Capitolo 3

■ 3.1 Liste e vettori

Puo' essere molto utile raccogliere insieme singoli oggetti e trattarli come uno solo. Una lista e' una successione ordinata di elementi racchiusa tra { } (in realta' l'abbiamo fino ad ora usata spesso!!). La lista {3,5,1} e' una collezione di 3 oggetti cosi' come la lista {3,"a",a1}

Le **Built-in Functions,** se applicate a una lista, restituiscono la lista dei valori che la funzione assume su ogni elemento della lista (cioe' sono **Listable**)

```
lista = {1, 2, 3, 4}
Cos[Pi / lista]
Sqrt[lista]
```

Puo' essere utile imparare a manipolare gli elementi di una lista. Se abbiamo una lista possiamo facilmente aggiungere un oggetto in fondo con la funzione **Append[lista,elem]** e all'inizio con **Prepend[lista,elem].**

```
Append[lista, 5]
Prepend[lista, 5]
```

Inoltre **AppendTo[s,elem]** e **PrependTo[s,elem]** assegnano a **s** la nuova lista ottenuta

```
AppendTo[lista, 5]
Length[lista]
(*numero di elementi che costituiscono la lista*)
PrependTo[lista, 0]
Length[lista]
```

Il comando **lista[[n]]** (o anche **Part[lista,n]**) fornisce l'elemento nella posizione **n** della **lista**

```
lista[[2]]
Part[lista, 3]
```

Per estrarre il primo o ultimo elemento si puo' usare il comando **First[lista]** o **Last[lista].**

```
First[lista]
Last[lista]
```

Delete[lista,n] elimina l'elemento nella posizione n. Per esempio

```
lista = {1, 2, 3, 4, 5, 6, 7, 8, 9};
Delete[lista, 5]
```

E' possibile eseguire operazioni algebriche sulle liste. Negli ultimi tre esempi le liste devono avere la stessa lunghezza.

```
{a, b, c} + 1
{a, b, c} * 2
{a, b, c} ^ 2
{a, b, c} + {d, e, f}
{a, b, c} * {d, e, f}
{a, b, c} ^ {d, e, f}
```

Per costruire una lista a partire da una espressione expr[i], e' utile usare il comando **Table[expr[i], {i,1,n}].** Nel seguente esempio **l2** e' definita usando **Set** mentre **l1** e' definita usando **SetDelayed.**

```
l2 = Table[i^2, {i, 1, 10}]
l1 := Table[2 + i (-1)^i, {i, 1, n}];
n = 10; l1
n = 5; l1
```

Concateno le liste **l1** e **l2** e, successivamente, unisco **l1** e **l2** eliminando gli elementi in comune e riordinandoli

```
Join[l1, l2]
Length[%]

Union[l1, l2](*riordina gli elementi*)
Length[%]

Reverse[l2](*inverte gli elementi della lista*)
```

Insert[list,elem,n] inserisce *elem* nella posizione *n* in *list*. Se *n*<0 la posizione e' contata dalla fine

```
l1
Insert[l1, x, 4]
```

Il comando **Table** si puo' anche usare per creare liste di liste. Usiamo poi **TableForm** per visualizzare la lista sotto forma di tabella (il primo indice forma le righe, il secondo le colonne)

```
Table[i + j, {j, 1, 2}, {i, 1, 3}]
% // TableForm

Table[i + j, {i, 1, 3}, {j, 1, 2}]
% // TableForm
```

Esempio. Generare la lista di liste {{0},{0,2},{0,2,4},{0,2,4,6},{0,2,4,6,8}} usando il comando **Table**, in due modi diversi.

```
Table[2 j,{i,0,4},{j,0,i}]
Table[j,{i,0,8},{j,0,i,2}]//TableForm
```

Esempio. Costruire una lista di 10 elementi, costituita solo da interi scelti random tra {-2,2}

```
Table[Random[Integer,{-2,2}],{10}]
```

