Breve guida all'uso di AMPL

Renato Bruni

AMPL (A Modeling Language for Mathematical Programming) è un linguaggio di modellazione per la programmazione matematica. Serve ad esprimere un problema di ottimizzazione in una forma che sia comprensibile dal solutore. È un linguaggio algebrico, cioè contiene primitive per esprimere la notazione matematica normalmente utilizzata nello scrivere i modelli.
Tutto viene espresso attraverso normali file di testo. Gli spazi e gli a capo sono generalmente liberi, nel senso che possono essere inseriti senza provocare errori, ma conviene rispettare un formato preciso per rendere agevole lettura ed interpretazione di questi file.
Occorre scrivere un file di modello, in cui è descritta la struttura logica del modello del problema in esame. I valori numerici del problema in esame possono essere inseriti nello stesso file di modello o in uno o più file di dati. Inserendo la struttura logica nel file di modello ed i dati nel file di dati è possibile cambiare i dati in modo semplice e senza pericolo di introdurre errori nel modello. Il file di modello ha obbligatoriamente estensione `.mod`, quello di dati estensione `.dat`. I dati possono anche essere letti, tramite opportuni comandi in altri file di testo.
Possono essere inoltre creati file di comandi (analoghi ai file batch del DOS) che hanno estensione `.run`.

Dichiarazione di parametri ed insiemi

I parametri sono i dati (non variabili) che compaiono nel modello che descrive il nostro problema di ottimizzazione. Un parametro può essere utilizzato solo dopo la sua dichiarazione. Si usa la parola chiave `param`, seguita dal nome del parametro.
La più semplice dichiarazione di parametro è:

```
param t;
```

AMPL è case sensitive, cioè le lettere maiuscole vengono considerate differenti da quelle minuscole. Un parametro va chiamato sempre nello stesso modo ogni volta che si fa riferimento ad esso. Ogni istruzione termina con '`;`'.
Si possono inserire commenti preceduti dal simbolo '`#`'.
Se un parametro è composto da un insieme di valori (ad esempio un vettore), serve un indice per distinguere i diversi valori. Nella dichiarazione, la dimensione del parametro viene indicata tra parentesi graffe.

```
param C;
param costi{1..C};      # indice numerico che va da 1 a C
```

Quanto sopra serve a dichiarare un parametro `C`, ed un insieme di parametri `costi` identificati da un indice numerico che varia da `1` fino a `C`.
Un insieme è semplicemente una lista degli elementi che lo costituiscono. Serve ad esempio per dichiarare un parametro indicizzato su un insieme, ma anche per varie altre cose. Un insieme si dichiara con la parola chiave `set`.

```
set VAR;                # insieme delle variabili
set VINC;               # insieme dei vincoli
parm a{VINC,VAR};       # matrice
```

Quanto sopra ha l'effetto di dichiarare un insieme `VAR` (i cui elementi non sono ancora stati specificati), un insieme `VINC` (i cui elementi non sono ancora stati specificati), ed un parametro bidimensionale `a` (cioè una matrice) con elementi identificati da coppie di valori, il primo appartenente all'insieme `VINC` ad il secondo all'insieme `VAR`.

Per fare riferimento ad una specifica componente di un parametro si scrive l'indice che individua quella componente tra parentesi quadre.

```
...costi[10]...
....a[i,j]...
```

`a[i,j]` è il valore della matrice `a` corrispondente all' *i-esimo* elemento dell'insieme `VINC` e al *j-esimo* elemento dell'insieme `VAR`. Il primo indice è quello di riga, il secondo quello di colonna.
Gli elementi di un insieme possono essere numeri o stringhe. Se poi quell'insieme serve ad indicizzare un parametro, bisogna ricordarsi che gli indici sono gli elementi dell'insieme, e quindi rispettivamente numeri o stringhe. Scegliendo numeri si ha il vantaggio di poter individuare tramite operazioni matematiche i vari elementi del parametro, cosa che in molti casi è necessaria. Scegliendo stringhe, al contrario, questo non è possibile, e le istruzioni risulteranno spesso più lunghe e macchinose, mentre si ha il vantaggio di poter usare nomi autoesplicativi.
Per accorgersi di eventuali errori nei dati, si possono imporre restrizioni sui parametri. Ad esempio, se sappiamo che un certo parametro `T` deve essere un intero maggiore di 1, possiamo scriverlo come limitazione nel modo seguente, in modo che ci venga segnalato subito quando ciò non è verificato.

```
param T >1 integer;
```

Alcuni parametri possono avere valori calcolati in base ad altri parametri

```
param costi{A};
param sommacosti := sum{i in A} costi[i];
```

Il parametro `sommacosti` ha come valore la somma dei parametri `costi`.

