given in the `t:etrn(s)` field on the `$prod:` line. Here `etrn(s)` is the value of the elasticity of transformation between outputs in sector `s`. The `s` subscript here indicates that there may be different values for this elasticity in the different sectors.

In the above `$prod:y(s)` block, there is no `s:` field. This means that the value of the substitution elasticity between inputs is 0 (the MPSGE default value when the field is omitted).

Nesting

Another feature is introduced here, namely the nesting notation.

This is indicated by the field **`va:esub(s)`** in the `$prod:y(s)` line and by the label **`va:`** in the `i:pf(f)` line.

[11] If each sector `s` only produces a single output, the `o:` line would be `o:pd(s)`. In that case, sector `s` would only produce the single output, namely commodity `pd(s)`.

[12] To see this look at the header SPPL on the data file `joint.har` supplied. [You will need to use the program ViewHAR to look at this file. Follow the procedure described in section 5.1.1.]

- If these `va:` references were omitted, it would be possible to substitute between all inputs. [The inputs in the above `$prod:y(s)` block are inputs of commodities – see the `i:pd(o)` line – and inputs of factors – see the `i:pf(f)` line.] The elasticity of substitution between these would be zero (since there is no `s:` field on the `$prod:y(s)` line.
- But the purpose of the `va:` label in the `i:pf(f)` line is to indicate that inputs of factors pf(f) – the inputs on this line – are to be first combined in their own nest to form a composite input denoted by `va`.[13] Then this composite commodity `va` can be substituted with the other inputs as shown in the other `i:` line (namely the commodity inputs pd(o)).

So the `va:` references above indicate that there is a two-level nest here.

- In the bottom level nest, the inputs are the different factors pf(f) and elasticity of substitution between them is `esub(s)` – see the `va:esub(s)` field on the `$prod:y(s)` line. [Note that the elasticity of substitution `esub(s)` may be different for different industries.]
- In the top level nest, the inputs are `va` and the various commodities `pd(o)`. Again the elasticity of substitution between these inputs is zero (since there is no `s:` field on the `$prod:y(s)` line).[14]

Here you cannot employ more labour or capital to decrease the quantity of intermediate goods used in the production. Labour and capital are only a substitute for one another. The composite input va is then combined with intermediate goods.

The bottom level nest is declared by the label `va:` (MPSGE allows this label to have up to 4 characters) on the `$prod:` line and on the `i:` line corresponding to primary factors.

2.2.3 Small Open Economy – OPEN.MGE

A natural extension to the models presented up until now is the representation of international trade flows. We assume a small open economy where the rest of the world is not explicitly modelled. Trading opportunities are summarized by simple production functions which allow the economy to transform exports into a good which we will call "foreign exchange" (its price is denoted by `pfx`) and which can then be used as input for producing imports.

First we show the whole of `open.mge` and then we discuss various parts of it.

```
$model:open

$sectors:
  y(s)              ! production
  a(s)              ! aggregate supply
  e(s)              ! export index
  m(s)              ! import index
  u                 ! utility index

$commodities:
  pd(s)             ! domestic price of commodity
  pa(s)             ! price of commodity
  pf(f)             ! price of primary factor
  pe(s)             ! price of export
  pm(s)             ! price of import
  pu                ! price index for utility
  pfx               ! real exchange rate

$consumers:
  ra                ! representative agent income
```

[13] `va` stands for value added.

[14] When the elasticity field is omitted, the MPSGE convention is that the value is taken to be zero.

```
$prod:y(s)          t:etrn(s)           va:esub(s)
  o:pd(s)           q:supply(s)
  o:pe(s)           q:export(s)
  i:pa(o)           q:interm(o,s)
  i:pf(f)           q:factor(f,s)       va:

$prod:a(s)          s:esubm(s)
  o:pa(s)           q:(supply(s)+import(s))
  i:pd(s)           q:supply(s)
  i:pm(s)           q:import(s)

$prod:e(s)
  o:pfx             q:export(s)
  i:pe(s)           q:export(s)

$prod:m(s)
  o:pm(s)           q:import(s)
  i:pfx             q:import(s)

$prod:u             s:esubc
  o:pu              q:(sum(s, demand(s)))
  i:pa(s)           q:demand(s)

$demand:ra
  d:pu
  e:pf(f)           q:endow(f)
* bbop is the Benchmark Balance of Payments Surplus (X-M)
  e:pfx             q:-bbop
```

Exports and Imports

In this model, we assume that both exports and imports are imperfect substitutes for domestic goods.

Exports `export(o)` are shown in the following CET function on the production side:

```
$prod:y(s)          t:etrn(s)           va:esub(s)
  o:pd(s)           q:supply(s)
  o:pe(s)           q:export(s)
  i:pa(o)           q:interm(o,s)
  i:pf(f)           q:factor(f,s)       va:
```

Here `etrn(s)` is the elasticity of transformation. Industry activity `y(s)` can be imperfectly transformed either into an exportable commodity `pe(s)` or into supplies `pd(s)` for domestic use.

Imports `import(o)` are shown in the following CES function on the consumption side:

```
$prod:a(o)          s:esubm(o)
  o:pa(o)           q:(supply(o)+import(o))
  i:pd(o)           q:supply(o)
  i:pm(o)           q:import(o)
```

This is the familiar Armington nest. For each `o`, imports `pm(o)` and domestic supplies `pd(o)` of commodity `o` are combined to produce the composite commodity `pa(o)` which is used by domestic firms and consumers. The elasticity of substitution (the Armington parameter) for commodity `o` is given by `esubm(o)`.

Foreign exchange `pfx`

The main new feature in `open.mge` is the introduction of the commodity `pfx` (foreign exchange). This is introduced in order to keep track of, and fix, the balance of trade.

The blocks containing `pfx` are the following :

```
$prod:e(o)
  o:pfx             q:export(o)
  i:pe(o)           q:export(o)

$prod:m(o)
  o:pm(o)           q:import(o)
  i:pfx            q:import(o)

$demand:ra
  d:pu
  e:pf(f)          q:endow(f)
* bbop is the Benchmark Balance of Payments Surplus (X-M)
  e:pfx            q:-bbop
```

As with all other commodities, supply and demand for `pfx` must be equal. However the artificial endowment `–bbop` accommodates the possibility that total exports may not equal total imports in the benchmark equilibrium, as we explain below.

From the `o:pfx` line in the `$prod:e(o)` block, you can see that `sum(o,export(o))` is part of the supply.

From the `i:pfx` line in the `$prod:m(o)` block, you can see that `sum(o,import(o))` is part of the demand.

What about the `e:pfx` line in the `$demand:ra` block?

> Normally an `e:` line indicates an endowment which means supply equal to the value in the `q:` field. But a negative endowment (that is, a negative number in the `q:` field) is actually counted as a (positive) demand (see section 3.2.4 for more details).
>
> Suppose firstly that the benchmark value for `bbop` is negative. Then the value in `q:-bbop` is positive, which counts as a supply of `pfx` equal to `-bbop`. Hence
>
> total demand for `pfx` is equal to `sum(o,import(o))`
>
> while
>
> total supply of `pfx` is equal to `sum(o,export(o)) + (-bbop)`.
>
> For balance, we need
>
> `sum(o,import(o)) = sum(o,export(o)) – bbop`
>
> which means that
>
> `bbop = sum(o,export(o)) – sum(o,import(o))`.
>
> Suppose secondly that the benchmark value for `bbop` is positive. Then the value in `q:-bbop` is negative. This counts as a positive demand for `pfx`. That is, `bbop` (not `–bbop`) is added to the demand for `pfx`. Thus
>
> total demand for `pfx` is equal to `sum(o,import(o)) + bbop`
>
> while
>
> total supply of `pfx` is equal to `sum(o,export(o))`.
>
> For balance, we need
>
> `sum(o,import(o)) + bbop = sum(o,export(o))`
>
> which means, again, that
>
> `bbop = sum(o,export(o)) – sum(o,import(o))`.
>
> Hence, in both cases, we see that the benchmark value for `bbop` must be equal to the familiar `X-M`, the benchmark balance of trade **surplus**.[15]
>
> In the benchmark data in the Header Array file `open.har` supplied with `open.mge`,
>
> total exports `sum(o,export(o))` are equal to 60 and

[15] You don't really need to consider the two cases. From the `e:pfx` line, just count `-bbop` as a supply (ignoring its sign). Then total supply is `X-bbop` (from the `o:pfx` and `e:pfx` lines) while total demand is `M` (from the `i:pfx` line). Hence `X-bbop=M` and so `bbop=X-M`.

total imports `sum(o,import(o))` are equal to 50.

Thus the benchmark balance of trade surplus is 10, which is the value of `bbop` in `open.har`.

This explains the `e:pfx` line in the `$demand:ra` block. The representative agent `ra` is assumed to be given an endowment equal to `-bbop` of this somewhat artificial commodity `pfx`. This is done to keep the supply and demand for `pfx` equal in the benchmark (and throughout a simulation).

Why is there a negative sign in the `q:-bbop` field?

- Economically, a balance of trade surplus can be thought of as a gift to the rest of the world. Hence it is subtracted from the income of consumer `ra`. This is the effect of the `q:-bbop` field.
- Mathematically, the sign in the MGE file could be omitted without invalidating the model. That is, we could have equally well put `q:bbop` as the `q:` field in that `e:pfx` line. But then, instead of `bbop` being interpreted as the balance of trade surplus, it would be interpreted as the balance of trade **deficit** (and the value –10 would need to be put into the Header Array file `open.har`).

In summary, usually economies don't have a zero trade balance in a given year but present a current account surplus or deficit. This possible trade imbalance is modelled by endowing foreign exchange to the representative agent equal to the imbalance.

2.2.4 Summary of MPSGE Features Introduced So Far

You have now learned about several important MPSGE features.

- Benchmark data can be shown by numbers in the `q:` fields (as in `TWOBYTWO.MGE`), or by symbols representing vectors or matrices (as in `SJMGE.MGE`).
- When vector notation is used in `$prod:` blocks, there are really several different `$prod:` blocks, one for each sector. [See the discussion of the `$prod:y(s)` block in `SJMGE.MGE`.]
- When vector notation is used in an `i:` line, this line indicates several different inputs (one for each value of the index). Similarly for `o:` lines. [See `SJMGE.MGE` for the `i:` line case.]
- When there are two or more `i:` lines, these represent different inputs. Similarly for `o:` lines. [See `JOINT.MGE` for the `i:` line case.]
- There are multiple outputs if an index in an `o:` line is different from the indexes in the `$prod:` line. [See `JOINT.MGE`.]
- Nesting is indicated by extra fields in the `$prod:` line and by corresponding labels in certain `i:` lines. [See the `va` nest in `JOINT.MGE`.]
- There can be negative endowments. [See `q:-bbop` in `OPEN.MGE`.]
- It is sometimes necessary to introduce somewhat artificial commodities to keep the accounting framework straight. [See `pfx` in `OPEN.MGE`.]

2.3 Adding Exogenous Taxes

This section introduces exogenous taxes. The important lesson is to keep track of what prices firms and consumers face. Until now, benchmark prices were supposed to be one but when you want to have pre-existing taxes, then it may not be possible to calibrate a benchmark equilibrium with all prices equal to one. You need to tell MPSGE what is the equilibrium price at the benchmark (if you do not specify its reference value, MPSGE assumes the default value equal to one). The relative prices of inputs fix the marginal rate of substitution and the relative prices of outputs fix the marginal rate of transformation.

2.3.1 Taxes on Inputs – TAXIN.MGE

In the benchmark SAM you have values received by firms and consumers, so that residual is government revenue. For simplicity you can introduce a government agent collecting taxes. In the present example, `taxin.mge`, the government agent also purchased goods, `govdmd`, which represents government expenditures.

Taxes on **inputs** are specified on a **net basis**. This means that,

if a tax has *ad valorem* rate *t* then the user price is *p(1+t)* where *p* is the market price.

Note also that the price in the `i:` field is always the market price.

Input taxes in MPSGE are specified as follows.

- On an `i:` line, you indicate the recipient of the tax by the mean of an `a:` field and give then the rate of the tax in a `t:` field.
- The tax agent is specified before the tax rate.
- Two or more taxes may be applied on a single `i:` line.
- A p: field is used to specify the reference price at the benchmark. If there is no p: field, the MPSGE convention is to proceed as if p:1 were present.[16]

Consumers are treated symmetrically, and there is thus no restriction on the consumer to whom the tax is paid. Typically, however, one consumer is associated with the government, `gov`, as is the case in the `taxin.mge` example.

In `taxin.mge`, there is provision for taxes to be levied on commodity inputs `pd(o)` and factor inputs `pf(f)` to sector `$prod:y(s)` as is shown below.

```
$prod:y(s)    t:etrn(s)         va:esub(s)
  o:pd(o)     q:supply(s,o)
  i:pd(o)     q:interm(o,s)     a:gov          t:bti(o,s)
  i:pf(f)     q:factor(f,s)     p:bpf(f,s)     a:gov     t:btf(f,s)     va:
```

- The user price in sector `s` for factor `f` is `pf(f)*[1 + btf(f,s)]`. The values bpf(f,s) in the p: field must be equal to this. [If not, the program solving the model will stop with a fatal error since the benchmark data are not internally consistent.]
- The user price in sector `s` for commodity `pd(o)` is `pd(o)*[1 + bti(o,s)]`. Since there is no `p:` field in this line, the program proceeds as if p:1 were present. Hence the benchmark values `bti(o,s)` must all be zero. That is, there are no taxes on commodity inputs in the benchmark.

In this benchmark data supplied with this model, taxes are only levied on capital used in the industrial sector `sind`.[17] The tax rate on capital in this sector is equal to 100% which implies a reference price for capital in the industrial sector equal to 2 when the consumer price is unity.

An important issue here is that the `p:` field depends on the benchmark value of the `t:` field if the model has been calibrated. This means that subsequent changes in tax rates affecting the `t:` field do not change the underlying technology characterized by the `q:` and `p:` field and its associated elasticity.

We encourage you to look at the whole of the **`taxin.mge`** file (which you have downloaded as part of the MGE2GP package – see section 1.1).

[16] If there are two or more taxes on an i: line, the value in the p: field must take all taxes into account.

[17] You can see this by using ViewHAR to look at the data in the Header Array file `taxin.har`. [Follow the method described in section 5.1.1.]

2.3.2 Taxes on Outputs – TAXOUT.MGE

Taxes on outputs are introduced in MPSGE in the same way as taxes on inputs apart from the interpretation.

Taxes on **outputs** are specified on a **gross basis**. This means that,

if a tax has *ad valorem* rate *t* then the user price is *p(1-t)* where *p* is the market price.

Note also that the price in the `o:` field is always the market price.

You can see from the missing `p:` field on the `o:` line in the following `$prod:` block from `taxout.mge` that there is no output tax in the benchmark equilibrium. However the `a:` field and the `t:` fields are specified as in the case of an input tax.

```
$prod:y(s)    t:etrn(s)        va:esub(s)
  o:pd(o)     q:supply(s,o)    a:gov          t:bto(s,o)
  i:pd(o)     q:interm(o,s)
  i:pf(f)     q:factor(f,s)    p:bpf(f,s)     a:gov      t:btf(f,s)     va:
```

Regarding the interpretation, the tax rate on inputs is specified on a net basis, while the tax rate on outputs is specified on a gross basis. In other words, the user cost of an input with market price *p* subject to an *ad valorem* tax at rate *t* is *p*(1+*t*), while the user cost of an output subject to an *ad valorem* tax at rate *t* is *p*(1-*t*).

In order to help remember the difference between taxes on inputs and outputs, you can think of a producer.

The producer's costs increase with the introduction of an input tax,
whereas the value of the producer's outputs decrease with an output tax.

The conventions for modelling taxes in MPSGE have an important implication. The application of an *ad valorem* tax rate *t* on all the outputs in a `$prod:` block in place of an *ad valorem* tax rate *t*/(1-*t*) on all the inputs in the same `$prod:` block has no effect on the equilibrium. In other words, an output tax t^o defined on a gross basis is equivalent to an input tax t^i defined on a net basis.

We encourage you to look at the whole of the **`taxout.mge`** file (which you have downloaded as part of the MGE2GP package – see section 1.1).

2.3.3 Tariffs and Export Subsidies – TARIFF.MGE

Tariffs and export subsidies follow the same logic as taxes on inputs or outputs.

Our example `tariff.mge` was produced by adding import and export taxes to the model in `open.mge` (see section 2.2.3).

Export subsidies are introduced in the export block while tariffs are introduced in the import block. You can see from `tariff.mge` that these two blocks are now the following :

```
$prod:e(o)
  o:pfx           q:export(o)         p:bpe(o)  a:gov     t:bte(o)
  i:pe(o)         q:(bpe(o)*export(o))
$prod:m(o)
  o:pm(o)         q:(bpm(o)*import(o))
  i:pfx           q:import(o)         p:bpm(o)  a:gov     t:btm(o)
```

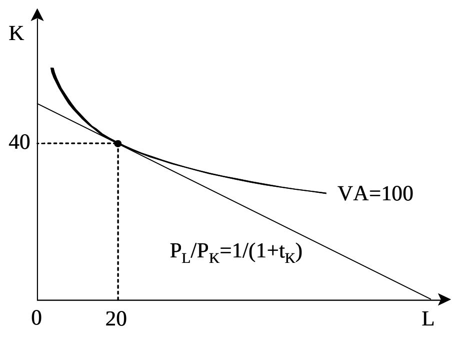
In open economy models, the modeller must chose units or prices. The convention we adopt in this example is that units are chosen such that all domestic prices are equal to one initially. However the tariffs increase the domestic price of imports and the export subsidies increase the domestic price of exports. The domestic price of imports at the benchmark is then `bpm`=1+`btm` and the domestic price of exports is `bpe`=1-`bte` for unity world prices. It implies that if domestic prices are equal to one at the benchmark, the world price for imports must equal 1/(1+`btm`) and for exports 1/(1-`bte`). Therefore `import` and `export` are interpreted as the value of the flows at world prices.

We encourage you to look at the whole of the **`tariff.mge`** file (which you have downloaded as part of the MGE2GP package – see section 1.1).

2.3.4 Calibrating With Taxes

When there are taxes, you cannot assume (as we did implicitly in section 2.1.1) that prices are all one.

In the following figure, benchmark quantities are *L0* = 20 (labor) and *K0* = 40 (capital) and there is an *ad valorem* tax of t_K=100% on capital. So you can think of P_L=1 (price of labor), P_K=2 (user price of capital) and *VA*=100 (value added) at the benchmark. The production function is calibrated using this price and quantity information. Benchmark quantities determine the anchor point with *L0* = 20 and *K0* = 40. Benchmark relative prices fix the slope of the isoquant line at that point and the elasticity describes the curvature of the isocost curve.[18]



To summarize behaviour of consumers and producers is represented in MPSGE by the specification of:

- Benchmark quantities.
- Benchmark prices.
- Elasticity at the benchmark point.

2.4 Modifying Behaviour via Rationing

As described above, MPSGE provides a rather tight straight-jacket (which comes from the Arrow-Debreu framework). The natural closure for your model has endowments and taxes exogenous. All central variables of the model are solved for. In particular, prices of the commodities are determined by the market clearing equations of the model. Usually the prices adjust so that all endowments are fully used.

In this section we introduce ways you can break out of this straight-jacket using rationing. We first consider a model where real wages are fixed, `employ.mge`, and then move to a model where consumption is fixed, `bop.mge`.

2.4.1 Linking Wages to Consumer Prices – EMPLOY.MGE

In the models presented until now, we have a fixed supply of primary factors. Suppose for simplicity that there is a single household that gets all the income from labour and capital. The MPSGE representation for this representative agent `ra` is done through the `$demand:` block :

```
$demand:ra
  d:pu
```

[18] Thought of in a formal way, benchmark quantities alone provide a zero order approximation (the single anchor point) of the underlying technology. Benchmark reference prices and reference quantities together provide a first order approximation (the isoquant line) and quantities, prices and elasticity parameters together provide a second-order approximation (the isocost curve) of the underlying technology.

```
  e:pf("lab")        q:endl
  e:pf("cap")        q:endc
```

where the prices `pu`, `pf("lab")` and `pf("cap")` are declared in the `$commodities:` block :

```
$commodities:
  pd(o)              ! domestic price of commodity
  pf(f)              ! price of primary factor
  pu                 ! price index for utility
```

The representative agent derives then his utility from the consumption of all `o` goods as you can see from the following :

```
$prod:u              s:esubc
  o:pu               q:(sum(o, demand(o)))
  i:pd(o)            q:demand(o)
```

where `esubc` is the elasticity of substitution between commodities `pd(o)`.

In this model, the total endowment of labour and capital are exogenous. When you solve the model, the prices `pf("lab")` and `pf("cap")` adjust to clear the markets for labour and capital. Demand for labour equals the total endowment of labour. Suppose that `endl_b` and `pf_b("lab")` are equal to the benchmark (that is, pre-simulation) quantity and price of labour respectively [in the discussion of MPSGE models, we often add "**`_b`**" to the names to denote benchmark values]. Then the pre-simulation value of labour is equal to `pf_b("lab")*endl_b`. Suppose that, in a simulation, the supply of labour is exogenous and unchanged. Then the post-simulation quantity of labour will also be `endl_b` while the post-simulation value of labour will be `pf_s("lab")*endl_b` [we add "**`_s`**" to the names to denote post-simulation values].

Suppose now that you wish to link wages `pf("lab")` to the consumer price index `pu`. For example, you may want to model a policy in which percentage changes in `pf("lab")` and percentage changes in `pu` must be the same. [This policy, called wage indexation, was used in Australia for many years as part of a centralised system of wage fixing.]

In order to model this policy, you need to add an equation that links `pf("lab")` and `pu`. Clearly such an equation is inconsistent with the simple model we have described above, in which `pu` and `pf("lab")` are determined to make their separate markets clear. So some other change must be made in the model to accommodate this behaviour.

Suppose that, in some simulation with the simple model (without wage indexation), `pu` increases more than `pf("lab")`. If `pf("lab")` were linked to `pu`, this says that `pf("lab")` should increase above the level which clears the labour market. In that case, the labour market would not clear and the demand for employment would be less than the supply of labour.

This suggests one way of modelling wage indexation. The key idea is that there may be less than full employment.

In MPSGE, this can be done by introducing a **rationing variable** `epl` into the endowment for labour. [This variable allows the model to break out of the normal MPSGE straight-jacket.] There are 3 changes in the model to make this change.

1. The `e:` line for `pf("lab")` in the `$demand:ra` block is changed to

```
  e:pf("lab")      q:endl           r:epl
```

2. The variable `epl` is added as an **auxiliary variable** via the statements

```
$auxiliary:
  epl                ! employment index
```

3. A `$constraint:` relating to this auxiliary variable `epl` is added. [This constraint block contains the equation which takes the model out of the straight-jacket.]

If your MGE file is to be used with GAMS or GEMPACK, this would be[19]

```
$constraint:epl
  pf("lab") =e= pu;
```

Alternatively, if your MGE file is to be used with GEMPACK, this could be

[19] You can write levels or linear constraint equations – see section 7.6 for details.

```
        $constraint:epl
          (linear)  pf("lab") = pu;
```

Most importantly, the interpretation of variables changes.

- In the version without rationing, `endl` (the quantity in the `q:` field in the `e:pf("lab")` line) represents the quantity of labor demanded and supplied.
- In the version `employ.mge` with rationing, the quantity of labor demanded (and supplied) depends on the values of `endl` and of the rationing variable `epl`. By definition, the quantity of labor demanded and supplied is equal to

  `endl*epl`

Thus, at the benchmark, the quantity of labour supplied (and demanded) `QS_RA_PFLAB_B`[20] is equal to `epl_b*endl_b,` while the value of labour supplied, `VS_RA_PFLAB_B,` is equal to `pf_b("lab") *epl_b*endl_b`. In a simulation, `endl` is often exogenous and unchanged, while `pf("lab")` and `epl` are usually endogenous (and changing). In that case, the post-simulation quantity of labour supplied (and demanded) `QS_RA_PFLAB_S` is equal to `epl_s*endl_b` while the value of labour supplied `VS_RA_PFLAB_S` is equal to `pf_s("lab")*epl_s*endl_b`. You can see that the ratio `QS_RA_PFLAB_S/QS_RA_PFLAB_B` is equal to `epl_s/epl_b`.

Hence the variable `epl` has a natural interpretation, namely as an index of labour supply. [Suppose that `endl_b` =100 and `epl_b` =1, and suppose also that `epl_s`=0.8. Then the pre-simulation and post-simulation quantities of labour are 100 and 80 respectively. The ration `epl_s/epl_b` (equal to 0.8) indicates that labour supply has fallen to 80% of its pre-simulation value due to the simulation shocks.]

The **rationing field** `r:epl` is in the `e:` line for `pf("lab")`. The value of `epl` affects the demand and supply for labor and so this variable `epl` enters into both the revenue equation for `ra` and the market clearing equation for `pf("lab")`.

- The **revenue equation** for `ra` is modified so that agent `ra` only accrues revenue corresponding to the actual amount of labour demanded. In the simple model (without wage indexation), the revenue accruing to `ra` from labour will be equal to `pf_b("lab")*endl_b` in the initial benchmark. [This is the price times the size of the endowment.] When rationing is present, the revenue accruing to `ra` from labour will be `epl_b*pf_b("lab")*endl_b` in the initial benchmark and will be `epl_s*pf_s("lab")*endl_b` (assuming that `endl_b` stays fixed) post-simulation. This revenue depends not only on `pf("lab")` and `endl` but also on the (endogenous) value of `epl`.
- The **market clearing equation** for labour says that demand equals supply for labour. In the simple model (without wage indexation), the quantity of labour supplied is equal to `endl_b` at the benchmark. When rationing is present, the benchmark quantity of labour supplied is equal to `epl_b*endl_b` while the post-simulation quantity of labour supplied is equal to `epl_s*endl_s`. Hence `epl` enters into the market clearing equation for labour.