Le liste (i cui elementi non sono liste) , in matematica e fisica, vengono anche chiamate vettori (*Mathematica* li interpreta come vettori colonna). Il prodotto tra liste altro non e' che il prodotto scalare tra vettori e si indica con **.** (**Dot**). Il prodotto vettoriale si denota con × (**Cross)**

```
l1 = {a, b, c}; l2 = {c, d, e};
l1 * l2(*moltiplica componente per componente*)
l1.l2  (*prodotto scalare*)
l1 × l2 (*prodotto vettoriale*)
```

Data una lista, i comandi **Apply[Plus,lista]** e **Apply[Times,lista]**, rispettivamente, sommano e moltiplicano tra loro gli elementi della lista.

```
Apply[Plus, l1]
Apply[Times, l1]
```

Possiamo quindi ridefinire il prodotto scalare tra vettori

```
ps[x_List, y_List] := Apply[Plus, x * y] /; Length[x] == Length[y]
```

VectorQ[x] (oppure **VectorQ[x,NumberQ]**) ha valore *True* se x e' un vettore (oppure un vettore di numeri).

```
VectorQ[2]
VectorQ[{2}]
VectorQ[{{a, b}, {c, d}}]
VectorQ[{a, b, c, d}]
VectorQ[{a, b, c, d}, NumberQ[{a, b, c, d}]]
```

Consideriamo la lista di liste

```
t = {{a, b}, {c, d}, {e, f}};
VectorQ[t]
```

Flatten[t] elimina le sottoliste ovvero rimuove le parentesi {} interne

```
Flatten[t]
VectorQ[%]
```

Esempio. Generare una tabella in cui, nella prima colonna ci sia $(\cos(x))^n$ e nella seconda colonna ci sia $(\cos(x))^n$ espresso in termini di angoli multipli, per n=1,..,5.

```
Table[{Cos[x]^n, TrigReduce[Cos[x]^n]}, {n, 1, 5}];
% // TableForm
```

Esempio. Definire (in due modi diversi) una funzione che, data una lista di numeri, ne dia la media aritmetica.

```
f1[x_ /; VectorQ[x, NumberQ]] := Apply[Plus, x] / Length[x] /; Length[x] > 0
f2[x_ /; VectorQ[x, NumberQ]] := Sum[x[[i]], {i, 1, Length[x]}] / Length[x]
```

Esempio. Definire (in due modi diversi) una funzione che, data una lista di numeri, ne dia la media geometrica.

```
f3[x_ /; VectorQ[x, NumberQ]] := Apply[Times, x]^Length[x]
f4[x_ /; VectorQ[x, NumberQ]] :=
 (Product[x[[i]], {i, 1, Length[x]}])^Length[x]
```

■ 3.2 Matrici

Dato che una lista puo' contenere altre liste, rappresentiamo una matrice come una lista di liste ovvero una matrice e' rappresentata come la lista delle sue righe

```
m = {{a, b}, {c, d}}
M = Table[a[i, j], {i, 1, 3}, {j, 1, 3}]
```

Analogamente **MatrixQ[x]** (oppure **MatrixQ[x,NumberQ]**) ha valore *True* se x e' una matrice (oppure una matrice di numeri).

Con il comando **M[[i,j]]** estraggo l'elemento nella posizione i,j di M, con il comando M[[n]] estraggo la riga n.

```
M[[1, 2]](*elemento nella posizione 1,2*)
M[[2]](*riga 2 di M*)
```

Il comando **//MatrixForm** rappresenta la matrice nella forma tradizionale. E' solo un disegno, non una matrice vera e propria. Non definite mai una matrice con **MatrixForm**

```
matrix = {{a, b}, {c, d}} // MatrixForm
m // MatrixForm    (*m e' una matrice 2 x2*)
{{a, b, c}} // MatrixForm  (*matrice 1 x3*)
{{a}, {b}, {c}} // MatrixForm   (*matrice3x1*)
```

Esempio. Definiamo una funzione che, data una matrice , restituisca la somma degli elementi.

```
f[m_ /; MatrixQ[m]] := Apply[Plus, Flatten[m]]
```

