Solving MPSGE Models Using GEMPACK

Laurent Cretegny[*], Mark Horridge, Ken Pearson

Centre of Policy Studies, Monash University

Thomas Rutherford

Department of Economics, University of Colorado

May 2004

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, defines 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. The full version of the paper will contain 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. The example models can be solved using the Demonstration Version of GEMPACK which can be downloaded (at no cost) from the web.

JEL classification : C63; C68; C88

Keywords : Modelling language; Computable General Equilibrium Models; Solving Economic Models

---

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

1. Introduction

MPSGE[1] 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, defines 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**[2]) 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).[3] 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 model 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. The example models can be solved using the Demonstration Version of GEMPACK which can be downloaded (at no cost) from the web – see section 1.3 for details.

The current version of MGE2GP is **Version 1, May 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 1.2) to obtain the latest version of the program and of the associated documentation.

You should be aware that this version of the software is aimed at pedagogical models, and we think that it should be useful for teaching applied general equilibrium modelling. However, we have not yet tested MGE2GP with large-scale models (for example, models with many zeros in the database). We expect that the current version will need further development before such models can be handled.

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.

---

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

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

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

- 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 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 be interested to learn about GEMPACK's capability for producing executable image versions of individual models. If you convert your MPSGE model to GEMPACK, you can produce a version of their model which 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. For example, section 2 introduces the MPSGE language syntax from scratch. It is primarily intended for readers (GEMPACK users) who have never used GAMS/MPSGE. This introduction goes through a number of simple models illustrating 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.

In section 4, we set out our plans for further work on this topic. References (including references to the complete GEMPACK user documentation) are given in section 5.

1.1 User Guide

We are preparing a user guide version of this paper. This will contain detailed hands-on instructions for solving the example MPSGE models we supply using GEMPACK. The material will be of particular interest to you if you have worked with GAMS/MPSGE and 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.

The user guide will also contains some advice about designing MPSGE models which are most suitable for conversion to GEMPACK. It will include 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 will also be documented in this user guide.

We expect to have this user guide version available around the middle of June 2004.

1.2 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.cretegyn.ch/mge2gp.htm

1.3 Downloading the Demonstration Version of GEMPACK

You will need a version of GEMPACK to carry out the simulations. 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.

2. General Equilibrium Modelling with MPSGE

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 data has 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 of the benchmark equilibrium. This replication 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 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 feeling about 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.

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 consumers. 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) are 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	

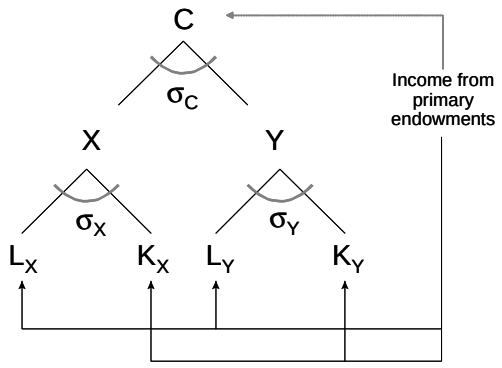
These data are presented above in a square social accounting matrix (SAM). The accounts labelled X and Y in this matrix refer to markets for final commodities. Account C corresponds to final consumption, an *activity* which transforms goods *X* and *Y* into a composite consumption good. The RA account 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.[4]

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:

[4] 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. A key advantage of the MPSGE approach is that it provides representations of both the equations of the model and of the calibration of parameters for the underlying functions. Here is the MPSGE model corresponding to the input data shown in these social accounts:

```
$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                ! consummer 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 MGE file starts off with variable declarations. In the `$commodities:`, `$sectors:` and `$consumers:` sections of the file variable names are introduced and classified according to how they

are to be interpreted in the model. Each `$sector:` in the model has an associated `$prod:` block in which the production sector inputs and outputs are described. Each `$consumer:` in the model has an associated `$demand:` block in which endowments and preferences are described. 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 vales – 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 first record of each `$prod:` block.

To be concrete, we will present the mathematical structure of the model represented by the preceding MPSGE model. In the 2 x 2 model technology is described 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

$$l_i \text{ and } k_i \text{ solve max } p_i f_i(k_i, l_i) - rk_i - wl_i \qquad i \in \{x, y\}$$

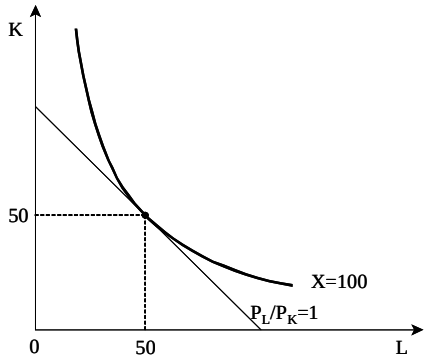
$$c_x \text{ and } c_y \text{ solve max } p_c U(c_x,c_y) - p_x c_x - p_y c_y$$

- Income balance by consumers

$$M = wL + rK$$

2.1.1 Calibration of TWOBYTWO

Perhaps the most important and analytically challenging 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 choices undertaken 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* with the *isocost line* 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 dashed line, representing combinations of K and L which have the same value as the benchmark quantity. The *calibration* of the function involves extrapolating from the benchmark point to the isoquant, and this is performed in the MPSGE model simply through the specification of an elasticity of substitution 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, you can jump ahead to section 3. 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,

- the benchmark data can be specified somewhere else in the program or indeed read from external files.
- you can use set notation when convenient, as in cases where there are many goods, factors, countries or consumers.

Combining these two features, data can be specified in a 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 production factors. 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[5] 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	qcomin				qhous	
LAB	1	3				
CAP	1	1				
fac	qfacin					
C						6
RA			4	2		
			Xfac			

You see immediately that the data may be grouped into different sub-matrices. These sub-matrices allow us to write down the MPSGE production and demand blocks in a more compact way. In this model, the production sector for output is as follows:

[5] See chapter 3 of Dixon *et al* (1992).

