AMPL：数理計画のためのモデリング言語

>_

第13章　コマンドスクリプト

include・*for*・*repeat*・*if* で解析作業を自動化する

`include / commands` `for` `repeat` `if-then-else` `break / continue` 文字列関数

本章の内容

AGENDA

13.1	スクリプトの実行：include と commands	ファイルからコマンド列を読み込む2つの方法
13.2	集合上の反復：for 文	集合の各要素について文を繰り返す
13.3	条件付きの反復：repeat 文	while / until で終了条件を指定する
13.4	条件の検査：if-then-else 文	条件に応じて実行する文を切り替える
13.5	ループの終了：break と continue	回をスキップ／ループを打ち切る
13.6	スクリプトを一歩ずつ実行する	single_step・step・next・skip でデバッグ
13.7	文字列の操作	連結演算子＆と文字列関数、文字列式の活用

なぜコマンドスクリプトか

- モデル開発の初期段階では対話的にコマンドを打ち込むが、定式化が落ち着くと
- **同じコマンド列を何度も繰り返し入力することになる**
- **スクリプト（script）＝ファイルに収めた、繰り返し使うためのコマンド列**
- for、repeat、if など、プログラミング言語のような制御構文を含められる
- → AMPL のコマンド言語で「小さなプログラム」が書ける
- 単発実行 include／commands から、繰り返し for／repeat、条件分岐 if、
- 文字列処理まで、この章で段階的に扱う

1. **読む**
   `model / data / include`
2. **繰り返す**
   `for / repeat`
3. **分岐する**
   `if-then-else`
4. **制御する**
   `break / continue`
5. **調べる**
   `single_step / step`

IN スクリプトの実行：include と commands

`include filename`

- 指定ファイルの内容で置き換えられる
- 他の文の途中にも現れてよい
- 終端のセミコロンは不要
- モードを変えない（model/dataの一般形）

`commands filename ;`

- include に似ているが、独立した「文」である
- 終端のセミコロンが必要
- 文が合法な文脈にのみ現れうる
- **for / repeat / if 内では commands が最善**

- model・data コマンドは、読み込み前にモードをモデル／データモードにする include の特別な場合
- 小さなモデルでは、宣言・data コマンド・データ文を1ファイルにまとめ model や include で一度に読み込める
- 起動時に option の既定値を変える include ファイルを自動で読み込ませることもできる（付録 A-14, A-23）

感度分析スクリプトを commands で実行

steelT.sa1 （繰り返すコマンドを1ファイルに）

```
solve;
display Total_Profit >steelT.sens;
option display_1col 0;
option omit_zero_rows 0;
display Make >steelT.sens;
display Sell >steelT.sens;
option display_1col 20;
option omit_zero_rows 1;
display Inv >steelT.sens;
let avail[3] := avail[3] + 5;
```

対話セッション：呼ぶたびに avail[3] が5増える

```
ampl: model steelT.mod;
ampl: data steelT.dat;
ampl: commands steelT.sa1;
MINOS 5.5: optimal solution found.
15 iterations, objective 515033
ampl: commands steelT.sa1;
MINOS 5.5: optimal solution found.
1 iterations, objective 532033
ampl: commands steelT.sa1;
MINOS 5.5: optimal solution found.
1 iterations, objective 549033
```

- commands steelT.sa1 の1行を打ち込むだけで、ファイル中の全コマンドが実行される
- display の出力はすべてファイル steelT.sens にリダイレクトされている（画面表示も可能）
- 同じ内容でも include で代用できるが、制御構文と組み合わせるスクリプトでは commands が安全

集合上の反復：for 文


```
for {添字式} コマンド;
for {添字式} { コマンド1; コマンド2; ... }
```

- ある集合の各要素について一度ずつ、文（または波括弧でまとめた複数の文）を実行する
- for と本体の間の式は任意の AMPL 添字式でよい
- コマンドを別ファイルに置いて commands で呼ぶ代わりに、波括弧内に直接書ける
- 集合が順序づけられていれば（1..T や ordered/circular 宣言）その順序で反復する
- 順序づけられていない集合では AMPL が順序を選ぶが、毎回同じ順序になる

1回の for で4回の感度分析を実行

```
ampl: model steelT.mod;
ampl: data steelT.dat;
ampl: for {1..4} commands
         steelT.sa1;
MINOS 5.5: optimal solution found.
15 iterations, objective 515033
MINOS 5.5: optimal solution found.
1 iterations, objective 532033
MINOS 5.5: optimal solution found.
1 iterations, objective 549033
MINOS 5.5: optimal solution found.
2 iterations, objective 565193
```