Abbiamo visto che il comando **Table** puo' anche essere usato per costruire matrici. Per esempio creiamo una matrice 3× 2 cioe' una lista di 3 elementi, di cui ciascun elemento e' una lista di due

```
m1 = Table[{i, i + 1}, {i, 3}];
MatrixForm[m1]
```

cioe' m1 e' una matrice con 3 righe e 2 colonne. Adesso costruiamo una matrice 2×3

```
m2 = Table[i j, {i, 2}, {j, 3}];
MatrixForm[m2]
```

oppure una matrice 3×2

```
m3 = Table[i + j, {i, 3}, {j, 2}];
MatrixForm[m3]
```

e una generica matrice 2×3 usando il comando **Subscript**

```
A = Table[Subscript[a, i, j], {i, 2}, {j, 3}]
MatrixForm[A]
```

e ora costruiamo una matrice 3×2

```
A = Table[Subscript[a, i, j], {i, 3}, {j, 2}]
MatrixForm[A]
```

Il seguente comando genera una matrice 2×3 i cui elementi sono numeri random

```
Table[Random[], {2}, {3}] // MatrixForm
```

Calcoliamo, rispettivamente, l'inversa, la trasposta, il determinante, gli autovalori, la dimensione della matrice **m** e costruiamo la matrice identita' di ordine n

```
m = {{a, b}, {c, d}}
Inverse[m] // MatrixForm
Transpose[m] // MatrixForm
m // Det
m // Eigenvalues
Dimensions[m]
IdentityMatrix[%[[1]]] // MatrixForm
```

Con **. (e non con *)** si denota anche il prodotto righe per colonne tra matrice e vettore o tra due matrici (attenzione che le dimensioni siano compatibili!)

```
X = {x, y};
m.X  (*il risultato e' un vettore a 2 componenti*)
mm = {{a, b, c}, {d, e, f}}(*mm e' una matrice 2 x3*)
m.mm(*il risultato e' una matrice 2 x3*)
```

```
mm.X (*le dimensioni non sono compatibili*)

Dot::dotsh :
 Tensors {{a, b, c}, {d, e, f}} and {x, y} have incompatible shapes.

{{a, b, c}, {d, e, f}}.{x, y}

Transpose[mm].X (*il risultato e' un vettore a 3 componenti*)
```

Il comando **RowReduce**, con il metodo di eliminazione di Gauss-Jordan, trasforma, mediante opportune combinazioni lineari delle righe, una matrice, in forma triangolare superiore in cui ogni elemento della diagonale principale e' 1 oppure 0. Per matrici quadrate non singolari **m** si ha **RowReduce[m]** e' la matrice identita'.

Consideriamo la matrice

```
m = {{a, b, c}, {d, e, f}};
MatrixForm[%]
RowReduce[m] // MatrixForm
```

Il comando **NullSpace** fornisce le eventuali autosoluzioni del sistema omeogeneo associato alla matrice **m** cioe' le eventuali autosoluzioni del sistema **m.X=0** (fornisce una base di vettori del kernel).

```
NullSpace[m]
X = First[%]
Simplify[m.X](*verifica*)
```

Nel prossimo paragrafo torneremo a parlare di matrici.

Esempio. Data la matrice M dipendente dal parametro reale k,

$$\begin{pmatrix} 1 & 2k & -1 & 0 \\ 4 & 1 & -2 & 1 \\ 1 & 7 & k & 0 \\ -k & 0 & 2 & k \end{pmatrix}$$

determinare per quali k , M e' invertibile e determinare l'inversa.

```
ClearAll[k, M, M2]

M = {{1, 2 k, -1, 0}, {4, 1, -2, 1}, {1, 7, k, 0}, {-k, 0, 2, k}};
MatrixForm[M]

Det[M]
Roots[% == 0, k]
```

La matrice e' invertibile quando $k \neq 1/2$ e si ha

```
M2 = Inverse[M];
MatrixForm[%]
```