In the simple model (without wage indexation) we often find it useful to think that `plab` is determined by the market clearing equation for labour (and that the price `pc` of commodities is determined by the market clearing equation for commodities).[21]

[20] In this discussion (and elsewhere in the paper), we often use names for variables which are similar to those used in the GEMPACK implementation of the models (as produced by the program MGE2GP). [For example, this name `QS_RA_PFLAB` has `RA` and `PFLAB` to indicate that it occurs in the `i:pf("lab")` line of the `$demand:ra` block. The `QS` at the start indicates Quantity Supplied.] And we consistently add "`_B`" to denote benchmark values and "`_S`" to denote post-simulation values.

[21] Many of the best expositors of model results find it useful to think of each equation as determining a specific endogenous variable. They know that the equations of the model are a simultaneous system, but they don't let that dissuade them from giving a sequential explanation of the results, starting from the shocks. We encourage you to do the same.

In the model with wage indexation, the constraint equation links `pf("lab")` and `pu`. It is still useful to think of `pu` as being determined by the market clearing equation for commodities. But then `pf("lab")` is determined by the constraint equation. So what does the market clearing equation for labour determine? Well, you can think of it as determining the value of the rationing variable `epl`. That is, the market clearing equation for labour determines the value of `epl`, which in turn determines the quantity `QS_RA_PFLAB` of labour supplied (and demanded) since `QS_RA_PFLAB=epl*endl` (and `endl` is usually exogenous and unchanged). Thus, in the model with wage indexation, it is useful to think of the market clearing as determining the quantity of labour supplied.

That is the major difference between the two models. In the simple model, the market clearing equation determines the price of labour. In the model with wage indexation, this equation determines the quantity of labour, while `pf("lab")` is determined by the constraint equation.

Notes About This Model

You need to be careful in interpreting the value assigned to `endl` in the `e:` line

```
  e:pf("lab")       q:endl            r:epl
```

In the simple model without rationing, `endl` is equal to the quantity of labour. In the model with rationing, the quantity of labour supplied and demanded is equal to `epl*endl`.

In a GEMPACK implementation of the simple model (without wage indexation), it would seem natural to hold `endl` on the database and to read it. It represents the quantity of labour supplied and demanded in the benchmark. In the model with wage indexation, you need to also hold the value of `epl` on the database since it is a vital part of the benchmark solution implied by the database. Now you have to be careful to look at both the values of `endl` and `epl` on the data base if you want to know the quantity of labour at the benchmark, since this quantity is equal not to `endl` but to `epl*endl`.

In the GEMPACK implementation of the model with wage indexation, we refer to `endl` as the **unrationed** quantity of labour. This variable `endl` has no natural economic interpretation (unlike `epl*endl`). The variable `endl` is included merely to keep the accounting correct in the model.

2.4.2 Allowing Trade Deficit or Surplus – BOP.MGE

The second example of rationing shows you how to let the balance of payments adjust to accommodate changes in GDP. In our small-open economy model (see `open.mge` in section 2.2.3), the standard closure is to consider that the representative agent may not increase or decrease his initial foreign lending, which means that the trade surplus is fixed to its benchmark value. The real price of foreign exchange adjusts then to clear the market.

Suppose now that you want to simulate the impact of an increase in aggregate consumption. Then, in order to let aggregate output adjust to the shock, you need to free one of its components. In this small model, GDP is composed of private consumption, government consumption and net exports. One way of getting out of the MPSGE straight-jacket is to let the balance of payments adjust for movements in GDP. In the example, `bop.mge`, this is implemented by the introduction of an auxiliary variable `bop` which defines an index for the balance of payments, in this case, an index for the surplus.

```
$demand:ra
  d:pu
  e:pf(f)         q:endow(f)
  e:pfx           q:-bbop          r:bop
```

The associated constraint is to fix aggregate consumption since this is the variable you need for the experiment. Note that aggregate consumption is denoted by `u` since it is a measure for utility.

```
$constraint:bop
  (linear)  u = 0;
```

The same discussion as in the first rationing example applies here except that `bop_s` may be smaller than zero. In the benchmark, `bop_b` is equal to one reflecting the initial surplus equal to `bbop`. A large increase in aggregate consumption then may cause trade balance to move toward deficit, meaning that `bop_s` is smaller than zero.

2.5 Endogenous Taxes

As we say at the beginning of section 2.4, the natural closure in MPSGE has exogenous taxes. However many empirical models do not fit into this structure. In particular when you consider some tax reform experiments, you modify the level of each replacement tax in order to maintain an equal yield. However, as a result of the endogenous response of prices and quantities, this will be true only at the benchmark when tax rates are exogenous. In order to perform differential (equal yield) tax policy analysis, it is therefore necessary to accommodate the endogenous determination of tax rates as part of the equilibrium computation.

Within MPSGE endogenous taxes are introduced through auxiliary variables. There are two fields associated with an endogenous tax.

- The `n:` field gives the name of the auxiliary variable which will allow the tax rate to adjust endogenously.
- The `m:` field specifies an exogenous tax multiplier.
- The actual tax rate is equal to the product of the values in the `m:` and `n:` fields.
- The auxiliary variable specified in the `n:` field is associated with its `$constraint:` equation. The equilibrium level of the auxiliary variable is selected to satisfy its associated equation even if the variable does not appear in the equation.

If the `m:` field is omitted, the MPSGE convention is that the multiplier is taken to be equal to 1 (that is, the default is `m:1`). If the value in the `m:` field is zero, there is no tax (since the m: field value is multiplied by the n: field value to give the actual tax rate).

2.5.1 Endogenous Taxes Example – DIFFTAX.MGE

This example is based in the TAXOUT model – see section 2.3.2. In DIFFTAX, the tax on domestic production (which is exogenous in TAXOUT) is made endogenous. The endogenous tax rate `to` (in the `n:to` field) adjusts so that real government demand remains unchanged.

The relevant parts of `DIFFTAX.MGE` are the `$prod:y(s)` block which shows the tax and the `$constraint:to` block which shows the equation used to keep real government demand fixed. These are shown below.

```
$prod:y(s)    t:etrn(s)          va:esub(s)
  o:pd(o)     q:supply(s,o)   a:gov  n:to      m:bto(s,o)
  i:pd(o)     q:interm(o,s)
  i:pf(f)     q:factor(f,s)   p:bpf(f,s)   a:gov   t:btf(f,s)  va:
```

```
$constraint:to
* Hold real government consumption fixed
* In the levels, real govt consumption is GOV/PG=SUM(o,DEMAND(o))
    (linear)  gov -pg = 0;
```

As you can se, we have chosen to write the constraint equation as a linearized equation, which is indicated by the qualifier "`(linear)`". This equation requires that the percentage change `gov` in government revenue remain equal to the percentage change `pg` in the price index for government expenditures, which leaves the percentage change in real government consumption `(gov-pg)` equal to zero.[22]

[22] An alternative modelling assumption would be to keep government revenue (rather than real demand) fixed. In that case, the `$constraint:to` block would be:
```
$constraint:to
* Hold real government income fixed
    (linear)  gov = 0;
```
We chose not to do this since this equation is not homogeneous in prices, which means that the homogeneity simulation we recommend with these models (see sections 4 and 5) does not give the expected results.

In the database `DIFFTAX.HAR` which accompanies this model, the value of `to` is zero, which means that there is no tax in the benchmark. The values of the multipliers `bto(s,o)` are all equal to 1. So, if `to` becomes nonzero in response to a shock, the same *ad valorem* tax rate will apply to all outputs `supply(s,o)`.

The simulation in `DIFFTAX.CMF` which accompanies this model, the benchmark ad valorem rate of tax on capital in sector `sind` is 100%. In `DIFFTAX.CMF`, this tax is removed and replaced by an endogenous tax on outputs of commodities, with to adjusting so that real government demand stays unchanged. Extra revenue for `gov` must be generated to compensate for the revenue lost by the removal of taxes on capital.

An alternative would be to keep government revenue fixed. In that case, the `$constraint:to` block would be:

```
$constraint:to
* Hold real government income fixed
    (linear)  gov = 0;
```

We chose not to do this since this equation is not homogeneous in prices, which means that the homogeneity simulation we recommend with these models (see sections 4 and 5) does not give the expected results.

2.6 Other Advanced MPSGE Features

2.6.1 Exception Handling

MPSGE is designed so that all and only those variables which are declared are actually employed in a particular model. This form of "explicit declaration" helps to catch nuisance bugs in large dimensional datasets. It means however that some care must be exercised when a database includes "missing goods". Considered, for example, a model in which `s` is the set of goods in the underlying database, and `p(s)` is the associated vector of prices for these commodities. If a subset of the goods is missing from the database (in the sense that there is zero production of commodities in this subset), then the declaration of `p(s)` must be restricted to those goods which actually appear in the model.

The operator `$` provides you a way of handling this kind of exceptions in declaration and definition blocks. It also allows you to define a specific production structure for elements `s` in a subset `t(s)` with respect to other elements in `s` but not in `t(s)`.

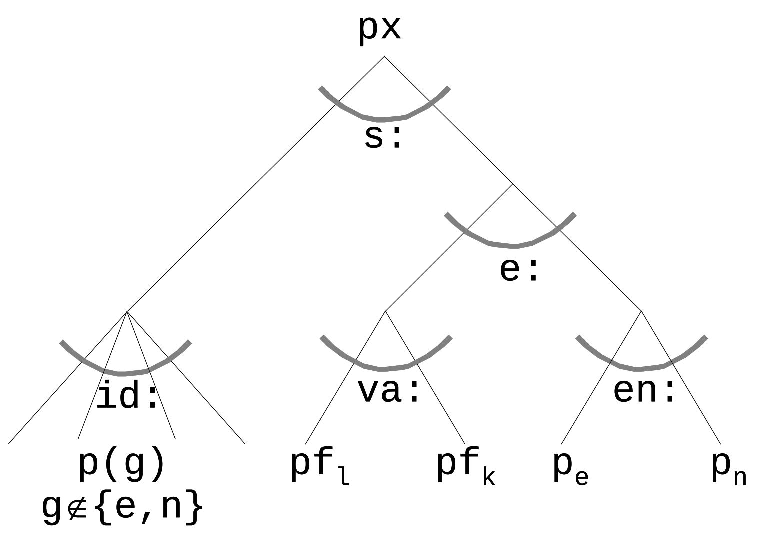
The exception operator `$` can be used on virtually any entry in the MPSGE input file. In particular, conditional assignments may be applied to nest assignments for inputs and outputs. This permits arbitrary assignments of elements from a single vector input to multiple nests.

2.6.2 Sets of Nests

As described in section 2.2.2, nest identifiers `s:` and `t:` are reserved for top level substitution and transformation elasticities, respectively. If you specify a different name with no "parent" nest, as `va:` in `joint.mge` for example, MPSGE supposes that this is an input subnest. In other words, `va:` is automatically assumed to be `va(s):`. When you want to introduce an output subnest, you must specify the output parent nest `t`, i.e. `id_nest(t):`.

Suppose now that you want to introduce a different subnest for each composite input or output in a `$prod:` block. Then, instead of listing all commodities with a different subnest identifier, you may wish to use the notation `s.tl:` for the subnest identifier. This will generate for you a *set of nests*, one for each element of set `s`. The letters `tl` tells MPSGE to display the individual element *text labels* of set `s`. Therefore, you don't need to specify a different subnest identifier, MPSGE is going to use each element label of set `s` as the subnest identifier for the composite commodity `s`.

The concept of parent nests is usually extended to more than two levels of nesting. This is illustrated by the figure below. Suppose that you have a 3-level nest structure and you want a given composite commodity in the third level to depend only on a specific composite commodity in the second level, then you need to make explicit the parent nest of the third-level composite commodity. On the `$prod:` line, the third-level nest identifier is therefore indicated by `id_nest(parent_id_nest):`.



2.6.3 Spanning Operator

Dealing with non-separable functions is possible in MPSGE. The spanning operator # lets you to introduce multiple inputs or outputs of a single commodity. For example, if you want a commodity `p` to enter each nest `s` of a cost function, you need to specify it through the spanning operator in the corresponding `i:` field, `i:p#(s)`. This tells MPSGE to introduce one input coefficient for each element of set `s`. When s is part of the function domain, then no argument is given to the spanning operator, i.e. `i:p#()` in our example.

The spanning operator is usually combined with the sets of nests feature for representing margins.

2.7 Scope of MPSGE

Experienced MPSGE users know how to break out of the straight-jacket in many different ways. For example, it is relatively straightforward to implement the following features in MPSGE:

- Technical change
- Margins
- Decreasing returns to scale
- Increasing returns to scale
- External economies of scale

Indeed, it is possible to implement even more complicated features like

- AIDS – Almost Ideal Demand Systems
- CRESH – Constant Ratio of Elasticity of Substitution Homothetic

though doing so may not be worth the trouble.

2.8 Documentation of MPSGE Syntax

You can find this in Appendix 1 (section 10).

3. Converting MPSGE Models to GEMPACK

Suppose that you have built a model by writing down an MPSGE file describing the model and collecting data that represent a solution of your model. If you wish to solve that model using GEMPACK, you proceed as follows.

In the text below, we suppose that the MPSGE file for your model is `MODEL.MGE`.

1. Organise any data which is not explicitly given in `MODEL.MGE` into a GEMPACK Header Array file `MODEL.HAR`. Include headers giving the elements of the sets referred to in the MGE file.
2. Run the program MGE2GP to convert the MPSGE representation of your model to a GEMPACK representation. For example, run MGE2GP to convert `MODEL.MGE` to a GEMPACK TAB file `MODEL.TAB`, a GEMPACK Command file `MODELHSIM.CMF` and a GEMPACK Stored-input file `TMODEL.STI` for running the GEMPACK program TABLO.
3. Use GEMPACK software to solve the model starting from `MODEL.TAB`, `TMODEL.STI`, `MODELHSIM.CMF` and `MODEL.HAR`.

Steps 1 and 2 can be done in any order although, ideally, step 1 is done before step 2.

We describe step 2 in this section. We describe step 3 in detail in sections 4 and 5 below. There are various ways of doing step 1, as we describe in section 7.4 below.

3.1 Converting an MGE File to GEMPACK TAB and Command Files

We provide a program MGE2GP that can be used to convert an MPSGE model to a form that can be used to solve the model using GEMPACK. You can download this program from the web – see section 1.1.

In this section we describe how you can run this program and give some documentation about it.

3.1.1 Running MGE2GP to Convert to GEMPACK

You can run the program MGE2GP most easily from the command line. That is, go to a DOS-like box and type in the relevant command.

> Suppose, for example, that you have `MODEL.MGE` in directory `C:\MODEL` and that program `MGE2GP.EXE` is in directory `C:\MYPROGRAMS`. Then, go to a DOS-like box, change directory into `C:\MODEL` and type in the command
>
> **C:\myprograms\mge2gp model**
>
> This will produce files `MODEL.TAB`, `TMODEL.STI` and `MODELHSIM.CMF` in directory `C:\MODEL`. The first of these is the GEMPACK TABLO Input file for the model, the second is a Stored-input file for running the GEMPACK program TABLO and the third is a GEMPACK Command file for carrying out a homogeneity simulation with the model.
>
> When you install the MGE2GP package (program `MGE2GP.EXE` and example models – see section 1.1), you will be advised to put the program `MGE2GP.EXE` into a directory which is on your PATH. In that case, you just need to type
>
> **mge2gp model**
>
> (whatever directory you are in).

Note that the program MGE2GP requires the suffix of the file containing the MPSGE model to be **`.MGE`**.

3.1.2 Nests

If your MGE file contains nests (see section 2.2.2), the program MGE2GP automatically rewrites your MGE file by adding extra `$prod:` functions to remove the nests. The TAB, STI and Command files produced are based on the rewritten MGE file.

Suppose, for example, that you have `MODEL.MGE` which contains nests. Then MGE2GP will produce a new file `MODEL_MGE2GP.MGE` which is an alternative MGE representation of the same model, but with extra `$prod:` blocks to remove the nests. Then MGE2GP produces `MODEL.TAB`, `TMODEL.STI` and `MODELHSIM.CMF` which are based on the rewritten `MODEL_MGE2GP.MGE`. However you can see the original `$prod:` functions in `MODEL.MGE` inside the comments in `MODEL.TAB`.

Consider for example `joint.mge` shown below. You can see that the `$prod:y(s)` block contains a nest because of the `va:esub(s)` on that line. [See section 2.2.2 for more details.]

```
$model:joint

$sectors:
  y(s)              ! production
  u                 ! utility index

$commodities:
  pd(o)             ! domestic price of commodity
  pf(f)             ! price of primary factor
  pu                ! price index for utility

$consumers:
  ra                ! representative agent income

$prod:y(s)          t:etrn(s)           va:esub(s)
  o:pd(o)           q:supply(s,o)
  i:pd(o)           q:interm(o,s)
  i:pf(f)           q:factor(f,s)       va:

$prod:u             s:esubc
  o:pu              q:(sum(o, demand(o)))
  i:pd(o)           q:demand(o)

$demand:ra
  d:pu
  e:pf(f)           q:endow(f)
```

When MGE2GP runs on `joint.mge`, it first rewrites the file to produce `JOINT_MGE2GP.MGE` as shown below. Note that the original `$prod:y(s)` block is broken into two `$prod:` blocks, namely the modified `$prod:y(s)` block shown below and the additional `$prod:d_va_y(s)` block. Notice also that the original nested `$prod:y(s)` block is shown as an MGE comment (lines begin with *) after the first line of the new version of this block.

```
$model:joint

$sectors:
        y(s)       ! production
        u          ! utility index
        d_va_y(s)  ! demand index for input nest va in sector y(s)

$commodities:
        pd(o)      ! domestic price of commodity
        pf(f)      ! price of primary factor
        pu         ! price index for utility
        p_va_y(s)  ! price index for input nest va in sector y(s)

$consumers:
        ra         ! representative agent income
```

```
$prod:y(s) s:0 t:etrn(s)
* Original, nested function is:
* $prod:y(s)          t:etrn(s)           va:esub(s)
*   o:pd(o)           q:supply(s,o)
*   i:pd(o)           q:interm(o,s)
*   i:pf(f)           q:factor(f,s)       va:
     o:pd(o)           q:supply(s,o)
     i:pd(o)           q:interm(o,s)
     i:p_va_y(s) q:(sum(f,factor(f,s)))

$prod:d_va_y(s) s:esub(s)
     o:p_va_y(s)  q:(sum(f,factor(f,s)))
     i:pf(f)           q:factor(f,s)

$prod:u             s:esubc
  o:pu              q:(sum(o, demand(o)))
  i:pd(o)           q:demand(o)

$demand:ra
  d:pu
  e:pf(f)           q:endow(f)
```

In the resulting file `JOINT.TAB`, the `$prod:y(s)` block (as rewritten in `JOINT_MGE2GP.MGE` – see above) is shown as a strong comment before the TABLO statements implementing that block.

3.2 The Equations and the TAB File Written

One of the reasons for providing the program MGE2GP is to let developers of MPSGE models see the equations underlying their model. In this section we describe the equations and give some details as to how they look in the TAB file written by MGE2GP.

3.2.1 The Equations

The equations underlying an MPSGE model are in four groups.

- There are equations associated with a `$demand:` block.
- There are equations associated with a `$prod:` block.
- There is one market clearing equation which are associated with every `$commodity`.
- There are the `$constraint:` equations. We don't say anything more about these equations here since they are visible in the MGE file.

3.2.2 Equations for a `$demand:` Block

We look at `$demand:` blocks first since the associated equations are simpler than for `$prod:` blocks.

The main equation associated with every `$demand:` block is the equation which adds up the total income for the agent.

> **Example**. Consider the `$demand:ra` block in TWOBYTWO.MGE. This block is
>
> ```
> $demand:ra
>   d:pc
>   e:pl              q:70
>   e:pk              q:80
> ```
>
> The income for agent **`ra`** is the sum of the value of the labour supplied plus the value of the capital rentals. In the notation used in the TAB file TWOBYTWO.TAB produced by MGE2GP, this equation is :
>
> ```
>   VI_RA = VS_RA_PL + VS_RA_PK ;
> ```
>
> The variables are `VI_RA` (**V**alue of **I**ncome for **`RA`**), `VS_RA_PL` (**V**alue of **S**upply by **`RA`** of commodity **`PL`**) and `VS_RA_PK` (ditto for **`PK`**).

There are also equations relating the price, quantity and value of these endowments. For labour, this equation is :

```
  VS_RA_PL = PL_L * QS_RA_PL ;
```

The variables on the right-hand side are `PL_L` (market price of commodity **PL** in **L**evels value – as distinct from the percentage change in the price, which is what the variable `pl` represents in the TAB file), and `QS_RA_PL` (**Q**uantity **S**upplied by **RA** of commodity **PL**).

3.2.3 Equations for a `$prod:` Block

The main equations associated with every `$prod:` block are :

♦ the CES demand and CET supply functions.

Example. Consider the `$prod:x` block in TWOBYTWO.MGE. This block is

```
$prod:x             s:1
  o:px              q:100
  i:pl              q:50
  i:pk              q:50
```

If you are a GAMS modeller, you would expect the CES demand function (in calibrated share form) for labour in sector X to look something like :

$L_x = L0_x \cdot (P_x/P0_x \cdot P0_l/P_l)^{\sigma} \cdot X/X0$

The linearized representation of this equation is :

$l_x = x - \sigma \cdot (p_l - p_x)$

Here you can think of

- l_x as representing the %-change in the demand for labour in this sector,
- x as representing the %-change in the total output of this sector,
- σ as being the elasticity of substitution between labour and capital (its value is 1.0 here because of the `s:1` in the first line of the `$prod:x` block),
- p_l as representing the %-change in the price of labour, and
- p_x as representing the %-change in the price of the output x of this sector.

This equation says that

- if there is no relative price movement between p_l and p_x (that is, if $p_l = p_x$), then the %-change in the demand for labour in sector x is equal to the %-change in the output of the sector.
- if the price p_l of labour increases by one percent more than the price p_x of commodity x, and if there is no change in the output of this sector (that is, $x = 0$), then the %-change in the demand for labour in this sector will fall by σ percent.

In the TAB file TWOBYTWO.TAB produced by MGE2GP, this equation is written as

```
  p_qd_x_pl = x - 1 * [pl - mc_x] ;
```

Here

- `p_qd_x_pl` represents the %-change (**p_**) in `QD_X_PL` (**Q**uantity **D**emanded in sector **X** of commodity **PL**),
- `x` represents the %-change in the output of sector x,
- you can see that σ is set equal to 1,
- `pl` represents the %-change in the price of labour, and
- `mc_x` represents the %-change in the **M**arginal **C**ost of commodity **X**.

♦ the zero profit condition which says that costs = revenue.

Example. Consider again the `$prod:x` block in TWOBYTWO.MGE. [This block is shown above.]

The zero profit equation in the TAB file TWOBYTWO.TAB produced by MGE2GP is written as the following linearized equation:

```
mr_x = mc_x ;
```

The variables are `mr_x` (%-change in **M**arginal **R**evenue from sector **X**) and `mc_x` (%-change in **M**arginal **C**osts in sector **X**).[23]

There are also

- the equations relating to any taxes.

Example. Consider the `$prod:y(s)` block in TAXOUT.MGE. This block is

```
$prod:y(s)    t:etrn(s)           va:esub(s)
  o:pd(o)     q:supply(s,o)  a:gov  t:bto(s,o)
  i:pd(o)     q:interm(o,s)
  i:pf(f)     q:factor(f,s)  p:bpf(f,s)  a:gov  t:btf(f,s) va:
```

Below we look at the equations relating to the tax `bto` on output (the `o:pd` line above).

- One equation calculates the value of the taxes on outputs. This equation is:
  ```
  TOVYPDGOV(s,o) = TORYPDGOV(s,o) * VS_Y_PD(s,o) ;
  ```
  The variables are
  `TOVYPDGOV` (**T**ax on **O**utput re**V**enue in sector **Y** on output **PD** going to agent **GOV**),
  `TORYPDGOV` (**T**ax on **O**utput **R**ate in sector **Y** on output **PD** going to agent **GOV**),
  `VS_Y_PD` (**V**alue of **S**upply in sector **Y** of commodity **PD**). [The market price (`pd` in this case) in an `o:` line is always inclusive of taxes – this is an MPSGE convention – see section 2.3.2.[24]]

- Another equation calculates the total revenue in this sector. This equation is :
  ```
  R_Y(s) = SUM{o, [VS_Y_PD(s,o) - TOVYPDGOV(s,o)]} ;
  ```
  The variable on the left-hand side is `R_Y` (total **R**evenue in sector **Y**). The variables on the right-hand side have been introduced just above. The `TOVYPDGOV` part goes to agent `GOV` and only the rest of `VS_Y_PD` is revenue for sector `Y`.

If you are a GEMPACK expert you will be a little surprised by the syntax in the two equations shown above. Here we are using a slightly schematic syntax (more like the GAMS syntax than the GEMPACK syntax). Of course the TAB file does have these equations written in strict GEMPACK syntax.

If you are a GAMS expert, you should note that the syntax used in the TAB file for the two equations above is slightly different from the way they are written above. You can look at the TAB file TAXOUT.TAB when you come to do a simulation with that model later, in section 5.3.

3.2.4 Supply, Demand and Market Clearing Equations

For every commodity declared in a `$commodity:` block, there is an associated market clearing equation which says that the total quantity demanded is equal to the total quantity supplied.

