Calcolo Numerico

A.A. 2013-2014

Esercitazione n. 2

10-03-2014

Lavorare con MATLAB

- In MATLAB tutte le variabili sono trattate come **matrici** (non a caso l'acronimo MATLAB sta per **MATrix LABoratory**). Anche gli scalari sono visti come una matrice.

scalari	->	matrici 1 x 1
vettori riga	->	matrici 1 x n
vettori colonna	->	matrici n x 1
matrici	->	matrici m x n

$\mathbf{v} = (v_1, \ldots, v_n)$

$\mathbf{v} = (v_1, \ldots, v_n)^T$

$$A = \begin{pmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{m1} \ldots a_{mn} \end{pmatrix}$$

Vettori riga

- In Matlab si possono definire facilmente vettori e matrici
- Un **vettore** si definisce elencando le sue componenti separate da uno spazio e racchiudendole tra parentesi quadre [ ]

- **Vettore riga**

```
>>  x = [10 20 30 40]
x =
     10     20     30     40
```

è equivalente a

```
>>  x = [10,20,30,40]
x =
     10     20     30     40
```

  - in questo caso le componenti sono separate da una virgola

Vettori colonna

- **Vettore colonna**

```
>> x=[10; 20; 30; 40]
x =
      10
      20
      30
      40
```

  - anche per visualizzare il contenuto di variabili che sono vettori si può usare il comando **disp**

```
>> disp(x)
     10
     20
     30
     40
```

Vettori

- Per convertire un vettore riga in uno colonna (e viceversa) si usa il comando ' (**apice**) che produce il trasposto della variabile a cui è applicato

```
>>  v=x'

v =

     10     20     30     40
```

- Per estrarre un elemento di un vettore:

```
nome_vettore(posizione elemento)
```

Esempio: estrarre il secondo elemento di v

```
>> v(2)

ans =

     20
```

Nota: Gli indici di un vettore sono sempre numeri interi e strettamente positivi e la numerazione parte da 1!

Il comando :

Per **estrarre** contemporaneamente più di un elemento di un vettore si usa il comando **:** (colon)

```
nome_vettore(inizio:fine)
```

Esempio: estrarre dal primo al terzo elemento di v

```
>> v(1:3)

ans =

     10     20     30
```

Esempio: estrarre dal terzo al quarto elemento di v

```
>> v(3:4)

ans =

     30     40
```

Il comando :

Esempio: estrarre tutti gli elementi di v

```
>> v(1:end)
ans =
     10     20     30     40
```

Oppure, se lo si vuole come vettore colonna,

```
>> v(:)
ans =
     10
     20
     30
     40
```

Il comando :

Per estrarre contemporaneamente più di un elemento di un vettore non consecutivi ed equispaziati

```
nome_vettore(inizio:passo:fine)
```

Esempio: estrarre gli elementi di v di indice pari (passo = 2)

```
>> v(2:2:end)
ans =
     20     40
```

Esempio: estrarre tutti gli elementi di v di indice pari ma da destra verso sinistra (passo = -2)

```
>> v(end:-2:1)
ans =
     40     20
```

Vettori

- Un vettore può essere usato per estrarre elementi non consecutivi e non equispaziati di un altro vettore

```
nome_vettore([pos1 pos2 pos3 …])
```

Esempio: Sia v = [7 1 3 7 0 8 3], estrarre gli elementi di v di indici 1 3 e 6

```
>> v([1 3 6])
ans =
     7     3     8
```

- Un' operazione che può risultare utile è quella di **eliminare** alcuni elementi in un vettore cambiandone allo stesso tempo la dimensione

```
>> x = 1:10;
>> x(1:3)=[]
x =
4 5 6 7 8 9 10
```

Generazione di vettori con :

Il comando : può essere usato anche per **generare vettori**

```
Nome_vettore = (minimo:incremento:massimo)
```

