応用一般均衡分析入門
第 11 章：NLP としての一般均衡モデル [*]

武田 史郎 [†]

Date: 2023/01/27,
Version 3.0

目次

1 導入

今回の内容について。

[*]このファイルの配布場所: https://shirotakeda.github.io/ja/research-ja/cge-howto.html

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

- ここまで一般均衡モデル（あるいは CGE モデル）を解くのに GAMS の「MCP（mixed complementarity problem） **ソルバー**」を使ってきた。
- 一般均衡モデルを解くとは通常均衡条件式から構成される連立方程式を解くことであるので、連立方程式を解くためのソルバーである MCP ソルバーを利用することが普通のアプローチと言える[1)]。
- しかし、一般均衡モデルを 「NLP（non-linear programming、**非線型計画法**）」、つまり非線形の最適化問題として設定し、NLP ソルバーを利用して解くという方法もある。この第 11 章では一般均衡モデルを NLP として解くアプローチについて説明する。
- MCP として解くアプローチを使うのなら、第 11 章の内容を読む必要はないが、NLP として解くというアプローチを使っている研究者も多い。そのようなモデルを理解するにはこの第 11 章の内容を理解しておく必要がある。

2 NLP としての一般均衡モデル

2.1 MCP による表現

これまで見た通り、一般均衡モデルとは結局のところ均衡条件式（利潤最大化条件、効用最大化条件、市場均衡条件等）から構成される非線形の連立方程式体系である[2)]。従って、モデルの内生変数を $\{x_1, \cdots, x_n\}$、外生変数を $\{a_1, \cdots, a_m\}$ としたとき一般均衡モデルは一般に次のように表現できる。

$$
\begin{aligned}
&f_1(x_1, x_2, \cdots, x_n; a_1, \cdots, a_m) = 0 \\
&f_2(x_1, x_2, \cdots, x_n; a_1, \cdots, a_m) = 0 \\
&\cdots \\
&f_n(x_1, x_2, \cdots, x_n; a_1, \cdots, a_m) = 0
\end{aligned}
$$

つまり、n 本の方程式から構成される連立方程式として表現される。

$\mathbf{f}$ を関数 f_i が要素の $n \times 1$ ベクトル、$\mathbf{x} = \{x_1, \cdots, x_n\}$ を $n \times 1$ ベクトル、$\mathbf{0} = \{0, \cdots, 0\}$ を $n \times 1$ ベクトル、$\mathbf{a} = \{a_1, \cdots, a_m\}$ を $m \times 1$ ベクトルとすると、上の方程式体系は次のように表現できる。

$$
\mathbf{f}(\mathbf{x}; \mathbf{a}) = \mathbf{0} \tag{1}
$$

ここで、外生変数 $\mathbf{a}$ を決めてやれば n 本の式から n 個の内生変数 $\mathbf{x} = \{x_1, \cdots, x_n\}$ が決定されるということになる[3)]。一般均衡モデルを MCP ソルバーで解くというのは、上のように連立方程

1) 厳密には、MCP ソルバーは MCP を解くためのソルバーであるが、連立方程式は MCP の一種であるので、MCP ソルバーを使て解くことができる。MCP について詳しくは第 4 章を参照して欲しい。
2) 実際には、単純な連立方程式ではなく MCP となっているケースもある。
3) 式が独立でないようなケースは除く。

式として解くということであり、実際、これまでそのようにして解いてきた。

2.2 NLP による表現

一般均衡モデル（or CGE モデル）を NLP（非線形の最適化問題）として解くには次のような方法をとる。まず、モデルに (1) 式のモデルとは無関係なダミーの変数 y を導入する。そして、以下のような最適化問題を設定する。

$$\begin{aligned} \max_{x_1,\cdots,x_n} \quad & y \\ \text{s.t.} \quad & y = 1 \\ & \mathbf{f}(\mathbf{x};\mathbf{a}) = \mathbf{0} \end{aligned}$$