Preceding the market clearing equation for any commodity are two Formula&Equations which calculate the total quantity demanded and supplied respectively.

[23] The actual equation in the TAB file is a little more complicated. See section 3.2.5 for a brief explanation.

[24] This is in contrast to the market price in an `i:` line. It is an MPSGE convention that the market price does not include any taxes in that case (see section 2.3.1). [For example, in the second `i:` line in the `$prod:y(s)` block, the market price `pf(f)` does not include the taxes going to consumer `gov`.]

All commodities are supplied to the market either as an initial positive endowment or as a produced good (or as both an initial positive endowment and a produced good). All commodities are taken from the market either as an exogenous "negative endowment" or as a production input or as a good demanded by households.

The total supply of a commodity `p(i)` is the sum of the following three terms:

- the sum of the quantities in all lines beginning `o:p(i)` in `$prod:` blocks.
- the sum of positive exogenous endowments which are found in `$demand:` blocks in lines beginning `e:p(i)` which have no `r:` field.
- the sum of positive endogenous endowments which are found in `$demand:` blocks in lines beginning `e:p(i)` which have an `r:` field. Here the supply is equal to the value in the `q:` field times the value in the `r:` field (see section 2.4). [The `r:` field is what makes this an endogenous endowment.]

The total demand for good `p(i)` is the sum of the following four terms:

- the sum of demands associated with `i:p(i)` records in `$prod:` blocks.
- the sum of final demands `d:p(i)` in `$demand:` blocks. For example, consider
  ```
  $demand:state(r,i)
          d:p(i)
  ```
  Here the associated demand for `p(i)` is `SUM[r, state(r,i)/p(i) ]`.
- the sum of negative exogenous endowments which are found in `$demand:` blocks in lines beginning `e:p(i)` which have no `r:` field.
- the sum of negative endogenous endowments which are found in `$demand:` blocks in lines beginning `e:p(i)` which have an `r:` field. Here the supply is equal to the value in the `q:` field times the value in the `r:` field (see section 2.4). [The `r:` field is what makes this an endogenous endowment.]

On the supply side there are two contributions to the market total: producer outputs and initial (positive) endowments. On the demand side there are three contributions to the market total: producer demands, negative "endowments" and final demands.

Example. Consider the market clearing equations for `pc` and `pl` in TWOBYTWO.TAB.

- **1. Total demand for `pc`**. The commodity `pc` occurs in a `d:` line in the `$demand:ra` block. Hence the total quantity demanded for `pc` is given by the equation (in the levels):
  ```
    TQD_PC = VI_RA / PC_L ;
  ```
  Here TQD_PC stands for the **T**otal **Q**uantity **D**emanded of commodity **PC**, while VI_RA stands for **V**alue of **I**ncome of consumer **RA** and PC_L is the levels value of the price of commodity PC.[25]
  2. Total supply of `pc`. The commodity `pc` also occurs in the line
  ```
     o:pc    q:150
  ```
  in the `$prod:c` block. Hence the pre-simulation total supply of commodity `pc` is equal to 150. In the TAB produced by MGE2GP, the equation for the total supply of commodity `pc` is written as:
  ```
    TQS_PC = QS_C_PC ;
  ```
  Here TQS_PC stands for the **T**otal **Q**uantity **S**upplied of commodity **PC**, while QS_C_PC stands for **Q**uantity **S**upplied in `$prod:`**`c`** corresponding to line `o:`**`pc`**. [QS_C_PC is equal to 150 (the value in the q: field in this line) at the start of the simulation. This value may change during the simulation.]
  3. The market clearing equation for commodity `pc` simply says that (in the levels)
  ```
     TQD_PC = TQS_PC ;
  ```

[25] An earlier equation in the TAB file calculates VI_RA (see section 3.2.2).

- **1. Total demand for `pl`**. There are i:pl lines in the $prod:x and $prod:y blocks. Hence the total quantity demanded for `pl` is given by the equation:
  TQD_PL = QD_X_PL + QD_Y_PL ;
  Here TQD_PL stands for the **T**otal **Q**uantity **D**emanded of commodity **PL**, while QS_X_PL and QS_Y_PL stand for **Q**uantity **S**upplied in `$prod:x` or `$prod:y` respectively relating to the relevant `i:pl` line. [These quantities are 50 and 20 respectively in the initial database.]
  2. Total supply for `pl`. The only relevant line is the `e:pl` line in the `$demand:ra` block. Hence the total quantity supplied for `pl` is given by the equation
  `TQS_PL = QS_RA_PL ;`
  [In the pre-simulation database, QS_RA_PL is equal to the value 70 in the `q:` field in the `e:pl` line in the `$demand:ra` block. Hence total supply equals total demand in the database.]
  3. The market clearing equation for commodity `pl` simply says that
  `TQD_PL = TQS_PL ;`[26]

3.2.5 The Equations in the TAB File

As you probably know, TAB files in GEMPACK can contain a mixture of levels and linearized equations.

Most of the equations in the TAB file are as discussed above (with the addition of syntax required by GEMPACK) – that is, are levels equations.

As you saw above in section 3.2.3 above, the CES demand and CET supply functions are written as linearized equations in the TAB file.

Some of the equations in the TAB file have extra conditions and extra terms attached. These are usually ways of handling modelling and numerical issues which arise if some quantity is zero.[27]

3.2.6 Grouping of Equations in the TAB File

Open the TAB file `TWOBYTWO.TAB` produced by running MGE2GP to convert `TWOBYTWO.MGE`. Indeed, we recommend that you open this file in the GEMPACK windows program TABmate (which you can do by double-clicking on the TABmate icon which should be on your desktop if you have GEMPACK installed on your PC). We recommend TABmate since that highlights the different parts of the GEMPACK syntax. You should see the following.

An initial section defining Coefficients `BALTOLI` and `BALTOLC`. You should ignore this for the present.

Then come statements declaring the variables in the `$sectors:`, `$commodities:`, `$consumers:` and `$auxiliary:` parts of the MGE file.

---

[26] If you look in TWOBYTWO.TAB as produced by MGE2GP, you will see some extra terms in these equations. For example, the equation for TQD_PL has the extra term
  IF[QS_RA_PL LT 0, -QS_RA_PL]
However you can tell (even if MGE2GP cannot) that this is irrelevant. [This term corresponds to the "negative" endowment case in the general rule for calculating total demand.]

[27] For example, the zero profit condition for `$prod:c` says that marginal revenue is equal to marginal costs in this production. So you would expect the (linearized) equation to read
```
    mr_c  =  mc_c  ;
```
[Here `mr_c` and `mc_c` are the percentage changes in the marginal revenue and costs respectively in `$prod:c`.] But the equation in the TAB file says
```
  IF[R_C NE 0,  mr_c – mc_c] + IF[ NOT[R_C NE 0], c] = 0 ;
```
The usual equation applies provided R_C (the total revenue in `$prod:c`) is nonzero. Otherwise the equation c=0 holds. Clearly in this model, R_C will never be zero. But in larger models and in production sectors over a vector of sectors, the revenue in one sector may be zero.

Then come the equations and other statements for each of the `$prod:` and `$demand:` blocks in the MGE file.

> For example, you can recognise the part of the code relating to the `$prod:x` block since the exact `$prod:x` block from the MGE file is shown as a comment in the TAB file.[28] In TWOBYTWO.TAB, first you will see the equations and other statements for the `$prod:x` block. You should be able to recognise several of the equations from the discussion above. For example, the CES demand function for labour in `$prod:x` is written as
>
> ```
> Equation (Linear) E_p_qd_x_pl   p_qd_x_pl =  x - 1 * [pl - mc_x] ;
> ```
>
> while the connection between the value `VD_X_PL` and the associated price `PL_L` and quantity `QD_X_PL` is written as
>
> ```
> Formula & Equation E_p_vd_x_pl  VD_X_PL = PL_L * QD_X_PL ;
> ```

Then you can see the statements implementing the other `$prod:` and `$demand:` blocks.

The code for a `$prod:` block often indicates (via a comment) the line (`i:` or `o:` line) from which the code comes.

Finally you will see the section for the market clearing equations. Again you can recognise these from the discussion above and from the comments in the TAB file.

> For example, the market clearing equation for commodity `pl` is written as
>
> ```
> ! Market clearing for pl !
> Equation (Levels) E_pl
>   # Market clearing equation for commodity pl #
>    TQD_PL = TQS_PL ;
> ```

Notice also that the market clearing equation for commodity `px` has been omitted to satisfy Walras law. The variable **`WALRASSLACK`** is introduced as a check of market clearing in this sector. The simulation result for `walrasslack` should always be zero (or very close to zero). See section 7.9 for more details about this.

The equations and other statements in TWOBYTWO.TAB are especially easy to read (even if you are not used to GEMPACK) since all variables are scalars. When there are sets and vector and matrix variables, the equations are similar, but have arguments to indicate the vector and matrix nature of the equations. You can see examples of that in section 7.10 below.

3.2.7 Tax Variables in the TAB File

The tax variables in the TAB file have the same arguments as the symbol in the `t:` field (or the `n:` field) to which they correspond.[29]

For example, in TAXIN.MGE you see

```
$prod:y(s)     t:etrn(s)          va:esub(s)
  o:pd(o)      q:supply(s,o)
  i:pd(o)      q:interm(o,s)     p:bpi(s)     a:gov     t:bti(s)
  i:pf(f)      q:factor(f,s)     p:bpf(f,s)   a:gov     t:btf(f,s)   va:
```

Corresponding to the `t:bti(s)` field in the above `i:pd(o)` line you will see

---

[28] In TAB files, ordinary comments start with an exclamation mark **!** and end with an exclamation mark. For example, see the comment

```
! $prod:x !
```

at the start of this part of the TAB file. TAB files can also contain so-called strong-comments which are sections of text which begin with **`![[!`** and end with **`!]]!`**. You can see the whole of the $prod:x block is reproduced as a strong comment. Note that this text is highlighted with a blue background in TABmate.

[29] This represents a change from Version 1 (June 2004) of MGE2GP where the tax variables in the TAB file always had "full rank".

```
Variable(Levels, Change, Linear_Name=c_bti)
 (All,s_1,s) BTI(s_1)
 # Pre-sim value of tax rate on pd(o) for "a:gov t:bti(s)" in $prod:y(s) # ;
```

in the TAB file `TAXIN.TAB`.

Corresponding to the `t:btf(f,s)` field in the above `i:pf(f)` line you will see

```
Variable(Levels, Change, Linear_Name=c_btf)
 (All,f_1,f)(All,s_1,s) BTF(f_1,s_1)
 # Pre-sim tax rate on pf(f) for "a:gov t:btf(f,s)" in $prod:d_va_y(s) # ;
```

in the TAB file `TAXIN.TAB`.

Note that the arguments (indexes) for variables `BTI` and `BTF` in the TAB file are the same as for those in the corresponding `t:` field in the MGE file.

3.3 The MGE2GP Code

The code for the program MGE2GP is written by Tom Rutherford and Ken Pearson. Tom and Ken could not have done this without Laurent Cretegny's knowledge of both GAMS/MPSGE and GEMPACK. For example, Laurent worked out the examples in section 2 and kept checking that we obtained the same results when these models were solved via GEMPACK or GAMS/MPSGE.

The code, which is written in Fortran 90, is in three parts.

- The MGE file is read and checked. Its contents are stored in suitable data structures.
- If there are nests, the original MGE file is rewritten to remove the nests – see section 3.1.2.
- The GEMPACK outputs (TAB, Command and TABLO STI files) are written.

Tom is responsible for the first two parts above while Ken is responsible for the third part.

If you find bugs, or if you have suggestions for improvements in the code, please pass them on to Tom (rutherford@colorado.edu)or Ken (Ken.Pearson@buseco.monash.edu.au).

If you are reporting a bug, please zip up into a single zip file your MGE file and HAR file. Also any output files which show the bug. Send the zip to Ken or Tom.

4. Simulations with the TWOBYTWO Model Using GEMPACK

In this section we show you how you can carry out simulations with the example models provided in the MGE2GP package (see section 1.1).

You will need a version of GEMPACK (Release 8 or later – see section 1.2) to carry out the simulations. If you do not already have GEMPACK, you can use the Demonstration Version of GEMPACK for solving the TWOBYTWO model and many of the models in section 5. You can download the Demonstration Version of GEMPACK from the web – see section 1.2.

In this section you will work with the TWOBYTWO example model. We take you through the different steps in great detail.

In section 5 below, you will carry out simulations with the other example MPSGE models supplied.

If you are an experienced GEMPACK user, we encourage you to glance quickly at the material in this section and in section 5 below. In particular, you may be slightly surprised by the material in section 5.2.1.

Below we show you how to carry out simulations with the first example model, namely **`twobytwo.mge`**. We assume that you have this file in the subdirectory **`c:\mge2gp`**.

4.1 Installing Program MGE2GP

You download the program MGE2GP from the web, as explained in section 1.1. The program MGE2GP is the file `MGE2GP.EXE`.

We recommend that you put this program in a directory which is on your DOS Path. For example, put it in the directory (usually `c:\gp`) in which your GEMPACK programs are installed.

The reason for putting `MGE2GP.EXE` into a directory which is on your Path is that you can then run the program from the Command line just by typing in its name. You don't need to specify the full path name of the EXE file.

4.2 Working at the Command Line

One way to solve the model is to work at the Command line (that is, to work in a DOS-like window). If you prefer to work in a Windows manner, please skip this section and go on to section 4.3.

If you decide to work at the Command line, you need to make sure that the directory in which your GEMPACK software is installed is on the PATH. Below we assume this.[30]

Go to your Command prompt environment. [If you are running Windows NT, 2000 or XP, you can get there by running cmd.exe. If you are running Windows 98 or ME, you can go to a DOS prompt.]

4.2.1 Run MGE2GP to Convert the MGE File to GEMPACK

Change directory into the directory (c:\mge2gp – see above) in which you have MGE file `twobytwo.mge`. You can do this via the commands

```
c:
cd \mge2gp
```

First convert the MGE file `twobytwo.mge` to GEMPACK TAB, STI and Command files by running MGE2GP. You can do this by entering the command[31]

[30] This directory is usually put on the PATH as part of the installation of GEMPACK. If you need to do this manually, see section 3.9 of GPD-6 for details.

[31] This assume that you have installed the program MGE2GP in a directory which is on your Path (as described in section 4.1).

```
mge2gp  twobytwo
```

This should create the three files `twobytwo.tab`, `ttwobytwo.sti` and `twobytwohsim.cmf.`

4.2.2 Run TABLO – Processes the TAB File

Then run the GEMPACK program TABLO by typing in the command

```
tablo  -sti  ttwobytwo.sti
```

[Yes, there are two "t"s at the start of this .STI file name.]

This will run the GEMPACK program TABLO taking inputs from the so-called **Stored-input** file `TTWOBYTWO.STI`. This instructs TABLO to process the TAB file `TWOBYTWO.TAB` and to produce output for GEMSIM.[32]

The main outputs are the so-called GEMSIM Auxiliary files `TWOBYTWO.GSS` and `TWOBYTWO.GST`. These are computer versions of all the statements in `TWOBYTWO.TAB`.

4.2.3 Run GEMSIM to Solve the Model

Then run the GEMPACK program GEMSIM by typing in the command

```
gemsim  -cmf  twobytwohsim.cmf
```

This will carry out the simulation specified in the Command file `twobytwohsim.cmf`. [You will look at this file in section 4.5 below. For the minute, please assume it makes sense and proceed as below to run the simulation.]

The program GEMSIM will produce a Solution file **`twobytwohsim.sl4`**. This contains the simulation results.

If you are interested in seeing how the simulation can be carried out using GEMPACK's windows interfaces, you will see this in section 4.3 below, which you should skip ahead to now.

Otherwise you need to run the GEMPACK Windows program **ViewSOL** which you can do by clicking on its icon (which should be on your desktop).[33] When ViewSOL begins to run, go to the ***File*** menu and click on ***Open*** . Navigate to the directory **`c:\mge2gp`** and open the Solution file **`TWOBYTWOHSIM.SL4`** just created. This contains the results of the simulation. You can now jump to section 4.4 below where we guide you through the results.

4.3 Using a Windows Version of GEMPACK

This assumes that you have access to a Windows version of GEMPACK. Either the Demonstration Version, an Executable-Image Version or a Source-Code Version is suitable.

Start the GEMPACK Windows interface **WinGEM** running by double clicking on its icon (which should be on your desktop). This should give the main WinGEM menu, as shown below, across the top of the screen. [You may need to look closely to see this since WinGEM is rather self-effacing and only occupies a small part of the top of you screen - the rest of the screen is as it was before you double-clicked on WinGEM.]

---

[32] It also instructs TABLO to always use change differentiation (see section 2.2.6 of GPD-2) when linearizing levels equations. This is why we ask you to use a Stored-input file rather than running TABLO interactively.

[33] There is no point in trying to look at the Solution file TWOBYTWOHSIM.SL4 directly in a text editor because it is a binary file, not a text file. ViewSOL will open the Solution file and display the simulation results on the screen.

WinGEM - GEMPACK for Windows

```
File  Simulation  HAFiles  OtherTasks  Programs  Options  Window  Help
```

4.3.1 Set the Working Directory

WinGEM uses the idea of a working directory to simplify choosing files and running programs. This working directory is where all the files for the model you are using are stored.

For the TWOBYTWO model examples here, the working directory needs to be the directory **c:\mge2gp** in which you have the file `twobytwo.mge`. To set this, first click on ***File*** in the main WinGEM menu. This will produce a drop-down menu. In the drop-down menu, click on the menu item

Change both default directories...

The notation we use for the sequence of clicks (first *File* then *Change both default directories)* is

File | Change both default directories...

In the file selection box that appears, choose drive **C:** (or the drive containing your directory \mge2gp if it is on a different drive). Then double-click on **C:\** (this will be at the top of the list of directories shown) and then double-click on the subdirectory **MGE2GP**. [Make sure that the directory name shown in blue above the selection box changes to C:\MGE2GP (or D:\MGE2GP etc if your \MGE2GP directory is on another drive).] Then click on the ***Ok*** button.

4.3.2 Run MGE2GP to Convert the MGE File to GEMPACK

You need to go to a DOS-like box to do this. To get a DOS-like box, select menu item ***File | Shell to DOS in Working Directory*** from WinGEM's File menu.

Then type in the command[34]

mge2gp twobytwo

The program MGE2GP.EXE should convert the MGE file `twobytwo.mge` to a GEMPACK TAB file, a Stored-input file for TABLO and a Command file for carrying out a homogeneity simulation. That is, this should create the three files `twobytwo.tab`, `ttwobytwo.sti` and `twobytwohsim.cmf.`

You will not need this DOS-like box, so you can close it by typing

exit

There are three steps involved in carrying out a simulation using GEMPACK.

Step 1 – Implement the model by running the GEMPACK program TABLO

Step 2 – Solve the equations of the model (run the GEMPACK program GEMSIM)

Step 3 – View the results

WinGEM will guide you through these steps and indicate what to do next. [You will do step 1 in section 4.3.3, step 2 in section 4.3.4 and step 3 in section 4.4.]

4.3.3 Run TABLO – Processes the TAB File

The TABLO Input file is called **TWOBYTWO.TAB**. It contains the equations of the TWOBYTWO model. Choose

[34] This assume that you have installed the program MGE2GP in a directory which is on your Path (as described in section 4.1).

Simulation | TABLO Implement...

A window for TABLO will appear.

In the menu for the TABLO window, select ***Options*** menu item. Then in this menu choose

Run from STI file

You will see that the text to the left of the **Select** button on the TABLO window now says "Stored Input file".

Now click on the **Select** button and choose the Stored-input file called **TTWOBYTWO.STI.** [Yes, there really are two "t"s at the start of this name.]

Now click on the ***Run*** button.

The program runs TABLO in a DOS box and when complete, returns you to the TABLO window. The program TABLO takes its inputs from the so-called **Stored-input** file TTWOBYTWO.STI. This instructs TABLO to process the TAB file TWOBYTWO.TAB and to produce output for GEMSIM.[35]

When TABLO has finished, you will be put back to the TABLO window. There should be no error and you should see a button ***Go to GEMSIM***. Click on this button to proceed to the next step in running a simulation: Step 2 – Solve the equations of the model.

4.3.4 Run GEMSIM to Solve the Model

The button takes you to the window for running the GEMPACK program GEMSIM.

First ***Select*** the Command file called **`TWOBYTWOHSIM.CMF`**. [You will look at this file in section 4.5 below. For the minute, please assume it makes sense and proceed as below to run the simulation.]

Click on ***Run*** to run GEMSIM with the Command file TWOBYTWOHSIM.CMF. When the simulation finishes (there should be no error), you see a window showing the accuracy of the simulation. There should be a smiling face for the Variables accuracy. [If the Data accuracy is frowning, you can ignore that since no data is read in this simulation – all the data is in the MGE file and hence in the TAB file for this simple model.] Click **Ok**.

Then you should see a button ***Go to ViewSOL*** on the GEMSIM window. Click on this button to look at the results, which you will do in section 4.4 below.[36]

4.4 Looking at the Results using ViewSOL

In this section we show you how to look at the results of your TWOBYTWOHSIM.CMF simulation. Whether you have come here from section 4.2.3 or from section 4.3.4 above, you should have the simulation results open in ViewSOL.

You will see the **Contents** page. This shows just one line **Macros** (since all variables are scalar variables for this model).[37]

To see the results for all variables, just **click** on the **Macros** row on the Contents list. You should see several rows. Each row has the name of a variable and four columns of numbers.[38]

---

[35] It also instructs TABLO to always use change differentiation (see section 2.2.6 of GPD-2) when linearizing levels equations. This is why we ask you to use a Stored-input file rather than running TABLO interactively.

[36] There is no point in trying to look at the Solution file TWOBYTWOHSIM.SL4 directly in a text editor because it is a binary file, not a text file. ViewSOL will open the Solution file and display the simulation results on the screen.

[37] Macro in ViewSOL just means scalar variable.

This simulation is one in which the price of the numeraire (which is **px**, the price of commodity **x**) is increased by one percent and all other exogenous variables are left unchanged. This is a so-called price homogeneity simulation. You would expect that all prices and dollar values should increased by one percent and that all quantities should remain unchanged. Below you will look at the simulation results to check this.

For example, you should see the following rows.[39]

```
            twobytwohsim      Pre two..   Post two..  Chng two..
c_vs_ra_pk        0.800       80.000      80.800      0.800
p_qd_y_pk         0.000       30.000      30.000      0.000
p_vd_y_pk         0.000       30.000      30.300      0.300
pl                1.000        1.000      1.010       0.010
px                1.000        1.000      1.010       0.010
```

The second last row above (the **pl** row) means that

- the price of labour has increased by 1.000 percent. [This is the simulation result for variable **pl** and it is in the column headed **twobytwohsim**.]
- the pre-simulation levels value for the price of labour is 1.000. [This is in the column header **Pre twobytwohsim**.]
- the post-simulation levels value for the price of labour is 1.010. [This is in the column header **Post twobytwohsim**.]
- the change in the levels value for the price of labour is 0.010. [This is in the column header **Chng twobytwohsim**.]

This is all consistent with what you expect for a price homogeneity simulation.

The last row above (the **px** row) is much the same as this, expect that this row is shown in red. That is because the variable **px** is exogenous in this simulation (see section 4.5 below). ViewSOL shows exogenous variables in red (the other exogenous ones in this simulation are **c_qs_ra_pl** and **c_qs_ra_pk** which are the changes in the supplies (or endowments) of labour and capital respectively).

The third row above (the **p_vd_y_pk** row) shows the results for this variable which represents the percentage change in the **V**alue **D**emanded in sector **Y** of commodity **PK** (capital). The numbers in this row mean that

- this value has increased by 1.000 percent.
- the pre-simulation value is 30.000. [This is the entry in the `q:` field in the `i:pk` row in the `$prod:y` block in TWOBYTWO.MGE. Market prices (including **pk**) are assumed to be one in the benchmark.]
- the post-simulation value is 30.300. [This is in the column header **Post twobytwohsim**.]
- the change in the levels value is 0.300. [This is in the column header **Chng twobytwohsim**.]

The second row above (the **p_qd_y_pk** row) shows the results for this variable which represents the percentage change in the **Q**uantity **D**emanded in sector **Y** of commodity **PK** (capital). The numbers in this row mean that this quantity has not changed from the pre-simulation quantity of 30.

---

[38] If you do not see four columns of numbers in each row, this is because of the Options setting in ViewSOL. Select ***File* | *Options*** from ViewSOL's main menu. Click to check the line **Show levels results (if present)**. Then click **OK** to close the options window. Then close ViewSOL and reopen the Solution file (in the same way as you opened it before). This time there should be four numbers in each row.

[39] You can change the number of decimal places shown via the drop-down box near the middle of the top row of your screen. Find the drop-down box which shows a figure (anywhere in the range 0 to 6) and change it to 3.

The first row above (the **c_vs_ra_pk** row) shows the results for this variable which represents the **Change** (not the percentage-change – the **c_** at the start is the clue) in the **V**alue **S**upplied by consumer **RA** of commodity **PK** (capital). The numbers in this row mean that this value changed from a pre-simulation value of 80 to a post-simulation value of 80.8, which is indeed an increase of 1 percent, as expected.

You should check the other results shown to see that all prices and dollar values have increased by 1 percent and that all quantities have remained unchanged.

This price homogeneity simulation is one of the few simulations whose results you can predict on theoretical grounds. Later, in section 4.6 below, you will carry out a simulation whose results cannot be easily predicted in advance.

Close ViewSOL in the usual windows way by selecting ***File* | *Exit*** from the main menu.

