GLPKスーパー簡易マニュアル

by GLPKで楽しく最適化しよう！
http://mukun_mmg.at.infoseek.co.jp/mmg/glpk/

平成17年7月5日: update

1 まえがき

GLPKとはGNU Linear Programming Kitの略で，GNUが無料配布しているLP/MIPソルバーである．ILOG社のAMPL(A Mathematical Programming Language)に準拠した文法のモデルが記述可能で以下のような機能を持つ．

1. モデルとデータの分離
2. 集合演算を駆使した抽象的なモデルの記述
3. モデル・入力データのエラーチェック

ここでは一般的な最適化問題の説明と必要最低限のモデル記述例を示す．

2 最適化問題

最適化問題 (optimization problem) とは一般的に以下のように記述される．任意の空間Xの部分集合Sと，X上で定義された実数値関数$f : X \to \mathcal{R}$が与えられたとき，fを最小化（または最大化）する解$\boldsymbol{x} \in$ Sを求める問題である．
一般に最適化問題は

$$\text{最小化 } f(x) \tag{1}$$
$$\text{条件 } \boldsymbol{x} \in \text{S} \tag{2}$$

fを目的関数(objective function)，Sを実行可能領域(feasible region)，式(2)を制約条件(constraint condition)という．また，任意の$\text{x} \in S$に対して$f(\text{x}^*) \leq$

$f(\mathrm{x})$ となる解 $\mathrm{x} \in S$ を最適解 (optimal solution) という．さらに $f(\mathrm{x}^*)$ を最適値 (optimal value) と呼ぶ.

なお，最大化問題は $-f(x)$ を最小化することと同義であるので，式 (1) の書き方でも一般性は失われない.

3 GLPK の実行

コマンドラインにて以下のコマンドを実行する.

```
glpsol -m モデルファイル名 -d データファイル名
```

モデルファイルやデータファイルについては後述する．なお，-m などはオプション指定子といって，他にも様々なオプションがある．glpsol -h と入力すれば詳細を知ることができるので，ここではよく使うオプションについて説明する.

-m	モデルファイルの入力
-d	データファイルの入力
-o	実行結果の出力
-y	実行過程の出力
--simplex	単体法を用いる (デフォルト指定)
--interior	内点法を用いる
--min	強制的に目的関数を最小化
--max	強制的に目的関数を最大化
--check	解かずにエラーチェックのみ
--nomip	MIP を LP 緩和で解く
--wmps	MPS 形式で出力 (他ソルバーとの互換性に)
--wlpt	CPLEX 形式で出力 (数式チェックなどに)
--wtxt	プレーンテキストで出力 (数式チェックなどに)

入力や出力指定があるオプションは，オプション指定子のすぐ後ろに空白を空けて記述する．例えば，glpsol -m model1.mod -d data1.dat -o result1.sol --wlpt cplexform.txt --min と指定すれば，モデル・データを読込み，結果を result1.sol，CPLEX 形式のモデルを cplexform.txt に書き込み，最小化する，という意味になる.

4 モデルファイルとデータファイル

GLPK では問題の定義と，データの定義を分離して行うことができる．このうち問題の定義を行う役割があるのが**モデルファイル**であり，拡張子は通常「.mod」

で，データの値を与えるを行う役割があるのが**データファイル**であり，拡張子は通常「.dat」である．定義を分離することにより，モデルファイルをあたかも問題を解くための独立したプログラムのように扱うことができ，後述のデータファイルをとりかえるだけで様々な問題を解くことができるのでモデルとデータは分けて記述するべきである．

5　集合

集合 (set) とはモデル中で取扱い対象となるものの集まりである．GLPK では set のキーワードを用いる．

5.1　モデルファイルの記述

形式

set　集合名 { 属する集合名，… };

例) 点の集合名を Node とする

```
set   Node;
```

5.2　データファイルの記述

要素の記述は「:=」の後に空白区切りで行う．
例)Node の要素を「1,2,3,4」とする

```
set Node := 1 2 3 4;
```

6　パラメータ

パラメータ (parameter) とは 0 個以上の集合の要素に依存して変わる定数である．GLPK では param のキーワードを用いる．

6.1　モデルファイルの記述

形式

param パラメータ名 { 属する集合名，… };

