

# Appunti delle Esercitazioni di Ottimizzazione V.O. AMPL: A Mathematical Programming Language

a cura di G. Liuzzi\*and V. Piccialli†

a.a. 2004-2005

## 1 Introduzione ad AMPL

AMPL(scaricabile all'indirizzo: <http://www.ampl.com/DOWNLOADS/index.html>) è un linguaggio di modellazione per la programmazione matematica. Serve ad esprimere un problema di ottimizzazione in una forma che sia comprensibile da un generico solutore. È un linguaggio algebrico, cioè contiene diverse primitive per esprimere la notazione matematica normalmente utilizzata nello scrivere problemi di ottimizzazione quali sommatorie, funzioni matematiche elementari, ecc. Ciascuna istruzione di AMPL deve terminare con un ';'. Questo vuol dire che, nello scrivere una istruzione, possiamo inserire tra le parole chiave del linguaggio quanti spazi e ritorni a capo vogliamo senza per questo generare errori. Spesso questa libertà di scrittura viene sfruttata per indentare il file dei comandi in modo da renderne più agevole la lettura. Per cui, benché scrivere

```
var x1; var x2; minimize obiettivo: x1+x2; subject to vincolo1: x1 >= 0;
subject to vincolo2: x2 >=0;subject to vincolo3: x1 <= 10;
subject to vincolo4: x2 <= 10; s.t. vincolo3: x1-x2 <= 0;
```

e

```
var x1;
var x2;

minimize obiettivo: x1+x2;

subject to vincolo1: x1 >= 0;
subject to vincolo2: x2 >=0;
subject to vincolo3: x1 <= 10;
subject to vincolo4: x2 <= 10;
s.t.          vincolo3: x1-x2 <= 0;
```

abbiano, sintatticamente, lo stesso significato, il secondo formato è certamente più leggibile del primo.

Sebbene AMPL consenta di scrivere in un unico file (con estensione `.mod`) un problema e farlo quindi risolvere al solutore, concettualmente è sempre meglio tenere ben separati il file di “modello”, in cui è descritta la struttura logica del modello del problema in esame, dal file dei “dati”, in cui invece sono scritti i valori numerici del problema stesso. Per uno stesso modello, i dati possono essere contenuti in uno o più file `.dat`. In questo modo, mantenendo

---

\*liuzzi@dis.uniroma1.it, <http://www.dis.uniroma1.it/~liuzzi>

†piccialli@dis.uniroma1.it

cioè fisicamente separati il modello dai suoi dati, è possibile cambiare i dati modificando solo il file relativo senza quindi il pericolo di introdurre errori nel modello. Il file di modello ha obbligatoriamente estensione `.mod`, quello di dati obbligatoriamente estensione `.dat`.

## 2 Esempio di modello di Programmazione Non Lineare

Il problema considerato è il seguente:

Un'industria chimica intende utilizzare della lamiera metallica residua, costruendo un serbatoio scoperto da adibire all'immagazzinamento di un prodotto liquido. La lamiera può essere tagliata e saldata a piacere, è disponibile per complessivi  $150m^2$  e la si vuole utilizzare tutta. Il serbatoio deve essere contenuto in un capannone a pianta quadrata, con lato di  $10m$ , e con tetto spiovente dall'altezza di  $4.5m$  a  $3m$ . Per semplicità di progetto, si assume che il serbatoio abbia la forma di un prisma retto, con base quadrata.

Determiniamo le dimensioni del serbatoio, in modo da massimizzare il volume del liquido che vi può essere contenuto.

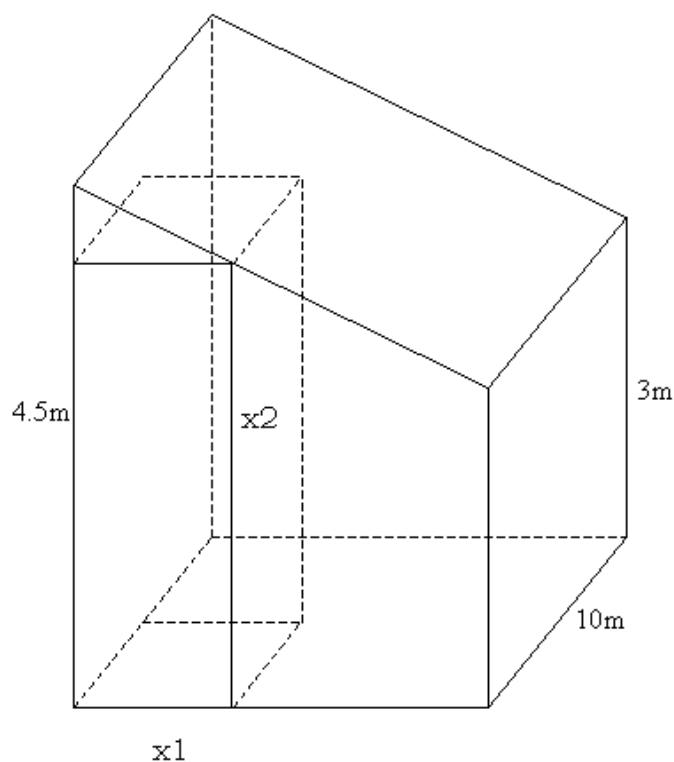


Figura 1: Serbatoio

## 2.1 Variabili di decisione

Le variabili di decisione sono  $x_1$  la misura del lato di base del serbatoio e  $x_2$  la misura dell'altezza. In AMPL le variabili si dichiarano con la parola chiave **var**. Tutte le variabili devono essere dichiarate prima di poter essere utilizzate. La più semplice dichiarazione di variabili è la seguente, che si adatta al nostro caso:

```
var x1;  
var x2;
```

E' possibile assegnare dei valori iniziali alle variabili nel seguente modo:

```
var x1:=5;  
var x2:=2;
```

Il solutore in questo caso è libero di modificare questi valori e anzi, sperabilmente, li cambierà migliorando il valore della funzione obiettivo.

## 2.2 Funzione obiettivo

Nel nostro esempio si vuole massimizzare il volume del liquido che può essere contenuto nel serbatoio, in forma analitica quindi la nostra funzione obiettivo è data da:

$$V = x_1^2 \times x_2$$

In AMPL la funzione obiettivo è dichiarata con la parola chiave **minimize** se la si vuole minimizzare o **maximize** se la si vuole massimizzare, seguita obbligatoriamente da un nome, dai due punti, e da un'espressione in cui possono comparire solo le variabili già definite. Nel nostro caso quindi, poiché si vuole massimizzare il volume del liquido contenuto nel serbatoio si otterrà :

```
maximize volume: (x1**2)*x2;
```

dove il simbolo \*\* in AMPL indica l'elevamento a potenza e il simbolo \* indica il prodotto.

