

## MPS/GE の利用方法

(2011/09/02)

武田史郎

## 概要

MPS/GE (mathematical programming system for general equilibrium) の利用方法の解説.

## 内容

1. 準備 .....	2
1.1. 本書を読む (MPSGE を利用する) ための準備 .....	2
1.2. MPSGE についての文書 .....	3
2. MPSGE における変数のタイプ .....	3
2.1. 各変数の説明 .....	4
2.2. MPSGE における変数と均衡条件の対応 .....	5
2.3. 変数宣言の例 (ex_decl_01.gms の例) .....	5
3. SAM の説明 .....	6
3.1. 表 2 の説明 .....	7
3.2. 例 2 (ex_prog_08.gms のデータ) .....	8
4. \$prod ブロックの説明 .....	11
4.1. 1 段階の CES 生産関数 (ex_prod_01.gms) .....	11
4.2. 2 段階の CES 生産関数 (ex_prod_02.gms) .....	13
4.3. 3 段階の CES 生産関数 (ex_prod_03.gms) .....	15
4.4. 生産物が複数ある (結合生産がある) ケース (ex_prod_04.gms) .....	16
4.5. 投入物に税金をかけるケース (ex_prod_05.gms) .....	18
4.6. ベンチマークにおいて投入物に対する税金が存在するケース (ex_prod_06.gms)	
19	
4.7. 生産物に対する税が存在するケース (ex_prod_07.gms) .....	20
4.8. 政府部門が存在するケース (ex_prod_08.gms) .....	21
5. MPSGE の内部動作 .....	23
5.1. Step 1 .....	24
5.2. Step 2 .....	24
5.3. Step 3 .....	24
5.4. 数式による表現 .....	24
5.5. 参照価格 (reference price) についての注 .....	25

6.	\$demand ブロックの説明.....	26
7.	\$report 命令の利用方法.....	27
8.	補助変数 (auxiliary variable) の利用方法.....	29
8.1.	\$auxiliary 命令と\$constraint 命令.....	29
8.2.	利用例 1 (内生的に変化する一括税) .....	30
8.3.	利用例 2 (内生的に変化する労働課税) .....	33
9.	MPSGE でのモデルの実行方法.....	35
9.1.	変数の初期値.....	35
9.2.	「アクティビティー変数」の値 .....	35
9.3.	モデルの解き方.....	35
9.3.1.	モデルを解くコード.....	35
9.3.2.	numeraire (ニューメレール) について.....	36
9.4.	結果の見方.....	36
9.4.1.	SOLVE SUMMARY ブロック .....	36
9.4.2.	内生変数の値.....	37
9.4.3.	MPSGE における MARGINAL 値の意味.....	38
10.	MPSGE のシンタックス (まとめ) .....	39
10.1.	MPSGE 特有の命令 .....	39
10.2.	変数の宣言部分 .....	40
10.3.	\$prod ブロック.....	41
10.4.	\$demand ブロック.....	42
10.5.	\$constraint ブロック.....	43
11.	MPSGE でのデバッグ.....	43
11.1.	Benchmark replication チェック.....	44
11.2.	Cleanup calculation によるチェック. ....	47
11.3.	スケール・ショックによるチェック. ....	48
11.4.	MPSGE のデバッグオプションによるチェック. ....	50
12.	モデル例.....	52
	参考文献.....	53

## 1. 準備

### 1.1. 本書を読む (MPSGEを利用する) ための準備

本稿を読む前に以下の3つの文書を読む必要がある。

- MPSGEではCES型関数の利用を前提としている。よって、CES型関数について深く把握しておく必要がある。CES型生産関数、それから導出される費用関数、条件付き要

素需要関数等については武田（2009b）を参照。

- MPSGEでは「双対アプローチ」を利用して一般均衡モデルを記述するので「双対アプローチ」について理解しておく必要がある。双対アプローチによる一般均衡モデルの記述については武田（2008）を参照。
- MPSGEは（その内部で）CES関数をcalibrated share formによって表している。よって、CES関数のcalibrated share formについて理解しておく必要がある。これについては武田（2009a）を参照。
- 3つの文書は <http://shirotaakeda.org/home-ja/research-ja/note-ja.html> にある。
- また、当然のことであるが、CGE分析をするにはミクロ経済学、及び一般均衡モデルの基本的性質について理解をしておく必要がある。

## 1.2. MPSGEについての文書

この文書以外に MPSGE を学ぶ際に参考になる文書

- MPSGE Guide
  - 公式のMPSGEマニュアル。
  - <http://www.gams.com/dd/docs/solvers/mpsge.pdf>
  - 一応、MPSGEの利用法を一通り説明しているが、少しわかりづらいかもしれない。
- Markusenのレクチャーノート
  - <http://spot.colorado.edu/~markusen/teaching.html>
  - 様々なタイプのモデルのサンプルコードがあり、非常に参考になる。
- Rutherfordによるサンプルのコード
  - MPSGEの開発者であるThomas F. Rutherfordのウェブサイトには多数のMPSGEを利用したコードが置いてある。
  - <http://www.mpsge.org/>

## 2. MPSGEにおける変数のタイプ

まず、MPSGE でモデルを記述する際に利用する変数のタイプについて説明する。変数のタイプには「アクティビティー変数 (activity variable)」、「価格変数 (price variable)」、「所得変数 (income variable)」、「補助変数 (auxiliary variable)」の4つがある。

変数のタイプ	宣言する命令	対応	関連する条件
アクティビティー変数 (生産量)	\$sector	sector (部門)	利潤最大化(ゼロ利潤条件)
価格変数	\$commodity	commodity (財)	市場均衡条件

所得変数	\$consumer	consumer (消費者)	予算制約条件
補助変数	\$auxiliary		制約式

## 2.1. 各変数の説明

- 「アクティビティー変数」, 「価格変数」, 「所得変数」はそれぞれ「sector」, 「commodity」, 「consumer」に対応している.
- アクティビティー変数
  - 一つのアクティビティー変数は一つのsector (部門) に対応する. これはそのsectorの活動水準を表す.
  - そのsectorが一つしか財を生産していないのなら, アクティビティーレベル=生産水準である.
  - 普通モデルを記述する際には消費と生産活動は別のタイプの活動として捉えられる. しかし, 「消費=財を投入して効用という財を生産する活動」とみなせる. この考え方に立ち, MPSGEでは消費を生産活動の一種と扱うことが多い. 以下でもこの考え方を採用する.
- 価格変数
  - 一つの価格変数は一つのcommodity (財) に対応する.
  - commodityには通常の財に加え, 生産要素も含む. 生産要素の場合には価格は要素価格となる.
- 所得変数
  - 一つの所得変数は一人のconsumer (消費者) に対応する.
  - consumerといっても通常の消費者・家計だけではない. モデルに政府を含める場合には政府もconsumerとして扱う.
  - 企業 (生産部門) については通常所得変数はない. これは, MPSGEでは「完全競争」+「収穫一定の技術 (CES生産関数)」を前提とするため, 完全分配により企業の利潤はゼロ (収入は全て投入物に分配される) となるからである.
- 補助変数 (auxiliary variable) については少し特殊であるので, また後に説明する.

表 1 : MPSGE における変数と均衡条件の対応関係

条件	式	対応する変数
利潤最大化条件 (ゼロ利潤条件)	$c_i(p_1, \dots, p_m) = p_i$	$\Rightarrow$ アクティビティー変数 $\{y_i\}$
市場均衡条件	$s_j(p_1, \dots, p_m, y_1, \dots, y_m)$ $= d_j(p_1, \dots, p_m, y_1, \dots, y_m, k)$	$\Rightarrow$ 価格変数 $\{p_j\}$
予算制約条件	$k = \sum_j p_j \bar{y}_j$	$\Rightarrow$ 所得変数 $\{k\}$

## 2.2. MPSGEにおける変数と均衡条件の対応

- MPSGEでは明示的にモデルの均衡条件を記述する必要はない。しかし、変数と均衡条件は次のように対応している（表 1参照）。
  - アクティビティ変数      ⇔      利潤最大化条件（ゼロ利潤条件）
  - 価格変数                    ⇔      市場均衡条件
  - 所得変数                    ⇔      予算制約条件
- つまり,
  - MPSGEでは均衡条件は「利潤最大化条件（ゼロ利潤条件）」, 「市場均衡条件」, 「予算制約条件」の3タイプに集約される。
  - 「利潤最大化条件⇒アクティビティ変数の決定」, 「市場均衡条件⇒価格変数の決定」, 「予算制約条件⇒所得変数の決定」という関係がある。
- 3つのタイプ以外の変数, 例えば投入量, 需要量といった変数はモデルには直接は出てこない。しかし, 標準では現れない変数も\$report命令によって定義・表示可能な場合がある。

## 2.3. 変数宣言の例（ex\_decl\_01.gmsの例）

ex\_decl\_01.gms というサンプル・プログラムの変数宣言部分.

```

*      アクティビティ変数 (sector) の宣言
$sectors:
  x      ! x 部門の生産量
  y      ! y 部門の生産量
  u      ! u 部門の生産量 (効用水準)

*      価格変数 (commodity) の宣言
$commodities:
  px     ! x 財の価格
  py     ! y 財の価格
  pu     ! u 財の価格 (効用の価格)
  pk     ! 労働の価格
  pl     ! 資本の価格

*      所得変数 (consumer) の宣言
$consumers:
  cons   ! 消費者の所得

```

コードの説明：

- 3つのタイプの変数はそれぞれ「\$sectors:」, 「\$commodities:」, 「\$consumers:」という命令によって宣言される.
- \$sectors
  - アクティビティ変数は「\$sectors」命令で宣言される.
  - 例では3つの変数 (sector) が宣言されている.
  - 宣言は一つの行で一つの変数に対して行う.
  - 「!」より後に変数の説明文を書く. この説明文はlistingファイルで変数の説明文として利用される.
  - 消費活動を生産活動の一種とみなしている. このため「効用水準」を表す「u」がアクティビティ変数として宣言されている.
- \$commodities
  - 価格変数は「\$commodities」命令で宣言.
  - 3つのsectorが生産する財 (x財, y財, u財) の価格以外に生産要素 (資本, 労働) の価格も宣言されている.
- \$consumers
  - 例ではconsという所得変数を持つ一人の消費者のみ宣言している.

### 3. SAM (Social Accounting Matrix) の説明

MPSGE の利用方法(シンタックス)をさらに詳しく説明する前に, データの説明を行う. CGE 分析ではベンチマーク・データを用意し, その「ベンチマーク・データにおいてモデルが均衡している」という前提で分析を行う. ベンチマーク・データの表現方法はモデルの表現方法と密接な関係があるので, まずベンチマーク・データをどう表現するかを説明する.

ベンチマーク・データの表現方法には様々な方法があるが, 以下では表 2 のような行列を用いる.

