Appunti sul linguaggio di programmazione AMPL

Roberto Cordone, Maurizio Bruglieri, Leo Liberti [1] (maintainer)

Milano, Ottobre 2001, ultima modifica 030618

[1] Dipartimento di Elettronica e Informazione, Politecnico of Milano, Piazza Leonardo da Vinci 32, 20133 Milano (Si prega di segnalare errori o punti poco chiari all'indirizzo e-mail `liberti@elet.polimi.it`).

Indice

Questa dispensa si propone di fornire una guida in italiano ai concetti fondamentali di uno dei software di modellazione più diffusi: AMPL. La trattazione è di livello elementare: leggendo queste pagine, chiunque abbia un minimo di dimestichezza con un linguaggio di programmazione dovrebbe poter imparare velocemente a formulare problemi di ottimizzazione lineare con AMPL, in modo da poterli trasmettere a qualsiasi risolutore e accedere semplicemente ai risultati.

Il primo capitolo spiega la filosofia sulla quale si fondano i software detti *generatori algebrici di modelli*. Il secondo introduce l'uso del software AMPL. Il terzo descrive la struttura e la sintassi di un modello formulato nel linguaggio AMPL. Si è volutamente data ai modelli una struttura più rigida di quella effettivamente richiesta dal linguaggio, allo scopo di guidare meglio l'utente inesperto lungo i passi necessari a tradurre un problema di ottimizzazione in un modello AMPL. Una trattazione più approfondita del linguaggio e delle possibilità che offre si può trovare in [1].

AMPL non è l'unico software di modellazione disponibile: altri software e linguaggi svolgono la stessa funzione. Essi hanno caratteristiche diverse, e soprattutto ognuno ha una sintassi propria, ma la struttura generale di un modello scritto nei diversi linguaggi tende ad essere sempre la stessa. Di tutti è in genere disponibile in rete una versione per studenti, capace di gestire problemi con un numero limitato di variabili e di vincoli:

- **GAMS** (*General Algebraic Modeling System*)
  al sito `http://www.gams.com`
- **MPL** (*Mathematical Programming Language*)
  al sito `http://www.maximal-usa.com/mpl/`
- **LINGO** al sito `http://www.lindo.com`

Quella del software AMPL si scarica dal sito `http://www.ampl.com`. La dispensa fa riferimento all'interfaccia grafica del software, che la casa produttrice non fornisce più ed è quindi disponibile solo a chi possieda il manuale delle versioni precedenti. Sono però indicate nello stesso sito diverse interfacce realizzate da utenti, che le offrono gratuitamente. Ovviamente, quanto si riferisce alla struttura dei modelli e alla sintassi AMPL mantiene tutta la sua validità.

Capitolo 1

I generatori algebrici di modelli

Fra gli anni '50 e gli anni '70, la programmazione matematica compì progressi sostanziali, cui, tuttavia, non corrispose un'applicazione altrettanto diffusa a problemi reali. Un forte ostacolo fu la difficoltà pratica di stendere i modelli, raccogliere e organizzare i dati, programmare gli algoritmi risolutivi e analizzare i risultati ottenuti.

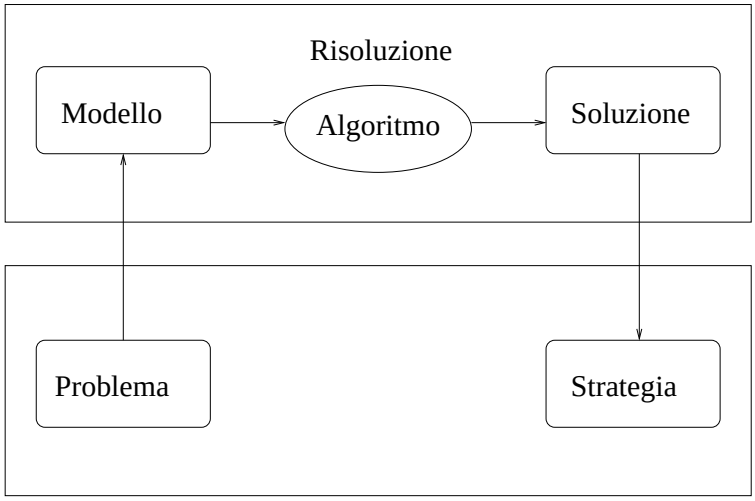

Figura 1.1: Schema dell'approccio modellistico per passare da un problema concreto a una strategia operativa

L'approccio modellistico, infatti, ha indubbi vantaggi, ma in esso, come ricorda la Figura 1.1, il passaggio dal problema alla strategia risolutiva è tutt'altro che immediato: si tratta di formulare prima un modello, poi di risolverlo, e infine di tradurre la soluzione in azioni concrete. Il passaggio dal problema al modello e quello dalla soluzione alla strategia operativa non sono banali, e richiedono anzi la dose maggiore di creatività. Anche il passaggio intermedio, però, (quello dal modello alla soluzione) è solo apparentemente astratto e matematico. Infatti, se si fa uso (come in genere avviene) di uno strumento informatico, questo passaggio non si riduce all'applicazione di un algoritmo, ma comprende (vedi

Figura 1.2):

1. la traduzione del modello (pensato, o scritto in linguaggio matematico) e dei dati (disponibili su carta, o in un database informatico) in strutture dati accessibili a un risolutore
2. la realizzazione di un risolutore, che trasformi i dati in una soluzione opportunamente codificata
3. la traduzione della soluzione in un formato accessibile all'utente (tabelle, grafici o altro)

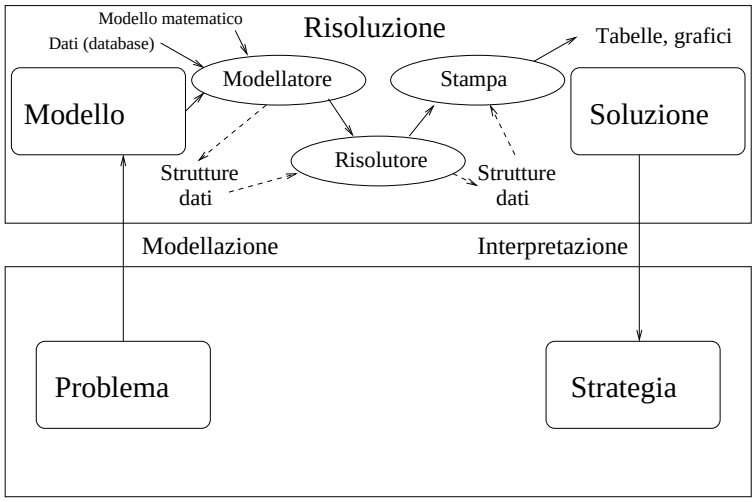

Figura 1.2: Dettaglio della fase di risoluzione nell'approccio modellistico

Grandi successi in campo matematico e un'enorme potenza di calcolo possono dar luogo a risolutori molto potenti, ma questi sono poco utili a livello applicativo se l'utente non dispone di un'interfaccia comoda verso il risolutore, ovvero di un software che gestisca modelli e dati appartenenti al modo reale e interroghi il risolutore. I *generatori algebrici di modelli*, fra cui AMPL, costituiscono quest'interfaccia, cioè si occupano del primo e del terzo passo, lasciando il secondo al risolutore. Le caratteristiche principali dei generatori algebrici di modelli sono:

- fornire un linguaggio semplice per descrivere modelli complessi, un linguaggio che sia contemporaneamente
  - *ad alto livello*, cioè comprensibile a un essere umano
  - *formalmente strutturato*, cioè accessibile a un risolutore
- permettere all'utente di comunicare con il risolutore attraverso file di testo anziché attraverso strutture dati, in modo da non richiedergli conoscenze informatiche approfondite e da poter formulare il modello con un semplice editor, qualunque sia la piattaforma su cui viene scritto e quella su cui viene risolto

- permettere all'utente di comunicare con diversi risolutori, in modo da poter sfruttare i più potenti sul mercato, ovvero quelli disponibili
- tenere distinti la *struttura logica* del modello (variabili di decisioni, obiettivi, vincoli e le loro relazioni) dal valore dei *dati numerici*

Si può descrivere questa situazione come segue:

$$\begin{array}{ccccccc} utente & \leftrightarrow & modello & \leftrightarrow & programma & \leftrightarrow & solutore \\ & & & & (es. AMPL) & & (es. CPLEX) \\ & & & & \uparrow & & \\ & & & & dati & & \\ & & & & \uparrow & & \\ & & & & database & & \end{array}$$

Utilizzando un generatore di modelli l'utente non si occupa più direttamente di interrogare il risolutore, ma può concentrarsi sulla stesura del modello, usando un linguaggio di alto livello che gli rende più semplice descrivere problemi reali anche molto complessi. Può affidare la gestione dei dati a un database esterno cui accedere quando necessario. Può sostituire il risolutore senza dover riscrivere il modello. Infine la separazione tra la struttura logica e i dati evita che piccole modifiche nella struttura del modello o nei dati comportino di riscrivere tutto: diventa infatti semplice usare lo stesso modello su dati differenti o applicare modelli diversi allo stesso problema.