## 2.3 Vincoli

Per quel che riguarda i vincoli abbiamo nel nostro esempio:

*Vincoli di disponibilità* : deve essere utilizzata esattamente una quantità di lamiera pari a  $150m^2$ . Quindi poiché il serbatoio è scoperto, la quantità di lamiera necessaria è pari all'area di base e alle 4 superfici laterali. Il vincolo risultante è quindi il seguente:

$$x_1^2 + 4x_1 \times x_2 = 150$$

In AMPL i vincoli vengono dichiarati con la parola chiave **subject to**. Questa parola chiave è opzionale, dato che ogni dichiarazione che non inizia con una parola chiave viene considerata un vincolo, e può essere abbreviata come **subj to** o **s.t.**. I vincoli devono obbligatoriamente avere un nome, seguito dai due punti, e una espressione in cui possono comparire solo le variabili già definite. Deve inoltre comparire un operatore di relazione ( $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ,  $=$ ). Il vincolo precedente quindi in AMPL viene così definito:

```
subject to lamiera: x1**2+4x1*x2=150;
```

*Vincoli di spazio*: il serbatoio deve essere collocato nel capannone, quindi si deve avere

$$x_1 \leq 10$$

Questo vincolo può essere scritto in due modi: o come per il vincolo di disponibilità

```
subject to base: x1 <= 10;
```

oppure può essere specificato come restrizione nella dichiarazione della variabile  $x_1$

```
var x1 <= 10;
```

Per quanto riguarda  $x_2$ , poichè l'altezza del capannone è variabile da 3 a 4.5m e il lato è 10m abbiamo:

$$x_2 \leq -0.15 \times x_1 + 4.5$$

Questo vincolo quindi si scrive :

```
subject to altezza: x2+0.15*x1 <= 4.5;
```

*Vincoli di non negatività* : Essendo  $x_1$  e  $x_2$  lunghezze si deve avere:

$$x_1 \geq 0, \quad x_2 \geq 0$$

In AMPL si possono scrivere i due vincoli corrispondenti nel seguente modo:

```
subject to nonneg1: x1 >= 0;
subject to nonneg2: x2 >= 0;
```

oppure si possono dichiarare le variabili nel seguente modo:

```
var x1 >= 0;
var x2 >= 0;
```

All'interno della dichiarazione della variabile  $x_1$  si possono includere sia il vincolo di non-negatività, che la limitazione superiore espressa dal primo vincolo di spazio, con la seguente sintassi:

```
var x1 >= 0, <=10;
var x2 >= 0;
```

Riassumendo il file che descrive il modello, che chiameremo **serbatoio.mod** è il seguente:

---

```
serbatoio.mod
```

```
var x1 >= 0, <= 10;
var x2 >= 0;

maximize volume: (x1**2)*x2;

subject to lamiera: x1**2+4*x1*x2=150;
subject to altezza: x2+0.15*x1 <= 4.5;
```

---

## 2.4 Separazione del modello dai dati e soluzione

Nello scrivere il problema in AMPL cerchiamo sempre di tenere separati il modello e i dati che al modello si riferiscono. Per far questo si ha bisogno di un modo per rappresentare i dati che compaiono nel modello e il cui valore rimane costante una volta avviato il processo risolutivo. In AMPL i dati sono detti parametri. Per dichiarare un parametro, si usa la parola chiave **param**, seguita dal nome del parametro. La più semplice dichiarazione di parametro è :

```
param T;
```

Un parametro non può essere utilizzato prima della sua dichiarazione, quindi la dichiarazione dei parametri si trova in testa al file .mod.

Il file che descrive il modello astraendosi dai dati nel caso del problema del serbatoio è quindi il seguente:

---

```
serbatoio.mod
```

```
param lato >= 0;
param hmin >= 0;
param hmax >= hmin;
param max_area;

var x1 >= 0,    <= lato;
var x2 >= 0;

maximize volume: (x1^2)*x2;

s.t. vin_h: x2 <= (lato-x1)*(hmax-hmin)/lato + hmin;
s.t. area: x1^2 + 4*(x2*x1) <= max_area;
```

---

I dati del problema vengono invece memorizzati in un file separato, che chiameremo serbatoio.dat. In questo file vengono assegnati dei valori ai parametri del problema (che non possono più essere cambiati in fase di risoluzione) e l'istruzione è costituita dalla parola chiave **param**, seguita dal nome del parametro (lo stesso che è stato dichiarato nel file .mod), dal simbolo := e da un valore.

```
param T:= 1;
```

Il file .dat può anche contenere un valore di assegnazione iniziale delle variabili. Questa assegnazione si effettua con la parola chiave **let**, seguita dal nome della variabile, dal simbolo := e da un valore. Come detto precedentemente, questa assegnazione può essere fatta anche all'interno del file .mod contestualmente alla dichiarazione delle variabili, ma in generale è consigliabile farla nel file .dat. Questo perchè in questo modo non si deve modificare il file .mod se si vuole cambiare i valori iniziali delle variabili.

Quindi il file che assegna i dati del modello è il seguente, dove le ultime due istruzioni sono assegnazioni dei valori iniziali alle variabili :

---

```
serbatoio.dat
```

```
param lato := 10;
param hmin := 3;
param hmax := 4.5;
param max_area := 150;

let x1:= 1;
let x2:= 1;
```

---

Per la soluzione di problemi di programmazione non lineare, esistono diversi solutori(MINOS, SNOPT, LOQO..), mentre per la programmazione lineare si utilizza un solutore specifico chiamato CPLEX.

Per risolvere, ad esempio con il solutore SNOPT, questo problema, al prompt di AMPL diamo le seguenti istruzioni

```
ampl:reset;
ampl: model serbatoio.mod;
```

```

ampl: data serbatoioio.dat;
ampl: option solver snopt;
ampl: solve;

```

dove l'istruzione `reset` serve a cancellare i dati o i modelli già caricati da AMPL ed è obbligatoria se si è già risolto un altro problema o il problema stesso. L'istruzione `model` serve a far caricare il file del modello, l'istruzione `data` a far caricare i dati corrispondenti, l'istruzione `solve` a far risolvere il problema. L'istruzione `option solver snopt` serve a far sì che AMPL utilizzi il solutore SNOPT. L'output prodotto dalla sequenza precedente di istruzioni è il seguente:

```

SNOPT 6.1-1(4)(Jun 2001):
Optimal solution found.
9 iterations, objective 176.7064536
Nonlin evals: obj = 12, grad = 11, constrs = 12, Jac = 11.

```

A questo punto se vogliamo farci dire quanto vale la funzione obiettivo all'ottimo e quali sono i valori ottimi per le due variabili dobbiamo digitare

```

display volume;
display x1,x2;

```

ottenendo

```

volume = 176.706

```

```

x1 = 7.18586
x2 = 3.42212

```

Lo studente controlli se la soluzione trovata utilizza tutta la lamiera disponibile ( $150m^2$ ).

### 3 Esempio di modello di Programmazione Lineare

Il problema considerato è il seguente:

Un'azienda produce tre prodotti: P1, P2, P3. La seguente tabella riporta il prezzo di vendita (in euro) e le ore di lavorazione richieste da ogni unità dei tre prodotti: Il costo di ogni ora

	P1	P2	P3
prezzo	2000	2500	4000
ore di lavorazione	10	12	20

di lavorazione è di 100 euro. La fabbricazione di tre prodotti richiede l'impiego di 4 materie prime (M1, M2, M3, M4), il cui costo (in euro/kg) è : 10 per M1, 15 per M2, 20 per M3, 10 per M4. La seguente tabella mostra i chilogrammi di materie prime richiesti da ogni unità dei tre prodotti: L'azienda può acquistare mensilmente 3000 kg di M1, 2000 kg di M2, 5000

	M1	M2	M3	M4
P1	2	10	4	2
P2	6	20	3	2
P3	7	2	20	15

kg di M3, 6000 kg. di M4. Formulare un problema di PL che massimizzi il profitto mensile.