表 2 : ベンチマーク・データの例

		sector (部門)			consumer (消費者)	行和
		x	y	u	cons	
commodity (財)	px	100		-100		0
	py		100	-100		0
	pu			200	-200	0
	pk	-25	-75		100	0

	pl	-75	-25		100	0
	列和	0	0	0	0	

以下では上の表を SAM (social accounting matrix, 社会会計表) と呼ぶが、これは本来の SAM とは少し異なっている。本来の SAM は、①要素は全てプラス、②スクウェア (正方形)、③列和と行和が等しいという形で表現される。表 2 は MPSGE におけるモデルの記述形式と対応するように SAM を表現しなおしたものである。本来の SAM については細江他 (2004) を参照されたい。

### 3.1. 表 2 の説明

表 2 の見方：

- 表 2 は実際に ex\_decl\_01.gms で利用しているベンチマーク・データである。
- 均衡を仮定する CGE 分析で利用するデータであるので、データ自体が均衡条件を満たすように作成されていなければならない。MPSGE での均衡条件とは以下の 3 つ
  - ゼロ利潤条件 (収入 = 費用)
  - 予算制約条件 (所得 = 支出)
  - 市場均衡条件 (供給 = 需要)
 上の SAM はこの均衡条件にきれいに対応するように作成されている。
- 行・列の意味
  - 表 2 は MPSGE のモデル記述方法に対応する 3 つの要素から構成されている。
  - 各行 ⇒ 「commodity (財)」を表現する。
  - 各列 ⇒ 「sector (部門)」, あるいは「consumer (消費者)」を表現する。
  - commodity, sector, consumer は MPSGE における変数の宣言と対応関係を持つ。
- セルの値の意味。
  - プラスの値 ⇒ 生産額, 供給額, 受け取り額を表す。
  - マイナスの値 ⇒ 需要額, 購入額, 支払い額を表す。
- 横方向
  - 横方向に見た場合には、連関表と同じで「commodity」を表す。
  - 例えば、1 行目は px というラベルが付いている。これは px という価格を持つ財 (x 財) の市場を表す。
  - 同様に、pl という行は、pl という価格で表現される財 (つまり、労働) の市場を表す。
  - 市場均衡の仮定より「供給 = 需要」が成立するため行和は常にゼロとなる。
- 縦方向
  - 縦方向では二種類に分かれる。一つが「sector」、もう一つは「consumer」。
  - sector

- これは財を生産する部門の収入, 費用, つまり各部門が何をどれだけ投入し, 何をどれだけ生産したかを表す. プラスが収入項目, マイナスが費用項目.
- ゼロ利潤条件より「収入=費用」が成り立ち列和はゼロとなる.
- consumer
  - これは消費者の所得の源泉と所得の使い道を表現する列である.
  - 「支出=所得」の関係より列和はゼロとなる.
- 消費活動 (u部門)
  - sectorの一つとして, uという部門がある. これは消費活動を表す部門である.
  - 通常, モデルを記述する際には, 生産側と消費側で異なった記述方法をとる. しかし, 消費活動は「財を投入して効用という財を生産する活動」とみなせる. そのような見方に立ち, ここでは消費活動を一種の生産活動として扱っている. MPSGEでも基本的にそのように扱う.

表 3: 表 2 の数値の意味

タイプ	行・列	意味
sector	xの列	x部門は資本, 労働をそれぞれ25, 75ずつ投入し, x財を100生産している.
	yの列	y部門は資本, 労働をそれぞれ75, 25ずつ投入し, y財を100生産している.
	uの列	u部門は消費活動を表す部門. 消費活動は各財を投入 (消費) することで効用を生産する活動とみなせる. x財, y財をそれぞれ100ずつ投入し, u財 (効用) を200生産している.
consumer	consの列	これはconsumerを表す列. 消費者は生産活動に資本, 労働を供給することでそれぞれ100ずつの所得を得て, それをu財 (効用) を購入するために支出している.
commodity	pxの行	x財はx部門によって100供給され, u部門によって100需要されている.
	pyの行	y財はy部門によって100供給され, u部門によって100需要されている.
	puの行	u財 (効用) はu部門によって200供給され, 消費者によって200需要されている.
	pkの行	資本は消費者によって100供給され, x部門が25, y部門が75需要している.
	plの行	労働は消費者によって100供給され, x部門が75, y部門が25需要している.

## 3.2. 例2 (ex\_prog\_08.gmsのデータ)

表 4 は ex\_prog\_08.gms で利用しているデータ.

表 4 : 政府と税金が加わったデータ (ex\_prog\_08.gms のデータ)

		sector (部門)				consumer (消費者)		行和
		x	y	u	g	cons	gov	
commodity (財)	px	125		-115	-10			0
	py		105	-85	-20			0
	pu			200		-200		0
	pk	-25	-75			100		0
	pl	-75	-25			100		0
	pg				30		-30	0
税	vtl	-15	-5				20	0
	vtx	-10					10	0
列和		0	0	0	0	0	0	

- 表 2 と異なるのは政府と税金が加えられている部分. 税の導入により, 縦方向に税のフローを表す行が加わる.
- 政府
  - 政府は cons と同様に一種の consumer (消費者) として表現されており, gov 列に政府の所得と支出が計上されている. 政府は税金を徴収し, それを政府支出 (「政府支出財」への支出) に利用する.
  - 政府についての「所得 = 支出」という関係から, gov 列の列和はゼロとなる.
- 政府支出財部門
  - 政府支出は x 財と y 財から構成される.
  - x 財と y 財がそれぞれ 10, 20 ずつ投入され, 「政府支出財 (その価格は pg)」という財が生産されるという仮定.
  - 政府支出はこの「政府支出財」の購入という形でモデル化.
- 政府支出財
  - 政府支出財の市場は pg 行で表現されている.
  - 政府支出財は, 政府支出財部門 (g 部門) によって 30 供給され, 政府が 30 購入する.
- 税金
  - vtl と vtx は commodity ではなく, 税を表す行.
  - vtl と vtx がそれぞれ労働と x 財に対する税を表している.
  - 例では, x 部門が労働と x 財に対する税をそれぞれ 15, 10 だけ支払い, y 部門が労働

に対する税を5支払っている。

- 税収 (30) は全て政府の所得になる。

表 5：表 4 の数値の意味

タイプ	行・列	意味
sector	xの列	x部門は資本、労働をそれぞれ25, 75ずつ投入し、労働に対し15, 生産物に対し10の税を支払い, x財を100生産している。
	yの列	y部門は資本、労働をそれぞれ75, 25ずつ投入し、労働に対し5の税を支払い, y財を105生産している。
	uの列	u部門は消費活動を表す部門。x財, y財をそれぞれ115, 85投入し, u財 (効用) を200生産している。
	gの列	g部門は「政府支出財」の生産部門。x財を10, y財を20投入し, 30の政府支出財を生産している。
consumer	consの列	これは民間の消費者を表す列。消費者は生産活動に資本、労働を供給することでそれぞれ100ずつの所得を得て, それをu財 (効用) を購入するために支出している。
	govの列	これは政府の所得・支出を表す列。政府は労働からの税を20, x財からの税を10受け取り, それを政府支出財の購入に支出。
commodity	pxの行	x財はx部門によって125供給され, u部門によって115, g部門によって10購入されている。
	pyの行	y財はy部門によって105供給され, u部門によって85, g部門によって20購入されている。
	puの行	u財 (効用) はu部門によって200供給され, 消費者によって200購入されている。
	pkの行	資本は消費者によって100供給され, x部門が25, y部門が75購入している。
	plの行	労働は消費者によって100供給され, x部門が75, y部門が25購入している。
	pg	これは政府支出財の市場を表す行。政府支出財はg部門によって30供給され, 政府が30購入している。
税	vtl	これは労働に対する税を表す行。x部門が25, y部門が5の労働に対する税を支払い, 政府がそれを受け取っている。
	vtx	これはx財に対する税を表す行。x部門が10支払い, 政府がそれを受け取っている。

以上, SAMの例を二つ挙げたが, 部門, 消費者, 市場, 税金がどれだけ増えようが後はパターンは同じである。

## 4. \$prodブロックの説明

## 4.1. 1段階のCES生産関数 (ex\_prod\_01.gms)

ex\_prod\_01.gmsを例にとる. ex\_prog\_01.gmsにおけるモデルの概要

- 閉鎖経済, 完全競争モデル.
- 二つの生産部門 (x部門, y部門).
- 規模に関して収穫一定の技術の下, x部門はx財, y部門はy財を生産する.
- 生産関数は資本, 労働という二つの生産要素のCES型関数.
- 生産要素は消費者が所有する. 生産要素の賦存量は外生的に一定.
- 効用関数はx財, y財のCES型関数.
- 税金なし, 政府部門なし.
- 数式での表現は表 6.

表 6 : ex\_prod\_01.gmsのモデルの均衡条件

タイプ	説明	数式	変数
利潤最大化条件 (ゼロ利潤条件)	x部門の利潤最大化条件	$c_x(r, w) = p_x$	$x$
	y部門の利潤最大化条件	$c_y(r, w) = p_y$	$y$
	u部門の利潤最大化条件	$c_u(p_x, p_y) = p_u$	$u$
市場均衡条件	資本市場の均衡条件	$\bar{k} = \frac{\partial c_x(r, w)}{\partial r} x + \frac{\partial c_y(r, w)}{\partial r} y$	$r$
	労働市場の均衡条件	$\bar{l} = \frac{\partial c_x(r, w)}{\partial w} x + \frac{\partial c_y(r, w)}{\partial w} y$	$w$
	x財市場の均衡条件	$x = \frac{\partial c_u(p_x, p_y)}{\partial p_x} u$	$p_x$
	y財市場の均衡条件	$y = \frac{\partial c_u(p_x, p_y)}{\partial p_y} u$	$p_y$
	u財市場の均衡条件	$u = cons/p_u$	$p_u$
予算制約	予算制約条件	$cons = r\bar{k} + w\bar{l}$	$cons$

## 表 6の説明

- $w$  と  $r$  はそれぞれ労働と資本の価格.  $c_x(r, w)$  と  $c_y(r, w)$  はそれぞれx部門とy部門の単位費用関数. 規模に関して収穫一定 (constant returns to scale, CRTS) の技術を仮定しているため, 技術は単位費用関数で表現できる.
- $p_x$  と  $p_y$  はそれぞれx財とy財の価格.