例) 点ごとに定義された費用を $Cost_i \quad i \in Node$ とする

```
param Cost{Node};
```

6.2 データファイルの記述

集合によって決まるパラメータの記述の際には記述 1 のように集合名の後に空白区切りで値を入れるか，記述 2 のように配列形式で値を入れることができる．例を以下に示す．

例) それぞれの費用を以下の通りとする．

点	費用
1	5
2	3
3	6
4	9

記述 1)

```
param Cost :=
            1   5
            2   3
            3   6
            4   9;
```

記述 2)

```
param Cost[1]  :=  5;
param Cost[2]  :=  3;
param Cost[3]  :=  6;
param Cost[4]  :=  9;
```

例) 二つ組の場合の行列形式 (データは以下の通り)

点	品目	価値
1	p	10
1	q	20
2	p	15
2	q	23
3	p	12
3	q	18

二つ組のパラメータの記述では、縦横の行列形式で記述することができる。縦が 1 番目の要素、横が 2 番目の要素である。座標のイメージで、縦横の交点が指定する添字となる。

ただし、パラメータ名の後ろに (tr) をつければ、縦横の役割が逆になる。tr は **転置行列 (transpose matrix)** の略である。記述 1) は標準形、記述 2) は tr 形を示す。

記述 1)

```
param Val: p q :=
          1 10 20
          2 15 23
          3 12 18
```

記述 2)

```
param Val(tr): 1 2 3:=
            p 10 15 12
            q 20 23 18
```

例) 二つ組以上の場合 (データは以下の通り)

発地	着地	資源	固定費用
1	2	1	10
1	2	2	15
1	3	1	13
1	3	2	17

記述 1)

```
param ArcResourceFC :=
                1 2 1 10
                1 2 2 15
                1 3 1 13
                1 3 2 17   ;
```

記述 2)

```
param ArcResourceFC[1,2,1]  :=  10;
param ArcResourceFC[1,2,2]  :=  15;
param ArcResourceFC[1,3,1]  :=  13;
param ArcResourceFC[1,3,2]  :=  17;
```

例) 共通する組に対してデータを代入する場合

同じ集合の組に対応するデータを代入する場合は同時に記述することができる. 以下のように，param の後ろを:で区切り，パラメータ名を続ければよい.
以下は枝・資源・品目によって決まるリードタイムとサイクルタイムを設定する例である.

発地	着地	資源	品目	リードタイム	サイクルタイム
1	2	1	1	1	3
1	2	2	1	2	6
1	3	1	1	1	4
1	3	2	1	2	8

記述)

```
param: LT CT:=
   1 2 1 1 1 3
   1 2 2 1 2 6
```

```
1 3 1 1 1 4
1 3 2 1 2 8   ;
```

7 変数

変数 (variable) とは 0 個以上の集合の要素に依存して変わる最適化対象となる値である．GLPK では var のキーワードを用いる．形式

var 変数名 { 属する集合名, ··· }

例)2 点間のフロー量を $\mathrm{Flow}_{i,j}\quad i \in Node,\ j \in Node$ を GLPK で表現すれば

```
var Flow{Node, Node};
```

7.1 変数のレンジ

変数名の後ろにレンジ指定をつけることにより，実数や整数，0-1 変数などの規定を行うことができる．2 つ以上つけるにはカンマで区切って指定すればよい．

指定なし	実数
integer	整数
binary	0-1 変数
等号/不等号	範囲指定

例) 変数 $x \in \mathcal{Z}^+$ を宣言する

```
set x integer, >=0;
```

8 制約条件

制約条件 (constraint condition) とは，全ての変数が満たすべき関係式である．GLPK では s.t. または subject to のキーワードを用いる．形式

s.t. 制約条件名 { 属する集合名, ··· } 式

例)

$$\sum_{r \in Res} x_{ir} \leq M \qquad \forall i \in Node$$

```
s.t. COND1{i in Node}:  sum{r in Res}x[i,r] <= M;
```

となる．$\forall$ は制約式名の後に付き，$\in$ は in，$\sum$ は sum キーワードになっているように，数式と 1 対 1 に対応する．

9 目的関数

目的関数 (objective function) とは，制約条件を満たしながら変数を変えることによって最大化 (maximize) または最小化 (minimize) することを目的とする関数である．
GLPK では maximize または minimize のキーワードを用いる．形式

minimize 目的関数名: 式

の形式で表現できる．
例) 目的関数 :

$$\min . \sum_{i \in Node} Cost_i x_i$$

```
minimize OBJ: sum{i in Node} Cost[i] * x[i];
```