```
$prod:xcom(sect)    s:1.0
  o:pc(sect)
+ q:(sum(good, qcomin(good,sect)) + sum(fac, qfacin(fac,sect)))
  i:pc(good)        q:qcomin(good,sect)
  i:pf(fac)         q:qfacin(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`".[6]
>
> ```
> $prod:xcom("s1")    s:1.0
>   o:pc("s1")    q:(sum(good, qcomin(good,"s1")) + sum(fac,
> facin(fac,"s1")))
>   i:pc(good)        q:qcomin(good,"s1")
>   i:pf(fac)         q:qfacin(fac,"s1")
>
> $prod:xcom("s2")    s:1.0
>   o:pc("s2")    q:(sum(good, qcomin(good,"s2")) + sum(fac,
> facin(fac,"s2")))
>   i:pc(good)        q:qcomin(good,"s1")
>   i:pf(fac)         q:qfacin(fac,"s1")
> ```
>
> Suppose that the set `fac` contains the two elements `labour` and `capital`. Then the `i:` line in the `$prod:xcom("s1")` block above is a shorthand for the two `i:` lines :
>
> ```
>   i:pf("labour")    q:qfacin("labour","s1")
>   i:pf("capital")   q:qfacin("capital","s1")
> ```
>
> That is, there are two lots of factor inputs into the production of `xcom("s1")`, namely `qfacin("labour","s1")` of labour and `qfacin("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:qcomin("s1","s1")
>   i:pc("s2")        q:qcomin("s2","s1")
> ```
>
> There are two inputs of commodities into the production of `xcom("s1")`, namely `qcomin("s1","s1")` of commodity `pc("s1")` and `qcomin("s2","s1")` of commodity `pc("s2")`.

In GAMS/MPSGE the symbols `qcomin`, `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:` line is an expression, which represents the sum over all production factors used in each sector. Note also that continuation lines are indicated in MPSGE by a **`+`** in column 1. Thus the `o:` record above spreads over two lines of the file.

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:

---

[6] 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`.

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

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:xfac(fac)
```

2.2.2 Joint Production – 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`, in which you have the following production sector for output:

```
$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 previous 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.

You can see from the `o:` field that industry `s` produces several outputs `o`. The set for the commodity differs from the one for the sector. Moreover each industry may supply a different quantity for each good as revealed by the dimension in `supply(s,o)`. The curvature of the production possibilities frontier is given in the `t:` field on the `$prod:` line where you specify the constant elasticity of transformation (CET) between outputs.

Another additional feature is introduced here, namely the nesting notation. In the Stylized Johansen model, all production factors may be substituted between them. Here you can not 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. This composite good, which can be referred to value added, is then combined with intermediate goods. The specification of this two-level nesting structure is as follows. The top level nest is declared by the `s:` field on the `$prod:` line. The second level nest is declared by a label with four or few characters on the `$prod:` line and on the `i:` line corresponding to primary factors. You can think of the inputs denoted with `va:` as combined to produce a composite input called value added. In this specific example, the top level of the constant elasticity of substitution function specifies an elasticity of zero (by default when the `s:` field is not specified) between intermediate goods and the value added composite of capital and labour. The elasticity for combining primary factors is given by `esub(s)` and may differ across industries.