つまり、制約条件として $\mathbf{f}(\mathbf{x};\mathbf{a}) = \mathbf{0}$ と $y = 1$ を設定し、その下で変数 y を最大化するように $\mathbf{x} = \{x_1, \cdots, x_n\}$ を選択するという問題である。この最適化問題を解くと、(1) 式をそのまま連立方程式として解いたときと同じ解を求められる。

通常の最適化問題では、制約条件を満たす内生変数は一意（unique）には決まらないが（だからこそ、その中から目的関数を最大化するものを選べるのだが)、ここでは「式 f の数= 変数 x の数」であるので、変数 y とは無関係に $\mathbf{f}(\mathbf{x};\mathbf{a}) = \mathbf{0}$ を満す $\mathbf{x}$ はユニークに決まることになる。従って、$\mathbf{f}(\mathbf{x};\mathbf{a}) = \mathbf{0}$ という連立方程式を解くのと同じことになる。つまり、最適化問題を解くという形式をとりながら、実質的には連立方程式を解くという作業をおこなっていることになる。実際これを解いた場合、最適化された y の値も形式上は導出されるが、y はモデルとは無関係のダミーの変数であるので、単に無視すればよいだけである（しかも、それは $y = 1$ という条件より最初から 1 になることは明白である)。

2.3 NLP のよる表現のプログラム例

benchmark_replication_check_mcp.gms は第 9 章で利用した benchmark_replication_check.gms と全く同じプログラムである。そして、benchmark_replication_check_nlp.gms は、benchmark_replication_check_mcp.gms を NLP のモデルとして書き直したプログラムである。以下、NLP で記述したときどの部分が変更されているかを説明する。

まず、以下のようにダミーの変数 obj が追加されている（上の説明での変数 y にあたるもの)。

```
variables
     obj                ダミー変数
```

さらに、目的関数を意味するダミーの式 e_obj も追加されている。

```
equations
     e_obj          目的関数（ダミー）
```

式の定義の部分で e_obj の定義が追加されている。

```
*         目的関数（ダミー）
e_obj .. obj =e= 1;
```

上の式では obj は 1 に等しいとしているが、それ以外の数でも全然かまわない。

第 9 章の benchmark_replication_check_mcp.gms ではモデルは次のように宣言・定義されていた。

```
*       ------------------------------------------------------------
*       モデルの宣言
$ontext
+ MCP タイプのモデルとしてモデルを定義する。
+ 各式に対してその式に対応する変数を指定する。

$offtext
model ge_sample_dual 一般均衡モデル（双対アプローチ）
      / e_c.c, e_c_va.c_va, e_y.y, e_v_a.v_a, e_a_x.a_x, e_a_v.a_v,
      e_a_f.a_f, e_e.e, e_d.d, e_p.p, e_p_va.p_va, e_p_f.p_f, e_u.u,
      e_m.m, e_x.x, e_vf.vf /;
```

MCP のモデルとして定義するには、モデルを構成する式を列挙するだけではなく、その式に対応する変数を指定する必要があった[4)]。

一方、NLP のモデルとして定義するには次のように書く。

```
*       ------------------------------------------------------------
*       モデルの宣言
$ontext
+ NLP タイプのモデルとしてモデルを定義する。

$offtext
model model_nlp NLP 形式のモデル
      / e_c, e_c_va, e_y, e_v_a, e_a_x, e_a_v, e_a_f, e_e, e_d, e_p,
      e_p_va, e_p_f, e_u, e_m, e_x, e_vf, e_obj /;
```

つまり、単にモデルを構成する式の名前を列挙するだけでよい。この意味で NLP のモデルの方が記述するのが簡単であると言える。

モデルを解く部分は次のように修正される。

4) 詳しくは第 4 章を参照されたい。

```
option nlp = conopt;
solve model_nlp using nlp maximizing obj;
```

