CGE 分析入門
第 9 章：CGE モデルのチェック *

武田史郎 †

Date: 2017/04/06,
Version 1.1

目 次

1 導入

今回の内容について

- CGE モデルを構築し、そのプログラムを書くという作業をおこなう際に、何の間違いもなく、正常なモデルをスムーズに作成できるというようなことはまずない。モデルの構造、デー

*ファイルの配布場所: http://shirotakeda.org/ja/research-ja/cge-howto.html
†京都産業大学経済学部. Website: http://shirotakeda.org/ja/

タ、プログラム等、様々な部分において誤りが混入するのが普通である。その原因は、概念、方法論、理論に関する誤りの場合もあれば、データ作成の際のミス、プログラムを記述する際のミス等、単純な不注意から生じる場合もある。

- いずれにせよ、様々な誤りをチェックし、正常なシミュレーションのプログラムを作成していくという作業が CGE 分析において非常に重要な部分を占めているのであるが、CGE 分析という手法自体があまり知られていないこともあり、モデルやデータをチェックする方法も十分には理解されていない。このため、場合によっては、誤ったモデルやデータを用いてシミュレーションがおこなわれていることもある。

- 筆者はよく CGE 分析についての相談や質問を受ける。その内容のほとんどは、「自分でモデル（プログラム）を作ってみたが上手く解けない」、あるいは「自分でモデル（プログラム）を作ってみたが、結果がどうもおかしい」というものがほとんどである。つまり、一応、自分で見よう見まねでモデル（プログラム）は作れるのであるが、それが上手く動かないときに「どうチェックすればよいのか？」、「どうやって問題を見つければよいのか？」ということがわからないということである。

- 第 9 章では、CGE モデルをチェックし、その誤り（バグ）を修正する（デバッグする）方法・手順について説明する[1]。CGE 分析をおこなうにあたって必須の従業な作業であるので、よく理解することが望ましい。具体的には、まず Benchmark replication によるチェックと Cleanup calculateion によるチェックを紹介する。これは CGE モデル特有のモデルのチェック・デバッグ方法である。さらに、ニュメレールによるチェック、外生変数の比例変化によるチェック、 ワルラス法則のチェック等、第 6 章で紹介した一般均衡モデルのチェック方法を再びここでも取り上げる。

2 Benchmark Replication

まず、「**benchmark replication（以下、BR）**」によるモデルのチェックを説明する。CGE モデルでは、「ベンチマーク・データの下で均衡が成立している」という前提を置き、パラメータが設定される。従って、最初にモデルを解いたときに、ベンチマークデータがそのままモデルの解に一致していなければならない。BR とは実際にそれが成り立っていることを意味する[2]。BR が成立しない、つまり、ベンチマークデータがモデルの解に一致していないとすると、それはモデルかデータに間違いがあるということになる。よって、benchamrk replication が成立するかどうかということで、モデル、データに誤り（バグ）が含まれるかどうかをチェックすることができる。さらに、この BR によるチェックではモデル、データのどこにバグがあるかを見つけることもできる。CGE 分析におけるモデル、データのチェック・デバッグの最も基本的な方法であり、自分で CGE 分析をおこなう際には必ず BR によるチェックをおこなうことが望ましい。

[1] バグ（bug）とはプログラムの誤り、欠陥のこと。デバッグ（debug）とはバグを修正することである。元々はコンピュータ用語で、主にプログラムの間違いに関連する用語であるが、ここではプログラムの間違いに限らず、間違い全般を指す用語として使う。

[2] ここではベンチマークデータがモデルの解に一致していることを「benchmark replication」と呼ぶが、この用語が一般に幅広く利用されているわけではないので注意。

2.1 Benchmark Replication チェックの方法

GAMS において MCP タイプのモデルを解く場合、通常、GAMS は解が見つかるまで内部で一定の計算を繰り返す仕組みになっている[3]。BR によるチェックをおこなうには、この繰り返し（iteration）の回数を 0 に設定して解けばよい。繰り返しの回数が 0 に設定されている場合、GAMS は変数の初期値が解となっているかどうかだけ判断して終了する。これにより、初期値（ベンチマーク・データ）が解になっているかどうかをチェックすることができる。

2.2 例

以下では、benchmark_replication_check.gms を例に使っていく。中身のモデルは第 8 章で利用した calibration_example.gms のモデルとほぼ同じである。モデルについては 第 5 章、SAM については 第 7 章、パラメータのカリブレーションについては 第 8 章を見るようにして欲しい。

このプログラムで BR チェックをしてみる。このプログラムでは、モデル名は ge_sample_dual となっている。この名前のモデルを解く際の繰り返し（iteration）回数を 0 にするには、solve 命令の前に

```
ge_sample_dual.iterlim = 0;
```

というコードを付け足せばよい。modelname.iterlim = xxx という命令によって、「modelname」という名前のモデルを解くときの iteration 回数を「xxx 回」に設定することができる。

この設定でモデルを解くと、次のような結果が出力されるはずである。

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

     MODEL   ge_sample_dual
     TYPE    MCP
     SOLVER  PATH                FROM LINE  372

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

 RESOURCE USAGE, LIMIT          0.016      1000.000
 ITERATION COUNT, LIMIT         0             0
 EVALUATION ERRORS              0             0
```

まず、

```
 ITERATION COUNT, LIMIT         0             0
```

の部分の数値が iteration の回数である。右側が「iteration の最大の回数」、左側が「実際に iteration が行なわれた回数」である。上で iteration 回数の limit を 0 と設定したので、どちらの数値も 0

[3]GAMS の MCP ソルバー（PATH や MILES）はニュートン法に類似するアルゴリズムでモデルを解いている。

になっている。

上の solve summary では、**「solver status」も「model status」も共に 1 以外の数値**になっている。これはモデルが正常に解けておらず、変数の初期値（ベンチマークデータ）が均衡条件を満たしていないことを意味している[4]。つまり、BR ができていないということである。これは結局、モデルかデータのどちらかがおかしいということを示している。

以下では、この計算結果を基にモデルのバグを修正（デバッグ）していく作業をおこなう。まず、計算結果における変数の値を初めから順番に見ていく。すると、変数 c の部分が次のような結果となっている。

