Introduzione all'Optimization Toolbox di MATLAB

Barbara Panicucci   Massimo Pappalardo   Mauro Passacantando

Indice

Capitolo 1

Introduzione a MATLAB

Obiettivo di questo capitolo è fornire una panoramica sulle funzioni basilari di MATLAB. In particolare verrà discusso:

- come avviare MATLAB, avere informazioni sull'ambiente di lavoro, uscire da MATLAB;
- come usare l'Help;
- come gestire una sessione di lavoro: immissione dei comandi, operazioni matriciali, file di funzione.

1.1 Avviare MATLAB

Per avviare MATLAB è sufficiente fare doppio clic sull'icona MATLAB; dopo pochi secondi viene visualizzata la schermata riportata in figura 1.1.

In essa si possono distinguere vari parti:

- la *barra dei menù* è la riga in alto contenente i nomi:

  File Edit View Web Window Help

  Ognuno di questi nomi identifica un elenco di comandi; ogni elenco di comandi si chiama *menù*; vi è quindi un menù *File* contenente una serie di comandi relativi ai file; un menù *Edit* contenente una serie di comandi che servono quando si scrivono i programmi etc.

- la *barra degli strumenti* consiste di icone che rappresentano operazioni consuete di MATLAB. Fare clic su una di queste icone equivale ad aprire il menù e selezionare la corrispondente opzione. Le prime sette icone da sinistra corrispondono alle opzioni *New File, Open File, Cut, Copy, Paste, Undo e Redo.* L'ottava icona avvia Simulink; la nona icona (quella con il punto interrogativo) permette di accedere alla guida di MATLAB. Il

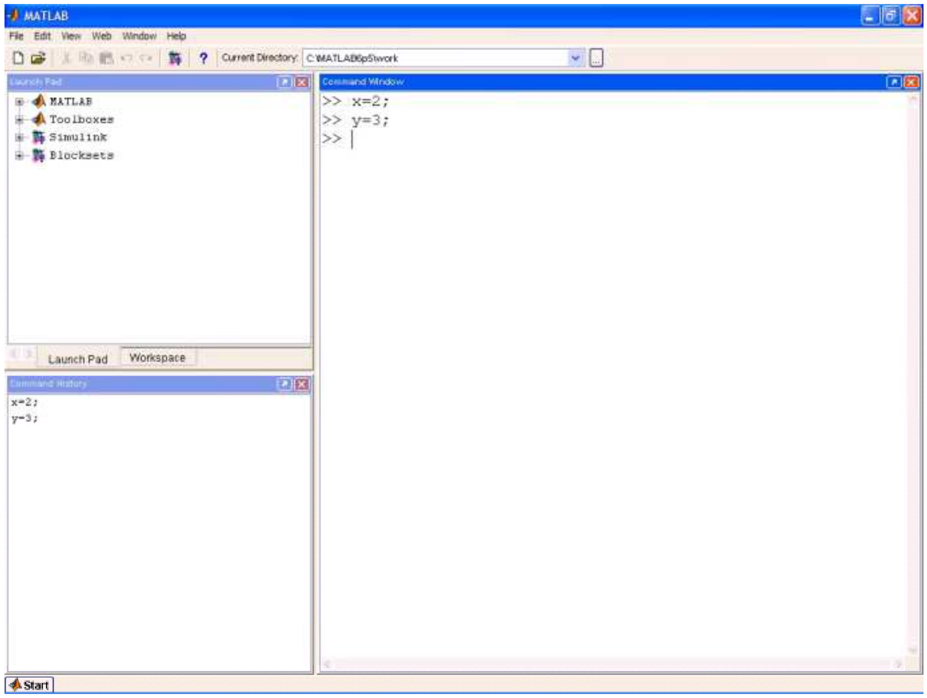


Figura 1.1: Schermata iniziale di MATLAB.

riquadro posto a destra della barra degli strumenti indica la cartella di lavoro corrente (*Current Directory*).

- *Command Window* (la finestra dei comandi) in cui si digitano i nomi dei comandi o le istruzioni da eseguire.

- *Command History* (la cronologia dei comandi) visualizza tutti i comandi che sono stati digitati durante la sessione di lavoro corrente.

- *Launch Pad* (strumenti di MATLAB) contiene un grafo ad albero i cui nodi rappresentano tutte le cartelle e i file di MATLAB. Se vicino ad un nodo c'è un +, significa che esso contiene cartelle e file che non sono visualizzati. Se si fa clic sul + vengono visualizzate le cartelle ed i file contenuti, e il + viene trasformato in - (facendo clic sul meno si torna alla situazione precedente).

- *Workspace* (area di lavoro) mostra i nomi e i valori di tutte le variabili utilizzate nella sessione di lavoro corrente.

È possibile modificare l'aspetto del desktop di MATLAB. Per ripristinare la configurazione standard è sufficiente fare clic su *View*, spostare la freccia del mouse su *Desktop Layout* e selezionare *Default*.

Per chiudere la sessione di lavoro ed uscire da MATLAB è sufficiente selezionare il comando *Exit MATLAB* dal menù *File* oppure digitare `quit`.

1.2 Come usare l'Help

Per avere una panoramica completa di tutta la documentazione di MATLAB è importante imparare ad utilizzare la guida interna. Per accedere alla guida è sufficiente fare clic sull'icona con il punto interrogativo, o selezionare l'opzione *Help* dal menù *View*. Sullo schermo apparirà il browser illustrato nella figura 1.2.

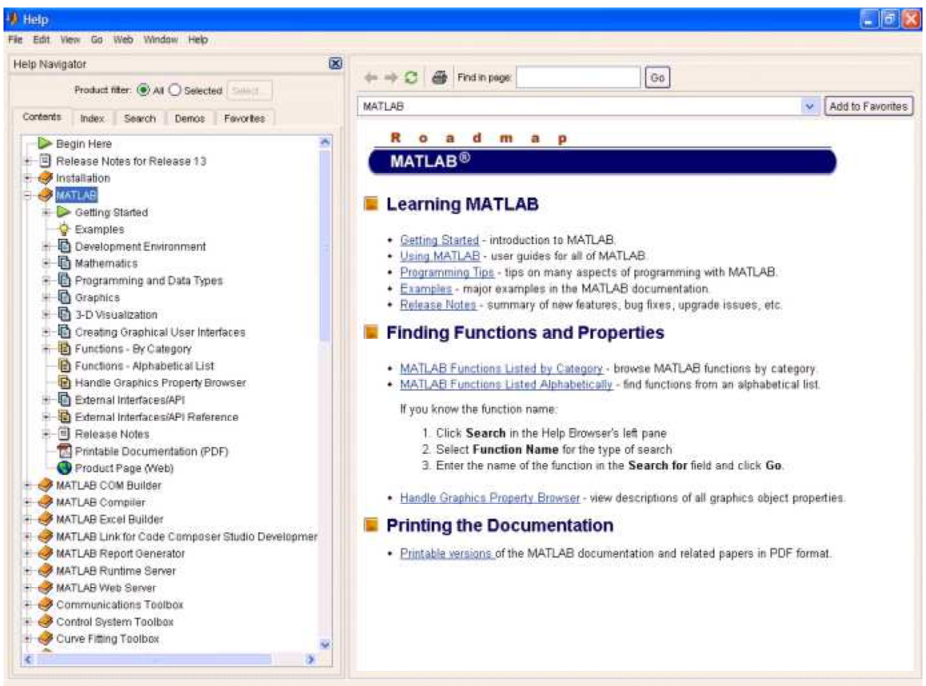


Figura 1.2: Help di MATLAB.

La finestra dell'*Help* è costituita da due pannelli:

- *Help Navigator*, sulla sinistra, in cui ci sono cinque schede:
  - *Contents*: un elenco di argomenti.
  - *Index*: un indice generale.
  - *Search*: per cercare tra tutti i documenti della guida quelli che contengono una parola o frase specifica.
  - *Demos*: dimostrazioni dell'uso di MATLAB.
  - *Favorites*: per visualizzare la lista dei documenti preferiti ed aggiungerne altri.

- *Display*, sulla destra, in cui viene visualizzata la documentazione.

1.3 Immettere i comandi

Quando nella finestra dei comandi compare il prompt di MATLAB (>>), il programma è pronto a ricevere nuove istruzioni. Se il cursore non si trova dopo il prompt, si può utilizzare il mouse per spostarlo. Quando invece sono in corso operazioni il prompt scompare. Per annullare una operazione, bisogna premere contemporaneamente il tasto `Ctrl` e `c`. A questo punto è possibile iniziare ad immettere i comandi. Il modo migliore per illustrare l'inserimento di un comando è quello di utilizzare un esempio. Digitando `1/700` dopo il prompt e premendo *Invio* si ottiene

```
ans =
    0.0014
```

MATLAB assegna la risposta ad una variabile temporanea chiamata `ans`, dall'inglese answer (risposta). Per default il risultato viene visualizzato con 4 cifre decimali. Il comando `format` permette di modificare il formato di uscita secondo la seguente tabella:

Comando	**Descrizione**
`format short`	4 cifre decimali
`format long`	14 cifre decimali
`format short e`	4 cifre decimali più l'esponente
`format long e`	15 cifre decimali più l'esponente
`format rat`	approssimazione razionale

Tabella 1.1: Formati numerici

Si osservi che il formato di default è `short`. Se digitiamo

```
>> format long
>> 1/700
```

otteniamo

```
ans =
    0.00142857142857
```

In modo analogo

```
>> format short e
>> 1/700
```

restituisce

```
ans =
    1.4286e-003
```

e

```
>> format long e
>> 1/700
ans =
    1.428571428571429e-003
```

Mentre,

```
>> format rat
>> 1/700
```

visualizza

```
ans =
    1/700
```

In particolare MATLAB può essere usato come calcolatrice. I simboli per svolgere le operazioni sono quelli standard, come descritto nella seguente tabella:

Simbolo	**Operazione**	**Formato di MATLAB**
^	elevazione a potenza	`a^b`
*	Moltiplicazione	`a*b`
/	Divisione	`a/b`
+	Addizione	`a+b`
-	Sottrazione	`a-b`

Tabella 1.2: Operazione aritmetiche

Se si commette un errore durante la digitazione , si ritorna ai comandi digitati in precedenza con la freccia in alto (↑) e si utilizza la freccia in basso (↓) per far scorrere al contrario la lista dei comandi(dal più vecchio al più recente). Per spostare il cursore a sinistra o a destra all'interno della riga corrente è sufficiente premere i tasti (←) o (→) Quando si trova il comando in cui si è commesso un errore, si modifica utilizzando il tasto *Canc* per cancellare il carattere che si trova davanti al cursore o quello *Backspace* per cancellare il carattere che si trova dietro il cursore. Poi si preme *invio* per eseguire il comando.

