応用一般均衡分析入門

第 3 章：GAMS の利用方法 *

武田 史郎 †

Date: 2022/04/01,
Version 3.0

目次

* 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/`

1 導入

今回の内容について

- 第 3 章ではひとまず CGE 分析の話は置いておき、GAMS (General Algebraic Modelling System) の利用方法を説明する。GAMS は数値計算ソフトウェアであり、様々な用途の数値計算に利用できるが、CGE 分析という分野で非常によく用いられている。この解説文書でも基本的にソフトウェアとしては GAMS を利用している。
- この文書で GAMS の基本的な利用方法について学んだら、後は以下のマニュアルを参考に知識を深めていくのがよい。ただし、マニュアルを最初から順に読んでいくというような必要はない。必要な機能、調べたい機能が出てきたときに、その都度、マニュアルを参照すればよい。
- 「GAMS User's Guide」
  - これが GAMS の開発元である GAMS corporation による公式のマニュアルである。基本的なことについてはこれをまず見るのがよい。
  - HTML 版: https://www.gams.com/latest/docs/UG_MAIN.html
- **「入門用のチュートリアル」**
  - 本書と同じように入門用の教材として以下の二つがある。
  - A GAMS Tutorial by Richard E. Rosenthal
  - Quick Start Tutorial
  - ただし、前者については説明に利用しているモデルが LP（線形計画法）であるので、

CGE 分析用の勉強としてはあまり適切な教材とは言えないかもしれない[1)]。

- **「Bruce McCarl によるマニュアル」**
  - 上の公式マニュアルは「公式」にもかかわらず、実は GAMS の機能を全て説明しているわけではない。
  - 公式のマニュアルでは説明されていないが、こちらの McCarl によるマニュアルで説明されていることもある。もし、ある命令について公式にマニュアルでは説明されていないのなら、こちらを確認してみるのがよい。
  - 注：このマニュアルは 2020 年 1 月時点での最新版でも February 28, 2016 のものであるので、少し内容が古くなっている。
  - HTML 版: https://www.gams.com/mccarlGuide/gams_user_guide_2005.htm
  - PDF 版: https://www.gams.com/mccarlGuide/mccarlgamsuserguide.pdf

2 GAMS のプログラムの作成・実行環境

GAMS で CGE 分析をおこなう際には

- プログラムの編集
- プログラムの実行
- 実行結果の分析

などの作業が必要になる。これらの作業を行なう方法として大きく分けて次の 3 つの方法がある。

1) 「GAMS Studio」という統合開発環境（integrated development environment、以下 IDE）を利用する。
2) 「GAMS IDE」という IDE を利用する。
3) プログラムはエディタで編集し、GAMS はコマンドラインから実行する。

[注] ここでの IDE とは GAMS のプログラムの作成、実行、計算結果のチェックなどを行うためのソフトウェアのことである。

昔から GAMS を利用している人間は 2（あるいは 3）の方法をとっている人が多いと思うが、これから GAMS を利用したいという人は 1 の方法を使うのがよい。GAMS Studio は GAMS でシミュレーションをおこなう際に必要になる様々な作業を容易に行なうための機能を提供する IDE である。GAMS に標準的に付属するソフトウェアであるので、GAMS をインストールすれば利用できる[2)]。

2 の GAMS IDE も IDE であるという点では、GAMS Studio と同じであるが、

1) 筆者（武田）もこれまで LP を解いたことはないので、LP についての知識は全くない。
2) ただし、GAMS Studio は GAMS の 25.1 版から提供されるようになったものであるので、それ以前の GAMS では使えない。

- GAMS IDE は Windows 版しかないのに対し、GAMS Studio は Windows、Mac、Linux のどの環境でも使える
- GAMS Studio はまだ開発途上ではあるが[3]、既に GAMS IDE よりも機能が豊富

というように全体的に見て、GAMS Studio の方が機能も豊富で使いやすくなっている。よって、特に理由がなければ新しい GAMS Studio を利用することを勧める。

ちなみに、3 は汎用のエディタ（例えば、Emacs, Vim, Atom、Sublime、Atom、秀丸など）でプログラムを編集し、実行はコマンドライン、あるいはエディタから GAMS を呼出して実行するという方法である。この方法では自分の好むエディタでプログラムの編集ができるなどの様々な利点があるが[4]、GAMS（及び、プログラミング全般）の初学者には少し難易度が高いと思われる。

3 GAMS Studio の利用法

以下では GAMS Studio を使うという前提で、GAMS の基本的な利用法を説明する。

3.1 GAMS のインストール

3.1.1 GAMS のダウンロード

まず、GAMS 自体をインストールする。GAMS は有料のソフトウェアであり、GAMS の機能をフルに利用するには、有料のライセンスを購入する必要がある。しかし、有料のライセンスがなくても、**「デモバージョン」**として使うことができる。デモバージョンは機能に制限が付いた状態のものであるが、基本的な使い方は有料版と変わらない。これから GAMS の使い方を学ぶという段階の人は、とりあえずデモバージョンの GAMS を利用すればよい。

GAMS は GAMS のウェブサイト（https://www.gams.com/）からダウンロードできる。ウェブサイト内のダウンロードのページ（https://www.gams.com/download/）にアクセスし、最新版のプログラムをダウンロードすればよい。GAMS には Windows 版、Mac 版、Linux 版があるので、自分の環境に合うものをダウンロードする。**以下では、Windows 版を前提に話を進める。** Windows 版には、32bit 版と 64bit 版があるが、よほど古いパソコンを使っているのでなければ 64bit 版が使えるはずであるので、64bit 版（x86_64 architecture 用）をダウンロードすればよい。

[注] Windows 以外のバージョンのインストールについてはマニュアルの以下の部分で説明されている。

- https://www.gams.com/34/docs/UG_MAIN.html#UG_INSTALL

3) 2022 年 1 月現在でも、機能のアップデートが頻繁におこなわれている。
4) 筆者は Emacs というエディタで GAMS のプログラムを編集している。Emacs には筆者が作成した GAMS mode for Emacs というマクロ（拡張機能）があり、それを利用することで GAMS Studio を使うとき以上に GAMS のプログラミング、実行が快適に行なえる。

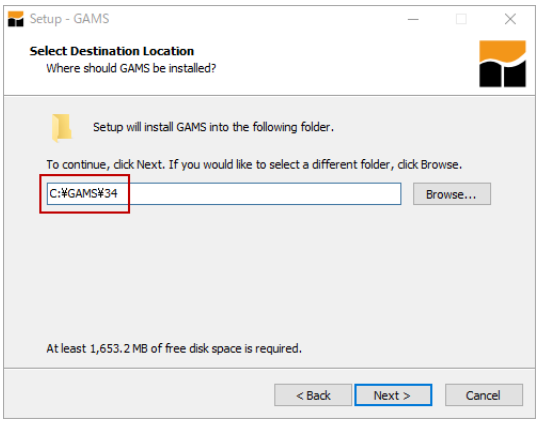
3.1.2　無料のデモバージョン用のライセンス

また、ページ右上にある「FREE DEMO」というボタンを押すと https://www.gams.com/try_gams/ というページに移動するが、ここから「無料のデモバージョン用のライセンス」の申請ができる。ここで申請すると、登録したメール宛てにライセンスファイル（正確には、ライセンスファイルに書き込む文字列）が送られてくる。

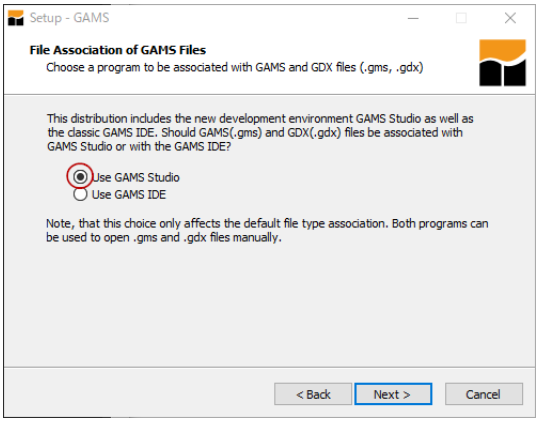
メールが届いたら、（例えば）「メモ帳」を起動し[5]、メールにある内容をコピー・貼り付けして、それを「gamslice.txt」という名前のファイルに保存する（とりあえず保存場所はどこでもよい）。

3.1.3　GAMS のインストール

先程ダウンロードしておいた GAMS のプログラムのファイルをクリックすれば、インストールできる。GAMS は既定の設定では以下のように C ドライブの GAMS フォルダの下にインストールされる（数字の 34 は GAMS のバージョンを表現しており、インストールするバージョンによって変わってくる）。



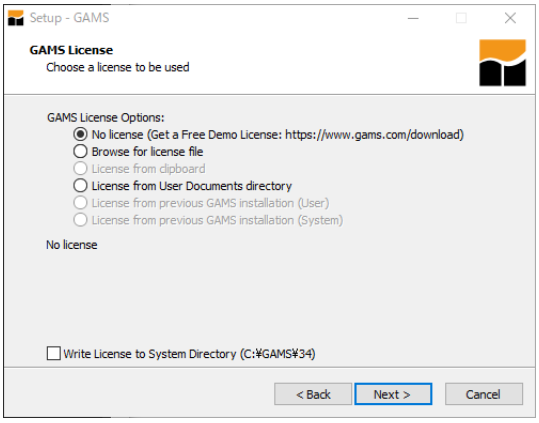
「Next」ボタンを押していくと、以下のように IDE として GAMS Studio を使うか、GAMS IDE を使うか聞かれるが、そのまま GAMS Studio を選べばよい。



次に、以下の画面でライセンスを聞かれる。ここで、「Browse for lisence file」を選んで、先程保存しておいた「gamslice.txt」を選択すればよい（有料のライセンスを持っている人は、自分のラ