### 3.1 Modello matematico

In questo caso le variabili di decisione  $x_1, x_2, x_3$  sono le quantità prodotte rispettivamente di P1, P2, P3. Non è necessario introdurre altre perché sia le ore di lavorazione che le materie prime sono proporzionali alle quantità dei tre prodotti. Si avranno inoltre dei vincoli di non negatività su queste variabili, in quanto non si possono produrre quantità negative di prodotti.

Per quel che riguarda la funzione obiettivo, si vuole massimizzare il profitto mensile che è dato dalla differenza tra ricavo e costi. Il ricavo per ogni singolo prodotto è dato dal prodotto del prezzo di vendita per la quantità prodotta. Il ricavo complessivo (in lire) è quindi:

$$2.000 \times x_1 + 2.500 \times x_2 + 4.000 \times x_3$$

I costi si dividono tra costi di lavorazione e costi di materie prime. I primi si ottengono moltiplicando il costo di una singola ora di lavorazione (euro 100) per il numero complessivo di ore necessarie a produrre i tre prodotti:

$$100(10 \times x_1 + 12 \times x_2 + 20 \times x_3)$$

Il costo totale delle materie prime si ottiene invece moltiplicando il costo della singola materia prima per la quantità utilizzata per la produzione dei tre prodotti:

$$10(2 \times x_1 + 6 \times x_2 + 7 \times x_3) + 15(10 \times x_1 + 20 \times x_2 + 2 \times x_3) \\ + 20(4 \times x_1 + 3 \times x_2 + 20 \times x_3) + 10(2 \times x_1 + 2 \times x_2 + 15 \times x_3)$$

La funzione obiettivo da massimizzare è quindi la seguente:

$$2.000 \times x_1 + 2.500 \times x_2 + 4.000 \times x_3 \\ - 100(10 \times x_1 + 12 \times x_2 + 20 \times x_3) - 10(2 \times x_1 + 6 \times x_2 + 7 \times x_3) \\ - 15(10 \times x_1 + 20 \times x_2 + 2 \times x_3) - 20(4 \times x_1 + 3 \times x_2 + 20 \times x_3) \\ - 10(2 \times x_1 + 2 \times x_2 + 15 \times x_3)$$

Passiamo ora ai vincoli; i vincoli che esprimono il fatto che c'è un limite massimo acquistabile mensilmente per ogni materia prima sono i seguenti:

$$(M1) \quad 2 \times x_1 + 6 \times x_2 + 7 \times x_3 \leq 3000 \\ (M2) \quad 10 \times x_1 + 20 \times x_2 + 2 \times x_3 \leq 2000 \\ (M3) \quad 4 \times x_1 + 3 \times x_2 + 20 \times x_3 \leq 5000 \\ (M4) \quad 2 \times x_1 + 2 \times x_2 + 15 \times x_3 \leq 6000$$

### 3.2 Generalizzazione del modello precedente

Il modello precedentemente descritto può essere generalizzato. Supponiamo di avere:

- un insieme di N prodotti  $\{P_1, \dots, P_N\}$  ( non più solo 3 prodotti, ma N prodotti)
- un insieme di T materie prime  $\{M_1, \dots, M_T\}$  ( non più solo 4 materie prime, ma T materie prime) necessarie per realizzare gli N prodotti
- per ogni prodotto  $P_i \in \{P_1, \dots, P_N\}$ , sia  $o_i$  il numero di ore di lavorazione necessario per il prodotto  $P_i$
- $co$  = il costo di una singola ora di lavorazione
- per ogni materia prima  $M_j \in \{M_1, \dots, M_T\}$ , sia  $cm_j$  il costo di acquisto di 1 kg della materia  $M_j$

- per ogni prodotto  $P_i \in \{P_1, \dots, P_N\}$ , sia  $pr\_i$  il prezzo di vendita del prodotto  $P_i$
- per ogni prodotto  $P_i \in \{P_1, \dots, P_N\}$  e per ogni materia prima  $M_j \in \{M_1, \dots, M_T\}$ , sia  $q\_ij$  la quantità di materia prima  $j$  necessaria per produrre un'unità del prodotto  $i$
- per ogni materia prima  $M_j \in \{M_1, \dots, M_T\}$ , sia  $qmax\_j$  la massima quantità della materia prima  $i$  acquistabile mensilmente

La formulazione di questo problema più generale è la seguente:

- le variabili di decisione saranno le quantità prodotte di ciascun prodotto:

$$x\_i = \text{unità di prodotto } i \text{ fabbricate settimanalmente, } i \in \{1, \dots, N\}$$

- la funzione obiettivo sarà data dalla differenza tra il ricavo e i costi:

$$\max\left\{\sum_{i=1}^N (pr\_i \times x\_i) - \sum_{i=1}^N (co \times x\_i) - \sum_{j=1}^T (cm\_j \times \sum_{i=1}^N q\_ij \times x\_i)\right\}$$

- si avranno tanti vincoli quante sono le materie prime per esprimere il limite massimo di acquisto della singola materia prima:

$$\sum_{i=1}^n (q\_ij \times x_i) \leq qmax\_j, \quad j = 1, \dots, T$$

Sono inoltre necessari i vincoli di non negatività delle variabili:

$$x\_i \geq 0 \quad i = 1, \dots, N$$

Vediamo ora come tradurre questo modello in AMPL. Per farlo si ha bisogno di alcune strutture dati molto usate in AMPL, gli *insiemi*. Un insieme deve prima di tutto essere dichiarato, e questo si fa con la parola chiave **set** seguita dal nome dell'insieme (nota: AMPL è case sensitive). Nel nostro caso:

set prodotti;

con cui semplicemente si dice ad AMPL che il nome prodotti rappresenta un generico insieme non meglio specificato. Per tenere separati i dati dal modello si mette la dichiarazione degli insiemi all'interno del file .mod e nel file .dat la loro definizione, cioè l'istruzione con cui si assegnano ad un dato insieme i suoi elementi. La definizione di un insieme si effettua con la parola chiave **set** seguita dal nome dell'insieme (dato nel .mod), dal simbolo **:=** e dagli elementi dell'insieme separati da uno spazio. Ad esempio nel nostro modello si ha nel file .mod:

```
set prodotti:= P1 P2 P3;
```

mentre nel file .dat:

```
set prodotti:= P1 P2 P3;
```

Avremo inoltre bisogno di un insieme di materie prime dichiarato nel file .mod con l'istruzione:

```
set materie_prime;
```

e così definito nel file .dat:

```
set materie_prime:= M1 M2 M3 M4;
```

Oltre agli insiemi si ha bisogno dei *parametri* AMPL, che questa volta sono vettori e matrici di valori. È possibile dichiarare vettori e matrici di parametri con una unica istruzione in cui si dichiara, oltre al nome del parametro, anche l'insieme entro cui varia l'indice, o gli indici, delle sue componenti. Ovviamente bisognerà usare un insieme già dichiarato. Così le istruzioni

```
set C;  
param costi{C};
```

dichiarano *C* come insieme e *costi* come vettore di parametri indicizzati sull'insieme *C*. Quindi, per maggiore chiarezza, il parametro *costi* avrà tante componenti quanti sono gli elementi dell'insieme *C*. Analogamente,

```
param N integer;  
param costi{1..N};
```

definiscono un parametro intero *N* ed un vettore di parametri *costi* con tante componenti quanti sono i numeri interi da 1 ad *N*. È anche possibile far comparire nella dichiarazione di un parametro, non un insieme ma due o più .

```
set VAR;  
set VINC;  
param a{VINC,VAR};
```

