MPSGE の利用方法 *

武田史郎 †

Date: 2023 年 5 月 8 日,
Version 2.0

概要

本文書は CGE モデルを解くための GAMS のソルバーである MPSGE (mathematical programming system for general equilibrium) の利用方法の解説書である。

目次

*このファイルの配布場所: https://shirotakeda.github.io/ja/gams-ja/mpsge-ja.html

†所属：京都産業大学経済学部．Website: https://shirotakeda.github.io/ja/

1 概要と準備

1.1 本書の概要

MPSGE とは GAMS (General Algebraic Modeling System) のソルバーの一つである。CGE モデルが利用する一般均衡モデルは数学的には非線形の連立方程式体系として記述される。GAMS において連立方程式を解くには通常モデルを構成する一本一本の数式をプログラムでも記述していく必要がある。

大規模な CGE モデルの場合、複雑な式を多数記述していかなければならないため手間がかかるのと同時に、プログラムの誤りが生じる可能性が非常に高くなる。MPSGE は一般均衡モデルを数式を使わない簡易的な方法で記述するための機能を提供してくれるソルバーである[1)]。

MPSGE を利用することで

- プログラムが非常に簡潔になる。
- プログラムの可読性が高くなり、モデルの構造も把握しやすくなるとともに、モデルのデバッグもしやすくなる。

1) GAMS のソルバーとは通常、ある問題を解くためのアルゴリズムを実装したもののことを指し、そのため解く問題の種類によって区別される。例えば、連立方程式（MCP）を解くためのソルバー、非線形計画法を解くためのソルバー、線形計画法を解くためのソルバーというようにである。MPSGE はソルバーと呼ばれてはいるが、MPC タイプの問題の記述を容易にするための機能のことであるので、通常の意味のソルバーとは異なることに注意して欲しい。

というメリットがある。そのため、GAMS において CGE モデルを記述する際に MPSGE が利用されることが非常に多い。本書はこの MPSGE の利用方法を解説した文書である。

1.2 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ので、CES 型関数について深く理解しておくことが望ましい。CES 型生産関数、及び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以下のページにある『応用一般均衡分析入門』の第 1 章で説明されている。
  - https://shirotakeda.github.io/ja/research-ja/cge-howto.html
- 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[2)]。双対アプローチによる一般均衡モデルの記述についてもやはり上記のページの第 1 章，及び第 2 章で解説されている。
- 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。これについては上記のページの第 8 章で解説されている。
- 上記の事項以外にも、本書では基本的に『応用一般均衡分析入門』の内容を前提として話しを進める。
- 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。
- 本書は GAMS の使い方自体は説明していない。GAMS の使い方についても『応用一般均衡分析入門』を見て欲しい。

1.3 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を紹介しておく。

- MPSGE Guide
  - 公式の MPSGE マニュアル。
  - https://www.gams.com/latest/docs/UG_MPSGE.html
  - 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。
- James Markusen のレクチャーノート
  - James R. Markusen のページ
  - 様々なタイプのモデルのサンプルコードがあり参考になる。ただし、扱われているモデルは小規模なもの（数値例的なもの）で、CGE モデルを扱っているわけではない。
- Thomas Rutherford のページ
  - 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。MPSGE だけではなく、CGE 分析全般の学習に役に立つサイト。

2) 直接、双対アプローチに基づき数式でモデルを記述するわけではないが、MPSGE の内部では双対アプローチでモデルが扱われている。

- https://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、次の表にある **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）	`$sectors`	sector（部門）	利潤最大化（ゼロ利潤条件）
価格変数	`$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

注: 宣言する命令には`$commodities`、`$consumers`、`$auxiliaries` という複数形を用いてもよい（命令として全く同じ意味になる）。`$sectors` のみは複数形を用いなければいけない。

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 が一種類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（厳密には基準化された生産水準）。
  - 通常、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」**と解釈することができる。この考え方に立ち、MPSGE では消費を生産活動の一種として記述することが多い。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）
  - 補助変数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）	$c_i\left(p_1,\ldots,p_m\right)=p_i$	$\rightarrow$	アクティビティー変数 $\{y_i\}$
市場均衡条件	$s_j\left(p_1,\ldots,p_m,y_1,\ldots,y_m\right)$ $=d_j\left(p_1,\ldots,p_m,y_1,\ldots,y_m,k\right)$	$\rightarrow$	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)。

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

つまり以下のような関係がある。

- 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 $\rightarrow$ アクティビティー変数の決定」、「市場均衡条件 $\rightarrow$ 価格変数の決定」、「予算制約条件 $\rightarrow$ 所得変数の決定」

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 MPSGE の記法（準備）

MPSGE では独自の形式でモデルを記述するが、まずその記述方法についての「最低限の一般的なルール」を説明しておく。

通常の GAMS では `$ontext` と `$offtext` で囲まれた部分はコメント部分として解釈され、プログラムとしては何ら意味を持たないが、MPSGE のモデルの記述は以下のように、`$ontext` と `$offtext` の中でおこなうことになっている。

```
$ontext

$model:model_name
...
MPSGE によるモデルの記述
```

```
...
$offtext

$sysinclude mpsgeset model_name
```