MCP モデルのときには、「`solve utility_max using mcp;`」というような書き方であったが、NLP モデルの場合は少し異なる。NLP モデルの場合はどの変数を最大（あるいは最小）にするかを「`maximizing obj`」の部分で指定する形になる。NLP ソルバーにはいくつか存在するが、ここでは `option nlp = conopt;` として CONOPT を指定している。他には MINOS、PATHNLP などがある。

NLP のモデルでも、モデルを構成する式は基本的に MCP とは変わらない。変わるのは

- 最大化（あるいは最小化）するダミーの変数を追加
- ダミーの目的関数を追加
- モデルの宣言・定義を修正（変数の指定は除去）
- Solve 命令の修正

というような部分だけである。

3 NLP 形式を用いるときのモデルのデバッグ方法

第 9 章では MCP 形式で記述した CGE モデルのデバッグ方法について説明した。そこでは、デバッグの際に**「変数の marginal 値」**の情報を利用した。MCP のモデルでは変数の marginal 値はその変数が関連付けられた式の乖離幅を表していた。

例えば、

```
*       財の市場均衡
e_p(i) .. y(i) =e= sum(j, a_x(j,i)*y(j)) + d(i);
```

という `e_p(i)` という式に対して、変数 `p(i)` という変数が関連付けられているときには、変数 `p(i)` の marginal 値である `p.m(i)` は

```
p.m(i) = y(i) - sum(j, a_x(j,i)*y(j)) - d(i)
```

を表している。

このように変数の marginal 値が式の乖離幅を表すようになっているのは、MCP のモデルの定義において式と変数の対応関係が指定されているからである。一方、NLP のモデルでは式と変数の対応関係の指定がない。そのため上記のように変数の marginal 値が式の乖離幅を表現するような関係がない。よって、NLP のモデルではデバッグの際に MCP のモデルのような手順をとることができない。しかし、NLP のモデルでは全く同じことを別の情報を見ることで行うことができる。

それは**「式の値」**の情報である。以下、NLP のモデルにおけるデバッグ方法について説明する。

3.1 GAMS における式の値の意味

GAMS では各変数に対して、LEVEL、LOWER、UPPER、MARGINAL の 4 つの値が存在した。同様に、式に対しても LEVEL、LOWER、UPPER、MARGINAL の値がある。この式の値を見ることで、式の乖離幅という情報を得ることができる。式の LEVEL 値、LOWER 値、UPPER 値はそれぞれ次のような値を表している（MARGINAL 値はデバッグには利用しないので以下では省略する）。

表 1：式の値

	LOWER	UPPER	LEVEL
=e=	rhs	rhs	変数パートの値
=l=	-Inf	rhs	変数パートの値
=g=	rhs	Inf	変数パートの値

式の値は表 1 の通りである。

- LEVEL 値
  - LEVEL 値が表すのは常にその式の**「変数パート」**の値である。
  - **変数パートとは（**variable **命令で宣言される）内生変数を含む部分**のことである。変数パートの値は内生変数を含む部分を全て式の左辺に移項した形で表現される。
- LOWER 値と UPPER 値
  - 一方、LOWER 値、UPPER 値が表す値は式において「=e=」、「=l=」、「=g=」のどれが利用されているかで変わってくる。
  - 例えば、式で =e= が利用されているのなら、LOWER 値、UPPER 値とも式の rhs を表すことになる（よって、二つは等しい値をとる）。
  - 「rhs」
    - rhs は式の**「定数パート」**の値のことである。
    - **定数パートとは定数（**parameter **や** scalar **で定義されるもの）を含む部分**のことである。定数パートの値は定数を含む部分を全て式の右辺に移項した形で表現される。
    - 実際には rhs という記号が表示されるわけではなく、定数パートの値が表示される。
  - 式で「=l=」が利用されているときには、LOWER は-Inf、UPPER は rhs を表す[5)]。
  - 式で「=g=」が利用されているときには、LOWER は rhs、UPPER は Inf を表す。