5)　テキストファイルを作成・編集できるソフトウェアならなんでもよい。

イセンスファイルを選べばよい)。



後は、そのまま「Next」ボタンを押していけばよい。

3.1.4 GAMS の無償版と有料版の差

上で述べたように、GAMS のフルの機能を利用するには有料のライセンスを購入する必要があるが、デモのライセンスでも制限がかかるが GAMS を利用できる。最初はこのデモバージョンで使い方を勉強するとよい。

デモバージョンの場合には次のような制限がかかる。

- 線形のモデル（LP、RMIP、MIP）については変数と制約の数はそれぞれ 2000 まで。
- その他のタイプのモデルについては変数と制約の数はそれぞれ 1000 まで。

この「応用一般均衡分析入門」という文書で使っているサンプルのプログラムでは非線形のモデルを利用しているので、「変数と制約の数がそれぞれ 1000 まで」という制限が問題になるが、基本的にこの制限内におさまるサイズのモデルしか使わないので、デモバージョンの GAMS でも解くことができる。

3.2 GAMS Studio の基本的な使い方

3.2.1 `trnsport.gms` を使った説明

以下では GAMS Studio の基本的な利用方法を解説する。ここでは、1) GAMS の実行、2) プログラムの誤り（バグ）の修正という最低限必要な利用方法のみを説明する。GAMS Studio のより詳しい利用法については、

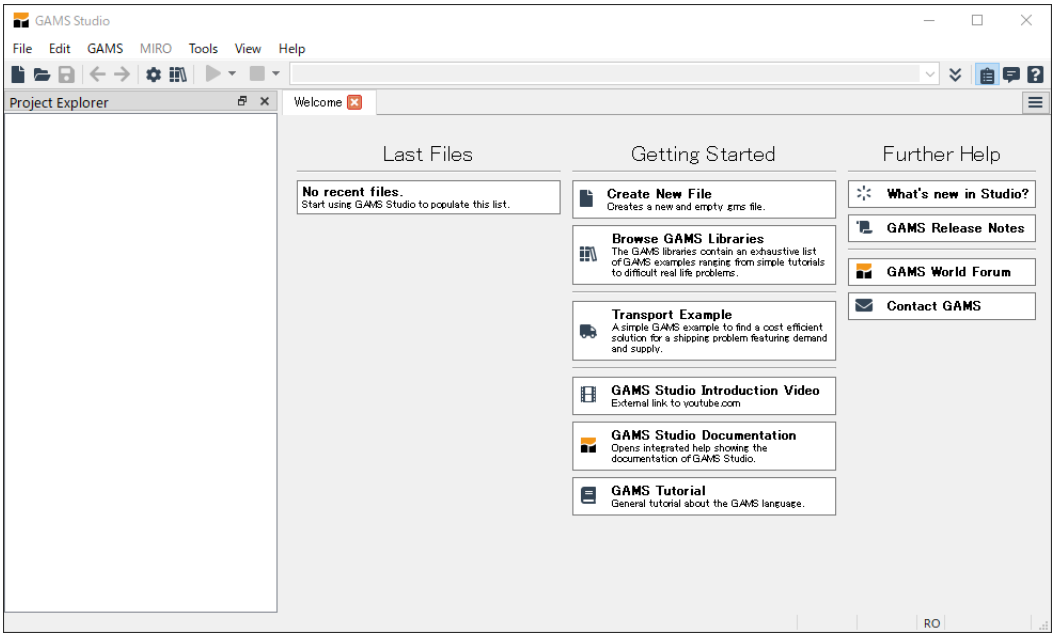
- https://www.gams.com/latest/docs/UG_studio_tutorial.html

を見て欲しい[6)]。また、YouTube の An Introduction to GAMS Studio という動画でも GAMS

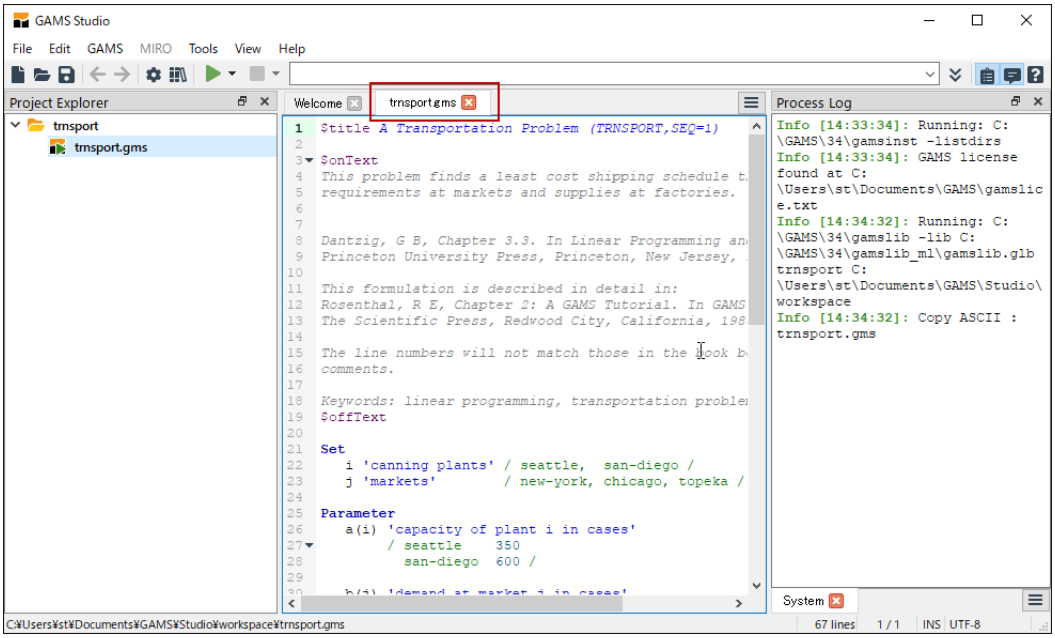
6) リンク先のマニュアルも包括的な説明ではないが。

Studio の使い方が説明されている（少し古く、英語であるが)。

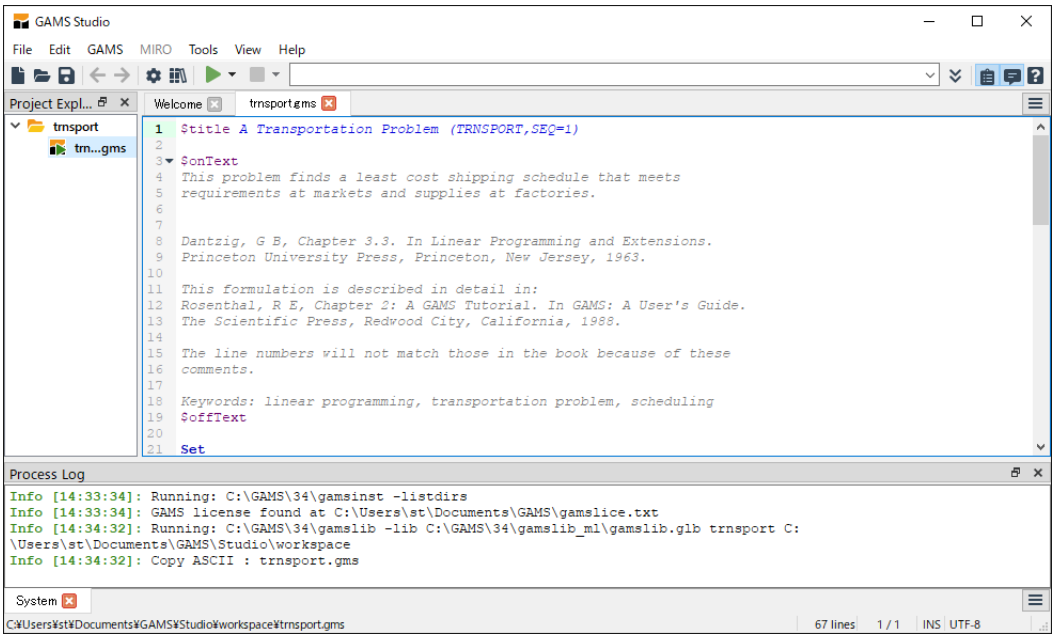
先程、GAMS をインストールしたが、正常にインストールできていれば、Windows のスタートメニューに GAMS Studio が登録されているはずである。それをクリックすれば GAMS Studio が起動する。まず最初に起動したときには以下のような画面になるはずである。



ここで、中央の「Transport Example」というボタンをクリックして欲しい。すると、GAMS の利用方法を学ぶ際によく利用される「trnsport.gms」というプログラムが開かれる。GAMS Studio でプログラムファイルを開くと、以下のように一つのタブとして開かれることになる（赤で囲んだ部分がタブ)。

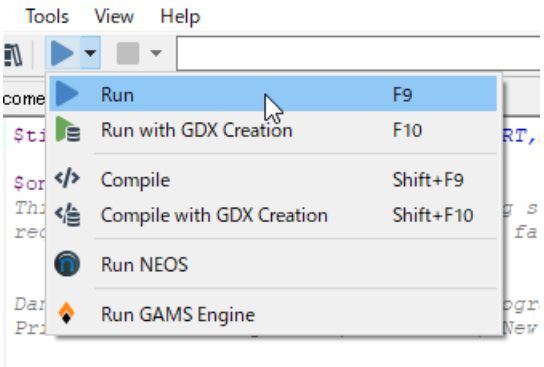


左側の 「Project Explorer」という窓（ペイン）には現在、開いているファイルが表示される。また、右側の「Process log」という窓には GAMS を実行した際の様々なログ（情報）が表示される。最初の段階ではこの画像のように右側に Process log 窓が表示されるが、各窓の表示場所は変更可能であり、例えば以下のように下に表示することもできる。

