Modelowanie w pakiecie AMPL

Pakiet AMPL

Pakiet AMPL jest narzędziem do rozwiązywania liniowych, nieliniowych i całkowitoliczbowych zadań programowania matematycznego. W jego skład wchodzą: algebraiczny język modelowania, różnorodne solvery służące do rozwiązywania modeli programowania matematycznego oraz okienkowy interfejs użytkownika. Pakiet umożliwia korzystanie z danych zawartych w zewnętrznych plikach tekstowych (ASCII).

Przygotowanie programu AMPL do pracy

W oknie poleceń należy wpisać polecenie '`ampl`' i nacisnąć `Enter`. Uruchomienie systemu AMPL potwierdzane jest zmianą symbolu w linii poleceń na '`ampl:`'. Od tej chwili polecenia są interpretowane przez program AMPL. Polecenia dla AMPL należy zawsze kończyć średnikiem.

Pracę z AMPL kończy się poleceniem '`quit`' lub '`end`' (ze średnikiem); następuje wówczas powrót do okna tekstowego, z którego AMPL został wywołany. Pracę w oknie tekstowym kończy się przez zamknięcie okna.

Podstawowe komendy programu AMPL

Do podstawowych komend należą:

`data` – przejście do trybu pracy *`data;`* wstawienie pliku z danymi,
`display` – wypisanie wartości funkcji celu, zmiennych, ograniczeń modelu,
`include` – wstawienie pliku,
`let` – zmiana wartości danych,
`model` – przejście do trybu pracy *`model`*; wstawienie pliku z modelem,
`objective` – wybranie zmiennych do funkcji celu,
`option` – ustawienie lub wypisanie opcji,
`quit` – wyjście z AMPL,
`reset` – kasowanie modelu/danych (umożliwia wprowadzenie następnego),
`solve` – rozwiązanie zadania,
`write` – zapisanie problemu do plików na dysk.

Język AMPL

Język AMPL (A Mathematical Programming Language) jest algebraicznym językiem modelowania problemów programowania liniowego, nieliniowego lub całkowitoliczbowego.

Definiowanie modelu zadania

Podczas definiowania modelu zadania należy określić nazwy parametrów, nazwy i typ zmiennych, funkcję celu, ograniczenia w postaci wyrażeń, zbiory indeksów (opcjonalnie).

Ogólne zasady konstruowania modelu są następujące:

- każde wyrażenie musi być zakończone średnikiem: `;',
- komentarze muszą zaczynać się od znaku: #,
- wszystkie zmienne są domyślnie traktowane jako ciągłe,
- do konstrukcji wyrażeń są używane operatory: *, /, -, +
- zmienne całkowite są dodatkowo określane w deklaracji jako: `integer`,
- zmienne binarne są dodatkowo określane w deklaracji jako: `binary`,

- wyrażenie :

$\forall i \in N \qquad \sum_{j \in M} x_{ij} = y_i$ zapisuje się następująco:

{ i in N}: sum{j in M.} x[i, j] = y[i];

- równoważne są dwa sposoby indeksowania danych:

{*i* in *A*,*j* in *B*} - wszystkie pary (*i*,*j*) *i* z *A*, *j* z *B*, przy czym *A*, *B* muszą byćwcześniej zadeklarowane jako zbiory,

{*i* in *1*..*N*,*j* in *1*..*M*} - wszystkie pary (*i*,*j*) *i* od *1* do *N*, *j* od *1* do *M*, przy czym *N*, *M* muszą być wcześniej zadeklarowane jako parametry,

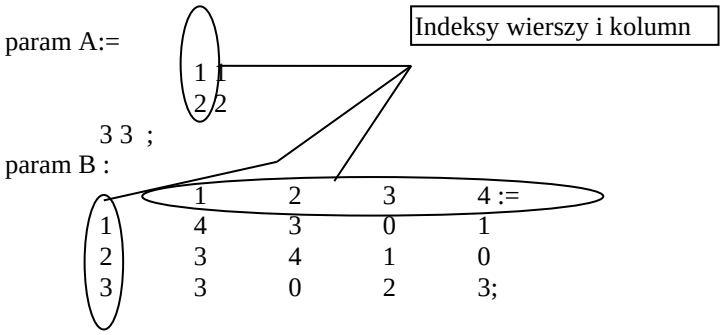
- zbiory i parametry są deklarowane odpowiednio poleceniami: `set` oraz `param`,

- zmienne są deklarowane poleceniem: `var`,

- funkcja celu jest deklarowana poleceniem: `minimize` lub `maximize`,

- ograniczenia są deklarowane poleceniem: `subject to`.

- sposoby definiowania zmiennych tablicowych :




```
param A:=
        1 1
        2 2
        3 3 ;
param B :
          1       2       3       4 :=
1         4       3       0       1
2         3       4       1       0
3         3       0       2       3;
param C default 0:=
        [0,*]   1 4     2 7     3 6     4 10    5 13
        [1,*]   3 3     4 7     5 10
        [2,*]   3 1     4 5     5 8;
```

- przykład definicji zbioru indeksów:

```
set N :=
        (0,*)   1       2       3       4
        (1,*)                   3       4;
```

Przykładowy sposób rozwiązania problemu

Procesor języka AMPL rozróżnia pliki ze względu na rozszerzenia.