10 その他

詳細な GLPK の仕様については，GNU 本家のリファレンス・マニュアルを参照されたい．ここではモデルの記述に役立つものについてのみピックアップして紹介する．

10.1 算術演算子

+	加算
-	減算
*	乗算
/	除算
less	$A > B$ ならば A-B, $A < B$ ならば 0
div	A/B の商
mod	A/B の剰余
** または ^	A の B 乗

10.2 論理/関係演算子

$A < B$	$A < B$
$A <= B$	$A \leq B$
$A > B$	$A > B$
$A >= B$	$A \geq B$
$A <> B$ または $A! = B$	$A \neq B$
A in B	$A \in B$
A not in B または A ! in B	$A \notin B$
A within B	$A \subseteq B$

10.2.1 条件により式を変える

形式

`if(条件式) then` 条件が真の時の式 `else` 条件が偽の時の式

例) 添字 $t \neq 1$ のとき X_{t-1}，$t = 1$ のとき `0`

```
if(t!=1)  X[t-1]  else 0
```

10.2.2 集合の切り出し条件

形式

: 切り出し条件

例)$i > j$ の場合のみ和を求める

$$\sum_{(i,j)\in Arc} w_{ij} \qquad (i \ > \ j)$$

```
sum{(i,j) in Arc :  i > j} w[i,j]
```

10.2.3 within によるエラーチェック

モデルファイルで `within` 演算子を用いて集合の範囲を規定しておけば想定外の要素の代入を防ぐことができ，エラーを予防することができる．

例)

```
set A;
set K within A;
```

としておけば集合 K は集合 A の部分集合なので，以下のデータはエラーとなる．

```
set A := 1 2 3 4;
set K := 5;
```

10.3 集合演算子

A union B	和集合	$A \cup B$
A diff B	差集合	A/B
A simdiff B	対称差集合	$A \oplus B$
A inter B	共通集合	$A \cap B$
A cross B	直積集合	$A \times B$
a .. b	順序集合	[a,b] に含まれる整数の順序集合を作成

10.3.1 setof 演算子

集合の切り出し演算子である．データファイルの記述の際に人間の入力ミスを防ぐためにも，同じような集合は入力しない方が無難である．
形式

`setof{(添字, ··· ) in 集合名 } (切り出し添字, ··· )`

例) 集合 ARP(枝 (i,j), 資源 (r), 品目 (p) の 3 つ組) から枝 (i,j)，品目 (p) の 2 つ組ペア AP を取り出す

```
set AP := setof{(i,j,r,p) in ARP}(i,j,p);
```

10.4 デフォルト値

パラメータデータを入力しなかった場合，自動的に使われる値である．

例) モデルファイルの宣言

```
param  p  default  9999;
```

こうしておけば，データファイルで入力しなかった場合，自動的にこのパラメータ p は 9999 となる．

10.5 display 文の利用

```
display 変数・パラメータ・制約式名;
```

display 文をモデル中に記述しておくと，指定した変数などの値を表示することができる．必要な変数の値だけを見たい場合，自動生成したパラメータの値が正しいかなどいろいろな目的で便利に使用することができる．

11 AMPLの利用

AMPLはGLPKの本家であるので，より多機能・高機能である．フリー版は300変数までに制限されているので実用的な利用にはつらいが，式の展開などの機能は制限無く使用できる．これを知っているとバグ発見に役立つことがあるので説明しておく．

11.1 AMPLの起動

AMPLはGLPKと違い，AMPLのシェル上でコマンドを入力していくタイプのインタフェースである．起動するにはコマンドライン上で以下のコマンドを実行する．

```
ampl
```

すると，プロンプトが「ampl:」に変わったと思う．これがAMPLが動作している状態である．この状態では以下のコマンドが実行できるので，よく使う物を抜粋して紹介する．

model	モデルファイルの入力
data	データファイルの入力
solve	実行する
option solver	ソルバーの選択
expand	数式の展開
show	変数・パラメータ・制約式名を表示
display	変数・パラメータ・制約式・計算結果などの値を表示
reset	入力したモデル・データを破棄
quit	AMPLシェルを終了
exit	AMPLシェルを終了
let	変数・パラメータを固定
commands	コマンドスクリプトを実行