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

Ovunque occorra usare una specifica componente del parametro *costi* oppure *a* si dovrà usare la notazione `[]` ovvero

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

in cui 10 deve essere compreso tra 1 ed *N*, e *i* e *j* devono essere elementi rispettivamente di *VINC* e *VAR*.

Contestualmente alla dichiarazione di un parametro è possibile specificarne alcune restrizioni. Per cui

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

restringe il parametro *T* ad essere un numero intero e maggiore di 1.

I parametri del nostro modello sono i costi delle materie prime, i prezzi di vendita dei prodotti, il numero di ore di lavorazione necessarie per ogni prodotto, il costo di ogni materia prima, le quantità di ogni materia prima necessaria per produrre un'unità di ogni prodotto, la massima quantità acquistabile mensilmente di ogni materia prima. I parametri vengono dichiarati nel file `.mod` e definiti nel file `.dat`. In generale un parametro non può essere usato se non viene prima dichiarato. In conclusione avremo la seguente dichiarazione di parametri:

```
param prezzi{prodotti};  
param costi_materie{materie_prime};  
param costo_ora;  
param ore{prodotti};  
param quant_max{materie_prime};  
param quant_nec{prodotti,materie_prime};
```

Una volta dichiarati i parametri si può passare alla dichiarazione delle variabili che sono indicizzate tramite l'insieme dei prodotti:

```
var x{i in prodotti} >=0;
```

Infine la funzione obiettivo e i vincoli (che sono tanti quante sono le materie prime):

```
maximize ricavo: sum{i in prodotti}(prezzi[i]*x[i])
-costo_ora*(sum{i in prodotti}(ore[i]*x[i]))
-sum{j in materie_prime}(costi_materie[j]*
sum{i in prodotti}(quant_nec[i,j]*x[i]));

subject to vinc_max{j in materie_prime}: sum{i in prodotti}
(quant_nec[i,j]*x[i])<=quant_max[j];
```

Notiamo che anche l'insieme dei vincoli è indicizzato sull'insieme delle materie prime, in quanto si hanno tanti vincoli quante sono le materie prime.

Ricapitolando, il file `prodotti.mod` è il seguente:

---

`prodotti.mod`

---

```
set prodotti;
set materie_prime;

param prezzi{prodotti};
param costi_materie{materie_prime};
param costo_ora;
param ore{prodotti};
param quant_max{materie_prime};
param quant_nec{prodotti,materie_prime};

var x{i in prodotti} >=0;

maximize ricavo: sum{i in prodotti}(prezzi[i]*x[i])
-costo_ora*(sum{i in prodotti}(ore[i]*x[i]))
-sum{j in materie_prime}(costi_materie[j]*
sum{i in prodotti}(quant_nec[i,j]*x[i]));

subject to vinc_max{j in materie_prime}: sum{i in prodotti}
(quant_nec[i,j]*x[i])<=quant_max[j];
```

---

Con questo file `.mod` abbiamo un modello generale, che può essere particolarizzato specificando i parametri nel file `.dat`.

Vediamo prima di tutto come si assegnano i valori ai vari tipi di parametri.

Per assegnare valori ad un parametro monodimensionale (vettore), occorre specificare le coppie indice valore. Quindi, avendo dichiarato nel file di modello

```
set indice;
param vettore{indice};
```

nel file dei dati un assegnamento ammissibile potrebbe essere il seguente

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

Per aumentare la leggibilità del file dei dati, conviene indentare l'istruzione `param` in modo da ottenere

```

param: vettore := A 1
          B 3;

```

Nelle due precedenti istruzioni il simbolo `:` che segue la parola chiave `param` è, in realtà, opzionale cioè potremmo non metterlo. Vedremo, tuttavia, che ci sono casi in cui è obbligatorio mettere i `“:”` e casi in cui, invece, è obbligatorio *non* mettere i `“:”`. Per esempio, un caso in cui l’uso dei due punti (`“:”`) dopo la parola `param` è obbligatorio, si ha quando vogliamo assegnare valori a due o più vettori di parametri monodimensionali e indicizzati sullo stesso insieme. Supponiamo per esempio che nel file del modello siano presenti le seguenti istruzioni

```

set indice;
param vett1{indice};
param vett2{indice};

```

Per assegnare valori ai due vettori di parametri AMPL ci offre la possibilità di usare una sola istruzione, e precisamente:

```

set indice := A B;
param: vett1 vett2 :=
  A   1   4
  B   3   7 ;

```

Qui il simbolo `“:”` è obbligatorio perchè serve per avvertire AMPL del fatto che stiamo per definire non un vettore ma due (o più) contemporaneamente.

Per un parametro bidimensionale, il discorso è solo leggermente più complicato. Infatti, avendo dichiarato matrice come

```

set dim1;
set dim2;
param matrice{dim1,dim2};

```

l’istruzione standard per assegnare valori a `matrice` sarebbe la seguente:

```

set dim1 := X Y Z;
set dim2 := A B C;
.
.
param: matrice := X A 1   X B 2   X C 3
                  Y A 3   Y B 1   Y C 2
                  Z A 7   Z B 5   Z C 4;

```

che prevede di specificare tutte le componenti mediante indicazione del primo indice, secondo indice e valore. Facciamo notare che anche questa volta il simbolo `“:”` è opzionale. Questo metodo presenta degli ovvi svantaggi, non ultimo quello di dover ripetere molte volte gli stessi indici. Per questo motivo è prevista anche la più concisa notazione

```

param matrice: A B C :=
  X 1 2 3
  Y 3 1 2
  Z 7 5 4;

```

In pratica, è come se la matrice venisse inserita per colonne. Il comando precedente, dal simbolo `“:”` in poi è esattamente uguale all’assegnazione di valori a tre parametri fittizi aventi il nome delle colonne `A`, `B` e `C` ed indicizzati sullo stesso insieme `dim1`. Per questo motivo possiamo dire che tutto va come se stessimo assegnando valori alla matrice *per colonne*.

Ovviamente, in questo caso, dopo la parola `param` abbiamo dovuto specificare il nome del parametro `matrice`, in modo tale da far capire ad AMPL che stiamo assegnando valori ad un parametro a due dimensioni.