4.5 Looking at the Command File

Here you will look inside the Command file `TWOBYTWOHSIM.CMF` to get an introduction to the use of Command files in GEMPACK.

- If you are working via WinGEM, go to the GEMSIM window (which should still be visible on your screen) and click on the **Edit** button. This will open the Command file `TWOBYTWOHSIM.CMF` in the windows editor **TABmate**.
- If you are working at the command line, run the GEMPACK windows program **TABmate** by double clicking on its icon which should be on your desktop. Then select ***File* | *Open*** from the main menu and open the file **`twobytwohsim.cmf`** which should be in directory `c:\mge2gp`.

The Command file `TWOBYTWOHSIM.CMF` is shown below (except that we have left out below a few comment lines). We describe in sections 4.5.1 to 4.5.4 below the various parts of this file. We explain in section 4.5.5 why you need a Command file as well as the TAB file for the model.

The GEMPACK Command File TWOBYTWOHSIM.CMF

```
Auxiliary files = twobytwo ;
! No data are read here, hence no updated data will be produced
Extrapolation Accuracy File = yes ;
Log File = yes ;
Method = Euler ;  ! or Gragg
Steps = 4 6 8 ;  ! change as needed

! Closure
! -------
!
exogenous
 c_QS_RA_PL ! Change in quantity of endowment
  ! change in quantity for e:pl in $demand:ra
 c_QS_RA_PK ! Change in quantity of endowment
  ! change in quantity for e:pk in $demand:ra
 px ! Percent change in (price of) a commodity
  ! price index for commodity X (%-change)
  ;
Rest endogenous ;

! Shocks
! ------
! Next is a shock to the market price in the omitted market.
! Hence this simulation is a nominal homogeneity simulation.
! [All prices and dollar values should increase by 1% and
!   all quantities prices should stay unchanged.]
```

```
Shock px = 1 ;

! Verbal Description
! ------------------
Verbal Description =
 Nominal homogeneity simulation, shocking px
  ;  ! End of Verbal Description
```

4.5.1 Comments in Command Files

Note that anything on a line after an exclamation mark **!** is a comment. Comments are there for you and for others who use your Command file. Comments are ignored by the program. So, for example, the whole of the line

```
! Closure
```

is a comment. Each comment ends at the end of the line (although there may be ! at the start of the next line to extend the comment over several lines).

4.5.2 The Closure

This specifies which variables are exogenous (that is, determined outside the model) and which are endogenous (that is, determined by the model).

You can see that the three variables **c_QS_RA_PL**, **c_QS_RA_PK** and **px** are exogenous and that the remaining variables are endogenous. The first two exogenous variables represent **C**hanges (**c_**) in the **Q**uantity **S**upplied by consumer **RA** of commodities **PL** (labour) and **PK** (capital) respectively. The third exogenous variable **px** represents the %-change in the market price of commodity **X**. Note the comments (text following an exclamation mark) in the Command file after the names of these variables – these comments should help you to decide what to shock when you prepare your own simulations.

GAMS users may be used to specifying the closure by fixing the level of exogenous variables by inserting **.FX** statements in their GAMS file.

4.5.3 The Shocks

In GEMPACK, "shock" is used to describe changes to an exogenous variable which move it away from its pre-simulation value. The values of the shocks are either %-changes or actual changes in the values of the exogenous variables. Any exogenous variables which are not shocked will remain at their pre-simulation value.

In this simulation, there is just one shock, namely a 1 percent increase in the variable **px**. That is, the market price of commodity X is assumed to increase by 1 percent (from its pre-simulation value of 1.0 to its post-simulation value of 1.01, as you saw when you looked at the **px** results in section 4.4 above. The other exogenous variables (endowments of labor and capital) are unchanged in this simulation.

4.5.4 The Rest of the Command File

Firstly there are the statements:

```
Auxiliary files = twobytwo ;
Extrapolation Accuracy File = yes ;
Log File = yes ;
Method = Euler ;  ! or Gragg
Steps = 4 6 8 ;  ! change as needed
```

- The first tells the program GEMSIM which model to solve – pretty important.
- The last two tell GEMPACK which solution method to use to solve the model. If you are new to GEMPACK, all you need to know at this stage is that these two statements will produce an

accurate solution of the nonlinear levels equations underlying the model in most case (including in the case of the shock in this simulation). We will tell you a little more about solution methods later in this document.

- The third statement ("log file = yes ;") tells GEMSIM to record all output going to the screen in a LOG file which will be called TWOBYTWOHSIM.LOG (the same name as that of the Command file, but with suffix .LOG).
- The second statement asks GEMSIM to produce a so-called Extrapolation Accuray file (which will be called TWOBYTWOHSIM.XAC). This file contains detailed information which expert users of GEMPACK can use to check the accuracy of the different results. If you are new to GEMPACK, you can ignore this for the time being.

Then there is the statement

```
Verbal Description =
 Nominal homogeneity simulation, shocking px
  ;  ! End of Verbal Description
```

[This is regarded as a single statement even though it extends over several lines. All Command file statements end with a semi-colon `";"`.]

This statement provides a so-called "verbal description" of the simulation. These words should be a summary of the simulation. They are stored with the simulation results and you can see them whenever you access these results. The idea is that, if you look at the results several months after you run the simulation, you should get a good idea of what the simulation is about from the verbal description. [If you open the Solution file in ViewSOL and click on **Description** in ViewSOL's main menu, you are shown the verbal description (plus some other information automatically added to it by the program).]

4.5.5 Why Have a Command File?

GAMS modellers are used to having the equations of their model and the instructions for solving all in the same file. In contrast, GEMPACK separates the different parts of a model.

- The equations are in the TAB file.
- The instructions for carrying out a simulation (the closure and shocks) are in a separate file which is called a Command file.[40]
- The pre-simulation data for the model is often in a separate file or files (as you will see in section 5 below).

4.6 Other Simulations with TWOBYTWO

The price homogeneity simulation above was not very interesting – indeed, it is really just a check that the model is not giving silly results.

You can specify other more interesting simulations by specifying different shocks and, perhaps, different closures. Here we will show you how to specify different shocks. We will introduce different closures in section 5 below.

4.6.1 Increasing the Labour Endowment – TWOBYTWOLB.CMF

As you saw in section 4.5.1 above, there are two other exogenous variables, namely the endowments (or supplies) of the two factors labour and capital. In this section we show you how to simulate an increase of 10 percent in the endowment of labour.

Preparing the Command File `TWOBYTWOLB.CMF`

[40] You can find more details about using GEMPACK Command files to specify a simulation in section 2.8 of GEMPACK document GPD-1.

In order to carry out this simulation, you need to prepare a suitable Command file. To do that, follow the steps below, in which you will start from the Command file `TWOBYTWOHSIM.CMF` and make changes to the shock and verbal description statements.

- Open the previous Command file **`TWOBYTWOHSIM.CMF`** in the program **TABmate**.
- Select menu item ***File* | *Save As…*** and save the file with the new name **`TWOBYTWOLB.CMF`**. [We suggest that names for Command files should reflect both the model and the sort of shock. Hence "LB" since this shock is about labour.]
- Change the name of the variable being shocked from **px** to **c_QS_RA_PL**. [It does not matter if you use upper or lower case, or a mixture. The program does not care about this, though you may prefer to use a mixture of cases to make your file more readable to humans.]
- How large should the shock be? As indicated above, we want you to simulate a 10 percent increase in the supply of labour. The variable c_QS_RA_PL represents the **change** (not the %-change) in the amount of labour. So, in order to get the right number in the shock statement, you need to know how much labour is initially available. To find that out you need to look at the q: field in the e:pl line the $demand:ra block in TWOBYTWO.MGE. As you can see the q: field in the e:pl line is q:70. Hence consumer RA is supplying 70 units of labour pre-simulation. A 10 precent increase amounts to 7 units. Hence the shock statement should be
  ```
  shock c_QS_RA_PL = 7 ;  ! a 10 percent increase
  ```
  [Don't forget the semi-colon after the "7". The words after the "!" are a comment.][41]
- You should change the verbal description. Perhaps change this so that it reads
  ```
  Verbal description = 10 percent increase in supply of labour ;
  ```
  (which you can put all on one line).
- This completes the changes, so click on the **Save** button near the top of the TABmate window. [Alternatively select menu item ***File* | *Save*** .]
- Then exit from TABmate using menu item ***File* | *Exit*** .

Running GEMSIM to Solve

Now that you have prepared a suitable Command file, you can run GEMSIM to carry out the simulation.[42]

- If you are working via WinGEM, make sure that your working directory is still `c:\mge2gp.` [Use ***File* | *Change both default directories…*** to check this.] Then open a GEMSIM window via menu item ***Simulation* | *GEMSIM solve…***. Then click on the **Select** button and select the Command file `TWOBYTWOLB.CMF` that you have just prepared. Then click on the **Run** button to run the simulation. When this has run (there should be no error), click **OK** since the accuracy results should be fine[43], and then click on the **Go to ViewSOL** button to look at the results.

[41] Provided that you are working with Release 8.0 (October 2002) or later of GEMPACK, an alternative way of specifying this shock in the Command file is to use the statement
```
percent_change QS_RA_PL = 10 ;
```
[See section 5.7 of GPD-3.]

[42] If you get errors when you run the simulation, you can compare your Command file TWOBYTWOLB.CMF with the one called TWOBYTWOLB-OK.CMF which we have supplied.

[43] You will see that the face for the variable results is smiling a bit (score of 8 out of a possible 10). You could increase that if you take more steps (for example, change "steps = 4 6 8 ;" to "steps = 8 10 12 ;" in the Command file.
You will also see that one result is only accurate to 0 figures. If you want to see which one, you can look in the Extrapolation Accuracy file TWOBYTWOLB.XAC. This is a text file so you can open it using TABmate. You will see that it is the variable **walrasslack** which is accurate only to 0 figures. [The accuracy summary shown in WinGEM uses the levels accuracy which is indicated by the figure after the "L" near the end of each line.] Most results are accurate to 6 figures "L6" or 5 "L5". However walrasslack (which should be zero) is very small but oscillating a little. That is why, somewhat misleadingly, it is shown to be accurate only to 0 figures. All in all this is a little complicated. If you are new to GEMPACK, we suggest that you ignore this for the time being.

- If you are working at the command line, go to a DOS-like box, (for example, via TABmate's menu item ***File | DOS in current folder***) and then change directory into `c:\mge2gp`. Then you can run the simulation via the command
  ```
  gemsim  -cmf twobytwolb.cmf
  ```
  When this has run (there should be no error), run **ViewSOL** and load in the simulation results which will be in file **`twobytwolb.sl4`**.

Looking at the Results

As before (see section 4.4), click on the **macro** line in the Contents page. You will see the results for all variables.

- First find the variable shocked (what colour are the numbers for exogenous variables in ViewSOL?). Is the shock correct?
- Next check a few of the results. For example, check that your results are approximately:
  `x` 4.088   `y` 3.886   `py` 0.9577 `pl` -4.6538   `pk` 4.8809
  (those above are all percentage changes)
- What would you expect to happen to the market price **pl** of labour? Why? Has this happened?
- What would you expect to happen to the market the price **pk** of capital? Has this happened? [Capital has become relatively scarce which is why its price has increased, while labour has become relatively abundant, which is why its price has fallen.]
- Why is the result for **px** zero? [Think about the closure.]
- What has happened to the market prices **px** and **py** of the commodities? Why? [Think about the prices of the input factors. Which sector is more labour intensive – see the data in the MGE file?]
- What has happened to the outputs of the two commodities? [The variables are **p_qd_c_px** (%-change in output of X) and **p_qd_c_py** (%-change in output of Y). For example, if you click on **p_qd_c_px** in the left-hand column, you will see in the bottom line of the ViewSOL screen a description of this variable "Quantity (level) in q: field of i:px in $prod:c". This description is taken from the line in the TAB file TWOBYTWO.TAB on which this variable is declared, as you can see if you open the TAB file.]

We do not attempt any further explanation of the results here (where we are concentrating on the mechanics of carrying out the simulations).

We suggest that you close ViewSOL in the usual windows way.

4.6.2 Other Simulations with TWOBYTWO

You can carry out other simulations by changing the shocks. For example,

- you might like to simulate the effects of an increase in the supply of capital, holding the supply of labour fixed and holding the numeraire `px` fixed. If so, the only shock statement will be
  ```
  shock c_QS_RA_PK = <number> ;
  ```
  [You might like to experiment with different numbers.]
- you might like to simulate the effects of increases in both the supply of labour and of capital (holding the numeraire `px` fixed). If so, the shock statements will be
  ```
  shock c_QS_RA_PL = <number> ;
  shock c_QS_RA_PK = <number> ;
  ```
  [You might like to experiment with different numbers.]

In each case, prepare a suitable Command file by editing an existing one. Then run GEMSIM to carry out the simulation and ViewSOL to look at the results.

4.7 Looking at the TAB File TWOBYTWO.TAB

Open the TAB file `TWOBYTWO.TAB` in TABmate. [For example, use ***File | Edit file*** from WinGEM's menu.] Look at the various parts of the file.

If you are a GAMS programmer,

- some of the equations in the TAB file will look familiar to you.
  For example, look at the equation called **E_ra**. This says that the total value of the income accruing to consumer `ra` is equal to the total value of the endowments of labor and capital. And there are easily understood equations for the values VS_RA_PL and VS_RA_PK just above.
- some of the equations are simple linearized versions of the obvious levels equations.
  For example, look at the equation called **E_p_vs_x_px**. As the comment above it in the code indicates, this relates to the `o:px` line in the `$prod:x` block. This equation is calculating the percentage change `p_vs_x_px` in the value supplied in `$prod:x` of commodity `px`. This percentage change is the sum of the percentage change `px` in the price of the commodity and of the percentage change `p_qs_x_px` in the corresponding quantity. [Of course the levels equation says that the value is equal to the price times the quantity.]
- you may be surprised by the behavioural equations (demand and supply functions). For example, look at the equation called **E_p_qd_x_pl**. This is the labor demand equation in sector `$prod:x` which comes from a CES function. GAMS programmers will be accustomed to working with nonlinear CES demand functions. Equation E_p_qd_x_pl is the well-known linearized version of the relevant CES function. The equation says:
  ```
  p_qd_x_pl = x – 1 * [pl – mc_x] ;
  ```
  Here `x` represents the percentage change in the activity level of sector `x`, `pl` represents the percentage change in the price of labor and `mc_x` represents the percentage change in the marginal cost of production in sector `x`. [Of course, `mc_x` is a weighted sum of percentage changes `pl` and `pk` in the prices of labor and capital – the two inputs to sector `x` – as you can see by looking at the equation called `E_mc_x`.[44]] The linearized equation `E_p_qd_x_pl` has two simple consequences:
  (i) If the activity level `x` in this sector increases by 1 percent, so does the demand for labor `p_qd_x_pl` in this sector.
  (ii) If the price of labor increases so that it becomes 1 percent higher than `mc_x`, then the demand for labor in this sector will fall by 1 percent. [The elasticity 1 is shown explicitly in the linearized equation above.] GEMPACK modellers find the linearized versions of these CES behavioural equations give more insight into the economics than the corresponding levels CES functions.
- you may be surprised by the large number or variables represented explicitly in the TAB file. Results (changes or percentage-changes) for all of these are automatically given in the Solution file which is produced when you carry out a simulation. For example, there are results for prices and quantities for all nests. [GAMS/MPSGE experts use the `$report:` section in an MPSGE file to generate reports of desired results. Any `$report:` section in your MGE file is ignored by MGE2GP (see section 7.8.1) since the GEMPACK results normally include all the results you could want.]

If you are a GEMPACK expert,

- you may be surprised to see how many levels equations are explicitly in the TAB file. You may learn something from this. For example, look at the market clearing equation called **E_pl** for commodity `pl`. Basically it says that
  ```
   TQD_PL = TQS_PL
  ```
  which obviously says (in the levels) that demand equals supply. Above this equation you can see the equally simple levels equations which calculate the values of demand and supply. [See section 3.2.4 for more details about this.]
- have a look at the large number of Assertions in the TAB file.
  1. Look at the two Assertions in the code for the `$prod:x` block in TWOBYTWO.TAB. The first one is an Assertion(Initial) which checks that costs and revenue are equal in the initial database for this sector x.[45] The second one is an Assertion(Always) which checks that costs and

[44] That equation looks a bit complicated because of the IF tests. Basically that equation says that
```
        mc_x = VD_X_PL*pl + VD_X_PK*pk.
```

[45] See section 7.1 for an explanation of the presence of BALTOI in this Assertion

revenue are equal in this sector after all steps in a multi-step calculation.
2. Look at the two Assertions about supply and demand of commodity pk near the end of the file. [Search for "equal for pk".] These check that supply equals demand for pk in the initial database and after all steps of a multi-step calculation. There are similar assertions for all other commodities.
These Assertions provide useful checks that any simulation is proceeding as expected. In order to satisfy them at all steps of a multi-step calculation, a fairly high degree of accuracy is required. You may need to increase the number of subintervals in order to satisfy these Assertions.

5. Solving the Other Example Models Using GEMPACK

In section 4 above, you carried out simulations with the first of the example MPSGE models, TWOBYTWO. In this section you will carry out simulations with the other MPSGE models we have supplied.

We give you hands-on instructions (less detailed than in section 4 above) for working with the SJMGE example in sections 5.1 and 5.2. You will carry out simulations with other example models in section 5.3.

5.1 The SJMGEHSIM Simulation with the SJMGE Model

In this section you will carry out simulations with the SJMGE version of the Stylized Johansen model. The MGE file for this model was introduced in section 2.2.1 above.

- If you are working at the Command line, change to the directory containing the MGE file **`sjmge.mge`** for the Stylized Johansen model.
- If you are working via WinGEM, set both default directories to the directory containing the MGE file **`sjmge.mge`** for the Stylized Johansen model.

First convert the MGE file `sjmge.mge` to GEMPACK TAB and Command files by running MGE2GP. As indicated in section 4.2.1 (command line) or 4.3.2 (WinGEM) above, you can do this by going to a DOS box in the directory in which you have the file `sjmge.mge` and entering the command[46]

mge2gp sjmge

This should create the three files `sjmge.tab`, `tsjmge.sti` and `sjmgehsim.cmf.`

Carry out the simulation following the steps in the either section 4.2 (command line) or 4.3 (using WinGEM) above, replacing TWOBYTWO by SJMGE.

Look at the simulation results from the SJMGEHSIM.CMF as in section 4.4 above. Note that prices increase by 1% and that quantities are unchanged.

5.1.1 Looking at the Starting Header Array File SJMGE.HAR

You can see the benchmark (or pre-simulation) data for this model by looking at the SAM shown in section 2.2.1. These data are grouped into matrices (for example, `qfacin` denotes the 2x2 matrix of factor inputs into the two sectors) as indicated there.

For GEMPACK, these data are held on what is called a **Header Array** file. Each matrix of data has an associated 4-character **header** which is used to refer to the array.

To look at the Header Array file **sjmge.har** containing the starting data for SJMGE, run the program ViewHAR and open this file SJMGE.HAR, following the instructions below.

- If you are working at the command line, you can type in the command
  viewhar sjmge.har
  and the program ViewHAR will start running and load in the file SJMGE.HAR.
- If you are working via WinGEM, select menu item ***HA files | View ViewHAR*** from WinGEM's main menu. [That is, click on **HA files** and then select menu item **View ViewHAR**.] This will start ViewHAR running. Then, go to ViewHAR's **File** menu and select menu item **Open Header Array file**. Select the file SJMGE.HAR. Then ViewHAR will load this file.

In either case ViewHAR will show the contents of the file SJMGE.HAR on the Contents screen.

[46] This assume that you have installed the program MGE2GP in a directory which is on your Path (as described in section 4.1).

Each of the rows corresponds to a different array of data on the file. Look at the column under the heading Name to see what data are in these arrays.[47]

```
   Header Type Size       Name

1  COMN   RE   SECTxSECT  Intermediate inputs of commodities to ind..
2  FACN   RE   FACxSECT   Intermediate inputs of primary factors - values
3  HOUS   RE   SECT       Household use of commodities - values
4  FAC    1C   2 length 12  Set FAC Factors
5  SECT   1C   2 length 12  Set SECT Sectors
6  GOOD   1C   2 length 12  Set GOOD Commodities
7  ENDW   RE   FAC        Endowments - values
```

The first array is the "Intermediate inputs of commodities to industries - values".
The Header **COMN** is just a label for this array. (Headers can have up to 4 characters.)
The array is of Type **RE.** The **R** means this is an array of real numbers. The **E** means that this array has set and element labelling (see chapter 5 of GPD-4).

Double click on **COMN** to see the numbers in this array.

```
COMIN         s1            s2            Total
s1            4.000000      2.000000      6.000000
s2            2.000000      6.000000      8.000000
Total         6.000000      8.000000      14.000000
```

Compare these numbers with the `comin` matrix in the SAM in section 2.2.1. The actual data in the file at this header is just the 2x2 matrix. ViewHAR calculates and shows the row and column totals.

To return to the Contents Screen, click on ***Contents*** in the ViewHAR menu.

Look at the other Header Arrays called **FACN** and **HOUS** to see where their numbers fit in the SAM.

Close ViewHAR in the normal Windows way by selecting ***File | Exit*** .

5.1.2 The Updated Data SJMGEHSIM.UPD

When you carry out a simulation via GEMPACK, the updated data is produced automatically. Ideally, the updated data shows how the economy will be after the shocks have worked their way through the system.

The data in the starting data file `SJMGE.HAR` could be thought of as representing quantities or as representing values. When you work with the GEMPACK version of the model produced by the program MGE2GP, you should think of the

starting data as representing values.

So the updated data should show you the new values (as they would be after the shocks have been applied).

The updated data after the SJMGEHSIM.CMF simulation are in Header Array file **SJMGEHSIM.UPD**. To check the data in it, run **ViewHAR** and open file `SJMGEHSIM.UPD`. In the simulation, all prices increased by 1% while all quantities remained unchanged. Hence all values should have increased by 1%. Check this by comparing the data in `SJMGEHSIM.UPD` with the corresponding data in the pre-simulation data file `SJMGE.HAR`. For example, the pre-simulation `HOUS` values are 2 (`s1`) and 4

[47] You may see extra columns. Exactly what you see is controlled by the section **For real matrices, Contents shows:** shown when you select menu item ***Options*** under the ***File*** menu.

(s2) while the post-simulation values at header HOUS in file SJMGEHSIM.UPD are 2.02 (s1) and 4.04 (s2), as expected.[48]

5.2 Increasing Endowment of Labor in SJMGE – SJMGELB.CMF

Look at the Command file SJMGELB.CMF. This Command file carries out a simulation with SJMGE in which the supply or labor is increased by 10%.

Open SJMGELB.CMF in TABmate. Look at the shock statement. This is

Percent_change QS_Y_PF("labor") = 10 ;[49]

In order to carry out this simulation with SJMGE, you only need to do the GEMSIM step following the method in section 4.2 or 4.3 above.[50]

Carry out this simulation by running GEMSIM, taking inputs from Command file SJMGELB.CMF.

Look at the simulation results by loading the Solution file SJMGELB.SL4 into ViewSOL. You may find it interesting to look at the following results. To follow the suggestions below, it will be helpful if you use the format "**Arrange vectors by name**". To do that, click on the ***Format…*** menu item on ViewSOL's main menu. Then click on **Arrange vectors by name** under "Vector Options". Then click on the **OK** button.

- First check that the shock has been applied correctly – this is always a good idea. To do that, look at the results for variable **c_QS_Y_PF**. [To do that, go to the **Contents** page of ViewSOL. You will see a line corresponding to the variable c_QS_Y_PF. In the **Name** column you see **Quantity (levels) for e:pf(fac) in $demand:y** which suggests that this variable relates to the supply of the factors. The "c_" are the start of the name indicates that this variable reports the changes in these endowments. To see the simulation results for this variable, click on c_QS_Y_PF in the Contents list.] The simulation results are in the first column (headed **sjmgelb).** Note that the simulation results show c_QS_Y_PF("labor")=0.4 and c_QS_Y_PF("capital")=0. These are changes (not percentage changes), as the "c_" prefix suggests. To check that the endowment of labor has increased by 10%, you need to check the pre-simulation level. To do that, look in the second column (the one headed **Pre sjmgelb**). You can see that the pre-simulation value for QS_Y_PF("labor") is 4 and (look at the next column headed **Post sjmgelb**) that the post-simulation value for QS_Y_PF("labor") is 4.4. So, as expected, the shock is an increase of 10% in the endowment of labor. And there is no change in the endowment of capital. To get back to the Contents page, click on the **Contents** menu item.

[48] On reflection you should not be surprised. Remember that the numbers in SJMGE.HAR represent quantities, and that quantities do not change in the homogeneity simulation. [GEMPACK experts used to the standard implementation of the Stylized Johansen model supplied with GEMPACK may be surprised. They need to remember that the data there in SJ.DAT represent dollar values, not quantities. So the data there increase by 1% in a homogeneity simulation.]

[49] QS_Y_PF("labor") is the name used in the TAB file for the amount (quantity) of labor available. The symbol QS_Y_PF(f) is a levels variable. Associated with this levels variable is the linear variable c_QS_Y_PF(f) which reports the changes in QS_Y_PF(f). Also in the TAB file is the linear variable p_QS_Y_PF(f) which represents the percentage-changes in QS_Y_PF(f). The pre-simulation value of QS_Y_PF("labor") is 4 – see the ENDOW("labor") value in the Header Array file SJMGE.HAR. Thus alternative shock statements

```
shock  c_QS_Y_PF("labor") = 0.4 ;
shock  p_QS_Y_PF("labor") = 10 ;
```

would produce the same results.

