Calcolo Numerico - Appunti

260236

marzo 2024

Prefazione

Ogni sezione di teoria presente in questi appunti, è stata ricavata dalle seguenti risorse:

- Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave. [1]

Altro:

GitHub repository

Indice

1 Equazioni non lineari

1.1 Introduzione

Il **calcolo degli zeri di una funzione** f reale di variabile reale o delle **radici dell'equazione** $f(x) = 0$, è un problema assai ricorrente nel Calcolo Scientifico.

In generale, *non è possibile* approntare metodi numerici che calcolino gli zeri di una generica funzione in un numero finito di passi. I metodi numerici per la risoluzione di questo problema sono pertanto necessariamente *iterativi*. A partire da uno o più dati iniziali, scelti convenientemente, essi generano una successione di valori $x^{(k)}$ che, sotto opportune ipotesi, convergerà ad uno zero α della funzione f studiata.

1.2 Il metodo di bisezione (o iterativo)

Sia f una funzione continua in $[a, b]$ tale che $f(a) f(b) < 0$. Per cui, vale il teorema degli zeri di una funzione continua, ossia f ammette almeno uno zero in (a, b).

Si supponga che ci sia un solo zero, indicato con α e nel caso in cui ce ne sia più di uno, individuare un intervallo tale che ne contenga solo uno.

Il **metodo di bisezione** (o **iterativo**) è una strategia che si suddivide nei seguenti passaggi:

1. **Dimezzare l'intervallo di partenza**;
2. **Selezionare tra i due sotto-intervalli ottenuti quello nel quale f cambia di segno agli estremi**;
3. **Applicare ricorsivamente questa procedura all'ultimo intervallo selezionato**.

Matematicamente parlando, dato $I^{(0)} = (a, b)$, e più in generale, $I^{(k)}$ il sotto-intervallo selezionato al passo k-esimo, si sceglie come $I^{(k+1)}$ il semi-intervallo di $I^{(k)}$ ai cui estremi f cambia di segno.

Questa procedura garantisce che ogni sotto-intervallo selezionato $I^{(k)}$ conterrà α. Questo poiché la successione $\left\{x^{(k)}\right\}$ dei punti medi dei sotto-intervalli $I^{(k)}$ dovrà ineluttabilmente convergere a α, in quanto la **lunghezza dei sotto-intervalli tende a** 0 per k che **tende all'infinito**.

Formalizziamo questa idea con un piccolo algoritmo. Ponendo:

$$a^{(0)} = a, \quad b^{(0)} = b, \quad I^{(0)} = \left(a^{(0)}, b^{(0)}\right), \quad x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2}$$

Al passo $k \geq 1$ il metodo di bisezione calcolerà il semi-intervallo $I^{(k)} = \left(a^{(k)}, b^{(k)}\right)$ dell'intervallo $I^{(k-1)} = \left(a^{(k-1)}, b^{(k-1)}\right)$, nel seguente modo (si ricorda che α è lo zero che si sta cercando):

1. Calcolo $x^{(k-1)} = \dfrac{a^{(k-1)} + b^{(k-1)}}{2}$
2. Se $f\left(x^{(k-1)}\right) = 0$:
   (a) Allora $\alpha = x^{(k-1)}$ e l'algoritmo termina.
3. Altrimenti, se $f\left(a^{(k-1)}\right) \cdot f\left(x^{(k-1)}\right) < 0$:
   (a) Si pone $a^{(k)} = a^{(k-1)}$
   (b) Si pone $b^{(k)} = x^{(k-1)}$
   (c) Si incrementa $k+1$ e si ripete ricorsivamente.
4. Altrimenti, se $f\left(x^{(k-1)}\right) \cdot f\left(b^{(k-1)}\right) < 0$:
   (a) Si pone $a^{(k)} = x^{(k-1)}$
   (b) Si pone $b^{(k)} = b^{(k-1)}$
   (c) Si incrementa $k+1$ e si ripete ricorsivamente.