Capitolo 2

Il software AMPL

Il software AMPL è in grado di tradurre modelli di Programmazione Matematica scritti nel linguaggio AMPL in un formato comprensibile a un generico risolutore di programmazione matematica.

2.1 Installazione di AMPL sotto Windows

È necessario scaricare da internet i seguenti files:

- `http://www.ampl.com/GUI/amplvb.zip`
- `http://netlib.bell-labs.com/netlib/ampl/student/mswin/cplex.zip`
- `http://www.ampl.com/NEW/TABLES/amplcml.zip`

Decomprimere `amplvb.zip` con WinZip e lanciare il programma di installazione `setup.exe`. Quando l'installazione avrà termine, selezionare la voce "Trova" dal menu Avvio e successivamente "File o cartelle". Cercare il file `ampl.exe` sul drive dove AMPL è stato installato, e individuarne la cartella di installazione (la scelta di default è la cartella `AMPLWIN` sotto la cartella `Programmi` nel drive `C:`). Decomprimere ora i files in `cplex.zip` e spostarli manualmente nella cartella di installazione di AMPL. Lanciando il file eseguibile `Amplwin.exe` situato nella cartella di installazione di AMPL, dovrebbe partire l'ambiente grafico sperimentale (per ora piuttosto scarno) di AMPL, come mostrato in fig (**??**).

Il terzo file scaricato, `amplcml.zip`, contiene una versione a linea di comando (per Windows) di AMPL, che è ridondante rispetto alla versione con interfaccia grafica, ma che risulta tuttavia utile per la cartella `MODELS` contenente gli esempi. Si decomprima dunque `amplcml.zip` specificando a WinZip di estrarre anche le cartelle.

AMPLè disponibile per diversi sistemi operativi, Linux incluso. Le istruzioni per lo scaricamento e l'installazione di AMPL sotto Linux è disponibile sul sito `www.ampl.org`.

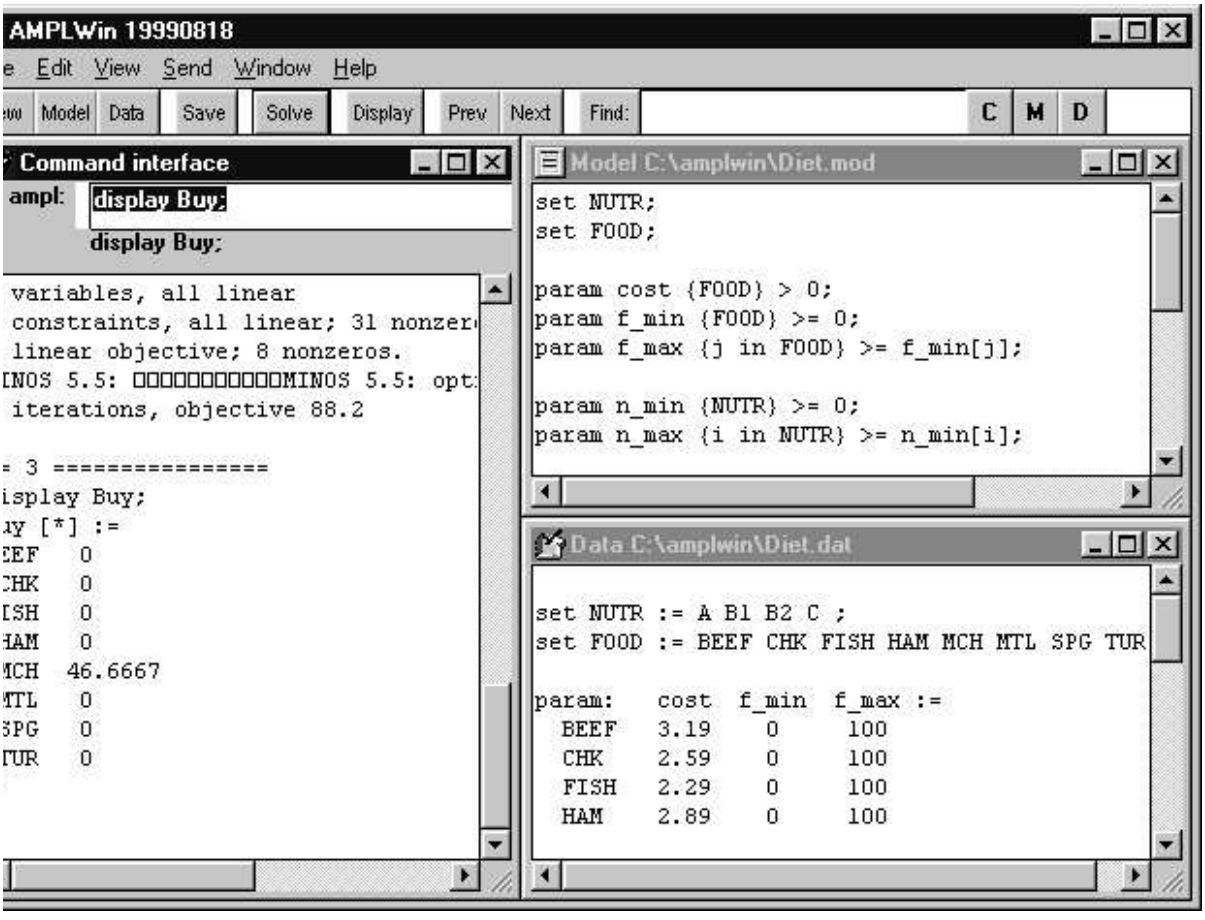

Figura 2.1: L'ambiente grafico di AMPL sotto Windows.

2.2 Scrivere e risolvere un modello con AMPL

Questa sezione insegna a caricare un modello già formulato, passarlo a un risolutore e accedere ai risultati. In genere, i generatori algebrici di modelli vengono distribuiti con esempi didattici, con i quali ci si può impratichire sull'uso del software. Il procedimento consiste di sette semplici passi:

- avviare AMPL
- caricare il file del modello
- caricare il file dei dati
- specificare il solutore esterno (es. CPLEX, MINOS, etc.)
- risolvere il modello
- vedere la soluzione

Avviare AMPL Selezionare la voce "AMPLWin" dal menu "Programmi" del menu di Avvio. La finestra principale di AMPL, come mostrato in fig. **??**, consiste di una barra dei menu, una barra dei pulsanti, un campo denominato "ampl:" per il comando da inserire, una finestra *Transcript* che mostra l'output di AMPL, una finestra *Model* e una finestra *Data* (vuote all'avvio di AMPL). La finestra *Model* mostra il file modello caricato, e la finestra *Data* mostra il file di dati caricato.

Caricare il file del modello Il secondo passo è caricare il file del modello. La cartella (directory) `MODELS` citata sopra contiene diversi modelli di esempio. Si tratta di semplici file di testo, che hanno obbligatoriamente l'estensione `.mod`. Nel seguito, faremo riferimento al file `diet.mod`. Per raggiungerlo:

1. premere il pulsante *Model* sulla barra dei pulsanti della finestra di AMPL (questo apre una finestra di selezione files);
2. raggiungere la cartella `MODELS`;
3. selezionare il file `diet.mod` e fare un doppio click su di esso.