- $c_u(p_x, p_y)$ は効用生産活動の単位費用（普通の言い方では単位支出関数）． $p_u$ は効用の価格．
- Shephardの補題より，単位費用関数を投入物価格で微分したものが単位需要関数となる．
- $u = cons/p_u$ は効用（という財の）市場の均衡条件． $u$ が効用の供給量， $cons/p_u$ が効用の需要量を表す．

以下は， $x$ 部門の生産関数をツリーで表現したもの．

```
*      部門 x の生産関数
      x
      / \ <- sig_x
      /   \
      /     \
      k       l
```

生産関数の特徴

- 投入物は資本，労働のみ．
- 資本と労働の代替の弾力性は $sig\_x$ ．

生産関数を定義するMPSGEのコード

```
*      部門 x の生産関数
$prod:x  s:sig_x
  o:px    q:x0
  i:pk    q:kx0  p:pk0
  i:pl    q:lx0  p:pl0
```

コードの説明．

- $x$ は $x$ 部門のアクティビティーレベル（＝生産量）， $pl$ は労働の価格， $pk$ は資本の価格．
- “0”付きのパラメータは全てベンチマークの値．
  - 例えば， $x0$ は $x$ のベンチマーク・データでの値．
  - $kx0$ と $lx0$ はそれぞれ $x$ 部門における資本と労働のベンチマーク投入量．
- まず， $\$prod:$ の後で，アクティビティー変数を指定する．これによりどの部門の生産関数を定義するかを指定する．
  - $\$prod:x$ は $x$ 部門の生産関数を指定しているということ．
- $s:$ は代替の弾力性の指定のための記号．
  - $s:$ の後に代替の弾力性を指定する．この例では $sig\_x$ というパラメータを指定している．CES 関数のトップレベルでの代替の弾力性は常に $s:$ という記号で指定する．

- 代替の弾力性を指定しない (s:を指定しない) と, デフォルト値である0 が指定される. つまり, デフォルトではLeontief 型生産関数となる.
- \$prod:ブロック内では次の3タイプの行
  - o:で始まる行 → これはその部門の生産物を指定する行 (生産物指定行)
  - i:で始まる行 → これはその部門の投入物を指定する行 (投入物指定行)
  - +で始まる行 → これは一行前の続きを表す.
- 生産物指定行.
  - o:で始まる行では生産物を指定する.
  - どの生産物かは「価格変数」で指定する.
  - 例ではo:pxと指定している. これは, この部門がpxという価格を持つcommodity (つまり, x財) を生産しているということを意味する.
  - その後, q:というフィールドで「参照数量 (reference quantity)」を指定する. 生産物指定行での「参照数量」にはベンチマークにおける生産量を指定しておけばよい.
  - 例では生産物指定行は一行のみ. これはこの部門が一種類の生産物しか生産していないということを意味する. 多数の財を結合生産しているケースでは, o:行が複数現れることになる.
- 投入物指定行
  - i:で始まる行では投入物を指定する. i:の後に投入物の「価格変数」を指定する.
  - 例では, pkとplが指定された二行だけ. これは投入物が資本, 労働の二つだけということの意味する.
  - 生産物指定行と同様に, q:には「参照数量」を入れる. 投入物指定行における参照数量にはベンチマークにおける投入量を入れればよい.
  - q:の後でp:によって「参照価格 (reference price)」を指定する.
  - 「参照価格」には投入物のベンチマークにおける価格を指定する.
  - 「参照価格」のデフォルト値は1. よって, もし参照価格が1ならp:フィールドは省略してよい. サンプルex\_prod\_01.gmsではpk0, pl0はともに1であるので以下のように書いても同じ.

```
*      部門 x の生産関数
$prod:x  s:sig_x
      o:px    q:x0
      i:pk    q:kx0
      i:pl    q:lx0
```

#### 4.2. 2 段階のCES生産関数 (ex\_prod\_02.gms)

今度はx部門の生産関数が2段階のCES型関数であるケース.

```

*      部門 x の生産関数
      X
      / \ <- sig_xx
      /   \
      /     \
y      / \ <- sig_x
      /   \
      /     \
      k     l

```

生産関数の説明：

- 二段階のCES型関数.
- 投入物は中間財 (y財) , 資本, 労働.
- トップレベルの代替の弾力性はsig\_xx, 資本・労働の間の代替の弾力性はsig\_x.

生産関数を定義するMPSGEのコード：

```

*      部門 x の生産関数
$prod:x s:sig_xx va:sig_x
o:px    q:x0
i:pk    q:kx0 p:pk0 va:
i:pl    q:lx0 p:pl0 va:
i:py    q:yx0 p:py0

```

コードの説明：

- ex\_prod\_01.gmsとの違いは中間財の投入の指定と弾力性の指定の部分.
- i:pyが中間財 (y財) の投入を指定している行. q;, p:については資本, 労働と同じようにベンチマークの投入量, 価格を指定しておく.
- 代替の弾力性の指定方法
  - この例ではトップレベルのネストの弾力性 (中間財と生産要素の弾力性) はsig\_xx, 資本・労働の弾力性はsig\_xというパラメータで与えられる.
  - まず, トップレベルの弾力性についてはex\_prod\_01.gmsと同様にs:の後に指定する.
  - 資本と労働の弾力性はva:の後に指定し, さらにそのva:が適用される (va:のネストに入ってくる) 投入物の行にva:という記号を加える.
  - トップレベルの弾力性は必ずs:で指定するが, それ以外の弾力性については「4文字以内」という条件を満たすのならどのような記号で表現してもよい. 例えば, 以下のように指定しても全く同じ.

```

*      部門 x の生産関数

```

```

$prod:x s:sig_xx kl:sig_x
  o:px  q:x0
  i:pk  q:kx0 p:pk0 kl:
  i:pl  q:lx0 p:pl0 kl:
  i:py  q:yx0 p:py0

```

- トップレベルの弾力性の記号sについては投入物の行に加えない。逆に言えば弾力性指定の記号がない行の投入物はトップレベルのネストに入ってくるということになる。

#### 4.3. 3 段階のCES生産関数 (ex\_prod\_03.gms)

x 部門の生産関数が 3 段階の CES 型関数であるケース。

```

*      部門 x の生産関数
      X
      / \ <- sig_xx
      /  \
      /    \
y      / \ <- sig_x
      /  \
      /    \
l      / \ <- sig_kr
      /  \
      /    \
      k      r

```

生産関数の説明：

- 3段階のCES型関数.
- 投入物は、中間財 (y財) , 資本, 労働, 天然資源 (r) .
- 代替の弾力性
  - トップレベルの代替の弾力性はsig\_xx,
  - 労働と資本・天然資源の間の代替の弾力性はsig\_x,
  - 資本と天然資源の間の代替の弾力性はsig\_kr

生産関数を定義するMPSGEのコード：

```

*      部門 x の生産関数
$prod:x s:sig_xx va:sig_x kr(va):sig_kr

```

```

o:px    q:x0
i:pk    q:kx0 p:pk0  kr:
i:pl    q:lx0 p:pl0  va:
i:py    q:yx0 p:py0
i:pr    q:rx0 p:pr0  kr:

```

コードの説明：

- **kr(va):**で資本・資源のネストの代替の弾力性を指定している。
- (va)の部分は上位のネストを表している。つまり、krはvaというネストの下にあるということの意味する。
- va:のように上位のネストの指定がない場合は、トップレベルの下ネスト（つまり、二段階目のネスト）ということになる。
- 仮に、単に「kr:sig\_kr」と指定したとすると、以下のような生産関数を指定していることになる。

```

*      部門 x の生産関数
      x
      /\  <- sig_xx
     / | \
    /  |  \
   y  1 / \ <- sig_kr
      /   \
     /     \
    k       r

```

#### 4.4. 生産物が複数ある（結合生産がある）ケース（ex\_prod\_04.gms）

一つの部門が複数の生産物を生産する（結合生産がある）ケース。

```

*      部門 x の生産関数
      xx      xy
      \      /
     \      /
    \ / <- eta_x
     |
    / \ <- sig_x
   /   \
  /     \

```

k	1
---	---

生産関数の説明：

- 投入側はex\_prod\_01.gmsと同じ.
- 産出側
  - x部門がx財だけではなく、y財も生産する.
  - xxがxの生産量、xyがyの生産量.
  - xxとxyの配分はCET（constant elasticity of transformation, 変形の弾力性一定）関数に従っておこなわれる.
  - xxとxyの間のEOT（elasticity of transformation）はeta\_x.

生産関数を定義するMPSGEのコード：

```
*      部門 x の生産関数
$prod:x  s:sig_x  t:eta_x
  o:px    q:xx0   p:px0
  o:py    q:xy0   p:py0
  i:pk    q:kx0   p:pk0
  i:pl    q:lx0   p:pl0
```

コードの説明：

- 投入側はex\_prod\_01.gmsと同じ.
- 二つの生産物があるので、生産物指定行（o:で始まる行）が二つとなる。o:pxはx財、o:pyがy財を指定.
- q:にはこれまでと同様にベンチマークの生産量を与える.
- 生産物指定行の参照価格.
  - これまで生産物側ではp:フィールド（参照価格）を設定してこなかった.
  - これは1生産物のケースでは「参照価格」が意味を持たないため、これは投入物でも同じ。1投入物のケースでは「参照価格」は意味を持たないため、p:フィールドを指定する意味がない.
  - 複数の生産物があるケースでは「参照価格」を指定する必要がある.
  - 「参照価格」にはベンチマークの生産物価格（px0とpy0）を指定しておく.
  - 生産物の「参照価格」についてもデフォルト値は1. よって、1なら省略可.
- 生産物間の変形の弾力性（elasticity of transformation, EOT）の指定.
  - 生産物はCET 関数（constant elasticity of transformation, 変形の弾力性一定型関数）に従って配分が決まる.
  - xをx部門の生産水準、xxをx部門のx財の生産量、xyをx部門のy財の生産量とすると、CET関数に従って配分されるとは以下の関係を持つこと.

$$x = \left[ \alpha^x (xx)^{\frac{\eta+1}{\eta}} + \alpha^y (xy)^{\frac{\eta+1}{\eta}} \right]^{\frac{\eta}{\eta+1}}$$

ただし、 $\eta$ はxxとxyとの間の変形の弾力性である。 $\eta = \infty$ のとき二つの財は完全代替ということ。

- EOTは「t:」という記号で指定する。上の例ではEOTにeta\_xを指定している。
- 生産物が複数あるケースでは変数xは生産量ではなく、生産水準（アクティビティーレベル）を表す。x部門による各財の生産量（x財とy財の生産量）の数値を得るには\$report命令で生産量を表現する変数を定義して表示させればよい。ex\_prod\_04.gmsでは\$reportを利用してx部門の生産量を表示している。

#### 4.5. 投入物に税金をかけるケース (ex\_prod\_05.gms)

- x部門の資本、労働投入に税金をかけるケース。
- ただし、ベンチマークでは税金はないものとする。
- 投入に対する税は従価税として扱う。
- 税収は直接消費者にlump-sumで還元される。
- 税金以外の部分はex\_prod\_01.gmsと全く同じ。

生産関数を定義するMPSGEのコード：

```
*      部門 x の生産関数
$prod:x  s:sig_x
  o:px      q:x0
  i:pk      q:kx0  p:pk0  a:cons t:tkx
  i:pl      q:lx0  p:pl0  a:cons t:tlx
```

コードの説明：

- 税金がかかる投入物の指定行に「a:」と「t:」という二つのフィールドが加わる。
- a:フィールド
  - a:は税収の行き先（tax agent）を指定するフィールド。
  - 税収が誰の所得となるかを指定するので、「所得変数」により指定する。
  - このモデルでは経済主体としては企業以外には消費者しか考えていない。従って、税収の行き先はそのまま消費者となり、消費者の所得consを指定する。
- t:フィールド
  - t:は税率（従価税率）を指定するフィールド。
  - 税率はそれぞれtkxとtlxというパラメータで表現されるので、この二つが指定されている。
  - 税率は「net base」での税率、つまり「(1+税率)×市場価格」という形式での税率。

- この例ではベンチマークにおいて税はかかっていないと仮定されている。つまり、ベンチマーク均衡では「 $tkx = tlx = 0$ 」と設定されている。
- 例えば、 $x$ 部門の労働投入に30%の税金をかけるというシミュレーションをおこなうなら、以下のような形でモデルを解けばよい。

```
*      Labor tax rate on sector x:
tlx = 0.3;

$include ex_prod_05.gen
solve ex_prod_05 using mcp;
```

#### 4.6. ベンチマークにおいて投入物に対する税金が存在するケース (ex\_prod\_06.gms)

- $x$ 部門の資本、労働投入に税金をかけるケース。
- 今度は、ベンチマークにおいて既に税金がかかっているケース。
- 投入に対する税は従価税として扱う。
- 税金は直接消費者にlump-sumで還元される。
- 税金以外の部分はex\_prod\_01.gmsと全く同じ。

