Linköpings universitet
Matematiska institutionen
Optimeringslära

Optimeringslära grundkurs för Y
Introduktion till AMPL
31 januari 2021

Introduktion till modelleringsspråket AMPL

AMPL är ett modelleringsspråk där man enkelt kan beskriva optimeringsproblem med hjälp av mängder, summor, m.m. För att lösa optimeringsproblemet måste man sedan anropa en lösare, det vill säga en optimeringsprogramvara. Vilken lösare som ska användas beror på vilket sorts optimeringsproblem som ska lösas (t.ex. LP-problem eller heltalsproblem). Vi kommer att använda CPLEX, vilket är en programvara som löser linjära och (blandade) heltaliga optimeringsproblem med simplexmetoden och trädsökning (branch-and-bound).

AMPL-versionen som vi ska använda körs direkt i ett terminalfönster under UNIX. AMPL finns även i DOS-version och för Windows-miljö; vissa versioner är gratis och kan laddas ner från AMPL:s hemsida (*www.ampl.com*). Här kan man också köra sina egna (eller testa andras) AMPL-filer direkt över nätet.

AMPL blir tillgängligt genom att man först ger kommandot `module add prog/ampl-demo/` i ett terminalfönster. Observera att detta kommando måste upprepas i varje fönster där AMPL ska köras.

1. Användning av AMPL[1]

Det behövs i det allmänna fallet tre filer då man ska lösa ett optimeringsproblem i AMPL: *modellfil, datafil* och *kommandofil*.

Kommandofilen (kallas här `test.run`), specificerar bland annat:

- Lösare som ska användas (i vårt fall CPLEX).
- Namn på modellfilen (t.ex. `test.mod`).
- Namn på indatafilen (t.ex. `test.dat`).
- Namn på resultatfil samt hur resultaten ska presenteras i denna fil.

AMPL anropas från ett terminalfönster med kommandot

```
ampl < test.run
```

Detta anrop startar AMPL, som därefter utför det som står i kommandofilen `test.run`.

[1]En utförlig beskrivning finns i boken: *AMPL, A Modeling Language For Mathematical Programming* av R. Fourer, D.M. Gay och B.W. Kernighan, The Scientific Press, 1993. (Finns även i senare upplagor.)

2. Modellformulering - ett litet exempel

Vi börjar med att beskriva hur ett litet problem med fyra variabler kan modelleras och lösas. Betrakta problemet nedan.

$$
\begin{array}{lrcrcrcrcl}
\max & z = & 5x_1 & + & 6x_2 & + & 2x_3 & + & 4x_4 & \\
\text{då} & & 3x_1 & + & 2x_2 & + & x_3 & + & 5x_4 & \leq & 80 \\
 & & 2x_1 & + & x_2 & + & 2x_3 & + & 4x_4 & \leq & 45 \\
 & & 3x_1 & - & 3x_2 & + & 4x_3 & + & 5x_4 & \geq & 80 \\
 & & x_1 & , & x_2 & , & x_3 & , & x_4 & \geq & 0
\end{array}
$$

Ett sätt att beskriva detta lilla problem är med modellfilen `small.mod` (se nedan). Här tar man med alla indata till modellen (målfunktions- och bivillkorskoefficienter, högerled, etc.) i modellfilen, och därmed behövs inte datafilen.

```
var x {1..4} >=0;                                          # Variabeldefinition

maximize z: 5*x[1] + 6*x[2] + 2*x[3] + 4*x[4];             # Målfunktion

subject to con1: 3*x[1] + 2*x[2] +   x[3] + 5*x[4] <= 80; # Villkor1
subject to con2: 2*x[1] +   x[2] + 2*x[3] + 4*x[4] <= 45; # Villkor2
subject to con3: 3*x[1] - 3*x[2] + 4*x[3] + 5*x[4] >= 80; # Villkor3
```
small.mod

Varje villkor måste ha ett eget namn (t.ex. con1, con2, etc.) och kommentarer i modellen görs efter tecknet `#`. Kommandofilen kan ha följande utseende.

```
reset;
options solver cplex;
model small.mod;
solve;
display z;
display x;
exit;
```
small.run

Det första kommandot, `reset`, initierar ('nollställer') AMPL. Därefter väljs vilken lösare som ska användas (`options solver`), varefter modellen läses in (`model`) och löses (`solve`). Kommandot `display` skriver ut resultatet. Efter anropet `ampl < small.run` erhålls följande resultat på skärmen.

```
z = 75

x [*] :=
1  10
2   0
3  12.5
4   0;
```

