GAMSマニュアル

最終更新日: 2020/03/13

本ファイルは、GAMS(General Algebraic Modeling System)の使用方法についてまとめている。
GAMSは松橋研究室でよく使用されている、最適化計算用のソフトウェア。

インストールの仕方

GAMSのダウンロード・インストール

https://www.gams.com/download/ にアクセスし、ライセンスのバージョン・使用環境（ラボのパソコンならWin-64bit）に適したインストールファイルをダウンロードする。このファイルを実行すれば、自動的にインストールされる。

MNAS/student/資料_document/GAMSライセンスファイル にライセンスを保存してあるので、必要に応じて利用されたい。

基本的には、Largeのライセンスを用いる（20ユーザーまで利用可能）。gurobiソルバーがどうしても必要な場合は、Smallのライセンスを用いる（5ユーザーまで利用可能）。

ライセンスファイルの使用

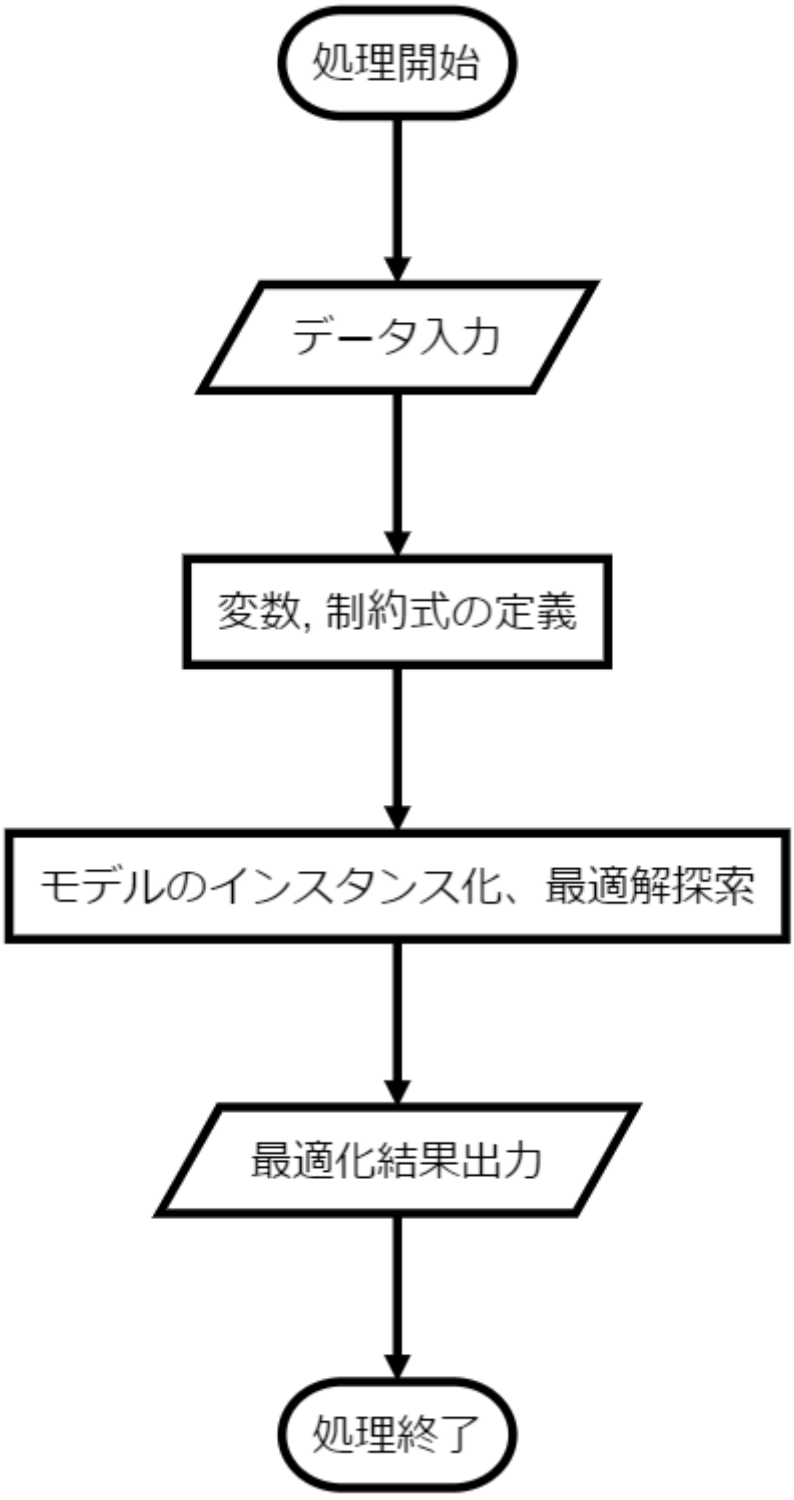
インストールが終わると、ライセンスの選択画面が現れる。その際、保存してあるライセンスファイル（テキストファイル）を選択するか、中身をコピーペーストすれば、ライセンス認証が完了する。