修正されたSAM :

Commodity (財)	Sectors (部門)			Consumers (消費者)
	x	y	u	cons
px	115		-115	
py		100	-100	
pu			215	-215
pl	-25	-75		100
pk	-75	-25		100
vtk	-10			10
vtl	-5			5

SAMの説明 :

- vtkとvtlの行がそれぞれ資本投入、労働投入に対する税金の支払いを表している。
- 税金は消費者にlump-sumで還元されるので、cons列にプラスとして入ってくる。
- 投入税は従価税、かつnet baseの税として処理するので、ベンチマークにおける $x$ 部門の資本、労働に対する税率をそれぞれ $tkx_0$ ,  $tlx_0$ とすると、 $tkx_0=10/75$ ,  $tlx_0=5/25$ で求めることができる。

生産関数を定義するMPSGEのコード：

```
*      部門 x の生産関数
$prod:x  s:sig_x
      o:px      q:x0
      i:pk      q:kx0  p:((1+tkx0)*pk0)  a:cons  t:tkx
      i:pl      q:lx0  p:((1+tlx0)*pl0)  a:cons  t:tlx
```

コードの説明：

- a:tというフィールドの指定はex\_prod\_05.gmsと全く同じ.
- 違うのは投入物の参照価格フィールド (p:) の指定.
- tkx0とtlx0はそれぞれ資本、労働に対するベンチマークの税率.
- ここまで参照価格にはベンチマークにおける投入物の価格を指定すると説明してきたが、正確には「ベンチマークにおける投入物のエージェント価格」を指定しなければならない. エージェント価格とはこの経済主体が直面する価格であり、このケースでは税込の投入物価格のことである.
- 税込の価格は資本、労働でそれぞれ  $(1+tkx0)*pk0$ ,  $(1+tlx0)*pl0$  で与えられる.

#### 4.7. 生産物に対する税が存在するケース (ex\_prod\_07.gms)

修正されたSAM：

Commodity (財)	Sectors (部門)			Consumers (消費者)
	x	y	u	cons
px	120		-120	
py	25	100	-125	
pu			245	-245
pl	-35	-75		110
pk	-85	-25		110
vtx	-20			20
vty	-5			5

SAMの説明：

- ベンチマークにおいてx部門の生産 (x財とy財) に税金がかかっているケース.
- vtx と vty がそれぞれx財とy財に対する税金を表す行.
- 120と25というx部門の生産額は市場価格表示. つまり,

- $120 = \text{生産者価格表示の生産額} + \text{生産物税額} = 100 + 20$
- $25 = \text{生産者価格表示の生産額} + \text{生産物税額} = 20 + 5$
- 生産物に対する税は従価税として扱う.
- さらに, 生産物に対する税率は「gross base」の形で表す. つまり,
  - $\text{生産者価格} = (1 - t) \times \text{市場価格}$
  - $t = \text{生産税額} / \text{市場価格表示の生産額}$
 という形式で税率  $t$  を表現する. 従って, 生産税率については常に「 $t < 1$ 」でなければならない.
  - 実際にベンチマークの生産税率を求めると,  $t_{xx0} = 20/120 = 0.1667$ ,  $t_{xy0} = 5/25 = 0.2$  となる.
- 税収は直接消費者にlump-sumで還元される.
- 生産物税以外の部分についてはex\_prod\_04.gmsと同じ.

生産関数を定義するMPSGEのコード:

```
*      部門 x の生産関数
$prod:x  s:sig_x  t:eta_x
      o:px      q:xx0  p:((1-txx0)*px0)  a:cons  t:txx
      o:py      q:xy0  p:((1-txy0)*py0)  a:cons  t:txy
      i:pk      q:kx0  p:pk0
      i:pl      q:lx0  p:pl0
```

コードの説明:

- a:とt:については投入物に対する税金のケースと同じ役割.
- ただし, t:で指定する税率は「gross baseの税率」.
- 生産物指定行の「参照価格」については, 投入物に対する税のケースと同様に, ベンチマークにおける「エージェント価格」を指定. この場合のエージェント価格は, 生産者が直面する価格であり「市場価格×(1-税率)」に等しい.

#### 4.8. 政府部門が存在するケース (ex\_prod\_08.gms)

- 表 2のSAMを前提とする.

政府支出財部門 (g部門) の生産関数:

```
      G
      / \ <- sig_g
      /   \
      /     \
```

x	y
---	---

生産関数の説明：

- x財, y財のCES型関数として「政府支出財（g財）」が生産される.
- 代替の弾力性はsig\_gで表現する.

以下, MPSGEのコードの修正点.

アクティビティー変数 (sector) の宣言：

```
*      アクティビティー変数 (sector) の宣言
$sectors:
  x          ! x 部門の生産量
  y          ! y 部門の生産量
  u          ! u 部門の生産量 (効用水準)
  g          ! g 部門の生産量 (政府支出水準)
```

説明：

- g 部門のアクティビティー変数が宣言に加わる.

価格変数 (commodity) の宣言

```
*      価格変数 (commodity) の宣言
$commodities:
  px         ! x 財の価格
  py         ! y 財の価格
  pu         ! u 財の価格 (効用の価格)
  pk         ! 労働の価格
  pl         ! 資本の価格
  pg         ! 政府支出財の価格
```

説明：

- 政府支出財の価格 (pg) が宣言に加わる.

所得変数 (consumer) の宣言

```
*      所得変数 (consumer) の宣言
$consumers:
  cons       ! 消費者の所得
  gov        ! 政府の所得
```

説明：

- 政府が consumer の一人として宣言に加わる.
- 政府の所得変数は gov で表現.

g 部門の生産関数（政府支出財部門）：

```
$prod:g s:sig_g
  o:pg      q:g0
  i:px      q:xg0 p:px0
  i:py      q:yg0 p:py0
```

説明：

- 普通の生産部門と全く同様の扱い.
- ex\_prod\_08.gms では sig\_g = 0 と設定している. つまり, 政府支出財の生産に各財は固定比率で投入されるということ.

政府の所得・支出の定義：

```
$demand:gov
  d:pg
```

説明：

- 全ての税について「a:gov」と指定しているため, 税金は全て政府の所得となる.
- 政府は何も賦存していないため「賦存財指定行」はなく, 「需要指定行」しかない. 「賦存財指定行」, 「需要指定行」については第 6 節を参照.
- 支出は全て政府支出財に費やされる. 税金が変化しただけ政府支出額が変化する.

## 5. MPSGE の内部動作

以下は ex\_prod\_06.gms における x 部門の特定化：

```
*      部門 x の生産関数
$prod:x s:sig_x
  o:px      q:x0
  i:pk      q:kx0 p:((1+tkx0)*pk0) a:cons t:tkx
  i:pl      q:lx0 p:((1+tlx0)*pl0) a:cons t:tlx
```

このようなコードの場合、生産関数（費用関数、要素需要関数）は次のように特定化される。

### 5.1. Step 1

まず、ベンチマークの単位費用  $cx_0$  と投入額シェア  $sh_{k0}, sh_{l0}$  が次のように計算される。

```
cx0 = ((1+tkx0)*pk0*kx0+ (1+tlx0)*pl0*lx0) / x0;
sh_k0 = (1+tkx0)*pk0*kx0 / (cx0*x0);
sh_l0 = (1+tlx0)*pl0*lx0 / (cx0*x0);
```

### 5.2. Step 2

Calibrated share form の形式で単位費用関数が特定化される。

```
cx = cx0 * (sh_k0*((1+tkx)*pk/((1+tkx0)*pk0))**(1-sig_x)
  + sh_l0*((1+tlx)*pl/((1+tlx0)*pl0))**(1-sig_x))**(1/(1-sig_x));
```

### 5.3. Step 3

要素需要関数が次のように特定化される。

```
kx = kx0 * ((cx / cx0) / ((1+tkx)*pk/((1+tkx0)*pk0)))**(sig_x) * x;
lx = lx0 * ((cx / cx0) / ((1+tlx)*pl/((1+tlx0)*pl0)))**(sig_x) * x;
```

$kx, lx$  はそれぞれ資本、労働に対する需要、 $x$  はベンチマークで 1 をとるように基準化された生産量。

### 5.4. 数式による表現

以上を数式で書くと

$$\bar{c}^x = \frac{(1 + \bar{t}_x^k) \bar{p}^k \bar{k}_x + (1 + \bar{t}_x^l) \bar{p}^l \bar{l}_x}{\bar{x}}$$

$$\theta_x^k = \frac{(1 + \bar{t}_x^k) \bar{p}^k \bar{k}_x}{\bar{c}^x \bar{x}}$$

$$\theta_x^l = \frac{(1 + \bar{t}_x^l) \bar{p}^l \bar{l}_x}{\bar{c}^x \bar{x}}$$

$$c^x = \bar{c}^x \left[ \theta_x^k \left( \frac{(1 + t_x^k) p^k}{(1 + \bar{t}_x^k) \bar{p}^k} \right)^{1-\sigma_x} + \theta_x^l \left( \frac{(1 + t_x^l) p^l}{(1 + \bar{t}_x^l) \bar{p}^l} \right)^{1-\sigma_x} \right]^{\frac{1}{1-\sigma_x}}$$

$$k_x = \bar{k}_x \left[ \frac{c^x / \bar{c}^x}{(1 + t_x^k) p^k / ((1 + \bar{t}_x^k) \bar{p}^k)} \right]^{\sigma_x} x$$

$$l_x = \bar{l}_x \left[ \frac{c^x / \bar{c}^x}{(1 + t_x^l) p^l / ((1 + \bar{t}_x^l) \bar{p}^l)} \right]^{\sigma_x} x$$

ただし、バー付きの値はベンチマークの値である。

### 5.5. 参照価格 (reference price) についての注

以上より、参照価格は次の性質を持つことになる。

1. 投入物（産出物）が一つの際には投入物（産出物）に対する参照価格を指定する意味はない。
2. 参照価格は相対的な大きさのみが意味を持つ。

これは参照価格がシェアを求めるためのみに用いられるためである。投入物が一つの際にはどのような参照価格を指定しようが常にその投入物のシェアは1となる。よって、1が成り立つ。2については、参照価格を定数倍してもシェアは不変となるからである。例えば、x部門の reference price filed を次のように変えたとする。

* 部門 x の生産関数	
\$prod:x s:sig_x	
o:px	q:x0
i:pk	q:kx0 p:((1+tkx0)*pk0*100) a:cons t:tkx
i:pl	q:lx0 p:((1+tlx0)*pl0*100) a:cons t:tlx

つまり、参照価格をそれぞれ 100 倍したとする。このとき

```

cx0 = 100*((1+tkx0)*pk0*kx0+ (1+tlx0)*pl0*lx0) / x0;
sh_k0 = 100*(1+tkx0)*pk0*kx0 / (cx0*x0);
sh_l0 = 100*(1+tlx0)*pl0*lx0 / (cx0*x0);

```

となり、 $cx_0$  は 100 倍となるが、投入シェアは変わらない。また、単位費用  $cx$  は

```

cx = cx0 * (sh_k0*((1+tkx)*pk/((1+tkx0)*pk0*100))**(1-sig_x)
  + sh_l0*((1+tlx)*pl/((1+tlx0)*pl0*100))**(1-sig_x))**(1/(1-sig_x))