Example 1

Data la funzione $f(x) = x^2 - 1$, si parta da $a^{(0)} = -0.25$ e $b^{(0)} = 1.25$, e si applichi il metodo di bisezione:

1. Con $a^{(0)} = -0.25$ e $b^{(0)} = 1.25$:
   (a) Si calcola il punto medio:
   $$x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2} = \frac{-0.25 + 1.25}{2} = 0.5$$
   (b) Si calcola la funzione con il punto medio come parametro:
   $$f(0.5) = 0.5^2 - 1 = -0.75$$
   (c) Dato che la funzione nel punto medio non è uguale a zero, l'algoritmo deve continuare. Per farlo, bisogna sostituire il punto medio con uno dei due estremi. Per decidere quale dei due sostituire, è necessario capire in quale cambia valore la funzione. Si verifica inizialmente con $a^{(0)}$:
   $$\begin{aligned} f\left(a^{(0)}\right) f\left(x^{(0)}\right) < 0 &= f(-0.25)\, f(0.5) < 0 \\ &= (-0.9375 \cdot -0.75) < 0 \\ &= 0.703125 \text{ ✗} \end{aligned}$$

(d) Si procede con l'algoritmo, provando adesso la $b^{(0)}$:

$$\begin{aligned} f\left(x^{(0)}\right) f\left(b^{(0)}\right) < 0 &= f(0.5)\, f(1.25) < 0 \\ &= (-0.75 \cdot 0.5625) < 0 \\ &= -0.421875 \checkmark \end{aligned}$$

(e) Si pone $a^{(1)} = x^{(0)} = 0.5$

(f) Si pone $b^{(1)} = b^{(0)} = 1.25$

(g) Si incrementa k, $k = k + 1 = 0 + 1 = 1$

2. Con $a^{(1)} = 0.5$ e $b^{(1)} = 1.25$:

(a) Si calcola il punto medio:

$$x^{(1)} = \frac{a^{(1)} + b^{(1)}}{2} = \frac{0.5 + 1.25}{2} = 0.875$$

(b) Si calcola la funzione con il punto medio come parametro:

$$f(0.875) = 0.875^2 - 1 = -0.234375$$

(c) Dato che la funzione nel punto medio non è uguale a zero, l'algoritmo deve continuare:

$$\begin{aligned} f\left(a^{(1)}\right) f\left(x^{(1)}\right) < 0 &= f(0.5)\, f(-0.234375) < 0 \\ &= (-0.75 \cdot -0.945068359375) < 0 \\ &= 0.70880126953125 \text{ ✗} \end{aligned}$$

(d) Si procede con l'algoritmo:

$$\begin{aligned} f\left(x^{(1)}\right) f\left(b^{(1)}\right) < 0 &= f(-0.234375)\, f(1.25) < 0 \\ &= (-0.945068359375 \cdot 0.5625) < 0 \\ &= -0.5316009521484375 \checkmark \end{aligned}$$

(e) Si pone $a^{(2)} = x^{(1)} = 0.875$

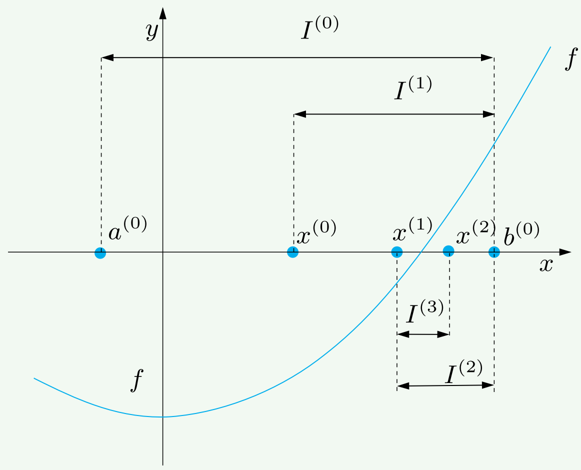
(f) Si pone $b^{(2)} = b^{(1)} = 1.25$

(g) Si incrementa k, $k = k + 1 = 1 + 1 = 2$

Si omettono i restanti calcoli per $k = 2, k = 3$, ma si lasciano qua di seguito i risultati:

- $I^{(2)} = (0.875, 1.25)$ e $x^{(2)} = 1.0625$
- $I^{(3)} = (0.875, 1.0625)$ e $x^{(2)} = 0.96875$

Nella seguente figura si possono vedere le iterazioni effettuate:



Iterazioni effettuate. [1]

Si noti che ogni intervallo $I^{(k)}$ contiene lo zero α. Inoltre, la successione $\left\{x^{(k)}\right\}$ converge necessariamente allo zero α in quanto ad ogni passo l'ampiezza $\left|I^{(k)}\right| = b^{(k)} - a^{(k)}$ dell'intervallo $I^{(k)}$ si dimezza.