Ett annat sätt att formulera problem är att införa parametrar. Betrakta ett allmänt LP-problem med olikhetsvillkor på formen.

$$
\begin{array}{lrcll}
\max z & = \sum_{j=1}^{N} c_j x_j & & & \\
\text{då} & \sum_{j=1}^{N} a_{ij} x_j & \leq & b_i & i = 1, .., M \\
 & x_j & \geq & 0 & j = 1, .., N
\end{array}
$$

Formuleringen i AMPL kommer att se ut enligt nedan. Fördelen med denna formulering är att modellen alltid kommer att se ut på samma sätt oberoende av data och problemstorlek. Alltså behöver man bara ändra i datafilen om man vill lösa ett större problem.

```
param N;               # Antal variabler
param M;               # Antal villkor
param c{1..N};         # Definition av kostnadsvektor
param a{1..M,1..N};    # Definition av villkorsmatris
param b{1..M};         # Definition av högerledsvektor

var x{1..N} >= 0;      # Variabeldefinition

maximize z: sum {j in 1..N} c[j]*x[j];

subject to con {i in 1..M} : sum {j in 1..N} a[i,j]*x[j] <= b[i];
```
lp.mod

`N, M, c, a` och `b` är parametrar. `N` anger antalet variabler och `M` antalet villkor. Mängden av alla variabler ges av `1..N` medan `1..M` är mängden av alla villkor. Instruktionen `sum {j in 1..N}` summerar således över alla j i variabelmängden. Parametrarna `c, a, b` är kostnadsvektorn, bivillkorsmatrisen respektive högerledsvektorn. Modellens variabler deklareras med `var`. (I modellfilen defineras alltså alla parametrar och variabler som ska användas, men inte deras värden och hur många de är.) I modellfilen skrivs slutligen också optimeringsmodellen in.

För problemet ovan ser datafilen, `lp.dat`, ut på följande sätt. Vi ger här värden på alla parametrar som deklarerats i modellfilen. Observera att vi har bytt tecken på sista villkoret för att göra om det till ett mindre-än-villkor, samt att värdena på vektorerna `c` och `b` är givna på olika sätt (som är ekvivalenta i AMPL).

```
param N := 4; #antal variabler
param M := 3; #antal villkor

param : c :=
1  5
2  6
3  2
4  4;

param a :   1  2  3  4 :=
1           3  2  1  5
2           2  1  2  4
3          -3  3 -4 -5;

param : b :=  1 80    2 45    3 -80;
```
lp.dat

Kommandofilen `lp.run` ser ut ungefär som `small.run`. Skillnaden är att även datafilen måste anges efter kommandot `data`. Kommandofilen innehåller även en extra finess, nämligen att resultatet skrivs ut på fil med hjälp av kommandot `> lp.res`.

```
reset;
options solver cplex;
model lp.mod;
data lp.dat;
solve;
display z > lp.res;
display x > lp.res;
```
lp.run

3. Modellformulering – produktion-lager

Lagervillkor kan formuleras på flera olika sätt, och en svårighet består i att det finns ett initiallager, det vill säga ett lager i tidperiod 0. Nedan visas ett exempel på hur lagervillkor kan modelleras i AMPL.

Vi har ett problem där man ska producera en produkt under T tidsperioder. Produktens efterfrågan, d_t, produktionskostnad, c_t, och produktionskapacitet, u_t, är givna för varje tidsperiod, $t = 1, .., T$. I slutet på varje tidsperiod måste man betala en lagerkostnad, l_t, för de produkter som man behöver lagra. Det ingående lagret är givet och uppgår till y_0 enheter (en konstant).

Problemformulering:

x_t = antal producerade enheter tidsperiod t
y_t = antal enheter i lager i slutet på tidsperiod t

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} c_t x_t + l_t y_t \\ \text{då} \quad & y_{t-1} + x_t = d_t + y_t \quad t = 1, .., T \\ & x_t \leq u_t \quad t = 1, .., T \\ & x_t, y_t \geq 0 \quad t = 1, .., T \end{aligned}$$