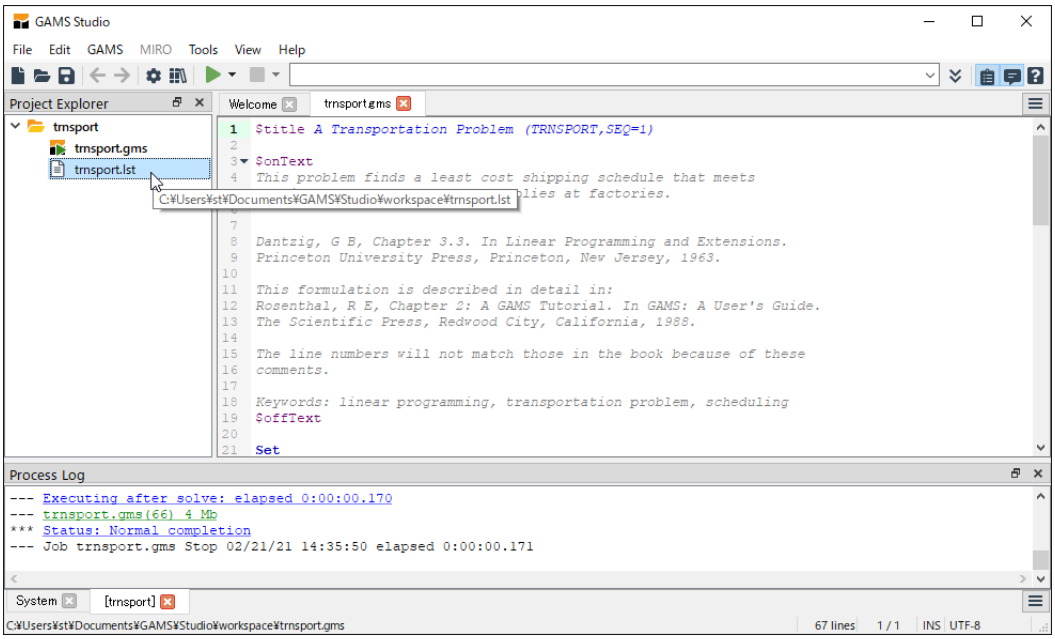


このような表示にするには、Process log 窓のタイトル部分をドラッグして下側に持ってくればよい。

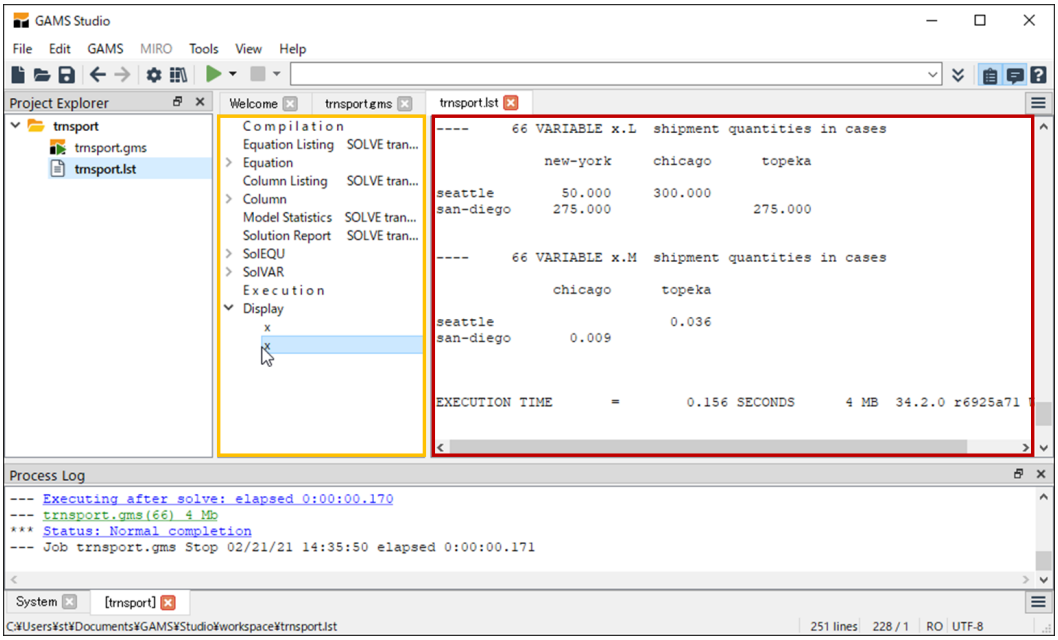
次に、現在、開いている `trnsport.gms` というプログラムを GAMS で実行してみよう。プログラムを実行するにはメニューバーの下のツールバーの緑色の△ボタンを押す（F9 キーを押すのでもよい）。



GAMS を実行すると、Process log 窓にログが表示されるとともに、左側の Project Explorer 窓に `trnsport.lst` というファイルが表示される。この「`.lst`」という拡張子のファイルが GAMS の計算結果が出力されるファイルであり、**「LST ファイル」**と呼ばれる。



Project Explorer 窓の `trnsport.lst` ファイルという部分をクリックすると、以下のようにその内容が表示される。赤く囲まれた窓が `trnsport.lst` の中身で、真中のオレンジ色で囲まれた部分は `trnsport.lst` の見出し部分をまとめた窓である。

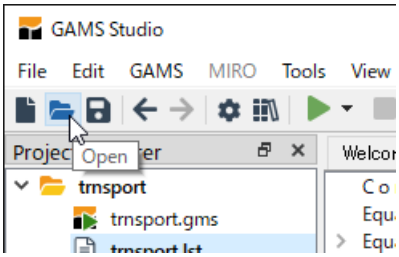


このようにして LST ファイルを見ることで計算の出力をチェックする。

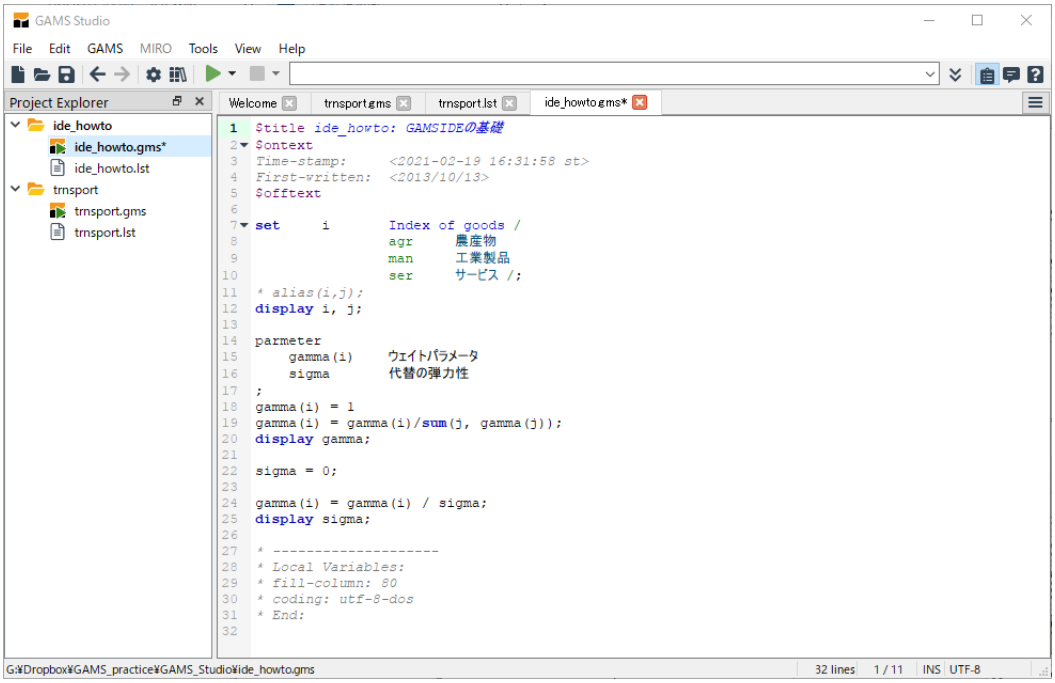
3.2.2 `ide_howto.gms` を使った説明

以下では、このファイルと一緒に配布されている `ide_howto.gms` というファイルを使って、GAMS Studio の利用方法を説明しよう。

新たにファイルを開くには、以下のようにツールバーのボタンをクリックする。



ide_howto.gms を選択し、開くと次のような表示になる。



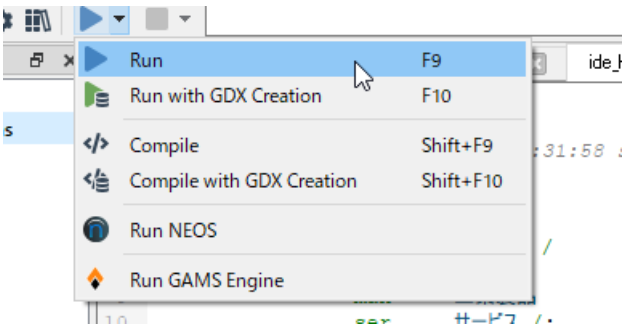
[注 1] ここでもし日本語部分が文字化けしているようであれば、適切な文字コードが選択されていない可能性がある[7]。別の文字コードを選択するには、メニューの「Edit」→「Encoding」→「Reload with」で別の文字コードを選択する。最初は「UTF-8」が選択されていると思うが、これを「Shift JIS」などに変えてみる。