Il valore $I^{(k)}$ può essere riassunto come:

$$\left|I^{(k)}\right| = \left(\frac{1}{2}\right)^{k} \cdot \left|I^{(0)}\right|$$

E di conseguenza l'**errore al passo k** può essere calcolato come:

$$\left|e^{(k)}\right| = \left|x^{(k)} - \alpha\right| < \frac{1}{2} \cdot \left|I^{(k)}\right| = \left(\frac{1}{2}\right)^{k+1} \cdot (b - a)$$

Inoltre, data una certa **tolleranza** ε, per **garantire che l'errore al passo** k **sia minore della tolleranza data** (ovvero, $\left|e^{(k)}\right| < \varepsilon$), basta applicare la seguente formula:

$$k_{\min} > \log_2\left(\frac{b-a}{\varepsilon}\right) - 1 \tag{1}$$

Dove $k_{\min}$ rappresenta il **numero <u>minimo</u> di iterazioni prima di trovare un intero che soddisfi la disuguaglianza**.

⚠ Possibile svantaggio

Il metodo di bisezione **non garantisce una riduzione monotona dell'errore**, ma solo il dimezzamento dell'ampiezza dell'intervallo all'interno del quale si cerca lo zero. Infatti, **non viene tenuto conto del reale andamento di** f e questo può provocare il mancato coinvolgimento di approssimazioni di α accurate.

1.3 Il metodo di Newton

Il **metodo di Newton** sfrutta la funzione f maggiormente rispetto al metodo di bisezione, usando i suoi valori e la sua derivata.

Si ricorda che la retta tangente alla curva $(x, f(x))$ nel punto $x^{(k)}$ è:

$$y(x) = f\left(x^{(k)}\right) + f'\left(x^{(k)}\right)\left(x - x^{(k)}\right)$$

Cercando un $x^{(k+1)}$ tale che la **retta tangente in quel punto sia uguale a zero** $y\left(x^{(k+1)}\right) = 0$, allora si trova:

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \qquad k \geq 0 \tag{2}$$

Purché la derivata prima nel punto $x^{(k)}$ sia diversa da zero, cioè $f'\left(x^{k}\right) \neq 0$.

Questa equazione consente di calcolare una successione di valori $x^{(k)}$ a partire da un dato iniziale $x^{(0)}$. In altre parole, il **metodo di Newton calcola lo zero di f sostituendo localmente a f la sua retta tangente**.

A differenza del metodo di bisezione, tale **metodo converge allo zero in un solo passo quando la funzione f è lineare**, ovvero nella forma $f(x) = a_1 x + a_0$.

⚠ Limitazione

La **convergenza** del metodo di Newton non è garantita **per ogni scelta** di $x^{(0)}$, ma **soltanto** per valori di $x^{(0)}$ **sufficientemente vicini** ad α, ovvero **appartenenti ad un intorno** $I(\alpha)$ sufficientemente piccolo di α.

Alcune osservazioni a seguito anche di questa limitazione:

- A seguito di questa limitazione, risulta evidente che se $x^{(0)}$ è stato scelto opportunamente e se lo zero α è semplice ($f'(\alpha) \neq 0$), allora il metodo converge.
- Nel caso in cui f è derivabile con continuità pari a due, allora si ottiene la seguente convergenza:

  $$\lim_{k \to \infty} \frac{x^{(k+1)-\alpha}}{\left(x^{(k)} - \alpha\right)^2} = \frac{f''(\alpha)}{2f'(\alpha)} \tag{3}$$

  Il significato è: se $f'(\alpha) \neq 0$ il metodo di Newton converge almeno quadraticamente o con ordine 2.

  In parole povere, **per k sufficientemente grande, l'errore al passo $(k+1)$-esimo si comporta come il quadrato dell'errore al passo k-esimo, moltiplicato per una costante indipendente da k.**
- Se lo zero α ha molteplicità m maggiore di 1, ovverosia:

  $$f'(\alpha) = 0, \ldots, f^{(m-1)}(\alpha) = 0$$

Allora il metodo di Newton è ancora convergente, purché $x^{(0)}$ sia scelto opportunamente e $f'(x) \neq 0 \; \forall x \in I(\alpha) \setminus \{\alpha\}$. Tuttavia in questo caso l'ordine di convergenza è pari a 1. In tal caso, l'ordine 2 può essere ancora recuperato usando la seguente relazione al posto dell'equazione 2 ufficiale:

$$x^{(k+1)} = x^{(k)} - m \cdot \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \qquad k \geq 0 \tag{4}$$

Purché $f'\left(x^{(k)}\right) \neq 0$. Naturalmente, questo **metodo di Newton modificato** richiede una conoscenza a priori di m.