Caricare il file dei dati Per caricare il file dei dati, procediamo come per il modello: premendo il pulsante *Data*, cercando nella cartella *MODELS* il file `diet.dat` e aprendolo con un doppio click.

Specificare il solutore esterno AMPL permette di risolvere problemi di programmazione matematica usando diversi risolutori. Con la procedura di installazione descritta sopra sono disponibili due solutori: CPLEX (per la programmazione lineare e intera), e MINOS (un solutore locale per la programmazione nonlineare continua). La scelta default di AMPL è quella di usare MINOS. Poiché questo corso concerne primariamente problemi di PL e PLI, sarà

necessario specificare, una volta caricato il modello e i dati, di usare CPLEX, mediante l'istruzione "`option solver cplex;`", da inserire nel campo denominato `ampl:` di immissione comando corrente.

Sul sito di AMPL sono disponibili parecchi solutori esterni. Ognuno deve essere installato manualmente nella cartella d'installazione di AMPL, e selezionato mediante un'opportuno comando

`option solver` *nome_solutore* ;

immesso nel campo dei comandi.

Risolvere il modello Una volta che si è specificato il risolutore, per risolvere il modello basta cliccare sul pulsante *Solve*. La finestra *Transcript* fornisce:

- una copia del file di modello e una del file di dati o eventuali errori di sintassi in uno dei files;
- informazioni sullo status di AMPL e qualche statistica sul problema;
- informazioni sullo status del solutore esterno (CPLEX segnala la soluzione ottima con la stringa "optimal solution").

Visualizzare la soluzione Per accedere alla soluzione si usa il comando

`display` *nome_variabile* ;

immesso nel campo dei comandi. Il solutore CPLEX stampa automaticamente il valore della funzione obiettivo.

Esistono comandi per mostrare diversi dettagli relativi alla soluzione, inclusi i valori delle variabili di slack, i prezzi ombra e così via.

Capitolo 3

Il linguaggio AMPL

AMPL è un linguaggio ad alto livello per descrivere modelli di Programmazione Matematica. In questo capitolo ne descriviamo la sintassi, a partire da alcune sue caratteristiche generali, per proseguire con i diversi elementi che costituiscono un tipico modello AMPL.

3.1 Caratteristiche generali del linguaggio

Sensibilità alle maiuscole

AMPL è *case sensitive*, cioè distingue le lettere maiuscole dalla minuscole. Così le dichiarazioni

```
var x1;
var X1;
```

introducono due variabili distinte, denominate $x1$ e $X1$. È buona norma non sfruttare questa possibilità.

Lunghezza delle righe

Le righe del programma hanno una lunghezza massima di 255 caratteri: quelli eccedenti vengono ignorati.

Commenti

È possibile, e consigliabile, aggiungere alle istruzioni del programma commenti che le rendano più chiare. I commenti sono introdotti dal simbolo `#`, che indica al software di ignorare il seguito della riga.

```
var x{Cibi};   #  quantità da acquistare per ciascun cibo
```

Separatori

Le diverse istruzioni del modello sono separate dal simbolo `;`

```
var x1;
var x2;
minimize obiettivo: x1 + x2;
```

Le singole parole di un modello possono invece essere separate da qualsiasi numero di spazi, di "a capo" e di tabulazioni. È bene far uso di questa libertà per dare al modello un formato più leggibile, eventualmente intervallando commenti alle parole.

```
set Nutrienti := Calorie Proteine Calcio Ferro Vitamine;

param Fabbisogno{Nutrienti} :=  3       # Calorie
                               70       # Proteine
                                0.8     # Calcio
                               12       # Ferro
                               75 ) ;   # Vitamine
```

Gli indici dei vettori a più dimensioni sono separati dal simbolo `,`

```
A[i,j]
```

Intervalli

Si può esprimere un insieme di valori ordinati attraverso i suoi estremi, separati dal simbolo `..`

```
set indice = 1..5;
```

3.2 Struttura di un programma AMPL

Sebbene sia lecito scrivere in un solo file AMPL sia il modello sia i dati, è concettualmente preferibile tenere separati questi due termini, costruendo:

- un *file di modello*, obbligatoriamente di estensione **.mod**, che descrive la struttura logica del modello (indici, variabili di decisione, funzione obiettivo e vincoli)
- un *file di dati*, obbligatoriamente di estensione **.dat**, che contiene i valori numerici del problema

Mantenendo fisicamente separato il modello dai dati, è possibile applicare lo stesso modello a dati diversi, o cambiare i dati senza dover modificare il modello, evitando il rischio di introdurvi errori.

Gli elementi principali di un programma AMPL sono:

- *insiemi*
- *dati*
- *variabili*
- *funzione obiettivo*
- *vincoli*

3.3 Insiemi

Gli insiemi "descrivono il mondo" in cui ci si muove, cioè definiscono il dominio del problema e la sua dimensione. Servono a raccogliere in un vettore o una matrice tutti i dati o le variabili che si riferiscono a termini omogenei, in modo da trattarli simultaneamente. Ciascun insieme del modello va:

- *dichiarato* nel file **.mod**, per indicare che un certo nome rappresenta un insieme; si impiega la parola chiave **set**, seguita dal nome dell'insieme e dal separatore ;

```
set Nutrienti;
```

- *definito* nel file **.dat**, per assegnare all'insieme gli elementi che ne fanno parte; si impiega la parola chiave **set**, seguita dal nome dell'insieme, dal simbolo := , dagli elementi separati da spazi o "a capo"; il separatore ; chiude l'istruzione.

```
set Nutrienti := Calorie Proteine Calcio Ferro Vitamine;
```

Si presti attenzione alla differenza tra "dichiarare" e "definire", che nel seguito verrà sottolineata meno fortemente.

Oltre che simbolici, gli insiemi possono essere numerici. In tal caso, si possono definire come intervalli, indicandone solo gli estremi.

```
set Mesi := 1..12;
```

La parola chiave **by** consente di specificare la distanza fra due elementi successivi dell'insieme:

```
set MesiDispari := 1..12 by 2;
```

equivale a

```
set MesiDispari := 1 3 5 7 9 11;
```

Ovviamente, **1..12 by 1** equivale a **1..12**.

Si fa riferimento a un elemento singolo di un insieme con la notazione

```
Mesi[i];
```

dove **i** è un valore numerico.

Operazioni su insiemi

Dati due insiemi **A** e **B**, è possibile compiere su di loro diverse operazioni, per definire insiemi derivati. Supponiamo che sia

```
set A := 1  3  5  7  9 11;
set B := 9 11 13 15 17;
```

Operazione	Risultato	Significato
A union B	1 3 5 7 9 11 13 15 17	elementi di **A** o di **B**
A inter B	9 11	elementi di **A** e di **B**
A diff B	1 3 5 7	elementi di **A** e non di **B**
A symdiff B	1 3 5 7 13 15 17	elementi di **A** o di **B**, ma non di entrambi
card(A)	6	numero degli elementi di **A**

Sottoinsiemi

Si può dichiarare un insieme come sottoinsieme di un altro attraverso la parola chiave **within**

```
set Mesi;
set MesiEstivi within Mesi;
```

La definizione di **Mesi** e **MesiEstivi** dovrà essere coerente con questa dichiarazione; altrimenti, AMPL segnalerà un errore.

Insiemi ordinati

In generale, gli insiemi sono *non ordinati*. È possibile dichiararli ordinati facendo seguire al nome dell'insieme la parola chiave **ordered**.

```
set A ordered;
```

Sugli insiemi ordinati si possono compiere operazioni ulteriori, rispetto agli insiemi ordinati.

Operazione	Risultato	Significato
first(A)	1	primo elemento di **A**
last(A)	11	ultimo elemento di **A**
next(3,A,2)	7	secondo elemento di **A** dopo **3**
prev(5,A,2)	1	secondo elemento di **A** prima di **5**
next(3,A)	5	primo elemento di **A** dopo **3**
prev(5,A)	3	primo elemento di **A** prima di **3**
ord(5,A)	2	posizione di **5** in **A**
ord0(4,A)	0	come sopra, ma rende **0** se **4** non è in **A**
member(3,A)	5	terzo elemento di **A**

Dato un insieme **A** ordinato e un insieme **B** generico (ordinato o no), la loro differenza **A diff B** è ordinata; gli altri insiemi derivati non lo sono.