Assegnazione di parametri ed insiemi

I valori dei parametri vengono assegnati (e non possono più essere cambiati) sempre attraverso la parola chiave `param`, seguita dal nome del parametro (come dichiarato nel file di modello), da `:=` e da un valore

```
param T:= 1;
param C := 20;
```

Gli elementi di un insieme si assegnano con la parola chiave `set`, seguita dal nome del insieme, da `:=` e da una sequenza di valori separati da spazi

```
set vinc :=  A B ;
set var  :=  X Y ;
```

Dopo l'assegnazione degli insiemi è possibile assegnare i valori ai parametri indicizzati con tali insiemi.
Per un parametro monodimensionale, occorre specificare gli indici seguiti dai valori che assume in corrispondenza dei vari indici. Per aumentare la leggibilità conviene scriverlo nel seguente modo. Notare che il punto e virgola che conclude l'istruzione è solo alla fine dell'ultima riga.

```
param vettore:=
            A 1
            B 3;
```

Per un parametro bidimensionale occorre specificare le coppie di indici seguite dai corrispondenti valori.

```
param   a:=
       X  A  6
       Y  A  2
       X  B  8
```

```
      Y  B  0;
```

Eventuali valori non presenti nell'elenco si intendono con valore non specificato.
Parametri bidimensionali possono essere anche rappresentati nella più leggibile forma tabellare: il nome del parametro è seguito da ':', da un elenco di indici che costituiscono una dimensione del parametro, da ':=', e dagli elementi dell'altro indice seguiti dalle sequenze di valori del parametro corrispondenti.

```
param   a:  X  Y:=
        A   6  2
        B   8  0;
```

Se una matrice è molto larga e poco alta, e quindi non è agevolmente visibile nella schermata, conviene scriverne la trasposta, facendo seguire al nome della matrice la parola `(tr)`.
Il formato con cui sono scritti i parametri indicizzati serve solo ad aumentare la leggibilità. Il solutore avrebbe letto correttamente la matrice `a` anche se fosse stata scritta

```
param a: X Y := A 6 2 B 8 0;
```

Per un parametro avente più di due dimensioni, basta indicare le *n-ple* dei valori degli indici seguite dal valore del parametro corrispondente a quella *n-pla* di indici. Esempio:

```
param cost:=
    X  A  ST    2
    X  A  DL    3
    X  B  ST    2
    X  B  DL    1
    Y  A  ST    2
    Y  A  DL    2
    Y  B  ST    2
    Y  B  DL    2;
```

I valori di un insieme numerico si possono assegnare enumerandoli

```
set NUMERI:={1,2,5,10};
```

O, se possibile, semplicemente indicando gli estremi.

```
set NUMERI:= 1..100;
```

È possibile indicare anche il passo tra numeri consecutivi. Ad esempio, i numeri pari fino a 100 sono

```
set PARI := 2..100 by 2;
```

Se si vuole imporre che un parametro `e` abbia un valore all'interno di un certo insieme `S` si può specificare tramite il comando seguente

```
param e symbolic in S;
```

Se un parametro deve essere modificabile, anziché assegnarlo con `:=` occorre inizializzarlo con il comando `default` e poi modificarlo col comando `let`

```
param peso default 100; # peso inizializzato a 100
.............
let peso := 200;        # peso aggiornato a 200
```

Infine, i valori dei parametri possono anche essere letti con il comando read sia da un qualsiasi file di testo sia immettendoli da tastiera. In questo caso il file di testo deve semplicemente riportare, separati da spazi, tutti i valori che volgiamo leggere, senza bisogno di indicare gli indici come nel file `.dat`

```
read r < valore.txt;  # legge r nel file valore.txt
```

Le seguenti istruzioni invece scrivono il valore del parametro peso e ne leggono il nuovo valore da tastiera

```
printf"Peso valeva %d", peso;
printf"\nDimmi il nuovo valore di peso ”;
read peso;
```

Ovviamente leggere valori da file di testo conviene prevalentemente per paramentri di grande dimensione. Ad esempio, per leggere uno dopo l'altro tutti gli elementi di una matrice occorre fare così (usando anche l'istruzione `for`)

```
for{j in T, i in S}
{
read r[j,i] < matrice_r.txt;
};
```