Dla użytkownika najważniejsze są dwa z nich:

- plik **.mod* zawiera zapis modelu - deklaracje zmiennych, parametrów, funkcję celu oraz ograniczenia modelu;
- plik **.dat* zawiera wartości danych modelu;

wszystkie będące plikami tekstowymi.

Należy zatem zapisać model w pliku tekstowym *nazwa1*.mod, a parametry w pliku *nazwa2*.dat, uruchomić program AMPL i wczytać pliki z modelem i danymi poleceniami:

```
model nazwa1.mod;
data nazwa2.dat;
```

Jeżeli nie pojawią się komunikaty błędów, można przystąpić do rozwiązania modelu. Polecenie:

```
solve;
```

spowoduje uruchomienie standardowego solvera MINOS.

```
MINOS 5.5: ignoring integrality of 2 variables
MINOS 5.5: optimal solution found.
3 iterations, objective 20885
```

Solver ten nie obsługuje zmiennych całkowitoliczbowych. Gdyby zatem rozwiązanie nie było całkowitoliczbowe, należałoby zmienić solwer na CPLEX poleceniem:

```
option solver cplex;
```

i ponownie rozwiązać problem poleceniem `solve`.

Do obejrzenia wyników służy polecenie `display`. Należy w nim wymienić nazwy zmiennych, funkcji, parametrów, np. `x, f_celu`.

```
display f_celu, x, c;
```

Wyniki można zapisać do pliku dodając przed średnikiem symbol '`>`' i nazwę pliku, np.:

```
display f_celu, x, c > nazwa3.out;
```

Przykład 1

Zadanie programowania liniowego	# zapis w języku AMPL plik *.mod

min $x_1+2x_2+3x_3$
przy ograniczeniach:
$-x_1+x_2+x_3 = 5$
$12x_1-9x_2+9x_3 >= 8$
$x_1,x_2,x_3 >= 0$
	var x1 >=0;
	var x2 >=0;
	var x3 >=0;
	minimize funkcja_celu:
	x1+2*x2+3*x3;
	# p.o.
	subject to ograniczenie1:
	-x1+x2+x3 = 5;
	subject to ograniczenie2:
	12*x1-9*x2+9*x3 >=8;

Ten sam przykład z zastosowaniem stałych modelu jako tablic jednowymiarowych.

# Plik *.mod	# Plik *.dat

var x{1..3} >=0;
param A{1..3};
param B{1..3};
param C{1..3};
 	
minimize funkcja_celu:
sum{i in 1..3} A[i]*x[i];
# p.o. 	
subject to ograniczenie1:
sum{i in 1..3} B[i]*x[i] = 5;
subject to ograniczenie2:
sum{i in 1..3} C[i]*x[i] >=8;
	param A:=
	1 1
	2 2
	3 3 ;
	param B:=
	1 -1
	2 1
	3 1;
	param C:=
	1 12
	2 -9
	3 9;

Przykład 2

Opis zadania:

N studentów musi przed sesją przeczytać M. Książek. Każdy i-ty student potrzebuje na przeczytanie j-tej książki t_{ij} czasu. Książki mogą być czytane w dowolnej kolejności, przy czym w danej chwili tylko jedna osoba może czytać daną książkę i nie może czytać więcej niż jednej książki jednocześnie.

Przykładowe rozwiązanie:

```
# plik *.mod
set CZASY;
set KSIAZKI;
set STUDENCI;
param MAC_TIJ{i in STUDENCI, j in KSIAZKI};  # czasy czytania książek przez studentów
var v{b in CZASY, i in STUDENCI, j in KSIAZKI} >=0, <=1;    # harmonogram czytania
var T_min;

minimize Czas_min:
        T_min;

subject to Sesja{i in STUDENCI}:
        sum{b in CZASY, j in KSIAZKI} v[b,i,j] <= T_min;
subject to JedenStudent{b in CZASY,j in KSIAZKI}:
        sum{i in STUDENCI} v[b,i,j] <= 1;
subject to JednaKsiazka{b in CZASY, i in STUDENCI}:
        sum{ j in KSIAZKI} v[b,i,j] <= 1;
subject to CalaKsiazka{ i in STUDENCI, j in KSIAZKI}:
        sum{b in CZASY} v[b,i,j] = MAC_TIJ[i,j];
```

```
#plik *.dat
set CZASY:= 1 2 3 4 5 6 7 8 9 10 11 12;
set KSIAZKI:= 1 2 3 4;
set STUDENCI:=  1 2 3;
param MAC_TIJ :
                        1       2       3       4 :=
        1               4       3       0       1
        2               3       4       1       0
        3               3       0       2       3;
```