参考動画

- https://www.youtube.com/watch?v=vDnjGA5PImE
- https://www.youtube.com/watch?v=vSe3YGkUVoc

プログラムの書き方

GAMSの基本的なプログラムの流れは以下の通り。



基本的にはこの流れで記述していけばよいので、プログラミング自体はそこまで必要にはならない。

キーワード

上図の上の処理から順に解説していく。

データ入力

- Sets
  添え字のこと。これを用いて各要素にアクセスする。
  例えば1日のシミュレーションをするとき、タイムステップが1時間ならtが1から24に動くようにセットする。(30分なら1~48)
  以下の例では、時間と発電機番号をセットとして定義しているが、例えばこれで、p(i,t)とアクセスすれば発電機iの時刻tにおける発電量を表していることになる。
  <定義の仕方>

  ```
  name    description     /range/
  ```

  <例>

  ```
  Sets
  t               time                        /1*24/
  i               generator_id                /1*27/
  ```

- Scalar
  スカラー値の定義、1個の値しか持たないため、Setsで定義された添え字ごとに違う値を持つなどの場合はParameterで定義。
  (ex. 太陽光発電の導入容量[GW]はシミュレーションを通して一定なのでスカラー値、太陽光発電の発電量は時刻に応じて変化するためParameter)
  下の例では、EVの総数を40台と定義。
  基本的に定数は大文字の名前をつけると変数と区別ができて可読性がいい。
  <定義の仕方>

  ```
  name    description     /value/
  ```

  <例>

  ```
  EV_NUM  Total number of EV /40/
  ```

- Parameter
  データタイプをParameterとするかTableにするかについては、GAMSのドキュメントにも書いてあるが、その値がいくつのセットに対して変化するかで決めればよい。
  基本的に、値が一つのセットに対して変化する場合はParameter、二つ以上のセットに対して変化する場合はTableで定義するのがよい。
  例えば、太陽光発電の発電量は時刻のみに対して変化するのでParameter、各家庭の電力需要みたいなデータは家のidと時刻の両方に対して変化するのでTableで定義するのがよい。

<定義の仕方>

```
parameter[s]
param_name[(index_list)] [text] [/ element [=] numerical_value
                                        {,element [=] numerical_value} /]
{,param_name[(index_list)] [text] [/ element [=] numerical_value
                                        {,element [=] numerical_value} /]} ;
```

<データ入力例>

```
Parameter PV_Data(t)        'PV発電量[MWh]'
/ 1 0
  2 0
  3 0
  :
  12 3500
  13 3200
  :
  24 0
/
;
```

- Table
  <定義の仕方>

```
table table_name[(index_list)]  [text] [EOL
                element               { element }     EOL
element    numerical_value       { numerical_value} EOL
{element    numerical_value       { numerical_value} EOL}] ;
```

<例> JEPXの代表日ごとのプライスは、時刻と代表日のセットに対して変化するので以下のようにTable型で定義するのがいい。

```
Table
JEPX_Price(d)   JEPX Price of each representative day
    SW      SH      MW      MH      WW      WH
1   5.2     6.4     7.2     6.8     6.4     7.0
2   6.3     6.2     7.0     7.4     7.3     6.9
            :
48  5.3     6.5     7.1     7.0     6.6     7.2
```

データ入力に関しては、csvファイルやexcelファイルで入力することもできる。この辺はなかなか煩雑なので、先輩の書き方とか参考に書いていくのがいいと思われる。

変数, 制約式の定義

変数の定義は以下のように行う

```
Variable_Type(s)
name    description
```

ここで、よく使われる変数の種類としては、

- Positive Variables (0以上)
- Binary Variables (バイナリ変数)
- Variables (一般的な変数)