Se una matrice è molto larga e poco alta, e quindi non agevolmente visibile nella schermata, conviene scriverne la *trasposta*, facendo seguire al nome della matrice la parola chiave (`tr`), ottenendo

```
param matrice(tr): X Y Z :=
    A 1 3 7
    B 2 1 5
    C 3 2 4;
```

Abbiamo visto che è possibile definire un parametro con tante componenti quanti sono gli elementi di un insieme semplicemente facendo seguire al nome del parametro, il nome dell'insieme racchiuso tra parentesi graffe. Nel caso in cui si vuole specificare un parametro con tante componenti quante sono le coppie di elementi del prodotto cartesiano di due insiemi, con scrittura analoga, si mettevano tra parentesi graffe i nomi dei due insiemi (o più), separati dalla virgola. Più in generale possiamo sostituire `{dim1,dim2}` con altre così dette *espressioni di indicizzazione*. Qui sotto ne riportiamo alcune:

```
{A}                # tutti gli elementi di A
{A,B}              # tutte le coppie di elementi uno
                  # di A e uno di B
{i in A, j in B}   # come sopra
{i in A, B}        # come sopra
{A, j in B}        # come sopra
{i in A: p[i]>0}   # tutti gli elementi di A tali che
                  # p[i] > 0
{i in A, i in B}   # tutte le coppie di elementi uno
                  # di A e uno di B purché uguali
```

Le precedenti espressioni di indicizzazione vengono usate nella dichiarazioni, oltre che dei parametri, come visto prima, anche degli insiemi, variabili, vincoli e per definire sommatorie e produttorie aritmetiche.

Tornando al nostro problema è in particolare possibile ridefinire il problema da cui eravamo partiti creando il seguente file `.dat`:

---

```
prodotti.dat

set prodotti:= P1 P2 P3;
set materie_prime:=M1 M2 M3 M4;
param: prezzi ore:=
    P1 2000000 10
    P2 2500000 12
    P3 4000000 20;
param: costi_materie quant_max:=
    M1 10000 3000
    M2 15000 2000
    M3 20000 5000
    M4 10000 6000;
param costo_ora:=100000;

param quant_nec: M1 M2 M3 M4 :=
    P1 2 10 4 2
    P2 6 20 3 2
    P3 7 2 20 15;
```

---

Per risolvere il nostro problema le istruzioni da dare ad AMPL sono le seguenti:

```
ampl: reset;
ampl: model prodotti.mod;
ampl: data prodotti.dat;
ampl: option solver cplex;
ampl: solve;
```

dove CPLEX è il solutore che si utilizza per la programmazione lineare. Il risultato che si ottiene è il seguente:

```
CPLEX 8.0.0: optimal solution; objective 409375
0 simplex iterations (0 in phase I)
```

Per vedere i valori della funzione obiettivo e delle variabili all'ottimo le istruzioni sono le seguenti:

```
ampl: display ricavo;
ricavo = 409375000
ampl: display x;
x [*] :=
P1 156.25
P2 0
P3 218.75 ;
```

## 4 Problema di localizzazione

Consideriamo il seguente problema:

Una compagnia petrolifera si rifornisce di greggio in tre città portuali, che indicheremo con A, B, C. Il porto B è ubicato 300Km ad Est e 400Km a Nord del porto A, mentre il porto C è situato 400Km ad Est e 100Km a Sud del porto B. La compagnia intende costruire una nuova raffineria per il greggio, ed intende localizzare questo nuovo impianto in modo tale da minimizzare la quantità totale di tubi occorrenti per collegare la raffineria ai porti A, B e C. Inoltre, per ragioni territoriali, non è possibile situare la raffineria né a Sud del porto A né entro un raggio di 360Km dallo stesso.

Al fine di formulare matematicamente il problema di localizzazione in esame, fissamo un sistema di coordinate cartesiane con origine nel porto A. Pertanto i tre porti avranno, in questo sistema di riferimento, le seguenti coordinate:  $A(0,0)$ ,  $B(3,4)$ ,  $C(7,3)$ . Ora indichiamo con  $x$  e  $y$  le coordinate del punto in cui dislocare la raffineria  $r$ . Un modello matematico che rappresenti il problema in esame è, ad esempio, il seguente

$$\begin{aligned} \min_{x,y} \quad & \sqrt{x^2 + y^2} + \sqrt{(x-3)^2 + (y-4)^2} + \sqrt{(x-7)^2 + (y-3)^2} \\ & x^2 + y^2 \geq 3.6^2 \\ & y \geq 0. \end{aligned} \tag{1}$$

A questo punto passiamo a vedere come trascrivere il modello matematico in un modello AMPL. Un modo rapidissimo per risolvere il problema in AMPL, consiste nello scrivere il seguente file `raffineria.mod`

---

```
raffineria.mod
-----
model;

var x;
var y >= 0;

minimize tubi: sqrt(x**2 + y**2) + sqrt((x-3)**2 + (y-4)**2)
              + sqrt((x-7)**2 + (y-3)**2);

s.t. fuori: x^2 + y^2 >= 3.6^2;

data;

let x:= 100;
let y:= 100;

option solver loqo; #seleziona LOQO come solutore di default
solve;

display tubi;
display x,y;
```

---

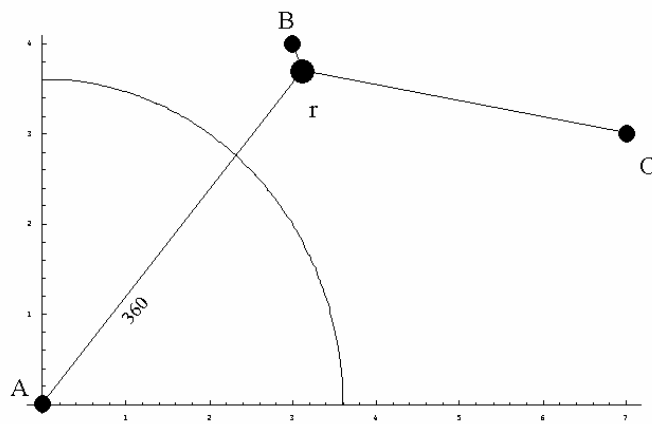
e quindi dare, al prompt di AMPL, i seguenti comandi

```
ampl: reset;
ampl: model raffineria.mod;
```

In questo modo, AMPL invocherà il solutore LOQQ [7] per risolvere il problema, producendo il seguente output a video

```
LOQQ 6.01: optimal solution (48 iterations, 111 evaluations)
primal objective 910.5436033
dual objective 910.5436313
tubi = 910.5436313
```

```
x = 311.097
y = 369.332
```



Quello che abbiamo fatto fin qui è stato di trascrivere brutalmente il modello matematico (1) in istruzioni comprensibili all'interprete di AMPL. Tuttavia, è possibile migliorare un poco il modello AMPL del problema di localizzazione, ad esempio rendendolo il più possibile indipendente dai dati. In particolare, cosa accadrebbe se aggiungessimo un porto D al nostro problema? Attualmente, questa semplice modifica comporterebbe la necessità di modificare il file `raffineria.mod` aggiungendo un termine nella funzione obiettivo in modo da considerare anche la distanza tra il nuovo porto D e la raffineria.