Esempio: Generare un vettore costituito da elementi compresi tra 1.5 e 2 con incremento 0.1

```
>> x=[1.5:0.1:2]

x =

    1.5   1.6   1.7   1.8   1.9   2.0
```

Esempio: Generare un vettore costituito da elementi compresi tra 100 e 80 con incremento -5

```
>> x=[100:-5:80]

x =

    100    95    90    85    80
```

Generazione di vettori con :

Esempio:

```
>> x=[3:0]
non produce niente!
```

- se non specificato, l'incremento è da intendersi pari a 1

Comando linspace

- Per generare **vettori equispaziati** contenuti in un certo intervallo si può usare anche il comando **`linspace`**
  - può essere molto utile nel caso si consideri un passo che non sia intero
  - al comando devono essere forniti come parametri di ingresso i due estremi dell'intervallo e il numero di elementi del vettore N (per default è 100). Restituisce un vettore di lunghezza N il cui i-esimo elemento è dato da

```
Nome_vettore=linspace(minimo, massimo, N)
```

```
x(i) = minimo + (i-1)*(massimo-minimo)/(N-1)
```

Esempio: Generare un vettore costituito da 10 elementi compresi tra 1.5 e 2.4

```
>> x=linspace(1.5,2.4,10)

x =

    1.5    1.6    1.7    1.8    1.9    2.0    2.1    2.2    2.3
  2.4
```

Vettori

- Se un vettore (o una qualsiasi istruzione) è troppo lunga, prima di andare a capo vanno aggiunti 3 punti **…**

```
>> x = [3 1 6 7 9 10 4 29 6 0 …
           4 5 8 2 4 ]
x =
    3   1   6   7   9   10   4   29   6   0   4   5   8   2   4
```

- Se un elemento di un vettore è una espressione, non bisogna lasciare spazi all'interno dell'elemento, oppure l'espressione va racchiusa tra parentesi tonde

```
>> x = [1  6  3*2+1  4]
x =
     1     6     7     4
```

Oppure

```
>> x = [1  6  (3*2+1)  4]
x =
```

Vettori

I vettori non vengono dimensionati. **La loro dimensione può essere modificata in corso di lavoro**

Esempio: Sia **`x = [3 1 4 5]`** e si assegni il valore 10 all'ottavo elemento di x

```
>> x =  [3 1 4 5]

x =

     3     1     4     5

>> x(8) = 10

x =

     3     1     4     5     0     0     0    10
```

alle posizioni non definite viene assegnato il valore zero

Vettori

Esempio: Sia `x = [3 1 4 5]` e si elimini l'elemento in posizione 3

```
>> x(3) =  []
x =
     3     1     5
```

[] indica il **vettore vuoto**

Per conoscere la lunghezza di un vettore si usa il comando `length(x)`

Esempio: determinare la lunghezza del vettore x sopra definito

```
>> length(x)
ans =
     3
```

Operazioni

In MATLAB sono definite le operazioni dell'algebra lineare numerica di **moltiplicazione per uno scalare** e di **somma** e **sottrazione** tra vettori. Tali operazioni agiscono **componente per componente** e restituiscono un vettore della stessa lunghezza

```
>> x = 1:5;
>> y = [50 10 30 40 20];
>> 2*x
ans =
2 4 6 8 10
>> 2./x
ans =
2.0000 1.0000 0.6667 0.5000 0.4000
>> x+y
ans =
51 12 33 44 25
>> y-x
ans =
49 8 27 36 15
```

```
>> x(1:3)+y
??? Error using ==> plus
Matrix dimensions must agree
```

Operazioni puntuali

Matlab estende le proprietà delle operazioni tipo somma e sottrazione anche ad altre operazioni, fra cui moltiplicazione e elevamento a potenza. Il vincolo è che i due vettori operandi abbiano lo **stesso numero di componenti**

```
>> a = 1:3;
>> b = a;
>> a.*b
ans =
1 4 9
>> a.^b
ans =
1 4 27
```