例えば，モデルファイルm1.modを読み込む時にはmodel m1.mod;と入力する．最後のセミコロンを忘れないこと！これは，AMPLシェル上でコマンドスクリプトというプログラムが実行できるためである．

コマンドスクリプトはファイルにまとめておけば，commandsコマンドで実行できる．例えば，option solver cplex; model m1.mod; data d1.dat; solve;という一連のコマンドをcom1.cmsに保存したとしよう．すると，毎回入力しなくてもcommands com1.cms;で実行できる．また，内部でループや値の固定ができるので，ケース分析にも使える．

expandコマンドは非常に便利で，モデルを展開した数式を表示できる．こうすれば，思わぬ記述ミスが発見できることが多い．

一度モデルを読込み，再度別のモデルを読み込む時は必ず，reset; すること．もしくは quit; でいったん終了するのも安全である．

デフォルトのソルバーは MINOS(マイノス) というソルバーで，これは MIP を解くことができない．MIP を扱うには，option solver cplex; として，CPLEX(シープレックス) というソルバーを読み込む必要がある．

AMPL も是非有効活用してほしい．

12　よくあるエラー

12.1　変数の範囲エラー

```
unbounded (or badly scaled) problem.
```

原因

- 実行可能領域が非有界になっている。

対策 1

- 制約が抜けていませんか？
- 特に、変数 ≥ 0 とする、非負条件を忘れていませんか？

12.2　添字数エラー

```
(変数名 or パラメータ名) must have * subscripts rather than #
```

※ただし，*,#には数字が入る

原因

- 本当はその変数の添字数は*だが，#個指定してしまっている．

対策

- 正しい個数に直してください
- 1 つの集合が 1 つの添字とは限らないから注意！
- 例) `set Node; set Arc{Node, Node}; var X{Arc};` としていたら，
  X は Node2 つ分の添字を持ちます．

12.3 添字範囲エラー

(変数名 or パラメータ名)[添字] out of domain

原因

- 添字がおかしい (定義外の集合要素になっている)

対策 1

- 添字の順番などをもう一度確認してください
- データを行列形式で入力した際，行と列を逆にしていませんか？
- 整数で {1..N} の様な範囲形式で指定した際，定義域をはみ出していませんか?

対策 2

- 不可解なときはだいたいこれです.
- 上記の範囲形式で集合やパラメータの添字を定義する際，定義には順番があります.
- GLPK では，データファイルに依存した順序集合の生成は許されません.
  内部では範囲が分かっているはずですが，モデルファイルだけから決定できないからでしょう.
  よく分かりませんが，とにかくこういう仕様になっています.

12.3.1 良い例

モデルファイル

```
param T; # これは範囲を規定するパラメータ
set Period := 1..T; # 必ずモデルファイルで範囲を指定します
```

データファイル

```
param T:=10; # 実際に数値を代入します．これで集合の個数を規定
```

12.3.2 悪い例

モデルファイル

```
param T; # これは範囲を規定するパラメータ
set Period; # 範囲が書かれていません.
```

データファイル

```
param T:=10; # 実際に数値を代入します.
set Period := 1..T; # 範囲を規定しているつもり. でもこれじゃダメ
```

結論から言って，範囲はモデルファイルで書いてください，ということである.

12.4 演算子のつかい間違い

```
syntax error in set statementset 集合名 =
```

原因

- 代入演算子は「:=」です. 「=」ではない.

対策

- :=に直してください.
- VB などと違って，代入と等号には区別があります.
- ついやってしまうので気をつけてください.
- param でも同様の現象があります.

12.5 比較型エラー

```
operand preceding = has invalid type
```

原因

- if で変数を比較するなどしている

対策

- 「変数 $x > 0$ だったら $y = 1$」みたいな論理条件は if では記述できません.
- 整数変数を使って定式化してください.
- if もパラメータを比較して添字範囲オーバーを防いだり，式の一般化などの目的で用いるものです.

英語だが，http://www.ampl.com/FAQ/にも様々なケースが紹介されているので参考にしてほしい.