1.3.1 Come arrestare il metodo di Newton

Data una tolleranza fissa ε, esistono due tecniche applicabili per capire quando è necessario fermarsi ed evitare di continuare ad iterare:

- La **differenza fra due iterate consecutive**, il quale si arresta in corrispondenza del più piccolo intero k_{min} per il quale:

$$\left| x^{(k_{\text{min}})} - x^{(k_{\text{min}}-1)} \right| < \varepsilon \tag{5}$$

  (test sull'incremento).

- Un'altra tecnica applicata anche per altri metodi iterativi è il **residuo** al passo k, il quale è definito come:

$$r^{(k)} = f\left(x^{(k)}\right)$$

  Che è nullo quando $x^{(k)}$ è uno zero di f. In questo modo, il metodo viene arrestato alla prima iterata k_{min}:

$$\left| r^{(k_{\text{min}})} \right| = \left| f\left(x^{(k_{\text{min}})}\right) \right| < \varepsilon \tag{6}$$

  Da notare che tale tecnica fornisce una **stima accurata dell'errore** solo quando $|f'(x)|$ è circa pari a 1 in un intorno di I_α dello zero α cercato.

  Attenzione! Se la derivata non è circa pari a 1 in un intorno dello zero cercato, la tecnica porterà:

  - Ad una **sovrastima** dell'errore se $|f'(x)| \gg 1$ per $x \in I_\alpha$
  - Ad una **sottostima** dell'errore se $|f'(x)| \ll 1$ per $x \in I_\alpha$

1.4 Il metodo delle secanti

Nel caso in cui la funzione f non sia nota, il metodo di Newton non può essere applicato. Per fortuna, arriva in soccorso il **metodo delle secanti**, il quale esegue una valutazione di $f'\left(x^{(k)}\right)$ andando a sostituire quest'ultima con un **rapporto incrementale calcolato su valori di f già noti**.

Più formalmente, assegnati due punti $x^{(0)}$ e $x^{(1)}$, per $k \geq 1$ si calcola:

$$x^{(k+1)} = x^{(k)} - \left(\frac{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}\right)^{-1} \cdot f\left(x^{(k)}\right) \tag{7}$$

❓ Quando converge?

Il metodo delle secanti converge a seguito di certe condizioni:

- **Converge ad $\boldsymbol{\alpha}$**, se:
  - α radice semplice[1];
  - $I(\alpha)$ è un opportuno intorno di α;
  - $x^{(0)}$ e $x^{(1)}$ sono sufficientemente vicini ad α
  - $f'(x) \neq 0 \ \forall x \in I(\alpha) \setminus \{\alpha\}$
- **Converge con ordine p super-lineare**, se:
  - $f \in \mathcal{C}^2(I(\alpha))$
  - $f'(\alpha) \neq 0$

  Ovvero, esiste una costante $c > 0$ tale che:

$$\left|x^{(k+1)} - \alpha\right| \leq c\left|x^{(k)} - \alpha\right|^p \qquad p = \frac{1+\sqrt{5}}{2} \approx 1.618\ldots \tag{8}$$

- **Convergenza lineare**, se:
  - Radice α è multipla.

  Come succederebbe usando il metodo di Newton.

[1] $f'(\alpha) \neq 0$

2 Laboratorio

2.1 Introduzione al linguaggio MATLAB

L'introduzione al linguaggio di programmazione MATLAB sarà molto rapido. Si assume dunque che l'interfaccia grafica sia familiare e che concetti base di programmazione (per esempio "che cos'è una variabile?") siano ben noti.