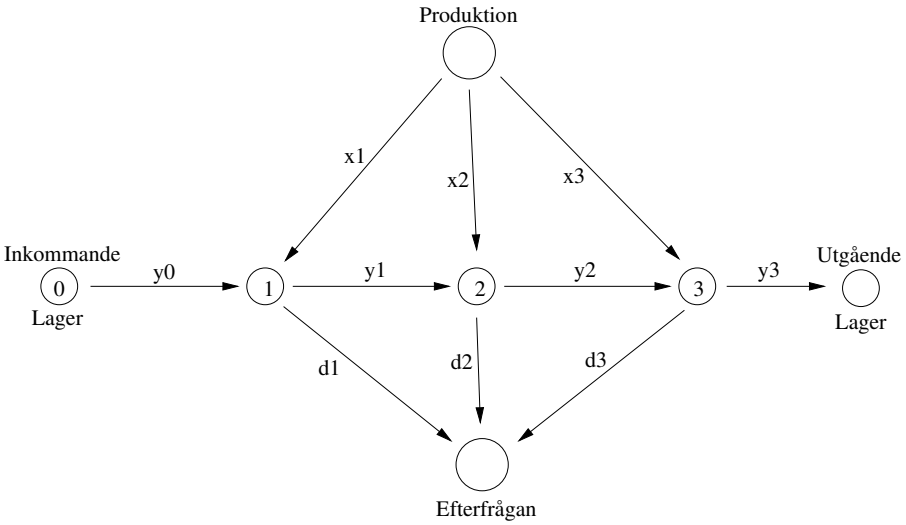
Detta problem kan i AMPL modelleras på följande sätt.

```
param T;           # antal tidsperioder
param initlager;   # initiallager
param c{1..T};     # produktionskostnad
param u{1..T};     # produktionskapacitet
param l{1..T};     # lagerkostnad
param d{1..T};     # efterfrågan

var x{1..T} >= 0;  # antal tillverkade enheter
var y{0..T} >= 0;  # antal lagrade enheter

minimize totalkostnad: sum {t in 1..T} (c[t]*x[t] + l[t]*y[t]);

subject to lager {t in 1..T} :
y[t-1] + x[t] = d[t] + y[t];

subject to initiallager :
y[0] = initlager;

subject to kapacitetsbegr {t in 1..T} :
x[t] <= u[t];
```

lager.mod

Att lägga märke till i denna modell är att variablerna y_t är definerade för $t = 0, .., T$, och att det enklaste sättet att ta hänsyn till initiallagret är att ha ett speciellt villkor för y_0. Notera också att villkorsnamnen är valda så att man enkelt förstår innebörden av varje villkorsgrupp.

Om man studerar ett problem med tre tidsperioder ser flödesschemat ut enligt nedan.



Nedan följer en datafil för tre tidsperioder.

```
param T := 3;              #antal tidsperioder
param initlager := 5;      #initiallager

param :  c   l    d   u:=
1       25   4   10  20
2       36   3   15  16
3       30   5   12  10;
```

lager.dat

För detta lilla problem går det naturligtvis även bra att skriva alla villkor i klartext, som nedan. För större problem blir dock denna metod naturligtvis ohanterlig.

```
var x{1..3} >= 0;
var y{0..3} >= 0;

minimize z: 25*x[1] + 36*x[2] + 30*x[3] + 4*y[1] + 3*y[2] + 5*y[3] ;

subject to lagerperiod1 : y[0] + x[1] = 10 + y[1];
subject to lagerperiod2 : y[1] + x[2] = 15 + y[2];
subject to lagerperiod3 : y[2] + x[3] = 12 + y[3];

subject to initiallager : y[0] = 5;

subject to kapacitetsbegrperiod1 : x[1] <= 20;
subject to kapacitetsbegrperiod2 : x[2] <= 16;
subject to kapacitetsbegrperiod3 : x[3] <= 10;
```

lager1.mod

4. Modellformulering - Dietproblem

Göta Lantmän tillverkar många olika fodertyper. Nu skall tre fodersorter för grisar introduceras på marknaden, Bas, Standard och Special. Behovet av respektive grisfodersort är givet. Ingredienserna i fodervarianterna ska ligga mellan vissa givna minimi- och maximinivåer när det gäller protein, kolhydrater samt vitamin X, för att rätt näringsinnehåll ska erhållas (givet i viktprocent).

Dessa ingredienser finns i Göta Lantmäns råvaror vete, råg, korn, havre och majs. Innehållet i respektive råvara (i viktsprocent), kostnaden (per kg) samt tillgänglig mängd (kg) är också givet.

Göta Lantmän önskar minimera råvarukostnaden för produktionen av de önskade fodermängderna.