```

のように分母の部分が 100 倍となるが、同時に  $cx_0$  も 100 倍になっているので、やはり変わらない。要素需要関数についても同様に変わらない。

## 6. \$demand ブロックの説明

以下では、`ex_prod_01.gms` を使って、`$demand` ブロックを説明する。

`ex_prod_01.gms` の `$demand` ブロック：

```

*      予算制約
$demand:cons
  d:pu
  e:pl      q:(e_l*s_l)
  e:pk      q:(e_k*s_k)

```

コードの説明：

- `$demand` ブロックは「所得の源泉」と「所得の使い道」を定義する。
- 一つの消費者（所得変数）に対して、一つの `$demand` ブロックを対応させる。
- `$demand` ブロックは、「d:」行と「e:」行の二つからなる。
- d:行（需要指定行）
  - これは所得の使い道を指定する行。所得によってどの財を購入するのかを指定する。
  - d:の後に購入する財の価格を指定する。
  - 上の例では、消費者は `pu`（効用）の購入に支出を利用する。

- e:行 (賦存財指定行)
  - これは「賦存財」を定義する行. 「賦存財」とは生産されずに **consumer** が所有している財.
  - e:の後に賦存している財の価格を指定, さらに q:フィールドで賦存量を指定する.
  - 上の例では, 消費者 (**cons**) は資本, 労働の二つの生産要素を保有しているので, 二つの行がある.
  - e:pl が労働の賦存を表す行で, 消費者は (e\_l\*s\_l) だけ労働を賦存し供給する. 同様に消費者は (e\_k\*s\_k) だけ資本を賦存し供給する.
  - 賦存財供給 (要素供給) により消費者 (**cons**) には,  $pl*e_l*s_l + pk*e_k*s_k$  だけの所得が入ってくることになる.
  - q:フィールドには「負の値」を指定してよい. この場合, **cons** が一定量の需要をするということになる.
- 所得の源泉としては税もあるが, 税については \$prod ブロック内の a:フィールドで指定するので, \$demand ブロックには現れない (lump-sum tax の場合は除く).
- また, \$demand ブロックでは「r:」という「rationing instrument」を指定するためのフィールドもあるが, これについては「補助変数」と一緒に使うので, 補助変数の説明の部分で説明する (第 8 節).
- \$demand ブロックについての注
  - 本来, \$demand ブロック, 特にその中の d:行ではもっと多様な指定をすることができる.
  - 例えば, これまで消費活動を一つの生産活動とみなし, **sector** として扱ってきたが, 消費活動をこの \$demand ブロックで表現することもできる. これについては, `ex_dem_01.gms` を参照.
  - しかし, 効用関数が複雑なケースでは \$demand ブロックで消費を表すのは難しく, アクティビティーの一つとして表すほうが自由度が大きい. このため, この説明では
    - 消費活動は効用という財を生産する活動 (**u** 部門)
    - 消費者は **u** 部門によって生産された効用を購入  
という後者の方法を採用している.

## 7. \$report 命令の利用方法

MPSGE で宣言・定義できる変数は基本的に「アクティビティー変数」, 「価格変数」, 「所得変数」の 3 タイプのみ. これら 3 タイプの変数については計算結果にその値がリポートされる. これら以外にも \$report 命令を利用することで変数を定義できる.

以下, `ex_report_01.gms` を利用して, \$report 命令の利用法を説明する.

## 7.1. 例 1

```

*      部門 x の生産関数
$prod:x  s:sig_x
      o:px      q:x0
      i:pk      q:kx0  p:pk0
      i:pl      q:lx0  p:pl0

$report:
      v:z_x      o:px  prod:x      ! x 財の生産量
      v:z_kx     i:pk  prod:x      ! x 部門の資本投入量
      v:z_lx     i:pl  prod:x      ! x 部門の労働投入量

```

説明：

- この例では、x 部門の生産量、資本投入量、労働投入量を表す変数を定義している。
- v:フィールド
  - このフィールドで定義する変数の名前を指定する。v:z\_x は z\_x という名前の変数ということ。
  - \$report 命令ではまずこの v:フィールドを持ってくる。
- v:フィールドの後にくるフィールドはどのような変数を定義するかで変わってくる。
- 「o:px prod:x」という指定では、x 部門における生産物 x という指定になる。生産物のケースでは「o:価格変数」という指定になる。
- 「i:pk prod:x」は x 部門における資本投入量という指定である。投入物の場合には「i:価格変数」という形の指定をする。
- 「変数 x」と「変数 z\_x」の違い。
  - どちらも生産量を表しているように見えるが両者は異なる。
  - z\_x は文字通り x 財の生産量を表すが、アクティビティーレベルを表す x はベンチマーク均衡における値が 1 となるように基準化された変数。つまり、「 $x = z_x / x_0$ 」によって定義された変数。
  - 通常、CGE 分析では変数の変化率にのみ関心があることが多い。その場合には、変化率をすぐに把握できるアクティビティー変数 x を見ればよいのだが、場合によっては生産量の絶対的な水準に関心がある場合もある。その場合には、z\_x のように生産量を表す変数を \$report 命令を使って定義すればよい。
- \$report 命令によって定義した変数は、通常の 3 タイプの変数と同様に、計算結果にその値が表示されるようになる。
- 行の最後の「!」の後はコメント行となる。変数の説明をここに書いておけばよい。ただし、\$sectors ブロック、\$commodities ブロックでの「!」のコメントは変数の説明文

として計算結果でも表示されるのに対し、`$report` ブロックでの「!」によるコメントは計算結果には表示はされない。

例 2 :

```
*      部門 u の生産関数 (効用関数)
$prod:u  s:sig_u
  o:pu    q:u0
  i:px    q:cx0  p:px0
  i:py    q:cy0  p:py0

*      予算制約
$demand:cons
  d:pu
  e:pl    q:(e_l*s_l)
  e:pk    q:(e_k*s_k)

$report:
  v:s_u    o:pu  prod:u      ! u の供給量
  v:d_u    d:pu  demand:cons ! 消費者の u の需要量
```

説明 :

- 変数 `s_u` については例 1 と同じように `$prod` ブロックに関連する変数である。u 部門における効用の生産量を表す変数として定義されている。
- 変数 `d_u` は `$demand` ブロックに関連する変数。
  - これは `cons` という消費者の財 `pu` (効用) に対する需要量を表す変数として定義されている。
  - `cons` の `demand` ブロックにおける `pu` という財への需要であるので、「`d:pu demand:cons`」という指定になる。

`$report` 命令では上にあげた例以外のタイプの変数も定義することができるが、とりあえずは `$prod` ブロックの「生産量」、`$demand` ブロックの「需要量」の定義をおぼえておけばよい。

## 8. 補助変数 (auxiliary variable) の利用方法

### 8.1. `$auxiliary` 命令と `$constraint` 命令

- 補助変数はモデルを拡張する際に利用される。
- 具体的には、sector に対する「税率・補助金率」、consumer から別の consumer への「トランスファー」等を内生的に決めるようなモデルにする場合に、税率、トランスファーを表す変数を補助変数として指定することになる。
- 補助変数は、その条件式を規定する \$constraint 命令と一緒に利用する。
- 一つの補助変数に対し、一つの \$constraint ブロックが対応することになる。
- 補助変数と constraint の組み合わせを使うことで、例えば、以下のようなシミュレーションができる。
  - x 部門の生産量がある一定の水準に達するように x 部門への生産補助金を決めるモデル。
  - 従価税ではなく、従量税（例えば炭素税）を使ったモデル。
  - 二重の配当分析で扱われる tax swap のモデル。
  - OBA (output-based allocation) 方式に基づく排出量取引制度。

## 8.2. 利用例 1（内生的に変化する一括税）

政府支出の変化に対し、一括税（lump-sum tax）の水準が内生的に変化するケース。

修正された SAM：

Commodity (財)	Sectors (部門)				Consumers (消費者)	
	x	y	u	g	cons	gov
px	100		-90	-10		
py		100	-80	-20		
pu			170		-170	
pl	-25	-75			100	
pk	-75	-25			100	
pg				30		-30
<b>vlst</b>					<b>-30</b>	<b>30</b>

SAM の説明：

- 一括税以外の税は存在しない。
- vlst が一括税のフローを表す行。
- 政府 (gov) は消費者 (cons) から 30 の一括税をとっている。

モデル：

- モデル上では、一括税は「政府 (gov) が初期賦存している効用 (u 財) を消費者 (cons) に販売する」という形式で導入される。
- 例えば、政府が効用を `vlst0` だけ初期賦存しており、それを消費者に売ったとすると消費者から政府への「`pu*vlst0`」だけの所得の移動がおこる。これを一括税とみなすという事。
- 一括税のモデル化についての注
  - 名目値が意味を持たない実物モデルであるので、一括税も結局なんらかの財の移動という形をとることになる。ここでは効用という財の移動という形で実質化している。
  - 必ずしも効用という財を使う必要はない。労働、資本、`x` 財、`y` 財の移動という形でもよい。

コードの変更：

まず、以下のように `vlst0` にベンチマークにおける一括税の額を入れておく。

```
Parameter
    vlst0      ベンチマークにおける lump-sum tax
;
vlst0 = sam("vlst","gov");
```

補助変数の宣言：

```
*      補助変数の宣言
$auxiliary:
    lst      ! 一括税
```

`lst` という一括税の水準を表す補助変数を宣言しておく。

`$demand` ブロックの修正：

```
*      民間消費者の予算制約
$demand:cons
    d:pu
    e:pl      q:(e_l*s_l)
    e:pk      q:(e_k*s_k)
    e:pu      q:(-vlst0)      r:lst

*      政府の予算制約
```

```
$demand:gov
  d:pg
  e:pu      q:(vlst0)    r:lst
```

コードの説明 :

- 修正されているのは「賦存財指定行」.
- 消費者 (cons)
  - 消費者側では
 
$$e:pu \quad q:(-vlst0) \quad r:lst$$
 と指定されている.
  - これまでの「賦存財指定行」では、賦存量を  $q$ :フィールドで指定してきた.
  - $r$ :フィールドは「rationing instrument」を指定するためのフィールドで、 $r$ :フィールドがある場合には、 $r$ :フィールドで指定された変数を掛け合わせたものが賦存量になる. つまり、上の例では「 $-vlst0*lst$ 」が賦存量となる.
  - 賦存量がマイナスであるので、消費者が「 $vlst0*lst$ 」だけの効用 ( $u$  財) を需要するという意味になる.
  - この指定により消費者の所得 (cons) から「 $-pu*vlst0*lst$ 」だけの所得が差し引かれることになる. これが政府への一括税を意味する.
- 政府 (gov)
  - 政府側では逆に
 
$$e:pu \quad q:(vlst0) \quad r:lst$$
 という指定となる.
  - 政府にとっての賦存量は「 $vlst0*lst$ 」となりプラスであるので、政府は「 $vlst0*lst$ 」だけの効用を供給すること.
  - これにより政府 (gov) に「 $pu*vlst0*lst$ 」だけの所得が加わることになる. これを一括税の受け取りとみなす.
- $r$ :フィールドの意味
  - $r$ :フィールドでは補助変数を指定する. この補助変数は内生的に変化するので、一括税の額 (量) が変化することになる.
  - 補助変数の値がどう決まるかは、`$constraint` ブロックで指定する.