Se le dimensioni non sono compatibili:

```
>> c= [1 2];
>> a.*c
Matrix dimensions must agree
```

Operazioni puntuali

Le operazioni precedenti (moltiplicazione puntuale, la divisione puntuale e l'elevamento a potenza puntuale) sono tipiche dell'ambiente MATLAB

- non hanno un corrispondente dal punto di vista dell'algebra lineare in quanto agiscono su vettori e matrici intesi come strutture di dati più che entità matematiche.
- L'istruzione **`x.*y`** utilizza la moltiplicazione puntuale tra vettori e fornisce un vettore con la proprietà che ogni sua componente è uguale al prodotto delle corrispondenti componenti dei vettori x e y.
- le stesse operazioni possono essere applicate nel caso di vettori colonna o più in generale nel caso di matrici. La cosa essenziale è che gli operandi siano dello stesso tipo ed abbiano le stesse dimensioni.

Operazioni

Uniche eccezioni a questa regola sono date dal caso in cui le precedenti operazioni vengano applicate tra un vettore ed una costante. In tal caso MATLAB considererà la costante come un vettore di pari dimensioni avente tutte componenti costanti. Ad esempio

```
>> x = 1:5;
>> x+1
ans =
2 3 4 5 6

>> 1-x
ans =
0 -1 -2 -3 -4

>> x.^2
1 4 9 16 25
```

Vettorizzazione e operazioni puntuali

- Molte funzioni predefinite in MATLAB accettano come argomenti dei vettori
  - questa caratteristica di MATLAB è molto importante in quanto consente di scrivere in forma molto chiara e compatta sequenze di istruzioni eliminando in molti casi l'uso di strutture e cicli che agiscono elemento per elemento

```
>> v = [4 9 16]
>> radici = sqrt(v)
   radici =
   2     3     4
```

Vettorizzazione e operazioni puntuali

Esempio: per costruire una semplice tabella di valori della funzione coseno nell'intervallo [0, π] possiamo procedere nel seguente modo

```
>> n = 5;
>> x = linspace(0,pi,n);
>> c = cos(x);
```

L'istruzione **c=cos(x)** applicata ad un vettore x restituisce un vettore c di uguali dimensioni e tipo con la proprietà che l'elemento di indice i è **c(i) = cos(x(i)).** Risulta quindi equivalente all'istruzione

```
>> c = cos(x);
>> c=[cos(x(1)) cos(x(2)) cos(x(3)) cos(x(4) cos(x(5))]
```

Funzioni

- Ecco alcune **funzioni MATLAB** che consentono di costruire particolari matrici e vettori.
  - queste funzioni MATLAB possono essere utilizzate con un diverso numero di parametri.
  - si consulti l'help per una descrizione dettagliata

Funzione	Significato
`linspace`	vettore riga di elementi equispaziati
`logspace`	vettore riga di elementi equispaziati in scala logaritmica
`zeros`	matrice contenente solo elementi uguali a zero
`ones`	matrice contenente solo elementi uguali a uno
`rand`	matrice contenente numeri casuali
`eye`	matrice identità
`diag`	matrice diagonale
`magic`	matrice a valori interi con somme uguali su righe e colonne

Funzioni

```
length(v) max(v) min(v) sum(v) norm(v) abs(v) sort(v)
```

Esempio

```
>> zeros(1,3)
   ans =
   0     0     0

>> ones(4,1)
   ans =
   1
   1
   1
   1
```

max e min

- **max (min)** restituisce il valore massimo (minimo) contenuto nel vettore. Se si usa la funzione con due parametri di output il primo valore è il valore massimo, il secondo l'indice del vettore per cui si ha il valore massimo

Esempio

```
>> v = [2 4 1 6 9 3];
>> max(v)
   ans =
    9
>> [massimo i_massimo] = max(v)
   massimo =
    9
    i_massimo =
    5
>> disp(v(i_massimo))
    9
```