Negli insiemi ordinati *circolari*, all'ultimo elemento segue il primo.

```
set A circular;
```

Collezioni di insiemi

Si può dichiarare un insieme di insiemi con la notazione

```
set Indice;
set Collezione{Indice};
```

che dichiara tanti insiemi quanti sono gli elementi dell'insieme **Indice**.

Insiemi pluridimensionali

Spesso è utile poter lavorare con insiemi pluridimensionali, vale a dire insiemi i cui elementi sono n-uple (si tratta sempre di n-uple *ordinate*). Gli insiemi pluridimensionali si dichiarano facendo seguire al nome dell'insieme la parola chiave **dimen** e il numero di termini della n-upla.

```
set TRIPLET dimen 3;
```

Si definiscono come gli insiemi monodimensionali, elencandone gli elementi. Questi vengono racchiusi fra parentesi tonde e i loro termini sono separati da virgole.

```
set TRIPLET := ( 1,7,1 ) ( Giugno,Luglio,Agosto ) ( 1,3,5 )
               ( 3,1,5 );
```

Si badi che gli elementi **(1,3,5)** e **(3,1,5)** possono coesistere in **TRIPLET** perché l'ordine delle loro componenti è significativo.

Quando un insieme pluridimensionale è il prodotto cartesiano di più insiemi monodimensionali, si può definire direttamente come tale, con la parola chiave **cross**

```
set X := 0 1 2 3;
set Y := 0 1 2;
set Z := 1 2 3;
set Punto3D := X cross Y cross Z;
```

Questo equivale a definire **Punto3D** come **(0,0,1) (0,0,2) (0,0,3) (1,0,1) ... (3,2,3)**.

Viceversa, le parole chiave **setof** e **in** consentono di definire un insieme monodimensionale come la *proiezione* su un indice di un insieme pluridimensionale.

```
set TRIPLET dimen 3;
set X := setof{(i,j,h) in TRIPLET} i;
```

Espressioni di indicizzazione

È possibile definire insiemi non solo esplicitamente, assegnando loro gli elementi o costruendoli a partire da altri insiemi, ma anche implicitamente, attraverso le così dette *espressioni di indicizzazione*. In genere queste espressioni si usano per definire un insieme senza attribuirgli un nome.

```
{A}                        # elementi di A
{A,B}                      # coppie ordinate di elementi,
                           # tratti rispettivamente da A e da B
{i in A,j in B}            # coppie ordinate di elementi,
                           # tratti rispettivamente da A e da B
{i in A: Costi[i] >= 10}   # elementi di A tali che il valore
                           # associato nel vettore Costi è >= 10
{i in A,i in A}            # coppie di elementi tratti da A e B,
                           # purché identici
```

L'insieme viene poi usato per fare da indice in una sommatoria, o nella definizione di un vettore o una matrice di dati, variabili, vincoli. In un'espressione di indicizzazione possono comparire, oltre agli insiemi componenti, anche gli indici che li scorrono. Questi sono necessari se l'espressione viene usata in un contesto nel quale è necessario poter individuare i singoli elementi dell'insieme (ad esempio, in una sommatoria, i cui termini dipendono dall'indice stesso).

```
sum{j in Cibi: Costi[j] >= 10} X[j];
```

L'istruzione somma i valori del vettore **X** corrispondenti agli indici nell'insieme **Cibi** i cui corrispondenti valori nel vettore **Costi** sono non inferiori a 10.

3.4 Dati

I dati sono i valori numerici che definiscono in dettaglio il problema che si vuole affrontare, una volta precisata la sua struttura logica. Essi sono assegnati una volta per tutte, al contrario delle variabili, che vengono modificate dal risolutore in modo da determinare la soluzione ottima al problema. Ovviamente, è possibile modificare i dati generando problemi diversi (per questo spesso i dati vengono chiamati *parametri*), ma ciò avviene sempre a monte del processo risolutivo: il risolutore non può alterare i dati.

Dichiarazione

Come gli insiemi, i dati vengono dichiarati nel file del modello **.mod** e definiti nel file dei dati **.dat**. Per dichiararli, si impiega la parola chiave **param**, seguita dal nome del dato.

```
param NumeroMesi;
```

È possibile dichiarare vettori o matrici di dati, facendo seguire al nome un'espressione di indicizzazione che stabilisca l'insieme cui corrispondono gli elementi del vettore o della matrice.

```
set Cibi;
set Nutrienti;

param Costi{Cibi};
param A{Nutrienti,Cibi};
```

L'espressione di indicizzazione può esplicitare gli indici

```
param Costi{j in Cibi};
param A{i in Nutrienti,j in Cibi};
```

ma non è necessario che lo faccia. Diventa invece necessario nel definire i vincoli, dato che in tal caso gli indici sono di solito ripresi nel corpo della definizione.

Per far riferimento al singolo dato, si usa la notazione con le parentesi quadre: **Costi[2]** è il valore associato al secondo elemento del vettore Costi, ovvero è il costo del secondo elemento dell'insieme **Cibi**, mentre **A[i,j]** è il valore associato all'i-esimo nutriente e allo j-esimo cibo nella matrice **A**.

Definizione

I dati vengono definiti con la parola chiave **param** e il simbolo **;**

```
param N := 10;
```

Vettori e matrici si definiscono elencando le coppie indice-valore, separate da spazi o "a capo" (questo spesso aumenta la leggibilità).

```
set Cibi := Pane Latte Carne Verdure;

param Costi := Pane    10
               Latte    5
               Carne   20
               Verdure  8;
```

Si possono dichiarare come dati anche gli estremi di un intervallo usato per definire un insieme:

```
param N integer;
set Costi{1..N};
```

È possibile definire simultaneamente diversi vettori

```
param : Costi    Disponibilita :=
Pane       10          4
Latte       5          5
Carne      20          2
Verdure     8          3  ;
```

Si noti il simbolo `:` dopo la parola chiave **param**: serve a indicare che ciascuna delle parole seguenti è il nome di un vettore di dati.

Con una sintassi analoga si possono definire le matrici colonna per colonna, anziché elencarne gli elementi uno per uno. Oltre all'indice di ogni colonna, bisogna riportare il nome della matrice.

```
param A: Calorie Proteine Calcio Ferro Vitamine :=
Pane       300       12      10     4      2
Latte      500       50     200    10     60
Carne      200      100      20    20     20
Verdure     50        3       5    15    100 ;
```

Se la matrice ha molte colonne e poche righe, per renderla più leggibile si può indicarne la trasposta, segnalandolo con la parola chiave `(tr)`

```
param A(tr): Pane Latte Carne Verdure :=
    Calorie   300   500   200      50
    Proteine   12    50   100       3
    Calcio     10   200    20       5
    Ferro       4    10    20      15
    Vitamine    2    60    20     100 ;
```

Per le matrici a più di due dimensioni, oltre che elencare gli elementi uno per uno, si possono riportare le "sezioni bidimensionali" ottenute fissando tutti gli indici tranne due. Ogni sezione è introdotta dall'indicazione degli indici fissati e di quelli rimasti liberi, racchiusa fra parentesi quadre. Nell'esempio che segue, una matrice tridimensionale viene sezionata rispetta alla seconda dimensione.

```
set DimX := I II III;
set DimY := 1 2;
set DimZ := a b c;
set Spazio := DimX cross DimY cross DimZ;

param Cubo :=
    [*,1,*] :   a   b   c :=
          I    10   4   3
          II    5   1  14
          III   3   2  20
    [*,2,*] :   a   b   c :=
          I    10   4   3
          II    5   1  14
          III   3   2  20 ;
```

Si notino gli asterischi che contraddistinguono gli indici non fissati e il fatto che il separatore `;` compare solo al termine e non fra una sezione e l'altra.