[50] You do not need to run TABLO again since the TAB file SJMGE has not changed. So the output from TABLO will be unchanged from when you ran it when doing the homogeneity simulation in section 5.1. But you do need to run GEMSIM since the shocks are different from those in that homogeneity simulation.

- Look at the changes in the prices of the factors – that is, look at the results for variable **`pf`**. Notice that the price of labor falls (by about 3.1%) while the price of capital rises (by about 6.6%). Is this what you expect?[51]
- Look at the prices of the commodities – that is, look at the results for variable **`pc`**. Can you understand why the price of the first commodity has fallen and why the price of the second commodity has fallen?[52]
- In order to understand which of the commodities became less expensive, you need to know which is more labor intensive. To do that, you need to look at the input data which is in Header Array file `SJMGE.HAR`. To look at that data, run the program **ViewHAR** and open the file `SJMGE.HAR`. The data about factor inputs to production is at header **FACN**. The Coefficient is called `FACIN`. Click on this header in the Contents page of ViewHAR. You will see the `FACIN` data. For example, `FACIN("labor","s1")`=1 and `FACIN("labor","s2")`=3. Which sector (`s1` or `s2`) is more labor intensive? [To see that, you need to look at shares. So click on the drop-down box in the top left-hand corner of the ViewHAR screen – the one which says **None**. Select **Col** to see the column shares. This tells you that 50% of the total factor input into `s1` is labor while 75% of the factor input into `s2` is labor.[53] Hence sector s2 is more labor intensive (and sector s1 is more capital intensive). Since the price of labor has fallen and the price of capital has risen (see earlier), you should not be surprised to see that the **`pc("s1")`** has fallen and that **`pc("cs2")`** has risen.
- Be careful about interpreting simulation results for prices. Above we said that certain prices have risen and that others have fallen. These sound like absolute statements. In fact, when we talk about simulation results for prices, we should always say **relative to the numeraire**. In this simulation, the numeraire is **`pw`** (the consumer price index). [To see this, look in the Command file SJMGELB.CMF. You can see that `pw` is exogenous and not shocked.] So above, when we said that some price has risen, we should have added "relative to pw".
- In the MGE file SJMGE.MGE you see the line
```
  e:pf(fac)          q:endow(fac)
```
  In the MGE file, `endow(fac)` represents the *quantity* of the factors. But in the TAB file, `ENDOW(fac)` is used to denote the *value* of the factors. So the **`p_ENDOW`** results represent percentage changes in the values of the factors. Look at the `p_ENDOW` results by clicking on the `p_ENDOW` in ViewSOL's Contents page. For labor, you will see percentage increase of 6.56% from a pre-simulation value of 4 to a post-simulation value of 4.26. You know that the quantity of labor increased by 10% (this is the shock). The price `pf("labor")` fell by 3.13%. That explains why the value `ENDOW("labor")` increased by less that 10%.[54]

5.2.1 The Updated Data after the SJMGELB.CMF Simulation

When you carry out a simulation via GEMPACK, the updated data is produced automatically. Ideally, the updated data shows how the economy will be after the shocks have worked their way through the system.

The data in the starting data file `SJMGE.HAR` could be thought of as representing quantities or as representing values. When you work with the GEMPACK version of the model produced by the program MGE2GP, you should think of the

[51] Which factor has become relatively scarce?

[52] What happened to the prices of the factors? Which commodity requires a larger share of labor input?

[53] The value shares are the same as the quantity shares since we are assuming that prices of the factors are originally 1.

[54] From the percentage changes in the quantity and price, the new value for ENDOW("labor") should equal
4 * 1.10 * 0.9687 = 4.26
as shown in the "post sjmgelb" column. [Quantity increase of 10% multiplies the value by 1.10. Price decrease of 3.17% multiplies the value by 0.9687.]

starting data as representing values.

So the updated data should show you the new values (as they would be after the shocks have been applied).

The updated data after the SJMGEHSIM.CMF simulation are in Header Array file **`SJMGELB.UPD`**. To check the data in this file, run **ViewHAR** and open file `SJMGELB.UPD`.

First look at the updated **`EDNOW`** data (which is at header **ENDW**). Note that the updated values are 4.26 (labor) and 2.13 (capital). These are consistent with the percentage changes in the quantities and prices reported in the simulation. [For details, look at the explanation of the `p_ENDOW("labor")` results – see a little earlier. A similar calculation works for capital, whose quantity has not changed and whose price has increased by 6.56%.]

Now look at the updated **`HOUS`** data (which is at header **HOUS**). [To do that, first click on **Contents** to get back from the `ENDOW` data to the Contents page, then click on header HOUS in the Contents.] You can see the post-simulation values of commodities `s1` and `s2` demanded by households. Again these values should be consistent with the simulation results for the associated prices and quantities. [For example, the price of commodity `s1` has increased by 0.63% while the quantity of commodity `s1` used by households – this is the `p_QD_W_PCGOOD("s1")` simulation result – has increased by 5.88%. Hence the value has increased by about 6.56% (from 2 to 2.13).

If you wish to compare with the pre-simulation values, you need to also open the original data in file SJMGE.HAR. Do that by running a second copy of **ViewHAR** and opening file SJMGE.HAR. You can see that the pre-simulation values of `HOUS` are 2 (s1) and 4(s2). The post-simulation values are about 2.13 (s1) and 4.26 (s2).[55]

5.2.2 Reversing the SJMGELB.CMF Simulation – SJMGELBBACK.CMF

One excellent way of testing a model and its updated data is to run a simulation which reverses the shocks.

For example, in `SJMGELB.CMF` you increased the amount of labor by 10%. The reversal simulation is to start from the data updated after the original simulation and to reduce the amount of labor by the corresponding amount. The simulation results from this reversal simulation should be opposite of those in the original simulation and the updated data after this reversal simulation should be the same as those from which the original simulation started.

In this section you will carry out the reversal `SJMGELBBACK.CMF` of the `SJMGELB.CMF` simulation, and will check the assertions in the paragraph above about the simulation results and the updated data.

The shock statement in `SJMGELB.CMF` s

```
Percent_Change QS_Y_PF("labor") = 10 ;
```

So, you might expect that the shock for the reversal simulation is to make a percentage change of –10%. But in fact the opposite of a 10% increase is a –9.090909% decrease.

To see this, consider some quantity `X` which is originally equal to `X0`. If we increase it by 10%, the new value `X1` will be given by `X1 = 1.1*X0`. Now, if we want to move from `X1` back to `X0`, the change required is `X0-X1=-0.1*X0`. Hence the percentage change required, starting from `X1`, is equal to

[55] The is a neat way to have both ViewHARs visible on the screen at the same time. Go to the ViewHAR which has the updated data SJMGELB.UPD loaded. Look in the bottom right-hand corner of the screen. You will see a green F or L, then to its left 6 arrows. To their left a shape that looks a bit like an X. If you click (very carefully) on the left hand part of that X, this ViewHAR will occupy just the left-hand part of the screen. Then go to the ViewHAR which has the original data SJMGE.HAR loaded and click (carefully) on the right-hand part of the X. This ViewHAR will then occupy the right-hand half of the screen. That makes it easy to compare pre- and post-simulation items in the data. [For example, make the QHOUS data visible in both ViewHARs.]

```
100*[-0.1*X0/X1] = 100*[-0.1*X0/(1.1*X0)] = 100*[-0.1/1.1] = -9.090909%.
```

Hence the shock statement in `SJMGELBBACK.CMF` is

```
Percent_Change QS_Y_PF("labor") = -9.090909 ;
```

When carrying out the reversal simulation, start from the data updated after the original simulation. Hence, in `SJMGELBBACK.CMF` you see the statement

```
File Input = sjmgelb.upd ;
```

since `SJMGELB.UPD` contains the data updated after the `SJMGELB.CMF` simulation.

Now look at the `SJMGELBBACK.CMF` Command file (by opening it in TABmate).

Then carry out the reversal simulation by running GEMSIM taking inputs from the Command file `SJMGELBBACK.CMF`.

5.2.3 Checking the SJMGELBBACK.CMF Results

This is best done by loading the results from the two simulations into ViewSOL. So, first open `SJMGELB.SL4` (the results from `SJMGELB.CMF`) into ViewSOL. Then, without closing ViewSOL, load `SJMGELBBACK.SL4` (the results from the reversal simulation) via ViewSOL's ***File | Open*** menu item.

Go to the ViewSOL **Format** menu and select **Arrange vectors by name**.

First look at the **`c_ENDOW`** results by clicking on this row in the Contents screen. You should see the following. The first 4 columns are the simulation, PreSim, PostSim and Change results from the original simulation. The last 4 columns are from the reversal simulation.

c_ENDOW	LB	Pre LB	Post LB	Ch LB	back	Pre back	Pst back	Ch back
Labor	0.2624	4.0000	4.2624	0.2624	–0.2624	4.2624	4.0000	–0.2624
Capital	0.1312	2.0000	2.1312	0.1312	–0.1312	2.1312	2.0000	–0.1312

Notice that the simulation results for the reversal are indeed the negative of the results from the original simulation. [These simulation results are the changes in the value of these endowments.]

Now try the **`pf`** results. These are percentage changes in the prices of the factors. You should see the following.

Pf	LB	Pre LB	Post LB	Ch LB	back	Pre back	Pst back	Ch back
Labor	–3.1271	1.0000	0.9687	–0.0313	3.2280	1.0000	1.0323	0.0323
Capital	6.5602	1.0000	1.0656	0.0656	–6.1563	1.0000	0.9384	–0.0616

At first glance, these don't look right. But they as expected are once you realize that prices are normalized back to one at the start of all simulations, including the reversal simulation.

First look at the percentage changes in the price of labor. Is 3.2280 the reverse of –3.1271? Well, yes, if you follow the general rule stated below.

The reversal of a percentage change of P is equal to –100*P/(100 + P).[56]

This says that the reversal of a decrease of –3.1271% is an increase of about 3.2280% (which you can check using the formula above and a calculator).

[56] To check this, suppose that the initial value is 100. Then after the first simulation, the value will be 100+P. Hence the reversal percent change is 100*Change/NewValue = 100*(-P)/(100+P).

Now try the **c_TQD_PC** results. These are changes in the quantity demanded of each of the two commodities. You should see the following.

c_TQD_PC	LB	Pre LB	Post LB	Ch LB	back	Pre back	Pst back	Ch back
s1	0.4708	8.0000	8.4708	0.4708	–0.4738	8.5248	8.0510	–0.4738
s2	0.8279	12.0000	12.8279	0.8279	–0.8253	12.8272	11.9619	–0.8253

Look at commodity `s1`. There is an increase of 0.4708 in the first simulation. The pre-simulation quantity is 8.0000 and the post-simulation quantity is 8.4708. Why is the pre-simulation quantity for the reversal simulation shown as 8.5248 (rather than 8.4708 – the post-sim quantity from the initial simulation)? This is because prices (including `PC`) are re-normalized to 1 at the start of each simulation. The dollar value for the output of commodity `PC(“s1”)` in the updated data `SJMGELB.UPD` is indeed 8.5248 (as you can see by adding across the s1 rows in the `COMIN` and `HOUS` matrices in the Header Array file `SJMGELB.UPD`). This is why the Pre-simulation number for the `SJMGELBBACK` simulation is shown as 8.5248. It explains why the changes (0.4708 and –0.4738) are not the negatives of each other. However the fractional changes 0.4708/8 and –0.4738/8.5248 are reversals of each other (as you can check with a calculator). Indeed, you can look at the `p_TQD_PC(“s1”)` results in ViewSOL where you will see the relevant percentage changes.

In summary, the results of the `SJMGELBBACK.CMF` simulation are the reversals of those for the `SJMGELB.CMF` simulation. But you have to work a bit hard to confirm this.

5.2.4 Checking the Updated Data after the SJMGELBBACK.CMF Simulation

The GEMPACK program **CMPHAR** can be used to compare the data on two Header Array files. You can run it via WinGEM using menu item ***HA files | CMPHAR Compare*** . Select SJMGE.HAR as Header Array file 1 and select SJMGELBBACK.UPD as Header Array file 2. Then click on the **Run** button. When this finishes, click on the **View print file** button.

Go to the end of the file. You will see a summary which should include something like the following.

```
 Total number of NON-CHARACTER data headers compared is 4.
 Total number of differences found is 12.
 Total of all absolute differences is 3.92198563E-05.
 Average of all absolute differences is 3.26832128E-06.
 Header where largest difference was found is 'COMN'.
 Largest absolute difference found is 8.10623169E-06.
 Total number of difference ratios reported is 12.
 Total of all difference ratios is 1.67688031E-05.
 Average of all difference ratios is 1.39740030E-06.
 Header where largest difference ratios was found is 'COMN'.
 Largest difference ratio found is 2.02656202E-06.
 Total number of all sign changes is 0.
 Total number of all differences positive to zero is 0.
 Total number of all differences negative to zero is 0.
```

The **largest absolute difference** is very small (about 0.000003). The **largest difference ratio** is also very small (about 0.000002).[57] These mean that the data in the two files are essentially identical, as we hoped and expected.

This is a further confirmation that the second simulation is the reversal of the first one.

[57] Scientific notation is used in the print file. For example, the largest difference ratio is 2.02656202E-06. Here the “E-06” means “multiplied to 1/10^6” – that is, multiplied by one over one million. The difference ratio between two values V1 and V2 is [V2-V1]/V1.

5.3 *Carrying Out Simulations with the Other Example Models*

Besides TWOBYTWO and SJMGE (which are covered above), the other examples models supplied are:

JOINT, TAXOUT, TAXIN, EMPLOY, OPEN, BOP, TARIFF and DIFFTAX.

The MGE files for these models are discussed in section 2 above.

In each case, when you run MGE2GP on these MGE files, you will get

- a Command file for carrying out a homogeneity simulation. The name has "HSIM" added after the name of the model. For example, this Command file is called TAXINHSIM.CMF for the TAXIN model.
- a Stored-input file which you should use in order to run TABLO. The name has "T" added before the name of the model. For example, this Stored-input file is called TTAXIN.STI for the TAXIN model.

We also supply a Command file which you can use to carry out a non-trivial simulation with the model. The name is the name of the model followed by suffix ".CMF". For example, this Command file is called TAXIN.CMF for the TAXIN model.

The steps for working with one of these models are always the same.

- Run the program MGE2GP to convert the MGE file to GEMPACK TAB, STI and Command files. [See section 4.2.1 (command line) or 4.3.2 (WinGEM) for details.]
- Run TABLO taking inputs from the Stored-input file. [For example, use TTAXIN.STI for model TAXIN.] This will produce output for GEMSIM. [See section 4.2.2 (command line) or 4.3.3 (WinGEM) for details.]
- Run GEMSIM using the HSIM Command file to carry out a homogeneity simulation. [For example, use Command file TAXINHSIN.CMF with the TAXIN model.] This will produce a Solution file containing the results. [See section 4.2.3 (command line) or 4.3.4 (WinGEM) for details.]
- Whenever you carry out a simulation, use ViewSOL to look at the results (as in section 4.4).
- You can then run GEMSIM to carry out the non-trivial simulation we have supplied. Use the Command file we have supplied. [For example, use the Command file TAXIN.CMF with the TAXIN model.] Look at the Command file to see what the shocks are.
- You can then carry out different simulations with the model. For example, change the shocks in the example Command file we have supplied.

We recommend that you go through these steps for several of the example models.

Below (in section 5.3.1), we discuss in detail a simulation with EMPLOY. We have chosen to do that example in detail in order to encourage you to work through a simple example in which rationing (see section 2.4) applies.

In sections 5.3.3 and 5.3.4 we give some details about the closures of these models.

5.3.1 A Simulation with EMPLOY

First run MGE2GP to convert EMPLOY.MGE to TAB, STI and Command files.

Then run GEMSIM to carry out the simulation in the Command file **EMPLOYTO.CMF** we have supplied. Below you will work through the results of this simulation.

First look at the Command file EMPLOYTO.CMF by opening it in TABmate.

What is the shock?

The shock statement is

```
Final_level  TOPYPDGOV("sind",o) = uniform 95 ;
```

Levels variable TOPYPDGOV is the power of the tax levied by `gov` on the outputs of commodity pd from sector `$prod:y(s)` [see the TAB file]. Hence the above Final_level statement means that the post-simulation values of the powers of these taxes on outputs of both commodities from sector "sind" should be 0.95. Since this is an output, the value of 0.95 means a tax (not a subsidy – see section 2.3.2) of 5% *ad valorem* rate.

What are the pre-simulation values of these tax rates?

You can see from the relevant line

```
  o:pd(o)     q:supply(s,o)     a:gov       t:bto(s,o)
```

in EMPLOY.MGE that the pre-simulation tax rates are given by the pre-simulation values of Coefficient BTO. Open the pre-simulation database file EMPLOY.HAR in ViewHAR and look at the BTO values. You can see that they are all zero. Hence there are no taxes on outputs of `pd` in the pre-simulation database and so in EMPLOYTO.CMF you are simulating the introduction of a 5% *ad valorem* tax.

In this model, there is a rationing variable EPL on the endowment of labor – see the `e:pf("lab")` line in the `$demand:ra` block. And there is a `$constraint:epl` block which gives the extra linearized equation

```
pf("lab") = pu ;
```

This equation says that the percentage changes in the price of labor `pf("lab")` and the percentage change in the price of (household) utility `pu` are equal. You might like to re-read section 2.4.1 to refresh your memory as to how this rationing works.

What happens to the amount of labor in this simulation?

Recall that the actual amount of labor is always equal to the product of the levels variables ENDL and EPL. [ENDL is the value in the `q:` field in the `e:pf("lab")` line and EPL is in the `r:` field in that line.]

The pre-simulation values are: 100 for ENDL and 1 for EPL – look at EMPLOY.HAR in ViewHAR to see these values. Hence the pre-simulation quantity of labor is 100*1=100.

To see the post-simulation values, open the Solution file EMPLOYTO.SL4 in ViewSOL. Go to the **macros** results and look at the simulation results for the two relevant variables, namely **c_ENDL** (change in ENDL) and **c_EPL** (change in EPL). You can also see the post-simulation levels values for these levels variables in the **Post employto** column. Notice that these post-simulation values are 100 for ENDL (this is exogenous and does not change in this simulation) and about 2.04 for EPL (which increases by about 104% from its pre-simulation value of 1). Hence the post-simulation quantity of labor is about 100*2.04=204. Thus the output tax results in a very large increase in the quantity of labor.

What happens to the prices of labor and capital?

To see this, look at the simulation results for variable pf. Note that these results are

pf("lab") = 0 and pf("cap") = –2.70.

Why does the price of labor not change?

It is tied to the variable `pu` by the `$constraint:epl` equation. And variable `pu` is the numeraire (which is not shocked). Hence you should interpret all price results as being relative to the numeraire `pu`.

5.3.2 Reversal Simulations

For each of these models, we also supply a Command file which reverses the standard simulation. For example, there is a Command file DIFFTAXBACK.CMF which reverses the shocks in DIFFTAX.CMF. In each case you can check that the simulation results are reversed (following the methods outlined in section 5.2.3) and that the updated data after the reversal simulation is the same as the initial data for the model (using CMPHAR, as explained in section 5.2.4).

5.3.3 The Standard Closure

The Command files for carrying out a homogeneity simulation have the "standard" closure for the model. In this standard closure,

- endowments are normally exogenous.
  For example, in TAXIN, the only endowments are the commodities `pf(f)`. You can see that variable c_ENDOW is exogenous in the file TAXINHSIM.CMF. In this model, c_ENDOW(f) is the change in the quantity of commodity `pf(f)` – see the `e:pf(f)` line in TAXIN.MGE.
  More precisely endowments which are exogenous unless they are rendered endogenous because they are rationed via an associated `r:` field (see section 2.4). And even then the quantity in the relevant q: field is on the exogenous list. For example, in EMPLOY, the variable c_ENDL (which is the change in the quantity in the q: field in the rationed e:pf("lab") line) is shown as exogenous. But remember that the actual quantity of labor used is not ENDL but rather the product ENDL*EPL since EPL is the rationing variable. So, although ENDL is exogenous, the actual quantity of labor used is not exogenous.
- The powers of the taxes are exogenous.
  For example, in TAXINHSIM.CMF you will see that variables TIYPDGOV and TID_VA_0001 are exogenous. The first of these is the power of the tax accruing to agent `gov` from the `i:pd(o)` line in the `$prod:y(s)` block. The second of these is the power of the tax accruing to agent `gov` from the `i:pf(f)` line in the `$prod:d_va_y(s)` block (which is written by the program MGE2GP when it splits the `va` nest in the original `$prod:y(s)` block (see section 3.1.2).
- the numeraire is exogenous.
- all other variables are endogenous.

5.3.4 Closure Changes

There are some simple ways in which you can change the closure.

- You can change the numeraire. As explained in section 7.9, the choice of numeraire is somewhat arbitrary. So you can experiment with different numeraires. For example, you could change the numeraire for TWOBYTWO from `px` to, say, `py` (or `pl`).
- We have chosen to include the powers of the taxes in the standard closure. Alternatively we could have chosen to include the tax rates in this closure. So, whenever you see the power of a tax in the closure, you could replace it with the corresponding rate. For example, in TAXIN you could replace the percentage-change `tiypdgov` in the power by the change `c_tirypdgov` in the associated rate.

5.4 Other Features of GEMPACK

Now that you have worked this far through the document, you are familiar with many of the important features of GEMPACK. In this section we make brief mention of a couple of other features you might like to be aware of.

5.4.1 AnalyseGE for Analysing Simulation Results

With modern software, it is easy to generate lots of numbers (model results). Understanding them is more difficult.

AnalyseGE is a windows program which aims to assist you in analysing simulation results. This gives you point and click access to the equations of the model, and to the data and consequences of it. You can decompose equations to see which parts are most important.

You can find a detailed hands-on introduction to AnalyseGE (in the context of a simulation with the Stylized Johansen model) in chapter 6 of GPD-8.

5.4.2 TABLO-Generated Programs for Simulations

Until now we have only introduced you to the program GEMSIM for carrying out simulations. GEMPACK users with a Source-code version of GEMPACK can produce so-called TABLO-generated EXEs for carrying out simulations.

TABLO is run slightly differently than when you intend to run GEMSIM. To produce a TABLO-generated program, you need to give response

wfp   ! Write a fortran program

instead of "pgs" at the Code stage of TABLO. Then TABLO produces a Fortran program especially tailored for fast solution with the model in question. For example, if you are processing SJMGE.TAB, TABLO will produce SJMGE.FOR.

Then you need to compile and link this program to produce the corresponding executable image (for example, to produce SJMGE.EXE from SJMGE.FOR). You can run this TABLO-generated executable image instead of GEMSIM to carry out simulations with the model. Compiling and linking require a suitable Fortran compiler and can only be done by those with a Source-code version of GEMPACK. You cannot produce TABLO-generated EXEs with an Executable-Image version (or the Demonstration version) of GEMPACK.

TABLO-generated programs have two main advantages over GEMSIM.

- They are much faster for large models. (See chapter 4 of GPD-8 for some figures.)
- You can distribute them to others who can use them to carry out simulations with the model. The persons you distribute them to can change the closure, shocks and starting data. They do not need a GEMPAC licence (except for large models when the inexpensive Introductory GEMPACK licence, costing only about $US200, is required).

5.4.3 Solution Methods

GEMPACK usually solves the model using Euler's or Gragg's method. These are methods for solving initial value problems. Usually three separate calculations with different numbers of steps (for example, 4, 6 and 8 steps) are done, followed by extrapolation. See chapter 7 of GPD-3 for more details.

All the Command files written by MGE2GP specify Euler's method. In most cases you could change this to Gragg's method by replacing the Command file statement

```
method = Euler ;
```

by

```
method = gragg ;
```

Usually Gragg's method is more accurate than Euler's method (for the same number of steps).

6. Larger Example Models

The example models discussed in sections 2 to 5 above are all pedagogical models which should be useful for learning and teaching about MPSGE and GEMPACK.

We also intend that MGE2GP will be able to handle "serious" models – that is, models which can be used for policy modelling.

The current version of MGE2GP can also handle at least one more serious model, namely the MINIMAL model, as we describe in section 6.1 below. We plan to develop MGE2GP so that it can handle other larger-scale example models such as ORANI-G and TERM.[58]

In order to handle such models, the TAB files written by MGE2GP must be able to handle zeros in the database, and they must be able to handle exceptions and domain restrictions (see sections 2.6.1 and 10.5). The TAB files currently written by MGE2GP are already fairly good in these respects.

- Some of the "IF" conditions you see in Formulas and Equations in the TAB files for the example models are there in order to handle zeros in the database.
- Some domain restrictions and exceptions can be handled by MGE2GP – see section 7.13.

6.1 The MINIMAL Model

This is a model developed by Mark Horridge.

MINIMAL is a slightly simplified CGE model designed by Mark Horridge. It contains a simplified treatment of taxes, and no margins or multiproduction. The usual database for MINIMAL has only 7 sectors.

This model is described at

http://www.monash.edu.au/policy/minimal.htm

The files for the MGE version of this model are in zip file **`MINLMGE.ZIP`** which is available on the MGE2GP web site (see section 1.1). For comparison with the standard MINIMAL model (as written by Mark Horridge), we supply on the MGE2GP web site the zip file **`MINIMAL-MGE.ZIP`**. You should download and unzip these files.

We supply (in `MINLMGE.ZIP`) the MGE file `MINLMGE.MGE` for this model and associated Header Array file `MINLMGE.HAR`. We also supply 3 Command files for carrying out simulations which

- change government consumption (`MINLMGE-GOV.CMF`),
- change household consumption (`MINLMGE-X3TOT.CMF`), and
- change the real wage (`MINLMGE-REALWAGE.CMF`).