Il punto e virgola posto alla fine di un comando indica a MATLAB di non visualizzare i risultati dell'istruzione sullo schermo.

1.4 Variabili e Costanti

I programmi in genere registrano dati in memoria; le zone di memoria in cui vengono registrati i dati si chiamano *variabili*. Per assegnare il valore ad una variabile MATLAB usa l'operatore di assegnazione indicato con il segno uguale (`=`) . Per esempio il comando $x = 2$, permette di registrare nella variabile x il valore 2. Per registrare nella variabile x il risultato dell'addizione tra 3 e il contenuto della variabile y si scrive:

```
>> x=y+3
```

Quindi per assegnare ad una variabile un valore si scrive:

- il nome della variabile
- il simbolo =
- il valore da inserire oppure un'espressione.

Sono ammessi anche assegnamenti simili a quelli del seguente esempio:

```
>> x=x+1
```

nella variabile x viene registrato il risultato dell'addizione tra il contenuto di x ed 1. Se, prima di questa istruzione, il contenuto di x era 8, dopo sarà 9.

L'operatore di assegnazione è diverso dall'operatore di uguaglianza che viene invece indicato con due segni uguale (`= =`) (vedi paragrafo 1.6). In particolare il comando $x = 5$ è diverso dal comando $5 = x$ che genera un messaggio di errore.

Il nome di una variabile deve iniziare con una lettera, che può essere seguita da una qualunque combinazione di lettere, cifre e caratteri di sottolineatura, ma non può essere più lungo di 32 caratteri. MATLAB distingue tra caratteri maiuscoli e minuscoli, per cui `A` e `a` identificano variabili diverse. Per conoscere il valore corrente di una variabile è sufficiente digitare il suo nome e premere *Invio*. Per sapere se la variabile x è già stata definita, basta digitare

```
>>exist('x')
```

Se MATLAB restituisce il valore 1, la variabile esiste; se restituisce il valore 0, la variabile non esiste. I nomi ed i valori di tutte le variabili utilizzate nella sessione di lavoro corrente si trovano nel *Workspace*. In alternativa,il comando

```
>> who
```

elenca i nomi di tutte le variabili, ma non indica i loro valori. MATLAB conserva l'ultimo valore di una variabile finché la sessione di lavoro corrente è aperta o finché la variabile non è eliminata espressamente. Il comando

```
>> clear
```

rimuove tutte la variabili dall'area di lavoro. Il comando

```
>> clear var1 var2 ...
```

consente di eliminare le variabili var1 var2 .... Prima di uscire da MATLAB è possibile salvare la sessione di lavoro con il comando

```
>> save
```

Questo comando salva tutte le variabili nel file matlab.mat. Se si desidera salvare la sessione di lavoro con un nome diverso è sufficiente digitare

```
>> save nomefile
```

Tutte le variabili saranno salvate nel file nomefile.mat. Il comando

```
>> load nomefile
```

(notare che non è necessario specificare l'estensione) ricarica tutte le variabili memorizzate in nomefile.mat mantenendo inalterato il nome con cui erano state memorizzate. Il comando

```
>> clc
```

cancella il contenuto della finestra dei comandi, lasciando inalterate le variabili.

In MATLAB ci sono anche delle costanti e variabili predefinite. Per esempio, digitando 1/0 si ottiene in uscita:

```
ans =
        Inf
```

Il simbolo `Inf` sta per ∞, `ans` è la variabile temporanea che contiene il risultato più recente. Mentre digitando `Inf-Inf` si ottiene

```
ans =
        NaN
```

Il simbolo `NaN`, acronimo per "Not a Number", indica un risultato numerico indefinito.

1.5 Vettori e matrici

Per creare un vettore riga basta digitare gli elementi all'interno di una coppia di parentesi quadre (`[ ]`), separandoli con uno spazio o una virgola (`,`). Ad esempio, il comando per creare il vettore $a = (2, 4, 10)$ è:

```
>> a=[2 4 10]
```

oppure

```
>> a=[2, 4, 10]
```

Per creare un vettore colonna si possono digitare gli elementi separati dal punto e virgola (`;`) oppure creare un vettore riga e fare il trasposto con il segno di apice (`'`). Ad esempio, il vettore $b = \begin{pmatrix} 1 \\ 5 \\ 7 \end{pmatrix}$ è definito dal comando:

```
>> b=[1; 5; 7]
```

oppure

```
>> b=[1, 5, 7]'
```

È possibile creare un vettore riga o colonna "accodando" un vettore ad un altro. Ad esempio, se

```
>> x=[2, 4, 20];
>> y=[9, 6, 3];
```

allora `z=[x, y]` dà come risultato:

```
z=
     2     4     20     9     6     3
```

mentre con il comando `z=[x'; y']` si ottiene:

```
z=
     2
     4
     20
     9
     6
     3
```

Per selezionare le componenti di indici $a, b, c, \ldots$ del vettore x si scrive `x([a, b, c, ...])`, ad esempio:

```
>> x=[4, 6, 1, 9, 3];
>> z=x([1, 2, 4])
z=
     4     6     9
```

Per generare un vettore x con elementi intervallati regolarmente da a a b con incremento pari a q si scrive `x=[a:q:b]`, ad esempio:

```
>> x=[0:2:8]
x=
     0     2     4     6     8
```

Se viene omesso l'incremento q, allora MATLAB lo pone uguale a 1:

```
>> x=4:6
x=
     4     5     6
```

Sono permessi anche incrementi negativi, ad esempio:

```
>> x=7:-2:3
x=
     7     5     3
```

Per avere un vettore x con m componenti intervallate regolarmente da a a b si usa il comando `x=linspace(a,b,m)`, ad esempio:

```
>> x=linspace(1,7,4)
x=
    1    3    5    7
```

Il comando `length(x)` fornisce il numero di componenti del vettore x, ad esempio:

```
>> length([3, 4, 8, 1])
ans=
    4
```

La somma e la sottrazione di due vettori riga (o due vettori colonna) della stessa lunghezza si effettuano rispettivamente con il `+` ed il `-`, ad esempio:

```
>> x=[3, 4, 8, 1];
>> y=[-2, 9, 1, 0];
>> x+y
ans=
    1    13    9    1
>> x-y
ans=
    5    -5    7    1
```

Il prodotto tra uno scalare ed un vettore si effettua con il `*`, ad esempio:

```
>> x=[3, -4, 8, 1];
>> 2*x
ans=
    6    -8    16    2
```

Anche il prodotto scalare tra un vettore riga ed un vettore colonna (con lo stesso numero di componenti) si effettua con il `*`, ad esempio:

```
>> x=[3, 4, 8, 1];
>> y=[-2; 9; 1; 0];
>> x*y
ans=
    38
```

Per creare una matrice si inseriscono gli elementi per riga separati da spazi o virgole e per passare alla riga successiva si usa il punto e virgola, ad esempio, per inserire la matrice $A = \begin{pmatrix} 1 & 2 & 3 & 6 \\ 3 & -5 & -8 & 12 \\ 2 & 0 & 1 & 9 \end{pmatrix}$, si digita:

```
>> A=[1, 2, 3, 6; 3, -5, -8, 12; 2, 0, 1, 9]
```

Per creare una matrice ottenuta dalla matrice A aggiungendo il vettore colonna $b = \begin{pmatrix} 3 \\ -5 \\ 0 \end{pmatrix}$ si scrive `[A, b]`, mentre per aggiungere la riga $c = (6, 5, 8, 3)$ si digita `[A;c]`.

Il comando `eye(n)`[1] genera la matrice identità di ordine n:

```
>> eye(3)
ans=
    1    0    0
    0    1    0
    0    0    1
```

Il comando `ones(m,n)` genera una matrice di ordine $m \times n$ i cui elementi sono tutti uguali a 1:

```
>> ones(3,2)
ans=
    1    1
    1    1
    1    1
```

mentre `zeros(m,n)` genera una matrice nulla $m \times n$:

```
>> zeros(3,4)
ans=
    0    0    0    0
    0    0    0    0
    0    0    0    0
```

Il comando `diag` ha due modi di funzionamento: se l'argomento è una matrice quadrata, fornisce gli elementi sulla diagonale; se l'argomento è un vettore, genera una matrice diagonale i cui elementi diagonali sono gli elementi del vettore. Ad esempio:

```
>> diag([6, 4, 9; 5, 8, 2; 7, 5, 1])
ans=
    6
    8
    1
```

e

```
>> diag([3, 5, 1])
ans=
    3    0    0
    0    5    0
    0    0    1
```

[1] La matrice identità si indica, di solito, con la lettera I che in inglese ha la stessa pronuncia della parola "eye".

Per sapere la dimensione di una matrice si usa il comando `size`:

```
>> A=[3, 5, 1; 6, 2, 8];
>> size(A)
ans=
    2     3
```

Per estrarre la i-esima riga della matrice A si usa il comando `A(i,:)`, mentre per estrarre dalla i-esima alla j-esima riga di A si usa `A(i:j,:)`. In modo analogo si estraggono le colonne.
Ad esempio, se $A = \begin{pmatrix} 2 & 4 & 10 & 13 \\ 16 & 3 & 7 & 18 \\ 8 & 4 & 9 & 25 \\ 3 & 12 & 15 & 16 \end{pmatrix}$, allora:

```
>> A(1:2,:)
ans=
      2  4  10  13
     16  3   7  18
>> A(:,3:4)
ans=
     10  13
      7  18
      9  25
     15  16
>> A(2:3,1:3)
ans=
     16  3  7
      8  4  9
```

La somma e la differenze tra matrici (dello stesso ordine) si effettuano rispettivamente con il segno `+` ed il segno `-` come tra due vettori, ad esempio:

```
>> A=[-7, 16; 4, 9];
>> B=[6, -5; 12, -2];
>> A+B
ans=
     -1  11
     16   7
```

Per fare il prodotto tra uno scalare ed una matrice basta usare il segno `*`, ad esempio:

```
>> A=[-7, 16; 4, 9];
>> 2*A
ans=
     -14  32
       8  18
```

Il prodotto riga per colonna tra una matrice A di ordine $m \times n$ e d una matrice B di ordine $n \times p$ si effettua anch'esso con il segno `*`, ad esempio:

```
>> A=[-7, 16; 4, 9];
>> B=[6, -5; 12, -2];
>> A*B
ans=
     150    3
     132  -38
```

Il rango di una matrice si calcola con il comando `rank`, ad esempio:

```
>> A=[10, 4; 4, 2];
>> rank(A)
ans=
    2
```

Il determinante di una matrice quadrata si calcola con in comando `det`, ad esempio:

```
>> A=[10, 4; 4, 2];
>> det(A)
ans=
    4
```

L'inversa di una matrice quadrata (con determinante non nullo) si trova con il comando `inv`, ad esempio:

```
>> A=[10, 4; 4, 2];
>> inv(A)
ans=
      0.5000  -1.0000
     -1.0000   2.5000
```

Gli autovalori e gli autovettori di una matrice A si calcolano con il comando `[v,d]=eig(A)` che restituisce una matrice v contenente nelle colonne gli autovettori di A, ed una matrice d i cui elementi diagonali sono i corrispondenti autovalori. Ad esempio:

```
>> A=[1, 0; 1, 2];
>> [v,d]=eig(A)
v=
          0   0.7071
     1.0000  -0.7071
d=
     2  0
     0  1
```

Se si cercano solo gli autovalori della matrice A, il comando `d=eig(A)` restituisce un vettore d contenente gli autovalori.