[注 2] 新たに自分でプログラム（日本語を含むプログラム）を作成する際には、既定値の「UTF-8」という文字コードで作成すればよい。

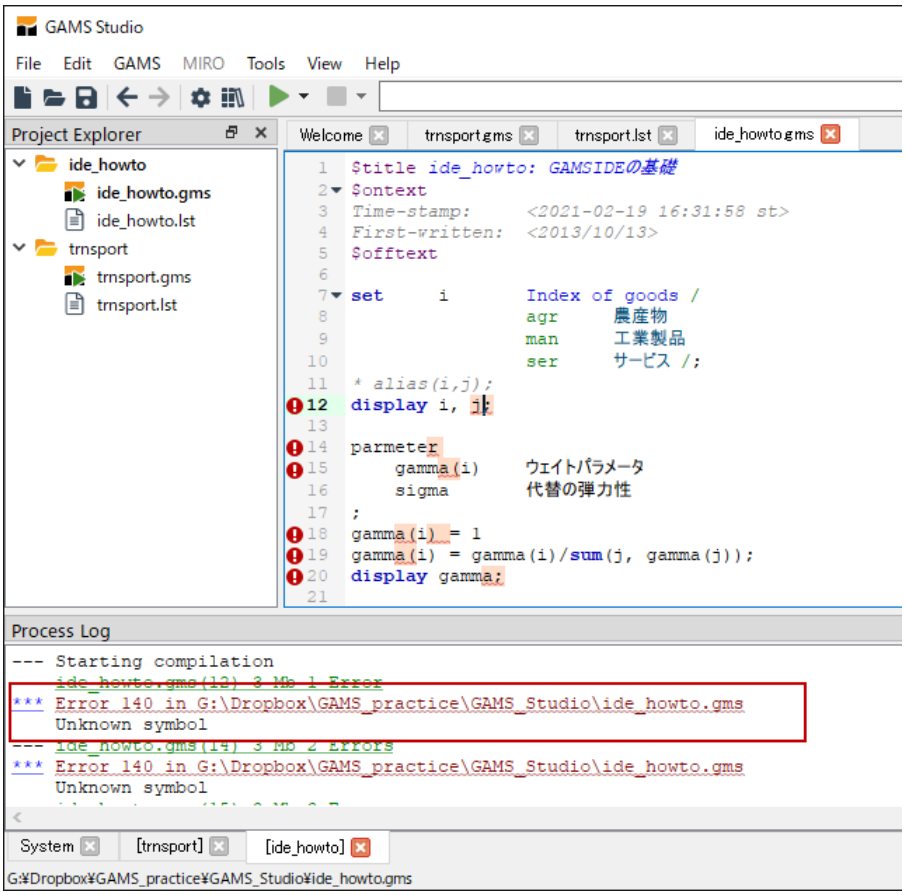
この ide_howto.gms を GAMS で実行してみよう。実行するには、再びツールバーの△ボタン

7) ファイルの文字コードが正しく認識されず、別の文字コードのファイルとして開かれてしまっているということ。GAMS Studio は文字コードの既定値として UTF-8 を用いる。GAMS IDE は日本語として Shift JIS コードしか対応していなかったので、GAMS IDE で作成したファイルを GAMS Studio で開くと文字化けする可能性が高い。その場合は、Shift JIS を選択すればよい。
GAMS での文字コードについては、というページでも扱っている。

を押す。



`ide_howto.gms` を実行すると、以下のように Process log 窓にたくさんの赤いメッセージが表示されるはずである。このように GAMS を実行した際にログに表示される**赤字のメッセージはエラーメッセージ**を表している。



大量のエラーメッセージが表示されているが、GAMS **ではエラーが出たら、一番上のエラーを見ればよい**。一度エラーが生じると、その後の誤りがない箇所もエラーとみなされることになり、エラーメッセージが大量に出ることになるが、基本的に最初のエラーしか意味がないので、最初のも

のだけを見ればよい。今回は次のようなメッセージが出ている。

```
*** Error 140 in G:Dropbox\GAMS_practice\GAMS_Studio\ide_howto.gms
    Unknown symbol
```

「Error 140」というメッセージの 140 という番号は**「エラー番号」**である。エラーの種類によってこの番号が変わる。エラーの意味はそのすぐ下に表示されている。今回は「Unknown symbol」という種類のエラーである。これは宣言（定義）されていないシンボルを使ったという意味である。

エラーメッセージの赤い行をクリックすると元の GAMS のプログラムファイル（gms ファイル）にジャンプできる。実際にクリックすると、`ide_howto.gms` の 12 行目の「`diplay i, j;`」という部分の「`j`」の後ろにカーソルが移動する。これは「`j`」が unknown symbol、つまり「`j`」が宣言されていないということを意味する。「`diplay i, j`」の一行上の `alias` 命令の前のコメント記号をとれば、`j` が宣言・定義されるので、そのように修正する[8)]。

```
10                      ser      サービス /;
11   * alias(i,j);
12   display i, j;
13
14   parmeter
```

修正後、また実行する（実行ボタンをクリックするか、F9 を押す）。すると、先程のエラーは消えるはずだが、また「unknown symbol」エラーが生じているはずである。先程と同様にクリックして GAMS のファイルに戻る。すると、次のような部分の「`parmeter`」の後ろに戻るはずである。これは「`parmeter`」が「unknown symbol」だということを意味している。これは「`parameter`」という命令の誤植であるので修正する。修正したらまた実行してみる。

```
9                       man      工業製品
0                       ser      サービス /;
1   alias(i,j);
2   display i, j;
3
4   parmeter
5       gamma(i)        ウェイトパラメータ
6       sigma           代替の弾力性
```

[注]「`parmeter`」が誤植であることは、上のようにプログラムを実行したときにエラーが生じることで確認できる。しかし、実行する前でも、「`parmeter`」が色付けされないことで誤植であることがわかる。GAMS Studio では GAMS の命令を色付けして表示する。色付けされていないことは

8) `alias(i,j)` という命令は `j` を `i` と同じものとして定義するという命令である。

命令と認識されていないことになるので、それで誤植を発見することができる。

再び実行すると次のようなエラーメッセージが出る。今度は先程とは違う「unrecognizable item」というエラーである。この部分で「;」を探したが見つからないというような意味である。再び赤い行をクリックして gms ファイルに戻る。

```
--- Starting compilation
--- ide_howto.gms(19) 3 Mb 1 Error
*** Error 409 in G:\Dropbox\GAMS_practice\GAMS_Studio\ide_howto.gms
    Unrecognizable item - skip to find a new statement
      looking for a ';' or a key word to get started again
--- ide_howto.gms(31) 3 Mb 1 Error
```

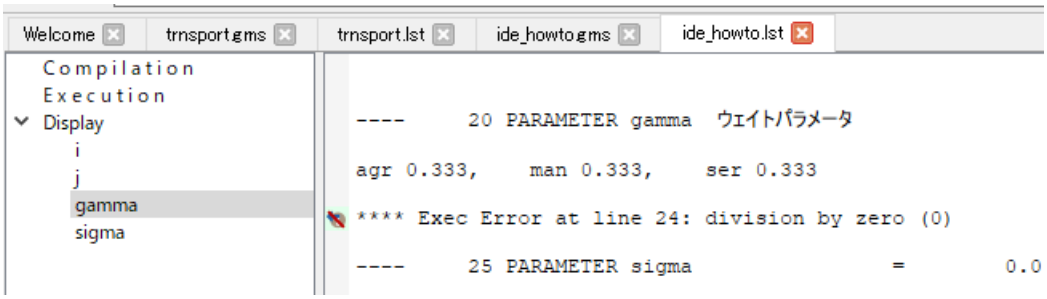
gms ファイルでは次の部分の gamma の後ろにジャンプする。ここは「gamma(i) = 1」の最後に「;」がないことがエラーの原因である。GAMS では命令の区切りの部分に「;」を書くルールになっている（ただし、省略可能な場合もあるが）。ここでは「;」がないので、次の行と連続していると判断され、そのためエラーが生じている。「;」を 1 の後に加え、再び実行する。

```
16      sigma        代替の弾力性
17  ;
18  gamma(i) = 1
19  gamma(i) = gamma(i)/sum(j, gamma(j))
20  display gamma;
```

実行すると、以下のように「Error at line 24」というエラーメッセージが生じる。エラーはエラーであるが、今回のエラーはこれまでのエラーとは少し性質が異なる。これまでのエラーは、「Compilation error」と呼ばれるタイプのエラーで文法が間違っているという意味のエラーであった。これに対し、ここでのエラーは、「Execution error」というプログラム実行時に生じるエラーである。具体的には、「division by zero」とあるように、プログラムの 24 行目において計算の過程で分数の分母に 0 が入ってしまったということによるエラーである。Execution error の場合には、エラー行の表示の下に「Status: Execution error」というような表示も追加される。

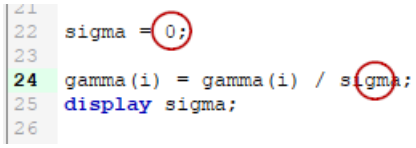
```
--- Starting execution: elapsed 0:00:00.004
--- ide_howto.gms(24) 4 Mb
*** Error at line 24: division by zero (0)
--- ide_howto.gms(25) 4 Mb 1 Error
*** Status: Execution error(s)
--- Job ide_howto.gms Stop 02/21/21 14:52:39 ela
```

Execution error の場合、エラーメッセージをクリックすると、gms ファイルではなく、以下のように計算結果が出力される LST ファイルの方にジャンプする。

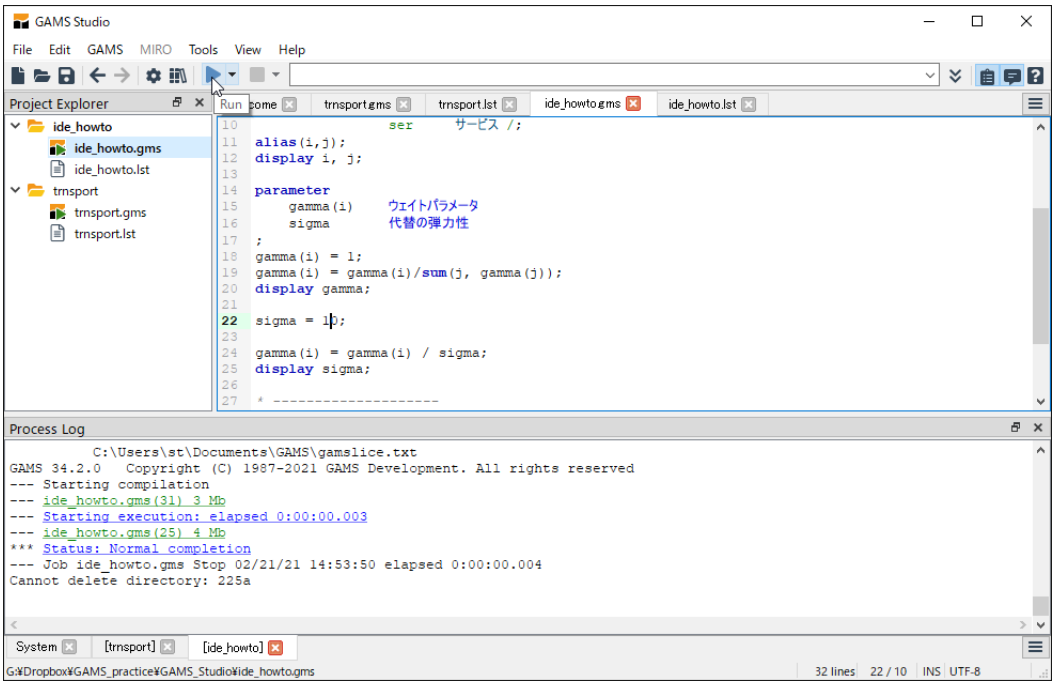


gms ファイルのエラー箇所にジャンプするには、エラーメッセージの行ではなく、その前の緑色の「ide_howto.gms(24) 4Mb」の行をクリックすればよい。