そして、一番最初の部分に $model: 命令によって、モデルの名前（上の例では「model_name」）を指定する。

また、$offtext の後に

```
$sysinclude mpsgeset model_name
```

という命令を必ず挿入することも MPSGE での必須の事項である。

以下では、「MPSGE によるモデルの記述」の部分でどのようにモデルを記述していくかを説明する。

2.4 変数宣言の例（ex_decl_01.gms の例）

以下で、ex_decl_01.gms というプログラムを例に、MPSGE でのモデルの記述方法を説明していく。まず、変数の宣言について説明する。

```
*       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

*       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

- アクティビティー変数は「$sectors」命令で宣言される。
  - 例では 3 つの変数が宣言されている。
- 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」がアクティビティー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
	pl	−75	−25		100	0
	pk	−25	−75		100	0
	列和	0	0	0	0	

以下では表 2 のデータを**「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（行の数＝列の数）、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、筆者による『応用一般均衡分析入門』の第 7 章を参照して欲しい。

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 つである。
  - 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。
- **横方向の見方**
  - 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。
- **縦方向の見方**
  - 縦方向では二種類に分かれる。一つが「sector」であり、もう一つは「consumer」である。
  - 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を表す。
    - ゼロ利潤条件より「収入＝費用」が成り立ちので、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。

- 生産された「効用」は消費者が購入することになる。通常は、消費者が財を消費（購入）して効用を得るという形式だが、ここでは 1) 効用部門が財を投入して効用という財を生産し、2) それを消費者が購入して効用を得るという二段階に形式上分割されている。

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 の表列	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。
	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。
	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。

3.2 例 2（ex_prod_08.gms のデータ）

表 4 は ex_prod_08.gms で利用しているデータである。

- 表 2 と異なるのは政府と税金が加えられている部分である。税の導入により、縦方向に税のフローを表す行が加わっている。
- **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。

表 4：政府と税金が加わったデータ（ex_prod_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
	pl	−75	−25			100		0
	pk	−25	−75			100		0
	pg				30		−30	0
税	vtl	−15	−5				20	0
	vtx	−10					10	0
	列和	0	0	0	0	0	0	

- **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、税を表す行である。
  - 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と想定されている。

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 を例にとって説明する。

モデルの概要

- 閉鎖経済 → 貿易はなし。
- 完全競争モデル → 全ての経済主体はプライステイカーとなる。
- 二つの生産部門（x 部門、y 部門）が存在している。
  - 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。

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。
	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。
	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。
	pg の行	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。

    - 生産関数は資本、労働という二つの生産要素の CES 型関数とする。
- 一人の代表的消費者が存在。
    - 効用関数は x 財、y 財の CES 型関数とする。
    - 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と仮定。
- 税金なし、政府部門なし。
- 数式での表現は表 6 にある。

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	$c_y(r,w) = p_x$	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	$u = \frac{cons}{p_u}$	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

- 表 6 はモデルを数式によって表現したものである。
- w と r はそれぞれ労働と資本の価格である。
    - $c_x(r,w)$ と $c_y(r,w)$ はそれぞれ x 部門と y 部門の単位費用関数を表している。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ので、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 財の価格である。
- $c_u(p_x,p_y)$ 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）であり、p_u は効用の価格である。
- 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）市場の均衡条件を意味する。u が効用の供給量、$cons/p_u$ は効用の需要量を表す。

以下は、x 部門の CES 生産関数をツリーで表現したものである。k と l を投入して X を生産する関係を表現している。

```
*         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、労働のみとしている。
- 資本と労働の代替の弾力性は sig_x で与えられている。

以上の生産関数は次のような MPSGE のコードで表現できる。

```
*       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
```

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）、px は x 財の価格、pl は労働の価格、pk は資本の価格を表している。
- “0” 付きのパラメータは全てベンチマークの値を表す。
  - 例えば、x0 は x のベンチマーク・データでの値である。
  - 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である。
- 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。
  - $prod: 命令と同じ行の 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（つまり、s: を指定しない）と、**デフォールト値である「0」が指定**される。つまり、**デフォールトでは** Leontief **型生産関数**となる。
- $prod: ブロック内では次の 3 タイプの行がある。
  - 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である。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である。これは投入物が資本、労働の二つだけということを意味する。
  - 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: フィールドによって**「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」のデフォールト値は 1 である。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 を指定されているので、以下のように p: フィールドを省略しても同じである。

```
*       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）

次は、x 部門の生産関数が 2 段階の CES 型関数であるケースを考える（ex_prod_02.gms というプログラムのモデル）。

```
*       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 型関数になっている。
- 投入物は中間財（y 財）、資本、労働の 3 種類。
- 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 で与えられる。

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_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である。
- i:py が中間財（y 財）の投入を指定している行である。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 型関数である。
- 投入物は、中間財（y 財）、資本、労働、天然資源（r）の 4 つ。
- 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。

この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(va) という書き方は、kr のネストが va と

いうネストの下にあるということを意味する。

- va: のように上位のネストの指定がない場合は、トップレベルの下のネスト（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   l / \ <- 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       l
```

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（結合生産）。
  - 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 で与えられる。

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  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の指定
  - 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である[3)]。
  - 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」についてもデフォールト値は 1 である。よって、1 なら省略可である。
- 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、η 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。

3) 1 生産物のケースで「参照価格」の指定が意味を持たない理由については、5.5 節を見て欲しい。同じことは投入物についても成り立つ。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はない。

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状況を想定する。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。
  - つまり、$(1+t)p$ のような形で税がかかってくるということである。
- 税収は直接消費者に lump-sum で還元されると仮定する。
- 税金以外の部分は ex_prod_01.gms と全く同じとする。

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: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し、消費者の所得変数 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となる。
- 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;
```

[注] 既に「t:」は変形の弾力性を指定するためのフィールドとして出てきたが、全く同じ t: が税率を指定するフィールドとしても用いられるので注意。ただし、変形の弾力性を指定する t: フィールドは一行目（$prod の行）にあるのに対し、税率を指定する t: フィールドは生産物指定行、投入物指定行に現れるという違いがある。

4.6 ベンチマークで投入物への税金が存在するケース （ex_prod_06.gms）

次に、ベンチマーク均衡において既に税金がかかっているケースを考える。

- x 部門の資本、労働投入に税金をかけるのは前と同じであるが、今度は、ベンチマークにおいて既に税金がかかっているとする。
- 他は前のモデルと同じ。

ベンチマークにおいて既に税金がかかっているということは、ベンチマークデータ（SAM）において既に税金が存在しているということである。

表 7：修正された SAM

		sector（部門）			consumer（消費者）	行和
		x	y	u	cons	
commodity（財）	px	115		−115		0
	py		100	−100		0
	pu			215	−215	0
	pk	−25	−75		100	0
	pl	−75	−25		100	0
税	vtk	−10			10	0
	vtl	−5			5	0
	列和	0	0	0	0	

修正された SAM（表 7）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 tkx0、tlx0 とすると、tkx0=10/75、tlx=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: と t: というフィールドの指定は ex_prod_05.gms と全く同じである。
- 異なるのは投入物の参照価格フィールド（p:）の指定である。
- 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を表している。
- 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）

表 8：修正された SAM（生産物に対する税が存在するケース）

		sector（部門）			consumer（消費者）	行和
		x	y	u	cons	
commodity（財）	px	120		−120		0
	py	25	100	−125		0
	pu			245	−245	0
	pk	−35	−75		110	0
	pl	−85	−25		110	0
税	vtx	−20			20	0
	vty	−5			5	0
	列和	0	0	0	0	

表 8 の 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を考える。
- vtx と vty がそれぞれ x 財と y 財に対する税金を表す行である。