などがある。基本的に変数名は小文字でつけるのがよい。
また、添え字に依存して変化するようなものは例えば以下のように定義する。

```
Positive Variable
p(i,t)                  Assigned Energy
```

制約式の定義は以下のように行う。

```
Equation(s)
name list of equations
;

equation_name..equation;
```

まず、一度制約式名のリストを記述し、;で区切った後、それぞれの制約式を記述していく。(よりよい書き方があれば修正してほしい、これだと制約式を追加、消去する際に、名前リストの該当箇所と実際の制約式の該当箇所の二か所を変更する必要がある。)
添え字により変化する変数を用いている式では、制約式名にも添え字をつけること。
制約式名に関してのTipsとしては、変数の定義をする際に "def_変数名" みたいな名前をつける人が多い。

例として変数として上げ代 (基準出力から定格出力までの差)を定義している制約式を下記に示す。コメントアウトには *を使用し、演算子は例えば以下のようになっている。

演算子	GAMSでの記法
=	=e=
<=	=l=
>=	=g=

<例>

```
Equation
* EDC Capacity definition
def_dp_upward(i,t)
        :
;

* Reserve
def_dp_upward(i,t)..dp_upward(i,t) =l= Max_Min_Limit(i,"Rated_Output") * Working_flag(t,i) - p(i,t);
```

モデルのインスタンス化、最適解探索

モデルを記述し終えたら実際に解く命令を出す。
その際の記述方法としては以下に示すような感じ。

<例>

```
Option MINLP = ANTIGONE;

Model  EDC_LFC  /all/;

Solve  EDC_LFC  minimizing z using MINLP;
```

1行目でオプションとしてソルバーの指定、2行目でモデル化を行い、3行目で実際に解いている。
1行目のソルバーの選択に関しては、代表的なソルバーを、モデル化時の注意点のところで記述したので、ライセンスファイルの中から使用できるものを選んでほしい。

2行目の/all/はこれまで定義した全ての制約式を用いて最適化を行うことを示す。そのため、定義したのに実際に記述していない制約式などがある場合はエラーになる。

3行目は、この例では目的関数のzを最小化するように命令している。
もちろん最大化の場合はmaximizingでいい。

最適化結果出力

最適化問題が解き終わったら、結果をcsvファイルに出力することで効率的に結果を確認できる。
基本的には先輩の出力の仕方などをみて書いていくのがよいと思うが、
出力なかなか思ったようにしてくれないこともあるので、試行錯誤でやっていくしかない。

いちよ解説だけすると、
(変数).Lで変数の値にアクセス、(Set名).tLでそのセットの値にアクセス、Putは書き込むコマンド、/が改行、Loopは繰り返し処理など。
あと途中に書いてある、"Assigned_Energy.PC = 5;"みたいなのをやっておくと、GAMSの出力が横に長くなった時にcsvで****と書かれる現象を防ぐことができる。

<例>

```
File Assigned_Energy /.\output\Kyusyu\assigned_energy.csv/;
Put  Assigned_Energy;
Assigned_Energy.PC=5;
Put "TIME", Loop(i, Put i.tL) Put "NaS" /;
Loop(t, Put t.tL Loop(i, Put p.L(i,t),) Put p_battery.L(t) /;);
```

を実行すると、

	1	2	…	27	NaS
1	700	0	…	250	50
2	700	0	…	240	30
:	:	:	:	:	:
24	700	0	...	250	40

みたいな結果が出力される。

モデル化時の注意点

- **最適化問題の種類について**
  変数とパラメータの選択次第で最適化問題の種類が異なることに注意する。基本的に線形で解きたいなら、以下のことをしてはならない。
  - **(変数)の二乗, log (変数), exp(変数)などの利用 (当たり前)**
  - **(変数) * (変数) を行う**
  - **条件式を用いて制約を書く時の条件に変数を使用する**