Problemformulering:

$$x_{ij} = \text{antal ton av råvara } i = \begin{cases} 1 & \text{vete} \\ 2 & \text{råg} \\ 3 & \text{korn} \\ 4 & \text{havre} \\ 5 & \text{majs} \end{cases} \quad \text{som används till fodersort } j = \begin{cases} 1 & \text{Bas} \\ 2 & \text{Standard} \\ 3 & \text{Special} \end{cases}$$

$$\begin{array}{lcl} c_i & = & \text{Kostnad per ton för råvara } i \\ a_{ki} & = & \text{Innehåll av näringsämne } k = \begin{cases} 1 & \text{Proteiner} \\ 2 & \text{Kolhydrater} \\ 3 & \text{Vitamin X} \end{cases} \text{ i råvara } i \\ l_{kj} & = & \text{Undre gräns för innehåll av näringsämne } k \text{ i fodersort } j \\ u_{kj} & = & \text{Övre gräns för innehåll av näringsämne } k \text{ i fodersort } j \\ b_j & = & \text{Behov av fodersort } j \\ d_i & = & \text{Tillgång av råvara } i \end{array}$$

$$\min \quad z = \sum_{i=1}^{5} c_i \left(\sum_{j=1}^{3} x_{ij} \right)$$

$$\begin{array}{lcccccll} \text{då} & l_{kj}b_j & \leq & \displaystyle\sum_{i=1}^{5} a_{ki}x_{ij} & \leq & u_{kj}b_j & k = 1\ldots3, j = 1\ldots3 & \text{Näringskrav} \\ & & & \displaystyle\sum_{i=1}^{5} x_{ij} & = & b_j & j = 1\ldots3 & \text{Efterfrågan} \\ & & & \displaystyle\sum_{j=1}^{3} x_{ij} & \leq & d_i & i = 1\ldots5 & \text{Råvarutillgång} \\ & & & x_{ij} & \geq & 0 & i = 1\ldots5, j = 1\ldots3 & \text{Ickenegativitet} \end{array}$$

Detta problem kan i AMPL formuleras på följande sätt. Naturligtvis kan man använda bättre (längre) namn på variabler, parametrar, etc., om så önskas.

```
set RÅVARA;
set FODERSORT;
set NÄRINGSÄMNE;

param c{RÅVARA};
param a{NÄRINGSÄMNE,RÅVARA};
param l{NÄRINGSÄMNE,FODERSORT};
param u{NÄRINGSÄMNE,FODERSORT};
param b{FODERSORT};
param d{RÅVARA};

var x{RÅVARA,FODERSORT} >= 0;

minimize z: sum {i in RÅVARA} c[i] * (sum{j in FODERSORT} x[i,j]);

subject to Näringskrav {k in NÄRINGSÄMNE, j in FODERSORT} :
l[k,j]*b[j] <= sum {i in RÅVARA} a[k,i]*x[i,j] <= u[k,j]*b[j];

subject to Efterfrågan {j in FODERSORT}:
sum{i in RÅVARA} x[i,j] = b[j];

subject to Råvarutillgång {i in RÅVARA}:
sum{j in FODERSORT} x[i,j] <= d[i];
```
gotalant.mod

Vi har här infört mängderna `RÅVARA, FODERSORT` och `NÄRINGSÄMNE` med hjälp av deklarationen `set`. Detta ger enkla och förståeliga modeller (och även datafilen blir mer lättläst). Den störst vinsten är dock att utdata (lösningen) är mycket enkel att tolka. Nedan följer datafilen (`gotalant.dat`) samt utdatafilen (`gotalant.res`).

```
set RÅVARA := Vete Råg Korn Havre Majs;
set FODERSORT := Bas Standard Special;
set NÄRINGSÄMNE := Protein Kolhydrater VitaminX;

param : c    d :=
Vete   1.5  99999
Råg    1.6  99999
Korn   1.0  99999
Havre  1.7   1000
Majs   2.5    500;

param a :    Vete Råg Korn Havre Majs :=
Protein       10  10   6    11    12
Kolhydrater   60  45  40    50    40
VitaminX     2.0 1.0 0.5   2.2   2.3;

param l :  Bas  Standard  Special :=
Protein       6       7         9
Kolhydrater  35      40        50
VitaminX    0.5     1.0       1.2;

param u :   Bas  Standard  Special :=
Protein      9999    9999      9999
Kolhydrater    55      60        70
VitaminX     9999    9999      9999;

param : b :=  Bas 500   Standard 300   Special 400;
```
gotalant.dat

```
z = 1400

x :=
Havre Bas          0
Havre Special      0
Havre Standard     0
Korn  Bas        500
Korn  Special    100
Korn  Standard   200
Majs  Bas          0
Majs  Special      0
Majs  Standard     0
Rag   Bas          0
Rag   Special      0
Rag   Standard     0
Vete  Bas          0
Vete  Special    300
Vete  Standard   100
;
```

gotalant.res

5. Syntax

Nedan beskrivs några av de viktigare kommandona i AMPL. Notera speciellt att AMPL skiljer på versaler och gemener ('stora och små bokstäver'), varför t.ex. x och X uppfattas som olika variabler. Vidare ignorerar AMPL radbrytningar; istället måste varje kommando eller deklaration avslutas med ett semikolon (;).