Per evitare tutto ciò, sarebbe auspicabile rendere il più possibile *parametrico* il file di modello. In sostanza quello che si fa è scrivere un file con estensione `.mod` in cui si descrive *astrattamente* il modello matematico, lasciando parametrici tutti i valori particolari per lo specifico problema in esame. In questa operazione siamo facilitati dall'avere a disposizione la formulazione matematica del problema di interesse, che ci dovrebbe guidare nella scrittura di un modello AMPL astratto.

Solamente in seguito, mediante scrittura di un file di dati, con estensione `.dat`, assegneremo i valori specifici per la particolare istanza del problema.

In altri termini, il file di modello rappresenterà una intera classe di problemi per esempio di localizzazione, come in questo caso. Mentre il file dei dati avrà lo scopo di particularizzare il modello astratto ad un caso fissato, una singola istanza del problema.

Secondo quanto appena detto, il problema di allocazione sarà quindi rappresentato dai file di modello e di dati seguenti

---

raffineria.mod

---

```
set PORTI;                                #dichiara PORTI come insieme non
                                           #meglio specificato di elementi

param xcoord{PORTI};
param ycoord{PORTI};
param pA symbolic in PORTI;
param raggio;

var x;
var y;

minimize tubi: sum{i in PORTI}sqrt((x-xcoord[i])**2 +
(y-ycoord[i])**2);

s.t. fuori: (x-xcoord[pA])**2 + (y-ycoord[pA])**2 >= raggio^2;
s.t. nosud: y >= ycoord[pA];
```

---

raffineria.dat

---

```
set PORTI := A B C;

param xcoord :=
  A  0
  B  300
  C  700;
param ycoord :=
  A  0
  B  400
  C  300;

param pA := A;
param raggio:= 360;

let x:= 1;
let y:= 1;

option solver loqo;
solve;

display tubi;
display x,y;
```

Come si vede, nel file di modello sono presenti le seguenti dichiarazioni:

- `set PORTI;` che dichiara `PORTI` come insieme di elementi non meglio specificati, nel senso che nel file di modello non si fa alcun riferimento esplicito né al numero di elementi dell'insieme né alla loro tipologia (numeri, lettere, stringhe, ...);
- `param xcoord{PORTI};` che dichiara `xcoord` come vettore di elementi. Più in particolare la presente dichiarazione ci dice che il vettore `xcoord` avrà tanti elementi quanti saranno gli elementi dell'insieme `PORTI` precedentemente dichiarato;

- `param pA symbolic in PORTI`; che dichiara `pA` come un generico elemento dell'insieme `PORTI`. Questa istruzione è molto utile ogni volta che si vuole esprimere un vincolo che riguarda un elemento specifico di un insieme. Infatti utilizzando questa sintassi non si specifica al momento della dichiarazione del modello l'elemento a cui si fa riferimento. Se quindi questo dovesse cambiare, basta modificare il file `.dat` in cui l'elemento viene specificato;
- `param dist_min`; che, infine, dichiara `dist_min` come una quantità parametrica scalare.

Osservando attentamente il file dei dati `raffineria.dat` notiamo che questo contiene una definizione per ciascun parametro e/o insieme del file `raffineria.dat`. In particolare:

- `set PORTI := A B C`; che definisce `PORTI` come insieme costituito dai quattro elementi `A,B,C`;
- `param xcoord := ...` che definisce ciascuna componente del vettore `xcoord`;
- `param pA := A`; che assegna al parametro `pA` l'elemento `A` dell'insieme `PORTI` (come richiedeva la dichiarazione);
- `let x := 1`; che fissa ad 1 il valore iniziale per la variabile `x`;

Infine, nel file `raffineria.dat`, per comodità, sono stati inserite anche le istruzioni

- `option solver loqo`; che imposta `LOQO` come solutore di default per `AMPL`;
- `solve`; che avvia la soluzione del problema usando il solutore di default;
- `display tubi`; che stampa a video il valore della funzione obiettivo nel punto soluzione;

Notiamo che sono stati cambiati i valori iniziali delle variabili. In questo caso il solutore `loqo` stampa a video il seguente risultato:

```
LOQO 6.01: iteration limit (500 iterations, 2109 evaluations)
primal objective 928.2440326
dual objective 214.4195297
```

che significa che il solutore non è riuscito a produrre una soluzione entro il numero massimo di iterazioni. A questo punto si hanno tre alternative:

1. aumentare il numero massimo di iterazioni consentite (il valore di default è 200) tramite il comando:

```
ampl: option loqo_options 'iterlim=10000';
```

Ma il risultato non cambia.

2. Cambiare solutore. In questo caso però sia `MINOS` che `SNOPT` non riescono a trovare la soluzione, producendole due schermate

```
MINOS 5.5: unbounded (or badly scaled) problem.
0 iterations
Nonlin evals: obj = 26, grad = 25, constrs = 26, Jac = 25.
```

e

```
SNOPT 6.1-1: The current point cannot be improved.
5 iterations, objective 29949.50796 Nonlin evals: obj = 5, grad = 4, constrs =
5, Jac = 4.
```

3. cambiare il punto iniziale. E in questo caso il solutore LOQO riesce a trovare la soluzione e la schermata ottenuta è :

LOQO 6.01: optimal solution (20 iterations, 39 evaluations) primal  
objective 910.5436033 dual objective 910.5436059

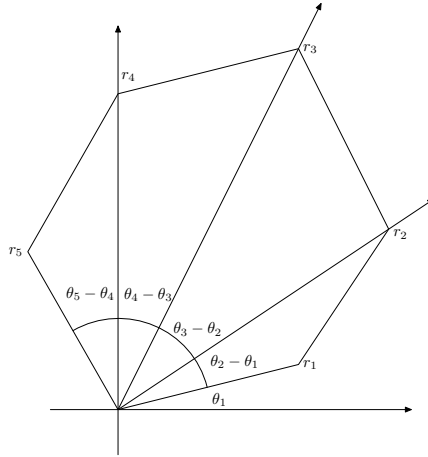
Questi risultati evidenziano l'importanza della scelta del punto iniziale nel caso di problemi non lineari.

## 5 Il problema del poligono di massima area

Fissato un numero  $n_v$  di vertici, il problema che vogliamo trattare in questa sezione è quello, molto comune p.es. nell'ingegneria meccanica, di determinare il poligono di diametro<sup>1</sup> unitario avente area massima [6],[2].

Siano  $(r_i, \theta_i)$   $i = 1, \dots, n_v$ , i vertici del poligono espressi in coordinate polari. Evidentemente, siccome ogni poligono è univocamente determinato mediante le coordinate dei suoi vertici, queste sono le variabili di decisione del nostro problema.