Istruzioni di controllo

Nel dichiarare un parametro, si possono imporre condizioni che il suo valore, specificato nel file dei dati, deve rispettare. Tali condizioni sono ridondanti, perché potrebbe farsene carico l'utente, ma è buona programmazione aggiungerle, per facilitare l'individuazione di errori. Così

```
param N > 1 integer;
```

specifica che il dato **N** è un intero maggiore di 1. Se non lo è, in fase di soluzione AMPL si arresta e dà una segnalazione di errore.

Un modo più raffinato di eseguire controlli di coerenza sui dati richiede l'uso della parola chiave **check**, seguita dal simbolo **:** e da un'espressione logica (vedi Sezione 3.7). L'espressione

```
check{j in Cibi}: sum{i in Nutrienti} A[i,j] >= 6;
```

controlla che la somma dei coefficienti **A[i,j]** su tutte le righe sia, colonna per colonna, maggiore o uguale a 6. Se non lo è, in fase di soluzione AMPL si arresta e dà una segnalazione di errore.

3.5 Variabili

Le variabili sono le grandezze che descrivono la soluzione del problema. Il loro valore deve essere determinato dal risolutore. Le variabili vengono dichiarate nel file del modello attraverso la parola chiave **var**, seguita dal nome della variabile ed, eventualmente, da restrizioni al suo valore. Queste ultime possono essere descritte da espressioni logiche oppure dalle parole chiave **integer** e **binary**

```
var x;
var y >= 0, <= SUP;
var z integer;
var w binary;
var v = 100;
```

dove **SUP** deve essere un dato dichiarato e definito. La variabile **y** deve rimanere nell'intervallo compreso fra 0 e **SUP**, la variabile **z** è intera e **w** può assumere solo i valori 0 e 1. Infine, alla variabile **v** viene assegnato un valore fisso: si tratta di un caso abbastanza degenere. Il linguaggio AMPL ammette quest'ultima possibilità per consentire esperimenti nei quali si fissa il valore di alcune variabili, lasciando le altre libere.

Anche le variabili si possono raccogliere in vettori o matrici grazie alle espressioni indicizzate.

```
set Cibi;

var Consumo{Cibi};
var Quantita{j in Cibi} >= 0, <= QuantitaMax[j];
```

Queste istruzioni dichiarano due variabili vettoriali indicizzate sull'insieme **Cibi**. Nella dichiarazione della prima variabile è equivalente indicare solo l'insieme di definizione della variabile (**Consumo{Cibi}**) o esplicitarne anche l'indice (**Consumo{j in Cibi}**). Nella seconda dichiarazione, invece, l'indice deve essere esplicito, perché compare anche in una restrizione e quest'ultima è diversa per ciascuna variabile nel vettore **Quantita**. Ovviamente, **QuantitaMax** è un dato vettoriale definito precedentemente.

Per far riferimento a una singola variabile in un vettore o matrice, si impiega la consueta notazione con le parentesi quadre.

```
Flusso[i,j]
```

Valori iniziali

Benché il valore delle variabili sia sottratto al controllo dell'utente e affidato interamente al risolutore (entro i vincoli stabiliti dal modello) è consentito imporre alle variabili un valore iniziale. Questo è molto importante nei problemi non lineari, dato che influenza la soluzione finale che viene determinata.

```
var v := 100;
```

Si noti la differenza rispetto alla restrizione $\mathbf{var\ v = 100}$, che obbliga la variabile **v** a rimanere pari a 100 sino alla fine.

3.6 Espressioni algebriche

La funzione obiettivo e i vincoli che la soluzione finale deve rispettare sono generalmente espressioni algebriche complesse, costruite a partire dai dati e dalle variabili. Esse impiegano i seguenti operatori, posti in ordine di precedenza decrescente:

Operatore	Simbolo
Potenza	^ (oppure **)
Numero negativo	-
Somma	+
Sottrazione	-
Prodotto	*
Divisione	/
Divisione intera	div
Modulo	mod
Differenza non negativa (max(a-b,0))	less
Sommatoria	sum
Produttoria	prod
Minimo	min
Massimo	max
Unione di insiemi	union
Intersezione di insiemi	inter
Differenza di insiemi	diff
Differenza simmetrica di insiemi	symdiff
Prodotto cartesiano di insiemi	cross
Appartenenza a un insieme	in
Non appartenenza a un insieme	notin
Maggiore, Maggiore o uguale	>,>=
Minore, Minore o uguale	<,<=
Uguale	= (oppure ==)
Diverso	<> (oppure !=)
Negazione logica	not (oppure !)
And logico	and
Quantificatore esistenziale	exists
Quantificatore universale	forall
Or logico	or (oppure ||)
If then else	if then else

Possono inoltre impiegare le seguenti funzioni:

Operatore	Simbolo
Valore assoluto	abs(x)
Arco seno	asin(x)
Arco seno iperbolico	asinh(x)
Arco coseno	acos(x)
Arco coseno iperbolico	acosh(x)
Arco tangente iperbolica	atanh(x)
Seno	sin(x)
Seno iperbolico	sinh(x)
Coseno	cos(x)
Coseno iperbolico	cosh(x)
Tangente	tan(x)
Tangente iperbolica	tanh(x)
Approssimazione intera per difetto	floor(x)
Approssimazione intera per eccesso	ceil(x)
Logaritmo naturale	log(x)
Logaritmo decimale	log10(x)
Esponenziale	exp(x)
Radice quadrata	sqrt(x)
Minimo fra più numeri	min(x1,x2,...,xn)
Massimo fra più numeri	max(x1,x2,...,xn)

3.7 Espressioni logiche

Nei modelli compaiono inoltre diverse espressioni di tipo logico. Le istruzioni **check** sono intrinsecamente espressioni logiche. Come vedremo nel seguito, anche i vincoli lo sono, dato che esprimono un confronto fra due espressioni algebriche attraverso gli operatori di relazione (<, <=, =, >= o >), che deve essere verificato.

Oltre al confronto numerico fra espressioni algebriche, un'espressione logica pò contenere i seguenti operatori classici.

Operatore	Significato
not	(not **a**) è vera quando **a** à falsa, e viceversa
and	(**a** and **b**) è vera quando sono vere sia **a** sia **b**
or	(**a** or **b**) è vera quando almeno una fra **a** e **b** è vera

Vi possono comparire le relazioni di appartenenza di un elemento a un insieme (**12 in Mesi**) o di un sottoinsieme a un insieme più ampio (**MesiEstivi within Mesi**).

Infine, due operatori logici di uso comune sono i quantificatori **exists** e **forall**, che generalizzano su più di due espressioni, rispettivamente, gli operatori **or** e **and**.

```
exists {j in Cibi} Costo[j] >= 3
forall {j in Cibi} Costo[j] >= 3
```

La prima espressione è vera quando almeno uno degli elementi dell'insieme **Cibi** ha costo maggiore o uguale a 3, la seconda quando tutti lo hanno.

If then else

L'espressione logica **if then else** svolge una funzione analoga all'espressione **(if x a b)** del linguaggio Lisp e al costrutto **(x ? a : b)** del linguaggio C. Essa valuta l'espressione logica che segue la parola chiave **if**. Se è vera, restituisce il risultato dell'espressione algebrica che segue la parola chiave **then**, altrimenti quello dell'espressione che segue la parola chiave **else**.

```
set Periodi ordered;
set Prodotti;

param LivelloMax1{Prodotti};
param LivelloMax2{Prodotti};

var Livello{p in Prodotti,t in Periodi}
            <= (if t = first(Periodi)
                  then LivelloMax1[p]
                  else LivelloMax2[p]);
```

Queste istruzioni definiscono un insieme ordinato di periodi temporali (ad esempio, settimane) e uno di prodotti, nonché due vettori che misurano le capacità di magazzino per ciascun prodotto. Il primo vettore si riferisce al primo periodo, il secondo agli altri. La variabile **Livello**, che descrive il livello di occupazione del magazzino non deve superare mai la capacità. Il costrutto **if then else** le impone un limite superiore pari a **LivelloMax1** nel primo periodo, a **LivelloMax2** negli altri.