Espressioni aritmetiche

I numeri vengono rappresentati con un eventuale segno, ed il separatore per le cifre decimali è il punto. La notazione esponenziale viene accettata.

```
-10
2.78
-4.8e3  # -4.8 per 10 alla 3, cioè 4800
```

Le funzioni aritmetiche che è possibile utilizzare sono:

Significato	indicato con
Valore assoluto di x	`abs(x)`
Arcoseno(x)	`acos(x)`
Arcoseno iperbolico(x)	`acosh(x)`
Arcocoseno(x)	`asin(x)`
Arcocoseno iperbolico(x)	`asinh(x)`
Arcotangente iperbolico(x)	`atanh(x)`
Seno(x)	`sin(x)`
Coseno(x)	`cos(x)`
Tangente(x)	`tan(x)`
Tangente iperbolica(x)	`tanh(x)`
Parte intera inferiore di x	`ceil(x)`
Parte intera superiore di x	`floor(x)`
Logaritmo naturale $\log_e$ x	`log(x)`
Logaritmo decimale $\log_{10}$ x	`log10(x)`
Esponenziale e^x	`exp(x)`
Radice quadrata di x	`sqrt(x)`
Minimo tra 2 o più numeri (x, y, z, …)	`min(x,y,z,…)`
Massimo tra 2 o più numeri (x, y, z, …)	`max(x,y,z,…)`

Gli operatori che è possibile utilizzare, in ordine di precedenza decrescente, sono:

Significato	Tipo	indicato con	o anche con
Potenza	Aritmetico	^	**
Numero negativo	Aritmetico	-	
Somma	Aritmetico	+	

Sottrazione	Aritmetico	-	
Moltiplicazione	Aritmetico	*	
Divisione	Aritmetico	/	
Divisione intera	Aritmetico	div	
Modulo	Aritmetico	mod	
Differenza non negativa: max(a-b,0)	Aritmetico	less	
Sommatoria	Aritmetico	sum	
Produttoria	Aritmetico	prod	
Minimo	Aritmetico	min	
Massimo	Aritmetico	max	
Unione di insiemi	Insiemistica	union	
Intersezione di insiemi	Insiemistica	inter	
Differenza tra insiemi	Insiemistica	diff	
Differenza simmetrica tra insiemi	Insiemistica	symdiff	
Prodotto cartesiano tra insiemi	Insiemistica	cross	
Appartenenza ad un insieme	Insiemistica	in	
Non appartenenza ad un insieme	Insiemistica	not in	
Maggiore, maggiore o uguale	Aritmetico	>, >=	
Minore, minore o uguale	Aritmetico	<, <=	
Uguale	Aritmetico	=	==
Diverso	Aritmetico	<>	!=
Negazione logica	Logico	not	!
And logico	Logico	and	&&
Quantificatore esistenziale logico	Logico	exists	
Quantificatore universale logico	Logico	forall	
Or logico	Logico	or	||
If then else	Aritmetico	if then else	

Esempi:

```
abs(sum {i in ORIG} offerta[i] – sum{j in DEST} domanda[j])
```

è il valore assoluto della somma delle offerte meno la somma delle domande

```
sum{i in A} d[i] + (if t=10 then a[i] else -a[i])
```

è la somma dei `d` più gli `a` se `t=10`, e meno gli `a` altrimenti.

Variabili

A differenza dei parametri, sono dei simboli il cui valore numerico deve essere calcolato dal solutore, e non indicato da noi. Rappresentano le grandezze incognite che vogliamo conoscere risolvendo il problema di ottimizzazione. Si dichiarano con la parola chiave `var`, ed appaiono solo nel file di modello. Le variabili devono essere dichiarate prima di poter essere utilizzate. La più semplice dichiarazione di variabile è

```
var x;
```

normalmente le variabili sono un insieme di valori indicizzato su in insieme, eventualmente con delle restrizioni.

```
var  x{p in VAR} >=0, <=LIMIT;
```

indica un insieme di variabili con indice che varia nell'insieme `VAR`, che devono essere tutte `>=0` e `<=` del parametro `LIMIT`. Altre possibili restrizioni sono `integer` per specificare variabili intere e `binary` per specificare variabili binarie.
È possibile, anche se inusuale, specificare restrizioni di uguaglianza sulle variabili.

```
var Buy{j in FOOD} = f_min[j];
```

questo vuol dire che siamo interessati solo alle soluzioni in cui le variabili hanno il valore dei corrispondenti parametri `f_min`.
È possibile, inoltre, specificare valori iniziali sulle variabili.

```
var Buy{j in FOOD} := f_min[j];
```

questo vuol dire che le variabili vengono inizializzate ai valori dei corrispondenti parametri `f_min`, ma il solutore le può poi modificare. Questo serve nel caso si voglia far lavorare il solutore a partire da una soluzione data. Si noti che il significato è completamente diverso dal caso precedente.