  以下これらについてすこし解説。
  - **(変数) * (変数) を行う**

  例えば起動停止計画を考える。火力機の発電量の変数をp、稼働フラグをU (変数) として、火力機の定格運転制約を考えると、

$$p_{i,t} \leq p_i^{max}$$

  これが動いている火力機のみ適用されるので、全ての辺に稼働フラグを掛けて、

$$p_{i,t} \cdot U_{i,t} \leq p_i^{max} \cdot U_{i,t}$$

となる。
式自体は上式で間違いないが、これを実装すると、左辺が、(変数)×(変数)になっているため、混合整数線形計画では解けない。
そこで、以下のように変形してやる必要がある。

$$p_{i,t} \leq p_i^{max} \cdot U_{i,t}$$

こうすれば、非稼働の場合は発電量は強制的に0になるし、稼働時は定格出力以下で制約してやることができる。
今回は火力機で説明したが、蓄電池の出力上限の制約や燃料電池など、このテクニックはどのモデルにおいても適用可能だと思う。

- **条件式を用いて制約を書く時の条件に変数を使用する**

モデルを書いていると、こういう条件だったらこういう制約が欲しいみたいなことがよくある。その際に、if文を使って対応できるのだが、この使い方には注意が必要である。
例えば深夜(0<= t <= 6)の時間帯では燃料電池が停止するみたいな制約を書く場合、

```
def_stop_at_midnight(i,t)$(t.val le 6).. fc_output(i,t) =e= 0;
```

とし、これは線形のまま解くことができる。
しかしながら、例えば同時に蓄電池の運用を最適化してるとして、蓄電池が放電 (p_battery > 0) しているときに燃料電池が停止するみたいな場合、同じ要領でかくと、

```
def_stop_at_midnight(i,t)$(p_battery.val g 0).. fc_output(i,t) =e= 0;
```

なるが (正しい記述方法は違うかもしれない...)、これをやると非線形になる点に気を付ける。
このミス意外としてしまうケースが多いので、注意が必要である。例えば、需要データや太陽光発電の発電量など完全に分かってるとする場合、これらは変数ではなくパラメータとして入力させるべきである。
つまり、最初の例のように、条件式の中で使用されている値がパラメータや添え字 (Sets) の場合、線形性を崩さずモデル化できる。
一方、後半の例のように、条件式の中で変数を使って評価した場合、非線形なモデルとなる。

- **ソルバーの選択に関して**
  基本的によく使われるソルバーを以下に示す
  - **MILP (Mixed Integer Linear Programming)の場合: gurobi, cplex**
  - **MINLP (Mixed Integer Non-Linear Programming)の場合: antigone**

基本的に中身を知らなくても最適化計算は行ってくれるが、個人的な考えでは具体的にどうやって解を探索していってるかのイメージは持っておいてほしい。
混合整数計画においてよく使われるのはBranch-and-BoundやBranch-and-cutと呼ばれる手法で、YouTubeなどで検索すればたくさんヒットするし、最適化ゼミでもやったことがあるのでその資料とかを参考に。

- **データの入力ファイルに関して**
  データの入力方法として.xlsxや.csvを使うことがあると思うが、.xlsxの方がデータの読み取り時間が長かったような気がする。
  一回のシミュレーションにおいては、実行時間のみ気にすればいいと思うが、モンテカルロシミュレーションなどにより繰り返し最適化計算を実行する場合には、入力データを読み取る時間も無視できなくなってくる。
  ただ、csvファイルでデータを入力すると、多くのcsvファイルが必要になるので、User-friendlyではなくなっていくかもしれない。

プログラムの実行と結果の確認

プログラムの実行とデバッグ

プログラム（.gmsファイル）を開いた状態でF9キーを押すか、Fileタブ→Runで実行できる。

プログラムに文法上の誤りがある場合、実行画面および結果ファイル（.lstファイル）にエラーが表示される。誤りがある部分以降はチェックされないので、まずは一番上のエラーから修正していくようにする。

よくある文法ミス

- 文末のセミコロン忘れ
- 変数名などの綴り（自動補完を積極的に活用しましょう）
- 等式・不等式の引数の過不足
  （引数を設定しているのにその後の記述で引数を書き忘れる・引数を設定していないのにその後の記述で引数を書いてしまう）

その他あれば随時書き足してください

結果の確認

プログラムの実行が完了すると、プログラム内で定義したアウトプットがなされるとともに、実行結果ファイル（.lstファイル）が生成・表示される。

その中の "MODEL STATISTICS" に記載されている "SINGLE EQUATIONS" がモデルの方程式・不等式の本数、"SINGLE VARIABLES" がモデルの内生変数の数を表す。