Ricerca Operativa - Laboratorio

Lezione 6 - Comandi avanzati e script in AMPL

Docente: Luigi De Giovanni

Dipartimento di Matematica “Tullio Levi-Civita”
Università degli Studi di Padova

luigi@math.unipd.it
https://www.math.unipd.it/∼luigi/

Corso di Laurea Magistrale in Matematica
Università degli Studi di Padova
a.a. 2019–2020

Parametri random (i)

Se in un modello alcune quantità non sono sono deterministiche, ma si conosce la distribuzione di probabilità che le caratterizza, può essere necessario definire dei parametri random, cioè generati in accordo ad opportune distribuzioni di probabilità.

Parametri random (i)

Se in un modello alcune quantità non sono sono deterministiche, ma si conosce la distribuzione di probabilità che le caratterizza, può essere necessario definire dei parametri random, cioè generati in accordo ad opportune distribuzioni di probabilità.

AMPL dispone di funzioni built-in, che generano osservazioni estratte da distribuzioni di probabilità. Per esempio:

```
param media > 0;
param varianza > 0;
param p = Normal(media,varianza);
```

Parametri random (ii)

È possibile utilizzare diverse funzioni di distribuzione:

Beta(a,b)	distribuzione beta con parametri a e b
Cauchy()	distribuzione di Cauchy
Exponential()	distribuzione esponenziale
Gamma(a)	distribuzione Gamma con parametro a
Normal(m,v)	distribuzione normale con media m e varianza v
Poisson(a)	distribuzione di Poisson con parametro a
Uniform(a,b)	distribuzione uniforme sull'intervallo $[a, b)$

Parametri random (ii)

È possibile utilizzare diverse funzioni di distribuzione:

Beta(a,b)	distribuzione beta con parametri a e b
Cauchy()	distribuzione di Cauchy
Exponential()	distribuzione esponenziale
Gamma(a)	distribuzione Gamma con parametro a
Normal(m,v)	distribuzione normale con media m e varianza v
Poisson(a)	distribuzione di Poisson con parametro a
Uniform(a,b)	distribuzione uniforme sull'intervallo $[a, b)$

La generazione di numeri pseudo–casuali dipende da un seme (seed) iniziale da assegnare al generatore. Con i comandi

```
option randseed;
option randseed valore;
```

si visualizza il seme utilizzato e si assegna al seme un dato valore (se valore>0) o un valore scelto in base al system clock (se valore=0).

Parametri random (iii)

Esempio 6.1

Siano dati $n = 200$ investimenti, ciascuno di costo c_i e ritorno r_i, $i = 1, \ldots, n$. Si chiede di selezionare un sottoinsieme di investimenti in modo tale che il loro prezzo totale non ecceda $b = 600$ (budget) e il ritorno sia massimizzato. I costi sono generati con distribuzione uniforme nell'intervallo $[5, 20)$ e i ritorni con distribuzione uniforme nell'intervallo $[3, 10)$.

Parametri random (iv)

Variabili: x_i è una variabile binaria che indica se l'investimento i è selezionato o meno, $i = 1, \ldots, n$.

$$\max \sum_{i=1}^{n} r_i x_i$$

$$\sum_{i=1}^{n} c_i x_i \leq b$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n$$

Parametri random (v)

investimenti.mod

```
param N;
param r{1..N} := Uniform(5,20);
param c{1..N} := Uniform(3,10);
param b;

var x{1..N} binary;

maximize f : sum{i in 1..N} r[i]*x[i];

s.t. v_budget : sum{i in 1..N} c[i]*x[i] <= b;
```

Parametri random (vi)

investimenti.dat

```
param N = 200;
param b := 600;
```

investimenti.run

```
reset;
model investimenti.mod;
data investimenti.dat;

option solver cplex;
solve;

display f;
display x;
```

Script in AMPL

Gli script sono sequenze di istruzioni memorizzate in un file `.run` che vengono eseguite una dopo l'altra e risultano molto utili per implementare l'esecuzione sequenziale di operazioni.

Script in AMPL

Gli script sono sequenze di istruzioni memorizzate in un file `.run` che vengono eseguite una dopo l'altra e risultano molto utili per implementare l'esecuzione sequenziale di operazioni.