```
-- VAR c  生産の単位費用

            LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6          3.0000         +INF            2.0000
man   1.0000000E-6          3.0000         +INF            2.0000
ser   1.0000000E-6          3.0000         +INF            2.0000
```

第 3 章で説明したように、MCP のモデルでは

「変数の marginal 値」＝「その変数が関連付けられた式の乖離幅（「左辺－右辺」の値）」

であった。変数 c はモデルの定義の部分で e_c.c というように、式 e_c に関連付けられている。従って、変数 c の marginal 値は式 e_c の乖離幅を表すことになる。式 e_c は次のように定義されていた。

```
*       生産の単位費用
e_c(i) .. c(i) =e=
          (sum(j, alpha_x(j,i)**sig(i) * p(j)**(1-sig(i)))
              + alpha_v(i)**sig(i) * p_va(i)**(1-sig(i)))**(1/(1-sig(i)));
```

よって、変数 c の marginal 値（c.m）は

```
c(i) - (sum(j, alpha_x(j,i)**sig(i) * p(j)**(1-sig(i)))
       + alpha_v(i)**sig(i) * p_va(i)**(1-sig(i)))**(1/(1-sig(i)))
```

を表すことになる。もし e_c が条件式の通り等号で満たされているとすると、この値はゼロになっていなければならないのであるが、上の marginal 値の結果はそれが成り立っていないことを示している。

式 e_c の定義自体がおかしい可能性もあるが、上の e_c の定義を見ても特におかしいところはなさそうである（モデル自体については 第 5 章を見るようにして欲しい）。そこで、次のようなコードを solve 命令の後（前でもいいが）に付け足して、e_c の左辺と右辺の値をチェックしてみる。

[4] MCP のモデルでは、solver status、model status とも 1 になることが、モデルが正常に解けていることを意味する。

```
parameter  chk_e_c;

chk_e_c(i,"LHS") = c.l(i);
chk_e_c(i,"RHS") =
    (sum(j, alpha_x(j,i)**sig(i) * p.l(j)**(1-sig(i)))
        + alpha_v(i)**sig(i) * p_va.l(i)**(1-sig(i)))**(1/(1-sig(i)));
display chk_e_c;
```

chk_e_c(i,"LHS") には e_c の左辺の値、chk_e_c(i,"RHS") には e_c の右辺の値が代入される。式が条件通り等号で満たされるなら、両者は等しい値になっていなければらない。display 命令で表示される chk_e_c の中身は次のような値になる。

```
--    381 PARAMETER chk_e_c

             LHS         RHS

agr        3.000       1.000
man        3.000       1.000
ser        3.000       1.000
```

e_c は単位費用の定義式で、左辺、右辺とも単位費用の値を表すのであるから両辺とも 1 の値をとるはずである。右辺は適切な値をとっているが、左辺、つまり変数 c(i) の値がおかしいことがわかる[5]。変数 c は solve 命令より少し前の部分で初期値を与えられているので、そこをチェックしてみる。すると、

```
c.l(i) = c0(i) + 2;
```

という記述になっている。本来の初期値 c0(i) に 2 が加えられている。これが e_c の LHS の値がおかしくなっている原因であることがわかる。ここを c.l(i) = c0(i); という正しい式に書き換え、モデルを解き直し、再び c の marginal 値をチェックしてみる。

```
-- VAR c  生産の単位費用

            LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6          1.0000        +INF             .
man   1.0000000E-6          1.0000        +INF             .
ser   1.0000000E-6          1.0000        +INF             .
```

今度は marginal 値が 0 になり、式 e_c が等号で満たされるようになったことがわかる。

BR によるモデルのチェックの基本的な考え方は以上の通りである。

1). 変数の marginal 値をチェックして、それが 0 になっているかを確認する。

[5] これはモデルやデータの誤りではなく、変数の初期値の与え方の誤りである。

2). もし 0 になっていないのなら、その変数が関連付けられた式のどこかがおかしいということになる。

3). 式の定義を確認してみる。定義が間違っているのなら式を修正する。

4). 式の定義がおかしくなければ、式の右辺、左辺の値をチェックし、どちらの値がおかしいかを調べる。

5). さらに、式に含まれるパラメータや変数の値をチェックし、どの部分の値がおかしいかをより詳しく見ていく。

以上のような手順で問題点を探していけばよい。以下、実際にこの手順に従いモデルのデバッグをおこなってみる。

続けて変数の値をチェックしていくと、今度は a_v の marginal 値が 0 になっていない。

```
-- VAR a_v  単位合成生産要素需要

           LOWER          LEVEL          UPPER         MARGINAL

agr           .             0.5714       +INF          -10.0000
man           .             0.6667       +INF          -10.0000
ser           .             0.5333       +INF          -10.0000
```

そこでまず、変数 a_v が関連付けられている式 e_a_v をチェックする。それは次のように定義されている。

```
*        単位合成生産要素需要
e_a_v(i) .. a_v(i) =e= (alpha_v(i) * c(i)/p_va(i))**(sig(i)) + err1(i);
```

余計なパラメータ err1(i) が右辺に足し合わされており、明らかに式の定義がおかしい。式からこの err1(i) を除去して、解き直してみると次の結果を得る。

```
-- VAR a_v  単位合成生産要素需要

           LOWER          LEVEL          UPPER         MARGINAL

agr           .             0.5714       +INF             .
man           .             0.6667       +INF             .
ser           .             0.5333       +INF             .
```

今度は a_v の marginal 値は全て 0 になっており、e_a_v が等号で満たされるようになった。

同様に変数をチェックを続けていく。次に問題があるのは変数 a_f である。