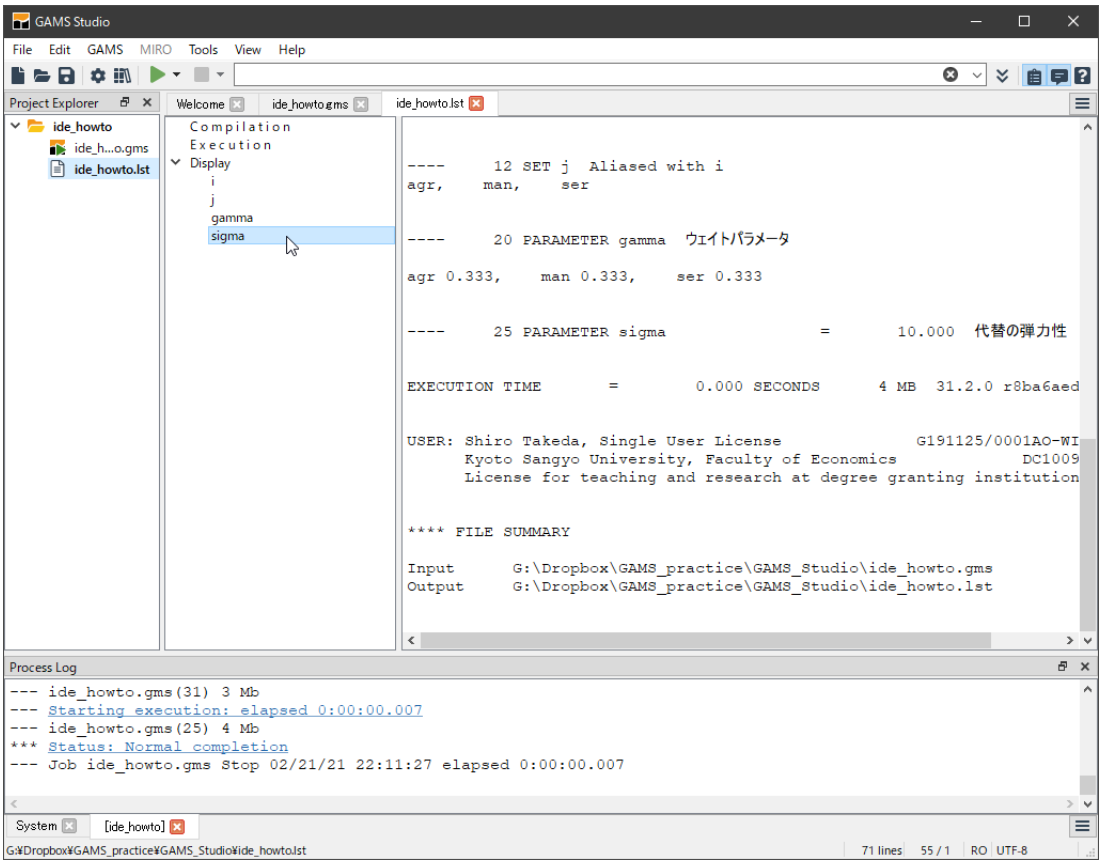
gms ファイルの 24 行目は以下の「gamma(i) = gamma(i) / sigma;」という行である。sigma が分母に入っているが、そのすぐ上の部分で sigma に 0 が代入されている。これが「division by zero」エラーの原因である。とりあえず、sigma に 0 ではない数値（例えば、10）を代入するように修正する。修正したら再び実行する。



今回は「Status: Normal completion」というメッセージで終わる。これは問題なく終了したということである。



問題なく終了した場合には、計算結果を確認すればよい。先程既に見たが、計算結果は LST ファイル（ide_howto.lst）というファイルに出力されるので、そちらを見て確認する。



以上が GAMS Studio で GAMS のプログラムを実行する基本的な手順である。特に、エラーが生じたときには

1) 一番上のエラーメッセージを確認
2) GMS ファイルのエラー箇所にジャンプ
3) エラーを修正し、再び実行

を繰り返し、エラーを除去していくという作業をおこなえばよい。

4 GAMS の利用法

4.1 GAMS の基礎

まず、GAMS における基本事項の説明をおこなう。

4.1.1 プログラムのファイルについてのルール

- GAMS のプログラムの**ファイルの拡張子は通常「gms」とする**。
  - ただし、他の拡張子を選択できないということではない。どのような拡張子も選択することはできる。
- 計算結果の出力ファイル
  - GAMS のプログラムを実行すると、計算結果は別のファイルに出力される。
  - 例えば、「sample.gms」というファイルを実行すると、「sample.lst」というファイルに結果が出力される。この計算結果が出力される、**「lst」という拡張子を持つファイルを LST ファイルと呼ぶ**。

4.1.2 GAMS の文法についての基本的ルール

- 変数などは全て**「事前に宣言（declaration）」**する必要がある。
- 大文字小文字の区別はおこなわれない。
  - 例えば、「price」、「Price」、「PRICE」は全て GAMS では同じと認識される。
  - これは GAMS の命令であろうが、変数やパラメータの名前であろうが同じである。
- 一続きの命令のかたまりの最後にはセミコロン「;」を付ける。
  - GAMS では**セミコロンが次に続く命令との区切りを表す記号**になっている。
  - このセミコロンは場合によって省略可能であるが、**エラーを減らすためにも最初はできるだけ省略せずに書くのがよい**。
- コメント行
  - アスタリスク「*」で始まる行はコメント行となる。
    - ［注］「*」は掛け算の記号でもあるが、行頭にある場合にはコメント行を表す記号になる。
  - $ontext と$offtext で囲まれた領域もコメント行となる。複数行に渡るコメントを記入するときにはこちらを使えばよい。

4.1.3 GAMS における重要な用語

- **集合**
  - $j = \{1, \cdots, n\}$、$f = \{\text{labor}, \text{capital}, \text{land}\}$ という表現の j、f にあたるものは GAMS では「set」と呼ばれる。以下ではこれを**「集合」**と呼ぶ[9)]。
- **パラメータ**

9) j や f は集合そのものではなく、集合の一要素を表わすインデックスのようなものであるが、以下では集合と呼ぶ。

- モデルにおいて外生変数にあたるもの、あるいは外生的なパラメータにあたるものは GAMS では「parameter」と呼ばれる。
- 例えば、効用最大化問題における「価格」と「所得」のような外生変数[10]、CES 効用関数内の代替の弾力性パラメータにあたるもののことである。
- 以下ではこれを**「パラメータ」**と呼ぶ。

- **変数**
  - モデルの内生変数にあたるものは GAMS では「variable」と呼ばれる。例えば、効用最大化問題における「消費量」のような変数である。
  - 以下ではこれを**「変数」**と呼ぶ。
- **式**
  - モデルを構成する方程式を GAMS では「equation」と呼ぶ。以下ではこれを**「式」**と呼ぶ。
- **［注］**
  - 最初に述べたように、「集合」、「パラメータ」、「変数」等はどれも、まず「宣言」しておく必要がある。
  - 上の説明において「変数」は内生変数にあたるものと説明したが、実際には「変数」として宣言したものを外生変数として利用することもできる。ただし、基本的には上の説明のように使い分ける。

4.1.4 変数の値

上で説明したように GAMS ではモデルの内生変数の対応するものを**「変数（variable）」**と呼ぶ。この**変数は 4 つの値を持つ**仕組みになっている。例えば x という変数を考える。

- **レベル値**（level value）
  - これはその変数 `x` がその時点において保持している値であり、普通の意味での変数 `x` の値と考えればよい。
  - レベル値は「`x.l`」という表現で得ることができる。例えば、その時点での `x` のレベル値を表示したいなら「`display x.l;`」で表示できる。また、`x` のレベル値に 10 を代入したいのなら「`x.l = 10;`」とすればよい。
- **下限値・上限値**（lower value と upper value）
  - 各変数に対し、その下限値（lower bound）と上限値（upper bound）が設定されている。
  - それぞれ「`x.lo`」、「`x.up`」というように表す。例えば、`x` の下限値を 0 にしたいのなら、「`x.lo = 0`」というように代入する。
  - 変数の下限値のデフォールト値はマイナス無限大（GAM の記法では`-inf`）であり、変数の上限値のデフォールト値はプラス無限大（GAMS の記法で `+inf`）である。下限値、上限値がマイナス無限大、プラス無限大ということは要するに下限も上限もないと設定されているということである。

10) 価格や所得は消費者にとっては外生変数であっても、一般均衡モデルを考えた場合であればモデル全体としては内生変数となるが。

- Marginal **値**（marginal value）
  - もう一つ**「**marginal **値」**と呼ばれる値がある。
  - これはどのようなタイプのモデルかによって変わってくるので、後に（第 4.5.3 節で）説明する。