Esercizi

- Creare un **vettore** **`x`** che ha componenti con valori compresi tra 0 e 100, estremi inclusi, con incremento costante pari a 0.5.
  - determinarne la lunghezza
  - estrarne gli elementi di indice pari e assegnarli alla variabile y
  - estrarre gli elementi di indice dispari di y procedendo da destra verso sinistra e assegnarli alla variabile z
  - eliminare il primo elemento di z
  - assegnare il valore –9 al ventesimo elemento di z.
  - sostituire il secondo elemento di z con 2*cos(π/4)
  - creare il vettore w costituito da tutti gli elementi di y seguiti dai primi 2 elementi di z e gli ultimi 3 di x
  - visualizzare il vettore w
  - calcolare massimo e minimo di w
  - utilizzare la funzione **`sort`** (cosa fa?)

Grafica

- MATLAB è molto più di un semplice software in grado di eseguire calcoli numerici (anche se in forma molto sofisticata). Con MATLAB è possibile realizzare **grafici di funzioni** anche in più dimensioni e realizzare veri e propri programmi. In MATLAB è possibile
  - disegnare funzioni in 2D e 3D
  - rappresentare dati memorizzati in vettori e matrici in molti modi differenti
- Il comando **`plot(x,y)`** si usa
  - per **rappresentare punti nel piano**
  - per disegnare il **grafico di una funzione** **`y=f(x)`**
    - **`x e y`** devono essere vettori di ugual misura

Figure

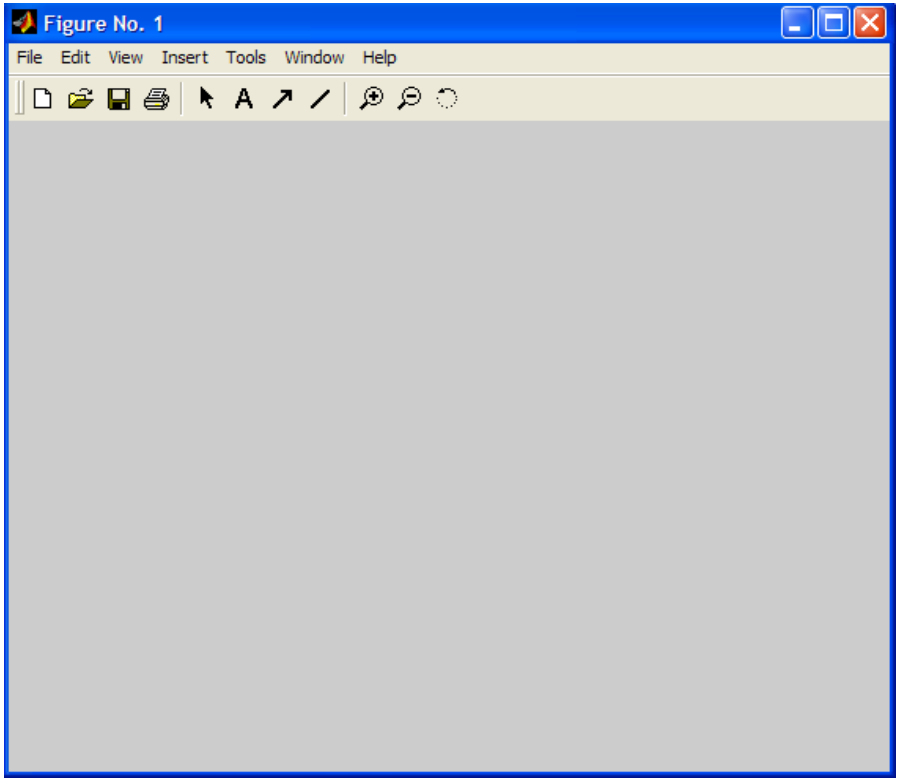
`figure` apre la finestra grafica al quale viene associato un numero