```
-- VAR a_f  単位生産要素需要
```

```
                LOWER          LEVEL          UPPER         MARGINAL

lab.agr           .            20.6250        +INF           20.0000
lab.man           .            20.4000        +INF           20.0000
lab.ser           .            20.6250        +INF           20.0000
cap.agr           .             0.3750        +INF             .
cap.man           .             0.6000        +INF             .
cap.ser           .             0.3750        +INF             .
```

cap の方は問題ないが、lab の方の mariginal 値が非ゼロである。そこで、変数 a_f が関連付けられている式 e_a_f の定義をチェックする。

```
*       単位生産要素需要
e_a_f(f,i) ..
          a_f(f,i) =e=
              (beta_v(f,i) * c_va(i) / ((1+t_f(f,i))*p_f(f)))**(sig_v(i));
```

式の定義自体には特に問題がないように見える。そこで式の右辺、左辺の値をチェックする。先程と同様に次のようなコードを solve の後に加えてモデルを解いてみる。

```
parameter  chk_e_a_f;

chk_e_a_f(f,i,"LHS") = a_f.l(f,i);
chk_e_a_f(f,i,"RHS")
    = (beta_v(f,i) * c_va.l(i) / ((1+t_f(f,i))*p_f.l(f)))**(sig_v(i));
display chk_e_a_f;
```

すると次のような結果となる。

```
--    390 PARAMETER chk_e_a_f

                LHS          RHS

lab.agr      20.625        0.625
lab.man      20.400        0.400
lab.ser      20.625        0.625
cap.agr       0.375        0.375
cap.man       0.600        0.600
cap.ser       0.375        0.375
```

lab の左辺、つまり a_f("lab",i) の値が明らかにおかしい。そこで、a_f(f,i) の初期値を設定している部分を確認すると、

```
a_f.l(f,i) = a_f0(f,i) + err3(f);
```

となっている。本来の初期値 a_f0 に加え、err3 という余分なパラメータが加わっており、明らか

にこの部分がおかしい。これを修正し、再び解き直すと

```
-- VAR a_f  単位生産要素需要

               LOWER          LEVEL          UPPER          MARGINAL

lab.agr          .             0.6250         +INF             .
lab.man          .             0.4000         +INF             .
lab.ser          .             0.6250         +INF             .
cap.agr          .             0.3750         +INF             .
cap.man          .             0.6000         +INF             .
cap.ser          .             0.3750         +INF             .
```

となる。今度は marginal 値は 0 となり、式は等号で満たされるようになった。

同様にチェックしていくと、次は変数 e に問題がある。

```
                          LOWER          LEVEL          UPPER          MARGINAL

-- VAR e                  .            365.0000        +INF            3.3410
```

そこで、変数 e の関連付けられた式 e_e の定義をチェックする。

```
*       支出関数
e_e .. e =e=
       u * (sum(i,
           (gamma(i))**(sig_c) * ((1+t_c(i))*p(i))**(1-sig_c)))**(1/(1-sig_c));
```

これも式の定義自体には問題がなさそうである。そこで右辺、左辺の値をチェックする。再び、次のようなコードを加え実行する。

```
parameter  chk_e_e;

chk_e_e("LHS") = e.l;
chk_e_e("RHS")
    = u.l * (sum(i,
        (gamma(i))**(sig_c) * ((1+t_c(i))*p.l(i))**(1-sig_c)))**(1/(1-sig_c));
display chk_e_e;
```

すると、次のような結果を得る。

```
--    381 PARAMETER chk_e_e

LHS 365.000,    RHS 361.659
```

右辺、左辺の値が乖離していることがわかる。これだけではどちらの値が間違っているのか（そ

れとも両方間違っているのか）がわからない。とりあえず、変数 e と u の初期値の設定を確認すると

```
e.l = e0;
u.l = u0;
```

となっている。さらに e0 と u0 の値を見ると、

```
*       効用水準
u0 = m0;
```

というように、両者には同じ値が設定されている。これ自体は意図通りの記述であるが、その上のパラメータ m0 の定義を見ると

```
*       所得額
m0 = sum(f, SAM("hh",f)) + err2;
```

となっている。ここで err2 という不適切な値が混ざっていることがわかる。そこで、この err2 を除去して解き直してみると次の結果となる。

```
                          LOWER          LEVEL          UPPER         MARGINAL

-- VAR e                  .            360.0000       +INF             .
```

正常な結果となっている。e の marginal 値がおかしかったのは、m0 の定義の部分が原因であったことがわかる。

さらに下をチェックしていくと、今度は変数 p の marginal 値がおかしい。

```
-- VAR p  財の価格

           LOWER          LEVEL          UPPER         MARGINAL

agr          1.0000         1.0000         1.0000        -2.8571
man   1.0000000E-6           1.0000        +INF            5.3333
ser   1.0000000E-6           1.0000        +INF           -8.0000
```

そこで、これまでと同様に、変数 p が関連付けられた式 e_p の定義をチェックする。

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

一見すると問題ないように思えるが、よく見ると a_x(j,i) のインデックスの指定がおかしい。

正しくは a_x(i,j) である。修正して解き直すと、