4.1.5 GAMS におけるモデルのタイプ

GAMS において実際にモデルを解く役割を果たすのは**「ソルバー（**solver**）」**と呼ばれる部分である。解く問題のタイプによって違うソルバーを選択する必要がある。GAMS が扱える問題のタイプは多様であり、全部で 15 種類程度ある。表 1 はその一部である。次の節では、非線形の連立方程式を解く問題を例として考える。GAMS における問題のタイプとしては、それは MCP というタイプに相当する。

［注］ MCP は本来、mixed complementarity problem のことであり、厳密には連立方程式を解く問題とは違う（連立方程式を解くという問題は MCP の一種にすぎない）。しかし、とりあえずは**「**MCP **＝連立方程式を解くという問題」**と考えておけばよい。MCP がどのようなタイプの問題なのかは第 4 章で詳しく扱う。

表 1：GAMS で扱える問題のタイプ（の一部）

GAMS **における表記**	**問題のタイプ**
LP	線形計画法（linear programming）。目的関数、制約式が線形の最適化問題。
NLP	非線形計画法（non-linear programming）。目的関数、制約式が非線形の最適化問題。
MCP	Mixed complementarity problem
MIP	変数が整数値をとるような最適化問題

4.2 例として取り上げる問題

以下では効用最大化問題を例として取り上げる。効用最大化問題は元々次のような**「非線形の最適化問題」**、いわゆる**「非線形計画法（**non-linear programming**）」**に分類される問題である。

$$\max_{\{d_1,\cdots,d_n\}} u = \left[\sum_i \gamma_i (d_i)^{\frac{\sigma-1}{\sigma}}\right]^{\frac{\sigma}{\sigma-1}}$$
$$\text{s.t.} \sum_i p_i d_i = m \qquad p \text{ and } m \text{ are given.}$$

ただし、u は効用、d_i は財 i の消費量、p_i は財 i の価格、m は所得、γ_i、σ は外生的なパラメータである（効用最大化問題について詳しくは第 1 章を参照して欲しい）。

しかし、この章の目的は CGE 分析における GAMS の利用法の解説である。CGE **モデルを解く（一般均衡モデルを解く）というのは基本的には連立方程式を解くという問題**であるので、「最適化

問題」の解き方ではなく、「連立方程式」の解き方を考えたい[11)]。そこで、上の効用最大化問題をそのまま最適化問題として扱うのではなく、連立方程式の問題に変換して扱うことにする。上の効用最大化問題を解くには、ラグランジュ乗数法を利用すればよい。このラグランジュアンは次のように表現できる。

$$\mathcal{L} = \left[\sum_i \gamma_i (d_i)^{\frac{\sigma-1}{\sigma}}\right]^{\frac{\sigma}{\sigma-1}} - \lambda \left[\sum_i p_i d_i - m\right]$$

これより、効用最大化の条件は次式で表すことができる。

$$\frac{\partial \mathcal{L}}{\partial d_i} = \left[\sum_j \gamma_j (d_j)^{\frac{\sigma-1}{\sigma}}\right]^{\frac{1}{\sigma-1}} \gamma_i (d_i)^{-\frac{1}{\sigma}} - \lambda p_i = 0 \quad \{d_i\}_{i=1,\cdots,n} \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\left[\sum_i p_i d_i - m\right] = 0 \quad \{\lambda\} \tag{2}$$

これを満たす $\{d_i\}$ が効用を最大化する消費量になる。これは n 本の (1) 式、1 本の (2) 式の合計 $n+1$ 本からなる連立方程式である。以下では、この連立方程式を解くという問題の GAMS での扱い方を説明する。

4.3 GAMS のコード

`utility_max.gms` が上の連立方程式を解くための GAMS のコードである。以下の説明は GAMS Studio において実際にプログラムを打ち込みながら読むようにして欲しい。プログラムの打ち込みは必ず間違えるものなので、全部を一度に打つのではなく、少し打ち込んだら GAMS を実行し、プログラムが正しいかを確認しながら少しずつ進めるようにするのがよい。

4.4 プログラムの解説

それでは以下でプログラムの解説をおこなっていこう。

```
$title 効用最大化問題
$ontext
Time-stamp:     <2022-01-11 22:25:11 st>
First-written:  <2013/10/11>
$offtext
```

まず、`$title` 命令はプログラムのタイトルを指定する命令である。これは省略してもよい。この指定が LST ファイルでプログラムのタイトルとして利用されることになる。`$ontext`〜`$offtext` は既に説明したようにコメントを挿入するための命令である。最初の部分にプログラムの説明などを書いておくのがよい。このプログラムではここにプログラムの作成日などを記入している。

11) つまり、均衡条件から成る連立方程式解くということである。ただし、CGE モデルを最適化問題として解くという方法もある。CGE モデルを最適化問題として解く方法については第 11 章で取り上げている。

4.4.1 集合の宣言・定義

```
*       -------------------------------------------------------------
*       集合の宣言・定義
set     i       Index of goods /
                agr     Agricultural goods
                man     Manufacturing goods
                ser     Services /;
```

ここでは set という命令によって i という集合を宣言・定義している。集合は「set」（あるいは、「sets」）という命令によって宣言・定義される[12)]。「set」命令の基本的な書式は次の通りである。

```
set     i       集合の説明 /
                i1      i1 の説明
                i2      i2 の説明
                i3      i3 の説明 /;
```

スラッシュ「/」で囲まれた中で集合の要素を指定する（緑色の部分）。上の定義では集合 i は i1、i2、i3 の三つの要素を含むことになる。

「集合の説明」（赤色の部分）は集合 i の説明文である。**集合の名前のすぐ後に記入された部分が説明文となる。**「i1 の説明」等の部分（オレンジ色の部分）は集合の要素に対する説明文である。「集合の説明」や「集合の要素の説明」は省略可能である。よって、仮に必要最小限の形にした場合次のように書くことができる。

```
set     i       / i1, i2, i3 /;
```

一行で集合の要素を複数指定したいなら、上のように集合の要素を「カンマ」で区切って並べればよい。set 命令について詳しくは GAMS User Guide を見て欲しい。

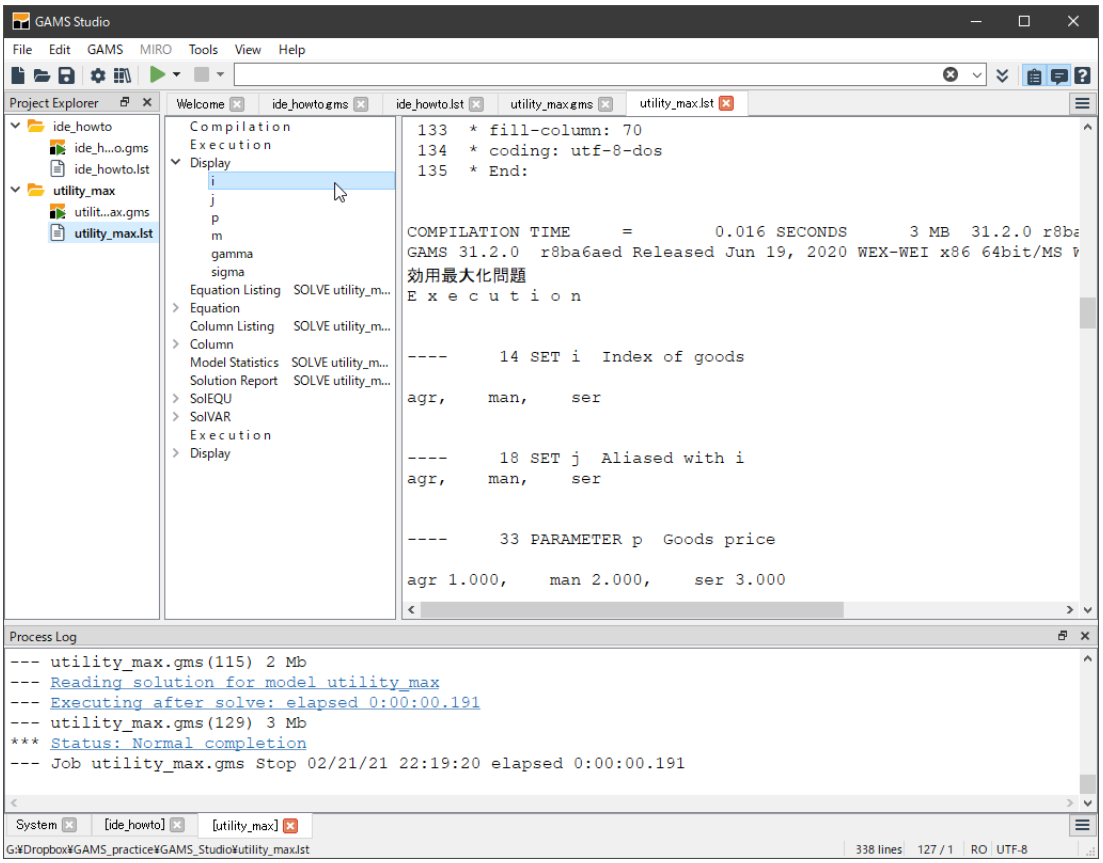
4.4.2 display 命令

```
display i;
```

これは i の中身を表示するという命令である。集合に限らず、何か（変数、パラメータ、集合等）の中身を表示するにはこの display 命令を利用する。この display 命令が記述されている部

12) set は集合を定義する命令であるが、複数にした sets も全く同じ機能を持つ命令であり、どちらを用いてもよい。以下で出てくる parameter、variable、equation なども同じで、複数にした parameters、variables、equations は全く同じ機能を持つ。

分については、実際に GAMS Studio で実行した結果においてどのような出力になっているか確認するようにして欲しい。例えば、上の部分の出力結果は次のような表示になるはずである。ちゃんと i の中身が agr、man、ser になっているか自分で確認して欲しい。