1.6 Operatori relazionali

MATLAB dispone di 6 operatori relazionali che consentono di confrontare variabili e vettori: Il risultato di un confronto può essere 0 (il confronto è falso) oppure 1 (il confronto è vero).

Simbolo	Descrizione
`==`	uguale a
`~=`	diverso da
`<`	minore di
`<=`	minore o uguale a
`>`	maggiore di
`>=`	maggiore o uguale a

Tabella 1.3: Operatori relazionali

Tale risultato può essere assegnato ad una variabile. Ad esempio, se $x = 2$ e $y = 3$, allora il comando `z=x<y` assegna alla variabile z il risultato del confronto $x < y$. In questo caso si ottiene:

```
z=
   1
```

Gli operatori relazionali permettono anche di confrontare, elemento per elemento, due vettori aventi lo stesso numero di componenti. Ad esempio, supponiamo che:

```
>> x=[6, 3, 9, 11];
>> y=[14, 2, 9, 13];
```

il comando `z=x<y` dà come risultato

```
z=
   1   0   0   1
```

mentre il comando `z=x<=y` dà come risultato

```
z=
   1   0   1   1
```

Gli operatori relazionali possono essere usati per selezionare gli elementi di un vettore. Ad esempio, il comando `z=x(x<y)` restituisce nel vettore `z` gli elementi di `x` che sono minori dei corrispondenti elementi di `y`, ossia il risultato è:

```
z=
   6   11
```

Il comando `z=find(x<y)` restituisce nel vettore `z` gli indici delle componenti del vettore `x` che sono minori delle corrispondenti componenti del vettore `y`, in questo caso il risultato è:

```
z=
   1   4
```

1.7 File script e file di funzione

Finora abbiamo operato in MATLAB in modalità interattiva, cioè i comandi sono stati immessi direttamente nella finestra dei comandi. Questo è conveniente solo per risolvere i problemi più semplici. Quando, invece, si devono ripetere più volte una serie di comandi, questi si possono salvare nei così detti M-file o file di tipo M. Questo tipo di file, detto *file script* contiene comandi MATLAB, quindi eseguire un file con estensione *m* equivale a digitare, uno alla volta, tutti i comandi registrati nel file. Per eseguire un file di tipo M, basta digitare il suo nome senza l'estensione *m*. Poiché contengono comandi, tali file sono chiamati anche *file di comandi*. Per creare un nuovo file script, per esempio il file prova.m, che contiene i comandi

```
A=[1 2 ;3 4];
x=det(A)
```

è sufficiente selezionare *New* dal menu *File* e poi *M-file*. Sullo schermo apparirà la finestra dell' *Editor/Debugger*.

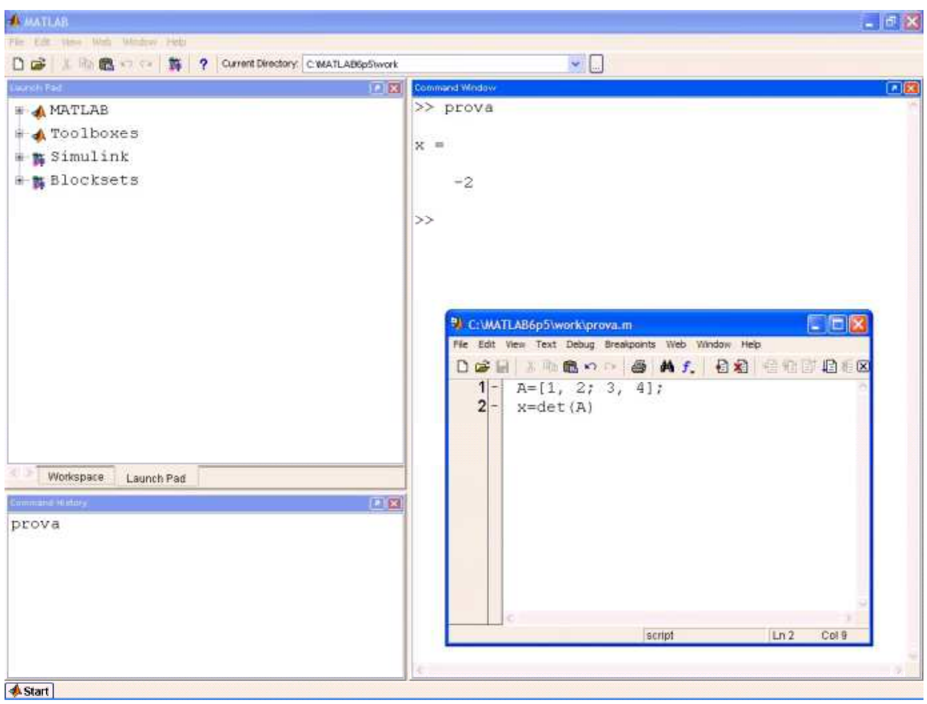


Figura 1.3: Esempio di un file script.

Si digitano i comandi e poi si salva il file selezionando l'opzione *Save* dal menu *File* nella finestra dell' *Editor/Debugger*: MATLAB fornisce per default il nome Untitled che verrà sostituito con il nome prova. Per salvare il file basa fare clic su *Save*. Una volta che il file è stato salvato, si esegue digitando il nome prova. I risultati del programma vengono visualizzati nella finestra dei comandi (vedi figura 1.3). Il nome di un *file script* segue le regole sui nomi delle variabili di MATLAB.

Un altro tipo di M-File sono i *file di funzione*. Diversamente dai file script, tutte le variabili in un file di funzione sono *locali* cioè sono disponibili soltanto all'interno della funzione. Per creare un file di funzione si apre un M-file come nel caso dei file script, e poi si definisce la funzione scrivendo nella prima riga del file

```
function [variabili di output] = nomefunzione(variabili di input)
```

Nelle successive righe si inseriscono i comandi che calcolano i valori delle variabili di output a partire dalle variabili di input. Consideriamo, ad esempio, la funzione che calcola l'area di un cerchio di raggio dato:

```
function y = areacerchio(r)
y=pi*r^2
```

La parola `function` deve essere scritta in lettere minuscole. Il nome della funzione (`nomefunzione`) deve essere uguale al nome del file in cui tale funzione è salvata (il file ha estensione m). La funzione viene chiamata digitando il suo nome dopo il prompt di MATLAB. Si noti che le variabili di output sono racchiuse tra parentesi quadre, mentre le variabili di input devono essere racchiuse tra parentesi tonde. Ad esempio:

```
>> areacerchio(10)
 ans=
     314.1593
```

Capitolo 2

Optimization Toolbox

2.1 La funzione `linprog`

Nell'*Optimization toolbox* di MATLAB, la funzione `linprog` risolve un problema di programmazione lineare della forma:

$$\begin{cases} \min\ c^{\mathsf{T}} x \\ A\,x \leq b \\ D\,x = e \\ l \leq x \leq u \end{cases} \tag{2.1}$$

dove c, x, b, e, l, u sono vettori e A, D sono matrici. Se, ad esempio, non ci sono vincoli di uguaglianza, si pongono `D=[ ]` ed `e=[ ]`.

La sintassi completa della funzione è la seguente:

```
[x, v, exitflag, output, lambda] = linprog(c, A, b, D, e, l, u)
```

dove gli input

```
c, A, b, D, e, l, u
```

definiscono il problema da risolvere, mentre gli output sono:

- `x` è una soluzione ottima del problema (2.1);
- `v` è il valore ottimo del problema (2.1);
- `exitflag` descrive la condizione di uscita della funzione `linprog`:

se `exitflag` > 0 allora	`linprog` converge verso `x`
se `exitflag` $= 0$ allora	`linprog` raggiunge il massimo numero di iterazioni senza convergere
se `exitflag` < 0 allora	il problema ha regione ammissibile vuota oppure `linprog` ha commesso un errore

- `output` è una struttura contenente informazioni sull'algoritmo usato da MATLAB;
- `lambda` è una struttura i cui campi contengono le variabili duali ottime corrispondenti ai vincoli del problema (2.1). I campi della struttura sono:

$$\begin{aligned} \text{ineqlin} = &\quad \text{vincoli di disuguaglianza} \\ \text{eqlin} = &\quad \text{vincoli di uguaglianza} \\ \text{upper} = &\quad \text{capacità superiore} \\ \text{lower} = &\quad \text{capacità inferiore} \end{aligned}$$

Esempio 2.1.1. Supponiamo di dover risolvere il problema di programmazione lineare:

$$\begin{cases} \max\ x_1 + 2\,x_2 + 3\,x_3 \\ 3\,x_1 + 4\,x_3 \leq 5 \\ 5\,x_1 + x_2 + 6\,x_3 = 7 \\ 8\,x_1 + 9\,x_3 \geq 2 \\ 0 \leq x_1 \leq 5 \\ 0 \leq x_2 \leq 4 \\ x_3 \geq 0 \end{cases} \tag{2.2}$$

Lo trasformiamo nella forma (2.1):

$$\begin{cases} -\min\ -x_1 - 2\,x_2 - 3\,x_3 \\ 3\,x_1 + 4\,x_3 \leq 5 \\ -8\,x_1 - 9\,x_3 \leq -2 \\ 5\,x_1 + x_2 + 6\,x_3 = 7 \\ 0 \leq x_1 \leq 5 \\ 0 \leq x_2 \leq 4 \\ 0 \leq x_3 \leq +\infty \end{cases}$$

e scriviamo:

```
>> c = [-1; -2; -3];
>> A = [3, 0, 4; -8, 0, -9];
>> b = [5; -2];
>> D = [5, 1, 6];
>> e = 7;
>> l = [0;0;0];
>> u = [5;4;inf];
```

Digitando il comando

```
>> [x, v, exitflag, output, lambda] = linprog(c, A, b, D, e, l, u)
```

si ottengono:

```
x=
    0.0000
    4.0000
    0.5000

fval=
    -9.5000

exitflag =
    1

output =
    iterations:  6
    algorithm:  'large-scale:  interior point'
    cgiterations:  0
    message:  'Optimization terminated.'

lambda =

ineqlin:  [2x1 double]
eqlin:  0.5000
upper:  [3x1 double]
lower:  [3x1 double]
```

quindi la soluzione ottima del problema (2.2) è (0, 4, 0.5) ed il valore ottimo è 9.5.