```
-- VAR p  財の価格

           LOWER          LEVEL          UPPER         MARGINAL

agr          1.0000         1.0000         1.0000          EPS
man   1.0000000E-6          1.0000        +INF              .
ser   1.0000000E-6          1.0000        +INF              .
```

と問題ない結果を得る[6]。このようにインデックスの順番を間違うのは、GAMS のプログラミングでよくあるミスであるので、デバッグの際のチェック項目の一つとして覚えておくとよい。

以上で全ての変数の marginal 値がゼロになったはずである。これは変数の初期値（＝ベンチマーク・データ）の下で全ての式が等号で満たされていることを意味し、それは結局、BR ができていることを意味する。後に、様々モデルのチェック方法を試すのであるが、この BR によるチェックが最も基本的なチェック方法であり、しかも BR によるチェックではモデルのバグの場所を見つけやすいという利点もある。まずはこの BR を完璧におこなうことが必要である。

注： ここまで初期均衡において全ての式が等号で成り立つという前提で説明してきた。しかし、場合によっては等号で成り立たない均衡を考える場合もある。例えば、初期時点では費用が価格に比べて高すぎるため（採算が合わないため）に生産されない財があるような状況を考えるには、初期均衡において c(i) = p(i) ではなく、c(i) > p(i) が成り立っていなければならない。このような状況を想定するなら、初期均衡において必ずしも全ての条件式が等号で成り立つわけではない。温暖化対策の分析において「バックストップ・エネルギー（backstop energy）」を考慮するときにそのような想定が必要になることが多い。

2.3 Benchmark Replication チェック（まとめ）

CGE モデルでは、「ベンチマーク・データの下で均衡が成立している」という前提を置き、パラメータが設定される。よって、実際にモデルを解いたときに、ベンチマークデータ（変数の初期値）が解に等しくなっていなければならない。BR とはそれをチェックすることである。ベンチマークデータが解に等しくなっていなければ、それはモデルかデータのどこかに間違いがあることになる。

BR によるモデルのチェック・デバッグの手順は以下の通りである。

1). まず、iteration の回数を 0 にしてモデルを解く。

2). 変数の marginal 値をチェックし、それが 0 になっているかを確認する。

3). もし 0 になっていないのなら、その変数が関連付けられた式のどこかがおかしいということになる。

4). 式の定義を確認してみる。定義が間違っているのなら式を修正する。

[6] EPS は GAMS において 0 ではないが、0 に非常に近い小さい数値を表す特殊な記号である。ほぼゼロと考えてよいので、marignal 値が EPS である場合は問題ない。

5). 式の定義がおかしくなければ、式の右辺、左辺の値をチェックし、どちらの値がおかしいかを調べる。

6). さらに、式に含まれるパラメータや変数の値をチェックし、どの部分の値がおかしいかをより詳しく見ていく。

7). 以上の手順で間違いを見つけ、修正をしていく。

8). 最終的に全ての式が等号で満たされ、BR ができたら終了である。

第 2.2 節の例では、間違いの原因はモデル（式）の定義や、パラメータの定義であったが、場合によってはベンチマークデータ（SAM データ）に誤りがあることもある。

`benchmark_replication_check.gms` の間違いを修正して、bechmark replication が成立するようになったプログラムが `benchmark_replication_check_corrected.gms` である。どう修正されているか参考にして欲しい。

2.4 注：Benchmark Replication でチェックできないバグ

BR によるチェックでは、均衡条件式が等号で満たされているか否かという点に着目する。よって、均衡条件式の右辺、左辺の値に影響を与えるようなバグは見つけることができるが、均衡条件式の値を変えないような種類のバグを見つけることはできない。

例えば、正しい式が

```
a_x(j,i) =e= (alpha_x(j,i) * c(i) / p(j))**(sig(i));
```

であるのに、プログラムでは次のように書かれていたとする。

```
a_x(j,i) =e= (alpha_x(j,i) * c(i) * p(j))**(sig(i));
```

通常二つの式で右辺の値は変わってくるが、`c(i)`、`pc(j)` が 1 の値をとるのなら、二つの式の右辺はどちらも同じ値になってしまう。実際、ここで考えているモデルでは `c(i)`、`pc(j)` は共に初期値として 1 をとるので、二つの式の右辺の値は一致する。従って、BR によるチェックではこの部分のバグを見つけることはできない。

次の節の cleanup calculation によるチェックについても同様のことが言える。このタイプのようなバグを見つけるには、BR、cleanup calculation によるチェック以外のチェック方法を使う必要がある。それについては、第 4 節で説明する。

3 Cleanup Calculation

次に「**cleanup calculation（以下、CC）**」を説明しよう。これもモデルのチェック、デバッグ方法の一つである。CC は作成したモデルを何もショックを与えずに、そのまま解いてみることで

ある[7]。前節で見たように、CGE 分析ではモデルの解とベンチマークデータが等しくならなければならない。実際にモデルを解いてみて、解がベンチマークデータに一致しているか確認するのが CC である。

BR チェックとの違いは iteration の回数を 0 にはしないことである。BR チェックでは iteration の回数を 0 にするため、「変数の初期値（＝ベンチマークデータ）がモデルの解に等しいかどうか」をチェックすることになる。一方、CC では「実際にモデルを解いて求められた解がベンチマークデータの値に等しいかどうか」をチェックすることになる。

benchmark_replication_check.gms では BR チェックの後の$exit 命令に続く部分に CC をおこなうコードがある。

```
*       -------------------------------
*       Clean-up calculation

ge_sample_dual.iterlim = 1000;

solve ge_sample_dual using mcp;
```

コードとしては、BR チェックにおいて 0 にセットしていた iteration 回数を十分大きな値に設定し直して、もう一度モデルを解くだけである。何らショック（外生変数の変化）を与えていないので、これで計算された解はそのままベンチマークデータに等しくなっていなければならない。

benchmark_replication_check.gms は多くのバグ（プログラムの誤り）を含んでいるので、解はベンチマークデータに等しくならない。実際に CC の部分を解いてみると、

```
-- VAR c  生産の単位費用

            LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6          0.8522        +INF           -0.1345
man   1.0000000E-6          1.4268        +INF           -0.1245
ser   1.0000000E-6          0.8966        +INF           -0.1255

-- VAR y  生産量

            LOWER          LEVEL          UPPER         MARGINAL

agr           .             0.0622        +INF           -0.1478
man           .           469.5820        +INF           -0.1574
ser           .           190.8489        +INF           -0.1590
```

というような計算結果になる。本来、1 に等しくなるはずの変数 c も 1 にはならないし、生産量 y も y0 とは異なる値になってしまう。このように、本来ベンチマークデータに等しくなるはずの変数の値がならないことによって、モデルやデータに問題があることがわかるのである。

一方、BR が成立する benchmark_replication_check_corrected.gms で CC を行なった場合には次のような結果になる。

```
-- VAR c 生産の単位費用
```

[7]「BR」と同様に、「CC」についても、この用語が一般に幅広く利用されているわけではないので注意。

```
           LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6          1.0000        +INF             .
man   1.0000000E-6          1.0000        +INF             .
ser   1.0000000E-6          1.0000        +INF             .