FOR パラメータ感度分析のスクリプト

図13-1

steelT.sa3

```
set AVAIL3;
param avail3_obj {AVAIL3};
param avail3_dual {AVAIL3};

let AVAIL3 := avail[3] .. avail[3]+15 by 5;

for {a in AVAIL3} {
   let avail[3] := a;
   solve;
   let avail3_obj[a] := Total_Profit;
   let avail3_dual[a] := Time[3].dual;
}

display avail3_obj, avail3_dual;
```

```
ampl: commands steelT.sa3;
: avail3_obj avail3_dual :=
32   515033      3400
37   532033      3400
42   549033      3400
47   565193      2980
;
```

- AVAIL3 は試したい avail[3] の値の集合
- for {a in AVAIL3} が各値 a について解き直す
- 添字 a はループ内の文でそのまま使える
- option solver_msg 0 でソルバーの出力を抑制

FOR 入れ子ループで整形表を作る

図13-2

steelT.tab1 （外側 1..T、内側 PROD の二重ループ）

```
printf "\nSALES";
printf {p in PROD}: "%14s   ", p;
printf "\n";

for {t in 1..T} {
   printf "week %d", t;
   for {p in PROD} {
      printf "%9d", Sell[p,t];
      printf "%7.1f%%", 100*Sell[p,t]/market[p,t];
   }
   printf "\n";
}
```

```
ampl: commands steelT.tab1;
SALES         bands     coils
week 1     6000 100.0%   307   7.7%
week 2     6000 100.0%  2500 100.0%
week 3     1399  35.0%  3500 100.0%
```

- 外側ループ {1..T} が表の1行を生成
- 内側ループ {p in PROD} がその行の各製品の項目を生成
- printf は改行 \n が現れるまで行を変えない
- 製品数や名前がデータ依存でも対応できる（1つの display文より汎用的）

REP 条件付きの反復：repeat 文

- 論理条件が満たされるかぎり反復を続ける
- while：条件が真であるかぎり継続
- until：条件が偽であるかぎり継続（真になったら終了）
- 条件なしなら無限ループ（break 等で終了）

`repeat while COND { ... }`	本体の前に検査 → 最初から偽なら0回
`repeat until COND { ... }`	本体の前に検査 → 最初から真なら0回
`repeat { ... } while COND`	本体の後に検査 → 少なくとも1回実行
`repeat { ... } until COND`	本体の後に検査 → 少なくとも1回実行

REP 感度を記録する repeat スクリプト

図13-3

steelT.sa4

```
set AVAIL3 default {};
param avail3_obj {AVAIL3};
param avail3_dual {AVAIL3};
param avail3_step := 5;

repeat {
   solve;
   let AVAIL3 := AVAIL3 union {avail[3]};
   let avail3_obj[avail[3]] := Total_Profit;
   let avail3_dual[avail[3]] := Time[3].dual;
   let avail[3] := avail[3] + avail3_step;
} until Time[3].dual = 0;

display avail3_obj, avail3_dual;
```

- 反復回数が事前に分からないので AVAIL3 を default {} で空集合に
- **= {} ではなく default {} が重要：**
- **let で以後も変更可能にするため**
- 解いた後で新しい avail[3] を集合に追加していく
- 浮動小数点誤差対策として option solution_precision 10 を設定
- 本体の後の until … なので少なくとも1回は solve される

条件の検査：if-then-else 文


```
if Make["coils",2] < 1500 then
   printf "under 1500\n";

if Make["coils",2] < 1500 then {
   printf "Fewer than 1500 coils.\n";
   let market["coils",2] :=
      market["coils",2] * 1.1;
}
else
   printf "At least 1500 coils.\n";

// else if の連鎖：
if Sell[p,t] = 0 then printf "    ";
else if Sell[p,t] < market[p,t] then
   printf "%7.1f%%", 100*Sell[p,t]/market[p,t];
else printf "  --- ";
```

- if 条件式 then 動作; が最も単純な形
- 動作は単一コマンドでも { … } でまとめた複数コマンドでもよい
- 任意の else が代替の動作を指定する（単一コマンドまたはブロック）
- 式が真なら then 側、偽なら else 側（あれば）を実行
- then／else の中身がさらに if 文でもよく、else if を連ねて多分岐にできる
- else は直前の対応可能な if と対にされる