param

Konstanter deklareras (i modellfilen) med `param`, på samma sätt som `var` deklareras. När en parameter ska tilldelas ett värde (i datafilen) ser syntaxen ut på följande sätt beroende på om det är en parameter, en parametervektor eller en parametermatris:

```
param N := 7;

param : vek := 1 50  2 75  3 100;

param matr : 1   2   3 : =
         1  80   9  77
         2  11 120  13;
```

Parametern `N` får värdet 7, vektorn `vek` tilldelas värden `vek[1]=50`, `vek[2]=75` och `vek[3]=100` samt matrisen `matr` tilldelas värden `matr[1,1]=80`, `matr[1,2]=9`, `matr[1,3]=77`, `matr[2,1]=11`, etc.

set

En mängd deklareras med `set`. En sådan kan bestå både av numeriska eller symboliska värden.

```
set CARS := SAAB VOLVO BMW;
set UddaNR := 1 3 5 7 9;
set VECKOR := 1..N;
```

var

Alla variabler måste deklareras. Variabelnamnet kan bestå av valfritt antal tecken.

```
var X;             # En flyttalsvariabel med namn X
var Y binary;      # Binär variabel
var Q integer;     # Heltalsvariabel
var X{1..8};       # Variabelvektor X[i], i=1..8
var X{1..8,1..20}; # Variabelmatris X[i,j]
var Weigth{CARS};  # Variabelvektor Weigth[i], i element i mängden CARS
var Y{1..N}>=0;    # Variabelvektor Y[i], Y[i]>=0, i=1..N
```

sum

Används i målfunktion och bivillkor för att utföra summering över ett antal variabelindex. Kan även användas för att konstruera dubbelsummor, trippelsummor, etc. (se exempel nedan).

```
sum{i in CARS} Weight[i]        # Summerar Weight för alla element i mängden CARS.
sum{i in 1..20} (x[i] - y[i])   # Summerar x[i]-y[i] för i=1 till 20.
sum{i in CARS,t in 1..T} x[i,t] # Summerar x för alla element i mängderna CARS
                                  och 1..T.
```

maximize / minimize

Specificerar målfunktionen. Notera att målfunktionen måste ges ett namn. Efter namnet följer ett kolon (:) och därefter målfunktionen.

```
maximize profit: sum{i in Units} c[i] * x[i];
```

subject to

Specificerar ett bivillkor eller en grupp av bivillkor. Notera att alla bivillkor måste namnges. Efter namnet följer ett kolon (:) och därefter bivillkoret.

```
subject to bivillkor1: x + y = 7;

subject to lager{i in Produkter}:
    Tillverkat[i] + Kvar[i] - Sålt[i] <= Lagerkap[i];

subject to biv3{i in N}:
    sum{j in 1..i} x[j] >= d[i];
```

display

Används i kommandofilen för att visa resultat från lösningen av ett problem.

```
display x;               # Skriver ut alla x-värden på skärmen
display x > file.res;    # Skriver ut alla x-värden på filen file.res
display constr1.dual;    # Skriver ut dualvariablerna till bivillkoren constr1
display constr1.slack;   # Skriver ut slacket för villkoren constr1
display x.rc;            # Skriver ut reducerad kostnad for variablerna x
```

aritmetriska och logiska uttryck

- if-then-else, for, repeat
- && ||
- < <= == ! = > >=

```
sum{i in I: i<5 && i != 1} # Summerar alla i i mängden I, där i är mindre än 5
                             och i skiljt från 1
repeat loop {              # Repeat sats och namn på repeatsats
  for{i in I}{             # For loop över alla i i mängden I
    if rc[i] < 0 then {    # If loop
      ...;
      continue;            # Fortsätter for-loopen (överflödig i detta exempel)
    }
    else break loop;       # Avslutar repeat-loopen
  }
}
```