以下、例を使って式の値について説明する。

3.1.1 例 1:

```
e_1 .. x + 2*y + 10 =e= 30;
```

5) Inf はプラス無限大、-Inf はマイナス無限大である。

x と y を変数とし、上のように定義された式において、x.l=20、y.l=5 であるとする。このとき、式 e_1 の LEVEL 値、LOWER 値、UPPER 値は次の値となる。

```
e_1.l = 20 + 2*5 = 30
e_1.lo = 30 - 10 = 20
e_1.up = 30 - 10 = 20

                      LOWER     LEVEL     UPPER    MARGINAL

---- EQU e_1          20.000    30.000    20.000       .    INFES
```

3.1.2 例 2

```
     e_2 .. x =e= y * (a / z)**(b);
```

以上の式で、x、y、z は変数、a と b はパラメータとし、

```
x.l = 20
y.l = 2
z.l = 2
a = 6
b = 2
```

であるとすると、式の値は以下のようになる。

```
e_2.l = 20 - 2*(6/2)**(2) = 2
e_2.lo = 0
e_2.up = 0

                      LOWER     LEVEL     UPPER    MARGINAL

---- EQU e_2            .         2.000      .          .    INFES
```

定数パート（rhs）はないので UPPER と LOWER はゼロである。

3.1.3 例 3

```
     e_3 .. a*x +  b*y =l= c;
```

において、x と y は変数、a、b、c はパラメータとし、

```
x.l = 10
y.l = 20
a = 6
b = 2
c = 100
```

とすると、

```
e_3.l = 6*10 + 2*20 = 100
e_2.lo = -inf
e_2.up = c = 100


                      LOWER     LEVEL     UPPER    MARGINAL

---- EQU e_3           -INF    100.000   100.000      .
```

となる（=l= が利用されているので、LOWER は「-inf」となる）。

3.1.4 式に対する値（まとめ）

式に対する値の定義より、もしその式が等号で満たされているなら

- LEVEL 値と LOWER 値が等しい
- LEVEL 値と UPPER 値が等しい

の少なくともどちらか一方が満たされているということになる。言い換えれば、**式の LEVEL 値が LOWER 値とも UPPER 値とも異なっているのなら、式は等号では満たされていない（つまり、式の右辺と左辺は等しくない）ということ**になる。この情報を元に第 9 章で行なったようなモデルのデバッグを NLP でも行なうことができる。

3.2 NLP モデルのデバッグの例

第 9 章でデバッグに利用した benchmark_replication_check_mcp.gms を NLP のモデルに書き直した benchmark_replication_check_nlp.gms を例にして、NLP モデルでのデバッグを行なってみる。含まれているバグは benchmark_replication_check_mcp.gms と全く同じである。まずは benchmark replication によるチェックを行なう。よって、MCP のときと同様にモデルの iterlim を 0 に設定してモデルを解く。

```
model_nlp.iterlim = 0;
```

```
solve model_nlp using nlp maximizing obj;
```

MCP モデルのときには計算結果の変数の値をチェックしていったが、NLP モデルの場合は式の値をチェックしていく。まず、上から見ていくと e_c の値は次のような値になっている。

```
---- EQU e_c  生産の単位費用

       LOWER     LEVEL     UPPER    MARGINAL

agr        .     2.000        .         EPS  INFES
man        .     2.000        .         EPS  INFES
ser        .     2.000        .         EPS  INFES
```

LOWER と UPPER の値は共に 0 であるのに対し、LEVEL 値は 2 である。3.1 節で説明したように、LEVEL 値が LOWER 値とも UPPER 値とも等しくなっていないので、これは e_c が等号で満たされていないということを意味する。

第 9 章で見たように、この原因は c(i) の初期値の設定がおかしいことにある（詳しいことは第 9 章の説明を参照されたい)。そこを修正して、解き直すと

```
---- EQU e_c  生産の単位費用

       LOWER     LEVEL     UPPER    MARGINAL

agr        .         .         .         EPS
man        .         .         .         EPS
ser        .         .         .         EPS
```