`$constraint` ブロックの追加 :

```
*      制約

*      補助変数 lst の決定条件: lst は政府支出 (g * g0) が外生的に与えら
*      れる水準 (s_gov * g0) に等しくなるように決定.

$constraint:lst
```

```
g *g0 =e= s_gov * g0;
```

コードの説明：

- `$constraint` ブロックでは補助変数の制約式（決定条件式）を指定する.
- `$constraint:`の後で補助変数の名前を指定.
- 書き方は普通の GAMS の式（equation）の書き方と同じ.
- 上の例では、政府支出の水準（g）がある外生的に与えられる水準（s\_gov）に等しくなるように `lst` が決まるということになる.
- 外生的に与えられる政府支出の水準が達成されるように一括税の水準が内生的に変化するという事.

### 8.3. 利用例 2（内生的に変化する労働課税）

二つ目の例は労働課税が内生的に変化するケース.

モデル：

- 一括税を引き下げる代わりに、その分労働課税を引き上げるケース.
- 政府支出を一定に保つという条件で労働課税を調整する. 労働課税率は内生的に決ってくる.
- ベンチマーク均衡では労働課税はないと仮定する.
- ベンチマーク・データは `ex_aux_01.gms` と同じものを使う.

モデルの変更

```
*      補助変数の宣言
$auxiliary:
  tlx      ! x 部門の労働に対する税率 (内生変数)
```

- 今度は労働税率が内生変数となる. 労働税率にあたる変数を補助変数として宣言する.

`$prod` ブロックの修正：

```
*      部門 x の生産関数
$prod:x  s:sig_x
  o:px      q:x0    p:px0
  i:pk      q:kx0    p:pk0
  i:pl      q:lx0    p:pl0  a:gov  n:tlx
```

コードの説明 :

- 通常の投入税と違うのは、税率を指定するのに **t:**フィールドではなく **n:**フィールドを利用していること.
- 内生的に変化する税率の場合には **n:**というフィールドを利用する.
- 他は税率が外生的なケースと変わらない.

**\$demand** ブロック :

```

*      予算制約

*      民間消費者の予算制約
$demand:cons
  d:pu
  e:pl      q:(e_l*s_l)
  e:pk      q:(e_k*s_k)
  e:pu      q:(-vlst0)

*      政府の予算制約
$demand:gov
  d:pg
  e:pu      q:(vlst0)

```

コードの説明 :

- 今回は一括税は外生的に与える. 従って, **r:**フィールドはなく, 通常の賦存財と同様に一括税を扱う.
- **vlst0** の変化が一括税の変化を意味する.

**\$constraint** ブロック :

```

*      制約

*      補助変数 tlx の決定条件: tlx は政府支出 (g*g0) が外生的に与えられる水
*      準 (s_gov * g0) に等しくなるように決定.
$constraint:tlx
  g * g0 =e= s_gov * g0;

```

コードの説明 :

- 補助変数として **tlx** を指定していることを除けば, **ex\_aux\_01.gms** の **\$constraint** ブロックと同じ.

- `ex_aux_02.gms` のシミュレーションでは `s_gov` は常に一定。よって、`g` も一定。これより上の条件は、一定の政府支出 (`g`) の水準を保つように労働課税率 (`tlx`) を決めるという意味になる。

上の二つの例では、`$constraint` ブロックの条件はどちらも政府支出を一定に保つという条件であったが、どのような条件を設定するかは自由に決めることができる。

## 9. MPSGE でのモデルの実行方法

### 9.1. 変数の初期値

- 「アクティビティー変数」、 「価格変数」 のデフォルトの初期値は1.
- 「補助変数」 のデフォルトの初期値は0.

もしデフォルト値とは異なる値を初期値としたい場合には、自分で指定する必要がある。

### 9.2. 「アクティビティー変数」 の値

アクティビティー変数はベンチマークにおいて 1 に等しくなるように基準化された形で定義される。例えば、`x` 部門の生産関数が以下のように定義されていたとする。

```
*      部門 x の生産関数
$prod:x  s:sig_x
      o:px   q:x0
      i:pk   q:kx0  p:pk0
      i:pl   q:lx0  p:pl0
```

この場合、「 $x \cdot x_0 = x \cdot \text{ベンチマークの生産量} = \text{生産量}$ 」というように `x` は定義される。つまり、`x` は生産量 (アクティビティーレベル) そのものではなく、ベンチマークにおいて 1 をとる変数として基準化された形で定義される。このようにベンチマークにおいて 1 と基準化されていることから、何らかのショックを与えた際のベンチマーク値からの変化率を容易に読み取ることができる。仮に生産量の水準そのものが知りたければ、`x*x0` で求めればよい (あるいは、`$report` 命令によって生産量を表す変数を定義すればよい)。

### 9.3. モデルの解き方

#### 9.3.1. モデルを解くコード

MPSGEでモデルを解くコード：

```
$include model_name.gen
solve model_name using mcp;
```

コードの説明：

- 通常のGAMSのモデルではsolve命令だけでモデルを解くことができる.
- MPSGEではsolveの前に一つ命令を加える必要がある.
- model\_nameの部分にはMPSGEの\$model:命令で定義したモデル名を指定する.
- MPSGEはMCP (mixed complementarity problem) の形式の問題を記述するので、使うsolverはMCP solver.

### 9.3.2. numeraire (ニューメレール) について

- 通常, MPSGE (というか多くのCGE分析) で記述するモデルは実物モデル, つまり名目変数についてゼロ次同次の性質を持つモデルである. さらに, Walras法則が成り立つ, つまり市場均衡条件の一本はredundantとなるモデルである.
- 「名目変数についてゼロ次同次」 + 「Walras法則」というモデルでは, 一つの価値基準財 (ニューメレール, numeraire) を指定して解く, つまりある一つの財の価格を1で固定して解くことが多い.
- しかし, MPSGEでは解く際に自動で名目値の水準のnormalizationが行なわれるので, numeraireを指定しなくて解くことが可能である. 特に理由がなければnumeraireを指定しなくてよい. 実際, この文書で利用しているサンプルのプログラムでは解く際にnumeraireを指定していない.
- ただし, numeraireを指定していなくても, モデル内で名目値が意味がない (相対価格, 実質値しか意味がない) のは同じであるので, 価格や金額を表す変数を計算結果として出す際には, なんらかのデフレーターを使って実質化する必要はある.
- また, numeraireを指定して解く必要はないが, 指定したければ指定してもよい. その場合には, 変数に".fx"というsuffixを付けて指定してやる. 例えば, 労働をnumeraireとしたければ

```
pl.fx = 1;
```

という指定をして解いてやればよい.

## 9.4. 結果の見方

MPSGEにおける計算結果の見方について.

### 9.4.1. SOLVE SUMMARYブロック

- まず、計算したらSOLVE SUMMARYブロックを見る。
- 見るのは「SOVER STATUS」と「MODEL STATUS」の値
- 表 7のように二つのSTATUSが1になっていない場合には、正常にモデルが解けていないということ。
- 表 8のようにどちらも1となっているときは、モデルが正常に解けているということ。

表 7：モデルが正常に解けていないケース。

S O L V E		S U M M A R Y	
MODEL	EX_DEBUG_01		
TYPE	MCP		
SOLVER	PATH	FROM LINE	447
****	SOLVER STATUS	2	Iteration Interrupt
****	MODEL STATUS	6	Intermediate Infeasible

表 8：モデルが正常に溶けているケース。

S O L V E		S U M M A R Y	
MODEL	EX_DEBUG_01		
TYPE	MCP		
SOLVER	PATH	FROM LINE	561
****	SOLVER STATUS	1	Normal Completion
****	MODEL STATUS	1	Optimal

#### 9.4.2. 内生変数の値

- モデルが正常に解けているのなら、内生変数（アクティビティー変数、価格変数、所得変数、補助変数）の値を確認すればよい。
- GAMSでは各変数に対して以下の「4つの値」が関連付けられている。
  - 「LOWER」, 「UPPER」, 「LEVEL」, 「MARGINAL」
  - LOWER：これは変数の「下限値」。MPSGEでは変数の下限値のデフォルトは0である。ピリオドはゼロを表す。
  - UPPER：これは変数の「上限値」。MPSGEでは変数の上限値のデフォルトは無限大(+inf)である。

- LEVEL：これが普通の意味での変数の値であり、その変数在那个時点でとっている値を示す。
- MARGINAL：これはその変数が関連付けられた条件式における乖離幅を表す。これについてはまた後で詳しく説明する。

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	0.916	+INF	.
---- VAR Y	.	0.993	+INF	.
---- VAR U	.	0.798	+INF	.

例えば、上のような計算結果となっているとすると「変数Xの下限值は0, 上限値は無限大, MARGINAL値は0, 値は0.916」ということ。他の変数について同様に解釈すればよい。

#### 9.4.3. MPSGEにおけるMARGINAL値の意味

変数のMARGINAL値は「その変数が関連付けられた条件式における乖離幅」を表している。

2.2節で見たように、MPSGEでは各変数はそれぞれ次のような条件と結びついていた。

- アクティビティー変数           ⇔     利潤最大化条件（ゼロ利潤条件）
- 価格変数                       ⇔     市場均衡条件
- 所得変数                       ⇔     予算制約条件

この関係より、変数のMARGINAL値は次のような意味を持つことになる。

変数	MARGINAL値の意味
アクティビティー変数   ⇒	当該部門の「超過費用＝費用－収入」
価格変数                 ⇒	当該財の「超過供給＝供給－需要」
所得変数                 ⇒	当該消費者の「超過所得＝所得－支出」

例えば、変数が次のようなMARGINAL値を持っていたとする。

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	0.916	+INF	10
---- VAR PK	.	0.859	+INF	-5

---- VAR CONS	.	210.000	+INF	100
---------------	---	---------	------	-----

これは次のように解釈できる.

- x部門は10だけ超過費用が発生している.
- 資本市場は5だけ超過需要の状態にある.
- 消費者の所得は支出を100上回っている.

この MARGINAL 値をチェックすることでモデルのデバッグをすることができる. これについては, 第 12 節で詳しく説明する.

## 10. MPSGE のシンタックス (まとめ)

### 10.1. MPSGE 特有の命令

- `$ontext-$offtext`
  - 通常の GAMS のコードでは, `$ontext-$offtext` はコメントブロックを表すが, MPSGE ブロックは`$ontext` と`$offtext` で囲む.
- `$model:[model name]`
  - モデル名を指定する. UNIX で MPSGE を実行するときには, 大文字で書いておく.
- `$sectors:`
  - `sector` を宣言する. `sector` はそれに関連付けられる「アクティビティー変数」によって宣言する.
  - 例えば, x 財を生産する x 部門を宣言するには x という変数をこの`$sectors:`ブロックに書く.
  - アクティビティー変数と呼んでいるが, 当該部門が 1 生産物しか生産しないのなら, アクティビティーレベル=生産量である.
- `$commodities:`
  - これは `commodity` を宣言する命令.
  - `commodity` はそれに関連付けられる「価格変数」で宣言する.
  - `commodity` には通常の財に加え, 生産要素も含める.
  - 例えば, `px` という価格を持つ x 財, `w` という価格を持つ労働という生産要素を宣言するには, `$commodities:`ブロックに `px` と `w` を書く.
  - `sector` に対し, その部門で生産する財の価格をここで指定するが, さらに生産されない財 (生産要素等の初期賦存がある財) についてもここで指定する.
- `$consumers:`
  - これは `consumer` を宣言する命令.
  - `consumer` はそれに関連付けられる「所得変数」で宣言する.