Per trovare una soluzione ottima del duale, possiamo dire che quella associata al vincolo di uguaglianza è 0.5, quelle associate ai due vincoli di disuguaglianza si ottengono con il comando

```
>> lambda.ineqlin
```

che fornisce la risposta

```
ans =
1.0e-008 *

0.2473
0.3363
```

che possiamo approssimare a 0, le tre variabili associate ai vincoli di *upper bound* sono:

```
>> lambda.upper

  ans =
  0.0000
  1.5000
  0
```

ed infine le tre variabili associate ai vincoli di *lower bound* sono:

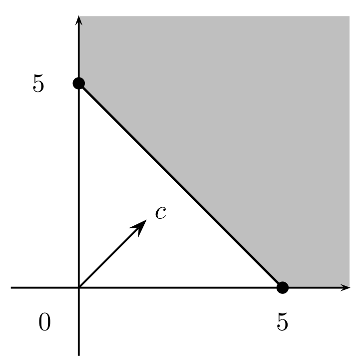
```
>> lambda.lower

ans =
1.5000
0.0000
0.0000
```

◇

Per risolvere problemi di programmazione lineare la funzione `linprog` utilizza, di *default*, un metodo a punti interni invece del simplesso. Quindi potrebbe non fornire un vertice ottimo nel caso in cui ci siano infinite soluzioni ottime. Ad esempio il problema

$$\begin{cases} \min\ x_1 + x_2 \\ x_1 + x_2 \geq 5 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases} \tag{2.3}$$

ammette infinite soluzioni ottime che formano il segmento di estremi $(0, 5)$ e $(5, 0)$.



Se poniamo

```
>> c = [1; 1];
>> A = [-1, -1];
>> b = -5;
>> l = [0;0];
```

il comando

```
>> x = linprog(c, A, b, [ ], [ ], l, [ ])
```

fornisce la soluzione ottima

```
x=
    2.5000
    2.5000
```

che non è un vertice del poliedro ammissibile del problema (2.3). Per risolvere il problema con l'algoritmo del simplesso (e trovare quindi un vertice ottimo) è necessario cambiare le opzioni della funzione `linprog` con il seguente comando:

```
>> options = optimset('largescale','off','simplex','on');
```

Ora il comando

```
>> x = linprog(c, A, b, [ ], [ ], l, [ ], [ ], options)
```

fornisce la soluzione ottima

```
x=
    5
    0
```

che è un vertice ottimo del problema (2.3).

2.2 La funzione quadprog

Il comando `quadprog` risolve un problema di programmazione quadratica della forma:

$$\begin{cases} \min \ \frac{1}{2} x^T Q x + q^T x \\ A\, x \leq b \\ Aeq\ x = beq \\ lb \leq x \leq ub \end{cases} \tag{2.4}$$

dove q, x, b, beq, lb, ub sono vettori e Q, A, Aeq sono matrici. Una forma della sua sintassi è

```
x=quadprog(Q,q,A,b,Aeq,beq,lb,ub)
```

che restituisce una soluzione ottima `x` del problema. Analogamente a `linprog` si possono avere più uscite: la forma generale del comando è:

```
[x,fval,exitflag,output,lambda]=quadprog(Q,q,A,b,Aeq,beq,lb,ub)
```

Esempio 2.2.1. Consideriamo il seguente problema quadratico:

$$\begin{cases} \max \ -x_1^2 + x_1 x_2 - 2x_2^2 + 2x_1 + 4x_2 \\ 2x_1 + x_2 \leq 8 \\ x_1 + 2x_2 \geq 2 \\ x \geq 0 \end{cases} \tag{2.5}$$

Lo riportiamo nella forma (2.4):

$$\begin{cases} -\min \ x_1^2 - x_1 x_2 + 2x_2^2 - 2x_1 - 4x_2 \\ 2x_1 + x_2 \leq 8 \\ -x_1 - 2x_2 \leq -2 \\ 0 \leq x_1 \\ 0 \leq x_2 \end{cases} \tag{2.6}$$

che equivale a

$$\begin{cases} -\min \; \frac{1}{2}(x_1 \; x_2)\begin{pmatrix} 2 & -1 \\ -1 & 4 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 2x_1 - 4x_2 \\ 2x_1 + x_2 \leq 8 \\ -x_1 - 2x_2 \leq -2 \\ 0 \leq x_1 \\ 0 \leq x_2 \end{cases}$$

Digitando:

```
>> Q=[2, -1; -1, 4];
>> q=[-2; -4];
>> A=[2, 1; -1, -2];
>> b=[8; -2];
>> lb=[0;0];
>> [x,fval]=quadprog(Q,q,A,b,[ ],[ ],lb,[ ])
```

si ottiene:

```
x=
    1.7143
    1.4286
fval=
    -4.5714
```

Osserviamo che calcolando gli autovalori della matrice Q si ottiene:

```
>> eig(Q)
ans=
    1.5858
    4.4142
```

per cui la matrice Q è definita positiva e di conseguenza x è l'unico punto di minimo del problema (2.6). Pertanto x è l'unico punto di massimo del problema (2.5) con valore ottimo uguale a 4.5714. $\diamond$

2.3 La funzione `fminbnd`

Il comando `fminbnd` risolve un problema unidimensionale del tipo:

$$\begin{cases} \min \; f(x) \\ x \in [a, b], \end{cases}$$

dove $f : \mathbb{R} \to \mathbb{R}$ è la funzione obiettivo e $[a, b]$ è l'intervallo in cui si cerca il minimo.

La forma più semplice del comando è

```
x=fminbnd('f',a,b)
```

che restituisce un valore di x che rende minima la funzione f nell'intervallo $[a, b]$; se si vogliono più uscite, la forma generale del comando è:

```
[x,fval,exitflag,output]=fminbnd('f',a,b)
```

Ad esempio, `fminbnd('cos',0,4)` restituisce il valore minimo della funzione coseno tra 0 e 4. Per trovare il minimo di una funzione più complessa, occorre definirla in un m-file.

Esempio 2.3.1. Se cerchiamo il minimo della funzione $f(x) = (x-3)^2 - 1$ nell'intervallo [0,5], costruiamo il file `f.m`:

```
function y=f(x)
y=(x-3)^2-1;
```

e poi digitiamo

```
>> x=fminbnd('f',0,5)
```

Il risultato è `x=3`. Il formato sintattico `[x,fval]=fminbnd('f',0,5)` restituisce anche il valore ottimo della funzione, cioè `fval=-1`. $\diamond$

2.4 La funzione fminsearch

Il comando `fminsearch` permette di trovare un minimo locale di un problema non lineare non vincolato del tipo

$$\begin{cases} \min \; f(x) \\ x \in \mathbb{R}^n \end{cases}$$

dove $f : \mathbb{R}^n \to \mathbb{R}$ è la funzione obiettivo. La forma più semplice del comando è:

```
x=fminsearch('f',x0)
```

dove `f` è una stringa che contiene il nome della funzione e `x0` è il punto di partenza che utilizza l'algoritmo. La sintassi più generale è:

```
[x,fval,exitflag,output]=fminsearch('f',x0)
```

Esempio 2.4.1. Trovare il minimo della funzione $f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. Definiamo la funzione obiettivo nel file `f.m`:

```
function y=f(x)
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

Scegliamo un punto iniziale, per esempio $(0, 0)$, e digitiamo

```
>> x=fminsearch('f',[0;0])
```

il comando ci restituisce

```
x=

     1
     1
```

◇

2.5 La funzione `fmincon`

Il comando `fmincon` permette di trovare un minimo locale di un problema non lineare vincolato del tipo:

$$\begin{cases} \min\ f(x) \\ A\ x \leq b \\ Aeq\ x = beq \\ g(x) \leq 0 \\ h(x) = 0 \\ lb \leq x \leq ub \end{cases} \tag{2.7}$$

dove x, b, beq, lb, ub sono vettori, A, Aeq sono matrici, g e h sono funzioni vettoriali (cioè ad ogni vettore x associano un vettore) e f è una funzione scalare (cioè ad ogni vettore x associa un numero reale). Le funzioni f, g, h possono essere non lineari.

La forma più semplice del comando è:

```
x=fmincon('f',x0,A,b,Aeq,beq,lb,ub,'vincoli')
```

dove `f` è una stringa che contiene il nome della funzione da minimizzare, il vettore `x0` è un punto iniziale (stima della soluzione) che deve essere fornito dall'utente, mentre `vincoli` è una stringa che contiene il nome della funzione che descrive i vincoli non lineari di uguaglianza e di disuguaglianza. Più in generale la sua sintassi è:

```
[x,fval,exitflag,output,lambda]=fmincon('f',x0,A,b,Aeq,beq,lb,ub,'vincoli')
```

Esempio 2.5.1. Consideriamo il problema:

$$\begin{cases} \max\ x_1 + 3x_2 \\ x_1 + x_2 \geq 2 \\ x_1^2 + 2x_2^2 \leq 15 \\ x_1 x_2 \geq 3 \\ x_1^2 + x_2^2 = 9 \\ x \geq 0 \end{cases} \tag{2.8}$$

Innanzi tutto trasformiamolo nella forma (2.7):

$$\begin{cases} -\min\ -x_1 - 3x_2 \\ -x_1 - x_2 \leq -2 \\ x_1^2 + 2x_2^2 - 15 \leq 0 \\ 3 - x_1 x_2 \leq 0 \\ x_1^2 + x_2^2 - 9 = 0 \\ 0 \leq x \end{cases}$$

definiamo la funzione obiettivo nel file `f.m`:

```
function y=f(x)
y=-x(1)-3*x(2);
```

Poi definiamo i vincoli non lineari nel file `vincoli.m`, ossia definiamo la funzione `vincoli` che ha 2 variabili di uscita, una relativa ai vincoli non lineari di disuguaglianza, l'altra relativa ai vincoli non lineari di uguaglianza:

```
function [g,h]=vincoli(x)
g(1)=x(1)^2+2*x(2)^2-15;
g(2)=3-x(1)*x(2);
h=x(1)^2+x(2)^2-9;
```

Per definire i vincoli lineari scriviamo:

```
>> A=[-1,-1];
>> b=-2;
>> lb=[0;0];
```

Usiamo il vettore $x^0 = (3, 0)$ come punto iniziale dell'algoritmo. Digitando