These are three of the standard simulations with the MINIMAL model.

For comparison we include the standard TAB file `MINIMAL.TAB` (as written by Mark Horridge), associated Header Array file `MINIMAL.HAR`[59], and three Command files for carrying out the simulations with minimal which correspond to the simulations with the `MINLMGE.MGE` version of the model. These three Command files are `MINIMAL-GOV.CMF`, `MINIMAL-X3TOT.CMF` and `MINIMAL-REALWAGE.CMF`.

To work with the `MINLMGE.MGE` version, we suggest that you proceed as follows.

[58] These models are documented at http://www.monash.edu.au/policy/oranig.htm and http://www.monash.edu.au/policy/term.htm respectively.

[59] The Header Array files `MINIMAL.HAR` and `MINLMGE.HAR` contain exactly the same data. We have just changed the headers in `MINLMGE.HAR` to be what the program MGE2GP expects.

- Run the program MGE2GP to convert MINLMGE.MGE to `MINLMGE.TAB`, `TMINLMGE.STI` and `MINLMGEHSIM.CMF`.
- Before running TABLO, add the statements in the supplied file **`MINLMGE-BOT.TAB`** to the bottom of `MINLMGE.TAB` (as produced by MGE2GP). [Use TABmate and copying and pasting.] The statements in `MINLMGE-BOT.TAB` transfer results for variables in `MINLMGE.TAB` (as produced by MGE2GP) to results for variables with the names used in MINIMAL.TAB. This makes comparison of simulation results between `MINIMAL.TAB` and the MGE version easier.[60]
- Run TABLO, taking inputs from Stored-input file `TMINLMGE.STI`.
- Carry out the homogeneity simulation by running GEMSIM, taking inputs from the Command file `MINLMGEHSIM.CMF`. Check that prices increase by 1% and that quantities remain unchanged.
- Carry out the three example simulations by running GEMSIM taking inputs from the Command files `MINIMAL-GOV.CMF`, `MINIMAL-X3TOT.CMF` and `MINIMAL-REALWAGE.CMF`. Look at the simulation results.

To compare the results with those from the original MINIMAL model, you need to carry out the simulations with it. You can do that via the following steps.

- Run TABLO, selecting TAB file `MINIMAL.TAB` and producing output for GEMSIM. [There is no condensation necessary so you do not need to run from a Stored-input file.]
- Carry out the three simulations by running GEMSIM taking inputs from the Command files `MINIMAL-GOV.CMF`, `MINIMAL-X3TOT.CMF` and `MINIMAL-REALWAGE.CMF`.

Then, for example, to compare the results for the GOV simulation, load the results in `MINIMAL-GOV.SL4` and `MINLMGE-GOV.SL4` into ViewSOL (in that order) so that you have both sets of results in ViewSOL at the same time. In the ViewSOL **Choose which solution to view** drop-down box (the one which shows lets you switch between the different solutions loaded), select **`1 MINIMAL-GOV`**. Then the contents page will show you the variables in `MINIMAL.TAB`. Look at the results for the variables **`p1tot`**, **`x1tot`** and **`p1prim`**. These percentage change results are available in both models. You should see very similar numbers. There are also some common variables on the page of Macro results.

6.1.1 Noteworthy Features in MINLMGE.MGE

- The downward sloping export demands in MINIMAL (see Excerpt 8 in `MINIMAL.TAB`) are implemented via the `$constraint:e(o)` equation in `MINLMGE.MGE`.
- The `$constraint:` equations are written to include so-called "shifters". See section 7.6.1 for an explanation. In particular, these shifters make it easy to change closures in natural ways, as described in section 7.6.1. For example, these equations make it possible to shock real wages (letting employment adjust) or to shock real consumption (letting the balance of trade adjust) – both simulations in the ORANI tradition but simulations which are not usually done with MPSGE models.

6.1.2 Reversals of the Simulations

We supply Command files for reversing the three standard simulations with both the `MINLMGE` and MINIMAL versions of this model. These Command files all have `BACK` in their names. In each case

[60] For example, variable `pd(o)` in MINLMGE.TAB corresponds to variable `p1tot(i)` in MINIMAL.TAB. You will see the equation

```
Equation E_p1tot  (all,i,IND) p1tot(i) = pd(IndToS(i)) ;
```

in `MINLMGE-BOT.TAB`. This means that results from `MINLMGE` for variable `pd(o)` are also reported as results for variable `p1tot(i)`. Here `IndToS` is a mapping from the set `IND` in `MINIMAL.TAB` to the set `S` (which is equal to the set `O` in `MINLMGE.MGE`).

you can check that the BACK simulation reverses the simulation results and that the updated data after the reversal simulation is the same as the starting data. [Follow the methods in sections 5.2.3 and 5.2.4 above.]

6.1.3 Technical Change not in MINLMGE.MGE

The current version of `MINLMGE.MGE` does not allow for any technical change. In particular, the variable `a1prim(i)` (All primary-factor augmenting technical change) in `MINIMAL.TAB` is not available in the MINLMGE version of the MINIMAL model. We may rectify this in a later version of `MINLMGE.MGE`.

7. Building Your Own Models

In this section we provide advice about writing down your own models in MPSGE format, assuming that you intend to convert them to GEMPACK in order to carry out simulations. We also give advice about preparing the data in a Header Array file.

Unlike the earlier parts of this paper (which are designed to be read in order), this section contains reference material. You may need to jump between different subsections, depending on your needs.

7.1 *Must the Starting Data Represent an Equilibrium?*

When you use GEMPACK to solve a model,

you must start from data representing an equilibrium

(that is, a balanced data set) of the model. The simulation produces a second equilibrium, represented by the updated data. The solution algorithms usually used by GEMPACK require this.[61] When you solve a model, the simulation results reported are changes or percentage changes from the pre-simulation equilibrium, so you can think of the simulations as reporting perturbations from an initial equilibrium.

This is why, in the TAB files written by MGE2GP, there are Assertions which check that the initial data are balanced.[62]

Example. If you look in TWOBYTWO.TAB into the code for the $prod:x block, you will see the following assertion.

```
Assertion (Initial)
 ! If this fails, the initial data is not balanced. !
 # Initial data. Costs = revenue for $prod:x #
   [R_X GE (1-BALTOLI)*C_X] AND  [R_X LE (1+BALTOLI)*C_X] ;
```

This checks that revenue `R_X` in that sector is equal to costs `C_X`. [Because there can be rounding when you add numbers on a computer, the complicated looking expressions here are simply a way of saying that `R_X=C_X` up to some reasonable accuracy. In the code `BALTOLI`=0.0001 so the assertion says that `0.9999*C_X  <=  R_X  <=  1.0001*C_X`.]

However, when using GAMS/MPSGE to solve models, the software does not demand an initial equilibrium since a model may be posed based on technology and preferences, and these need not be determined from a calibrated benchmark. Nonetheless, in most applications, data sets are balanced because you usually want to start from an equilibrium.

7.2 *Preparing Data in a Header Array File*

You might use the various GEMPACK tools to prepare the data in this form initially. Alternatively, if you have prepared the data in a GAMS form, you can use tools such as GAMS2HAR or GDX2HAR [see http://www.monash.edu.au/policy/gp-gams.htm] to convert the data into a GEMPACK Header Array file.

7.2.1 Data on the File

For every relevant name in the MGE file, the GEMPACK TAB file produced by MGE2GP expects to find the pre-simulation (that is, benchmark) values at some header on the associated Header Array file. For example, if you have

[61] See section 7.8 of GPD-3 for details.

[62] The are also Assertions which check that the data at the start of each step of a multi-step calculation are also balanced – these are checks that the simulations are proceeding as expected.

```
  i:pd(o)     q:interm(o,s)       a:gov      t:bti(o,s)
```

in a `$prod:` block, the TAB file expects to find data for the pre-simulation values of `interm` and `bti` at different headers on the Header Array file. See section 7.2.3 below to find out what headers are expected.

7.2.2 Sets on the File

For every set referred to in the MGE file, the GEMPACK TAB file produced by MGE2GP expects to find the elements of this set at a header on the associated Header Array file. For example, if you have

```
  i:pc(goods)        q:qcomin(goods,sectors)
```

in a `$prod:` block, the TAB file expects to find the elements of sets `goods` and `sectors` at different headers in the associated Header Array file. See section 7.2.3 below to find out what headers are expected.

7.2.3 Names of Headers

The program MGE2GP uses the following algorithm to work you the Header name where it expects to find the values of a Coefficient or the elements of a Set.

- Omit all underscore "_" and "@" characters (if any).
- If the resulting name has 4 or fewer characters and that header has not already been used, use that. [For example, expect values of `qcom` at header "QCOM".]
- Omit vowels (a,e,i,o,u) from the right-hand end of the resulting name until the resulting name has 4 or fewer characters (if so, use that if that header has not already been used) or until there are no vowels remaining. [For example, expect elements of set FACTOR at header "FCTR" and expect values of Coefficient FOODS at header "FODS".]
- Truncate the resulting name to 4 characters and use that header if it has not already been used. [For example, expect values of Coefficient QGOODSIN at header "QGDS" if that header has not already been used.]
- Otherwise, take the first 3 characters of the resulting name and try adding digits (0-9) at the end until find a header name that has not been used. [For example, expect names of elements of set FACTORS at header "FCT1".]

[We plan that a later version of MGE2GP will look at the Header Array file you have prepared and be able to find the header at which a Coefficient values or the elements of a Set can be found. This will be simpler for users who can then ignore the algorithm described above.]

7.2.4 Introduction to Header Array Files

Each array is held at a different "header". Headers are limited to 4 characters. Arrays on Header Array files can hold real, integer or character data. See section 4.1 of GPD-1 for more details.

7.3 Why Use Expressions in q: Fields?

Experienced MPSGE users may be surprised that we encourage the use of expressions in `q:` fields (and other fields) in many cases.

Example. Consider the `$prod:w` block in SJMGE.MGE.

```
$prod:w             s:1.0
  o:pw              q:(sum(good, hous(good)))
  i:pc(good)        q:hous(good)
```

Since there is only a single output (commodity `pw`), the `q:` field value in the `o:pw` line must be the SUM shown if the data is balanced.

Similarly, in the `$prod:xcom(sect)` block in SJMGE.MGE there are sums in the `o:pc(sect)` line.

```
$prod:xcom(sect)    s:1.0
  o:pc(sect)  q:(sum(good, comin(good,sect)) + sum(fac, facin(fac,sect)))
  i:pc(good)        q:comin(good,sect)
  i:pf(fac)         q:facin(fac,sect)
```

Again, since there is only one output from the block, the `q:` field value in the `o:pc(sect)` line must be the expression shown if the data is balanced.

An equally valid MPSGE representation of the Stylized Johansen model would have the two blocks above written as

```
$prod:w             s:1.0
  o:pw              q:vw
  i:pc(good)        q:hous(good)

$prod:xcom(sect)    s:1.0
  o:pc(sect)        q:com(sect)
  i:pc(good)        q:comin(good,sect)
  i:pf(fac)         q:facin(fac,sect)
```

This introduces two extra Coefficients `vw` and `com(sect)`. In the GEMPACK implementation, the values of these Coefficients will be read from the pre-simulation database. Also, after a simulation is run, the updated values of these will be written to the updated data file.

We have provided this alternative representation (in file **`SJMG2.MGE`**) amongst the examples supplied with MGE2GP. If you look at the associated Header Array file SJMG2.HAR, you will see the values of `vw` and `com` at the relevant headers on that file.

The `vw` and `com` data held in SJMG2.HAR are redundant in the sense that they can be deduced from other data on the file. At least, this is true if the data in SJMG2.HAR represent an equilibrium of the model, which is always assumed when converting to GEMPACK (see section 7.1). That is the key reason why we have suggested the use of expressions in `q:` (and other) fields.

- If you are prepared to hold redundant data on the HAR file, then you don't need to write expressions in such fields. [An example is SJMG2.MGE.]
- If you prefer not to hold redundant data on the HAR file, you need to write expressions in `q:` (and other) fields in cases where the value can be deduced from the values in other fields (assuming that the data are balanced – that is, come from an equilibrium of the model). [An example is SJMGE.MGE.]

7.4 Arranging to Get All Relevant Data Updated

One of the important features of GEMPACK is that updated data is produced after every simulation. Ideally this updated data file represents the state of the economy it has fully adjusted to the shocks. An example is in section 5.2.1.

The updated data file only contains updated (that is, post-simulation) versions of data which is **read** from a data file when the simulation is carried out.

Ideally all input data into a model should appear in the updated data.

> For example, you may want to apply shocks in stages. Start from the initial data and apply one group of shocks. Then start from the updated data from the first simulation and apply a second group of shocks. Suppose that all data inputs to your model are updated in the updated data. Then, if you combine the two sets of simulation results (changes and percentage changes), the combined result will be what you would get if you applied the two groups of shocks in a single simulation, starting from the original data. However, if some data inputs are not updated, it is not at all clear what the second simulation is achieving. Indeed, the results from it may be misleading.

MGE2GP checks whether all data in your model can be updated. If not, no updated data is produced when you carry out simulations with the model.

The circumstances in which data in various fields in your MGE file can be updated are described in detail in the subsections below. For example, no updated data will be produced

- if you have a number (rather than a name) in any field in your MGE file. [See section 7.4.1 for details.]
- if you have a `q:` field consisting of a single name which is not of "full rank". [See section 7.4.2 for details and an explanation of "full rank".]
- in certain other cases which are described in sections 7.4.3 to 7.4.5 below.

Accordingly you should avoid these if you want to use the updated data. We give details in the subsections below.[63]

Note that, even if your MGE file does not conform to the above advice, the simulation results (that is, the changes and percentage changes, and the original and post-simulation levels results reported on the Solution file and viewed via ViewSOL) are still correct. For example, although `TWOBYTWO.MGE` has numbers rather than names in all relevant fields in the MGE file, the simulation results you see using ViewSOL (see, for example section) are still correct. These results would not change if you modified the MGE file to contain names rather than numbers.

7.4.1 Avoid Numbers in Fields

The TAB file written by MGE2GP expects to read from the Header Array file for the model data corresponding to any name used in the MGE file. However, nothing is read from the Header Array file when a number (rather than a name) is used in a field in the MGE file.

> Example. Consider the following `$prod:` block.
> ```
> $prod:x              s:1
>   o:px               q:qx
>   i:pl               q:50
>   i:pk               q:qk
> ```
> The corresponding TAB file will expect to read the values of `qx` and `qk`. [These will be Coefficients in the TAB file.] But there is no Read statement corresponding to the `q:50` field. [Instead there will be a Formula(Initial) which sets some Coefficient equal to 50.]

7.4.2 Use Names of Full Rank in q: Fields Which are Not Expressions

This section only applies to `q:` fields not containing an expression – for example,
```
    q:(sum,o,demand(o))
```
[An expression must begin with "(" and end with ")" – see section 10.2.]

It does apply to `q:` fields which contain a minus sign followed by a name – for example,
```
    q:-endl.
```

What we mean by "full rank" is best illustrated by an example.

Example 1. Consider the following `$prod:` block

```
$prod:xcom(sect)    s:1.0
  o:pc(sect)        q:com(sect)
  i:pc(good)        q:comin(good,sect)
  i:pf(fac)         q:facin(fac)
```

Look at the last line. Since there is no set `sect` in the `q:` field, this means that, in the pre-simulation data, the different `xcom` sectors use the same quantities of factor inputs. Even though this may be true in the pre-simulation data, it may not be true once shocks are applied. [For example, one sector may contract and another expand. Then they will use different amounts of the factors.] The TAB file written

[63] Note that t: fields never cause problems about updated data. This is because tax variables in the TAB file have the same arguments as those in the MGE file – see section 3.2.7.

must allow for these to become different. Accordingly you would see in the TAB file a Levels Variable QD_XCOM_PF declared to stand for the quantity of `pf` input into these sectors.

```
Variable(Levels, GE 0, Linear_Name=p_qd_xcom_pf)
 (All,sect_1,sect)(All,fac_1,fac) QD_XCOM_PF(sect_1,fac_1)
 # Quantity (level) in q: field of i:pf(fac) in $prod:xcom(sect) # ;
```

And you would find the following formula assigning the initial (that is, pre-simulation) values for this variable:

```
Formula (Initial) (All,sect_1,sect)(All,fac_1,fac)
  QD_XCOM_PF(sect_1,fac_1) = FACIN(fac_1) ;
```

There would also be a Coefficient called `FACIN` declared. Its values will be read but they cannot be updated since `facin(fac)` is not of full rank. So, if the updated data file was produced, the updated data would not contain the updated FACIN values. Instead (and this would be really misleading), the updated data file would contain the original (that is, the pre-simulation) values of FACIN. This would mean that the data on the updated data file would probably not be a valid data set (it would not be balanced) and so would not represent an equilibrium for the model.

This is why the updated data file is not produced when you carry out simulations starting from MGE files like this.

For example, if you run MGE2GP to process a model which includes the `$prod:` block in Example 1 above, you will see the following warning:

```
Warning while Processing function $prod:xcom(sect)
Processing line beginning i:pf(fac)
Processing "q:facin(fac)"
 %% Warning. Message follows.
 "facin(fac)" will not be updated - better to add full rank
symbol here and to read it.
  [Hence no updated data will be produced by simulations.]
This occurred while translating to GEMPACK.
See the accompanying documentation for more information.
```

So, the moral is, if you are concerned about the validity and integrity of the updated data,

always use names of full rank

When you run the program MGE2GP, you may see warnings about names not being of full rank. Those warnings relate to what we have described in this section.

Checking Full Rank

How do you tell if a name has full rank? You simply need to check if the entry in the relevant field contains all the indices in the `$prod:` field and all indices in the `o:` or `i:` field at the start of the relevant line in the `$prod:` block.

Example. Consider

```
$prod:y(s1,s2)
  i:pd(s2,o)    q:x3(s1,o,s2)
  o:pd(s2,o)    q:x4(o,s1)
```

The `q:x3(s1,o,s2)` is of full rank but the `q:x4(o,s1)` is not of full rank since index `s2` is not present.

7.4.3 When p: Fields Can Not Be Updated

A `p:` field is only needed if there are taxes on an `o:` or `i:` line.

- If a `p:` field contains a number, no updated data is produced (for the reasons explained in section 7.4.1). In particular, if there is no `p:` field on an `o:` or `i:` line containing taxes, then `p:1` is implied (see section 10.6.1), and so no updated data is produced.

- If a `p:` field contains a single symbol (for example, `p:bpo(s)`) then this `p:` field cannot be updated if the symbol does not contain all arguments appearing in `t:` fields on the line. For example, in
```
  i:pf(f)   q:factor(f,s)       p:bpf(f)     a:gov     t:btf(f,s)
```
  the `p:` field does not contain the argument `s` occurring in `t:btf(f,s)`. [The reason for updated data not being produced in this case is similar to the reasons in section 7.4.2.]

7.4.4 When m: Fields Can Not Be Updated

An `m:` field is only needed if there are endogenous taxes (that is, an `n:` field) in an `o:` or `i:` line (see section 10.6.1).

- If an `m:` field contains a number, no updated data is produced (for the reasons explained in section 7.4.1). In particular, if there is no `m:` field in an `o:` or `i:` line containing an `n:` field, then `m:1` is implied (see section 10.6.1), and so no updated data is produced.

7.4.5 Other Cases When Fields Cannot Be Updated

Updated data is not produced

- if there is a `q:` field containing an expression [beginning with "(" and ending with ")"] in an `e:` line which contains an `r:` field.

7.5 Use GAMS Syntax for Expressions and Equations in MGE Files

It is common to use expressions in some of the fields, especially `q:` fields (see section 7.3). In these fields, you should use GAMS syntax (rather than GEMPACK syntax) for SUMs. For example

```
q: (SUM(com,xhous(com))
```

[When MGE2GP writes the TAB file, this will be translated to the GEMPACK form, namely `SUM(com_1,COM, XHOUS(com_1))`.]

GEMPACK experts will need to take care to use sets as indexes (for example, the set `com` in the example above).

GAMS experts should note that, if they want to write down a sum over 2-tples, they should write the expression as a double sum rather than using tuples since this is what MGE2GP can handle. For example, use

```
SUM(s, SUM(t, x1(s,t) ))
```

rather than

`SUM( (s,t), x1(s,t))` [not allowed]

You should also use GAMS syntax when you write `$constraint:` equations, as explained in subsection 7.6 below.

You need to use GAMS syntax (rather than GEMPACK syntax) in the MGE file because

- MGE files can be used to model with GAMS (in which case, GAMS syntax has always been required) or can be translated to GEMPACK. You should not have to change the file you may have used with GAMS when you want to translate it to GEMPACK.
- you cannot reasonably be expected to guess what indexes MGE2GP will use when translating sets to GEMPACK indexes. [For example, sometimes `sum(com,xhous(com))` will be translated to `SUM(com_1,COM, XHOUS(com_1))` in the TAB file and sometimes it will be translated to `SUM(com_1@,COM, XHOUS(com_1@))` .]

7.6 Writing $constraint: Equations

You can write levels or linear equations here. If your MGE file is also to be used in conjunction with GAMS/MPSGE, you will want to write the `$constraint:` equations as levels equations since GAMS/MPSGE expects that.

- If you write a linear `$constraint:` equation, you must begin the equation with the qualifier **`(linear)`**. For example,
  ```
  $constraint:epl
    (linear)  pf("lab") = pu ;
  ```
- If you write a levels `$constraint:` equation, you do not put any qualifier before the equation, and you should use GAMS syntax. For example,[64]
  ```
  $constraint:xx
      gov =e= 0 ;
  ```

In either case, you should use GAMS syntax for SUMs.

If you write a levels equation, the program MGE2GP converts `$commodities`, `$sectors` and `$consumers` to the appropriate levels forms used in the TAB file. ["_L" is added at the end of the names of `$commodities` and `$sectors` and "VI_", which stands for "Value of Income" is added before the names of `$consumers`.]

For example, the equation in

```
$constraint:xx
    gov =e= 0 ;
```

is written in the TAB file as

```
    Equation (Levels)  VI_GOV = 0 ;
```

by MGE2GP.

Do not use GEMPACK quantifiers, for example "(all,f,fac)", when writing linear or levels `$constraint:` equations. For example,

```
$constraint:rls
  (Linear)   pf(fac) – px(fac) = 0 ;

$constraint:fs(fac)
  pf(fac) =e= px(fac) ;
```

[When MGE2GP writes the TAB file, the equations will be translated to the GEMPACK form. For example, the latter example above (a levels equation) will be translated to

```
   Equation (levels) (all,fac_1,fac) pf_L(fac_1) = pc_L(fac_1) ;
```

MGE2GP adds the quantifiers and changes to levels names used in the TAB file if necessary – see section 7.6.2 for details about name changes.]

7.6.1 Shifters in $constraint: Equations

In the ORANI spirit, we have made it easy for users to add so-called "shifters" to `$constraint:` equations.

There are several examples in `MINLMGE.MGE` for the MINIMAL model (see section 6.1). We describe the $constraint:epl equation in this context.

```
$constraint:epl
* With this constraint equation, employment level is free to adjust
*  endogenously and real wage (PL/PC) is fixed (or exogenous).
*  [Could shock realwage here.]
* (linear)  pl - pc = realwage ;
  pl/pc =e= realwage ;
```

[64] In GAMS syntax, use `=e=` between two sides of an equation.

Putting `realwage` by itself on the right-hand side of the above levels equation lets MGE2GP recognise that this should be made into a "shifter" variable in the TAB file and included as an exogenous variable in the standard closure in Command files for the model. For example, if you want to increase the real wage (which is one of the standard simulations with the MINIMAL model – see `MINLMGE-REALWAGE.CMF` and `MINIMAL-REALWAGE.CMF` in section 6.1), you can do this simply by shocking the variable `realwage`.

In general, if you put a single symbol on the left-hand side or the right-hand side of a levels `$constraint:` equation, and provided that this symbol has the same arguments (indexes) as the equation, MGE2GP recognises that this should be made into a shifter variable in the TAB file and included as an exogenous variable in the standard closure in Command files.

Note that a possible closure change in such cases is to swap the shifter variable with the variable in the `$constraint:` field. That turns off the `$constraint:` equation. This is also in the ORANI tradition of having several different closures.

For example, in MINLMGE.TAB derived from MINLMGE.MGE, the standard closure is to have `realwage` exogenous and `epl` endogenous. That is, wages are exogenous and employment adjusts. However an alternative closure is to have employment exogenous and wages endogenous. You can achieve this by adding the statement

```
swap  realwage = epl ;
```

to the Command files for MINLMGE.TAB. [This statement sets `realwage` endogenous and `epl` exogenous since the effect of a swap statement is to reverse the exogenous/endogenous settings of the two variables – see section 5.2.3 of GPD-3.]

7.6.2 Variable Names in $constraint: Equations may be Changed by MGE2GP

When MGE2GP writes the TAB file, the names of variables in levels `$constraint:` equations may be changed to the appropriate TAB file names.

- Central price and sector variables have **`_L`** added to their names.
  For example, in
  ```
  $constraint:fs(fac)
    pf(fac) =e= px(fac) ;
  ```
  the equation is written in the TAB file as
  ```
     Equation (levels) (all,fac_1,fac) pf_L(fac_1) = pc_L(fac_1) ;
  ```
- Income variables have **`VI_`** added at the start of their names.
- Names that represent quantities in the MGE file often represent values in the TAB file. These names will be changed to the TAB file name that represents the relevant quantity.
  For example, the $constraint:efx equation from MINLMGE.MGE
  ```
  $constraint:efx
    efx / sum(o, e(o)*export(o)) =e= befx ;
  ```
  is written as
  ```
    Formula & Equation E_efx
      BEFX = efx/SUM(o_1,o,e(o_1)*(-1.0)*UQS_RA_PD(o_1));
  ```
  in the TAB file `MINLMGE.TAB`. The `export(o)` in the MGE file refers to the unrationed quantity of exports. But `export(o)` represents the unrationed value of exports in the TAB file, while
  `–UQS_RA_PD(o)` represents the unrationed quantity of exports in the TAB file.

will be translated to the GEMPACK form. For example, the latter example above (a levels equation) will be translated to

```
   Equation (levels) (all,fac_1,fac) pf_L(fac_1) = pc_L(fac_1) ;
```

MGE2GP adds the quantifiers and changes to levels names used in the TAB file if necessary.]