となり、LEVEL 値は LOWER 値とも UPPER 値とも等しくなり、e_c は等号で満たされるようになる。

続けて式をチェックしていくと、e_a_v が以下のような値になっている。

```
---- EQU e_a_v  単位合成生産要素需要

       LOWER     LEVEL     UPPER    MARGINAL

agr    10.000        .    10.000         EPS  INFES
man    10.000        .    10.000         EPS  INFES
ser    10.000        .    10.000         EPS  INFES
```

LEVEL 値が 0 であるのに対し、LOWER 値と UPPER 値はともに 10 であり LEVEL 値と等し

くないので、やはり式が等号で満たされていない。この原因は e_a_v の定義において、余計なパラメータが加えられていることにあった。それを除去して、解き直すと次のようになる。

```
---- EQU e_a_v  単位合成生産要素需要

       LOWER     LEVEL     UPPER    MARGINAL

agr        .         .         .         EPS
man        .         .         .         EPS
ser        .         .         .         EPS
```

3 つの値が全て等しくなり、e_a_v が等号で満たされるようになった。

あとのパターンは同じである。第 9 章では変数の MARGINAL 値の情報でバグを見つけていったが、ここでは式の値を用いることを除いて同じようにすればよい。benchmark_replication_check_nlp.gms から benchmark replication で見つけられるバグを除去したものが benchmark_replication_check_corrected_nlp.gms である。

4 MCP vs NLP

第 2 節で見たように、MCP でも NLP でも同じようなモデルを解くことができた。また、第 3 節で見たように、どちらでも同じようにモデルのデバッグをすることもできる。それでは結局どちらの形式でモデルを記述することが望ましく、どちらのソルバーを利用することが望ましいであろうか。ここでは両者の比較を行ないたい。

話を進める前に、考えるケースを整理しておく。まず次の 2 つのケースが考えられる。

- ケース 1：モデルがそもそも MCP の形式でしか表現できない。
- ケース 2：モデルが連立方程式で表現できる。

ケース 1 のように MCP の形式でしかモデルが表現できない場合には、NLP に変換するということできず、そのまま MCP として扱うしかない。また、ソルバーとしては MCP ソルバーが必要になる。一方、ケース 2 の場合には複数の選択がある[6)]。

- ケース 2-1：連立方程式であるが、それを MCP ソルバーで解くように MCP の形式にそって記述する。
- ケース 2-2：モデルを NLP に変換し、NLP ソルバーで解く。

ケース 2 の場合、モデルを MCP の一種として扱い、MCP ソルバーで解くか、モデルを NLP に変換して、NLP ソルバーで解くかの二つの選択肢がある。これらの点について、以下でもう少し詳しく説明したい。

6) ここでは二つしか挙げないが、5 節で説明するように、CNS に変換して、CNS ソルバーで解くという選択もある。

4.1 モデルが MCP のケース（ケース 1）

ここまで扱ってきた一般均衡モデル（CGE モデル）は全て均衡条件式が等号で表現されるようなモデルであった。この場合はこれまで説明したように、MCP の一種として記述するのも、NLP として記述するのもどちらも可能である。しかし、ケース 1 のように、モデルが連立方程式ではなく、均衡条件式が不等号を含む MCP の形式となる場合もある。

MCP について詳しくは第 4 章を見て欲しいが、MCP が何かを簡単に復習しておこう。まずこれまでのモデルでは、部門 i の価格を p_i、単位費用を c_i としたとき、利潤最大化条件は

$$c_i - p_i = 0$$

と表現され、この式を満たすように生産量 q_i が決定されたと考えた。しかし、実際には部門 i の生産量が 0 になってしまうケースも考えられる。そのような場合には、第 4 章で見たように利潤最大化条件は

$$c_i - p_i \geq 0 \qquad q_i \geq 0 \qquad (c_i - p_i)q_i = 0 \tag{2}$$

というように修正される。このように均衡条件式に不等号、及び complementary slackness 条件（2 式の 3 番目の式）が含まれる式が入ってくる場合が MCP であった。MCP を NLP に変換して解くということは難しいので、MCP を解くには基本的にはそのまま MCP として扱う必要がある[7)]。そして、ソルバーとしては MCP ソルバーを用意する必要がある。この場合は、記述形式やソルバーを選択するような余地はない。