Verifichiamo

```
Simplify[M.M2] // MatrixForm
Simplify[M.M2] == IdentityMatrix[4]
```

Esempio. Siano u={1,6,3,0},v={3,7,0,1},w={-1,5,6,-1} tre vettori di R^4 e sia S lo spazio da essi generato. Determinare la dimensione d di S, determinare una base per S e determinare 4-d vettori che, insieme alla base trovata, formano una base in R^4 .

```
u = {1, 6, 3, 0}; v = {3, 7, 0, 1}; w = {-1, 5, 6, -1};
```

```
gauss = RowReduce[{u, v, w}];
MatrixForm[gauss]
```

La dimensione di S e' 2 dato che 2 vettori riga non nulli formano la matrice ridotta. Una base per S e' costituita dalle prime due righe della matrice ridotta

```
b1 = gauss[[1]]
b2 = gauss[[2]]
```

Per trovare una base in R^4 dobbiamo determinare b3 e b4 tale che {b1,b2,b3,b4} siano linearmente indipendenti. Proviamo con

```
b3 = {0, 0, 1, 0}; b4 = {0, 0, 0, 1};
```

Verifichiamo che effettivamente sono linearmente indipendenti.

```
RowReduce[{b1, b2, b3, b4}] // MatrixForm
```

Esempio. Siano u={2,3,1,4},v={1,2,1,1},w={k,2k,k^2,4k}. Determinare k tale che u,v,w siano linearmente indipendenti in R^4 .

```
u = {2, 3, 1, 4}; v = {1, 2, 1, 1}; w = {k, 2 k, k^2, 4 k};

RowReduce[{u, v, w}] // MatrixForm
```

Se k≠1 trovo tre righe non nulle e quindi i vettori sono linearmente indipendenti. Studiamo a parte il caso k=1.

```
RowReduce[{u, v, w} /. k → 1] // MatrixForm
```

Anche per k=1 i vettori sono linearmente indipendenti.

■ 3.3 Risoluzione di equazioni e di sistemi

Un'espressione come x^2+2 x-7==0 rappresenta una equazione in *Mathematica*. Per il segno di uguale si usa il comando **==** (l'uguale logico) e non **=** (**Set**) che e' usato per assegnare un valore a qualche cosa. Per risolvere un'equazione algebrica f1==f2, rispetto alla variabile x, si possono usare i comandi **Solve[f1==f2,x]** oppure **Roots[f1==f2,x]** . **Solve** fornisce le soluzioni nella forma **x→valore**. **Roots** fornisce le soluzioni usando gli operatori logici **&&** e **||**. Con questi comandi *Mathematica* cerca di fornire una espressione esplicita della soluzione. Se considero un polinomio in una variabile di grado al piu' 4 allora questo e' sempre possibile: *Mathematica* utilizza le formule di Cardano e di Ludovico Ferrari.

```
Clear["Global`*"]
Roots[x^2 + 2 x - 7 == 0, x]
Solve[x^2 + 2 x - 7 == 0]
```

Se vogliamo le soluzioni senza le frecce usiamo la regola di sostituzione immediata

```
x /. %
```

Abbiamo le soluzioni rappresentate come una lista. Se vogliamo assegnare il nome x1 e x2 alle soluzioni, estraggo il primo e il secondo elemento dalla lista

```
x1 = %[[1]];
x2 = %%[[2]];
{x1, x2}
```

Nel caso di equazioni con parametri, devo specificare rispetto a quali variabili risolviamo l'equazione

```
Solve[a x + b y + c == 0, y]
Solve[a x + b y + c == 0, x]

Solve[x^6 == 1]
x /. %

Plot[x^6 - 1, {x, -2, 2}, AxesLabel → {"x", "y"}]
```

In questo caso *Mathematica* non riesce a trovare le soluzioni

```
Solve[x^5 + x^2 + 1 == 0]
Roots[x^5 + x^2 + 1 == 0, x]
Plot[x^5 + x^2 + 1, {x, -3, 3}, AxesOrigin → {0, 0}]
```

e conviene usare i comandi **NSolve[**f**1==**f**2,x]** oppure **NRoots[**f**1==**f**2,x]** che risolvono direttamente l'equazione cercando numericamente le soluzioni e non le soluzioni esatte.