Gli script in AMPL possono includere comandi che permettono di generare cicli e/o ripetizioni di istruzioni basandosi sul soddisfacimento o meno di condizioni logiche:

- for
- repeat while
- repeat until
- if then else

Istruzione `for` (i)

L'istruzione `for` permette di ripetere una sequenza di operazioni una volta per ogni membro di un insieme, individuato tramite *un'espressione indicizzante*. La sintassi è la seguente:

```
for {elemento in INSIEME}{
      istruzione1;
      istruzione2;
      istruzione3;
            .
            .
            .
}
```

Istruzione `for` (ii)

Esempio:

```
set INSIEME = {1..10};
var x{INSIEME};
```

Invece di scrivere:

```
display x[1];
display x[2];
display x[3];
  .
  .
  .
```

si può scrivere in maniera molto più sintetica

```
for {i in INSIEME}{
     display x[i];
}
```

Istruzione `for` (iii)

L'istruzione `for` può essere utilizzata per risolvere più volte un problema cambiando ogni volta uno o più dati.

Istruzione `for` (iii)

L'istruzione `for` può essere utilizzata per risolvere più volte un problema cambiando ogni volta uno o più dati.

Supponiamo di avere un modello di ripartizione di risorse nel quale sono definite delle disponibilità massime di ciascuna delle risorse. In questo caso si potrebbe scrivere uno script di questo tipo:

```
for {i in 1..10}{
     let disp["risorsa1"] := disp["risorsa1"] + 1;
     solve;
     display i, profitto;
}
```

In questo modo il problema viene risolto dieci volte aumentando ogni volta di una unità la disponibilità di `risorsa1`.

Istruzione `repeat while` e `repeat until` (i)

L'istruzione `repeat` permette di ripetere l'esecuzione di una sequenza di istruzioni fino al verificarsi di una certa condizione logica.

Istruzione `repeat while` e `repeat until` (i)

L'istruzione `repeat` permette di ripetere l'esecuzione di una sequenza di istruzioni fino al verificarsi di una certa condizione logica.

- Il ciclo definito dal `while` viene ripetuto in presenza di una condizione logica vera

Istruzione `repeat while` e `repeat until` (i)

L'istruzione `repeat` permette di ripetere l'esecuzione di una sequenza di istruzioni fino al verificarsi di una certa condizione logica.

- Il ciclo definito dal `while` viene ripetuto in presenza di una condizione logica vera
- Il ciclo definito dall'`until` viene ripetuto in presenza di una condizione logica falsa

Istruzione `repeat while` e `repeat until` (i)

L'istruzione `repeat` permette di ripetere l'esecuzione di una sequenza di istruzioni fino al verificarsi di una certa condizione logica.

- Il ciclo definito dal `while` viene ripetuto in presenza di una condizione logica vera
- Il ciclo definito dall'`until` viene ripetuto in presenza di una condizione logica falsa
- Il controllo della condizione può essere all'inizio o alla fine del ciclo

Istruzioni `repeat while` e `repeat until` (ii)

La sintassi del `repeat while` è la seguente:

```
repeat while (condizione_logica) {
     istruzioni
}
```

oppure

```
repeat {
     istruzioni
} while (condizione_logica)
```

Istruzioni `repeat while` e `repeat until` (iii)

La sintassi del `repeat until` è analoga:

```
repeat until (condizione_logica) {
     istruzioni
}
```

oppure

```
repeat {
     istruzioni
} until (condizione_logica)
```

Istruzioni `repeat while` e `repeat until` (iii)

La sintassi del `repeat until` è analoga:

```
repeat until (condizione_logica) {
     istruzioni
}
```

oppure

```
repeat {
     istruzioni
} until (condizione_logica)
```

N.B. In entrambi i casi, inserire la verifica della condizione logica prima o dopo le istruzioni corrisponde a flusso di esecuzione differente: ad esempio, se il controllo è all fine, le istruzioni saranno comunque eseguite almeno una volta.