- 120 と 25 という x 部門の生産額は市場価格表示。つまり、
  - 120 = 生産者価格表示の生産額 + 生産物税額 = 100 + 20
  - 25 = 生産者価格表示の生産額 + 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、
  - 生産者価格（エージェント価格） = (1 - t)× 市場価格
  - t = 生産税額 / 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、txx0 = 20/120 = 0.1667、txy0 = 5/25 = 0.2 となる。
- 税収は直接消費者に lump-sum で還元されると仮定する。
- 生産物税以外の部分については ex_prod_04.gms と同じである。

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  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である。
  - ただし、t: で指定する税率は上で述べたように「gross base の税率」である。
- 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）

ここまでは税金を考えても、税収は全てそのまま消費者に還元されると想定したため、政府は実質的には存在しなかった。ここでは、政府が税金で集めかお金を独自に支出するケースを考える。表 4 の SAM を前提として考える。

政府支出財部門（g 部門）の生産関数

```
      G
     / \ <- sing_g
    /   \
```

```
   /     \
  x       y
```

政府支出財の生産関数の説明

- 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 のコードの修正点を説明する。まず、アクティビティー変数（sector）の宣言は以下のように修正される。

```
*       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）
```

g 部門のアクティビティー変数（g）が宣言に加わる。
次に価格変数（commodity）の宣言の部分に、政府支出財の価格（pg）の宣言が加わる。

```
*       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
```

さらに、所得変数（consumer）の宣言の部分で、政府が consumer の一人として加わる。政府の所得変数は gov で表現されている。

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

$prod 命令によって、g 部門の生産関数（政府支出財部門）を次のように特定化する。

```
$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をしている。
- ex_prod_08.gms では sig_g = 0 と設定している。これは結局、Leontief 型関数を仮定することになるので、政府支出は各財を固定比率で購入することに使われるという意味になる。

最後に、政府の所得・支出が定義される。

```
$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、税収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 節で詳しく説明する。
- 支出は全て政府支出財に費やされる。税収が変化した分だけ政府支出額が変化することになる。

5 MPSGE の内部動作

ここまででわかるように、MPSGE では直接モデルを数式で表現するのではなく、特殊な記法を用いて表現する。この節では、MPSGE の記法により、MPSGE の内部においてどのようなモデルが作成されているかを生成しているかを説明する。

以下は ex_prod_06.gms における x 部門の特定化のコードである。

```
*       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)  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ことになる。

5.1 Step 1

まず、ベンチマークの単位費用 cx0 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[4)]

```
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)*pkx0)))**(sig_x) * x;
lx = lx0 * ((cx / cx0) / ((1+tlx)*pl/((1+tlx0)*plx0)))**(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

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

4) Calibrated share form については、筆者の『応用一般均衡分析入門』の第 8 章を見て欲しい。

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

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）が一つのときには投入物（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。投入物が一つのとき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、cx0 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 cx0 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。
- $demand ブロックは、「d:」行と「e:」行の二つからなる。
- d: 行（需要指定行）
  - これは所得の使い道を指定する行である。所得によってどの財を購入するのかを指定する。
  - d: の後に購入する財の価格を指定する。
  - 上の例では、消費者は pu（「効用」という財）の購入に支出を利用することになる。
- e: 行（賦存財指定行）
  - これは「賦存財」を定義する行。「賦存財」とは consumer があらかじめ保有している財のことである。
  - e: の後に賦存している財の価格を指定、さらに q: フィールドで賦存量を指定する。
  - 上の例では、消費者（cons）は資本、労働の二つの生産要素を保有しているので、二つの行がある。
  - e:pl が労働の賦存を表す行で、消費者は（e_l*s_l）だけ労働を賦存し供給する。同様に消費者は（e_k*s_k）だけ資本を賦存し供給する。
  - 賦存財供給（要素供給）により消費者（cons）には、pl*e_l*s_l+pk*e_k*s_k だけの所得が入ってくることになる。
  - q: フィールドには「負の値」を指定してよい。この場合、cons が一定量の需要をするということになる。
- 所得の源泉としては税もあるが、税については$prod ブロック内の a: フィールドで指定するので、直接は $demand ブロックには現れない（lump-sum tax の場合は除く）。
- また、$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$demand ブロックで表現することもできる。
    - これについては、ex_dem_01.gms を参照して欲しい。
  - しかし、効用関数が複雑なケースでは$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、この文書では
    - 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する

    という形式を採用している。

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 タイプのみである。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$report 命令を利用することで、値を表示する変数を定義できる。以下、ex_report_01.gms を利用して、$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である。
  - $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 / x0」と定義された変数である。
  - 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。プログラムの意味をわかりやすくするためにも変数の説明を加えておくのがよい。

7.2 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 $prod ブロックの「生産量」、「投入量」、$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。 補助変数を上手く利用することで多様なモデルを作成することができる。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を作成することも、税率、トランスファーを表す変数を補助変数として利用することで可能になる。
- 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（ex_aux_01.gms）

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モデルを考える。政府支出を増加させたときに、それに応じて消費者からの一括税の水準が内生的に調整されるようなモデルのことである。

表 9 の SAM の説明

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

モデルの説明

表 9：修正された SAM（一括税がある SAM）

		sector（部門）				consumer（消費者）	
		x	y	u	g	cons	gov
commodity（財）	px	100		−90	−10		
	py		100	−80	−20		
	pu			170		−170	
	pk	−25	−75			100	
	pl	−75	−25			100	
	pg				30		−30
税	vlst					−30	30

- 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こと。
- 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");
```

一括税を表す補助変数 lst を宣言する。

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

$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」である。
- 消費者側（cons）
  - 消費者側では

    ```
    e:pu      q:(-vlst0)      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。
- 政府側（gov）
  - 政府側では逆に

    ```
    e:pu      q:(vlst0)       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ということ。
  - 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に対する制約式（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（実質の）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こと。

8.3 利用例 2：内生的に変化する労働課税（ex_aux_02.gms）

次に、労働課税が内生的に変化するモデルを考える。

モデル

- 政府支出を一定に保つという条件の下で、一括税の水準を引き下げる代わりに労働課税を引き上げるというケースを考える。
- 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。

モデルの修正

```
*       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部分。
- **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 ブロックの修正

```
*       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 ブロックの修正

```
*       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 は常に一定であるので、g も一定になる。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」のデフォールトの初期値は 1 である。
- 「補助変数」のデフォールトの初期値は 0 である。

もしデフォー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*x0=x*ベンチマークの生産量=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