Il comando **Solve** (ma non **Roots**) si puo' anche usare per risolvere equazioni non algebriche

```
Solve[Log[Sqrt[x]] == Sqrt[Log[x]], x]
```

Bisogna fare attenzione perche' a volte compare un messaggio che ci avverte che stiamo perdendo delle soluzioni

```
NSolve[Sin[x] == 0.12, x]
Plot[{Sin[x], 0.12}, {x, -50, 50},
 PlotStyle → {{RGBColor[0, 1, 0]}, RGBColor[1, 0, 1 / 2]}]
```

oppure che *Mathematica* non riesce a trovare le soluzioni

```
Solve[Log[x] * Exp[x] == 1 / x^2, x]
Plot[{Log[x] * Exp[x], 1 / x^2}, {x, 1, 2}]
```

Una possibile alternativa per ottenere le soluzioni e' usare il comando **FindRoot[**f**1==**f**2,{x,x0}]** che cerca, usando il metodo delle tangenti di Newton, una soluzione numerica dell'equazione di partenza partendo dall'approssimazione x=x0. Questo comando trova al piu' una radice e converge solo se x0 e' vicino a una soluzione (si potrebbe, per esempio, tracciare un grafico di f1-f2 per vedere approssimativamente dove cadono gli zeri).

```
FindRoot[Log[x] * Exp[x] == 1 / x^2, {x, 1}]
FindRoot[Cos[x] == x, {x, 0}]

Plot[Tan[x] - x, {x, -5, 5}]
FindRoot[Tan[x] == x, {x, 1}]
FindRoot[Tan[x] == x, {x, 2}]
FindRoot[Tan[x] == x, {x, 3}]

Plot[3 Cos[x] - Log[x], {x, 0, 15}]
FindRoot[3 Cos[x] == Log[x], {x, 5}]
```

I comandi **Solve**, **NSolve**, **FindRoot** si possono usare per risolvere simultaneamente un sistema di equazioni, anche con parametri.

```
eq1 = a x + y == 0;
eq2 = 2 x + (1 - a) y == 1;
Solve[{eq1, eq2}, {x, y}]
```

E' anche possibile usare il comando **Reduce**, che fornisce una lista completa delle soluzioni sotto forma di espressione logica...ma a volte un po' troppo lunga. Notate l'uso dell' **And** logico **&&** e dell'**Or** logico **||**. Nella lettura delle soluzioni tenete a mente che **&&** ha la precedenza su **||**.

```
Reduce[x^2 == 3, x]
Reduce[a x == b, x]
Reduce[a x^2 == b, x]
Reduce[{eq1, eq2}, {x, y}]

Reduce[{x + y^2 == 2, x + y == 0}, {x, y}]
Needs["Graphics`ImplicitPlot`"]
ImplicitPlot[{x + y^2 == 0, x + y == 0}, {x, -3, 3}]
```

Esempio. In un triangolo la somma dell'altezza h e della base b e' 7 e la differenza e' 5. Trovare l'area del triangolo.

```
Solve[{h + b == 7 , h - b == 5}, {h, b}] // Flatten

(h * b) / 2 /. %
```

Esempio. In un trapezio isoscele, detti b,B e h rispettivamente la base minore, la base maggiore e l'altezza, si ha che hbB=10, h+b+B=12, b+B+2h=20. Determinare l'area del trapezio.

```
Clear["Global`*"]
Solve[{b B h == 10, b + B + h == 12, b + B + 2 h == 20}, {b, B, h}]
```

Dovendo essere b<B si ha che

```
sol = %[[1]]
```

Quindi

```
((b + B) h / 2) /. sol
```

Esempio. Determinare l'equazione della circonferenza passante per i punti (1,2), (2,1), (1,1).