- 政府を導入する際には政府も一種の `consumer` としてここで宣言する.
- 例えば, `m` という所得を持つ代表的家計という `consumer` を宣言するには, `$consumers:` ブロックに `m` を書いておく.
- `$auxiliary:`
  - これは「補助変数 (auxiliary variable)」を宣言する命令.
  - 内生的に変化する税率, 内生的に変化するトランスファーを表す変数を定義できる.
  - 補助変数一つにつき一つの `$constraint:` 命令が対応する. よって, `$auxiliary` 命令と `$constraint` 命令はセットで利用する.
- `$prod:[sector name]`
  - `sector` の生産関数を定義する命令.
  - 一つの `sector` (アクティビティー変数) に対して, 一つの `$prod` ブロックが対応する.
- `$demand:[consumer name]`
  - `consumer` の予算制約を定義する命令.
  - 一つの `consumer` (所得変数) に対して, 一つの `$demand` ブロックが対応する.
- `$constraint:[auxiliary variable]`
  - 補助変数に対する制約式 (条件式) を定義する.
  - 一つの補助変数に対して, 一つの `$constraint` ブロックが対応する.
- `$report:`
  - 「アクティビティー変数」, 「価格変数」, 「所得変数」, 「補助変数」以外の変数を定義する命令.
  - 「生産量」, 「投入量」を表す変数を定義できる.
  - 主に結果をリポートするための変数を定義するための命令だが, ここで定義した変数を `$constraint` ブロック内で利用もできる.

## 10.2. 変数の宣言部分

アクティビティー変数 (`sector`) の宣言:

```
$sectors:
  x          ! x 部門の生産量
  y          ! y 部門の生産量
  u          ! u 部門の生産量 (効用水準)
```

説明:

- `sector` (アクティビティー変数) の宣言
- 「!」の後に変数の説明を書く.

価格変数 (`commodity`) の宣言:

```

$commodities:
  px      ! x 財の価格
  py      ! y 財の価格
  pu      ! u 財の価格 (効用の価格)
  pk      ! 労働の価格
  pl      ! 資本の価格

```

所得変数 (consumer) の宣言 :

```

$consumers:
  cons    ! 消費者の所得

```

### 10.3. \$prod ブロック

- `$prod:`のすぐ後にアクティビティ変数 (sector) を指定.
- `s:`
  - CES 関数のトップレベルのネストにおける代替の弾力性を指定する.
  - 省略した場合は 0 が指定される. つまり, Leontief 型関数となる.
- `t:`
  - CET 関数の変形の弾力性を指定する.
  - これは生産物が複数あるケースで指定する.
  - やはりデフォルト値は 0.
- `q:`
  - 「参照数量 (reference quantity)」を指定するフィールド.
  - 生産物指定行では「ベンチマーク生産量」, 投入物指定行では「ベンチマーク投入量」を指定しておけばよい.
  - デフォルト値は 1.
- `p:`
  - 「参照価格 (reference price)」を指定するフィールド.
  - 生産物指定行では生産物のベンチマーク価格, 投入物指定行では投入物のベンチマーク価格を指定する.
  - 参照価格は「agent 価格」で指定する.
    - 投入物のケース : agent 価格 = (1 + 税率) × 市場価格
    - 生産物のケース : agent 価格 = (1 - 税率) × 市場価格
  - デフォルト値は 1.
- `a:`
  - 税収の行き先 (tax revenue agent) を指定するフィールド
  - 行き先は「所得変数」で指定する.
- `t:`

- 税率を指定するフィールド.
- 税率は従価税の形式で指定する.
- 税率の形式は投入物のケースと生産物のケースで異なる.
  - 投入物に対する税: 「net base」 →  $(1 + \text{税率}) \times \text{市場価格}$
  - 生産物に対する税: 「gross base」 →  $(1 - \text{税率}) \times \text{市場価格}$
- 同じ投入物, 生産物に複数の税を導入することも可能. この場合には, 一つの投入物・生産物指定行に複数の t:フィールドを含める.
- n:
  - 内生的に変化する税率を指定するフィールド
  - 税率は「補助変数」として宣言しておく.
  - それ以外は t:と同じ.
- m:
  - これは内生的に変化する税率に対する乗数
  - 例えば

```
$prod:x s:sig_x
  o:px      q:x0
  i:pk      q:kx0 p:((1+txk0)*pk0) a:cons m:txk0 n:etxk
  i:pl      q:lx0 p:pl0
```

というコードでは「txk0\*etxk」が内生的に変化する資本に対する税率となる.

- 上のモデルと以下のコードは基本的に同じモデルになるが, 下記のコードでは「etxk」が税率となる. 実質的なモデルの違いはないが, 上のコードの場合, etxkの値を見ることでベンチマークの税率から何%変化しているかをすぐ読み取れることができるという利点がある.

```
$prod:x s:sig_x
  o:px      q:x0
  i:pk      q:kx0 p:((1+txk0)*pk0) a:cons n:etxk
  i:pl      q:lx0 p:pl0
```

- また, 多数の部門の税率を比例的に変化させたいというときには, m:フィールドに各部門のベンチマークの税率を指定し, n:フィールドの変数を変化させるという形にすればよい.

#### 10.4. \$demand ブロック

- d:ライン (需要・支出指定行)
  - p:
    - ここに当該 consumer が需要する財の価格変数を指定する.
  - d:ラインでも consumer の効用関数を指定することができるが, ここではそのやり方の説明は省略する.
- e:ライン (賦存財指定行)

- p:
  - 賦存財の「価格変数」を指定.
- q:
  - ここには「参照数量」を指定する. 賦存財の場合, 初期賦存量を指定すればよい.
  - マイナスの値を指定することもできる. この場合, 一定量の需要を表すことになる.
- r:
  - これは「rationing instrument」を指定する. 財の賦存量を内生的に変化させる場合に利用する.
  - rationing instrument には「補助変数」を指定する.

### 10.5. \$constraint ブロック

- 各補助変数に対しては, その変数がどのように決定されるかを示す条件式を指定してやる. \$constraint はそのための命令.
- \$constraint ブロックでは, 通常の GAMS の数式の書き方を利用する. つまり, 「=」ではなく「=e=」というような書き方をし, さらに終わりにセミコロンをつける.

## 11. MPSGE による記述から数式への記述の書き換え

追加

## 12. MPSGE でのデバッグ

一口にデバッグといっても, プログラムの文法の誤り (シンタックス・エラー) を修正する, 自分の意図通りのモデルになるようにチェックをする, データのチェックをするなど, 様々なケースがある. このうち, シンタックス・エラーについては GAMS 実行時にエラーメッセージが表示され, そのメッセージに従って誤りを修正すればよいので, 比較的対処しやすい (エラーについては, [GAMS User's Guide](#) や [McCarl GAMS User Guide](#) が詳しい). これに対して, モデル・データ自体の誤り, すなわち経済学的に意味のあるモデル・データになっているのか, 自らの意図するモデル・データになっているのかについては, 必ずしもエラーメッセージが出るわけではないので, デバッグする (間違いを修正する) のが比較的難しい. 以下では, 後者の意味でのデバッグの方法について説明する.

MPSGE において、モデル・データが適切に作成されているかをチェックするには次のような手段がある。

- Benchmark replication チェック.
- Cleanup calculation によるチェック.
- スケール・ショックによるチェック.
- MPSGE のデバッグオプションによるチェック.

以下、ex\_debug\_01.gms を例にして説明する。

### 12.1. Benchmark replication チェック

Benchmark replication とは、ベンチマーク・データの下でモデルが均衡状態にあるかをチェックすることである。ex\_debug\_01.gms では次のような形でモデルを解いている。

```
ex_debug_01.iterlim = 0;
$include ex_debug_01.gen
solve ex_debug_01 using mcp;
```

一行目の

```
ex_debug_01.iterlim = 0;
```

はモデルを特裁の iteration 回数を 0 に設定する命令である。GAMS では iteration の上限値は「model\_name.iterlim=回数」で設定することができる。

iteration の回数を 0 に設定して解くということは、変数の初期値が均衡条件式を満たしているかどうか判断するだけで終わりということを意味する。MPSGE (CGE 分析) では、ベンチマーク・データが均衡状態にあるという前提で分析をおこなう。この前提が実際に成り立っているかを判断するのが benchmark replication チェックである。仮に成り立っていないとすると、モデル、あるいはデータに誤りが存在するということになる。

ex\_debug\_01.gms を実際に実行してみると、次のような結果が出る。

```

                SOLVE      SUMMARY

MODEL   EX_DEBUG_01
TYPE    MCP
SOLVER  PATH                      FROM LINE  447
```

```
**** SOLVER STATUS      2 Iteration Interrupt
**** MODEL STATUS      6 Intermediate Infeasible
```

SOLVER STATUS と MODEL STATUS がともに 1 でないことから、モデルが正常に解けていない（この場合、ベンチマーク・データが均衡条件を満たしていない）。

変数の値を見ると次のようになっている。

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	1.000	+INF	20.000
---- VAR Y	.	1.000	+INF	30.000
---- VAR U	.	1.000	+INF	.
---- VAR PX	.	1.000	+INF	.
---- VAR PY	.	1.000	+INF	-30.000
---- VAR PU	.	1.000	+INF	-10.000
---- VAR PK	.	1.000	+INF	.
---- VAR PL	.	1.000	+INF	-10.000
---- VAR CONS	.	210.000	+INF	.

所得変数を除いたどの変数も初期値の 1 が LEVEL 値になっているのがわかる。均衡条件が満たされていないという結果であったが、どこが満たされていないかは変数の MARGINAL 値を見ることで判断できる。第 9.4.3 節の説明に従い、MARGINAL 値の解釈をすると、

- X の MARGINAL 値は 20。つまり、x 部門で 20 の超過費用が生じている。
  - Y の MARGINAL 値は 30。つまり、y 部門で 30 の超過費用が生じている。
  - PY の MARGINAL 値は -30。つまり、y 財市場で 30 の超過需要が生じている。
  - PU の MARGINAL 値は -10。つまり、u 財市場で 10 の超過需要が生じている。
  - PL の MARGINAL 値は -10。つまり、資本市場で 10 の超過需要が生じている。
- となる。

この情報はモデル・データのチェックをするためのヒントになる。実際、上の情報を元に以下のようにデバッグができる。

まず、x 部門で超過利潤が生じているということは、x 部門の生産関数の指定（\$prod ブロックの指定）に誤りがある可能性が高い。実際に x 部門の \$prod ブロックを見てみると

```
*      部門 x の生産関数
$prod:x  s:sig_x
          o:px      q:x0
```

```

i:pk      q:kx0  p:pk0
i:pl      q:(lx0 + 20)  p:pl0

```

となっている。労働投入の指定行における参照数量に 20 が足されている。これにより費用が 20 だけ増加することになり、超過費用が生じることになっていることがわかる。

同様に y 部門の \$prod ブロックを見ると

```

*      部門 y の生産関数
$prod:y  s:sig_y
  o:py      q:(y0 - 30)
  i:pk      q:ky0  p:pk0
  i:pl      q:ly0  p:pl0

```

である。生産量の参照数量において 30 引いていることが、超過費用の原因だということがわかる。y 財市場で 30 の超過需要が生じているのも、この生産量の参照数量で 30 差し引かれているため、ベンチマークの供給が 30 低くなってしまっているからである。

労働市場で 30 の超過需要が生じているが、これは x 部門の \$prod ブロックにおいて労働投入の参照数量に誤りがあったことが原因である。ただし、x 部門の指定における誤りは 20 だけの超過需要しか説明できない。そこで、労働の供給側を見てみると

```

*      予算制約
$demand:cons
  d:pu
  e:pl      q:(e_l*s_l + 10)
  e:pk      q:(e_k*s_k)

```

というように労働の賦存量の指定部分に 10 が足されている。x 部門の指定の誤りと賦存量の指定の誤りが組み合わさり、30 という超過需要が生じていることが確認できる。

以上のように ex\_debug\_01.gms には 3 箇所の誤りが存在することがわかる。3 箇所の誤りを修正して解いてみると、SOLVER STATUS も MODEL STATUS もともに 1 となり、しかも全ての変数の MARGINAL 値はゼロ（あるいは、ほぼゼロに近い値）になるはずである。

このように

- ① iteration 回数をゼロにして解く。
- ② MARGINAL 値がゼロになっていない変数を探す。
- ③ アクティビティー変数の場合 ⇒ その部門の \$prod ブロックをチェック。
- ④ 価格変数の場合 ⇒ その財の需要・供給がある \$prod ブロック、\$demand ブロックをチェック。

- ⑤ 所得変数の場合 ⇒ \$demand ブロック, tax agent の指定 (a:) をチェック.  
 という手順を繰り返すことで, モデルの誤りを容易に発見・修正することができる.