MPSGE でモデルを解く際のコードの書き方。

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が、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 である。

9.3.2 numeraire（ニュメレール）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

以下では、MPSGE における計算結果の見方について説明する。

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。

以下のように二つの 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を意味する。

```
               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
```

一方、以下のようにどちらも 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を意味する。

```
               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、「LOWER」、「UPPER」、「LEVEL」、「MARGINAL」 の「4 つの値」が関連付けられている。
  - LOWER：これは変数の「下限値」である。MPSGE では変数の下限値のデフォールトは 0 である。ピリオドはゼロを表す。
  - UPPER：これは変数の「上限値」である。MPSGE では変数の上限値のデフォールトは無限大（+inf）である。
  - LEVEL：これが普通の意味での変数の値であり、その変数がその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。

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

```
                        LOWER     LEVEL     UPPER    MARGINAL

-- VAR X                .         0.916     +INF       .
-- VAR Y                .         0.993     +INF       .
-- VAR U                .         0.798     +INF       .
```

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。

2.1 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）

この節で、これまで説明した MPSGE のシンタックスについてまとめる。

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`$sectors:` ブロックに書く。
  - アクティビティー変数と呼んでいるが、当該部門が 1 生産物しか生産しないのなら、アクティビティーレベル＝生産水準（厳密には基準化された生産水準）である。
- `$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ー変数（sector）を指定する。
- 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:
  - **[注]** これは一行目の t: フィールド。
  - この t: では 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必要がある。
  - やはりデフォールト値は 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。
  - デフォールト値は 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

- デフォールト値は 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（こちらは二行目以降にある t: 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 ＋税率）× 市場価格
    - 生産物に対する税：「gross base」→（1 − 税率）× 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ようなときには、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

ex_mpsge_mcp_01.gms は同じモデルを MPSGE と MCP の両方の形式を用いて記述したプログラムである。記述形式は異なるがどちらも同じモデルであり、実際、両者のシミュレーション結果は同じになることが確認できる。MPSGE で記述したモデルがその裏でどのように式の形で表現されるかがわかるので、参考にして欲しい。本当に MPSGE の仕組みを理解するということは、MPSGE で記述したモデルを MCP の形式で表現することができるということである。自分でも MPSGE のモデルを MCP 形式で書き換えることができるようになることが望ましい。

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 Guide や McCarl 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;
```

はモデルを解く際の 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。

```
               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
```

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 MARIGNAL 値は 20。つまり、x 部門で 20 の超過費用が生じている。
- Y の MARIGNAL 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、労働市場で 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 部門で超過費用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 10 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、10 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所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                    .             1.0051        +INF      -5.82645E-13
-- VAR Y                    .             1.0838        +INF      -3.69482E-13
-- VAR U                    .             0.8732        +INF      -1.42109E-13
-- VAR PX                   .             1.1436        +INF      -1.16529E-12
-- VAR PY                   .             1.5149        +INF      7.958079E-13
-- VAR PU                   .             1.3162        +INF      -3.29692E-12
-- VAR PK                   .             0.8512        +INF      -3.95062E-12
-- VAR PL                   .             1.1353        +INF      2.060574E-12
-- VAR CONS                 .           229.8715        +INF      6.593837E-12
```

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)。

このように規模に関して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 を見て欲しい）。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を意味する。
- 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、変形の弾力性 t に 0 という値が指定されている（ただし、生産物は 1 つなので変形の弾力性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 MPSGE における技術進歩の導入方法

生産関数（f）が次のような形式で与えられているとする。

$$y = f(\mathbf{x}) = f(x_1, \cdots, x_n) \tag{1}$$

ただし、y は生産量、x_i は i 財の投入量とする。また、f は一次同次の生産関数とする。

ここで、以下のようなパラメータ λ_y、λ_i を導入する。

$$y = \lambda_y f(\lambda_1 x_1, \cdots, \lambda_n x_n) \tag{2}$$

λ_y や λ_i の値が上昇すると、同じ投入量でより多くの生産量を実現することができる。よって、λ_y や λ_i の上昇を技術進歩と解釈することができる。

マクロ経済学では、投入物が資本（K）と労働（L）のみである以下のような生産関数がよく用いら

れる。

$$Y = \lambda_Y f(\lambda_K K, \lambda_L L) \tag{3}$$

λ_Y、λ_K、λ_L はそれぞれ投入全体についての効率性パラメータ、資本投入についての効率性パラメータ、労働投入についての効率性パラメータを表している。λ_Y は「全要素生産性（total factor productivity）」と表現されることが多い。

上記のような生産関数では以下のような呼び方が使われる。

- λ_Y の上昇：「ヒックス中立的（Hicks neutral）な技術進歩」
- λ_L の上昇：「ハロッド中立的（Harrod neutral）な技術進歩」
- λ_K の上昇：「ソロー中立的（Solow neutral）な技術進歩」

CGE モデルでは、投入物には本源的生産要素だけではなく、中間財も含まれることになるが、λ のようなパラメータによって技術進歩を表現することはよくある。以下では、λ の変化を MPSGE で導入する方法について説明する。

MPSGE での実装を考えるので、生産関数は次のような CES 型を想定する。

$$y = \lambda_y \left[\sum_i \beta_i (\lambda_i x_i)^{\frac{\sigma-1}{\sigma}} \right]^{\frac{\sigma}{\sigma-1}} \tag{4}$$

13.1 数式による表現

生産関数が (4) 式で与えられるとき、それに対応する calibrated share form の生産関数は次式となる[5)]。

$$y = \lambda_y \bar{y} \left[\sum_i \theta_i \left(\frac{\lambda_i x_i}{\bar{x}_i} \right)^{\frac{\sigma-1}{\sigma}} \right]^{\frac{\sigma}{\sigma-1}} = \lambda_y \bar{y} \left[\sum_i \theta_i \left(\frac{x_i}{\bar{x}_i / \lambda_i} \right)^{\frac{\sigma-1}{\sigma}} \right]^{\frac{\sigma}{\sigma-1}} \tag{5}$$

ただし、バー付きの変数はベンチマークデータの値であり、さらに λ は初期時点では全て 1 という値をとると仮定している。

13.2 MPSGE における実装

今、仮に投入物は 2 個のみとすると、技術進歩パラメータ（λ）がないときの (5) 式に対応する MPSGE のコードは次のような形になる。