7.7 Where Possible, Do Not Rely on Set Aliases

This section is intended mainly for GAMS/MPSGE experts.

The program MGE2GP only reads the MPSGE specification of the model. This specification does not include any GAMS Alias statements (which will appear in the .GMS file above the MPSGE part of your .GMS file containing the MPSGE model).

The program MGE2GP has to make inferences about the sets from what it reads in the MGE file. For example, it has to try to decide if two sets are in fact equal (that is, aliased in the GAMS sense) or if one set is a subset of another set.

The program MGE2GP can do a better job of translating the MGE file if you make minimal reliance on Aliases in the MGE file itself. So try to use just one symbol for each different set, especially in the declaration blocks (that is, the `$sectors`, `$commodities`, `$consumers` and `$auxiliary` sections). Also try to use the same sets in the first line of `$prod:` and `$demand:` blocks as were used in the declaration of the corresponding `$sector` and `$consumer` respectively. Of course you can use aliases in the other lines in `$prod:` and `$demand:` blocks.

Example. In `OPEN.MGE` (see section 2.2.3) you will see

```
$sectors:
  y(s)              ! production
  a(s)              ! aggregate supply
  e(s)              ! export index
  m(s)              ! import index

$commodities:
  pd(s)             ! domestic price of commodity
  pa(s)             ! price of commodity
  pf(f)             ! price of primary factor
  pe(s)             ! price of export
  pm(s)             ! price of import

$prod:y(s)          t:etrn(s)          va:esub(s)
  o:pd(s)           q:supply(s)
  o:pe(s)           q:export(s)
  i:pa(o)           q:interm(o,s)
  i:pf(f)           q:factor(f,s)      va:
```

- In the .GMS file we first wrote down to solve this model using GAMS/MPSGE, there was an
  `Alias(s,o) ;`
  statement. Also many of the declarations were written down over the set `o` rather than the set `s`.[65] With that MGE file, MGE2GP had a hard time figuring out if `s` and `o` were equal sets (since it does not read the Alias statement). So it makes life easier for MGE2GP if the set `s` is used in all the declarations and also if the `$prod:y(s)` block uses the same set `s` that y is declared over (rather than starting `$prod:y(o)` as it could in a .GMS file).
- Although the set `o` is intended to be the same as the set `s`, MGE2GP is only able to infer that `o` is a subset of `s` (as you will see if you look near the start of the file OPEN.TAB produced by MGE2GP).

7.8 Advice About Writing MPSGE Files for Conversion to GEMPACK

This summarises advice given elsewhere in this section.

- As far as possible, don’t use alias sets, especially in the declaration blocks. See section 7.7.
- Use names of full rank in fields in your MGE file. See section 7.4.2.

[65] For example, `e(o)`, `pd(o)` and `pe(o)`.

- Use GAMS syntax in expressions and equations. See section 7.5.
- Use expressions in fields if you want to avoid carrying around redundant data on the HAR file. See section 7.3.
- Use fairly short names in your MGE file if you want the TAB file names to be meaningful. See section 7.11.2.

7.8.1 $report: Section is Ignored by MGE2GP

You should be aware that MGE2GP ignores any `$report:` section in your MGE file. This is because GEMPACK Solution files already contain results for the large number of change and percentage-change variables introduced in the model.

7.9 Walras Law and the Numeraire

Walras law says that one equation in a general equilibrium model is a consequence of all the other equations in the model. It is usual to identify a suitable equation and to omit it from the equations in the TAB file (that is, the equations actually solved).

The program MGE2GP looks to omit a suitable **market clearing equation** to omit in accordance with Walras law, and it usually finds one, as explained below.

MGE2GP omits the market clearing equation for the first scalar commodity in the `$commodities:` section. The corresponding commodity (actually, its price) is the numeraire in the usual homogeneity simulation in the Command file written by MGE2GP.

For example, in `bop.mge` there are three scalar commodities, namely `pu` (price index for utliity), `pg` (price of a government output unit) and `pfx` (real exchange rate). The first of these in the `$commodities:` section is `pu`. Hence the market clearing equation for `pu` is omitted in `bop.tab` and this variable `pu` is exogenous in `bophsim.cmf` and is shocked by one percent. Notice also that the levels variable `WALRASSLACK` is added to the TAB file and that its value is determined by the Formula & Equation

```
Formula & Equation E_c_walrasslack
 WALRASSLACK = IF[ NOT[ TQD_PU GT 0], PU_L - 1 ] +
   IF[ TQD_PU GT 0, TQD_PU - TQS_PU ] ;
```

The market clearing equation for `pg` is

```
Equation (Levels) E_pg
  # Market clearing equation for commodity pg #
   IF[ NOT[ TQD_PG GT 0], PG_L - 1 ] +
   IF[ TQD_PG GT 0,
     TQD_PG - TQS_PG ] = 0 ;
```

As you can see, it is somewhat arbitrary whether `pu` or `pg` (or `pfx`) is selected as the numeraire. You can change between them (without changing the equation omitted). When you change numeraire, all quantity percent-change results remain unchanged (fortunately). All that happens is that percent-change results for prices and dollar values change to reflect the different price being held constant.[66]

[66] Suppose that you carry out two simulations with model `bop` which differ only in the choice of numeraire. Suppose that `pu` is the numeriare in the first simulation, and suppose that, amongst of the first simulation are: pg=1.1, pfx=0.2 and y("s1")=2.3 (where s1 is the first sector and y is the percentage change in its activity level). Of course the simulation result for `pu` is zero since it is held fixed. What would the corresponding simulation results be if `pg` were taken as the numeraire. Well, since prices are now reported relative to `pg`, the results would be (approximately) pg=0 (it is held fixed), pu=-1.1 (still 1.1% below pg), pfx=-0.9 (still 0.9% below pg) and y("s1")=2.3 (unchanged). If you find this surprising, try it our with any simulation with model `bop`.

However, if there is no scalar commodity in the `$commodities:` section, MGE2GP produces the TAB and Command files but ends with a warning that no equation has been left out to satisfy Walras law. It is your job to identify a suitable equation and to omit it. You will also need to add the corresponding price as an exogenous variable in the Command file and you will need to shock that variable if you wish to carry out the usual homogeneity simulation.

> For example, we provide a file `jointh.mge` in which there are several households and no scalar commodity. The natural equation to omit to satisfy Walras law is the market clearing equation for one household. The market clearing equation for `pu(h)` (price index for utility) is
>
> ```
> Equation (Levels) E_pu
>   # Market clearing equation for commodity pu #
>  (All,h_1,h)
>    IF[ NOT[ TQD_PU(h_1) GT 0], PU_L(h_1) - 1 ] +
>    IF[ TQD_PU(h_1) GT 0,
>      TQD_PU(h_1) - TQS_PU(h_1) ] = 0 ;
> ```
>
> In order to identify one of these equations to omit, you need to pick one of the elements of the set H of households and then write down only the market clearing equations for the remaining households. You could to that with the following statements
>
> ```
> Set House1 (h1) # The first household # ;
> Subset House1 is Subset of H ;
> Set RestHouse = H – House1 ;  ! Complement set !
> Equation (Levels) E_pu
>   # Market clearing equation for commodity pu #
>  (All,h_1,RestHouse)
>    IF[ NOT[ TQD_PU(h_1) GT 0], PU_L(h_1) - 1 ] +
>    IF[ TQD_PU(h_1) GT 0,
>      TQD_PU(h_1) - TQS_PU(h_1) ] = 0 ;
> ```
>
> [This assumes that "h1" is the name of the first element of the set H. Notice that you write down the market clearing equation for `pu` over the set `RestHouse`, not the whole set `H`.][67]

7.9.1 Always Check the Result for WALRASSLACK

Whenever you look at simulation results via ViewSOL, you should always check that

the result for c_WALRASSLACK is approximately zero.

If not, something has gone wrong.

7.10 The Equations and the TAB File

In section 3.2 above, we introduced the equations underlying an MPSGE model. You saw how the TAB file is arranged and looked at some of the equations for the `TWOBYTWO.TAB` file.

The general arrangement of the TAB files written by MGE2GP is always as indicated there. First come statements declaring the variables in the `$sectors:`, `$commodities:` and `$consumers:` parts of the MGE file. Then come the equations and other statements for each of the `$prod:` and `$demand:` blocks in the MGE file. Finally there are the market clearing equations.

[67] If you want to add WALRASSLACK to check the omitted equation you could add the lines
```
Variable (Levels, Change) WALRASSLACK ;
Equation (Levels) E_c_WALRASSLACK
  # Check of omitted market for household h1 #
 WALRASSLACK = IF[ NOT[ TQD_PU("h1") GT 0], PU_L("h1") - 1 ] +
   IF[ TQD_PU("h1") GT 0,      TQD_PU("h1") - TQS_PU("h1") ] ;
```
Here use the element `"h1"` in place of the index.

In section 3.2 you looked at some of the equations in `TWOBYTWO.TAB`. The equations and other statements in `TWOBYTWO.TAB` are especially easy to read (even if you are not used to GEMPACK) since all variables are scalars. When there are sets and vector and matrix variables, the equations are similar, but have arguments to indicate the vector and matrix nature of the equations.

> **Example**. Consider `SJMGE.TAB` written from `SJMGE.MGE` by MGE2GP. Open this file `SJMGE.TAB` in TABmate. You should see the different `$prod:` and `$demand:` blocks in the TAB file.
>
> Have a look in the code for the `$prod:xcom(sect)` part of the MGE file. You can see the CES demand for factors (labor and capital) is written as
>
> ```
> Equation (Linear) E_p_qfacin
>   (All,fac_1,fac)(All,sect_1,sect) p_qfacin(fac_1,sect_1) =
>     xcom(sect_1) - 1.0 * [pf(fac_1) - mc_xcom(sect_1)] ;
> ```
>
> This is a linearized equation, as the qualifier (Linear) indicates. The name of this equation block is E_p_qfacin. The so-called **quantifier (All,fac_1,fac)** indicates that there is one equation for each different element `fac_1` in the set `fac`. These quantifiers are required as part of the syntax in a TAB file.
>
> - GAMS users will be surprised by the indexes `fac_1` and `sect_1` in this statement. They would expect something more like
>   ```
>   p_qfacin(fac,sect) =
>    xcom(sect) - 1.0 * [pf(fac) - mc_xcom(sect)] ;
>   ```
> - GEMPACK users may also be surprised by the names `fac_1` and `sect_1` of these indexes. In a hand-written GEMPACK TAB file you might expect to see "f" and "s" used instead. The program MGE2GP often adds "_1" to the name of the set to produce an index name.
>
> The equation calculating the total supply of commodity `pf` is written as
>
> ```
> Formula & Equation E_TQS_PF
>   # Total quantity of supply for commodity pf #
>    (All,fac_1@,fac) TQS_PF(fac_1@) =
>        IF[ENDOW(fac_1@) GT 0,[68] ENDOW(fac_1@)] ;
> ```
>
> As you would expect, this just says that the total supply for each factor `fac_1@` is equal to `ENDOW(fac_1@)` (which is the quantity in the `q:` field in the `e:pf(fac)` line in the `$demand:y` block). Both GAMS and GEMPACK users will probably be surprised to see the odd-looking index **`fac_1@`** used by MGE2GP in this equation.[69] You will also see this sort of index used in other supply and demand equations in `SJMGE.TAB`.[70]

7.11 Technical Details about the TAB File Written

7.11.1 Editing TAB and Command File

Editing the Command File

You will certainly need to edit the Command file to change the shocks and verbal description if you want to carry out other simulations besides the homogeneity simulation. You may need to modify the closure and/or solution method.

[68] The "IF" expression "IF[ENDOW(fac_1@) GT 0," is because q: fields may contain negative endowments – see section 3.2.4.

[69] In TAB files, names must begin with a letter. They can contain other letters (a-z), digits (0-9) and the two characters "_" [underscore] and "@". See section 4.2.1 of GPD-2 for details.

[70] If you look at the equation for total demand of commodity `pa` in `OPEN.TAB`, you will see that the both the indexes `s_1@` and `s_1` are used.

Editing the TAB File

You may need to make small changes to the TAB file produced by the program MGE2GP.

- You may need to modify the Headers referred to so that the headers in the Read statements in your TAB file match the headers on MODEL.HAR.
- If MGE2GP cannot identify an equation to omit to satisfy Walras law, you may need to edit the TAB file. See section 7.9 for details.
- If your `$constraint:` equations are not translated well by MGE2GP (see section 7.6), you may need to edit them on the TAB file.

7.11.2 Names of the Variables and Coefficients

As you have seen in section 3.2, MGE2GP tries to make up names for Variables and Coefficients that make it clear what block and line they come from. For example, look at the names `QD_X_PL` in `TWOBYTWO.TAB` and `TOVYPDGOV` in `TAXOUT.TAB`.

Of course these names become too long if the names used in the MGE file are long. If that happens, the program MGE2GP replaces the last 4 or so characters in the name by digits in order to satisfy the limits on the length of names of Coefficients and Variables in TAB files.

> For example, suppose that you change `y(s)` to `prod(s)` and change `pd(o)` to `pdom(o)` in `TAXOUT.MGE`. Then, if the general pattern were followed, the name `TOVYPDGOV` would become `TOVPRODPDOMGOV` which is too long, so MGE2GP instead uses the name `TOVPRODP0001`.

7.11.3 Sets

At present, for every set referred to in the MGE file, the GEMPACK TAB file produced by MGE2GP expects to find the elements of this set at a header on the associated Header Array file. See section 7.2.2.

7.11.4 Subsets

MGE2GP infers subset statements from the syntax and semantic rules for MPSGE files (see section 10). These inferences are made as follows.

- Suppose that the `$sectors:` section contains
  ```
    a(s)              ! aggregate supply
  ```
  and that there is a `$prod:` block beginning `$prod:a(o)`. Then the syntax rule that each sector must be defined in just one `$prod:` block (see section 10.1) implies that the sets `S` and `O` must be equal sets. Hence MGE2GP adds the statements
  ```
  Subset O is Subset of S ;
  Subset S is Subset of O ;
  ```
  to the TAB file.
- Suppose that the `$consumers:` section contains
  ```
    ra(h)             ! representative agent income
  ```
  and that there is a `$demand:` block beginning `$demand:ra(g)`. Then the syntax rule that each consumer must be defined in just one `$demand:` block (see section 10.1) implies that the sets `H` and `G` must be equal sets. Hence MGE2GP adds the statements
  ```
  Subset G is Subset of H ;
  Subset H is Subset of G ;
  ```
  to the TAB file.
- Suppose that the `$commodities:` section contains
  ```
    pc(sect)          ! price of commodity sect
  ```
  and that there is a `$prod:` block containing the line
  ```
    i:pc(good)        q:qcomin(good,sect)
  ```

Then the set `GOOD` must be a subset of the set `SECT`. Hence MGE2GP adds the statement

```
Subset GOOD is Subset of SECT ;
```

to the TAB file. [Similarly for other `o:` lines in a `$prod:` block or for `e:` and `d:` lines in a `$demand:` block.]

7.11.5 Sets over which Sector and Consumer Variables are Declared

Usually the TAB file Variables associated with the `$sectors:` and `$consumers:` parts of the MGE file are declared with arguments ranging over the same sets as in that part of the MGE file. However, different sets may be used in some cases.

- Suppose that the `$sectors:` section contains
  ```
    a(s)              ! aggregate supply
  ```
  Normally MGE2GP will declare `a` to be a Variable ranging over the set `S`. But suppose that the `$prod:` for variable a begins `$prod:a(o)`. Then the syntax rules imply that the sets `S` and `O` must be equal sets (see section 7.11.4). MGE2GP declares a to be a Variable ranging over the set O.
  ```
  Variable (all,o_1,o) a(o_1)
       # aggregate supply (%-change) ;
  ```
  in the TAB file. [The choice of "O" rather than "S" is made for technical reasons.]

- Suppose that the `$consumers:` section contains
  ```
    ra(h)             ! representative agent income
  ```
  Normally MGE2GP will declare `ra` to be a Variable ranging over the set `H`. But suppose that the `$demand:` block for `ra` begins `$demand:ra(g)`. Then the syntax rules imply that the sets `H` and `G` must be equal sets (see section 7.11.4). MGE2GP declares `ra` to be a Variable ranging over the set G.
  ```
  Variable (all,g_1,g) ra(g_1)
     # representative agent income (%-change) # ;
  ```
  in the TAB file. [The choice of "G" rather than "H" is made for technical reasons.]

7.11.6 The TAB File is a Mixed Levels/Linear File

As you have seen from the earlier examples (see sections 3.2 and 4.7), the TAB file written by MGE2GP contains many explicitly levels variables and equations. It turned out to be most convenient and natural to write the accounting relations in levels and to write the behaviour (the CES demand and supply functions) as linearized equations. The TAB file written by MGE2GP is a good example of the strategy suggested in Harrison *et al* (1994).

There are linear equations in the TAB file are those involving the elasticities in `$prod:` blocks.

For example, consider the following `$prod:` block in `joint.mge.`

```
$prod:u                 s:esubc
  o:pu                  q:(sum(o, demand(o)))
  i:pd(o)               q:demand(o)
```

The only linear equations associated with this block are

```
Equation (Linear) E_p_qs_u_pu
   p_qs_u_pu = u + 0 * [pu - mr_u] ;
Equation (Linear) E_p_demand
  (All,o_1,o) p_demand(o_1) =  u - ESUBC * [pd(o_1) - mc_u] ;
```

[These involve the elasticities `ESUBC` (input substitution) and 0 (output substitution).]

The equations determining the percentage changes in marginal revenue and marginal costs associated with each `$prod:` block are also written in the TAB file as linear equations, as is the zero profit condition in each `$prod:` block.

The market clearing equations and the equations calculating total supply and demand for each commodity are levels equations.

For example, several of these equations from `twobytwo.tab` are set out in section 3.2.4.

Links between levels and linear variables are often made using the `Linear_Var=` and `Linear_Name=` Variable qualifiers.[71]

For example, you can see several of these in `open.tab` as written by MGE2GP.

7.11.7 Assertions

There are Assertions which check that the pre-simulation (that is, benchmark) values read in are a solution of the model (or, alternatively, that the pre-simulation data is balanced). The current GEMPACK implementation of an MGE file (as produced by the program MGE2GP) assumes that you are starting from a balanced database – see section 7.1. If the initial data is not balanced, this Assertion will fail and the simulation will not run.

There are also Assertions which check that the data remain balanced (to within some sensible tolerance) at the start of each step of a multi-step simulation. If one of these Assertions fails, it usually means that the solution being produced is not very accurate. The remedy is to increase the accuracy. You can do that by specifying more subintervals in the Command file.

For example, have a look at the Assertions that "costs = revenue for $prod:xcom" in `SJMGE.TAB`. Notice that the checks of the pre-simulation data are Assertion(Initial)s while the checks at the start of each step are Assertion(Always) since they are done on each step.

7.12 Restrictions on MGE Files

7.12.1 d: Fields in a $demand: Block

The program MGE2GP does not support more than a single `d:` field in a single `$demand` block. Formally, consumers may only demand a single commodity. Utility functions must be mapped onto the `$prod` block for the demanded goods. A `d:` record in a `$demand` block has a single field – no `q:` or `p:` fields are permitted.

This restriction is part of the new syntax rules for MPSGE (see section 10.1), so the program MGE2GP will always impose this restriction.

7.12.2 Restrictions on Names in MGE Files

The program MGE2GP imposes the following restrictions on the names used in the MGE file.

[Details about the maximum length of names in GEMPACK, and the characters allowed in names in GEMPACK TAB files, can be found in section 4.2.1 of GPD-2.]

- Names used for market prices in the `$commodities` section, and names used in the `$sectors` section, are limited to 10 characters. This is because the program MGE2GP adds "_L" at the end of these names and creates a GEMPACK Coefficient (and Levels Variable) from the resulting name. [Coefficient names in GEMPACK are limited to 12 characters.]
- Names used in the `$consumers` section are limited to 9 characters. This is because the program MGE2GP adds "VI_" at the start of these names and creates a GEMPACK Coefficient (and Levels Variable) from the resulting name.
- Names of sets are limited to 12 characters since set names in GEMPACK have this limit.
- Names of set elements (they appear inside double quotes "" in the MPSGE file) are limited to 12 characters since set elements in GEMPACK have this limit.

[71] These qualifiers were introduced in Release 8.0 (October 2002) of GEMPACK. See section 2.2.2 of GPD-2.

- Names of what GAMS calls Parameters appearing in the various fields (for example, in q: and t: fields) are limited to 12 characters since these will be used as names of GEMPACK Coefficients (which are limited to 12 characters).
- All names in the MPSGE file which are used as GEMPACK names must begin with a letter and contain only letters (A to Z, a to z), digits (0 to 9), underscores "_" or the character "@".
- Certain names are restricted in GEMPACK and so cannot be used as the names of sets or Coefficients. For example, SUM and EQ should not be used in this way. See section 4.2 of GPD-2 for a list of GEMPACK reserved words. If you use one of these restricted names in your MGE file, you will not find out about the error until you run the GEMPACK program TABLO to check the file. [The program MGE2GP makes no attempt to screen for GEMPACK reserved words.]

7.13 MPSGE Features Net Yet Supported

Some advanced MPSGE features are not supported at present by the program MGE2GP when it converts to GEMPACK. These include the following.

- At present **domain restrictions** (see section 10.5) are only allowed in declarations of `$commodities` or `$sectors`, but are not handled in declarations of `$consumers`.
  For example,
  ```
    pe(s)$export(s)   ! price of export
  ```
  is allowed in the `$commodities` section.
- At present **domain restrictions** (via $ exceptions – see sections 2.6.1 and 10.5) are only allowed
  on `$prod:` blocks,
  on `o:` or `i:` fields in `$prod:` blocks.
  Exceptions in other places are not handled and will probably result in an error.
  For example,
  ```
  $prod:e(s)$export(s)
    i:pe(s)$x1(s)    q:export(s)
  ```
  are allowed.
- The spanning operator `#` (see section 2.6.3) is not allowed.
- Sets of nests labelled with text labels – for example, `s.tl` (see section 2.6.2) – are not allowed.

If the program encounters such features in the MPSGE file, it will stop with a fatal error.

We plan to develop MGE2GP over the next year or so in order to gradually eliminate some of these restrictions.

7.14 Known Limitations on the Current Version of MGE2GP

We are aware of some unfortunate limitations on the current version of MGE2GP. We expect that others will come to light when the program is used more widely. We will be grateful if users alert us when they find something in this category (see section 3.3).

We apologise for these limitations. We want to at least be able to warn users about them. We also hope to fix them in future versions of MGE2GP.

Limitations we know about at present are:

- When deciding whether to produce updated data, MGE2GP should check if each coefficient in the MGE file occurs in at least one position from which it can be updated. [A coefficient can be updated if it occurs by itself or with just a negative sign in front of it in any "exogenous" field (for example, a `t:` field) or with the appropriate rank in any "endogenous" field (for example, `q:` and `p:` fields are "endogenous" fields).]
- When recognising one side of a `$constraint:` equation as a shifter, MGE2GP should check that this symbol contains every equation index exactly once (since, otherwise, the Formula part of the Formula & Equation written in the TAB file by MGE2GP will not be valid).

- Some MGE files lead to TAB files which have redundant equations. For example, the current version of MINLMGE.MGE (see section 6.1) contains the following two lines in the `$demand:ra` block.
  ```
  e:pj      q:(-invest)
  e:pg      q:(-govexp)
  ```
  If the brackets were omitted in either of the q: fields, the TAB file written by MGE2GP would have redundant equations. [This is why we have put the brackets in.]
- The syntax checking of expressions in the MGE file (see, for example, section 7.5) is not very robust. It will usually work ok if correct syntax is used in the MGE file. But it does not have good error trapping or reporting in the case of syntax errors.
- The translation of GAMS-like expressions, especially those in levels `$constraint:` equations (see section 7.6), to GEMPACK is limited. While MGE2GP can do simple translations, its knowledge in this area is at present limited. The translation may fail in complicated cases.

7.15 Likely Future Changes to MGE2GP

We have identified some changes to MGE2GP which would, we think, make it easier to use. We plan to make these changes over the next few months. We welcome comments and suggestions from users about these and other similar matters.

7.15.1 MGE2GP Could Know the Contents of the HAR File

At present MGE2GP knows nothing about the headers on the associated HAR file, or about the names of the Coefficients whose values are normally read from these headers. We are planning to let MGE2GP look at the HAR file (as well as the MGE file) and to use its knowledge of the headers and Coefficients there to

- put the actual headers on the file in the relevant Read statements in the TAB file (rather than having to use some rules such as those set out in section).
- check the headers at which to find the elements of the various sets.

7.15.2 Possible New Rule for Reading Set Elements from the HAR File

We are wondering about introducing a new rule, as follows, into the MGE2GP code.

- Any set in the declaration part of the MGE file must have its elements on the HAR file.
- For other sets occurring in the definition part of the file, if we can infer that the set is equal to a set in the declaration part, do not read its elements from HAR file, instead put set equality statement.
- For other sets in the definition part, read their elements from the HAR file.