Imponiamo che il cerchio passi per i tre punti e risolviamo il sistema di 3 equazioni in 3 incognite

```
Clear["Global`*"]
eq1 = (x - x0)^2 + (y - y0)^2 - r^2 == 0 /. {x → 1, y → 2};
eq2 = (x - x0)^2 + (y - y0)^2 - r^2 == 0 /. {x → 2, y → 1};
eq3 = (x - x0)^2 + (y - y0)^2 - r^2 == 0 /. {x → 1, y → 1};
sol = Solve[{eq1, eq2, eq3}, {x0, y0, r}]
```

Naturalmente e' accettabile solo la soluzione con r>0. L'equazione della circonferenza e' data da

```
Simplify[((x - x0)^2 + (y - y0)^2 == r^2) /. %[[2]]]
```

Esempio. Determinare le intersezioni tra l'ellisse di equazione 4x^2+y^2=16 e la parabola x=y^2-1.

Visualizziamo i grafici delle due funzioni per vedere se effettivamente si incontrano:

```
Needs["Graphics`ImplicitPlot`"]
g1 = ImplicitPlot[4 x^2 + y^2 == 16, {x, -3, 3}, DisplayFunction → Identity];
g2 = ImplicitPlot[x == y^2 - 1, {x, -3, 3}, DisplayFunction → Identity];
Show[{g1, g2}, DisplayFunction → $DisplayFunction,
 Ticks -> {{0, 0.5, 1, 1.5, 2}, {1, 2, 3, 4}}, AspectRatio -> Automatic]

Solve[{4 x^2 + y^2 == 16, x == y^2 - 1}, {x, y}]
```

Scarto le soluzioni complesse e trovo i due punti

```
punto1 = {x, y} /. %[[3]]
punto2 = {x, y} /. %%[[4]]
```

■ 3.4 Sistemi Lineari

Se dobbiamo risolvere un *sistema lineare* generico con n equazioni e m incognite nella forma M.x=B possiamo usare, oltre ai comandi **Solve[M.x==B]** e **Reduce[M.x==B]**, il comando **LinearSolve[M,B],** che fornisce la soluzione sotto forma di lista

```
Clear["Global`*"]
M = {{1, 5}, {2, 1}};
B = {a, b};
X = LinearSolve[M, B]
Simplify[M.X - B](*verifica*)
```

Esempio. Discutete e risolvete il sistema lineare: x+2 y-z=0; x+y+3 z=1; 2 x+2 y+z=-1.

Definiamo (per righe) la matrice **M** dei coefficienti, il vettore **B** termine noto e il vettore **X** delle incognite.

```
Clear["Global`*"]
M = {{1, 2, -1}, {1, 1, 3}, {2, 2, 1}};
MatrixForm[M]
B = {0, 1, -1};
X = {x, y, z};
Dimensions[M][[1]] == Dimensions[M][[2]]
```

M e' una matrice quadrata. Verifichiamo che **M** e' non singolare. Questo lo possiamo fare o calcolando il determinante di **M**

```
Det[M]
```

oppure riducendo **M** in forma triangolare superiore e verificando che non ci siano righe formate solo da 0.

```
RowReduce[M] // MatrixForm
```

Dato che **M** e' non singolare il sistema ammette 1! soluzione che possiamo trovare in diversi modi (notate il modo diverso in cui viene fornita la soluzione)

```
Solve[M.X == B, X]   (*primo*)

LinearSolve[M, B]  (*secondo*)

Reduce[M.X == B, X] (*terzo*)
```

Un altro modo di risolvere il sistema e' calcolare la matrice inversa di **M** e moltiplicarla per il termine noto

```
Inverse[M].B (*quarto*)
```

Oppure applicare il metodo di eliminazione di Gauss con il comando **RowReduce** alla matrice completa del sistema cioe' alla matrice ottenuta aggiungendo il vettore termine noto **B** alla matrice **M** come ultima colonna. Procediamo cosi'

```
Transpose[M];
Append[%, B](*aggiungo una riga alla matrice trasposta*);
MB = Transpose[%](*e' la matrice completa*);
S = RowReduce[MB];
MatrixForm[S];
X = Transpose[S][[4]](*estraggo la quarta riga*)

M.X - B(*verifica*)
```