このケース 1 のような状況、つまりモデルが単なる連立方程式ではなく、MCP の形式になってしまうことが CGE 分析において多いかというと、実際にはあまり多いわけではない。ただ、温暖化対策の CGE 分析では、モデルが MCP の形式になる場合がしばしばある。例えば、温暖化対策の分析では新しい技術が重要になるが、そういった新しい技術を導入する際に MCP の形式で記述する必要が出てくる場合がある。現在の状況では採算が合わないが、電気料金が数倍に上昇したら導入されるような発電技術のようなケースである。このような技術は当初 $c_i - p_i > 0$ であるので、$q_i = 0$ で均衡しているが、p_i が上昇してくるため、ある段階で $c_i - p_i = 0$ となり $q_i > 0$ と変わることになる。このような状況をモデルに導入するには、連立方程式ではだめで、MCP として記述する必要がある。

CGE モデルが MCP にならないことが多い理由

上記のように、CGE 分析ではモデルが MCP の形式になることは多くない。つまり、生産量、投入量、消費量等がゼロになってしまうケースが少ない。この理由は生産関数や効用関数に CES

7) 連立方程式を NLP に変換したのと同じように、MCP を NLP に変換する方法もあるのかもしれないが、筆者にはよくわからない。

関数を利用することが多いためである。生産関数や効用関数が CES 関数である場合には、投入量や消費量がゼロになることはまれである[8)]。

4.2 モデルの定義部分の記述という観点

ケース 1 の場合には選択の余地はないが、ケース 2 の場合には、どちらの記述、ソルバーを選択するか（ケース 2-1 のように対応するか、ケース 2-2 のように対応するか）が問題になる。一つの判断基準がモデルの記述がどちらが容易かという点である。まず、MCP **モデルではモデルの定義（宣言）部分で式を列挙するだけではなく、式と変数の対応関係も指定する必要があった**。それに対して、NLP **モデルでは式を列挙するだけですむ**。よって、記述の容易さという点では NLP モデルの方が優れていると言える。

しかし、筆者は手間は増えるとしても、MCP モデルの記述の方が望ましいと考える。それは、MCP モデルの記述のように式と変数の対応関係を意識することで、モデルの構造を深く理解できるからである。MCP モデルでは式と変数の対応関係を指定するが、その対応は当然しっかりした意味に基づいておこなう必要がある。対応の方法については第 2 章で説明したように基本的には次のようにおこなえばよい。

- 何らかの変数を定義する式なら、それによって定義される変数。例えば、生産関数なら、それによって定義される生産量の変数
- 市場均衡条件なら、それによって決まる財・生産要素の価格
- 利潤最大化条件、費用最小化条件などの最適化行動のための条件なら、その条件に基づいて決定される変数

式の数が多いときには、一個一個の式と変数を対応させていくことは繁雑な作業になるが、そうすることでモデルの構造をしっかり考えることにもなり、それはモデルのデバッグ作業を容易にする。

逆に言えば、NLP モデルの場合には全体としての式の数と変数の数しか考える必要がないため、個々の条件式と変数の関係に対する意識が低くなりやすい。また、モデルが間違っているにもかかわらず、たまたま全体としての式の数と変数の数が一致してしまうケースでは間違いを見付けにくくなってしまう。

筆者はよく CGE 分析についての質問を受ける。多くの質問は自分でモデルを作ったが、上手く動かないという質問である。上手く動かないというモデルは多くの場合 NLP で記述されている。このようにモデルの構造が理解できていない要因の一つが NLP モデルの記述方法にあると考えられる。