8. Future Work

Our translation of the MPSGE language to GEMPACK is incomplete. There are a number of important MPSGE features which are currently not supported by MGE2GP – see section 7.13.

During the next year or so, we plan to gradually extend MGE2GP so that it can handle more and more of these features. As part of this process we will implement versions of several standard GEMPACK models using the MPSGE formulation. We hope that these models will include ORANI-G and TERM. We may try to teach MGE2GP to write explicit Complementarity statements so that the output TAB files can handle cases in which activities are driven to zero.

We plan to use MGE2GP in teaching and also for model and software comparison. We hope that others may do this also.

Over the much longer term, we expect that MPSGE might provide a starting point for the development of an even more effective non-algebraic languages for applied general equilibrium models. It would, for example, be intriguing to implement a Windows-based expert system to assist in model formulation in the same way that TABmate and AnalyseGE facilitate the analysis of operational models.

9. References

Brooke, T., D. Kendrick and A. Meeraus (1988), *GAMS: A User's Guide*, The Scientific Press, Redwood City, California.

Brooke, T., D. Kendrick and A. Meeraus (1998), *GAMS: A User's Guide*, GAMS Development Corporation, Washington.

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, W.J., K.R. Pearson, A.A. Powell and E.J. Small (1994), 'Solving Applied General Equilibrium Models Represented as a Mixture of Linearised and Levels Equations', *Computational Economics*, vol. 7, pp. 203-223.
[A preliminary version was *Impact Preliminary Working Paper* No. IP-61, Monash University, Clayton, September 1993, pp. 20.]

Hertel, T.W., J.M. Horridge and K.R. Pearson (1992), 'Mending the Family Tree: A Reconciliation of the Linearised 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.]

Rutherford, T. F. (1985), 'MPS/GE user's guide', Department of Operations Research, Stanford University.

Rutherford, T. F. (1987), 'Applied general equilibrium modeling', PhD dissertation, Department of Operations Research, Stanford University.

Rutherford, T. F. (1995), 'Extensions of {GAMS} for complementarity problems arising in applied economics', *Journal of Economic Dynamics and Control*, pp. 1299-1324.

Rutherford, T. F. (1999), 'Applied General Equilibrium Modeling with MPSGE as a GAMS Subsystem: An Overview of the Modeling Framework and Syntax', *Computational Economics*, 14, pp. 1-46.

9.1 GEMPACK Documentation

Harrison, W.J. and K.R. Pearson (2002), *An Introduction to GEMPACK*, GEMPACK Document No. 1 **[GPD-1],** Monash University, Clayton, Sixth edition, October 2002, pp.207+9.

Harrison, W.J. and K.R.Pearson (2002), *TABLO Reference*, GEMPACK Document No. 2 **[GPD-2]** Monash University, Clayton, Fourth edition, October 2002, pp.191+10.

Harrison, W.J. and K.R.Pearson (2002), *Simulation Reference: GEMSIM, TABLO-generated Programs and SAGEM*, GEMPACK Document No. 3 **[GPD-3]**, Monash University, Clayton, Second edition, October 2002, pp.262+12.

Harrison, W.J. and K.R.Pearson (2002), *Useful GEMPACK Programs*, GEMPACK Document No. 4 **[GPD-4]** Monash University, Clayton, Second edition, October 2002, pp.138+10.

Harrison, W.J. and K.R. Pearson (2002), *Installing and Using the Source-Code Version of GEMPACK on Windows PCs with Lahey Fortran*, GEMPACK Document No. 6 **[GPD-6],** Monash University, Clayton, 11th edition, October 2002, pp.39+7.

Harrison, W.J. and K.R. Pearson (2002), *Installing and Using the Executable-Image Version of GEMPACK on Windows PCs*, GEMPACK Document No. 7 **[GPD-7],** Monash University, Clayton, 8th edition, October 2002, pp.33+6.

Harrison, W.J. and K.R. Pearson (2002), *Getting Started with GEMPACK: Hands-on Examples*, GEMPACK Document No. 8 **[GPD-8],** Monash University, Clayton, Third edition, October 2002, pp.110+8.

10. Appendix 1 – Syntax Rules for MPSGE Files

These are the syntax and semantic rules for the current version (Version 2, May 2004) of MPSGE files.

10.1 New Restrictions

The syntax and semantic rules are as laid out in Rutherford (1999), but with the following restrictions:

- Each sector must be defined in just one `$prod:` block.
- Each consumer must be defined in just one `$demand:` block.
- The declared domain for sectors must exactly match the domain of the associated `$prod:` blocks.
- The declared domain for consumers must exactly match the domain of the associated `$demand:` block.
- A `d:` field within a `$demand:` block is now interpreted as a pointer to the consumers associated consumption good. Each `$demand:` block must contain exactly one `d:` field, and that must be the sole field in the record – `q:` and `p:` fields are not permitted.

10.2 Basic Rules of Syntax

The following rules apply the statements which define an MPSGE model, independent of whether the model is solved in GAMS or GEMPACK.

All input is free format (spaces and tabs are ignored), except for keywords which must be preceded by a "$" appearing in column 1.

Input is organized into records which may span one or more liens. Continuation lines which are indicated by a "+" in column 1.

Text is not case sensitive.

Numeric fields may be specified as computed values using expressions involving GAMS syntax based on parameters/coefficients or constants. Fields based on computed values must be enclosed in parentheses.

10.3 MPSGE Section Heading Keywords

There are nine MPSGE keywords which appear in an MPSGE model declaration. These are

- **`$MODEL: modelID`**

  This statement assigns an identifier to the present model. This must be the first statement within the MGE} file. All records appearing prior to the `$MODEL` keyword within the MGE file are treated as comments.

  modelID must be both a legitimate model identifier and a valid file name. This name is used to form modelID.GEN.

- **$SECTORS:, $COMMODITIES:, $AUXILIARY:, $CONSUMERS:**

  These four keywords define the variables which are used to define the MPSGE model. Entries in these blocks share the same syntax.

- **`$AUXILIARY:`**

  The `$AUXILIARY` keyword is only used in models with side constraints and endogenous taxes or rationed endowments.

- **`$PROD:sector`**

This statement indicates the start of a set of records which define technology and taxes for a particular activity.

- **`$DEMAND:consumer`**

  This statement indicates the start of a set of records which define initial endowments and preferences for consumers in the model.

- **`$CONSTRAINT:auxiliary`**

  This statement specifies the equilibrium condition which is to be associated with a particular auxiliary variable. Equilibrium conditions are written following the conventional rules for expressing equations in an algebraic form. An auxiliary constraint may reference variables corresponding to commodity prices, production activity levels, consumer income levels or report variables.

10.4 Variable Declarations

MPSGE requires that variables be declared according to their specific role in a model. The four MPSGE keywords `$SECTORS:`, `$COMMODITIES:`, `$CONSUMERS:` and `$AUXILIARY:` therefore refer to the four different types of unknowns which may appear in a general equilibrium model.

When variables are defined according to their classification the compiler is then able to verify that when a variable appears within a function declaration it is properly referenced.

Here is a typical declaration block, one that refers to three classes of production activities in a given model:

```
$SECTORS:
   Y(R,T)  ! Output in region R in period T
   K(T)    ! "Aggregate capital stock, period T"
   I0      ! Base year investment
```

In the MPSGE syntax trailing comments (signified by "!") are interpreted as variable descriptors which subsequently appear in the solution listing. When a variable descriptor contains a punctuation symbol such as ",", it is required to enclosed in quotes.

10.5 Domain Restriction

MPSGE is designed so that all and only those variables which are declared are actually employed in a particular model. This form of "explicit declaration" helps to catch nuisance bugs in large-dimensional datasets.

The explicit declaration feature in MPSGE means that "wildcard declarations", such as `X(*,*,*)` are not permitted. It also means that some care must be exercised when a database includes "missing goods". Consider, for example, a model in which g is the set of goods in the underlying database, and `P(g)` is the associated vector of prices for these commodities. If a subset of the goods are missing from the database, then the declaration of `P(g)` must be restricted to those goods which are actually appear in the model. If `y0(g)` is a vector in which a value of zero indicates that the associated good is not in the dataset. Appropriate declaration for the price vector could then be:

```
$COMMODITIES:
        P(g)$y0(g)      ! Commodity price vector
```

Detailed cross-checking can be somewhat annoying for users accustomed to the conventional algebraic syntax in which the declared domain of a variable may include many more scalar elements than are actually used in the model.

In order to simplify the declaration of "irregular domains" for variables in MPSGE, the declaration syntax accommodates the conditional operator "$".

To illustrate how this might work in a more complicated situation, suppose that a dynamic, multi-sectoral, multi-regional model accounts for the production of good i in region r and time period t with

`X(i,r,t)`. Suppose that base year production levels are given by `x0(i,r)`, and suppose, furthermore, that the model includes an upper bound on production in later periods given by `u(i,r,t)`.

We would then only want to declare `X(i,r,t)` when both `x0(i,r)` and `u(i,r,t)` are nonzero. The MPSGE declaration to do this could be:

```
$SECTORS:
  X(i,r,t)$(min(x0(i,r),u(i,r,t)) > 0)  ! Production levels
```

The computed conditional ("`$()`") is handy because it avoids the need to introduce any additional symbols into the program. If, however, the same conditional restriction appears at several points in a model it may be helpful to define a set with which to control declarations for several variables.

Using a coefficient array, the previous example could be written:

```
$SECTORS:
        X(i,r,t)$IX(i,r,t)      ! Production levels
        ...
$COMMODITIES:
        PX(i,r,t)$IX(i,r,t)     ! Output prices
        RK(i,r,t)$IX(i,r,t)     ! Capital returns
        ...
```

10.6 Function Declarations

A general equilibrium model is based on technology, preferences, policy instruments (most notably tax rates) and primary factor endowments. Technology and policy instruments are described in `$PROD:` blocks. Preferences and primary factor endowments are described in `$DEMAND:` blocks.

10.6.1 $PROD Block Syntax

Most of the power and subtleties of the MPSGE framework for applied general equilibrium analysis centers on the `$PROD` tables. The information conveyed in a `$PROD` block includes which is characterized:

- Reference quantities for inputs and outputs.
- Reference producer prices of inputs and outputs.
- Exogenous taxes applied to inputs and outputs.
- Endogenous taxes applied to inputs and outputs.

The nested constant elasticity of substitution (CES) / constant elasticity of transformation (CET) elasticity structure at the reference point.

All records in a `$PROD` block must begin with either an

- `I:` or `O:` record. The recognized labels in these records within a `$PROD` block include:
- `Q:` Reference quantity. Default value is 1. When specified, it must be the second field, immediately following `I:` or `O:`.
- `P:` Reference price. The default value of the reference price is 1.
- `A:` Tax revenue agent. This field must be followed by a consumer name.
- `T:` Tax rate field identifier. (More than one tax may apply to a single entry.)
- `N:` Endogenous tax. This label must be followed by the name of an auxiliary variable.
- `M:` Exogenous tax multiplier. The *ad valorem* tax rate is the product of the value of the endogenous tax (as in the `N:` field) and this multiplier. If the `M:` field is not present in a line containing an `N:` field, `M:1` is assumed.

- `va:, kle:`, etc. Nesting assignments referencing identifiers introduced in the `$PROD` record. Only one such label may be applied at a time, although several nesting assignemnts may be included on a given line when exception operators identify only a single nest assignment to be included.

A given structure of nest and subnest elasticities corresponds to a particular set of own- and pairwise elasticities and a specific second-order approximation to the Jacobian matrix at the benchmark point.

10.6.2 A Simple $PROD Block

Here is a typical production function, one in which factors of production from each region r are combine in a Cobb-Douglas function to produce output:

```
$PROD:Y(r)  s:1
O:P(r)      Q:y0(r)
I:W(f,r)    Q:x0(f,r)   P:w0(f,r)
```

The price variables which appear in this function are `P` and `W` (both declared as `$COMMODITIES:`). The activity variable in the function is `Y`, a variable which must have been declared in `$SECTORS`:. Both `f` and `r` which appear in this function are sets, and `y0`, `x0` and `w0` must have been declared previously in the program as parameters.

This `$PROD` block characterizes a Cobb-Douglas production function in which the elasticity of substitution between inputs is one, as indicated by "`s:1`" in the first line. The `s:` field in a `$PROD` block relates the top level substitution.

The `O:` label indicates an output, and the `I:` label indicates an input. The `Q:` fields in both records represent "reference quantities". The `P:` field in the input record refers to the reference price of the input.

10.6.3 $PROD Block with Nested Functions

The following production function describes a nested CES production function, one in which good `X` is produced using labor, capital and intermediate inputs:

```
$PROD:X  s:esub  id:0  va:(esub/5)
O:PX        Q:x0
I:PY(g)     Q:yx0(g)  id:
I:PL        Q:lx0     P:pl0  va:
I:PK        Q:kx0     P:pk0  va:
```

The first record in the `$PROD` block contains the nest identifier for top level inputs (a reserved name), `s:`. The record introduces two new nest identifiers, `id:` and `va:`. These nest identifiers are arbitrary names with up to four characters. The nest identifier for top level outputs is `t:`. Both `t:` and `s:` are reserved names.

In the `$PROD` block shown here, commodities `PY(g)` enter jointly in fixed proportions in nest `id:`, while labor (`PL`) and capital (`PK`) enter as part of the value-added nest (`va:`).

10.6.4 $PROD Block -- Joint Outputs

The following production function describes a production function which produces goods `D` and `E` as joint products:

```
$PROD:E  t:eta
O:PD        Q:d0        P:pd0
O:PE        Q:e0        P:pe0
I:PY        Q:y0
```

The first record in this `$PROD` block specifies a value for t:, the top-level elasticity of transformation. Commodities with prices `PD` and `PE` are produced using inputs of the commodity with price `PY`.

10.6.5 Taxes on Inputs in a $PROD Block

An input record in a production block may contain the following fields related to *ad valorem* taxes:

- `A:` Tax revenue agent. This field must reference a consumer name.
- `T:` Tax rate field identifier. (More than one tax may apply to a single entry.)
- `N:` Endogenous tax. This label must be followed by the name of an auxiliary variable.
- `M:` Exogenous tax multiplier. The *ad valorem* tax rate is the product of the value of the endogenous tax (as in the `N:` field) and this multiplier. If the `M:` field is not present in a line containing an `N:` field, `M:1` is assumed.

Rules relating to the application of taxes on inputs are as follows:

A tax revenue agent must be specified prior to a `T:` or `N:` field. One important extremely valuable service provided by MPSGE is that taxes may not be applied without consistent accounting of associated tax revenue in consumer incomes. A tax therefore cannot be introduced without have specifying a consumer which receives the revenue.

If revenue from a particular tax is to be paid to more than one consumer, then two (or more) tax fields may be introduced on a single input with revenue accruing to different consumers. In the following example two taxes are applied on the input of `PL`, one of which accrues to consumer `FED` (the federal government) and another which accrues to consumer `STATE` (the state government):

```
      I:PL   Q:l0   A:FED  T:tlfica  A:STATE  T:tlinc
```

The tax rate on an input is evaluated on a net basis. The user cost of an input is the market (net) price marked up by all applicable taxes. In the previous example the user cost of labor is given by:

```
      PLUSER = PL * (1 + tlfica + tlinc)
```

The net tax must be greater than -1, i.e. the user cost of any input must be positive.

Endogenous taxes are described by a pair of fields, labelled `N:` and `M:`. The `N:` field identifies the auxiliary variable which is proportional to the tax rate, and the `M:` field indicates the proportionality factor. Consider the input record:

```
      I:PL Q:l0 A:FED N:TFED M:tlfica  A:STATE  N:TSTATE M:tlinc
```

In this case the user cost of labor is given by:

```
      PLUSER = PL * (1 + TFED*tlfica + TSTATE*tlinc)
```

10.6.6 Taxes on Outputs in a $PROD Block

An output record in a production block may contain fields relating to proportional taxes:

- `A:` Tax revenue agent. Must be followed by a consumer name.
- `T:` Tax rate field identifier. (More than one tax may apply to a single entry.)
- `N:` Endogenous tax. This label must be followed by the name of an auxiliary variable.
- `M:` Exogenous tax multiplier. The *ad valorem* tax rate is the product of the value of the endogenous tax (as in the `N:` field) and this multiplier. If the `M:` field is not present in a line containing an `N:` field, `M:1` is assumed.

Rules relating to the application of taxes on outputs are as follows:

A tax revenue agent must be specified prior to a `T:` or `N:` field.

If revenue from a particular tax is to be paid to more than one consumer, then two (or more) tax fields must be introduced. In the following example two taxes are applied on the output of `PX`, one of which accrues to consumer `FED` (the federal government) and another which accrues to consumer `STATE` (the state government):

```
        I:PX   Q:l0   A:FED  T:tvat  A:STATE  T:tsales
```

The tax rate on an output is evaluated on a gross basis. The user price of an output is the market (gross) price marked down by all applicable taxes. In the previous example the producer price of X is given by:

```
        PXUSER = PX (1 - tvat - tsales)
```

The gross tax must be less than 1, i.e. the producer price of any output must be positive.

Endgenous taxes are described by a pair of fields, labelled N: and M:. The N: field identifies the auxiliary variable which is proportional to the tax rate, and the M: field indicates the proportionality factor. Consider the output record:

```
        I:PX   Q:l0   A:FED  N:TFED  M:tvat  A:STATE  N:TSTATE M:tsales
```

In this case the user cost of labor is given by:

```
        PXUSER = PX * (1 - TFED * tvat - TSTATE * tsales)
```

10.6.7 Representing Technical Change

Suppose that good X is produced using inputs of K and L. Write the production function in calibrated share form:

```
        X = X0 * (theta*(L/L0)**rho + (1-theta)*(K/K0)**rho)**(1/rho)
```

At reference prices PL0, PK0, the demands for labor and capital are L0 and K0, and the level of output is X0.

We use this point to define the value share, i.e.

```
        theta = PL0*L0/(PL0*L0+PK0*K0)
```

Now, suppose we have a technical change parameter which is labor augmenting, so:

```
        X=X0*(theta*(L*gamma/L0)**rho+(1-theta)*(K/K0)**rho)**(1/rho)
where gamma>1.
```

We can alter the function by setting L0′=L0/gamma, but we need to hold theta fixed, we let PL0′=gamma*PL0.

In the MPSGE program, technical change then is represented as:

```
        $PROD:X s:esub
        O:PX Q:X0
        I:PL Q:(L0/gamma) P:(PL0*gamma)
        I:PK Q:K0 P:PK0
```

The intuitive explanation for introducing gamma in the reference price is that this incorporates the "rebound effect" which is commonly part of technical change.

10.6.8 $DEMAND Block Syntax

A demand block provides an indicator of the commodity which a consumer demands and and a list of initial endowment. The information provided in a $DEMAND block includes:

- Q: Reference quantity.
- R: Rationing instrument indicating an auxiliary variable.

Here is a typical demand function for a representative household in region r who holds exogenous endowments of labor and sector-specific capital:

```
        $DEMAND:RH(r)
        E:PL(r)       Q:le0(r)
        E:RK(r,s)     Q:ke0(r,s)
        D:PC(r)
```

If this representative consumer earns no tax revenue, the following value will be assigned to the consumer income variable:

```
RH(r) = PL(r) * le0(r) + sum(s, RK(r,s)*ke0(r,s));
```

Here is an example of a demand block for a consumer which holds exogenous endowment of labor and an endogenous endowment of foreign exchange, based on the value of the auxiliary variable BOPDEF

```
$DEMAND:HH(h)
E:PL        Q:labor(h)
E:PFX       Q:bopdef0(h)      R:BOPDEF(h)
D:PCON
```

In this example, assuming no tax revenue accrues to `HH(h)`, the value assigned to the consumer income variable is:

```
HH(h) = PL * labor(h) + PFX * BOPDEF(h) * bopdef0(h);
```

10.6.9 $CONSTRAINT Syntax

Auxiliary constraints in MPSGE models conform to standard GAMS equation syntax for complementarity problems. The may reference variables defined under any of the four classes, `$SECTORS`, `$COMMODITIES`, `$CONSUMERS` and `$AUXILIARY.`

Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated `POSITIVE` (the default), the auxiliary constraint should be expressed as a "greater or equal" inequality (`=G=`). If an auxiliary variable is designated `FREE`, the associated constraint must be expressed as an equality (`=E=`).

```
$CONSTRAINT:TAU
      G =G= X * Y;
```

It is important to remember that the equations associated with complementarity problems are *oriented*. In the default case, where an auxiliary variable has a lower bound of zero, then complementary slackness implies that whenever the left-hand side of the associated constraint exceeds the right-hand side, the variable value must be zero.

If you are unsure about how to orient a constraint for an auxiliary variable, think about how a constraint might become slack in case the variable is zero. Complementary slackness in the case of the central variables is quite clear: if equilibrium unit cost exceeds equilibrium unit revenue for a particular activity, its level must be zero. If equilibrium supply of a good exceeds demand, the associated market price must be zero. In the case of auxiliary variables, the complementarity idea may enter differently. Here are a few examples:

- Classical unemployment:

  ```
  $CONSTRAINT:U
  PL / PC =G= minwage;
  ```

  The real wage is rigid downward, then the unemployment rate must be zero if the real wage exceeds the mandated minimum wage. Given the way that complementarity constraints are defined in GAMS, the constraint shown above is **not** equivalent to the constraint:

  ```
  minwage =L= PL/PC;
  ```

- Income support payments: when a transfer payment to a particular industry is defined by a target level of sectoral sales

  ```
  $CONSTRAINT:TRNPAY
      Y =G= YLEVEL;
  ```

  When formulated in this way the transfer payment must be zero if the equilibrium output level exceeds the target.

11. Appendix 2 – Algebraic Form of the TWOBYTWO Model

For completeness, we show here two ways of writing down the mathematical structure of the TWOBYTWO model described in section 2.1.

Technology is described by production functions, $f_x(k_x, l_x)$ and $f_y(k_y, l_y)$, and preferences are described by a utility function, $U(c_x,c_y)$. (All of these functions are assumed to be linearly homogeneous, so $f_x(\lambda k_x, \lambda l_x) = \lambda f_x(k_x, l_x)$ etc.) The representative consumer in the model is endowed with capital, *K*, and labour, *L*, and she consumes good *C*, a commodity produced through the "production function" $U(c_x,c_y)$.

The model is based on profit-maximizing/cost-minimizing producers, and budget-constraint utility maximizing consumers. Technologies exhibit constant returns to scale, and all agents are price-takers. The equilibrium conditions for this model could be represented as:

- Market clearance for goods

$$f_x(k_x, l_x) = c_x$$

$$f_y(k_y, l_y) = c_y$$

- Market clearance for factors

$$K = k_x + k_y$$

$$L = l_x + l_y$$

- Optimization by producers who choose

$$l_i \text{ and } k_i \text{ to solve max } p_i\, f_i(k_i, l_i) - rk_i - wl_i \qquad i \in \{x, y\}$$

- Optimization by consumers who choose

$$c_x \text{ and } c_y \text{ to solve max } U(c_x,c_y) \text{ s.t. } p_x\, c_x + p_y\, c_y = wL + rK$$

In the MPSGE framework, however, individual preferences correspond to a single commodity, so the model can be represented with optimization appearing only on the production side of the economy. Optimization of utility enters the model through profit maximization within the *U()* sector. The equilibrium conditions are then represented by:

- Market clearance for goods

$$f_x(k_x, l_x) = c_x$$

$$f_y(k_y, l_y) = c_y$$

$$U(c_x,c_y) = M/p_c$$

- Market clearance for factors

$$K = k_x + k_y$$

$$L = l_x + l_y$$

- Optimization by producers

$$l_i \text{ and } k_i \text{ solve max } p_i\, f_i(k_i, l_i) - rk_i - wl_i \qquad i \in \{x, y\}$$

$$c_x \text{ and } c_y \text{ solve max } p_c\, U(c_x,c_y) - p_x\, c_x - p_y\, c_y$$

- Income balance by consumers

$$M = wL + rK$$

12. Appendix 3 – Changes from Earlier Versions of MGE2GP

We describe briefly changes between earlier versions of MGE2GP. If you are familiar with one of these earlier versions you should read the relevant parts of this section since that will probably be the most efficient way of finding out what you need to know in order to move to the current version of MGE2GP.

12.1 Changes from Version 1 (June 2004) to Version 1.1 (July 2004)

The values on the database are treated as values, not quantities. The updated data are balanced and so can be used as a starting point for simulations.

Many of the example simulations come with Command files which reverse the shocks. [See, for example, SJMGELBBACK.CMF in section 5.2.2.] These "back" simulations are good tests of the model and the updated data.

The MINIMAL model (see section 6.1) is now supplied with MGE2GP.

Tax rate and power variables corresponding to `t:` fields take their arguments from those in the MGE file. Ditto for `p:`, `m:` and `n:` fields. See section 3.2.7. [In Version 1, these variables had all possible arguments.]

Tax rates are now updated.

Coefficients which occur twice outside of an expression in q: fields but are full rank are now updated. [See section 6.4.3 of the June 2004 version of this document.]

Coefficients occurring with a negative sign (for example, `q:-bbop`) are now updated.

If there is an `r:` field in an `e:` line, simulations do not produce correct updated data if there is an expression beginning `"("` in the `q:` field. [But `q:-bbop` is ok.] See section 7.4.5.

No updated data is produced if some part of the data is not updatable. See section 7.4.

Exactly when updated data is valid is now documented. See section 7.4.

`$constraint:` equations can have shifters on them – see section 7.6.1.

`$constraint:` equations may be written in the TAB file with some MGE names replaced by TAB file names – see section 7.6.2.