Esempio. Determinare, se esiste, la soluzione del sistema **M.X=B** dove

```
Clear["Global`*"]
M = {{3, 4, 5, 0}, {1, 2, 8, 3}, {0, 3, -1, 1}, {3, 1, 1, 5}};
B = {1, 3, 5, 1};

RowReduce[M] // MatrixForm
```

M e' non singolare quindi il sistema ammette una e una sola soluzione. Risolviamo il sistema con **RowReduce**. Dobbiamo aggiungere il vettore termine noto **B** alla matrice **M** come ultima colonna. Procediamo cosi'

```
Transpose[M]
Append[%, B](*aggiungo una riga alla matrice trasposta*)
MB = Transpose[%]
S = RowReduce[MB];
MatrixForm[S]
```

L'ultima colonna e' la soluzione cercata

```
sol = Transpose[S][[5]]
```

Verifichiamo

```
M.sol - B
```

• Se il sistema non ammette soluzione questi metodi si comportano in maniera diversa. Consideriamo il sistema x+5y=2, 2x+10y=11.

```
Clear["Global`*"]
M = {{1, 5}, {2, 10}}; B = {2, 11}; X = {x, y};
MatrixForm[M]
Solve[M.X == B, X]
```

Solve trova una lista vuota {} di soluzioni.

```
LinearSolve[M, B]
```

LinearSolve mi avverte con un messaggio di errore.

```
RowReduce[{{1, 5, 2}, {2, 10, 11}}] // MatrixForm
```

RowReduce trova che il sistema non ammette soluzione dato che la seconda equazione e' 0=1!

• Se il sistema ammette infinite soluzioni, **LinearSolve** non le trova tutte. Consideriamo il sistema x-y=2; 3x-3y=6.

```
Clear["Global`*"]
M = {{1, -1}, {3, -3}}; B = {2, 6};
LinearSolve[M, B]
```

Solve ci avverte con un messaggio di errore, ma le trova tutte

```
sol = Solve[{x - y == 2, 3 x - 3 y == 6}, {x, y}]
```

La soluzione fornita da **Solve** in forma parametrica e'

```
solpar = {x, y} /. sol[[1]] /. y → t
```

Anche **RowReduce** ci da' il risultato corretto

```
RowReduce[{{1, -1, 2}, {3, -3, 6}}] // MatrixForm
```

Esempio. Risolvere il seguente sistema al variare del parametro k

x-y+kz=k+1; kx-y+z=2; x-ky+z=3-k^2.

```
Clear["Global`*"]
M = {{1, -1, k}, {k, -1, 1}, {1, -k, 1}}; MatrixForm[M]
Dimensions[M];
If[Dimensions[M][[1]] == Dimensions[M][[2]],
 Roots[Det[M] == 0, k], Print["M non e' una matrice quadrata"]]
B = {k + 1, 2, 3 - k^2};
X = {x, y, z};
```

Per i valori di k che non annullano il determinante il sistema ammette 1! soluzione, che ci fornisce **Solve** oppure **LinearSolve**

```
LinearSolve[M, B]
```

```
Solve[M.X == B, X]
```

Reduce fornisce una soluzione piu' dettagliata, che include anche i casi in cui **M** e' singolare

```
Reduce[M.X == B, X]
```

Analizziamo cosa accade in corrispondenza dei valori che annullano il determinante (k=1 e k=-2): con **RowReduce** stabiliamo quante soluzioni ci sono:

```
RowReduce[M /. k → 1] // MatrixForm
(*∞^2 soluzioni*)
```

```
RowReduce[M /. k → -2] // MatrixForm
(*∞^1 soluzioni*)
```