4.3 ソルバー（購入）の選択

GAMS で大規模な CGE モデルを解くとすると、有料のソルバーを購入する必要がある。モデルを MCP として記述して解くのなら MCP ソルバー、NLP として解くのなら NLP ソルバーを購入する必要がある。**資金に余裕があるのなら、MCP ソルバーと NLP ソルバーの両方を購入すれば**

8) 元々、投入量や消費量がゼロである財は別であるが。

よい。しかし、資金に余裕がないのなら、どちらか片方しか購入できないかもしれない。その場合、どちらを購入するべきかを考えたい。

先程、既に述べたようにケース 1 のような場合にはもう選択を考える必要はなく、MCP ソルバー（PATH）を購入するしかない。問題はケース 2 のケースであるが、実は現在の GAMS のソルバーは一つのソルバーで複数のタイプの問題を解くことができるケースが多い。例えば、PATH と CONOPT というソルバーは次の種類の問題を解ける。

- Solver PATH が扱える問題 → MCP、CNS、NLP[9)]
- Solver CONOPT が扱える問題 → NLP、LP、CNS、DNLP、QCP

このうち、CNS（constrained non-linear system）は MCP とよく似ており、要するに「連立方程式を解くという問題」と考えてよい（詳しくは 5 節で説明する）。PATH は MCP がメインのソルバー、CONOPT は NLP がメインのソルバーであるが、PATH を使っても、CONOPT を使っても、連立方程式も NLP もどちらも解くことができるということである。その意味で、どちらのソルバーを使っても両方の問題に対処できると言える。

ただし、MCP を解けるのは PATH の方だけなので、NLP、CNS だけではなく MCP も扱いたいなら PATH を選択するということになるだろう。一方、扱う CGE モデルが MCP ではなく単なる連立方程式の形式であり、最適化問題として NLP だけではなく LP 等も解きたいというような場合には、CONOPT を選択するのがよい。NLP と CNS さえ解ければよいというのなら、PATH も CONOPT を扱えるので、どちらのソルバーでもよいということになるが、同じタイプの問題を扱えると言っても、ソルバーの性能は異なる。例えば、GTAP8inGAMS にはサンプルとして CNS としてモデルを解くプログラムの両方が付いている。このサンプルのプログラムはソルバー PATH では解けるが、CONOPT で解けない。このように同じタイプの問題を扱えるといっても、ソルバーによって解けたり解けなかったりするので、どちらでもいいということにはならない。

一応、筆者の意見を書いておくと

- 扱う主な問題は CGE モデルを解くという問題
- LP（線形計画法）を解く機会はない

という人はソルバー PATH を購入するのがよいのではないかと思う。一方、

- 扱う主な問題は NLP
- LP を解く機会もあり
- CGE モデルを解くことはそれほど多くなく、解くとしてもそれほど大きなモデルではない

以上のような人は、ソルバー CONOPT を購入するのがいいかと思う。

9) 正確には PATH を購入すると付属してくる、PATHNLP を使うと NLP も解けるということである。

5 CNS としての一般均衡モデル

連立方程式（system of nonlinear equations）を解くという問題を NLP に変換して解く方法を紹介したが、GAMS ではもう一つ CNS として解くという方法もある。

CNS（constrained nonlinear system）とは次のような問題である（"Constrained Nonlinear System (CNS)"）。

$$\begin{aligned} \text{Find} \quad & \mathbf{x} = \{x_1, \cdots, x_n\} & & \\ \text{s.t.} \quad & F_i(\mathbf{x}) = 0 & i = \{1, \cdots, n\} & \quad (3) \\ & L_i \leq x_i \leq U_i & i = \{1, \cdots, n\} & \quad (4) \\ & G_j(\mathbf{x}.) \leq b_j & & \quad (5) \end{aligned}$$