Se immaginiamo di posizionare l'ultimo vertice del poligono, quello di coordinate  $(r_{n_v}, \theta_{n_v})$  nell'origine, ovvero imponendo  $(r_{n_v}, \theta_{n_v}) = (0, \pi)$ , e se facciamo l'ipotesi che gli angoli  $\theta_i$  siano ordinati dal più piccolo al più grande (vedi figura) ,



possiamo esprimere l'area del poligono stesso mediante l'espressione

$$\frac{1}{2} \sum_{i=1}^{n_v-1} r_{i+1} r_i \sin(\theta_{i+1} - \theta_i),$$

ovvero come somma delle aree degli  $n_v - 1$  triangoli (ciascuno con un vertice nell'origine) in cui è possibile suddividere il poligono. Vediamo ora di formalizzare i vincoli del problema. Anzitutto abbiamo i vincoli che impongono l'ordinamento sugli angoli, ovvero

$$\theta_i \leq \theta_{i+1} \quad i = 1, \dots, n_v - 1.$$

In secondo luogo avremo un gruppo di vincoli che servono a garantire che i poligoni abbiano diametro non superiore ad uno. Quindi, richiamando la definizione di diametro di un poligono tali vincoli sono

$$D_{i,j}^2 = r_i^2 + r_j^2 - 2r_i r_j \cos(\theta_i - \theta_j) \leq 1 \quad i = 1, \dots, n_v - 1 \text{ e } j = i + 1, \dots, n_v. \quad (2)$$

<sup>1</sup>Il diametro di un poligono è pari alla massima distanza tra due suoi vertici

In realtà, stando alla descrizione del problema, avremmo dovuto considerare il seguente (unico) vincolo

$$\max_{\substack{i=1, \dots, n_v-1 \\ j=i+1, \dots, n_v}} \{D_{i,j}^2\} = \max_{\substack{i=1, \dots, n_v-1 \\ j=i+1, \dots, n_v}} \{r_i^2 + r_j^2 - 2r_i r_j \cos(\theta_i - \theta_j)\} = 1.$$

Tuttavia, considerando il fatto che, comunque scelto un poligono di diametro inferiore ad uno, ne esiste sempre uno, simile al primo ma con diametro esattamente pari ad uno, nel precedente vincolo possiamo considerare il segno di minore o uguale anziché proprio di uguale. A questo punto si capisce il perché della presenza dei vincoli (2). Infatti, piuttosto che affermare che il massimo di  $D_{i,j}^2$  sia minore o uguale di 1, possiamo, equivalentemente richiedere che ogni  $D_{i,j}^2$  sia minore o uguale di 1.

Ci saranno poi dei vincoli che servono a fissare l'ultimo vertice nell'origine e cioè

$$r_{n_v} = 0, \quad \theta_{n_v} = \pi.$$

Infine aggiungiamo dei vincoli di box sulle variabili

$$0 \leq r_i \leq 1, \quad 0 \leq \theta_i \leq \pi \quad i = 1, \dots, n_v.$$

Per scrivere in AMPL un file di modello per il problema sopra descritto utilizziamo l'insieme particolare `1..nv` che ha un ordinamento intrinseco che è quello dei numeri naturali. Un possibile file `.mod` è il seguente:

---

```

polygon.mod
-----
model;

param nv integer > 2;          # number of vertices in the
polygon param pi := 3.14159265358979; # approximation of pi

var    r {i in 1..nv};        # polar radius
in1..nv};                    # polar angle

maximize polygon_area:
    0.5*sum{i in 1..nv-1} r[i+1]*r[i]*sin(theta[i+1] - theta[i]);

s.t.    r_bounds{i in 1..nv}: 0.0 <= r[i] <= 1.0; s.t.
theta_bounds{i in 1..nv}: 0.0 <= theta[i] <= pi;

s.t.    fix_theta_nv: theta[nv] = pi; s.t.    fix_r_nv:    r[nv]
=0.0;

s.t.    ordered_theta{i in 1..nv-1}: theta[i] <= theta[i+1];

s.t.    distance {i in 1..nv-1,j in i+1..nv}:
    r[i]^2 + r[j]^2 - 2*r[i]*r[j]*cos(theta[j] - theta[i]) <= 1;

```

---

Notiamo che nel precedente file `polygon.mod`, il numero di vertici del poligono `nv` essendo stato dichiarato come parametro, non ha ancora un valore fissato. Inoltre, non è stato assegnato alcun punto iniziale per le variabili. Prima di poter risolvere il problema, è dunque necessario specificare all'interprete AMPL quale valore attribuire ad `nv` e con quali valori inizializzare le variabili. Un modo per completare queste operazioni, e poter quindi risolvere il problema, consiste nello scrivere un file di dati (`polygon.dat`) come il seguente.

---

```

polygon.dat
-----

```

---

```
# Largest-small polygon problem

data;

param nv := 3;      # Number of vertices

let {i in 1..nv-1} r[i] := 0.5; # Initial values
let {i in 1..nv-1} theta[i] := pi*i/nv;
```

---

Per chiedere ad AMPL di risolvere il problema usando come solutore SNOPT [5], al prompt dei comandi di AMPL dobbiamo digitare le seguenti istruzioni:

```
ampl: reset;
ampl: model polygon.mod;
ampl: data polygon.dat;
ampl: option solver snopt;
ampl: solve;
```

Otteniamo il seguente output

```
SNOPT 6.1-1(4)(Jun 2001): Optimal solution found. 10 iterations,
objective 0.4330127026 Nonlin evals: obj = 8, grad = 7, constrs =
8, Jac = 7.
```

Per vedere quale è la soluzione trovata scriviamo al prompt di AMPL il seguente comando:

```
ampl: printf{i in 1..nv}: "%10.6f, %10.6f,\n",r[i]*cos(theta[i]),r[i]*sin(theta[i]);
```

che restituisce le coordinate *cartesiane* dei vertici del poligono trovato.

```
0.500000, 0.866025,
-0.500000, 0.866025,
0.000000, 0.000000,
```

Come si vede chiaramente anche nella figura, il poligono di area massima con tre vertici è, come ci saremmo potuti ragionevolmente aspettare, un triangolo equilatero.

Provando a risolvere il problema con  $\overline{n_v = 9}$ , e quindi modificando solo la definizione del parametro `nv` nel file di dati `polygon.dat`, otteniamo la soluzione riportata in figura. È interessante notare come, all'aumentare del numero dei vertici del poligono, la soluzione ottima tenda ad una circonferenza di diametro unitario.

Lo studente provi, modificando il file `polygon.dat`, a cambiare il punto iniziale e/o il numero dei vertici del poligono, tenendo presente che, per le limitazioni imposte sulla versione studenti di AMPL, non è possibile risolvere questo problema quando  $n_v$  sia superiore a 24.