-- VAR y  生産量

           LOWER          LEVEL          UPPER         MARGINAL

agr            .          140.0000       +INF             .
man            .          300.0000       +INF             .
ser            .          150.0000       +INF             .
```

解はちゃんとベンチマークデータの値に等しくなっており、CC で期待される結果となっている。ただし、CC で期待される結果が出ても、バグがないわけではない。CC で期待される結果（解がベンチマークデータに等しくなるという結果）が出なければ、モデルかデータに問題があるということであって、期待される結果が出たら問題ないということではない。実際、後に見るように、このプログラムには他にも多くのバグが含まれている。

3.1 Cleanup Calculation を行なった場合の solve summary

第 2 節で見たように、BR ができなければモデルやデータに問題があることになる。その場合、モデルの solve summary においては基本的にエラーの表示、つまり、solver status や model status が 1 以外の値をとることになる。一方、CC においては、モデルやデータに問題があっても、solver status や model status は 1 の値をとり、正常にモデルが解けているという表示になることがある。、以下は `cleanup_calc_check.gms` において CC をおこなった結果である。

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

     MODEL   ge_sample_dual
     TYPE    MCP
     SOLVER  PATH                FROM LINE  381

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

 RESOURCE USAGE, LIMIT          0.015      1000.000
 ITERATION COUNT, LIMIT         3          1000
 EVALUATION ERRORS              0             0
```

solver status も model status も共に 1 になっており、モデルが正常に解けていることを示している。一方、変数の値を見ると、

```
-- VAR c  生産の単位費用

           LOWER          LEVEL          UPPER         MARGINAL
```

```
agr   1.0000000E-6          1.0000        +INF              .
man   1.0000000E-6          0.9951        +INF              .
ser   1.0000000E-6          0.9997        +INF              .

-- VAR y  生産量

            LOWER          LEVEL          UPPER         MARGINAL

agr           .           143.2681        +INF              .
man           .           291.2507        +INF              .
ser           .           157.3207        +INF              .
```

のようにベンチマークデータの値には等しくなっていない。よって、モデル（かデータ）に問題があることがわかる。このようにモデルにバグがあるとしても、solver status や model status は正常な値をとるということはよくある。というのは、solver status や model status の値が示すのは、単に「モデルが計算上正常に解けているか」どうかであり、「経済学的に正しいモデルやデータになっているか」、あるいは「作成者の意図通りのモデル、データになっているか」を示すわけではないからである。CC では solver status や model status の値が正常かどうかではなく、解がベンチマークデータに等しくなっているかどうかをチェックしなければならない。

3.2 Benchmark Replication と Cleanup Calculation の違い

BR によるチェックではモデルにバグがあるかどうかに加え、変数の marginal 値の情報を基にどこにバグがあるかを推測することができた。一方、CC によるチェックではバグがあるかどうかは判断できるが、それがどこにあるかを判断することは難しい。というのは、CC では iteration 回数を 0 としないため、仮にモデルにバグがあったとしても、モデルが解けた後の変数の marginal 値は 0 になってしまうことが多いからである。marginal 値が 0 であれば、どの式が問題なのか推測することができない。このため、バグが存在するかどうかしか判断できない。

それでは、BR チェックに加えて、CC によるチェックをする意味はあるのかという疑問を持つかもしれない。これは場合による。例えば、この文書で例として利用しているプログラムでは BR によるチェックで十分で、CC によるチェックをする意味はない。BR によるチェックをおこなえば CC によるチェックをする必要は全くなくなる。しかし、データの規模が非常に大きいような状況では CC によるチェックが意味を持ってくることがある。それは次のような理由による。

この文書で利用している仮想的な SAM は 6×6 というサイズの非常に単純なデータであり、データ上の誤差のようなものはほぼない。ところが、データの規模が大きい場合には、データに誤差が入ってきやすい（例えば、SAM の列和が行和と微妙に違ってくる等）。そして、そのデータの誤差のために **BR チェックの際に変数の marginal 値が 0 にならないというケースが生じてくる**。このような場合、benchmark replication チェックの結果を解釈しづらくなる。例えば、SAM の要素の値が 10 〜 1000 というような水準をとる状況で、ある変数の marginal 値が、0.123 という小さ目の値であったとする。もし単純で誤差のない SAM を前提としていたら、この非ゼロの marginal 値はモデルのバグを意味する可能性が非常に高い。しかし、誤差を含んだデータを用いているのなら、これがモデルのバグによるものなのか、それともデータの誤差によるものなのか判断することが難しい。もしモデルのバグによるなら、それは修正すべきであるが、データの誤差によるものならば修正しなくてもよいかもしれない（多少の誤差は許容せざるを得ないかもしれない）。

このような場合には、CC をおこなう意味が出てくる。CC をおこなってみて、もし解がベンチマークデータから大きく乖離したのなら、marginal 値が非ゼロであることの原因はモデルのバグにある可能性が高い。一方、解がほとんどベンチマークデータと一致したのなら、marginal 値が非ゼロである理由はデータの誤差にある可能性が高い。実際には微妙な数値が出てきて、結局どちらか判断するのが難しいケースも出てくるのだが、BR だけではモデルのバグか、データの誤差か判断が難しいときには、とりあえず CC をおこなってみるのがよい。それによって、どちらかがはっきりする場合があるからである。