4.4.3 普通の命令と$付きの命令の違い（※）

ここまで命令として、$title のように $ マークが付く命令と、set のようにアルファベットのみからなる命令の二つが出てきた。前者のタイプの命令は「Dollar Control Option」と呼ばれる。両者の違いは、普通の命令が主に計算時に実行されるのに対し、$ 付き命令は計算の前に実行されるという点である。正確に言えば、dollar control option は GAMS の「compilation time」に実行されるのに対し、普通の命令は「execution time」に実行されるということになる。また、普通の命令は GAMS での計算に関係する命令であるのに対し、$ 付きはプログラミングに関係する一般的な命令であることが多い。

4.4.4 alias 命令（集合のコピー）

```
*       alias 命令によって i と同じ中身を持つ j を宣言
alias(i,j);
display j;
```

この alias というのは i と同じ中身を持つ j という集合を宣言する命令である。全く同じ要素を持った別の名前の集合を作成したいときには、この alias 命令を利用すればよい。

[集合についての注]：上のように直接要素を指定して定義する集合は「static set」と呼ばれる集合である。static set は一度定義すると中身が変わらない集合である。一方、可変的な要素を持つ「dynamic set」と呼ばれる集合もある。それについてまた別のところで説明する。

4.4.5 パラメータの宣言・定義

```
$ontext
外生変数である価格と所得を表すパラメータ。
$offtext
parameter
    p(i)        Goods price
    m           Income
;
```

ここでは p(i) と m という 2 つのパラメータを宣言している。パラメータは「parameter」という命令で宣言する。「parameter」命令の一般的な書式は次の通りである。

```
parameter
    a(i)        a の説明
    b           b の説明
;
```

上のコードでは a(i) と b という二つのパラメータが宣言されている。「a の説明」、「b の説明」（オレンジ色の部分）はそれぞれパラメータ a(i) と b の説明文である。

a というパラメータは、それが集合 i の要素に対して一つずつの値を持つものとして宣言されている。例えば、集合 i の要素が i1、i2、i3 であれば、a("i1")、a("i2")、a("i3") の 3 つが宣言されるということ。宣言部分では単に a というようにしておいて、後で a("i1") = 1 というように使うこともできるが、初めから a(i) と宣言しておいた方がどのようなパラメータなのか明確になるというメリットがある。

集合のときと同様に説明文は省略可能であり、カンマで区切って並べれば一行で複数のパラメータが宣言できる。よって、

```
parameter a(i), b;
```

というような宣言も可能である。

```
p("agr") = 1;
p("man") = 2;
p("ser") = 3;
m = 100;
display p, m;
```

この部分では宣言されたパラメータ p と m に実際に値を代入し、その中身を display 命令によって表示させている。p は集合 i 上で定義されたので、i の各要素に対して p の値が代入されている。

このプログラムでは、パラメータの宣言と値の代入が別々に行われているが、これは同時に行うこともできる。同時に行うには次のように書けばよい。

```
Parameter
    p(i)        Goods price / agr   1
                              man   2
                              ser   3 /
    m           Income      / 100 /
;
```

パラメータの宣言部分で同時に値を設定するときには上のように「/」の記号を用いる。

```
$ontext
その他に外生的に与える値。
$offtext
parameter
    gamma(i)    Weight parameter
    sigma       Elasticity of substitution
;
*       gamma に値を代入
gamma(i) = 1;
gamma(i) = gamma(i)/sum(j, gamma(j));
display gamma;

sigma = 0.5;
display sigma;
```

この部分も前の部分と同様にパラメータの宣言・値の代入をしている。具体的には gamma と

sigma という効用関数内のパラメータを宣言、値を代入している。

```
gamma(i) = 1;
gamma(i) = gamma(i)/sum(j, gamma(j));
```

この部分の計算により、各 gamma(i) は結局 1/3 という値を持つことになる（出力結果で確認して欲しい)。2 行目の書き方からわかるように、他のプログラミング言語と同様、GAMS においても**「= 記号は等号ではなく、代入を表す記号（演算子)」**として使われる。

4.4.6 変数の宣言

```
$ontext
内生変数は variables 命令で宣言
$offtext
variables
    d(i)        Demand
    lambda      Lagrange multiplier
;
```

この部分ではモデルの内生変数にあたるもの（変数）を宣言している。変数は「variables」命令（あるいは、「variable」命令）で宣言する。ここでは需要量を表す d(i) とラグランジュ未定乗数を表す lambda の二つを宣言している。

- 「variables」命令の書式は「parameter」命令の書式とほぼ同じである。
- ただし、「variables」命令の場合は宣言と同時に値を代入するということはない。

4.4.7 式の宣言

```
*       ---------------------------------------------------------------
*       式の宣言
$ontext
式は equations 命令で宣言

式の名前は「e_ + その式に対応する変数名」としておく。
$offtext
equations
    e_d(i)      1st order condition for d(i)
    e_lambda    1st order condition for lambda
;
```

ここでは式（equation）を宣言している。式は「equations」命令で宣言する。「equations」命

令の書式は「variables」命令の書式とほぼ同じである。

式の名前は基本的に自由であるが、ここでは

- ラグランジュアンを需要量で偏微分した (1) 式に対応する式を e_d(i)
- ラグランジュアンを lambda で偏微分した (2) 式に対応する式を e_lambda

という名前にしている。説明文の部分についてはこれまでと同様である。

4.4.8 式の定義

```
*       ------------------------------------------------------------
*       式の定義

e_d(i) .. (sum(j, gamma(j)*(d(j))**((sigma-1)/sigma)))**(1/(sigma-1))
          * gamma(i) * (d(i))**(-1/sigma) - lambda * p(i) =e= 0;

e_lambda .. - (sum(i, p(i)*d(i)) - m) =e= 0;
```

ここは宣言された式の定義をおこなっている。つまり、実際どのような式か指定されている。式の定義は次のような形式でおこなわれる。

```
式の名前 .. 定義;
```

つまり、まず 1) 式の名前を書き、2) ピリオドを 2 つ書き、その後、3) 式の定義を書き、4) 最後に「;」を付けるという形式である。注意点としては、式の定義の中で「=」、「≥」、「≤」を表現するにはそれぞれ「=e=」、「=g=」、「=l=」という表記を用いるということである[13)]。

式の中身は (1) 式、(2) 式そのままである。

4.4.9 モデルの宣言・定義

```
*       ------------------------------------------------------------
*       モデルの宣言・定義
$ontext
モデルに含まれる式を指定することによってモデルを定義。

「式の名前. 式に対応する変数名」という形式で指定。例えば

    e_d.d

というのは、「e_d」という式に対して、「d」という変数を対応させるという
こと。
```

13) 第 4.4.5 節で説明したように、GAMS のプログラムでは「=」は等号ではなく、代入の記号である。

```
$offtext
model utility_max Utility maximization / e_d.d, e_lambda.lambda /;
```

ここではモデルの宣言と定義をしている。モデルの場合、宣言と定義は同時におこなう。モデルの宣言は「model 命令」によっておこなう。基本的な書式は次の通りである。

```
model モデル名 説明文 / 式の指定 /;
```

上の例では、「utility_max」がモデル名、「Utility maximization」が説明文である。

モデルの宣言はそのモデルがどの式によって構成されるかを指定することでおこなう。よって、式の名前を列挙するという形式になる。さらに、MCP タイプのモデルでは式と変数の対応関係も指定する。これは、**「式の名前. 変数名」**という形で指定する。上のコードでは次のようにモデルが指定されている。

```
/ e_d.d, e_lambda.lambda /
```

この指定ではまず「e_d」と「e_lambda」の 2 つの式が指定されている。つまり、モデルはこの 2 本の式から構成されるということである。さらに、式「e_d」に対しては変数「d」が、式「e_lambda」に対しては変数「lambda」が対応させられている。このように「式と変数」の組み合わせによってモデルを定義する必要がある理由については詳しくは第 4 章で説明する。とりあえずは、式に対してはその式に対応する変数を結びつける形でモデルを指定するということだけ覚えておけばよい。まとめると、モデルは次のように定義する。

1) モデルを構成する式を指定する（式の名前を列挙する）
2) 各式に対し、その式に対応する変数を指定する。

[モデルの定義についての注]

式と変数を組み合わせてモデルを定義するのは MCP というモデルのタイプに特有の方法であり、例えば最適化問題（NLP）等の場合は式のみを指定すればよい。MCP タイプのモデルであっても、必ずしも式に変数を対応させなくてもいい場合があるが、これも詳しくは第 4 章で説明する。

4.4.10 変数の下限値・レベル値の設定

```
*       ---------------------------------------------------------------
*       変数の下限値を指定
$ontext
「変数名.lo」が変数の下限値を表す。
$offtext
```

```
d.lo(i) = 0;
lambda.lo = 0;

*       ---------------------------------------------------------------
*       変数の初期値を指定
$ontext
「変数名.l」が変数のその時点における値を表す。この後、すぐにモデルを解くので、こ
こで指定した値がモデルを解く際の初期値として利用されることになる。
$offtext
d.l(i) = 10;
lambda.l = 10;
```

ここではまず**「変数の下限値」**を指定している。第 4.1.4 節で説明したように下限値は「変数名.lo」に値を代入することで指定する。変数 d（消費量）も lambda （ラグランジュ乗数。このケースでは所得の限界効用に等しい）も非負の値をとるべき変数であるので、下限値に 0 を指定している。もし、下限値に何も設定しなければマイナス無限大が設定される（つまり、下限はなしということになる）。

さらに、ここでは**「変数の初期値」**も設定している。最後の 2 行では変数のレベル値を設定している。「レベル値」とは、プログラムのその時点での変数の値であり、普通の意味での変数の値のことであった（第 4.1 節参照）。この後、すぐにモデルを解く命令があるので、**ここで指定した変数の値がそのままモデルを解く際の変数の初期値として利用されることになる。**よって、このレベル値の指定は変数の初期値を指定していることになる。

d と lambda のどちらにも初期値として 10 を設定している。通常、初期値が解の値に近いほどモデルを解きやすくなるので、解がどのような値をとるかを事前に推測できるのなら、その情報を元に変数の初期値を設定することが望ましい。ここでは解の値について特に情報はないので、適当に初期値を設定している（つまり、10 という値の根拠は特にないということ）。