LinearSolve non le trova tutte, **Solve** le trova ma ci avverte con un messaggio di errore,

```
LinearSolve[M /. k → 1, B /. k → 1]
```

```
Solve[(M /. k -> 1).X == (B /. k → 1), X]
```

Le soluzioni fornite da **Solve**, in forma parametrica sono

```
solpar = {x, y, z} /. %[[1]]
```

```
LinearSolve[M /. k → -2, B /. k → -2]
```

```
Solve[(M /. k -> -2).X == (B /. k → -2), X]
```

Le soluzioni fornite da **Solve**, in forma parametrica sono

```
solpar = {x, y, z} /. %[[1]]
```

Esercizio 1.Trovare, se esiste, la soluzione del sistema

3x+y+6z=2;2x+y+3z=7;x+y+z=4

usando **Solve, LinearSolve** e **RowReduce**.Verificate che la soluzione trovata e' corretta.

Esercizio 2. Trovare le soluzioni del sistema

x+5y+6z=1;x+3z=4

usando **Solve, LinearSolve** e **RowReduce**. Confrontate le soluzioni ottenute.

■ 3.5 Disequazioni

Molto utile per risolvere disequazioni contenenti polinomi e funzioni razionali e' il comando **InequalitySolve** che si trova nel pacchetto Algebra'InequalitySolve'. Carichiamolo

```
Clear["Global`*"]
Needs["Algebra`InequalitySolve`"]
```

e proviamo a vedere come funziona.

```
InequalitySolve[x - 1 > 3, x]
InequalitySolve[x^2 < -2, x]
InequalitySolve[3 x^2 - 8 x + 5 ≥ 0, x]

InequalitySolve[3 x^4 - 8 x + 5 > 0, x] // N
Plot[3 x^4 - 8 x + 5, {x, 0, 2}]

InequalitySolve[(x^3 - 5 x + 4) / (x^2 - 5) ≥ 0, x] // N
Plot[(x^3 - 5 x + 4) / (x^2 - 5), {x, -3, 3}]
```

Nel caso in cui le disequazioni non siano algebriche, *Mathematica* ci avverte che le soluzioni che trova potrebbero non essere corrette

```
InequalitySolve[Sin[x]^2 - 2 Cos[3 x] ≥ 0, x] // N
```

disegnando il grafico della funzione visualizziamo in quali intervalli $\sin^2 x - 2\cos(3x) \geq 0$

```
Plot[Sin[x]^2 - 2 Cos[3 x], {x, -5, 5}]
```

oppure non trova nessuna soluzione

```
InequalitySolve[Log[x^2 + 1] - Sin[x^2] ≥ 0, x]
```

Mathematica puo' anche risolvere sistemi di disequazioni

```
InequalitySolve[x^2 ≥ 1 && x^2 - x ≤ 2, x]

InequalitySolve[x^2 - 1 > 0 && 2 x^3 - 4 ≤ 0, x]

InequalitySolve[x^2 - 1 > 0 && 2 x^3 - 4 ≤ 0 && x^3 - 5 x^2 - 1 > 0, x]
```

oppure risolvere disequazioni in cui compaiono due o piu' variabili.

```
InequalitySolve[{x - y ≤ 1, x + y > 0}, {x, y}]

InequalitySolve[{x^2 - y^2 ≤ 1, x + y > 0}, {x, y}]

InequalitySolve[{x^2 - y^2 == 1, x + y > 0}, {x, y}]
```

Il *package* Graphics`InequalityGraphics` consente di disegnare l'insieme delle soluzioni di sistemi di disequazioni in due variabili, nella regione limitata da {x,xmin,xmax} e {y,ymin,ymax}

```
Needs["Graphics`InequalityGraphics`"]

InequalityPlot[{x - y ≤ 1, x + y > 0}, {x, -2, 2}, {y, -1 / 2, 4}]
```

```
InequalityPlot[{x^2 - y^2 ≤ 1, x + y > 0}, {x, 1 / 1, 3}, {y, -1 / 2, 3}]
```

Troveremo utile questo pacchetto quando studieremo il grafico di funzioni.