Istruzione `if then else` (i)

L'istruzione `if` permette di valutare una condizione logica ed eseguire delle istruzioni se questa condizione è vera:

```
if (condizione_logica) then {
     istruzioni
}
```

Istruzione `if then else` (i)

L'istruzione `if` permette di valutare una condizione logica ed eseguire delle istruzioni se questa condizione è vera:

```
if (condizione_logica) then {
     istruzioni
}
```

Si usa l'`else` per indicare il verificarsi della condizione alternativa:

```
if (condizione_logica) then {
     istruzioni_1
} else {
     istruzioni_2
}
```

Istruzione `if then else` (ii)

Se possono verificarsi più condizioni:

```
if (condizione_logica_1) then {
     istruzioni_1
} else if (condizione_logica_2) then {
     istruzioni_2
} else if (condizione_logica_3) then {
     istruzioni_3
}
   .
   .
   .
} else {
     istruzioni_n
}
```

Istruzioni `continue` e `break` (i)

L'istruzione `continue` serve per interrompere l'iterazione corrente di un ciclo `for` o `repeat`.
L'istruzione `break` serve per uscire da un ciclo `for` o `repeat`.

Istruzioni `continue` e `break` (i)

L'istruzione `continue` serve per interrompere l'iterazione corrente di un ciclo `for` o `repeat`.
L'istruzione `break` serve per uscire da un ciclo `for` o `repeat`.

- L'istruzione `continue` termina un'iterazione saltando tutte le istruzioni che seguono, portando il controllo all'esecuzione del test da effettuare prima della successiva iterazione

Istruzioni `continue` e `break` (i)

L'istruzione `continue` serve per interrompere l'iterazione corrente di un ciclo `for` o `repeat`.
L'istruzione `break` serve per uscire da un ciclo `for` o `repeat`.

- L'istruzione `continue` termina un'iterazione saltando tutte le istruzioni che seguono, portando il controllo all'esecuzione del test da effettuare prima della successiva iterazione
- L'istruzione `break` termina l'iterazione e esce dal ciclo in maniera definitiva

Istruzioni `continue` e `break` (ii)

```
repeat while (condizione_logica_1) {
      . . .
      if (condizione_logica_2) then {
            continue;
      }
      . . .
}
```

In questo caso le istruzioni che seguono il `continue` non verranno eseguite e il controllo passa alla verifica della `condizione_logica1`

Istruzioni `continue` e `break` (iii)

```
repeat while (condizione_logica_1) {
      . . .
      if (condizione_logica_2) then {
           break;
      }
      . . .
}
istruzione
```

In questo caso il ciclo è definitivamente interrotto e il controllo passa a `istruzione`.

Istruzioni `continue` e `break` (iii)

```
repeat while (condizione_logica_1) {
      . . .
      if (condizione_logica_2) then {
            break;
      }
      . . .
}
istruzione
```

In questo caso il ciclo è definitivamente interrotto e il controllo passa a `istruzione`.

In modo del tutto analogo, `continue` e `break` si utilizzano all'interno di un ciclo `repeat until` o di un ciclo `for`.

Il comando `display`

Il comando `display` permette di eseguire delle stampe a video dei risultati.

- Può visualizzare più elementi con una sola istruzione:

```
var x{I,J};
display x;
```

Il comando `display`

Il comando `display` permette di eseguire delle stampe a video dei risultati.

- Può visualizzare più elementi con una sola istruzione:

```
var x{I,J};
display x;
```

- Può essere utilizzato con espressioni indicizzanti:

```
display {i in I, j in J: x[i,j] > 0 } x[i,j];
```

Il comando `display`

Il comando `display` permette di eseguire delle stampe a video dei risultati.

- Può visualizzare più elementi con una sola istruzione:

```
var x{I,J};
display x;
```

- Può essere utilizzato con espressioni indicizzanti:

```
display {i in I, j in J: x[i,j] > 0 } x[i,j];
```

Il comando `display`

Il comando `display` permette di eseguire delle stampe a video dei risultati.

- Può visualizzare più elementi con una sola istruzione:

```
var x{I,J};
display x;
```

- Può essere utilizzato con espressioni indicizzanti:

```
display {i in I, j in J: x[i,j] > 0 } x[i,j];
```

- Il formato di scrittura è rigido e non è possibile inserire frasi di commento nella visualizzazione.