`figure(n)` `n` è il numero associato alla finestra grafica

Ad ogni figura è associato un identificativo che consente di gestirne le proprietà grafiche

```
>> figure(1)
```

```
>> id = figure;
```



Plot - sintassi

```
plot(vettore_x, vettore_y, 'opzioni')
```

- **vettore_x** e **vettore_y** devono avere lo stesso numero di componenti, sono i vettori dei dati (ascisse e ordinate dei punti)
- **opzioni:** è una stringa opzionale che definisce il tipo di colore, di simbolo e di linea usato nel grafico

Esempi colore: **m** magenta, **r** rosso, **g** verde, **b** blu, **w** bianco, **k** nero, **y** giallo

Esempi tipo di linea: **-** continua (default), **--** tratteggiata, **:** punteggiata, **-.** Punto-linea

Esempi simbolo: **+** croce, **o** cerchietto, ***** asterisco, **x** ics,….

help plot per vedere quali sono le varie opzioni

Plot - opzioni

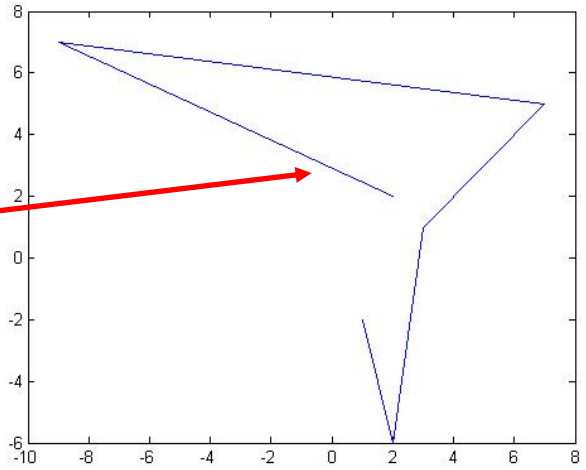
- Alcune opzioni del comando plot

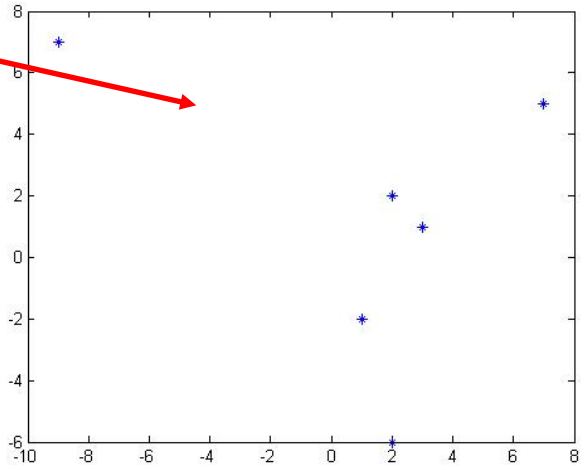
Colore		Simbolo		Linea	
y	giallo	.	punto	-	linea continua
m	magenta	o	circoletto	:	linea punteggiata
c	ciano	x	per	-.	linea punto
r	rosso	+	più	--	linea tratteggiata
g	verde	*	asterisco		
b	blu	s	quadratino		
w	bianco	d	diamante		
k	nero	v	triangolo		

Rappresentazione di punti

Per rappresentare dei punti nel piano

```
>> x = [1 2 3 7 -9 2];
>> y = [-2 -6 1 5 7 2];
>> plot(x,y)
>> figure(2)
>> plot(x,y,'*')
```