Espressioni indicizzate

L'impiego principale delle espressioni logiche è nelle *espressioni indicizzate*. Queste servono a derivare dagli insiemi già definiti altri insiemi, da impiegare nelle sommatorie o nel definire vettori di dati, di variabili, di vincoli. È possibile dare a questi insiemi un nome simbolico e impiegarlo nel modello, oppure usare direttamente l'espressione indicizzata, senza assegnarle un nome. Si definisce l'insieme derivato facendo seguire al nome di quello originario il simbolo **:** e un'espressione logica: solo gli elementi per i quali l'espressione logica è vera vanno a far parte dell'insieme derivato.

```
# nel file dei dati
set CibiCostosi := {j in Cibi} Costo[j] >= 3;

# nel file del modello
var SpesaCibiCostosi =
      sum{j in CibiCostosi} Costo[j] * Quantita[j];
```

oppure direttamente

```
# nel file del modello
var SpesaCibiCostosi =
      sum{j in Cibi: Costo[j] >= 3} Costo[j] * Quantita[j];
```

Un altro esempio, più complesso, calcola la spesa per cibi il cui contenuto in tutti i fattori nutritivi sia maggiore o uguale a 2. Si determina attraverso una condizione logica il sottoinsieme dei cibi **j** in cui **A[i,j]** $>=$ **2** per tutti i nutrienti **i**.

```
var SpesaCibiNutrienti =
      sum{j in Cibi: forall {i in Nutrienti} A[i,j] >= 2 }
        Costo[j] * Quantita[j];
```

Vediamo un esempio in cui un'espressione indicizzata si usa per definire una matrice di variabili. È dato un insieme di punti su una mappa (**set Punti;**) con indice i; si vuole esprimere il numero di persone che si spostano da un punto all'altro con una variabile **Flusso{Punti,Punti}**, i cui due indici variano nell'insieme dei punti. La variabile però è definita solo quando i due indici si riferiscono a punti diversi.

```
set Punti;
var Flusso{i in Punti,j in Punti: j <> i};
```

3.8 Funzione obiettivo

La funzione obiettivo specifica la grandezza del problema di cui si vuole trovare il valore ottimale. Viene introdotta dalla parola chiave **minimize** o **maximize**, seguita da un nome (obbligatorio), dal simbolo **:** e dall'espressione che definisce la funzione obiettivo in termini dei dati e delle variabili.

```
minimize CostoTotale : sum{j in Cibi} Costi[j] * Quantita[j];
```

È possibile definire più funzioni obiettivo (anche un vettore di funzioni obiettivo indicizzato su un insieme): AMPL considera solo la prima e ignora le altre. Usando AMPL da linea di comando è infatti possibile controllare quale funzione obiettivo viene ottimizzata.

3.9 Vincoli

I vincoli distinguono le soluzioni ammissibili da quelle inammissibili. Vengono introdotti dalla parola chiave **subject to** (o **subj to**, o **s. t.**), seguita da un nome (obbligatorio), dal simbolo **:**, da un'espressione e dal consueto separatore **;**. L'espressione consiste nel confronto di due espressioni algebriche attraverso un operatore di relazione (**<**, **<=**, **=**, **>=** o **>**)

```
subject to vincolo: x <= 1;
```

Anche i vincoli possono essere indicizzati

```
subject to Fabbisogno{i in Nutrienti}:
  sum{j in Cibi} A[i,j] * Quantita[j] >= Richiesta[i];
```

Questa istruzione dichiara un vincolo per ogni elemento dell'insieme **Nutrienti**. Il vincolo impone per ogni fattore nutritivo nell'insieme **Nutrienti** che la somma dei prodotti fra i coefficienti **A[i,j]** e le quantità di cibo **Quantita[j]** soddisfi il fabbisogno del fattore stesso. Come dire:

```
subject to FabbisognoCalorie:
  sum{j in Cibi} A[Calorie,j] * Quantita[j] >= Richiesta[Calorie];
subject to FabbisognoProteine:
  sum{j in Cibi} A[Proteine,j] * Quantita[j] >= Richiesta[Proteine];
subject to FabbisognoCalcio:
  sum{j in Cibi} A[Calcio,j] * Quantita[j] >= Richiesta[Calcio];
subject to FabbisognoFerro:
  sum{j in Cibi} A[Ferro,j] * Quantita[j] >= Richiesta[Ferro];
subject to FabbisognoVitamine:
  sum{j in Cibi} A[Vitamine,j] * Quantita[j] >= Richiesta[Vitamine];
```

Non è chi non veda i vantaggi della forma compatta.

Per comodità e leggibilità entrambi i termini della relazione possono contenere variabili e costanti. Le variabili possono essere ripetute nella stessa relazione o nei suoi due termini. Nei vincoli semplici (cioè senza indici) possono comparire variabili semplici, vettori costanti, sommatorie su vettori o matrici di variabili purché estese a tutti gli indici. Nei vincoli vettoriali, invece, occorre che l'indice del vincolo sia compatibile con gli indici residui (non utilizzati) nel vincolo stesso.

Vincoli e limiti sulle variabili

Imporre una restrizione al valore di una variabile o definire un vincolo sono modi equivalenti di distinguere le soluzioni ammissibili da quelle inammissibili. In generale, il primo tipo di descrizione consente ai risolutori di programmazione lineare di affrontare il problema in modo più efficiente. Però riducono l'informazione disponibile in uscita, dato che le variabili duali vengono riportate solo per i vincoli espliciti, e non per le restrizioni sulle variabili.

Capitolo 4

Visualizzare e analizzare la soluzione

4.1 Visualizzare i risultati

Per accedere ai dati, ai risultati e a tutti gli elementi del modello, si usa l'istruzione **display**, che dà loro un formato analogo a quello impiegato per definirli.

```
ampl: display Cibi;
set Cibi := Pane Latte Carne Verdure;
```

L'insieme che segue la parola chiave **display** può anche essere costruito con un'espressione indicizzante.

```
ampl: display {j in Cibi: Costi[j] >= 10};
set {j in Cibi: Costi[j] >= 10} := Pane Carne;
```

Lo stesso avviene per i dati e per le variabili, i vincoli, la funzione obiettivo. Per visualizzare i dati è sufficiente che essi siano stati caricati, mentre per gli altri termini finché il problema non è risolto viene visualizzato solo il valore iniziale, che non è significativo.

Visualizzare le variabili

Se si richiede di visualizzare una variabile, AMPL ne restituisce il valore. È però possibile accedere ad altre informazioni interessanti, facendo seguire al nome della variabile una specificazione introdotta da un punto:

- **.val** indica il valore (equivale a non aggiungere nulla)
- **.lb** indica il limite inferiore sulla variabile (specificato dall'utente e rafforzato dalla fase di *presolve*
- **.lslack** indica la differenza fra il valore ottimo e il limite inferiore (**.val** - **.lb**)

- **.lrc** indica il costo ridotto rispetto al limite inferiore (di quanto cambia l'ottimo per piccole modifiche del limite inferiore)
- **.ub** indica il limite superiore sulla variabile (specificato dall'utente e rafforzato dalla fase di *presolve*
- **.uslack** indica la differenza fra il limite superiore e il valore ottimo (**.ub - .val**)
- **.urc** indica il costo ridotto rispetto al limite superiore (di quanto cambia l'ottimo per piccole modifiche del limite superiore)
- **.slack** si riferisce al limite più vicino al valore ottimo
- **.rc** si riferisce al limite più vicino al valore ottimo

Visualizzare i vincoli

Se si richiede di visualizzare un vincolo, AMPL restituisce il valore della corrispondente variabile duale. È però possibile accedere ad altre informazioni interessanti, facendo seguire al nome del vincolo una specificazione introdotta da un punto. Per chiarirne il significato, bisogna immaginare di aver posto il vincolo nella forma $\textit{lower bound} \leq \textit{vincolo} \leq \textit{upper bound}$, dove *vincolo* contiene tutte le variabili e *lower bound* e *upper bound* sono costanti.