```
$prod:y  s:sig
    o:py        q:y_0
    i:px1       q:x1_0  p:px1_0
    i:px2       q:x2_0  p:px2_0
```

ただし、0 付きのパラメータはベンチマーク値を表している。

5) これについては、『応用一般均衡分析入門』の第 A-1 章を見て欲しい。

13.3 ヒックス中立的な技術進歩

「ヒックス中立的な技術進歩」、つまり λ_y の上昇を入れるには次のように修正すればよい。

```
$prod:y  s:sig
    o:py        q:(lambda_y*y_0)
    i:px1       q:x1_0          p:px1_0
    i:px2       q:x2_0          p:px2_0
```

(5) 式を見ればわかるように、λ_y はベンチマークの生産量 $\bar{y}$ にかかる形で入ってきている。よって、MPSGE のコードでは同じようにベンチマークの生産量を指定する参照数量 `y_0` にかかるように入れればよい。

13.4 個々の投入物についての技術進歩

次に個々の投入物についての技術進歩、つまり λ_i の上昇であるが、(5) を見ると λ_i はベンチマークの投入量 $\bar{x}_i$ を割る形で入っている。よって、以下のように導入すればよいように思える。

```
$prod:y  s:sig
    o:py        q:(lambda_y*y_0)
    i:px1       q:(x1_0/lambda_1)  p:px1_0
    i:px2       q:(x2_0/lambda_2)  p:px2_0
```

この記述は一見正しいように思えるが、実はこれでは λ_i の変化のみを導入したことにはならない。これは MPSGE のカリブレーション方法を理解するとわかる。

第 5 節で見たように、MPSGE は参照数量と参照価格を用いて、ベンチマークの投入シェアの値を計算する仕組みになっている。上のような指定では投入シェアは次のように計算される。

$$\theta_i = \frac{\bar{p}_i^x \bar{x}_i / \lambda_i}{\sum_j \bar{p}_j^x \bar{x}_j / \lambda_j}$$

この式の右辺は明らかに λ_i の値に依存している。よって、λ_i の変化にともない、投入シェア θ_i も変化してしまうことになる。しかし、ここで考えたいのは純粋に λ_i の変化の効果なので、ベンチマークの投入シェアは固定しておかなければいけない。これが上のような記述が問題になる理由である。

それでは、投入シェアは固定したまま、λ_i の変化の影響のみを考えるにはどうすればよいかと言うと、以下のように記述すればよい。

```
$prod:y  s:sig
    o:py        q:(lambda_y*y_0)
    i:px1       q:(x1_0/lambda_1)  p:(px1_0*lambda_1)
    i:px2       q:(x2_0/lambda_2)  p:(px2_0*lambda_2)
```

つまり、投入物の参照価格の部分に逆にかける形で λ_i を入れてやればよい。こうすると、

$$\theta_i = \frac{\bar{p}_i^x \lambda_i \bar{x}_i / \lambda_i}{\sum_j \bar{p}_j^x \lambda_j \bar{x}_j / \lambda_j} = \frac{\bar{p}_i^x \bar{x}_i}{\sum_j \bar{p}_j^x \bar{x}_j}$$

となり、投入シェアは λ_i には依存しなくなるからである。

参照価格に技術進歩パラメータを入れるのはおかしいように思えるかもしれないが、第 5 節で説明したように、参照価格というのは単にベンチマークにおける投入シェアを計算することに使われるだけであるので、投入シェアが正しく計算されるのなら、他は特に問題ない。

13.5 プログラム例

`ex_tech_01.gms` が技術進歩を導入したプログラムの例である。モデルやデータは基本的に `ex_prod_01.gms` のものと同じである。`ex_tech_01.gms` ではまず基準均衡を求めたあとに、次の 5 つのシナリオを解いている。

- s1: 10%のヒックス中立的な技術進歩
- s2: 10%のハロッド中立的な技術進歩
- s3: 10%のソロー中立的な技術進歩
- s4: s2 + s3
- s5: 10%のハロッド中立的な技術進歩 + 20%のソロー中立的な技術進歩
- s6: s2 で、参照価格は変化させないケース
- s7: s5 で、参照価格は変化させないケース

どのケースでも技術進歩は x 部門のみで生じると仮定している。

s4 は実質的に s1 と同じであるので、この二つのケースは同じ結果になるはずである（チェックしてみて欲しい)。また、前節までで説明したように、投入側で技術進歩が生じる場合には、参照価格にも技術進歩パラメータを入れる必要があるが、それをしない場合を s6、s7 で解いている。s2 と s5 とは結果が異なってくるはずである。

14 その他

14.1 MPSGE で扱えるモデル

MPSGE を利用することで CGE モデルを非常に簡単に記述できるようになる。従って、CGE 分析をおこなう際には積極的に MPSGE を利用するのがよいが、MPSGE を用いることの欠点もある。それは、MPSGE は記述できるモデルに制限があるということである。**MPSGE は、基本的に生産関数が CES 型であるモデルを前提としている**。従って、フレキシブルな生産関数や固定要素が存在するようなモデルを MPSGE で記述することは難しい。同様に、効用関数についても CES 関数を前提としており、その他の関数型を記述することはできない。

15　モデル例

この節では MPSGE を使って記述したモデルの例を 3 つ紹介する。

1)　これまでのモデルをまとめたモデル。
2)　排出量取引による排出規制を考慮したモデル。
3)　炭素税による排出規制のモデル。

15.1　これまでのモデルをまとめたモデル

表 10：ex_full_01.gms のデータ

		sector（部門）					consumer（消費者）		行和
		x	y	z	u	g	cons	gov	
commodity（財）	px	140	−10		−110	−20			0
	py	−15	115		−80	−20			0
	pz	5		15	−20				0
	pu				210		−210		0
	pk	−30	−75	−5			110		0
	pl	−75	−25	−10		110			0
	pg					40		−40	0
税	vtls						−10	10	0
	vtl	−15	−15					20	0
	vtx	−10						10	0
	列和	0	0	0	0	0	0	0	0

ex_full_01.gms というプログラムのモデル

- x 部門、y 部門、z 部門の三部門が存在する。
- y 部門、z 部門はそれぞれ y 財、z 財を生産するが、x 部門は x 財に加え、z 財も生産している（結合生産）。
- u は効用を生産する部門、g は政府支出財を生産する部門である。
- 生産要素は労働と資本の二つ。
- 税
    - 税は消費者への一括税、労働課税、x 部門への生産税の 3 つを想定している。

ex_full_01.gms のシミュレーションの説明

- ex_full_01.gms では以下のシナリオを分析している。
- C1: 賦存量の比例的増加
  - 生産要素の賦存量、政府支出を比例的に増加させるシミュレーション。
  - このシミュレーションでは、全ての数量変数が同じだけ比例的に変化する結果になるはず。
- C2: 労働課税率を 2 倍に上昇
  - 労働課税率を 2 倍に上昇させるシミュレーション。
  - 消費者の効用はどう変化するか？
- C3: Uniform な労働税率に変更
  - ベンチマークでは各部門における労働課税率は異なっている。これを全ての部門に対し一様な税率に変更するシミュレーション。
  - 政府支出の水準が変化しないように税率を設定。
  - 効用はどう変化するか？改善するか？
- C4: 一括税以外の税を撤廃
  - 一括税以外の税を撤廃する (税率をゼロにする) シミュレーション。
  - 効用はどう変化するか？改善するか？

シミュレーションで実際にどのような結果になるかは、自分で ex_full_01.gms を実行して確認してみて欲しい。

15.2 排出量取引による排出規制を考慮したモデル

CO2 の排出があり、その CO2 の排出を排出量取引制度（キャップアンドトレード、以下 C&T）によって削減するモデル。

15.2.1 モデルの概要

モデルとベンチマーク・データ

- モデルとベンチマーク・データには ex_full_01.gms と同じようなものを利用。
- ただし、以下の点が異なる。
  - 労働、資本に加え、資源という生産要素を加える。資源の価格は pr という変数で表す。資源は z 部門のみで利用される。
  - このモデルでは z 財を化石燃料とみなし、z 部門を化石燃料部門とする。よって、z 財の投入や消費にともない CO_2 が発生することになる。

15.2.2 排出規制

- z 財の投入・消費からは CO_2 が排出される。
  - z 財の排出係数を δ で表す。よって、「CO_2 排出量＝ $\delta \times z$ 財の量」となる。
  - ただし、ここの CO_2 排出はあくまで数値例にすぎないため、シミュレーションでは簡単化のため「$\delta = 1$」と置いている。

- 排出規制としてはキャップ・アンド・トレード型の排出量取引を利用する。
- 1 単位の z 財を利用（投入、消費）するには δ 単位の排出権（排出枠）が必要になる。よって、1 単位の z 財を利用することの費用 $\left(c_z^A\right)$ は p_z を z 財の価格、p_{CO2} を排出権の価格とすると、次のように表せる。

$$c_z^A = p_z + p_{\mathrm{CO2}}\,\delta \tag{6}$$

- 排出権の供給はキャップとして政府が外生的に設定する。排出権価格は排出権の需要と供給が等しくなるように決まる。

15.2.3 排出権のモデル化

排出権取引を導入するということは (6) 式の関係を導入するということである。これは次のような `$prod` ブロックを導入することで実現されている。

[コード A]