Il comando `printf` (i)

In alternativa a `display` si può utilizzare il comando `printf`, nel formato:

```
printf <stringa di controllo formato> ,
                <elenco variabili da visualizzare>
```

Esempio:

```
printf "Profitto = %d (migliora del %f\%)\n" , prof, migl;
```

Il comando `printf` (i)

In alternativa a `display` si può utilizzare il comando `printf`, nel formato:

```
printf <stringa di controllo formato> ,
                <elenco variabili da visualizzare>
```

Esempio:

```
printf "Profitto = %d (migliora del %f\%)\n" , prof, migl;
```

La stringa di controllo è racchiusa tra apici e contiene:

- caratteri da visualizzare
- carattere speciale `%` seguito da una specifica di formato
- caratteri speciali preceduti da `\`

Il comando `printf` (i)

In alternativa a `display` si può utilizzare il comando `printf`, nel formato:

```
printf <stringa di controllo formato> ,
                <elenco variabili da visualizzare>
```

Esempio:

```
printf "Profitto = %d (migliora del %f\%)\n" , prof, migl;
```

La stringa di controllo è racchiusa tra apici e contiene:

- caratteri da visualizzare
- carattere speciale `%` seguito da una specifica di formato
- caratteri speciali preceduti da `\`

L'elenco delle variabili contiene, separate da virgole, le variabili da visualizzare nell'ordine specificato all'interno della stringa di controllo.

Il comando `printf` (ii)

Alcuni caratteri speciali:

- `\n` inserisce un a capo (va a capo nella riga successiva)
- `\%` inserisce un simbolo `%`
- `\\` inserisce un simbolo `\`

Il comando `printf` (ii)

Alcuni caratteri speciali:

- `\n` inserisce un a capo (va a capo nella riga successiva)
- `\%` inserisce un simbolo `%`
- `\\` inserisce un simbolo `\`

Alcune possibili specifiche di formato sono:

- `%d` per il formato numerico intero
- `%f` per il formato floating point (con numeri decimali)
- `%e` per il formato floating point con notazione esponenziale
- `%s` per il formato stringa.

Esempi (i)

Esempio 6.2

Con riferimento all'Esempio 3.2 (produzione di parquet), creare uno script che permetta di calcolare il profitto al variare delle ore disponibili del settore falegnameria. Le ore possono variare da un minimo di 2 000 ad un massimo di 2 100 (ad intervalli di 10 ore). Si vuole ottenere, per ogni variazione, un messaggio che comunichi se il corrispondente profitto è accettabile (maggiore di 5450) o meno. Interrompere il ciclo appena si trova un profitto accettabile.

Esempi (ii)

parquet2.run

```
reset;

model parquet.mod;
data parquet.dat;

option solver cplex;

for {i in 2000..2100 by 10} {
    let max_ore['Faleg'] := i;
    printf "\nOre reparto falegnameria = %d \n",
             max_ore['Faleg'];
    solve;
    if (f > 5450) then {
        printf "Profitto accettabile (f = %d) \n", f;
        break;
    } else {
        printf "Profitto non accettabile (f = %d) \n", f;
    }
}
```

Esempi (iii)

Esempio 6.3

Con riferimento all'Esempio 6.1 (investimenti), creare uno script che permetta di calcolare il ritorno atteso sulla base di 100 prove eseguite con parametri casuali diversi.

Suggerimento: usare

```
option solver_msg 0
```

per evitare la visualizzazione dei messaggi del solver, e

```
option solver_msg 1
```

per riabilitare la visualizzazione.

Esempi (iv)

investimenti2.run

```
reset;
model investimenti.mod;
data investimenti.dat;
option solver cplex;

option solver_msg 0;
option randseed 0;

param num_prove = 10;
param ritornoTotale;
let ritornoTotale := 0;
for {i in 1..num_prove} {
    printf "Istanza %d \n" , i;
    solve;
    let ritornoTotale := ritornoTotale + f;
}
printf "\nritorno medio = %20.3f\n" , ritornoTotale / num_prove;

option solver_msg 1;
```

Esercizi proposti (i)

Esercizio 6.1 (i)

Un'azienda automobilistica produce tre diversi modelli di autovettura: un modello economico, uno normale ed uno di lusso. Ogni autovettura viene lavorata da tre robot: **A**, **B** e **C**. I tempi necessari alla lavorazione sono riportati, in minuti, nella tabella seguente insieme al profitto netto in euro realizzato per autovettura:

	Economica	**Normale**	**Lusso**
A	20	30	62
B	31	42	51
C	16	81	10
Prezzo	1000	1500	2200