2.2.3 Small Open Economy – OPEN.MGE

A natural extension to 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” and which can then be used as input for producing imports. In `open.mge`, these blocks 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)
```

The production blocks show that world prices are fixed which characterize the small-country assumption. Furthermore we assume that both exports and imports are imperfect substitutes for domestic goods. This is represented by a 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:
```

with `etrn(s)` being the elasticity of transformation and by a 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)
```

where the elasticity of substitution is given by `esubm(o)`.

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

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

In this example, exports are larger than imports leading to a trade surplus. This benchmark surplus is then represented by a negative endowment, `-bbop`, of foreign exchange, `pfx`. In other words, the representative agent has initial foreign lending which finance the trade surplus.

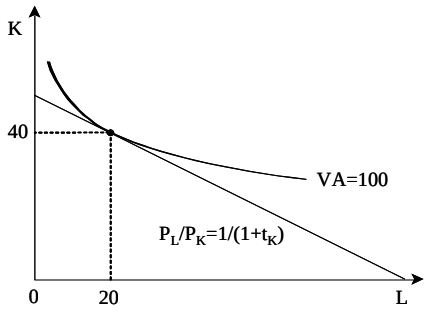
2.3 Adding Exogenous Taxes

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.

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

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

The following figure shows you how a production function is calibrated using these parameters. Benchmark quantities determine an anchor point with *L0* = 20 and *K0* = 40. Benchmark relative prices fix the slope of the isoquant at that point and the elasticity describes the curvature of the indifference curve.



Thought of in a formal way, benchmark quantities alone provide a zero order approximation of the underlying technology. Benchmark reference prices and reference quantities together provide a first order approximation and quantities, prices and elasticity parameters together provide a second-order approximation of the underlying technology.

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.

In this example benchmark taxes are only on capital used in the industrial sector. If you suppose that its consumer price is one, then the producer price is (1+*t*), where *t* is the ad-valorem rate of the tax. On the other hand, if you assume that the producer price is unity, then the consumer price is 1/(1+*t*). In this example, the tax rate is equal to 100% which implies a reference price for capital in the industrial sector equal to 2 when the consumer price is unity.

Input taxes in MPSGE are specified as follows. On a `i:` line, you indicate the recipient of the tax by the mean of a `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. 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 it is the case in this example.

```
$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:
```

When you have a pre-existing tax in the benchmark as it is the case here for the capital used in the industrial sector, you need to specify the reference price at the benchmark by the mean of `p:` field. This price is measured as a user cost which means that the reference price for production inputs equals one plus the benchmark ad-valorem tax rate.

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.

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. 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 `t:` field are specify as in the case of the 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 $p(1-t)$. You can think of a producer whose cost increases with the introduction of an input tax, whereas its value of outputs decreases with an output tax.

This convention of modelling taxes in MPSGE has 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.

2.3.3 Tariffs and Export Subsidies – TARIFF.MGE

Tariffs and export subsidies follow the same logic as taxes on inputs or outputs. Referring to the small open economy model, export subsidies are introduced in the export block while tariffs are launched 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.

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

where the prices pu, pf("lab") and pf("cap") are declared as commodities :

```
$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("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("lab")_b*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("lab")_s*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 MPSGE file is to be used with GAMS, this would be

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

If your MPSGE file is to be used with GEMPACK, this could be

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

At the benchmark, the quantity of labour supplied (and demanded) QS_RA_PFLAB_B[7] is equal to epl_b*endl_b, while the value of labour supplied, VS_RA_PFLAB_B, is equal to pf("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("lab")_s *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 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("lab")_b*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("lab")_b*endl_b in the initial benchmark and will be epl_s*pf("lab")_s*endl_b (assuming that endl_b stays fixed) post-simulation. This revenue depends not only on pf("lab") 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 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 detemined by the market clearing equation for labour (and that the price pc of commodities is determined by the market clearing equation for commodities).[8]

---

[7] 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.

[8] 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,

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 not 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 for accommodating changes in GDP. In our small-open economy model, 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 component. In this small model, GDP is composed of private consumption, government consumption and net exports. One way of getting out the MPSGE straight-jacket is to let the balance of payment adjusts 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 payment, 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.