```
>> [x,fval]=fmincon('f',[3; 0],A,b,[ ],[ ],lb,[ ],'vincoli')
```

si ottiene

```
x=
    1.7321
    2.4495

fval=
    -9.0805
```

Quindi la soluzione ottima del problema (2.8) è $x = (1.7321, 2.4495)$ ed il valore ottimo è 9.0805. $\diamond$

Capitolo 3

Risoluzione di problemi di ricerca operativa mediante MATLAB

3.1 Problemi di programmazione lineare

Esempio 3.1.1. Una fabbrica di detersivi produce due tipi di saponi che passano attraverso 4 fasi di lavorazione: le ore necessarie per ogni fase di lavorazione per quintale di prodotto sono riportate nella tabella che segue, in cui compaiono anche le ore mensili a disposizione per ciascuna fase.

	Fase a	Fase b	Fase c	Fase d
Sapone A	1.5	1.5	3	2.5
Sapone B	2.5	2	3	4
Ore mensili disponibili	155	200	240	400

Il guadagno netto è di 2100 euro per quintale di sapone A e 3400 euro per quintale di sapone B. Quanti quintali dei saponi A e B bisogna produrre per massimizzare il guadagno?

Se indichiamo con x_1 il numero di quintali prodotti del sapone A e con x_2 il numero di quintali prodotti del sapone B, il problema si può formulare come segue:

$$\begin{cases} \max\ 2100\ x_1 + 3400\ x_2 \\ 1.5\ x_1 + 2.5\ x_2 \leq 155 \\ 1.5\ x_1 + 2\ x_2 \leq 200 \\ 3\ x_1 + 3\ x_2 \leq 240 \\ 2.5\ x_1 + 4\ x_2 \leq 400 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

Per risolverlo con la funzione `linprog` dobbiamo trasformarlo in un problema di minimo:

$$\begin{cases} -\min\ -2100\ x_1 - 3400\ x_2 \\ 1.5\ x_1 + 2.5\ x_2 \leq 155 \\ 1.5\ x_1 + 2\ x_2 \leq 200 \\ 3\ x_1 + 3\ x_2 \leq 240 \\ 2.5\ x_1 + 4\ x_2 \leq 400 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

COMANDI DI MATLAB

`funzione obiettivo`	`c=[-2100;-3400]`

`vincoli`	`A=[1.5, 2.5; 1.5, 2; 3, 3; 2.5, 4]`
	`b=[155; 200; 240; 400]`
	`lb=[0; 0]`

`Comando risolutivo`	`[x,fval]=linprog(c,A,b,[ ],[ ],lb,[ ])`

SOLUZIONI

Soluzione ottima	(45, 35)
Valore ottimo	213500

ANALISI DI SENSIBILITÀ

Supponendo di portare a 168 le ore mensili a disposizione per la fase a, determinare la nuova strategia di produzione ottima, il nuovo guadagno ed il costo per ogni ora lavorativa aggiuntiva per la fase a affinché la nuova strategia sia conveniente rispetto a quella vecchia.

Nuova soluzione ottima	(32,48)
Nuovo valore ottimo	230400
Costo per ogni ora aggiuntiva	< 1300

◇

Esempio 3.1.2. Una azienda deve produrre almeno 500 litri di un cocktail utilizzando tre tipi di succhi di frutta S_1, S_2 e S_3. La disponibilità ed il costo dei diversi tipi di succhi di frutta sono indicati nella seguente tabella:

Tipo di succo	Disponibilità max in litri	Costo in euro per litro
S_1	560	1.5
S_2	260	1
S_3	950	4

La dose di miscelatura per il cocktail è: non più del 25 % di S_1, non meno del 50 % di S_2 e non meno del 40 % di S_3. Si vuole determinare la combinazione dei tre tipi di succhi di frutta che minimizzi la spesa.

Indicando con x_1 il numero di litri del succo S_1, con x_2 il numero di litri del succi S_2 e con x_3 il numero di litri del succo S_3, si ha il seguente modello di programmazione lineare:

$$\begin{cases} \min\ 1.5\ x_1 + x_2 + 4\ x_3 \\ x_1 \leq 0.25(x_1 + x_2 + x_3) \\ x_2 \geq 0.5(x_1 + x_2 + x_3) \\ x_3 \geq 0.4(x_1 + x_2 + x_3) \\ x_1 + x_2 + x_3 \geq 500 \\ 0 \leq x_1 \leq 560 \\ 0 \leq x_2 \leq 260 \\ 0 \leq x_3 \leq 950 \end{cases}$$

`COMANDI DI MATLAB`

`funzione obiettivo`	`c=[1.5; 1; 4]`

`vincoli`	`A=[3, -1, -1; 1, -1, 1; 2, 2, -3; -1, -1, -1]`
	`b=[0; 0; 0; -500]`
	`lb=[0; 0; 0]`
	`ub=[560; 260; 950]`

`Comando risolutivo`	`[x,fval]=linprog(c,A,b,[ ],[ ],lb,ub)`

SOLUZIONI

Soluzione ottima	(40, 260, 200)
Valore ottimo	1120

ANALISI DI SENSIBILITÀ

Supponendo che la massima disponibilità del secondo succo di frutta aumenti di 40 litri, diventando quindi di 300 litri, determinare la nuova soluzione ottima e la nuova spesa minima.

Nuova soluzione ottima	(0,300,200)
Nuova spesa minima	1100

◇

Esempio 3.1.3. Un'impresa produce un bene in 2 stabilimenti, situati a Pontedera e a Rosignano. La produzione viene immagazzinata in 2 depositi a Pisa e a Livorno e poi distribuita alla rete di vendita al dettaglio. I dati riguardano il costo unitario di trasporto, la capacità produttiva massima settimanale dei 2 stabilimenti e le statistiche di vendita settimanale di ognuno dei 2 depositi.

	Pisa	Livorno	Capacità produttiva massima
Pontedera	10	30	105
Rosignano	35	6	80
Vendita	110	46	

Indichiamo con x_1 la quantità di merce spedita da Pontedera a Pisa, con x_2 quella spedita da Pontedera a Livorno, con x_3 quella da Rosignano a Pisa e con x_4 quella da Rosignano a Livorno. Il modello di programmazione lineare è il seguente:

$$\begin{cases} \min\ 10\ x_1 + 30\ x_2 + 35\ x_3 + 6\ x_4 \\ x_1 + x_2 \leq 105 \\ x_3 + x_4 \leq 80 \\ x_1 + x_3 = 110 \\ x_2 + x_4 = 46 \\ x \geq 0 \end{cases}$$

COMANDI DI MATLAB

funzione obiettivo	`c=[10; 30; 35; 6]`

vincoli	`A=[1, 1, 0, 0; 0, 0, 1, 1]`
	`b=[105; 80]`
	`Aeq=[1, 0, 1, 0; 0, 1, 0, 1]`
	`beq=[110; 46]`
	`lb=[0; 0; 0; 0]`

Comando risolutivo	`[x,fval]=linprog(c,A,b,Aeq,beq,lb,[ ])`

SOLUZIONI

Soluzione ottima	(105, 0, 5, 46)
Valore ottimo	1501

ANALISI DI SENSIBILITÀ

Supponendo che la capacità produttiva dello stabilimento di Pontedera diventi 110, determinare la nuova soluzione ottima e la nuova spesa minima.

Nuova soluzione ottima	(110, 0, 0, 46)
Nuova spesa minima	1376

◇

Esempio 3.1.4. Un'industria siderurgica ha tre stabilimenti che necessitano di 50, 70 e 60 tonnellate di acciaio a settimana. L'acciaio può essere acquistato da due fornitori. Il primo può fornire al massimo 30 tonnellate a settimana a ciascun stabilimento, mentre il secondo può fornire al massimo 40 tonnellate a settimana a ciascun stabilimento. Inoltre, a causa di altri impegni, il primo fornitore non può fornire, in totale, più di 100 tonnellate a settimana e deve fornire non meno di 25 tonnellate a settimana al terzo stabilimento. La seguente tabella indica i costi unitari di trasporto (euro/ton) dai fornitori agli stabilimenti.

Fornitori	Stabilimenti		
	1	2	3
1	2	3	5
2	3	3.6	3.2

Determinare come si deve rifornire l'industria per minimizzare il costo di trasporto.

Indichiamo con x_1, x_2, x_3 le tonnellate di acciaio spedite rispettivamente dal primo fornitore ai tre stabilimenti e con x_4, x_5, x_6 le tonnellate di acciaio spedite rispettivamente dal secondo fornitore ai tre stabilimenti. Il modello di programmazione lineare è il seguente:

$$\begin{cases} \min\ 2\ x_1 + 3\ x_2 + 5\ x_3 + 3\ x_4 + 3.6\ x_5 + 3.2\ x_6 \\ x_1 + x_4 = 50 \\ x_2 + x_5 = 70 \\ x_3 + x_6 = 60 \\ x_1 + x_2 + x_3 \leq 100 \\ 0 \leq x_1 \leq 30 \\ 0 \leq x_2 \leq 30 \\ 25 \leq x_3 \leq 30 \\ 0 \leq x_4 \leq 40 \\ 0 \leq x_5 \leq 40 \\ 0 \leq x_6 \leq 40 \end{cases}$$

COMANDI DI MATLAB

funzione obiettivo	`c=[2; 3; 5; 3; 3.6; 3.2]`

vincoli	`A=[1, 1, 1, 0, 0, 0]`
	`b=100`
	`Aeq=[1, 0, 0, 1, 0, 0; 0, 1, 0, 0, 1, 0; 0, 0, 1, 0, 0, 1;]`
	`beq=[50; 70; 60]`
	`lb=[0; 0; 25; 0; 0; 0]`
	`ub=[30; 30; 30; 40; 40; 40]`

Comando risolutivo	`[x,fval]=linprog(c,A,b,Aeq,beq,lb,ub)`

SOLUZIONI

Soluzione ottima	(30, 30, 25, 20, 40, 35)
Valore ottimo	591

ANALISI DI SENSIBILITÀ

Supponendo che il primo fornitore possa spedire al massimo 35 tonnellate a settimana ai tre stabilimenti, determinare la nuova soluzione ottima e la nuova spesa minima.

Nuova soluzione ottima	(35, 35, 25, 15, 35, 35)
Nuova spesa minima	583

◇

Esempio 3.1.5. Uno stabilimento produce tre diversi tipi di pitture per l'edilizia: una economica, una normale ed una di extra qualità. Ogni pittura viene lavorata da tre linee di produzione A, B e C. I tempi (in minuti) necessari alla lavorazione di ogni quintale, la disponibilità delle linee di produzione ed i profitti dei tre tipi di pittura sono indicate in tabella:

	Economica	Normale	Extra	Disponibilità
A	20	30	62	480
B	31	42	51	480
C	16	81	10	300
Profitto	100	150	220	

La quantità della pittura extra deve essere non più del 20% del totale, mentre quella economica deve essere non meno del 40% del totale. Determinare le quantità dei tre diversi tipi di pittura in modo da massimizzare il profitto.

Indichiamo con x_1 la quantità prodotta di pittura economica, con x_2 quella di pittura normale e con x_3 quella di pittura extra. Il modello di programmazione lineare è il seguente:

$$\begin{cases} \max\ 100\ x_1 + 150\ x_2 + 220\ x_3 \\ 20\ x_1 + 30\ x_2 + 62\ x_3 \leq 480 \\ 31\ x_1 + 42\ x_2 + 51\ x_3 \leq 480 \\ 16\ x_1 + 81\ x_2 + 10\ x_3 \leq 300 \\ x_3 \leq 0.2(x_1 + x_2 + x_3) \\ x_1 \geq 0.4(x_1 + x_2 + x_3) \\ x \geq 0 \end{cases}$$

che equivale a:

$$\begin{cases} -\min\ -100\ x_1 - 150\ x_2 - 220\ x_3 \\ 20\ x_1 + 30\ x_2 + 62\ x_3 \leq 480 \\ 31\ x_1 + 42\ x_2 + 51\ x_3 \leq 480 \\ 16\ x_1 + 81\ x_2 + 10\ x_3 \leq 300 \\ -0.2\ x_1 - 0.2\ x_2 + 0.8\ x_3 \leq 0 \\ -0.6\ x_1 + 0.4\ x_2 + 0.4\ x_3 \leq 0 \\ x \geq 0 \end{cases}$$

COMANDI DI MATLAB

`funzione obiettivo`	`c=[-100; -150; -220]`

`vincoli`	`A=[20, 30, 62; 31, 42, 51; 16, 81, 10; -0.2, -0.2, 0.8; -0.6, 0.4, 0.4]`
	`b=[480; 480; 300; 0; 0]`
	`lb=[0; 0; 0]`

`Comando risolutivo`	`[x,fval]=linprog(c,A,b,[ ],[ ],lb,[ ])`

SOLUZIONI

Soluzione ottima	(8.9594, 1.6078, 2.6418)
Valore ottimo	1718

ANALISI DI SENSIBILITÀ

Supponendo che i minuti a disposizione della linea di produzione C diventino 360, determinare la nuova soluzione ottima ed il nuovo profitto ottimo.

Nuova soluzione ottima	(7.7152, 2.6020, 2.5793)
Nuovo profitto ottimo	1729

◇

3.2 Problemi di flusso su reti

Esempio 3.2.1. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	Produzione
A		8–15	5–9				11
B			7–10	2–2			5
C				15–12	9–15		4
D					10–15	2–2	
E						10–14	
F							
Richiesta				7	5	8	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

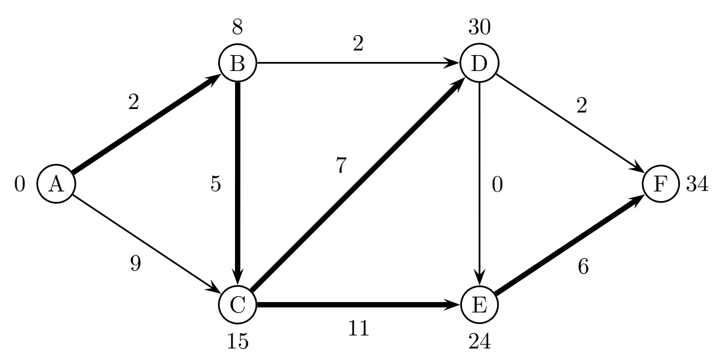
Il problema si può modellizzare come un flusso di costo minimo:

$$\begin{cases} \min\ c^T x \\ Ex = b \\ 0 \le x \le u \end{cases}$$

dove

$$\begin{aligned} x &= (x_{AB},\ x_{AC},\ x_{BC},\ x_{BD},\ x_{CD},\ x_{CE},\ x_{DE},\ x_{DF},\ x_{EF})^T \\ c &= (8,\ 5,\ 7,\ 2,\ 15,\ 9,\ 10,\ 2,\ 10)^T \\ b &= (-11,\ -5,\ -4,\ 7,\ 5,\ 8)^T \\ u &= (15,\ 9,\ 10,\ 2,\ 12,\ 15,\ 15,\ 2,\ 14)^T \end{aligned}$$

ed E è la matrice di incidenza del grafo. Risolvendo tale problema mediante la funzione `linprog`, otteniamo il flusso ottimo (riportato sul seguente grafo) ed il valore ottimo. A partire dagli archi non vuoti e non saturi, costruiamo l'albero di copertura B (indicato in grassetto), da cui ricaviamo i potenziali ai nodi avendo fissato a zero il potenziale del nodo A.



SPESA TOTALE = 368

ANALISI DI SENSIBILITÀ

Supponiamo che la richiesta del nodo E aumenti di 3 unità. Determinare lo stabilimento in cui conviene aumentare la produzione di 3 unità ed indicare la nuova spesa totale.

Aumentando la produzione di 5 unità nello stabilimento A si avrebbe una spesa totale di 440, aumentandola nello stabilimento B la spesa sarebbe di 416, mentre per lo stabilimento C sarebbe di 395, quindi la soluzione è:

Stabilimento	C
Nuova spesa totale	395

◇

Esempio 3.2.2. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	G	Produzione
A		8–40	2–40	4–60				60
B					6–40			
C		7–35			4–40	5–20	5–20	
D			5–60					10
E								
F				4–30				
G					9–40	1–35		10
Richiesta		15			60	5		

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

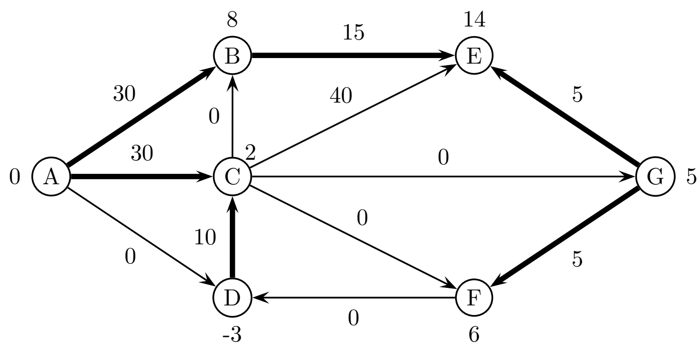
Il problema si può modellizzare come un flusso di costo minimo:

$$\begin{cases} \min \ c^T x \\ Ex = b \\ 0 \le x \le u \end{cases}$$

dove

$$\begin{aligned} x &= (x_{AB},\ x_{AC},\ x_{AD},\ x_{BE},\ x_{CB},\ x_{CE},\ x_{CF},\ x_{CG},\ x_{DC},\ x_{FD},\ x_{GE},\ x_{GF})^T \\ c &= (8,\ 2,\ 4,\ 6,\ 7,\ 4,\ 5,\ 5,\ 5,\ 4,\ 9,\ 1)^T \\ b &= (-60,\ 15,\ 0,\ -10,\ 60,\ 5,\ -10)^T \\ u &= (40,\ 40,\ 60,\ 40,\ 35,\ 40,\ 20,\ 20,\ 60,\ 30,\ 40,\ 35)^T \end{aligned}$$

ed E è la matrice di incidenza del grafo. Risolvendo tale problema mediante la funzione `linprog`, otteniamo il flusso ottimo (riportato sul seguente grafo) ed il valore ottimo. A partire dagli archi non vuoti e non saturi, costruiamo l'albero di copertura B (indicato in grassetto), da cui ricaviamo i potenziali ai nodi avendo fissato a zero il potenziale del nodo A.



SPESA TOTALE = 650

ANALISI DI SENSIBILITÀ

Supponiamo che il costo per aumentare di 5 unità la capacità dell'arco (C, E) sia 30. Dire se conviene effettuare tale incremento e, in caso affermativo, determinare la nuova soluzione ottima e la nuova spesa complessiva.

Conviene	SI
Nuova soluzione ottima	(25, 35, 0, 10, 0, 45, 0, 0, 10, 0, 5, 5)
Nuova spesa totale	640

◇

Esempio 3.2.3. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	Produzione
A		8–15	5–12	1–2	1–5		8
B			7–30	8–21			6
C					15–15		9
D						10–22	
E				10–21		8–14	
F							