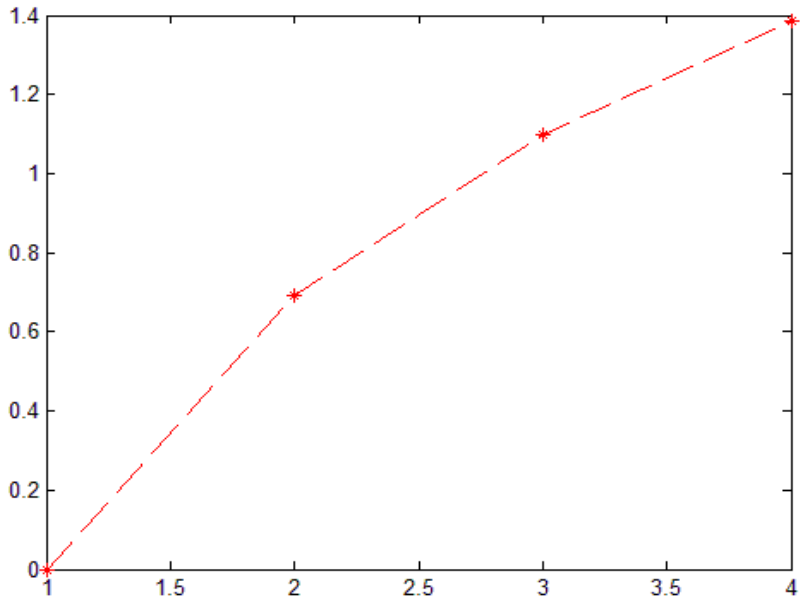

```
>> plot(y)
```

realizza il grafico del vettore y rispetto ai propri indici

Grafici di funzione

- Esempio: Siano **`x = [1 2 3 4]`** e **`y = log(x)`**. Disegnare **`y`** in funzione di **`x`** usando una linea tratteggiata rossa e marcando i punti della curva con asterischi
  - si creano due vettori x e y contenenti rispettivamente la successione di valori nell'intervallo ed i corrispondenti valori della funzione

```
>> x = 1:4;
>> y = log(x);
>> plot(x,y, 'r--*')
```



Se volessimo produrre un grafico privo di spigoli è sufficiente aumentare il numero di punti in modo che i segmenti di raccordo siano cosi piccoli da dire l'idea di una linea continua

Esempio

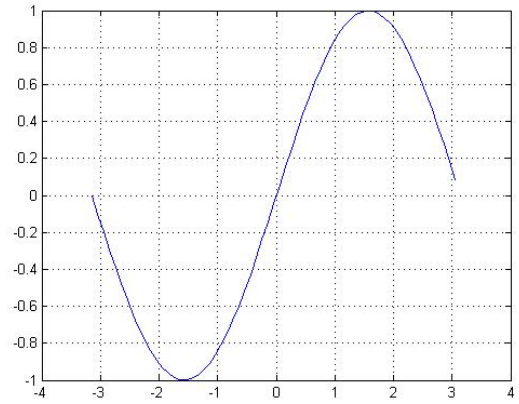
Per plottare la funzione **y=sin(x)**

```
x = [-pi:.01:pi];
y = sin(x);
plot(x,y)
```