4 その他の一般均衡モデルのチェック方法

「BR によるチェック」と**「CC によるチェック」**という二つのモデルのチェック方法を見た。この二つの方法は、「ベンチマークデータの下でモデルが均衡状態にある」という CGE 分析の前提を利用しており、その意味で CGE 分析に特有のモデルのチェック方法である。一方、第 6 章において一般均衡モデルのチェック方法を幾つか紹介した。この節では、それらのチェック方法を含め、BR、CC 以外のモデルのチェック方法を一通りおこなってみる。CGE 分析をおこなうにあたって、モデルのチェックは極めて重要な位置を占める作業であるので、この節の内容もしっかり理解することが望ましい。

例として、other_check.gms というプログラムを利用していく。このプログラムのモデルは benchmark_replication_check_corrected.gms のモデルと全く同じであり、BR が成立するように修正済みのプログラムである。しかし、BR が成立したからと言って、バグがないわけではないし、実際多くのバグを含んでいる。それを以下ではいろんなシミュレーションをおこなうことで見つけていく。

4.1 ニュメレールによるチェック

まず、「ニュメレールを用いたチェック」をおこなう。これは 第 6 章でおこなったものである。other_check.gms のモデルは 第 6 章のモデルとほぼ同じであり、価格についての 0 次同次性が成り立つはずである。よって、1 に設定されているニュメレールの価格を仮に 2 に上昇させると、全ての名目変数（価格変数、金額変数）の値が 2 倍になる一方、数量変数（及び、実質変数）の値は全く変わらないという結果になるはずである。これが実際に成り立つかどうかをチェックするため、以下のコードを実行する。これには CC の後の$exit 命令を消せばよい。

```
*       ------------------------------
*       ニュメレールによるチェック

p.fx("agr") = 2;

solve ge_sample_dual using mcp;

p.fx("agr") = 1;
```

実行すると次のような結果となる。まず、solve summary の部分は

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

```
     MODEL   ge_sample_dual
     TYPE    MCP
     SOLVER  PATH                FROM LINE  388

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

 RESOURCE USAGE, LIMIT          0.015      1000.000
 ITERATION COUNT, LIMIT      1087          1000
 EVALUATION ERRORS              0             0
```

となり、そもそもエラーとなる。さらに、変数の値は（エラーが出ているので変数の値に意味がないのは明白であるが)、

```
-- VAR c  生産の単位費用

           LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6         1.9769        +INF           -0.0229
man   1.0000000E-6         2.0603        +INF           -0.0228
ser   1.0000000E-6         1.9825        +INF           -0.0229

-- VAR y  生産量

           LOWER          LEVEL          UPPER         MARGINAL

agr          .             0.0390        +INF           -0.0231
man          .           408.5779        +INF           -0.0227
ser          .           228.9071        +INF           -0.0229
```

となっており、価格変数は 2 倍になっていないし、数量変数の値は変わってしまっている。以上より、価格の 0 次同次性が満たされていないことがわかる。これはモデルのどこかがおかしいということを意味している。

次にどこがおかしいかを見つけたいのだが、はっきり言って、この結果だけからモデルのどこがおかしいかを見つけるのは少し難しい。というのは、様々なタイプのバグがこの原因となりうるからである。しかし、価格の 0 次同次性が満たされていないということであるから、とりあえずは価格の 0 次同次性が violate されているところはないかと探していけばよいであろう。実際に間違っているのは次の部分である。

```
 *       単位投入需要
e_a_x(j,i) ..
          a_x(j,i) =e= (alpha_x(j,i) * c(i) * p(j))**(sig(i));
```

この式の右辺には c(i) と p(j) が乗法の形で入っており、価格が 2 倍になれば当然右辺は変化することになる。しかし、左辺は数量変数であるので、ニュメレールの価格を変えても変わらないはずである。右辺は変わるのに、左辺は変わらない関係になっているということで、この式がおか

しいことがわかる。正しくは次の式となる。

```
 *       単位投入需要
e_a_x(j,i) ..
          a_x(j,i) =e= (alpha_x(j,i) * c(i) / p(j))**(sig(i));
```

このようになっていれば、価格が 2 倍になっても右辺は分母、分子が 2 倍ずつになるだけで不変となり、左辺が不変であることと整合的になる。実際、このように修正してモデルを解き直すと、次のような解を得る。

```
-- VAR c  生産の単位費用

            LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6          2.0000        +INF             .
man   1.0000000E-6          2.0000        +INF             .
ser   1.0000000E-6          2.0000        +INF             .

-- VAR y  生産量

            LOWER          LEVEL          UPPER         MARGINAL

agr           .           140.0000        +INF             .
man           .           300.0000        +INF             .
ser           .           150.0000        +INF             .
```

価格は 2 倍になり、生産量は不変である。ちゃんと価格の 0 次同次性が成立しており、正常な結果となっている。

以上のように、ニュメレールの価格を変えてみて、価格の 0 次同次性が成り立っているかどうかをチェックすることで、モデルにバグがあるかどうかを判断する方法がある。

4.2 外生変数の比例変化によるチェック

次に、「外生変数の比例変化によるチェック」をしてみよう。これも 第 6 章でおこなったチェックである。other_check.gms のモデルは 第 6 章のモデルとほぼ同じであり、全ての関数形に 1 次同次関数を利用している。このため、全ての外生変数（このモデルでは労働と資本の賦存量）を $x\%$ 変化させると、全ての内生的な数量変数（生産量、消費量等)、及び金額変数（生産額、所得額、支出額等）が $x\%$ だけ変化し、価格変数は変化しないという結果になるはずである。これを実際に確認するため、以下のコードを実行する。other_check.gms では「ニュメレールによるチェック」の下の$exit 命令を消せばよい。

```
*       -------------------------------
*       外生変数の比例変化によるチェック

v_bar(f) = v_bar0(f) * 1.5;
```

```
solve ge_sample_dual using mcp;

v_bar(f) = v_bar0(f);
```