## 12.2. Cleanup calculation によるチェック.

Cleanup calculation も前節での benchmark replication と同様にベンチマーク・データの下でモデルが均衡状態にあるかどうかをチェックするものである. ベンチマーク・データの下で均衡が成立しているならば, 何もショックを与えないでモデルを解いた場合に, 変数の初期値がそのままモデルの解になっていなければならない.

前節で見つけた誤りを修正せずに

```
*      Exit A:
$exit
```

という部分の \$exit 命令をコメントアウトして解いてみる. すると, 今度は

```
ex_debug_01.iterlim = 10000;
$include ex_debug_01.gen
solve ex_debug_01 using mcp;
```

という形でモデルを解くことになる. 計算結果は次のようになる.

```

                S O L V E      S U M M A R Y

MODEL    EX_DEBUG_01
TYPE     MCP
SOLVER   PATH                      FROM LINE  572

**** SOLVER STATUS      1 Normal Completion
**** MODEL STATUS      1 Optimal
```

SOLVE STATUS, MODEL STATUS の値は正常にモデルが解けていることを示している. これはエラーがなく, 正常に解が見つかったということである. しかし, これはモデル, データの正しさを意味するわけではない. 経済学的なモデル・データの正しさと連立方程式が解ける (解を持つ) かどうかは必ずしも関係がないからである.

実際、解を見てみると

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	0.916	+INF	.
---- VAR Y	.	0.993	+INF	.
---- VAR U	.	0.798	+INF	.
---- VAR PX	.	1.147	+INF	1.4409E-8
---- VAR PY	.	1.511	+INF	-1.320E-8
---- VAR PU	.	1.316	+INF	-1.197E-7
---- VAR PK	.	0.859	+INF	1.6208E-8
---- VAR PL	.	1.129	+INF	-2.075E-8
---- VAR CONS	.	210.000	+INF	1.8337E-7

のように初期値 (1) からずれてしまっている (所得変数はそのまます得額が入るので元々 1 にはならない) . これはモデル, データの設定に誤りがあり, ベンチマーク・データが均衡条件を満たしていないということを意味する.

以上のように

- iteration 回数の制限を増やして, 何もショックを与えずにモデルを解く.
  - 出てきた変数の値 (解) が初期値の 1 に等しいかどうかをチェックする
- ということで, モデル, データの正しさをチェックすることができる. これを **cleanup calculation** という.

### 12.3. スケール・ショックによるチェック.

もう一つモデルの正しさをチェックする方法として, 「スケール・ショック」を与えるという方法がある.

MPSGE で記述するモデルは基本的に「規模に関して収穫一定」の技術を持つモデルである (効用関数も含む) . このようなモデルにおいて, 外生的に与えられる賦存量を表すパラメータを比例的に, 例えば X%変化させると

- 全ての数量変数が X%ずつ変化
- 価格変数は変化しない

という結果が導かれるはずである. 逆に言えば, これが成り立たないとするとデータかモデルのどちらかがおかしいことになる. この性質を利用してモデルをチェックするのがスケール・ショックによるチェックである.

(誤った部分は修正しないで) `ex_debug_01.gms` の次の部分の `$exit` をコメントアウトしモ

デルを解く.

```
*      Exit B:
$exit
```

すると、今度は

```
s_l = 1.1;
s_k = 1.1;

$include ex_debug_01.gen
solve ex_debug_01 using mcp;
```

というようにパラメータ  $s_l$  と  $s_k$  を変化させてモデルを解くことになる. この  $s_l$  と  $s_k$  は元々1という値をとるパラメータであり、モデルには  $\$demand$  ブロックに入っている.

```
$demand:cons
  d:pu
  e:pl      q:(e_l*s_l)
  e:pk      q:(e_k*s_k)
```

$s_l$  と  $s_k$  を 1.1 に変えるということは、労働と資本の賦存量を 1.1 倍、つまり 10%増加させて解くということである. この場合、全ての数量変数が 10%上昇、価格変数は不変という結果がでるはずである. 実際に解いてみると、

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	1.009	+INF	.
---- VAR Y	.	1.089	+INF	.
---- VAR U	.	1.048	+INF	.
---- VAR PX	.	1.041	+INF	.
---- VAR PY	.	0.964	+INF	-1.921E-8
---- VAR PU	.	1.002	+INF	-1.332E-8
---- VAR PK	.	1.080	+INF	.
---- VAR PL	.	0.927	+INF	-2.020E-8
---- VAR CONS	.	210.000	+INF	5.8863E-8

となり、期待通りの結果にはならない（誤りを直していないのでこれは当たり前）。

このように収穫一定のモデルの性質を利用してモデルのチェックをすることができる。

**【注】規模の経済性が存在するモデルでは上の性質は元々成り立たないので、この方法による モデル・データのチェックはできない。**

#### 12.4. MPSGE のデバッグオプションによるチェック.

MPSGE には幾つかのデバッグ用のオプションがある（詳しくは MPSGE のマニュアル）。以下では、funlog オプションだけ説明する。

以下、ex\_debug\_02.gms を例にして説明する。funlog オプションをオンにするには以下のようにプログラムを書く。

```
option sysout = on;

$ontext
$model:ex_debug_02

$funlog:.true.
```

つまり、まず sysout オプションをオンにし、その後、\$funlog:に true を設定すればよい。これでモデルを実行すると、次のような情報が LST ファイルに出力される。

```
Function evaluation for: X
```

T	N	PBAR	P	QBAR	Q	KP	ELAS
IA	S	1.0000E+00	1.0000E+00	1.3000E+02	1.3000E+02		0.10
OA	T	1.0000E+00	1.0000E+00	1.3000E+02	1.3000E+02		0.00
IA	VA	1.0000E+00	1.0000E+00	1.1000E+02	1.1000E+02	S	0.50
IA	KR	1.0000E+00	1.0000E+00	8.5000E+01	8.5000E+01	VA	0.10
O	PX	1.0000E+00	1.0000E+00	1.3000E+02	1.3000E+02	T	
I	PL	1.0000E+00	1.0000E+00	2.5000E+01	2.5000E+01	VA	
I	PK	1.0000E+00	1.0000E+00	7.5000E+01	7.5000E+01	KR	
I	PY	1.0000E+00	1.0000E+00	2.0000E+01	2.0000E+01	S	
I	PR	1.0000E+00	1.0000E+00	1.0000E+01	1.0000E+01	KR	

これは x 部門の生産関数がどのように特定化されているかを示している。

Function evaluation for: X							
T	N	PBAR	P	QBAR	Q	KP	ELAS
IA	S	1.0000E+00	1.0000E+00	1.3000E+02	1.3000E+02		0.10
OA	T	1.0000E+00	1.0000E+00	1.3000E+02	1.3000E+02		0.00
IA	VA	1.0000E+00	1.0000E+00	1.1000E+02	1.1000E+02	S	0.50
IA	KR	1.0000E+00	1.0000E+00	8.5000E+01	8.5000E+01	VA	0.10

まず、この部分は CES 関数のツリー構造を表現している。

- 一列目と二列目：
  - ここはネストの数を表す。
  - 投入側 (IA) に 3 つのネストがあり、産出側 (OA) に 1 つのネストがあるということの意味する。
  - 二列目の記号は各ネストの弾力性を指定する記号。
- 右端の列：
  - 右端の列は各弾力性にどのような値が与えられているかを表す。上の例では、s (トップレベルの代替の弾力性) に 0.10, va に 0.5, kr に 0.1, 限界変形率 t に 0 という値が指定されている (ただし、生産物は 1 つなので限界変形率には実質的な意味はない)。
- KP 列
  - ここはネストの構造を表す。ここに一つ上のネストが表示される。
  - 例えば、va に対しては s という記号が KP 列に表示されている。これは va というネストは s の一つ下ということを示す。同様に、kr は va の下にあるということが確認できる。

この情報から自分の意図通りのツリー構造になっているか確認できる。

O	PX	1.0000E+00	1.0000E+00	1.3000E+02	1.3000E+02	T
I	PL	1.0000E+00	1.0000E+00	2.5000E+01	2.5000E+01	VA
I	PK	1.0000E+00	1.0000E+00	7.5000E+01	7.5000E+01	KR
I	PY	1.0000E+00	1.0000E+00	2.0000E+01	2.0000E+01	S
I	PR	1.0000E+00	1.0000E+00	1.0000E+01	1.0000E+01	KR

この部分は各投入物がどのネストに投入されているか、またどのような参照数量、参照価格が指定されているかを表している。例えば、労働は (PL) は va というネストに投入され、その参照数量と価格にはそれぞれ 25 と 1 が指定されている。同様に、資源 (pr) は kr と

いうネストに投入され、参照数量と価格にはそれぞれ 10 と 1 が指定されている。

この情報によって各投入物がどこにどれだけ投入されることになっているかを確認することができる。

### 13. モデル例

- 排出量取引のモデル
- 炭素税のモデル

追加予定. . .

## 参考文献

- 武田史郎 (2008) 「双対アプローチによる一般均衡モデルの記述」 .  
<http://shirotaeda.org/home-ja/research-ja/note-ja.html>
- 武田史郎 (2009a) 「CES 関数のcalibrated share form」 .  
<http://shirotaeda.org/home-ja/research-ja/note-ja.html>
- 武田史郎 (2009b) 「関数型のレファレンス」 .  
<http://shirotaeda.org/home-ja/research-ja/note-ja.html>
- 細江宣裕・我澤賢之・橋本日出男 (2004) 『テキストブック 応用一般均衡モデリング：プログラムからシミュレーションまで』, 東京大学出版.
- Rutherford, Thomas F. (1999) “Applied General Equilibrium Modeling with MPSGE as GAMS Subsystem: An Overview of the Modeling Framework and Syntax,” Computational Economics, Vol. 14, No. 1-2.