Obiettivo

È ciò che vogliamo massimizzare o minimizzare nel nostro problema di ottimizzazione. Può essere uno solo. È dichiarato con la parola chiave `maximize` o `minimize`, seguita obbligatoriamente da un qualsiasi nome per questo obiettivo, dai due punti, e da una espressione in cui possono comparire solo gli insiemi, i parametri e le variabili già definiti. È conclusa, al solito, da '`;`'.
La più semplice dichiarazione di obiettivo è

```
minimize obiettivo: x;
```

Significa che vogliamo che il solutore trovi un valore per le variabili tale da rendere minimo il valore di x.

```
minimize total_cost: sum{j in VAR} cost[j]*x[j];
```

Vuol dire minimizzare la somma dei prodotti delle variabili per i rispettivi costi. Di regola, nell'obiettivo compaiono sempre le variabili.
L'espressione può essere complicata, come nel caso di

```
minimize obj: sum{i in N_H, j in N_W} (z[i,j]
- sum{r in N_H, s in N_W} x[r,s] * vdef[i-r] * hdef[j-s])^2
+ q[1]*( sum{k in N_H1, h in N_H} (x[k+1,h] - x[k,h] - vgap[k,h])^2 )
+ q[2]*( sum{l in N_H, m in N_W1} (x[l,m+1] - x[l,m] - hgap[l,m])^2 );
```

Conviene, per quanto possibile, evitare espressioni complicate, in quanto rendono il processo di soluzione molto più lento.

Vincoli

Sono delle specifiche che dobbiamo soddisfare nel nostro problema di ottimizzazione. Sono dichiarati con la parola chiave `subject to`. Questa parola chiave può essere abbreviata in `s.t.` ed è opzionale, dato che ogni dichiarazione che non comincia con una parola chiave viene considerata vincolo. Devono avere un nome, seguito dai due punti, e una espressione in cui possono comparire solo gli insiemi, i parametri e le variabili già definiti. Deve ovviamente comparire un operatore di relazione (`<`, `<=`, `>`, `>=`, `=`) Sono conclusi, al solito, da '`;`'.
La più semplice dichiarazione di vincolo è

```
subject to vinc: x <=4;
```

Vuol dire che vogliamo che il solutore trovi una soluzione in cui la variabile `x` valga al più 4 (questo tipo di semplice vincolo poteva anche essere specificato come restrizione nella dichiarazione della variabile x).

```
subject to time: sum {p in PROD} rate[p]*make[p] <= disp;
```

Vuol dire che la somma dei prodotti delle componenti di stesso indice di `rate` e `make` deve essere minore di `disp`.
I vincoli possono essere indicizzati:

```
s. t. limiti{s in VINC}: sum{I in VAR} a[s,i]*x[i] <= b[s];
```

Se l'insieme `VINC` è formato dai quattro elementi `A, B, C, D`, equivale ai quattro vincoli

```
s. t. limite_a: sum{I in VAR} a[A,i]*x[i] <= b[A];
s. t. limite_b: sum{I in VAR} a[B,i]*x[i] <= b[B];
s. t. limite_c: sum{I in VAR} a[C,i]*x[i] <= b[C];
s. t. limite_d: sum{I in VAR} a[D,i]*x[i] <= b[D];
```

Con gli strumenti sintattici visti è possibile esprimere direttamente una grande varietà di modelli.

File di comandi

Avviando l'eseguibile `ampl.exe` si ha una finestra in cui è possibile lavorare da riga di comando, cioè scrivendo le istruzioni da tastiera. Ad esempio, per scegliere come solutore cplex si scrive

```
option solver cplex;
```

Per usare un file di modello `pian.mod` ed un file di dati `pian.dat` si scrive

```
model pian.mod;
data pian.dat;
```

Infine, per risolvere il modello corrente si usa il comando

```
solve;
```

e per vedere i valori dei vari elementi del modello (ad esempio le variabili `x`) si usa

```
display x;
```

Tutti questi comandi possono essere anche scritti in un file di comandi, cioè un file di testo con estensione `.run`, dove è possibile usare anche istruzioni di controllo del flusso come `for` o `repeat` per eseguire in sequenza diverse operazioni. Tale file va poi lanciato con l'istruzione

```
commands esegui.run;
```