ただし、F_i は非線形の関数である。(3) 式の部分だけ考えればスクウェア（$n \times n$） の非線形の連立方程式（system of nonlinear equations）であるが、それに加えて変数に制約が付加されているのが CNS の特徴である。しかし、(4) 式、(5) 式の制約は解において binding になることは意図されていない。制約式はあくまで解をある範囲に制限すること、及び F_i が定義されない範囲に入ることを防ぐためのものである。よって、実質的には「CNS ＝非線形の連立方程式」と考えればよい。

GAMS で解く場合に、ダミーの変数を加えて NLP として解く方法と比較し、CNS として解く方法は次のような利点がある。

- CNS ではモデルがスクウェアになっているかのチェックがおこなわれる
- ソルバーが解やモデルについての情報を表示してくれる
- 解く時間が短かくなる可能性がある

以上のような理由から、NLP に変換して NLP ソルバーで解くのではなく、CNS として解くケースが増えてきた。例えば、

- PEP モデル Robichaud et al. (2012) [10)]

では CNS を使って解くというプログラムも含まれている。CONOPT や PATH 等のソルバーは CNS も扱えて、しかも NLP に変換するよりも CNS として解く方がプログラムも簡単であるので、以下で CNS として解く方法についても、一応簡単に説明しておく。

`benchmark_replication_check_cns.gms` が `benchmark_replication_check_mcp.gms` を CNS のモデルとして書き換えたものである。変更点は非常に少ない。まず、CNS にしてもモデルの式の記述自体は全く変わらない。変わるのは、モデルの定義の部分である。モデルの定義の部分では CNS は NLP と同様に単に式を列挙するだけにする。

10) https://www.pep-net.org/pep-standard-cge-models [2022-01-20 アクセス]

```
 model model_cns CNS 形式のモデル
      / e_c, e_c_va, e_y, e_v_a, e_a_x, e_a_v, e_a_f, e_e, e_d, e_p,
      e_p_va, e_p_f, e_u, e_m, e_x, e_vf /;
```

MCP のモデルではニュメレールの指定を次のようにおこなっていた。

```
*       ------------------------------------------------------------
*       ニュメレールの指定
p.fx("agr") = 1;
```

こうすると p.fx("agr") の値は 1 に固定される。さらに、この p("agr") が対応付けられている式 e_p("agr")（これは AGR の市場均衡条件式）がモデルから除去される。

一方、CNS では式と変数の対応を指定しないため、p.fx("agr") = 1; と設定しても式が 1 本取り除かれるというようなことがおきない。このため**「内生変数の数＝式の数 − 1」**となってしまいモデルがスクウェアではなくなってしまう。この場合、CNS ではモデルのエラーとなり、次のようなメッセージが出る。

```
**** CNS models must be square
```

CNS では「内生変数の数＝式の数」となるように p.fx("agr") = 1; という指定はとる必要がある。あるいは、それは残した上で、モデルの定義の部分において AGR の市場均衡条件式を除去するように自分で書き換えればよい。

モデルを解く部分では次のように記述する。

```
* option cns = path;
option cns = conopt;
solve model_cns using cns;
```

モデルのデバッグは NLP の場合と同じ方法でおこなえばよい。

6 終わりに

この章では、CGE モデルを NLP として解く方法について説明した。後の章では CGE モデルを MCP として解いていくが、NLP として解く方法を利用している研究者も少なからずいるので、理解しておくとよい。

参考文献

Robichaud, V., A. Lemelin, H. Maisonnave, and V. Decaluwé (2012). *No PEP-1-1 A User Guide*. Tech. rep. URL: http://www.pep-net.org/programs/mpia/pep-standard-cge-models/ (cit. on p. 14).

7 履歴

- 2022-01-16: 説明の追加・修正。
- 2018-07-20: 説明の追加・修正。
- 2017-03-15: 説明の修正。
- 2015-02-08: 最初に作成。