4.4.11 solve 命令

```
*       ---------------------------------------------------------------
*       モデルを解く
option mcp = path;
solve utility_max using mcp;
```

ここではモデルを解いている。モデルを解くには solve 命令を利用する。どのようなタイプのモデルか（MCP、NLP、LP か）によって solve 命令の書式が変わってくるが、MCP タイプのモデルの場合は次のように書けばよい。

```
solve utility_max using mcp;
```

つまり、「solve モデル名 using mcp;」というように書けばよい。「option mcp = path;」というのはソルバーを指定している命令である。これは MCP タイプの問題を解くのに PATH というソルバーを使うという意味である[14)]。デフォールトのソルバーが設定されており、それが PATH である場合には、このように指定しなくても PATH が使われることになるが、ここでは念のため明示的に PATH を指定している。

4.4.12 結果の表示

モデルを解き、それが正常に解けた場合には、解いた結果の変数の値が出力される。しかし、分析では全ての変数の値に興味があるのではなく、一部のみに興味がある場合がほとんどである[15)]。このような場合には、関心がある変数の値のみを抜き出して表示させるようにした方がよい。

それには、結果を表示させるためのパラメータを宣言し、変数の値をそれに代入して、表示させるようにする。以下はそのような作業のためのコードである。

```
Parameter
    results      結果をまとめて表示するためのパラメータ
;
results(i,"price") = p(i);
results("/","income") = m;
results(i,"demand") = d.l(i);
results("/","lambda") = lambda.l;
results(i,"exp") = p(i)*d.l(i);
results("sum","exp") = sum(i, results(i,"exp"));
results("/","utility") =
    (sum(i, gamma(i)*(d.l(i))**((sigma-1)/sigma)))**(sigma/(sigma-1));

display results;
```

上のコードでは results というパラメータを宣言して、そこにモデルの解を代入している。d(i) と lambda という変数の値を results に代入しているが、変数の値（レベル値）を取り出すには「.l」を付けた「d.l(i)」、「lambda.l」というような表記を利用する必要がある。

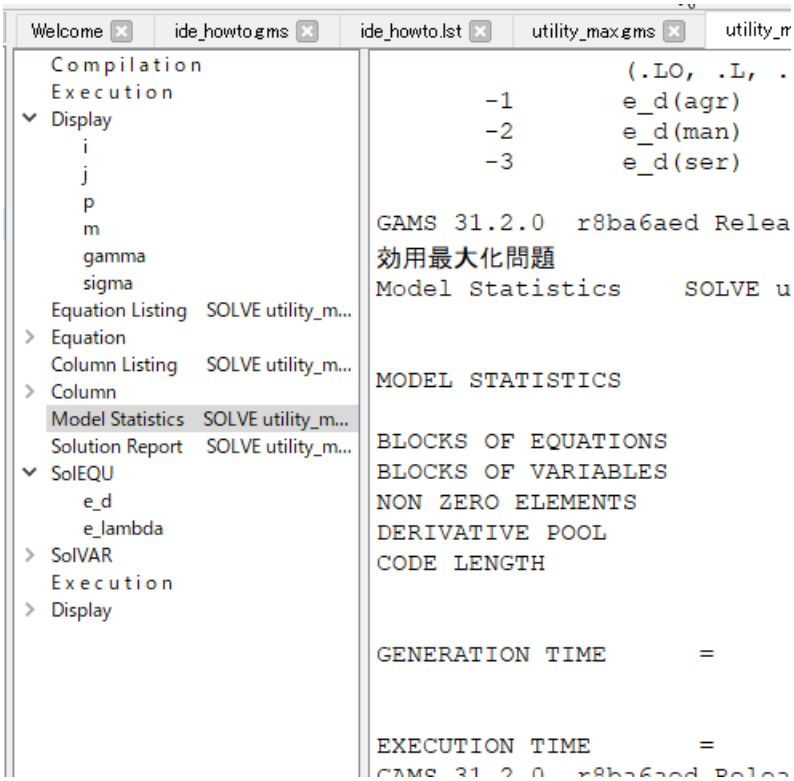
4.5 計算結果の見方

第 4.3 節のコードを実行し、特にエラーが生じなかったとすると、解が計算できているはずである。以下では、MCP タイプの問題を解いた場合の計算結果の確認方法を解説する。モデルの結果をチェックするにはとりあえず「Solution Report」と「REPORT SUMMARY」の二つの部分を見れ

14) 同じタイプの問題に複数のソルバーが対応している場合は、ソルバーの指定が可能になる。例えば、MCP タイプは PATH と MILES の二つのソルバーが対応している。

15) CGE モデルでは変数の数が数万個に及ぶことも少なくはないが、通常、分析上、関心があるのはそのうちのほんの僅かの変数である。

ばよい。



4.5.1 Solution Report

まず，計算したら「Solution Report の SOLVE SUMMARY ブロック」を見る．ここに以下の情報がレポートされている。

- 「MODEL」の部分に解いているモデル名
- 「TYPE」の部分に解いているモデルのタイプ（ここでは MCP）
- 「SOLVER」の部分に利用しているソルバー名（ここでは MCP ソルバーである PATH を利用）

特に、重要なのは「SOVER STATUS」と「MODEL STATUS」の値である。以下の例のように二つの STATUS がともに 1 になっていれば正常にモデルが解けているということである。

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

     MODEL   utility_max
     TYPE    MCP
     SOLVER  PATH                FROM LINE  115

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

一方、以下のように「SOVER STATUS」と「MODEL STATUS」が 1 になっていない場合はモデルが正常に解けていないことを意味する。まずはこの部分をチェックし、モデルが正常に解けているかどうかを確認する。

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

     MODEL   utility_max
     TYPE    MCP
     SOLVER  PATH                FROM LINE  115

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

utility_max.gms のプログラムでは正常に解けるはずである。もし正常に解けなかったら、それはどこかでプログラムを打ち間違えているはずである。

4.5.2 変数の値のチェック

モデルが正常に解けているのなら変数の値を確認すればよい。変数の値を見るには左側の窓の「SolVAR」という部分の変数をクリックすればよい。変数の値を表示している部分が右側の窓に表示される。

```
---- VAR d  Demand

       LOWER     LEVEL     UPPER    MARGINAL

agr      .       24.118     +INF  2.4569E-7
man      .       17.054     +INF  -1.246E-7
ser      .       13.925     +INF  -2.042E-8

                      LOWER     LEVEL     UPPER    MARGINAL

---- VAR lambda          .        0.175     +INF       .

  lambda  Lagrange multiplier
```

第 4.1 節で指摘した通り、GAMS では各変数に対して、「下限値（lower value）」,「上限値（upper value）」,「レベル値（level value）」,「marginal 値（marginal value）」という「4 つの値」が関連付けられている。それぞれの値が LST ファイルにおける「LOWER」、「LEVEL」、「UPPER」、「MARGINAL」の部分に表示されている。

上の例では、変数 d("agr") の下限値は 0（注：LST ファイルの計算結果におけるピリオドは 0

を表している)、上限値は無限大(+INF は正の無限大を表す記号)、レベル値は 24.118、marginal 値はほぼゼロ(2.4569E-7=2.4569/10^7)となっている。他の変数についても同様に解釈すればよい。

4.5.3 MCP モデルにおける Marginal 値の意味

Marginal 値の値はモデルによって意味が違ってくる。MCP **モデルでは変数の** marginal **値は、その変数が対応づけられた式の「乖離幅」を表している**。例えば、d(i) という変数を考える。モデルの定義の部分(第 4.4.9 節)で見たように、この変数は e_d(i) という式に対応づけられている。e_d(i) は次のように定義されていた。

```
e_d(i) .. (sum(j, gamma(j)*(d(j))**((sigma-1)/sigma)))**(1/(sigma-1))
          * gamma(i) * (d(i))**(-1/sigma) - lambda * p(i) =e= 0;
```

「乖離幅」とはこの式の右辺・左辺の乖離幅のことである。例えば、d(i) の marginal 値が 10 である場合は

```
(sum(j, gamma(j)*(d(j))**((sigma-1)/sigma)))**(1/(sigma-1))
          * gamma(i) * (d(i))**(-1/sigma) - lambda * p(i) = 10
```

となっていることを意味する。

上の例では d("agr") の marginal 値はほぼゼロであった。これは e_d("agr") という式の乖離幅がゼロということであり、それは e_d("agr") という式が等号で満たされているということである。モデルが正常に解けているのなら、e_d("agr") は等号で満たされていなければいけないが、d("agr") の marginal 値によって実際にそれが満たされていることを確認できる。

モデルが正常に解けているような場合には基本的にレベル値のみをチェックすればよい。しかし、モデルの作成段階では marginal 値の情報を利用してモデルをデバッグ(モデルの誤りを修正)することができる。このデバッグ方法については、第 9 章で詳しく説明する。

5 GAMS のエラーメッセージ

追加予定

とりあえず、compilation error については "Fixing Compilation Errors" を、execution error については "Finding and Fixing Execution Errors and Performance Problems" を見るのがよい。

6 問題

問題 1

以下のような支出最小化の問題を考える。

$$\min_{\{h_1,\cdots,h_n\}} e = \sum_i p_i h_i$$
$$\text{s.t.} \quad \left[\sum_i \gamma_i (h_i)^{\frac{\sigma-1}{\sigma}}\right]^{\frac{\sigma}{\sigma-1}} = u$$

この問題を連立方程式の問題に書き換え、GAMS で解きなさい。パラメータ γ_i、σ の値、価格の値は第 4.3 節で利用されている値をそのまま利用しなさい。また、u の値には第 4.3 節の問題において最大化された効用の値として計算される値を設定しなさい。

7 履歴

- 2022-01-12: 説明の追加・修正。
- 2021-02-22: GAMS IDE の説明を GAMS Studio の説明に変更。
- 2018-07-20: 説明の追加・修正。
- 2017-03-15: 説明の修正。リンクのアップデート。