Richiesta				7	5	11	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

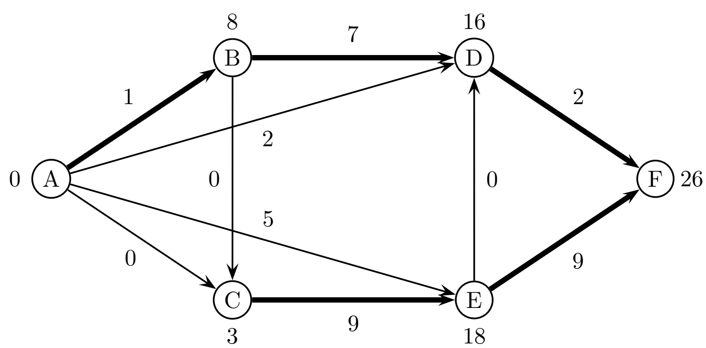
Il problema si può modellizzare come un flusso di costo minimo:

$$\begin{cases} \min\ c^T x \\ Ex = b \\ 0 \leq x \leq u \end{cases}$$

dove

$$x = (x_{AB},\ x_{AC},\ x_{AD},\ x_{AE},\ x_{BC},\ x_{BD},\ x_{CE},\ x_{DF},\ x_{ED},\ x_{EF})^T$$
$$c = (8,\ 5,\ 1,\ 1,\ 7,\ 8,\ 15,\ 10,\ 10,\ 8)^T$$
$$b = (-8,\ -6,\ -9,\ 7,\ 5,\ 11)^T$$
$$u = (15,\ 12,\ 2,\ 5,\ 30,\ 21,\ 15,\ 22,\ 21,\ 14)^T$$

ed E è la matrice di incidenza del grafo. Risolvendo tale problema mediante la funzione `linprog`, otteniamo il flusso ottimo (riportato sul seguente grafo) ed il valore ottimo. A partire dagli archi non vuoti e non saturi, costruiamo l'albero di copertura B (indicato in grassetto), da cui ricaviamo i potenziali ai nodi avendo fissato a zero il potenziale del nodo A.



SPESA TOTALE = 298

ANALISI DI SENSIBILITÀ

Supponiamo che la richiesta del nodo F aumenti di 5 unità. Determinare lo stabilimento in cui conviene aumentare la produzione di 5 unità ed indicare la nuova spesa totale.

Aumentando la produzione di 5 unità nello stabilimento A si avrebbe una spesa totale di 428, aumentandola nello stabilimento B la spesa sarebbe di 388, mentre per lo stabilimento C sarebbe di 413, quindi la soluzione è:

Stabilimento	B
Nuova spesa totale	388

◇

Esempio 3.2.4. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	Produzione
A		8–10	5–15			10
B				9–19	7–17	3
C		15–15			9–19	7
D					10–8	
E						
Richiesta				14	6	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

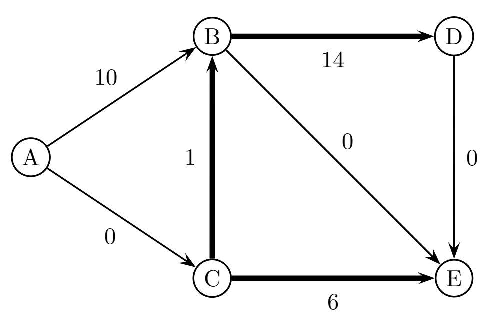
Il problema si può modellizzare come un flusso di costo minimo:

$$\begin{cases} \min\ c^T x \\ Ex = b \\ 0 \leq x \leq u \end{cases}$$

dove

$$\begin{aligned} x &= (x_{AB},\ x_{AC},\ x_{BD},\ x_{BE},\ x_{CB},\ x_{CE},\ x_{DE})^T \\ c &= (8,\ 5,\ 9,\ 7,\ 15,\ 9,\ 10)^T \\ b &= (-10,\ -3,\ -7,\ 14,\ 6)^T \\ u &= (10,\ 15,\ 19,\ 17,\ 15,\ 19,\ 8)^T \end{aligned}$$

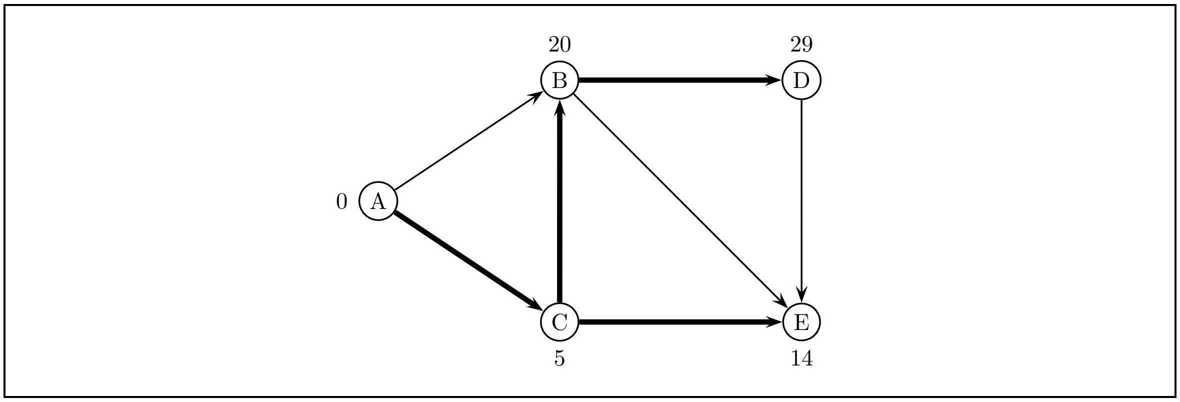
ed E è la matrice di incidenza del grafo. Risolvendo tale problema mediante la funzione `linprog`, otteniamo il flusso ottimo (riportato sul seguente grafo) ed il valore ottimo. In grassetto sono indicati gli archi non vuoti e non saturi.



SPESA TOTALE = 275

Poiché gli archi non vuoti e non saturi sono 3, il flusso ottimo è degenere. Quindi, per calcolare i potenziali ottimi dobbiamo ricavare una base associata al flusso ottimo che sia duale ammissibile, cioè per formare l'albero di copertura B è necessario aggiungere un arco a quelli in grassetto. Tale arco non può essere né l'arco (B,E), né l'arco (D,E) perché formano un ciclo con gli archi in grassetto.

Se formiamo l'albero B aggiungendo l'arco (A,C) agli archi in grassetto, possiamo ricavare i potenziali:



Per controllare se tali potenziali sono ottimi, calcoliamo i costi ridotti degli archi non in B:

$$\begin{aligned}\overline{c}_{AB} &= 8 + 0 - 20 = -12 < 0\\ \overline{c}_{BE} &= 7 + 20 - 14 = 13 > 0\\ \overline{c}_{DE} &= 10 + 29 - 14 = 25 > 0\end{aligned}$$

Poiché il costo ridotto dell'arco saturo (A,B) è negativo ed i costi ridotti degli archi vuoti (B,E) e (D,E) sono positivi, allora i potenziali indicati in figura sono ottimi.

ANALISI DI SENSIBILITÀ

Supponiamo che la richiesta del nodo E aumenti di 4 unità. Determinare lo stabilimento in cui conviene aumentare la produzione di 4 unità ed indicare la nuova spesa totale.

Aumentando la produzione di 4 unità nello stabilimento A si avrebbe una spesa totale di 331, aumentandola nello stabilimento B la spesa sarebbe di 287, mentre per lo stabilimento C sarebbe di 311, quindi la soluzione è:

Stabilimento	B
Nuova spesa totale	287

◇

3.3 Problemi di programmazione non lineare

Esempio 3.3.1. Risolvere il seguente problema:

$$\begin{cases} \min\ 5\ x_1^2 + 3\ x_2^2 - 2\ x_1 x_2 + \log(x_1^2 + 4\ x_2^4 + 1) + \sin(x_2 - x_1) \\ (x_1, x_2) \in \mathbb{R}^2 \end{cases}$$

COMANDI DI MATLAB

funzione obiettivo	`function y=f(x)`
	`y=5*x(1)^2+3*x(2)^2-2*x(1)*x(2)+log(x(1)^2+`
	`4*x(2)^4+1)+sin(x(2)-x(1));`

punto iniziale	`x0 = [3; 6]`
Comando risolutivo	`[x,fval] = fminsearch('f',x0)`

SOLUZIONI

Soluzione ottima	(0.30590, -0.1370)
Valore ottimo	-0.1

◇

Esempio 3.3.2. Risolvere il seguente problema:

$$\begin{cases} \min\ 5\ x_1^2 + 10\ x_2^2 + 10\ x_3^2 + 14\ x_1x_2 + 12\ x_1x_3 + 16\ x_2x_3 + x_1 + 3\ x_2 + 5\ x_3 + 20 \\ x_1 + x_3 \leq 2 \\ 2\ x_1 - 2\ x_2 - x_3 \leq 1 \\ 2\ x_1 - 2\ x_2 + 4\ x_3 = 3 \\ x \geq 0. \end{cases}$$

COMANDI DI MATLAB

funzione obiettivo	`Q = [10, 14, 12; 14, 20 16; 12, 16, 20]` `q = [1; 3; 5]`

vincoli lineari	`A = [1, 0, 1; 2, -2, -1]`
	`b = [2; 1]`
	`Aeq = [2, -2, 4]`
	`beq = 3`
	`lb = [0; 0; 0]`

Comando risolutivo	`[x,fval,exitflag,output,lambda]=` `quadprog(Q,q,A,b,Aeq,beq,lb,[ ])`

SOLUZIONI

Soluzione ottima	(0, 0, 0.75)
Valore ottimo	29.375
Moltiplicatori	lambda.ineqlin = (0, 0) lambda.eqlin = -5 lambda.lower = (0, 25, 0)

◇

Esempio 3.3.3. Risolvere il seguente problema:

$$\begin{cases} \min\ 2\ x_1^2 + x_2^3 + 7\ x_3^2 + \log(x_4+1) + x_5 \\ 5\ x_1^2 + 2\ x_2^2 + x_3^3 \leq 8 \\ x_1 + x_2 - x_3 = 1 \\ x_2 \leq 1 \\ x_5 \leq 5 \\ x \geq 0. \end{cases}$$

COMANDI DI MATLAB

funzione obiettivo	`function y=obiettivo(x)` `y=2*x(1)^2 + x(2)^3 + 7*x(3)^2 + log(x(4)+1) + x(5);`

vincoli lineari	`A = [0, 1, 0, 0, 0; 0, 0, 0, 0, 1]`
	`b = [1; 5]`
	`Aeq = [1, 1, -1, 0, 0]`
	`beq = 1`
	`lb = [0; 0; 0; 0; 0]`

vincoli non lineari	`function [g,h]=vincoli(x)`
	`g=5*x(1)^2 + 2*x(2)^2 + x(3)^3 - 8;`
	`h=0;`

punto iniziale	`x0 = [1; 1; 1; 1; 1]`
Comando risolutivo	`[x,fval,exitflag,output,lambda]=` `fmincon('obiettivo',x0,A,b,Aeq,beq,lb,[ ],'vincoli')`

SOLUZIONI

Soluzione ottima	(0.3333, 0.6667, 0, 0, 0)
Valore ottimo	0.5185
Moltiplicatori	lambda.ineqlin = (0, 0) lambda.eqlin = -1.3338 lambda.lower = (0, 0, 1.3338, 1, 1) lambda.ineqnonlin = 0

◇

Capitolo 4

Esercizi

4.1 Problemi di programmazione lineare

Esercizio 4.1.1. Un'azienda produce hamburger, ognuno dei quali deve pesare almeno 100 grammi, costituiti da carne di manzo, carne di maiale e carne di pollo. In ogni hamburger ci devono essere non più di 26 grammi di grasso e almeno 3 grammi di proteine. Nella tabella che segue sono indicate, per ogni tipo di carne, le percentuali di grasso e di proteine contenute ed il costo al grammo.

	Carne di manzo	Carne di maiale	Carne di pollo