BRK ループの終了：break と continue

continue

現在の回を終了し、次の回の検査から実行を続ける

break

for／repeat ループを完全に終了し、直後の文へ制御を移す

名前付きループ

repeat name { … } と break name; で外側ループを直接指定

steelT.sa7（図13-5）：変化があった時だけ記録

```
repeat {
   let avail[3] := avail[3] + 1;
   solve;
   if Time[3].dual = previous_dual
      then continue;
   let AVAIL3 := AVAIL3 union {avail[3]};
   ...
   if Time[3].dual = 0 then break;
   let previous_dual := Time[3].dual;
}
```

- break・continue は最も内側のループにのみ適用される
- 外側のループを直接抜けたい場合はループに名前を付ける：
- repeat sens_loop { … }
- for {t in 1..T}
- if Time[t].dual<2700 then
- break sens_loop;
- ループ名は repeat／for の直後、break／continue の直後に置く

スクリプトを一歩ずつ実行する

- option single_step 1; で1コマンドずつ止まる「シンボリックデバッガ」に
- **step / step n**
- 次の1個／n個のコマンドを実行
- **next / next n**
- 複合コマンド（for/repeat/if）の中に入らず、その先へ進む
- **skip / skip n**
- 次の1個／n個を実行せずに飛ばす
- 止まった位置は ファイル名:行番号(文字番号) で示される
- 一歩ずつ実行中も任意の AMPL コマンドを打ち込める（display 等で状態確認）
- <n>ampl: プロンプトはこのモードにいることを示す

対話セッション：<2>ampl は入れ子水準を示す

```
ampl: option single_step 1;
ampl: commands steelT.sa7;
steelT.sa7:2(18)   data ...
<2>ampl: step
steelT.sa7:4(36)   option ...
<2>ampl: step
steelT.sa7:5(66)   option ...
<2>ampl: display avail[3],
   Time[3].dual, previous_dual;
avail[3] = 22
Time[3].dual = 3620
previous_dual = 3620
<2>ampl: step
steelT.sa7:17(317)   continue ...
```

& 文字列の操作：関数と演算子

&	連結演算子。2つの文字列（数値は自動変換）をつなげる	`"WEEK" & t → "WEEK3"`
length(s)	文字列 s の文字数を返す	`length("FISH") = 4`
match(s,t)	s の中で t が最初に現れる位置。なければ0	`match("CHK","H") = 2`
substr(s,p,n)	位置 p から始まる長さ n の部分文字列	`substr("A_BEEF",1,1) = "A"`
sub / gsub	正規表現 t の最初／すべての出現を置換	第2引数は正規表現として解釈
sprintf(...)	printf と同じ書式変換で文字列を返す（出力しない）	`sprintf("WEEK%3.1f",0.3)`

& 文字列式でファイル名・オプションを組み立てる

13.7

- model, data, commands のファイル名、リダイレクト先（>, >>）、
- option の値、table の string-list などに文字列式が使える
- いずれの場合も文字列式は括弧で囲む

- 添字 j や i, j はループ中で文字列に自動変換される
- ループを通るたびに異なるファイル・異なるオプション値を設定できる

diet1.dat〜diet3.dat を順に解き diet1.out〜.out へ保存

```
model diet.mod;
set CASES = 1 .. 3;

for {j in CASES} {
   reset data;
   data ("diet" & j & ".dat");
   solve;
   display Buy >("diet" & j & ".out");
}
```

組み合わせを変えて4通りの CPLEX 指示で解く

```
model sched.mod;  data sched.dat;
option solver cplex;

set DIR1 = {"primal","dual"};
set DIR2 = {"primalopt","dualopt"};

for {i in DIR1, j in DIR2} {
   option cplex_options (i & " " & j);
   solve;
}
```

まとめ

1. **スクリプトはファイルに収めたコマンド列**

   include はどこにでも現れる置換、commands は独立した文。制御構文と併用するなら commands が安全

2. **for 文で集合上を反復**

   任意の添字式・任意の集合上で反復でき、入れ子にして printf と組み合わせれば整形表も作れる

3. **repeat 文で条件付き反復**

   while／until を本体の前後どちらに置くかで、最低実行回数（0回 or 1回）が決まる

4. **if-then-else と break／continue でループ制御**

   多分岐や条件付きスキップ・終了、名前付きループで外側ループも直接制御できる

5. **文字列演算子 & と関数、single_step デバッグ**

   文字列式でファイル名・オプション値を動的に組み立て、step/next/skip で挙動を検証できる