```
*       排出権
$prod:azc  s:0
    o:pzc  q:z0
    i:p("z")    q:z0     p:p0("z")
    i:pco2      q:co20   p:1e-6
```

コードの説明

- `pzc` が排出権への支払い込みの z 財の価格（6 式の p_z^A）、`p("z")` は z 財の価格（6 式の p_z）、`pco2` は排出権の価格（6 式の p_{CO2}）である。
- `z0` はベンチマークにおける z 財の総生産量、`co20` はベンチマークにおける CO_2 排出量であり、δ =`co20/z0` である。
- この `azc` という部門は z 財と排出権を投入して、排出権への支払い込みの z 財を生産するというような生産活動を表している。
- 排出権の参照価格
  - 排出権の参照価格（`i:pco2` の行の参照価格）には「`1e-6` $= 10^{-6} = 0.000001$」という値が指定されている。
  - 初期均衡では排出規制がないため排出権価格は 0 であるが、0 を参照価格にすることは意味がないので、0 に近い非常に小さな値を指定している。
- 上記の`$prod` ブロックの指定により排出権への支払い込みの z 財の価格（単位費用）は次のように特定化される（これについて詳しくは第 5 節を参照して欲しい）。

$$c_z = \bar{c}_z\left(\theta_z\frac{p_z}{\bar{p}_z} + \theta_{\mathrm{CO2}}\frac{p_{\mathrm{CO2}}}{\bar{p}_{\mathrm{CO2}}}\right) = \bar{c}_z\left(\theta_z\frac{p_z}{\bar{p}_z} + \theta_{\mathrm{CO2}}\frac{p_{\mathrm{CO2}}}{0.000001}\right) \tag{7}$$

ただし、

$$\bar{c}_z = \bar{p}_z + 0.000001 \times \delta$$
$$\theta_z = \frac{\bar{p}_z}{\bar{p}_z + 0.000001 \times \delta}$$
$$\theta_{\mathrm{CO2}} = \frac{0.000001 \times \delta}{\bar{p}_z + 0.000001 \times \delta}$$

であり、$\bar{p}_{\mathrm{CO2}} = 0.000001$、$\delta$=co20/z0 という関係を用いている。

- この単位費用を用いて、azc 部門のゼロ利潤条件（価格=単位費用）を次式のように表現できる。

$$p_z^A = c_z$$

- 以上のように特定化される (7) 式は実は (6) 式と全く同じ関係を表している。これは c_z を次のように書き換えることができるからである。

$$\begin{aligned} c_z &= (\bar{p}_z + 0.000001 \times \delta)\left[\theta_z \frac{p_z}{\bar{p}_z} + \theta_{\mathrm{CO2}} \frac{p_{\mathrm{CO2}}}{0.000001}\right] \\ &= \left[\bar{p}_z \frac{p_z}{\bar{p}_z} + 0.000001 \times \delta \frac{p_{\mathrm{CO2}}}{0.000001}\right] \\ &= p_z + p_{\mathrm{CO2}}\,\delta \end{aligned}$$

- 上の記号を使うと排出権に対する需要は「co20*azc」と表現できる。
- 注: 実際のベンチマーク均衡における単位費用は

$$\bar{c}_z = \bar{p}_z$$

であるが、コード A の特定化では

$$\bar{c}_z = \bar{p}_z + 0.000001 \times \delta$$

となる。下の関係を想定すると厳密には均衡条件は満たされないが、第二項目の部分はほぼゼロになるので結局大きな問題は生じない（これが排出権の参照価格に非常に小さい値を指定する理由である）

$prod ブロックの意味

- azc の $prod ブロックでは z 財と排出権が投入されているが、その代替は 0 に設定されている。つまり、z 財と排出権が Leontief 型関数を通じて投入されるという形式になっている。
- 排出権取引の下では、化石燃料の投入量に応じて排出権を比例的に購入しなければならない。その関係が固定係数の生産関数という形で表現されている。

政府の予算制約の指定