- **.body** indica il valore del vincolo
- **.lb** indica il limite inferiore sul vincolo
- **.lslack** indica la differenza fra il vincolo e il limite inferiore (**.body - .lb**)
- **.ldual** indica la variabile duale associata al limite inferiore
- **.ub** indica il limite superiore sul vincolo
- **.uslack** indica la differenza fra il limite superiore e il vincolo (**.ub - .body**)
- **.udual** indica il costo ridotto rispetto al limite superiore
- **.slack** si riferisce al limite più vicino al valore ottimo
- **.dual** si riferisce al limite più vicino al valore ottimo (equivale a non aggiungere nulla)

Controllare il formato

L'istruzione **print** equivale a **display**, ma indica di non applicare alcun formato. L'istruzione **printf**, invece, impone il formato specificato nella stringa che segue immediatamente il comando, secondo le convenzioni del linguaggio C.

4.2 Analisi di sensitività

Per analisi di sensitività si intende lo studio delle variazioni della soluzione ottima e del valore ottimo di un problema di decisione in conseguenza ai cambiamenti dei dati del problema. In particolare nel caso di un problema di programmazione lineare valuteremo le variazioni che si manifestano nella soluzione e nel valore ottimo in seguito a cambiamenti dei valori numerici dei coefficienti c della funzione obiettivo e del vettore b dei termini noti dei vincoli. Per dettagli teorici sulla analisi di sensitività si faccia riferimento a [**?**].
L'analisi di sensitività è uno strumento di fondamentale importanza sul piano applicativo. Essa infatti consente al decisore di valutare la robustezza delle conclusioni del modello rispetto alle inevitabili imprecisioni con cui i coefficienti del problema sono stimati. Inoltre permette al decisore di formulare delle previsioni sulla soluzione e sul valore ottimo in scenari diversi. Ad esempio in un problema di mix produttivo l'analisi di sensitività consente al decisore di individuare quali sono le risorse completamente utilizzate e di prevedere l'aumento della funzione obiettivo quando si rendono disponibili ulteriori quantità di tali risorse. A scopo esemplificativo consideriamo il seguente problema di mix produttivo.

4.2.1 Il problema del caseificio

Un caseificio vuole pianificare la produzione giornaliera di burro, ricotta e mozzarella avendo a disposizione 7 operai e 120 litri di latte. La massima domanda di ciascun prodotto è di 9, 23 e 18 Kg rispettivamente. La quantità di latte richiesta per ogni Kg di prodotto è rispettivamente di 20, 2 e 5 litri. Ciascuno operaio può produrre ognuno dei 3 prodotti e la massima quantità che produrrebbe se si occupasse solamente di uno di essi è rispettivamente di 2 , 7 e 3 Kg. Infine il prezzo di vendita per ogni Kg di prodotto è di 2.5, 1.1 e 2 euro.

1. Formulare un modello AMPL per determinare il piano di produzione che massimizza il guadagno totale.
2. Per quali prodotti la domanda è soddisfatta?
3. Conoscendo i valori delle variabili di slack dei vincoli di domanda che cosa si può dire sui prezzi ombra?
4. Quali sono le risorse sature?
5. A quale massimo costo il caseificio assumerebbe 1 ulteriore operaio?
6. Di quanto dovrebbe aumentare il guadagno unitario del burro per iniziare a produrlo?
7. Il caseificio vuole valutare la possibilità di produrre panna. Sapendo che la produzione di 1Kg di panna richiede 15 litri e che ogni operaio può produrre al massimo una quota di 4 Kg di panna, a partire da quale prezzo conviene iniziare a produrla? Si assuma che non ci siano limiti di domanda.

Per risolvere il primo punto, formuliamo il problema in termini matematici e poi traduciamo questa formulazione nella sintassi di AMPL.

1. FORMULAZIONE.

   - *Indici*: sia i un indice sull'insieme dei prodotti $\{Burro, Ricotta, Mozzarella\}$.
   - *Parametri*:
     - sia d_i la domanda massima per il prodotto i ($d_1 = 9$, $d_2 = 23$, $d_3 = 18$);
     - sia l_i la quantità di latte richiesta per produrre 1 Kg del prodotto i ($l_1 = 20$, $l_2 = 2$, $l_3 = 5$);
     - sia v_i il prezzo di vendita per il prodotto i ($v_1 = 2.5$, $v_2 = 1.1$, $v_3 = 2$);
     - sia q_i la massima quantità del prodotto i fabbricata da ciascun operaio ($q_1 = 2$, $q_2 = 7$, $q_3 = 3$);
     - sia N il numero di operai disponibili ($N = 7$);
     - sia L la quantità totale di latte disponibile ($L = 120$);
   - *Variabili*: sia x_i la quantità di prodotto i realizzata. I limiti sulle variabili sono $x_i \geq 0$.
   - *Funzione obiettivo*:
     $$\max \sum_i v_i x_i$$
   - *Vincoli*:
     (a) (domanda): per ogni i, $x_i \leq d_i$ (3 vincoli);
     (b) (disponibilità latte): $\sum_i l_i x_i \leq L$ (1 vincolo);
     (c) (disponibilità operai): $\sum_i \frac{x_i}{q_i} \leq N$ (1 vincolo).

   MODELLO DI AMPL.

```
#Caseificio.mod - File del modello AMPL - Caseificio

set PROD; # insieme dei prodotti

param d{PROD}>=0; # domanda massima per i prodotti

param l{PROD}>=0;
 # quantita' di latte richiesta per ogni Kg di prodotto

param v{PROD}>=0; # prezzo di vendita dei prodotti

param q{PROD}>=0;
 # massima quantita' di ciascun prodotto fabbricata da 1 operaio
```

```
param N; # numero di operai disponibili

param L; # quantita' di latte disponibile

var x{PROD} >=0; # Kg prodotti per ogni tipo di prodotto

maximize guadagno_totale: sum{i in PROD} v[i]*x[i];

subject to domanda{i in PROD}: x[i]<=d[i]; # vincolo di domanda

subject to disp_latte: sum{i in PROD} l[i]*x[i]<= L;
 # vincolo sulla disponibilita' del latte

subject to disp_operai: sum{i in PROD} x[i]/q[i]<=N;
  # vincolo sulla disponibilita' degli operai
```

Dati di AMPL.

```
#Caseificio.dat - File del dati - Caseificio

set PROD:= Burro, Ricotta, Mozzarella;

 param:       d   l   v    q :=

 Burro        9  20  2.5   2
 Ricotta     23   2  1.1   7
 Mozzarella  18   5  2     3 ;

param N:=7;

param L:=120;
```

Soluzione di AMPL.

I comandi da dare a AMPL per generare la soluzione di questo problema sono:

```
model caseificio.mod; data caseificio.dat; option solver cplex;
solve; display guadagno_totale; display x; quit;
```

E la soluzione ottenuta è:

```
CPLEX 8.0.0: optimal solution; objective 47.58571429
 x [*] :=
     Burro   0
Mozzarella  11.1429
   Ricotta  23
;
```

2. Per rispondere alla seconda domanda occorre visualizzare il valore delle variabili di slack dei vincoli di domanda. A tale scopo digitiamo il comando

```
display domanda.slack;
```

ottenendo i valori:

```
domanda.slack [*] :=
     Burro  9
Mozzarella  6.85714
   Ricotta  0
;
```

La domanda è soddisfatta solo per la ricotta dato che la ricotta è l'unico prodotto che ha la variabile di slack del corrispondente vincolo di domanda con valore 0.

3. Dal teorema degli scarti complementari sappiamo che per ogni vincolo il prodotto della variabile di slack per la corrispondente variabile duale, cioè prezzo ombra, deve essere nullo. Perciò dai valori delle variabili di slack dei vincoli di domanda calcolati al punto precedente possiamo dedurre che il prezzo ombra del burro e della mozzarella devono valere 0 mentre il prezzo ombra della ricotta può essere anche diverso da 0. Questo risultato può essere anche interpretato in chiave economica. Infatti i prezzi ombra di un vincolo rappresentano la variazione della funzione obiettivo in corrispondenza di una variazione infinitesima del termine noto e perciò nel nostro caso l'aumento del guadagno totale in corrispondenza di un aumento infinitesimo della domanda di ciascun prodotto. Dal momento che la domanda di burro e di ricotta è maggiore dei loro livelli di produzione la soluzione ottima e il valore ottimo non cambiano se aumentiamo la domanda di questi prodotti di qualunque quantità e quindi i prezzi ombra corrispondenti devono essere nulli.