このコードでは生産要素の賦存量を 50%増加させている。よって、計算結果では数量変数、金額変数は 50%増加するはずである。この計算結果を見てみると

```
-- VAR c  生産の単位費用

           LOWER          LEVEL          UPPER         MARGINAL

agr   1.0000000E-6         1.0000        +INF             .
man   1.0000000E-6         1.0310        +INF             .
ser   1.0000000E-6         1.0020        +INF             .

-- VAR y  生産量

           LOWER          LEVEL          UPPER         MARGINAL

agr          .            93.2085        +INF             .
man          .           530.8523        +INF             .
ser          .           248.3273        +INF             .

                      LOWER         LEVEL         UPPER        MARGINAL

-- VAR e                .        551.5988         +INF            .

  e  支出
```

となる。価格変数（単位費用）は変化してしまっているし、生産量、支出額の変化率は 50%になっていない。これはモデルのどこかがおかしいことを示唆している。

どこがおかしいかであるが、これについても簡単にバグの位置を見つけるのは難しいかもしれない。答えを言ってしまうと、

```
*       消費需要
e_d(i) .. d(i) =e=
          u * (gamma(i)*(e/u0)/((1+t_c(i))*p(i)))**(sig_c);
```

という部分がおかしい。この式には、数量変数としては d(i)、u の 2 つ、金額変数としては e の 1 つが出てきている。d(i) と u が同率で変化したとすると、右辺、左辺が同率で変化することになるので問題はない。しかし、右辺にはもう一つ e の変化が入ってくるので、このままでは右辺、左辺の変化が釣り合わなくなってしまう。ここの部分は実際には、e/u0 ではなく、e/u となっていなければならない。こうすると両辺の変化がちょうど釣り合う形になる。このように修正して、モデルを解き直すと以下の結果となる。

```
-- VAR c  生産の単位費用

       LOWER     LEVEL     UPPER    MARGINAL
```

```
agr 1.0000E-6     1.000     +INF       .
man 1.0000E-6     1.000     +INF       .
ser 1.0000E-6     1.000     +INF       .

-- VAR y  生産量

       LOWER     LEVEL     UPPER    MARGINAL

agr      .      210.000     +INF       .
man      .      450.000     +INF       .
ser      .      225.000     +INF       .

                       LOWER     LEVEL     UPPER    MARGINAL

-- VAR e                .      540.000     +INF       .

  e  支出
```

価格変数は不変で、数量変数、金額変数は 50%ずつ増加している。期待通りの結果となっている。

以上のように、全ての関数に一次同次関数を用いているモデルでは、全ての外生変数を比例的に変化させると、数量変数、金額変数も同率だけ比例的に変化する一方、価格変数は変化しないという結果が成り立つモデルが多い。これが実際に成り立つかどうかをチェックすることで、モデルのバグをチェックすることができる。

4.3 政策の変化

試しに政策を変化させてみることで、モデルの挙動をチェックする方法もある。other_check.gms のモデルでは、初期時点では税率はゼロであるが、消費税と生産要素への税を表す変数を組み込んでいる。税率をプラスに変えて、税を導入するシミュレーションをおこなってみる。具体的には、次のようなシミュレーションをおこなってみる。

```
*       ------------------------------
*       政策変化

t_c(i) = 0.1;
t_f(f,i) = 0.2;

solve ge_sample_dual using mcp;
```

つまり、消費税を 10%、生産要素の投入への税を 20%とするという政策である。これを実行す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   ge_sample_dual
     TYPE    MCP
     SOLVER  PATH                FROM LINE  411
```

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

 RESOURCE USAGE, LIMIT          0.000      1000.000
 ITERATION COUNT, LIMIT         3          1000
 EVALUATION ERRORS              0             0
```

モデルは正常に解けている。しかし、変数 p の部分を見ると

```
-- VAR p  財の価格

       LOWER     LEVEL     UPPER    MARGINAL

agr     1.000     1.000     1.000    32.494
man 1.0000E-6     0.988     +INF        .
ser 1.0000E-6     0.999     +INF        .
```

という値になっている。変数 p("agr") の marginal 値が 0 になっていない。今、シミュレーションでは財 agr（農産物）をニュメレールとして、その価格 p("agr") を 1 に固定している[8]。この p("agr") は農産物の市場均衡条件を表す式 e_p("agr") に関連付けられているので、式 e_p("agr") は連立方程式体系からは drop（除外）される[9]。しかし、このモデルは「ワルラス法則（Walras' law）」が成り立たなければいけないモデルであるので、自動的に農産物市場は均衡するはずである[10]。

変数 e_p("agr") は式 e_p("agr") に関連付けられているので、その marginal 値は式 e_p("agr") の乖離幅を表す。式 e_p("agr") の乖離幅は、その定義より、農産物市場の超過供給量を表しているので、農産物市場が均衡しているのなら、変数 e_p("agr") の marginal 値は 0 になっていなければならない。上の結果のように p("agr") の marginal 値が非ゼロということは、農産物の市場が均衡しておらず、ワルラス法則が満たされていないことを意味している。これはやはりモデルにバグがあることを示唆している。

政策（税）を導入しない状況では正常に解けていたのが、政策を導入したことで、問題が生じるようになったので、今回のバグは税に関連する部分にあるということが推測できる。実際におかしいのは次の部分である。

```
*       所得の定義式
e_m .. m =e= sum(f, p_f(f)*v_bar(f))
             + sum((f,i), t_f(f,i)*p_f(f)*a_f(f,i)*v_a(i));
```

所得に生産要素税の収入が含まれているが、消費税の収入が含まれておらず、消費税収入の行き場がなくなっている。税率が 0 であるのなら税収は生じないので、この部分の間違いは表面化していなかったが、税を導入したことで問題が表面化したのである。正しくは次のようなプログラム

[8] p("agr") を 1 に固定しているので、変数の LOWER、LEVEL、UPPER の全ての値が 1 になっている。

[9] MCP のモデルでは、ある変数を固定すると、その変数が関連付けられている式は方程式体系から除外されることになる。

[10] ワルラス法則（超過需要額の総額が 0）が成り立つなら、n 個の市場があるとき、$n-1$ 個が均衡すれば、残りの一つの市場も均衡する

になる。

```
*       所得の定義式
e_m .. m =e= sum(f, p_f(f)*v_bar(f))
             + sum((f,i), t_f(f,i)*p_f(f)*a_f(f,i)*v_a(i))
             + sum(i, t_c(i)*p(i)*d(i));
```