Esercizi proposti (ii)

Esercizio 6.1 (ii)

I robot **A** e **B** sono disponibili per 8 ore al giorno, mentre il robot **C** è disponibile per 5 ore al giorno. Il numero di autovetture di lusso prodotte non deve superare il 20% del totale mentre il numero di autovetture economiche deve costituire almeno il 40% della produzione complessiva. Se si produce più del 50% di autovetture economiche, allora c'è da pagare una penale pari a 10 000 euro. Supponendo che tutte le autovetture vengano vendute, formulare un problema di Programmazione Lineare che permetta di decidere le quantità giornaliere da produrre per ciascun modello in modo tale da massimizzare i profitti rispettando i vincoli di produzione. Inoltre, stabilire per quali valori compresi tra la metà e il doppio del valore base (a intervalli di 1000 euro) si preferisce pagare la penale.

Suggerimento: usare espressioni del tipo

```
param auto_min symbolic in SET_AUTO
```

per indicare i modelli sottoposti a restrizioni fuori dal `.mod`.

Esercizi proposti (iii)

$$\max\ 1000x_1 + 1500x_2 + 2200x_3 - 10000\delta$$
$$20x_1 + 30x_2 + 62x_3 \leq 480$$
$$31x_1 + 42x_2 + 51x_3 \leq 480$$
$$16x_1 + 81x_2 + 10x_3 \leq 300$$
$$x_3 \leq 0.2(x_1 + x_2 + x_3)$$
$$x_1 \geq 0.4(x_1 + x_2 + x_3)$$
$$x_1 - 0.5(x_1 + x_2 + x_3) \leq M\delta$$
$$x_i \geq 0, \quad i = 1, \ldots, 3$$
$$\delta \in \{0, 1\}$$

Esercizi proposti (iv)

autovetture.mod

```
set ROBOT;
set AUTOVETTURE;

param prezzo{AUTOVETTURE} >= 0;
param disp_max{ROBOT} >= 0;
param rich_tempo{ROBOT,AUTOVETTURE} >= 0 , integer;

param perc_max >= 0;
param auto_max symbolic in AUTOVETTURE;
param perc_min >= 0;
param auto_min symbolic in AUTOVETTURE;
param perc_penale >= 0;
param penale >= 0;
param big_M := 10^6;

var x{AUTOVETTURE} >=0, integer;
var delta binary;
```

Esercizi proposti (v)

```
maximize profitto: sum{j in AUTOVETTURE}
prezzo[j]*x[j] - penale*delta;

s.t. v_robot {i in ROBOT} : sum{j in AUTOVETTURE}
rich_tempo[i,j]*x[j] <= disp_max[i];

s.t. v_max : x[auto_max] <= perc_max *
                  (sum{j in AUTOVETTURE} x[j]);

s.t. v_min : x[auto_min] >= perc_min *
                  (sum{j in AUTOVETTURE} x[j]);

s.t. v_penale : x[auto_min] <= perc_penale *
                   sum{j in AUTOVETTURE} x[j] + big_M*delta;
```

Esercizi proposti (vi)

autovetture.dat

```
set AUTOVETTURE := economica normale lusso;
set ROBOT := A B C;

param:    prezzo :=
economica    1000
normale      1500
lusso        2200;

param: disp_max :=
  A          480
  B          480
  C          300;
```

Esercizi proposti (vii)

```
param rich_tempo :  economica    normale      lusso :=
            A           20          30          62
            B           31          42          51
            C           16          81          10;

param perc_max := 0.2;
param auto_max := 'lusso';
param perc_min := 0.4;
param auto_min := 'economica';
param penale := 10000;
param perc_penale := 0.5;
```

Esercizi proposti (viii)

autovetture.run

```
reset;
model autovetture.mod; data autovetture.dat;
option solver cplexamp;
solve; display profitto, x, delta;

option solver_msg 0;
param max_penale;
let max_penale := 2*penale;
let penale := 0.5*penale;
repeat {
  solve;
  printf "la penale %f %s e il profitto netto è %f\n",
      penale,
      if (delta) then "si paga" else "NON si paga",
      profitto;
  let penale := penale + 1000;
} until ( penale > max_penale );
option solver_msg 1;
```