```
*         政府の予算制約
$demand:gov
```

```
    d:pg
    e:pg        q:(vtlump0*s_lump)  r:t_lump$fl_lump
    e:pco2      q:co2lim
```

- 排出権は政府が発行し、排出権の売却収入は政府が得る。
- co2lim は排出権の発行量（供給量）であり、賦存財として政府の所得（gov）の中に入ってきている。

x 部門の$prod ブロック

```
*       部門 x の生産関数
$prod:q("x")  s:0  lkz:sig_1("x")  kz(lkz):sig_2("x")
    o:p("x")    q:q0("x")   p:((1-tq0("x"))*p0("x"))  a:gov
+   t:tq("x")$(not fl_qx)   n:t_qx$fl_qx  m:tq0("x")$fl_qx
    i:p("y")    q:int0("y","x")    p:p0("y")
    i:pl        q:ld0("x")  p:((1+tl0("x"))*pl0)  lkz:  a:gov  t:tl("x")
    i:pk        q:kd0("x")  p:pk0  kz:
    i:pzc       q:int0("z","x")    p:pzc0  kz:
```

- 上は x 部門の$prod ブロックである。
- z 財の投入については排出権価格込みの価格（pzc）を用いて記述している。y 部門や u 部門についても同様である。

排出権収入の扱い

- 排出権取引を導入すると政府には排出権収入が入ってくる。この排出権収入をどう扱うかを考える必要がある。
- ex_permit.gms では二つの処理の方法を考慮している。
  - A-1）排出権収入の分だけ消費者の一括税の額を減税する。
  - A-2）排出権収入の分だけ x 部門にかかっている生産税を減税する。
- A-1 について
  - ベンチマーク均衡では消費者に一括税が課されている。A-1 は排出権収入が入る分だけこれを減税するということである。これは排出権収入をそのまま一括で消費者に渡しているのと同じである。
  - 厳密には、排出権収入分だけ一括税を減税するというわけではなく、（実質の）政府支出（g）がベンチマークの値と同じ値を保つように一括税の額を調整している。
- A-2 について
  - ベンチマーク均衡では x 部門の生産に税が課されている。A-2 は排出権収入が入る分だけこれを減税するということである。

  - 厳密には、排出権収入分だけ生産税を減税するというわけではなく、（実質の）政府支出（g）がベンチマークの値と同じ値を保つように生産税を減税するということをおこなっている。
  - 元々、生産物への税は経済に歪みをもたらす効果を持っている。従って、その生産物税を減税することで経済の効率性が改善する。排出規制自体は経済には負担になるが、減税による経済効率の改善の効果が十分大きければ、排出規制の導入でかえって消費者の効用が上昇するということがありうる。これがいわゆる**「二重の配当」**である[6)]。
  - シミュレーションでは実際に二重の配当が生じるかどうかを確認してみる。
- **コード**
  - A-1 の導入方法は `ex_aux_01.gms` での内生的な一括税の導入方法と全く同じである。
    - 一括税が内生的に変化するように`$demand` ブロックに rationing instrument を入れる。
  - A-2 の方法は `ex_aux_02.gms` での方法を使っている（ただし、労働課税ではなく、生産税を内生的に変化させているが）。
    - x 部門の生産税に `n:` フィールドと `m:` フィールドを用いて、内生的な生産税を導入している（`m:` フィールドについては第 10.3 節を参照して欲しい）。

pco2 変数の初期値の設定