---

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.

```
$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 in the beginning of the last section, 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 field `n:` gives the name of the auxiliary variable which will scale the tax rate. The field `m:` specifies the multiplier. If the `m:` field is omitted, the multiplier assumes a default value of unity. If the value in the `m:` field is zero, the tax does not apply. 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 the variable does not appear in the equation.

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, 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.

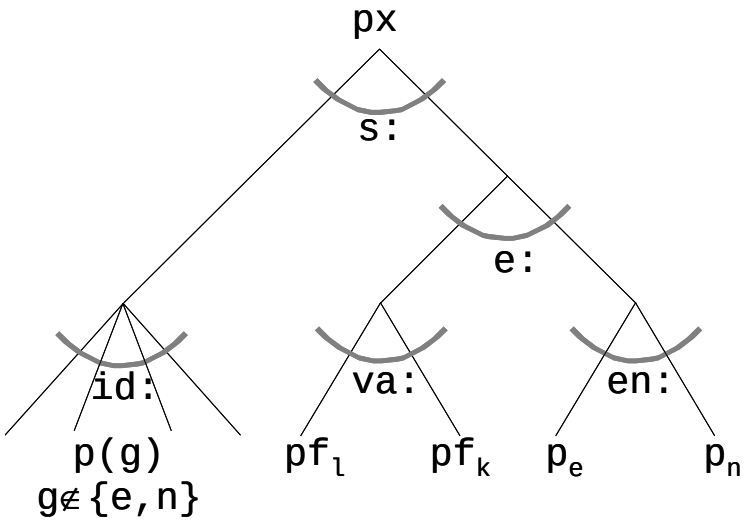
2.6.2 Sets of Nests

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` 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 may not be worth the trouble.

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 and a GEMPACK Command file MODELHSIM.CMF.
3. Use GEMPACK software to solve the model starting from MODEL.TAB, 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. Steps 1 and 3 are discussed in the user guide.

3.1 Converting an MPSGE 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 1.2.

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 and MODELHSIM.CMF in directory C:\MODEL. The first of these is the GEMPACK TABLO Input file for the model and the second 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 1.2), 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).

3.1.2 Nests

If your MGE file contains nests, the program MGE2GP automatically rewrites your MGE file by adding extra `$prod:` functions to remove the nests. The TAB 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 `$prod:` blocks to remove the nests. Then MGE2GP produces MODEL.TAB and MODELHSIM.CMF which are based on the rewritten MODEL_MGE2GP.MGE. However you can see the original `$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.

```
$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 `$prod:y(s)` function is broken into two `$prod:` functions, namely the new `$prod:y(s)` function and the additional `$prod:d_va_y(s)` function. 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 function.

```
$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)` function (as rewritten in JOINT_MGE2GP.MGE – see above) is shown as a strong comment before the TABLO statements implementing that function.

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 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 linearised 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 in levels value :

```
R_X = C_X ;
```

  The variables are `R_X` (total **R**evenue from sector **X**) and `C_X` (total **C**osts in sector **X**).

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 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.[9]]

  - 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 variables 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 loose 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.

3.2.4 Market Clearing Equations

For every commodity declared in the `$commodity:` block, there is an associated market clearing equation.

- When the commodity is in a `d:` line in a `$demand:` block (for example, `pc` in TWOBYTWO.MGE), this equation simply says that the value of the income for the agent in the `$demand:` block is equal to the market price of the commodity times the quantity supplied of this commodity.
- For other `$commodities`, this equation simply says that demand equals supply.

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

  - The commodity `pc` occurs in a `d:` line in the `$demand:ra` block. Hence the market clearing equation for `pc` is (in the levels):
    `VI_RA = PC_L * QS_PC ;`
    Here QS_PC stands for the **Q**uantity **S**upplied of commodity **PC**. In the TAB file produced by MGE2GP, this equation is written as the following linearised equation:
    `ra = pc + c`
    This says that the percentage change `ra` in the income of consumer `ra` is equal to the sum of the percentage changes in the price `pc` and the quantity `c` of the composite consumption good.

---

[9] 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. [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`.]

- The market clearing equation for `pl` is :
  `QD_X_PL + QD_Y_PL = QS_RA_PL`
  The variables on the left-hand side are `QD_X_PL` (**Q**uantity **D**emanded in production of **X** of commodity **PL**) and `QD_Y_PL` (ditto for the production of **Y**). The variable on the right-hand side is `QS_RA_PL` (**Q**uantity **S**upplied by consumer **RA**).

3.2.5 The Equations in the TAB File

As you probably know, TAB files in GEMPACK can contain a mixture of levels and linearised 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 0 above, the CES demand and CET supply functions are written as linearised equations in the TAB file.

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.

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.[10] 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.

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.

---

[10] 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.

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 #
     QD_X_PL +  QD_Y_PL = QS_RA_PL ;
```

while the market clearing equation for commodity `pc` is written as

```
! Market clearing for pc !
Equation (Linear) E_pc
  # Market clearing equation for commodity pc #
     ra = c + pc ;
```

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).

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.

4. 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. This means that these features are not allowed when using MGE2GP and will cause the program to stop with an error message. These include

- Larger models containing large numbers of zeros in the database,
- Endogenous taxes (see section 2.5),
- Exception handling (see section 2.6.1),
- Sets of nests (see section 2.6.2),
- Spanning operator (see section 2.6.3) and
- Explicit Complementarities (to handle models where activities are driven to zero).

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 MINIMAL, ORANIG and TERM.

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 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.

5. 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 equilibruim 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.