In MATLAB, l'assegnazione di scalari a delle variabili è classica, quindi si utilizza il simbolo uguale: `a = 1` (assegnazione del valore `1` alla variabile `a`). Inoltre, il linguaggio è *case sensitive*, di conseguenza la variabile `a` è diversa dalla variabile `A`. Alcuni comandi utili e generali:

- `help` *`nome-comando`*, per avere informazioni in più riguardo al comando *`nome-comando`*;
- `clear` *`nome-variabile`*, per rimuovere la variabile *`nome-variabile`* dalla memoria. Se non viene inserito il *`nome-variabile`*, vengono rimosse tutte le variabili dalla memoria.
- `who`, per visualizzare le variabili attualmente in memoria.
- `clc`, per ripulire la *Command Window*.

Well-known variables

Esistono alcune variabili che sono ben note e hanno valori prestabiliti. Tra le più importanti:

- `pi`, che rappresenta il π e MATLAB gli assegna il valore `3.1416`
- `i`, che rappresenta l'unità immaginaria e MATLAB gli assegna il valore `0.0000 + 1.0000i`
- `eps`, che rappresenta il più piccolo valore rappresentabile nel calcolatore (PC) attualmente in uso. Solitamente, `eps` ritorna il valore `2.2204e-16`.

Questo tipo di variabili possono essere ridefinite, ma <u>non</u> è una *good practice*.

Cambiare il formato delle variabili: `format`

Il comando `format` è utilizzato per cambiare il formato con cui sono rappresentate le variabili. MATLAB non cambia la precisione della variabile (quindi non si ottiene una precisione maggiore dopo la virgola), ma modifica soltanto la rappresentazione. Di default MATLAB utilizza una rappresentazione di tipo `short`. Tra i più utilizzati (di default `pi` è uguale a `3.1416`):

- `default` per reimpostare la rappresentazione di default.
- Decimale:
  - `short`, rappresentazione a 5 cifre:

```
>> format short
>> pi

ans =
    3.1416
```

  - `long`, rappresentazione a 15 cifre:

```
>> format long
>> pi

ans =
   3.141592653589793
```

- *Floating point*:
  - `short e`, rappresentazione a 5 cifre floating point:

```
>> format short e
>> pi

ans =
   3.1416e+00
```

  - `long e`, rappresentazione a 15 cifre floating point:

```
>> format long e
>> pi

ans =
     3.141592653589793e+00
```

Altri formati si possono trovare nella documentazione ufficiale.

Assegnamento di vettori e matrici

- **Vettore riga**, si può creare utilizzando uno spazio tra i valori o una virgola `,`:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> b = [1, 2, 3, 4]

b =
     1     2     3     4
```

- **Vettore colonna**, si crea usando il punto e virgola `;`:

```
>> a = [1; 2; 3; 4]

a =
     1
     2
     3
     4
```

Talvolta può essere utile la generazione automatica di un vettore riga (sono ammessi anche i valori negativi e con la virgola ovviamente):

- **Vettore riga generato linearmente**, si crea usando i due punti e specificando il valore di inizio e il valore di fine:

```
>> a = [1 : 4]

a =
     1     2     3     4
```

- **Vettore riga generato usando un passo**, si crea usando i due punti e specificando (in ordine) il valore di inizio, il "salto", e il valore di fine. Nel caso in cui il salto sia troppo grande e si superi il valore di fine, MATLAB prenderà il primo valore ammissibile:

```
>> % Generazione con passo 1
>> a = [1 : 1 : 4]

a =
     1     2     3     4

>> % Generazione con passo 2
>> a = [1 : 2 : 5]

a =
     1     3     5

>> % Generazion con passo 2 (fine non raggiunta)
>> a = [1 : 2 : 6]

a =
     1     3     5

>> % ... ma cambiando l'upper bound
```

```
>> a = [1 : 2 : 7]

a =
     1     3     5     7
```

- **Vettore riga generato con valori uniformemente distanziati**, si crea usando la funzione `linspace`, la quale accetta tre parametri:
  - `x1`, valore di partenza.
  - `x2`, valore di fine.
  - `n`, numero di valori da generare; se non specificato, di default è `100`; se il valore inserito è zero o minore, viene creato un vettore vuoto.