definiamo l'intervallo in cui vogliamo disegnare la funzione

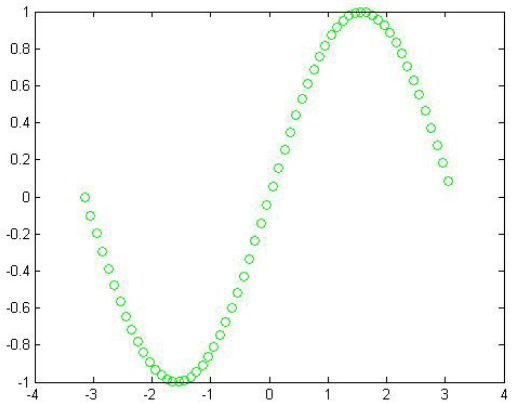
definiamo la funzione

disegniamo la funzione



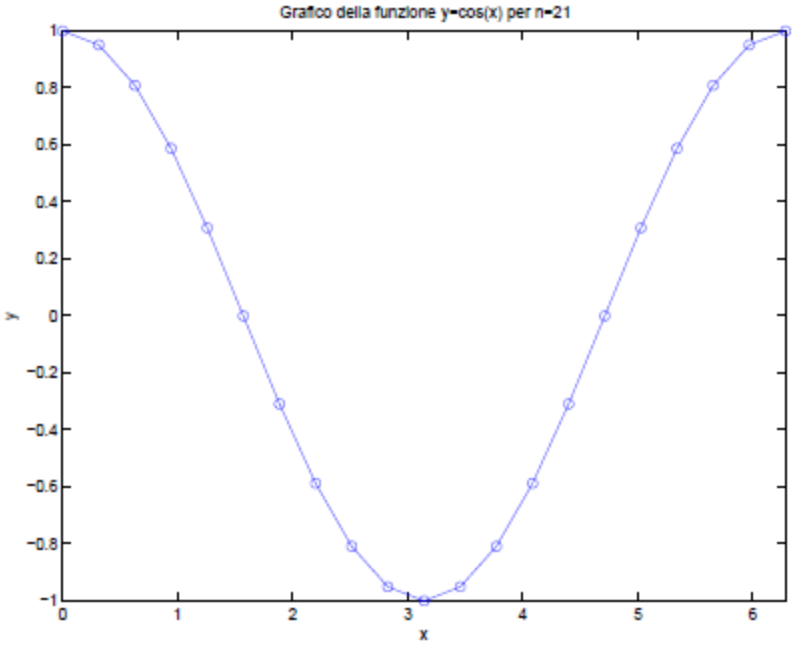

```
figure(2)
plot(x,y, '°g')
```

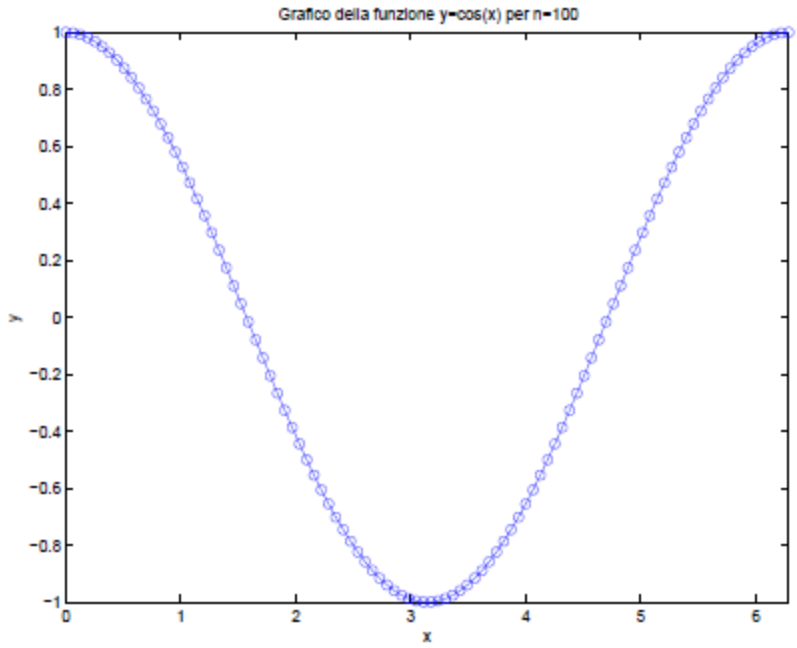
è possibile inserire un terzo parametro di input



Esempio

Rappresentazione grafica della funzione **`y=cos(x) in [0, 2π]`**





Grafici sovrapposti

- Sia **x** il vettore delle ascisse e **y1 e y2** due vettori aventi la stessa lunghezza di **x**. Per visualizzare sulla stessa finestra si usa il comando **hold on** che funziona come un interruttore acceso/spento; tale comando fa sovrapporre tutti i grafici successivi nella stessa finestra grafica fino a quando non si digita il comando **hold off**

```
>> figure
>> plot(x,y1)
>> hold on
>> plot(x,y2)
>> hold off
```