モデルを実行し直すと、solver status も model status も共に 1 となり、さらに

```
-- VAR p  財の価格

       LOWER     LEVEL     UPPER    MARGINAL

agr     1.000     1.000     1.000 4.263E-14
man 1.0000E-6     1.000     +INF        .
ser 1.0000E-6     1.000     +INF        .
```

というように、p("agr") の marginal 値もほぼゼロとなり、ワルラス法則が成立する結果となる。

ワルラス法則については、この節になって言及したが、この種のモデルで常に成り立つはずの性質であるので、どのようなシミュレーションをおこなう場合であっても、常にワルラス法則をチェックするのが望ましい。逆に言えば、どのようなシミューレションをおこなう際でも、ワルラス法則が成り立っているかどうかをチェックすることで、モデルにバグがないかどうかを判断するための情報を得ることができるということである。

ワルラス法則の成立を簡単にチェックするためには、例えば、

```
parameter  chk_walras    "Excess supply for the numeraire (agr)";
```

という chk_walras というパラメータを導入し、solve 命令の後に

```
solve csf_model using mcp;

chk_walras = round(p.m("agr"), 4);
abort$chk_walras "ワルラス法則が満されていません！", chk_walras;
```

というようなコードを加えるとよい。パラメータ chk_walras にはニュメレール（agr）の marginal 値（超過供給量）が入り、それが 0 でなければ abort 命令によりそこでプログラムがストップするというコードである[11]。このようにしておけばいちいちその度にニュメレール価格の marginal 値を目視でチェックしなくとも、自動的にワルラス法則がチェックできる。

other_check.gms のバグを全部修正したプログラムが other_check_corrected.gms である。比較して、どこが問題かをもう一度チェックして欲しい。

[11] round は四捨五入によって数をまるめる命令である。例えば、round(x,4) とすると、x の小数第 4 位を四捨五入する。

4.4 まとめ

チェック方法のまとめ。

[ニュメレールによるチェック]
ニュメレールの価格を本来の 1 から違う値に変化させてみて、価格についての 0 次同次性が成り立っているかをチェックする。価格変数、金額変数のみ同率で変化し、数量変数は不変の結果になれば問題ない。

[外生変数の比例変化によるチェック]
外生変数を比例的に変化させてみる。数量変数、金額変数が同率で変化し、価格変数は変化しないという結果となれば問題ない。ただし、このチェックはモデルが 1 次同次関数からのみ構成されるときにしか使えない。

[ワルラス法則のチェック]
ワルラス法則が成り立っているかどうかをチェックする。成り立っていなければモデルのどこかがおかしいということになる。どのようなシミュレーションをおこなうとしても、常にワルラス法則はチェックするようにしたほうがよい。ただし、ワルラス法則が成り立たないモデルを利用するなら、このチェックは意味はない（普通はそのようなモデルは使わないであろうが)。

[政策の変化によるチェック]
政策を変化させてモデルの挙動をチェックする。

4.5 その他のチェック方法

経済学のモデルでは、ある政策の効果が事前に明らかな場合がある。例えば、

- 閉鎖経済、完全競争、代表的家計、政府部門はない

というようなモデルでは、競争均衡の状態はパレート効率的であり、政府が介入すると代表的家計の効用は必ず低下するということがわかっている。

このように、ある政策の効果が明らかであるときに、実際にその政策を導入してみて期待される効果が生じるかを確認することで、チェックするという方法がある。期待される効果が生じないのなら、モデル、データ、プログラムのどこかがおかしいということになる。

この章で利用してきたモデルも上のように競争均衡がパレート効率的になるモデルであるので、政府の政策を導入すれば代表的家計の効用は低下するはずである。以下では、MAN に対する消費課税と消費補助金という二つの政策を導入したときに実際に代表的家計の効用が低下するかをチェックしてみる。`policy_simulation.gms` というプログラムでこのシミュレーションをおこなっている。

初期均衡では、代表的家計の効用水準は「360」となっている。これに対し、`t_c("man") = 0.2;` というように「MAN に対して 20%の消費課税」を導入すると、効用水準は「359.307」となる。効用水準は消費課税により低下する。同様に、`t_c("man") = - 0.2;` というように「MAN に対して 20%の消費補助金」を導入すると、効用水準は「358.962」となり、やはり低下する。このこと自体はモデルの正しさを証明するものではないが、このようなシミュレーションをしてみてもし逆の結果が出たなら明らかにモデルがおかしいということになる。

最後に、全ての財に同率の消費税を導入するというシミュレーション（t_c(i) = 0.2;）もしている。この場合、効用水準（他の数量変数も）は初期均衡から変わらない。つまり、このモデルでは同率の消費税は全く効果がないということを意味する。消費税を導入しても全く効果がないという結果はおかしいと思われるかもしれないが、これが正しい結果である。というのは、このモデルは一次同次の関数形から構成され、同率の消費税は財の間の相対価格を変化させないためである。このシミューレションでも、仮に消費税の導入により均衡が変化したとすると、どこかにバグがあるということがわかる。

5 例

もう一つ例として check_exercise.gms というプログラムがある。これは第 8 章で利用した calibrated_share_form.gms をベースにしたプログラムである。モデルは other_check.gms のモデルと実質的には同じであるが、calibrated share form を利用してモデルを記述しているという違いがある。

これをどのようにデバッグしていくか説明したものが、check_exercise_corrected.gms というファイルである。チェック・デバッグ方法についての詳細はプログラム自体に記述している。基本的な考え方はここまで説明してきた通りであるが、デバッグ手法・手順の具体的な例として参考にして欲しい。

6 履歴

- 2017-03-15: 説明の修正。