```
>> % Vettore riga identico a: a = [1 : 4]
>> linspace(1, 4, 4)

ans =
     1     2     3     4

>> % Chiedendo piu' valori, MATLAB andra' ad utilizzare i
    decimali
>> linspace(1, 4, 6)
ans =

     1.000000000000000e+00     1.600000000000000e+00
     2.200000000000000e+00     2.800000000000000e+00
     3.400000000000000e+00     4.000000000000000e+00
```

Le matrici possono essere create a mano o usando la combinazione delle tecniche viste in precedenza:

- **Matrice**, le righe si creano usando gli spazi e le colonne si creano usando il punto e virgola:

```
>> a = [1 2 3 4; 5 6 7 8; 9 10 11 12]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> a = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

- **Matrice creata usando la generazione lineare dei vettori**, si possono utilizzare le tecniche precedenti e i punti e virgola:

```
>> % Usando a = [x : y]
>> a = [1 : 4; 5 : 8; 9 : 12]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12

```

```
>> % Usando a = [x : y : z]
>> a = [1 : 2 : 7; 9 : 2 : 15; 17 : 2 : 23]
a =

     1     3     5     7
     9    11    13    15
    17    19    21    23

>> % Usando linspace
>> a = [linspace(1, 4, 4); linspace(5, 8, 4); linspace(9, 12,
    4)]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Operazioni su vettori e matrici

- **Trasposizione**, la classica operazione eseguita con le matrici o vettori, si esegue con la keyword `'` oppure usando la funzione `transpose`:

```
>> a = [1 2 3 4]

a =
      1      2      3      4

>> a'

ans =
      1
      2
      3
      4

>> transpose(a)

ans =
      1
      2
      3
      4
```

- **Somma e sottrazione**
  - Tra **vettore e scalare**:

```
>> a = [1 2 3 4]

a =
      1      2      3      4

>> a + 1

ans =
      2      3      4      5

>> a - 1

ans =
      0      1      2      3
```

  - Tra **vettore e matrice**:

```
>> a = [1 2 3 4]

a =
      1      2      3      4

>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
      1      2      3      4
      5      6      7      8
      9     10     11     12

>> a + b

ans =
```

```
      2      4      6      8
      6      8     10     12
     10     12     14     16

>> a - b

ans =
      0      0      0      0
     -4     -4     -4     -4
     -8     -8     -8     -8
```

– Tra **matrice e scalare**:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
      1      2      3      4
      5      6      7      8
      9     10     11     12

>> b + 1

ans =
      2      3      4      5
      6      7      8      9
     10     11     12     13

>> b - 1

ans =
      0      1      2      3
      4      5      6      7
      8      9     10     11
```

– Tra **matrice e matrice**:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
      1      2      3      4
      5      6      7      8
      9     10     11     12

>> c = [13 14 15 16; 17 18 19 20; 21 22 23 24]

c =
     13     14     15     16
     17     18     19     20
     21     22     23     24

>> b + c

ans =
     14     16     18     20
     22     24     26     28
     30     32     34     36

>> b - c

ans =
    -12    -12    -12    -12
    -12    -12    -12    -12
    -12    -12    -12    -12
```

- **Prodotto**
  - **Prodotto matriciale**:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> c = [13 14 15; 16 17 18; 19 20 21; 22 23 24]

c =
    13    14    15
    16    17    18
    19    20    21
    22    23    24

>> b * c

ans =
   190   200   210
   470   496   522
   750   792   834
```

  - **Prodotto punto per punto**, in MATLAB è possibile moltiplicare ogni cella di una matrice (o vettore) per la corrispettiva cella della matrice (o vettore) moltiplicata. La keyword utilizzata è `.*`:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> c = [13 14 15 16; 17 18 19 20; 21 22 23 24]

c =
    13    14    15    16
    17    18    19    20
    21    22    23    24

>> b .* c

ans =
    13    28    45    64
    85   108   133   160
   189   220   253   288

>> d = [1 2 3 4]

d =
     1     2     3     4

>> b .* d

ans =
     1     4     9    16
     5    12    21    32
     9    20    33    48
```

- **Potenza**
  - **Potenza matriciale**:

```
>> b = [1 2 3; 4 5 6; 7 8 9]

b =
     1     2     3
     4     5     6
     7     8     9

>> b^2

ans =
    30    36    42
    66    81    96
   102   126   150
```

  - **Potenza punto per punto**, come per il prodotto, è possibile elevare al quadrato ogni valore della matrice (o vettore):