## 6 Problema del riciclaggio di rifiuti

Consideriamo il seguente problema:

un'azienda opera nel settore del riciclaggio del vetro e della plastica. L'azienda può acquistare rifiuti di due tipi: indifferenziato e multimateriale in quantità (in tonnellate) e al costo (in euro per tonnellata) riportati nella seguente tabella:

rifiuto	quantità max	costo
indifferenziato	150	125
multimateriale	90	190

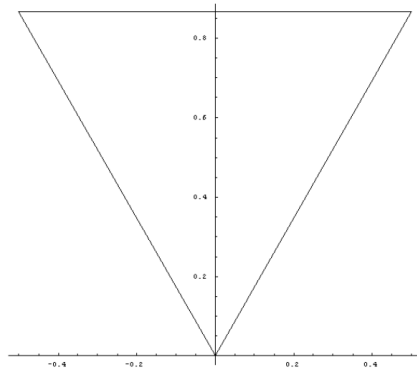


Figura 2: Poligono ottenuto nel caso  $nv = 3$

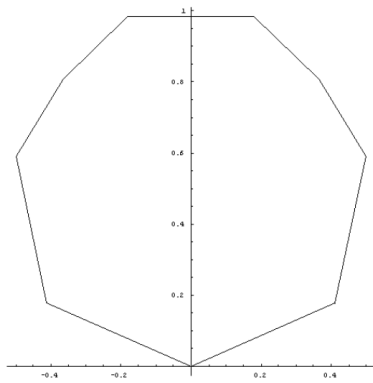


Figura 3: Poligono ottenuto nel caso  $nv = 9$

Per separare vetro e plastica dal rifiuto sia indifferenziato che multimateriale è necessario utilizzare un processo di separazione al costo (in euro per tonnellata) riportato in tabella:

	costi	
	indifferenziato	multimateriale
vetro	90	20
plastica	65	15

Per ciascuno dei due tipi di rifiuto il quantitativo percentuale di vetro e plastica ottenibile è riportato nella seguente tabella:

rifiuto	% max di vetro	% max di plastica
indifferenziato	30	25
multimateriale	60	30

Il rifiuto residuo dei processi di separazione deve essere eliminato in una discarica al costo di 15 euro per tonnellata. L'azienda rivende il vetro e la plastica riciclate e si suppone che il mercato sia in grado di assorbire tutta la quantità prodotta e che l'azienda non sia obbligata ad acquistare tutto il rifiuto disponibile. I prezzi di vendita (euro per tonnellata) sono riportati nella seguente tabella

	prezzo
vetro	300
plastica	250

Scrivere un modello di PL che consenta di massimizzare il profitto.

Le variabili di decisione del problema sono la quantità acquistata dei due tipi di rifiuti, che indicheremo con  $y_j$  e la quantità di vetro e plastica estratta da ogni tipo di rifiuto che indicheremo con  $x_{ij}$ . Generalizzando si avranno un insieme costituito da  $n$  tipi di rifiuti (in questo caso multimateriale e indifferenziato,  $n = 2$ ) e un insieme costituito da  $m$  materiali estraibili da ogni tipo di rifiuti (in questo caso plastica e vetro,  $m = 2$ ). Se indichiamo con  $p_i$  il prezzo di vendita dei materiali estratti dai rifiuti, con  $c_j$  il costo di acquisto di ogni tipo di rifiuto, con  $csep_{ij}$  il costo di separazione del materiale  $i$  dal rifiuto  $j$  e con  $cres$  il costo di smaltimento del rifiuto residuo la funzione obiettivo, che rappresenta il ricavo, è la seguente:

$$\max \sum_i^m \sum_j^n p_i x_{ij} - \sum_j^n c_j y_j - \sum_i^m \sum_j^n csep_{ij} x_{ij} - cres \left( \sum_j^n (y_j - \sum_i^m x_{ij}) \right)$$

Per quel che riguarda i vincoli, abbiamo il vincolo che limita la massima quantità acquistabile di ogni tipo di rifiuto (indicata con  $qmax_j$ ):

$$y_j \leq qmax_j, \quad j = 1, \dots, n$$

e il vincolo che limita la massima percentuale ( $perc_{ij}$ ) di ogni materiale ottenibile da ogni tipo di rifiuto:

$$x_{ij} \leq perc_{ij} y_j, \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

A questo punto passiamo a vedere come trascrivere il modello matematico in un modello AMPL. Il file che descrive il modello è il seguente:

---

rifiuti.mod

```

reset;
set rif;
set mat;
param p{mat};
param qmax{rif};

```

```

param c{rif};
param csep{mat,rif};
param cres;
param perc{rif, mat};

var x{mat,rif}>=0;
var y{rif}>=0;

maximize ricavo: sum{i in mat, j in rif}prezzi[i]*x[i,j]
-sum{j in rif}costo[j]*y[j]-sum{j in rif,i in mat}costi[i,j]*x[i,j]
-cres*sum{k in rif}(y[k] -sum{l in mat}x[l,k]);
s.t. vin1{j in rif}:y[j]<=qmax[j];
s.t. vin2{j in rif,i in mat}:x[i,j]<= perc[j,i]*y[j];

```

---

mentre quello che descrive i dati del problema è:

rifiuti.dat

---

```

set rif:=mm, ind;
set mat := p,v;
param cres:=30;
param prezzi:= p 300
                v 250;
param : qmax   costo :=
mm 90   190
ind 150 125 ;

param csep: mm   ind:=
p   15   65
v   20   90;

param perc: p    v:=
mm 30   60
ind 25  30;

```

---

Le istruzioni per far risolvere il problema sono le seguenti:

```

ampl: reset;
ampl: model rifiuti.mod;
ampl: data C:\rifiuti.dat;
ampl: option solver cplex; ampl: solve;

```

e il risultato che si ottiene è:

```

CPLEX 8.0.0: optimal solution; objective 4060200
0 dual simplex iterations (0 in phase I)

```

Per vedere i valori delle variabili all'ottimo, le istruzioni sono le seguenti:

```
AMPL: display x;  
x := p ind 3750  
    p mm 2700  
    v ind 4500  
    v mm 5400 ;
```

```
AMPL: display y;  
y [*] := ind 150  
        mm 90;
```

## Riferimenti bibliografici

- [1] I. BONGARTZ, A. R. CONN, N. I. M. GOULD, AND PH. L. TOINT, *CUTE: Constrained and unconstrained testing environment*, ACM Transaction on Mathematical Software, 21 (1995), pp. 123–160.
- [2] A. BONDARENKO, D. BORTZ, AND J. J. MORÉ, *COPS: Large-scale nonlinearly constrained optimization problems*, Tech. Rep. ANL/MCS-TM-237, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, 1998. Revised October 1999.
- [3] R. FOURER, D. M. GAY, AND B. W. KERNIGHAN, *AMPL a modeling language for mathematical programming*, body & fraser publishing company, Massachusetts, 1993.
- [4] A. FRIEDMAN, *Free boundary problems in science and technology*, Notices Amer. Math. Soc., 47(2000), pp. 854-861.
- [5] P. E. GILL, W. MURRAY, AND M. A. SAUNDERS, *SNOPT: an SQP algorithm for large-scale constrained optimization*, SIAM J. Optimization, (2001).
- [6] R. L. GRAHAM, *The largest small hexagon*, Journal Combin. Th., 18(1975), pp. 165-170.
- [7] D. F. SHANNO AND R. J. VANDERBEI, *An interior point algorithm for nonconvex nonlinear programming*, Computational Optimization and Applications, 13 (1999), pp. 231–252.
- [8] B. A. MURTAGH AND M. A. SAUNDERS, *A projected Lagrangian algorithm and its implementation for sparse non-linear constraints*, Math. Programming Studies, 16 (1982), pp. 84–117.