4. Per determinare quali sono le risorse completamente utilizzate dobbiamo visualizzare i valori delle variabili di slack del vincolo di disponibilità del latte e del vincolo di disponibilità degli operai. Attraverso i comandi

   ```
   display disp_latte.slack;
   display disp_operai.slack;
   ```

   otteniamo i valori:

   ```
   disp_latte.slack = 18.2857
   disp_operai.slack = 0
   ```

   e poichè la sola variabile di slack nulla è quella degli operai possiamo concludere che l'unica risorsa completamente utilizzata è quella della disponibilità di operai.

5. Per stabilire a quale massimo prezzo il caseificio sarebbe disposto ad assumere un ulteriore operaio occorre chiedere ad AMPL il valore del prezzo ombra del vincolo di disponibilità degli operai. Infatti il prezzo ombra di

un vincolo indica la variazione della funzione obiettivo in corrispondenza di una variazione infinitesima del termine noto di quel vincolo. Attraverso il comando

```
display disp_operai;
```

otteniamo il valore del prezzi ombra:

```
disp_operai = 6
```

e perciò l'azienda è disposta ad assumere un ulteriore operaio al massimo a 6 euro.

6. Attualmente il burro non viene prodotto e per poterlo iniziare a produrre occorre aumentare il suo guadagno unitario dell'opposto del valore del costo ridotto del livello di produzione del burro. Infatti in un problema di massimo (minimo) il costo ridotto di una variabile fuori base indica la diminuzione (aumento) della funzione obiettivo quando tale variabile aumenta di una quantità infintesima. Tramite il comando

   ```
   display x.rc;
   ```

   si ottiene

   ```
   x.rc [*] :=
        Burro  -0.5
   Mozzarella   0
      Ricotta   0
   ;
   ```

   Perciò se forzassimo il caseificio a produrre una unità di burro il guadagno diminuirebbe di 0.5 euro e quindi affinchè la produzione di burro sia conveniente occorre che il guadagno unitario del burro aumenti di almeno 0.5 euro.

7. Per rispondere a questa ultima domanda basta introdurre una variabile associata al nuovo prodotto e imporre che il suo costo ridotto sia nullo. I costi ridotti si calcolano con la relazione che li lega ai costi e ai prezzi ombra:

   $$r_j = c_j - \sum_i \lambda_i a_{ij}$$

   dove r_j indica la j-esima componente del vettore dei costi ridotti, cj il j-esimo coefficiente della funzione obiettivo, λ_i il prezzo ombra dell'i-esimo vincolo e a_{ij} l'assorbimento della risorsa i da parte del prodotto j (elemento (i,j) nella matrice dei vincoli).

   Nel nostro caso, $r4 = c4 - \sum_i \lambda_i a_{i4}$, cioè $0 = c4 - 6/4$. Quindi $c4 = 1.5$ Euro.

4.3 Salvare i risultati

Per salvare su file di testo i risultati, basta far seguire all'istruzione il simbolo `>` o `»` e il nome del file. Il simbolo > costruisce un file contenente quel solo valore (eventualmente sovrascrivendolo, se il file esiste già); il simbolo » apre il file o lo crea e vi aggiunge in fondo i risultati.

```
ampl: display {Cibi} > Insiemi.out;
ampl: display {Nutrienti} >> Insiemi.out;
```

4.4 Modificare un modello

Operando da linea di comando, è possibile cambiare il modello senza modificare il file. L'applicazione tipica è sperimentare per verificare la correttezza del modello o studiarne le proprietà. Si tratta essenzialmente di rilassare vincoli o fissare variabili.

Per rilassare un vincolo, si usa la parola chiave **drop** seguita dal nome del vincolo e dal punto e virgola.

```
drop vincolo;
drop Fabbisogno[Calcio];
```

È possibile rilassare interi vettori di vincoli oppure una selezione ottenuta con espressioni di indicizzazione.

```
drop Fabbisogno;
drop Fabbisogno{i in Nutrienti: Richiesta >= 2};
```

Per riattivare il vincolo, basta eseguire il comando **restore** con la stessa sintassi.

```
restore vincolo;
restore Fabbisogno{i in Nutrienti: Richiesta >= 2};
```

Si può fissare una variabile al suo valore corrente attraverso la parola chiave **fix**, seguita dal nome della variabile oppure da un'espressione indicizzante che determina un sottoinsieme di variabili da fissare e dal loro nome.

```
var x := 10;
var Quantita{j in Cibi} := QuantitaIniz[j];

fix x;
fix {j in Cibi: Costi[j] >= 10} Quantita[j];
```

Per rendere di nuovo libera la variabile, basta eseguire il comando **unfix**.

Appendice A

Uso di AMPL da linea di comando

Si può eseguire il software AMPL anche da linea di comando, senza impiegare l'interfaccia grafica. La casa produttrice ha, anzi, interrotto la fornitura dell'interfaccia stessa.

A.1 Caricare ed eseguire modelli

Per raggiungere l'ambiente a linea di comando, è sufficiente aprire una finestra DOS, portarsi nella directory dove AMPL è installato e scrivere il comando **ampl** al prompt di sistema. Il programma si avvia e risponde con il prompt

```
ampl:
```

in attesa di istruzioni. È possibile accedere a questa modalità anche dall'interfaccia grafica, aprendo la finestra *Commands*, che presenta lo stesso prompt.

Al prompt si può digitare qualsiasi istruzione AMPL: il software la eseguirà quando si sarà premuto il tasto di invio. Qualora si prema il tasto senza aver scritto un'istruzione completa (terminata dal punto e virgola), il software modifica il prompt e attende il completamento

```
ampl: sum{j in Cibi}
ampl? Costo[j];
ampl:
```

Viceversa, è possibile scrivere più istruzioni in sequenza sulla stessa linea, separandole con punti e virgole: il software le esegue in sequenza appena si preme il tasto di invio.

```
ampl: model diet.mod; data diet.dat; solve;
```

Per eseguire un modello conservato nei file **Diet.mod** (modello) e **Diet.dat** (dati), si esegue la stessa sequenza di operazioni descritta nel capitolo sull'interfaccia:

1. si carica il file del modello, con l'istruzione **model** seguita dal nome del file di modello

2. si carica il file dei dati, con l'istruzione **data** seguita dal nome del file dei dati

3. si lancia il risolutore, con l'istruzione **solve** (il software risponde indicando il risolutore impiegato, alcuni dettagli sul procedimento e il valore della soluzione)

Si può costruire un problema caricando diversi file di modello e di dati con le istruzioni **model** e **data**, che vengono aggiunti via via al modello corrente. Si possono anche scrivere direttamente dal prompt parti del modello. Le istruzioni vengono aggiunte sempre al modello: per aggiungerle ai dati, bisogna farle precedere dal comando **data** (senza indicazioni di file).

Il comando **objective** permette di indicare quale obiettivo fra quelli definiti in precedenza il risolutore deve ottimizzare, una volta lanciato.

Lanciato il risolutore, AMPL determina se i dati rispettano le istruzioni di controllo e segnala eventuali violazioni. Quindi, esegue una fase di *presolve*, nella quale individua le variabili fissate da condizioni stringenti e rafforza le restrizioni imposte dall'utente combinandole fra loro. Il risultato è in generale un problema ridotto, oppure la dimostrazione che il problema non ha soluzioni ammissibili. Infine, parte il risolutore vero e proprio, che può essere scelto attraverso l'istruzione **option solver** seguita dal nome e dal punto e virgola. Se si scrive solo **option solver;**, AMPL risponde con il nome del risolutore corrente.

Per abbandonare AMPL, si scrive al prompt **end** o **quit**.

Bibliografia

[1] R. Fourer, D. M. Gay and B. W. Kernighan. *AMPL: A Modeling Language For Mathematical Programming.* Boyd & Fraser Publishing Company, Danvers, Massachussets, (1999)

[2] Brunetta, D'Amico and Luzzi *Appunti sul linguaggio di programmazione MPL.* Dispense per uso interno, (2000)

[3] C. Vercellis, *Modelli e Decisioni.* Esculapio, Bologna, (1997).