Percentuale di grasso	20	32	15
Percentuale di proteine	8	5	3
Costo per ogni grammo	6	3.6	4

Determinare da quanti grammi di carne di manzo, di maiale e di pollo deve essere formato ogni hamburger in modo da minimizzare la spesa totale.

`COMANDI DI MATLAB`

`funzione obiettivo`	
`vincoli`	
`Comando risolutivo`	

SOLUZIONI

Soluzione ottima	
Valore ottimo	

ANALISI DI SENSIBILITÀ

Supponendo che ogni hamburger debba contenere almeno 5 grammi di proteine, determinare la nuova soluzione ottima e la nuova spesa totale.

Nuova soluzione ottima	
Nuova spesa totale	

Esercizio 4.1.2. Per la produzione di un bene, un'impresa può usare tre diversi procedimenti P_1, P_2 e P_3, ognuno dei quali richiede l'uso di tre macchine A, B e C. In tabella sono indicate le ore di utilizzo di ogni macchina da parte di ogni procedimento, le ore totali disponibili per ogni macchina ed il profitto corrispondente ad ogni procedimento:

Macchina	P_1	P_2	P_3	Disponibilità
A	2	1	3	50
B	4	2	3	50
C	3	4	2	50
Profitto	15	18	10	

Determinare le ore di utilizzo dei tre procedimenti in modo da massimizzare il profitto.

`COMANDI DI MATLAB`

`funzione obiettivo`	
`vincoli`	
`Comando risolutivo`	

SOLUZIONI

Soluzione ottima	
Valore ottimo	

ANALISI DI SENSIBILITÀ

Supponendo che il profitto derivante dal primo procedimento sia 14, determinare la nuova soluzione ottima ed il nuovo profitto.

Nuova soluzione ottima	
Nuovo profitto	

4.2 Problemi di flusso su reti

Esercizio 4.2.1. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	Produzione
A		8–14	3–8				11
B			7–15	1–1			5
C				16–16	11–18		4
D					10–15	1–1	
E						10–14	
F							
Richiesta				7	5	8	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

SPESA TOTALE =

ANALISI DI SENSIBILITÀ

Supponiamo che la richiesta del nodo D aumenti di 4 unità. Determinare lo stabilimento in cui conviene aumentare la produzione di 3 unità ed indicare la nuova spesa totale.

Stabilimento	
Nuova spesa totale	

Esercizio 4.2.2. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	Produzione
A		8–15	5–9	7–14			7
B			1–5		11–16		3
C					1–5	9–14	
D			10–4			11–12	
E						10–14	
F							
Richiesta				2	2	6	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

SPESA TOTALE =

ANALISI DI SENSIBILITÀ

Supponiamo che la richiesta del nodo F aumenti di 4 unità. Determinare lo stabilimento in cui conviene aumentare la produzione di 4 unità ed indicare la nuova spesa totale.

Stabilimento	
Nuova spesa totale	

Esercizio 4.2.3. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	Produzione
A		8–15	5–9	7–14			9
B					11–16		5
C		2–1			1–3	9–14	
D			10–4			11–12	
E						10–14	
F							
Richiesta				5	3	6	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

SPESA TOTALE =

ANALISI DI SENSIBILITÀ

Supponiamo che la richiesta del nodo E aumenti di 10 unità. Determinare lo stabilimento in cui conviene aumentare la produzione di 10 unità ed indicare la nuova spesa totale.

Stabilimento	
Nuova spesa totale	

Esercizio 4.2.4. Una ditta deve distribuire merce su una rete secondo le seguenti specifiche e disponibilità, cercando di minimizzare la spesa totale di spedizione.

Costo–Capacità	A	B	C	D	E	F	Produzione
A		18–15	5–19				11
B				7–17			15
C		2–20		15–12	9–15		4
D					10–15	6–2	
E						12–13	
F							
Richiesta				17	5	8	

Rappresentare su un grafo la distruzione ottima della merce ed i corrispondenti potenziali. Specificare inoltre la spesa totale.

SPESA TOTALE =

ANALISI DI SENSIBILITÀ

Supponiamo che il costo per aumentare di 3 unità la capacità dell'arco (B, D) sia 10. Dire se conviene effettuare tale incremento e, in caso affermativo, determinare la nuova soluzione ottima e la nuova spesa complessiva.

Stabilimento	
Nuova spesa complessiva	

4.3 Problemi di programmazione non lineare

Esercizio 4.3.1. Risolvere il seguente problema:

$$\begin{cases} \min\ 2\,x_1^2 + x_2^2 - 2\,x_1\,x_2 + 3\,x_1 - 2\,x_2 + 10 \\ x_1 + 5\,x_2 \leq 13 \\ 2\,x_1 + x_2 \leq 8 \\ x_1 - 5\,x_2 \leq 4 \\ 4\,x_1 + x_2 \geq -5 \end{cases}$$

COMANDI DI MATLAB

`funzione obiettivo`	
`vincoli`	
`Comando risolutivo`	

SOLUZIONI

Soluzione ottima	
Valore ottimo	
Moltiplicatori	

Esercizio 4.3.2. Risolvere il seguente problema:

$$\begin{cases} \min \ e^{x_1^2+x_2^2+x_3^2} + \sin\left(\frac{5\ x_1\, x_3}{1+x_2^4}\right) \\ (x_1, x_2, x_3) \in \mathbb{R}^3 \end{cases}$$

COMANDI DI MATLAB

`funzione obiettivo`	
`punto iniziale`	`x0 = [-2; 5; -10]`
`Comando risolutivo`	

SOLUZIONI

Soluzione ottima	
Valore ottimo	

Esercizio 4.3.3. Risolvere il seguente problema:

$$\begin{cases} \max\ 5\ x_1 + 7\ x_2 \\ x_2 \leq \frac{1}{2}\ x_1 \\ x_2 \geq -\frac{1}{2}\ x_1 \\ x_1^2 + x_2^2 \leq 16 \\ (x_1 - 5)^2 + x_2^2 \leq 9 \end{cases}$$

COMANDI DI MATLAB

funzione obiettivo	
vincoli lineari	
vincoli non lineari	
punto iniziale	x0 = [4; 0]
Comando risolutivo	

SOLUZIONI

Soluzione ottima	
Valore ottimo	
Moltiplicatori	

Soluzioni degli esercizi

Problemi di programmazione lineare

Esercizio 4.1.1.

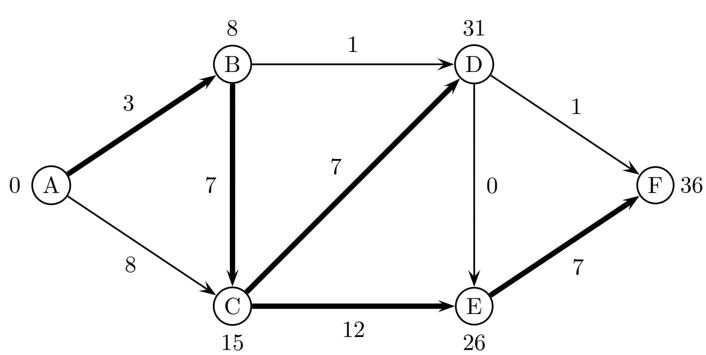
SOLUZIONI	Soluzione ottima	(0, 64.7059, 35.2941)
	Valore ottimo	374.1176
ANALISI DI SENSIBILITÀ	Nuova soluzione ottima	(16, 60, 24)
	Nuovo spesa totale	408

Esercizio 4.1.2.

SOLUZIONI	Soluzione ottima	(10, 5, 0)
	Valore ottimo	240
ANALISI DI SENSIBILITÀ	Nuova soluzione ottima	(0, 6.25, 12.5)
	Nuovo profitto	237.5

Problemi di flusso su reti

Esercizio 4.2.1.

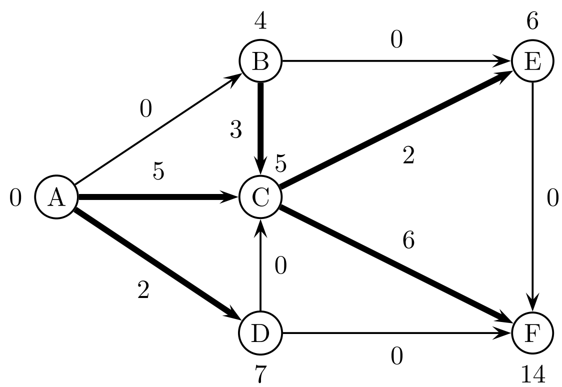


SPESA TOTALE = 413

ANALISI DI SENSIBILITÀ

Stabilimento	C
Nuova spesa totale	477

Esercizio 4.2.2.

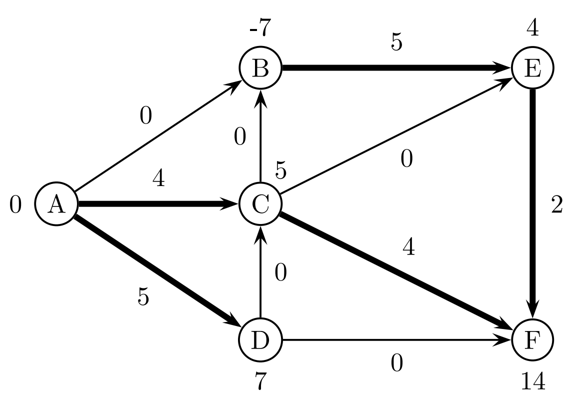


SPESA TOTALE = 98

ANALISI DI SENSIBILITÀ

Stabilimento	A
Nuova spesa totale	154

Esercizio 4.2.3.

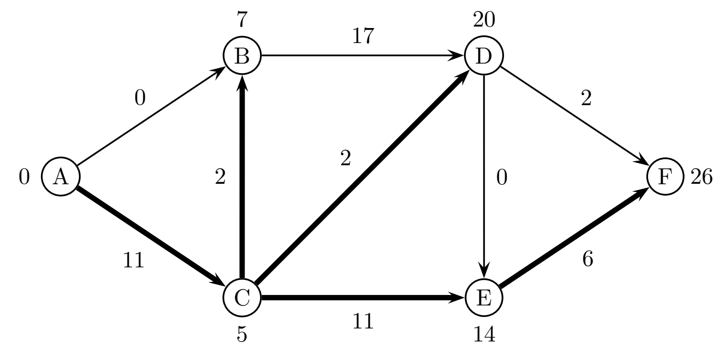


SPESA TOTALE = 166

ANALISI DI SENSIBILITÀ

Stabilimento	B
Nuova spesa totale	276

Esercizio 4.2.4.



SPESA TOTALE = 391

ANALISI DI SENSIBILITÀ

Conviene	SI
Nuova soluzione ottima	(0, 11, 19, 4, 0, 11, 0, 2, 6)
Nuova spesa complessiva	389

Problemi di programmazione non lineare

Esercizio 4.3.1.

SOLUZIONI

Soluzione ottima	(-1.24, -0.04)
Valore ottimo	7.03
Moltiplicatori	lambda.ineqlin=(0, 0, 0, 0.46)

Esercizio 4.3.2.

SOLUZIONI

Soluzione ottima	(-0.4352, 0, 0.4352)
Valore ottimo	0.6489

Esercizio 4.3.3.

SOLUZIONI

Soluzione ottima	(3.5777, 1.7889)
Valore ottimo	30.4105
Moltiplicatori	lambda.ineqlin=(3.6006, 0) lambda.ineqnonlin=(0.9504, 0)