```
*         変数の初期値
pco2.l = 0;
```

ベンチマーク均衡では、排出権の供給量（co2lim）に非常に大きい値を設定しておく。この結果、「排出権への需要<排出権の供給」となり、排出権の価格は 0 とならなければいけない。このため上記のコードでは排出権価格の初期値に 0 を設定している。

15.2.4 シミュレーション結果

表 11 はシミュレーションのシナリオである。ベンチマーク均衡に加え、C1〜C4 の均衡を求める。

表 11：シミュレーションのシナリオ

シナリオ名	内容
C1	CO_2 を 2%削減 + 排出権収入分だけ一括税を減税
C2	CO_2 を 5%削減 + 排出権収入分だけ生産税を減税
C3	CO_2 を 2%削減 + 排出権収入分だけ一括税を減税
C4	CO_2 を 5%削減 + 排出権収入分だけ生産税を減税

- 表 12 はシミュレーションの結果を表している。

6) 「二重の配当」については、筆者が執筆した環境経済・政策学会 (2018) の「二重の配当」の項、あるいは Takeda (2007) を参考にして欲しい。

表 12：シミュレーション結果

	C1	C2	C3	C4
CO2	−5.000	−10.000	−5.000	−10.000
PCO2	0.207	0.373	0.194	0.362
t_lump	0.103	−0.528	1.000	1.000
t_qx	1.000	1.000	0.038	−0.671
U	−0.029	−0.258	0.369	0.320

- CO2 は CO_2 排出量のベンチマーク均衡からの変化率（%）、PCO2 は（効用の価格でデフレートした）排出権価格（プログラムにおける `pco2/pu` の値）、t_lump、t_qx はそれぞれベンチマーク均衡での値を 1 と基準化した一括税、x 部門への生産税の水準である。
- 最後の U は効用水準のベンチマーク均衡からの変化率（%）を表している。

シミュレーション結果

- 削減率が高くなるほど排出権の価格は高くなっていることがわかる。
- C1 では一括税の水準がそれぞれ約 90%低下している。
- C2 では排出権収入が大きく増加するので、一括税ではなく、家計への一括のトランスファー（マイナスの一括税）になっている。
- C3 では生産税の水準が約 97%低下している。
- C4 では排出権収入が大きく増加するので、生産税ではなく、生産補助金（マイナスの生産税）になっている。
- 一括税の軽減のケースでは効用が減少している。
- 一方、C3、C4 では CO_2 を削減しているにもかかわらず効用が上昇している。これはいわゆる**「(強い) 二重の配当」**が生じているということである。C1、C2 のケースが示すように、CO_2 の削減自体は効用にマイナスの影響をもたらすが、同時に歪みがある税を軽減することでそのプラスの効果が働く。さらに、このケースではプラスの効果がマイナスの効果を上回り、効用が上昇するという結果になっている。
- **注**
  - この数値例のシミュレーションでは強い二重の配当が生じたが、実際の CGE 分析では二重の配当が生じる可能性はそれほど高いわけではない。

15.3 炭素税による排出規制を考慮したモデル

15.3.1 モデルの概要

- 次に炭素税による排出規制のモデル（`ex_carbon.gms`）を提示する。
- 基本的には `ex_permit.gms` と同じであるが、排出規制の部分のみ異なっている。
- 同じ排出規制であっても、理論的な観点からは、排出権取引よりも炭素税の方がよりオーソドック

スで単純なものと思えるかもしれない。しかし、MPSGE でのモデル化という観点ではむしろ排出権取引の方が単純であり、炭素税はトリッキーな方法を用いる必要がある。その理由は**「炭素税は一種の従量税」**であるためである。

- これまで MPSGE で税を扱う方法を説明してきたが、全て従価税であった。これは MPSGE では直接扱えるのは従価税のみだからである。
- しかし、**「補助変数（auxiliary variable）＋制約式（$constraint 命令）」**を上手く利用することで、MPSGE でも従量税を扱うことができる。以下でもその方法を用いて、炭素税を導入する。

15.3.2 炭素税のモデル化方法

1 単位の z 財を利用（投入、消費）するには δ 単位の排出権（排出枠）が必要になる。よって、1 単位の z 財を利用することの費用 $\left(p_z^A\right)$ は、p_z を z 財の価格、t_{CO2} を炭素税率（CO_2 一単位あたりの税額）とすると、次のように表せる。

$$p_z^A = p_z + \delta t_{\mathrm{CO2}} \tag{8}$$

炭素税は「CO_2 一単位に対して○円」という形式をとるのでこのような従量税の形になる。しかし、MPSGE では

$$p_z^A = (1 + \tau_{\mathrm{CO2}})p_z \tag{9}$$

のような従価税しか扱えない。これが MPSGE では炭素税をそのままでは扱うことができない理由である。

MPSGE で炭素税を扱うには、元々は (8) 式のような従量税の形式を (9) 式の形式に変換する必要がある。(8) の形式の税が (9) の形式の税と同じ税を意味するには、以下の関係が成り立っていなければならない。

$$\tau_{\mathrm{CO2}}\, p_z = \delta t_{\mathrm{CO2}}$$

t_{CO2} を外生的に与えたときに、この式が満たされるように τ_{CO2} を調整すれば実質的に (8) 式の税となる。この式を書き直すと

$$\frac{\tau_{\mathrm{CO2}}\, p_z}{p_u} = \delta \frac{t_{\mathrm{CO2}}}{p_u} \tag{10}$$

となる。

この (10) 式を制約式としてモデルに導入することで、炭素税の形式の税を導入することができる。具体的には、「実質の炭素税率 (t_{CO2}/p_u)」を外生的に設定し、(10) 式によって従価税率 (t_{CO2}) を内生的に変化させる」という方法をとる。

注：実質の炭素税率を外生的に設定する理由

- 「実質の炭素税率 (t_{CO2}/p_u)」を外生的に設定すると説明したが、なぜ炭素税 $= t_{\mathrm{CO2}}$ そのものでは

なく、実質の炭素税率なのか疑問に思うかもしれない。つまり、

$$\tau_{\mathrm{CO2}}\, p_z = \delta t_{\mathrm{CO2}} \tag{11}$$

で、t_{CO2} を外生的に設定するようにすればよいのではないかということである。

- しかし、(11) の関係を導入することは問題がある。(11) 式は p_z という絶対価格に対する制約であるため、(11) **式ではモデルにおいて価格についての 0 次同次性が満たされなくなってしまう**からである（価格についての 0 次同次性については『応用一般均衡分析入門』の第 2 章を見てほしい）。一方、(10) 式は p_z/p_u という相対価格に対する制約であるため、価格についての 0 次同次性を損なうことはない。
- 価格についての 0 次同次性が満たされないモデルでは、ニュメレールを変えると均衡が変わってしまうということが起こる。通常そのようなモデルは望ましくないので、(11) 式ではなく、(10) 式を利用する。

15.3.3 コードの説明

まず、以下のように t_co2 という補助変数を宣言しておく。これは z 財にかかる従価税である。

```
*       補助変数の宣言
$auxiliary:
    t_co2               ! Carbon tax rate
    t_lump$fl_lump      ! Adjustment by lump-sum tax
    t_qx$fl_qx          ! Adjustment by output tax
```

azc 部門で z 財にその従価税がかかるようにする。内生的に変化する従価税であるので、n: フィールドで導入する。

```
$prod:azc  s:0
    o:pzc  q:z0
    i:p("z")    q:z0     p:p0("z")  a:gov  n:t_co2
```

最後に$constraint 命令で (10) 式の関係を導入する。この関係によって t_co2 の水準が決まるようにする。tco2 は外生的に与える実質炭素税率を表すパラメータである。

```
$constraint:t_co2
    t_co2*p("z") / (pu*delta) =e= tco2;
```

15.3.4 炭素税（価格政策）と排出権取引（数量政策）の同値性

- 環境経済学に限らず、経済学では様々な分野において、**「価格規制と数量規制の同値性」**が成り立つことが多い。この節で利用している CGE モデルでも、価格規制（炭素税）と数量規制（排出権

取引）の間で同値性が成り立つ。つまり、炭素税と排出権取引のどちらでも同じ均衡を実現することができるということである。

- もう少し厳密に言うと、
  - 排出量に X という量のキャップをかけ、排出権取引のモデルで均衡を計算する → その均衡を A と呼ぶ。
  - 均衡 A で実現した実質排出権価格を炭素税のモデルにおいて実質炭素税率に設定し、均衡を求める → その均衡を B と呼ぶ。
  - 「均衡 B における CO_2 排出量＝元々のキャップ X」が成り立ち、さらにその他の変数についても均衡 A と均衡 B の値は等しい。

  ということである。
- `ex_carbon.gms` では実際に上の同値性が成り立つかを計算している。その際、`ex_permit.gms` で計算された実質排出権価格を `ex_carbon.gms` では実質排出税率として利用している。

16 分析例の紹介

追加予定。

参考文献

Takeda, S. (2007) “The Double Dividend from Carbon Regulations in Japan,” *Journal of the Japanese and International Economies*, Vol. 21, No. 3, pp. 336–364, DOI: 10.1016/j.jjie.2006.01.002.

環境経済・政策学会（編） (2018) 『環境経済・政策事典』, 丸善出版.