```
>> b = [1 2 3; 4 5 6; 7 8 9]

b =
     1     2     3
     4     5     6
     7     8     9

>> b.^2

ans =
     1     4     9
    16    25    36
    49    64    81
```

Funzioni intrinseche per vettori e matrici

Qua di seguito si elencano le funzioni più importanti da utilizzare per i vettori e le matrici.

- `size`, restituisce la dimensione del vettore o della matrice nel formato *`righe colonne`*. Specificando anche un valore (o vettore) come parametro, la funzione restituisce la dimensione (un vettore contenente le dimensioni richieste) nella "dimensione" richiesta:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> size(a)

ans =
     1     4

>> size(a, 2)
ans =

     4

>> b = [1 2 3; 4 5 6; 7 8 9]

b =
     1     2     3
     4     5     6
     7     8     9

>> size(b)

ans =
     3     3

>> size(b, [2, 3])

ans =
     3     1
```

- `length`, restituisce la lunghezza del vettore e per le matrici restituisce il numero degli elementi per ogni riga:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> length(a)

ans =
     4

>> b = [1 2 3 4 5 6; 7 8 9 10 11 12]

b =
     1     2     3     4     5     6
     7     8     9    10    11    12

```

```
>> length(b)

ans =
     6
```

- `max`, `min`, calcolano rispettivamente il massimo e il minimo valore delle componenti di un vettore; per le matrici viene presa in considerazione ogni colonna e calcolato il massimo o minimo:

```
>> a = [ 1 2 3 4]

a =
     1     2     3     4

>> max(a)

ans =
     4

>> min(a)

ans =
     1

>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

ans =
     7     1     1     4
     2     3     9    10
     8     1     7     1

>> max(b)

ans =
     8     3     9    10

>> min(b)

ans =
     2     1     1     1
```

- `sum`, `prod`, calcola rispettivamente la somma e il prodotto degli elementi che compongono il vettore; nel caso di una matrice, viene presa in considerazione ogni colonna e calcolata la somma o il prodotto. Inoltre, i due comandi possono prendere un argomento in più per eseguire il calcolo in una dimensione specifica (cosa sensata con le matrici):

```
>> a = [ 1 2 3 4]

a =
     1     2     3     4

>> sum(a)

ans =
    10

>> prod(a)

ans =
    24
```

```
>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

b =
      7      1      1      4
      2      3      9     10
      8      1      7      1

>> sum(b) % per colonne

ans =
     17      5     17     15

>> sum(b, 2) % per righe

ans =
     13
     24
     17

>> prod(b)

ans =
    112      3     63     40
```

- `norm`, la norma di un vettore o di una matrice. Passando un vettore o un matrice, viene calcolata di default la norma euclidea (norma 2):

$$||\mathrm{v}||_2 = \sqrt{\sum_{i=2}^{\texttt{length(v)}} \mathrm{v}_i^2}$$

  Passando un valore aggiuntivo, esso rappresenterà l'ordine della norma:

$$||\mathrm{v}||_n = \left(\sum_{i=2}^{\texttt{length(v)}} |\mathrm{v}_i|^n\right)^{\frac{1}{n}}$$

  Infine, con `inf` viene calcolata la norma infinito:

$$||\mathrm{v}||_\infty = \max_{1 \leq i \leq \texttt{length(v)}} |\mathrm{v}_i|$$

```
>> a = [1 2 3 4]

a =
      1      2      3      4

>> norm(a)

ans =
      5.477225575051661e+00

>> norm(a, 2)

ans =
      5.477225575051661e+00

>> norm(a, 3)
```

```

ans =
      4.641588833612779e+00

>> norm(a, inf)

ans =
      4

>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

b =
      7      1      1      4
      2      3      9     10
      8      1      7      1

>> norm(b, 2)

ans =
      1.711222312384884e+01

>> norm(b, inf)

ans =
     24
```

- `abs`, rappresenta il valore assoluto e restituisce il vettore o matrice dopo aver applicato il valore assoluto a ciascun elemento:

```
>> a = [-1 -2 -3 -4]

a =
     -1     -2     -3     -4

>> abs(a)

ans =
      1      2      3      4

>> b = [7 1 1 -4; 2 3 -9 10; 8 1 -7 1]
b =

      7      1      1     -4
      2      3     -9     10
      8      1     -7      1

>> abs(b)

ans =
      7      1      1      4
      2      3      9     10
      8      1      7      1
```

Riferimenti bibliografici

[1] A. Quarteroni, F. Saleri, and P. Gervasio. *Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave*. UNITEXT. Springer Milan, 2017.

Index