- Equivalentemente si può usare la seguente istruzione che usa in modo automatico linee tipo differenti per i diversi grafici

```
>>  plot(x,y1,'opzioni',x,y2,'opzioni')
```

  - (le opzioni si possono omettere)

Grafici

Esistono molte possibilità per personalizzare un grafico: etichette

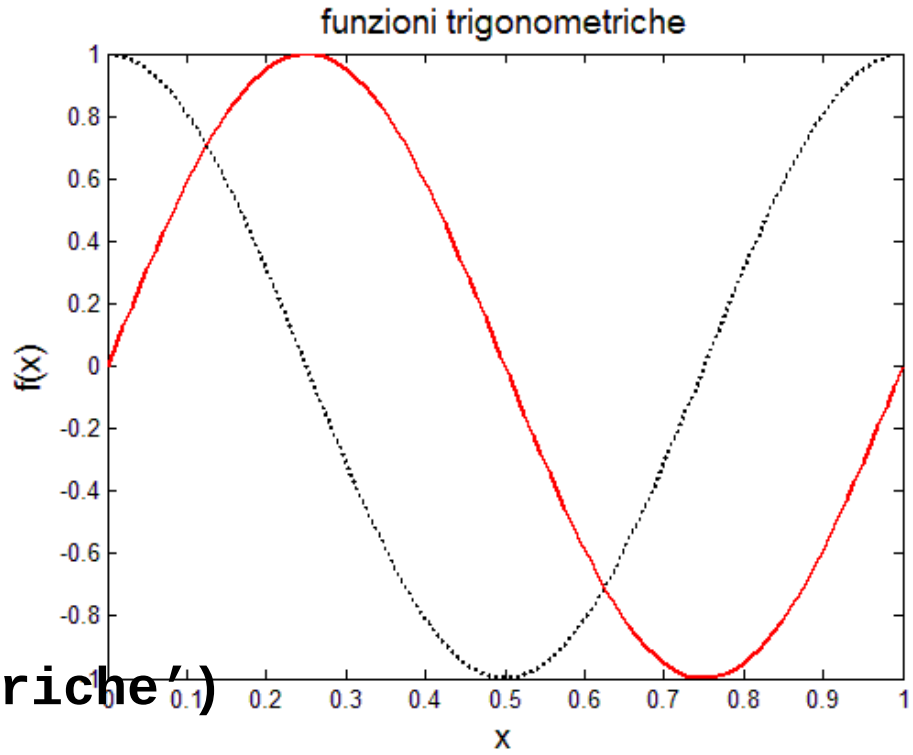
`title('stringa')`	titolo del grafico
`xlabel('stringa')`	etichetta per l' asse delle ascisse (asse x)
`ylabel('stringa')`	etichetta dell'asse delle ordinate (asse y)
`grid`	inserisce una griglia nel grafico
`legend('stringa1', 'stringa2',…)`	legenda
`axis([xmin xmax ymin ymax])`	regola la dimensione degli assi coordinati (determina il rettangolo nel quale si vogliono visualizzare i dati).
`axis('equal')`	usa la stessa scala sulle ascisse e le ordinate

`set, get`

Grafici

Esempio: Sia x un vettore di 10 elementi equidistanti e contenuti nell'intervallo [0,1]. Definire i vettori $y = \sin(2\pi x)$ e $z = \cos(2\pi x)$. Disegnarli sullo stesso grafico usando tratti diversi e colori diversi nel rettangolo [0 1]x[-1 1]. Etichettare gli assi rispettivamente con 'x' e 'f(x)' e dare il seguente titolo 'funzioni trigonometriche'.

```
>> x = linspace(0,1,10);
>> y = sin(2*pi*x);
>> z = cos(2*pi*x);
>> figure
>> plot(x,y,'r',x,z,'b:')
>> axis([0 1 -1 1])
>> xlabel('x')
>> ylabel('f(x)')
>> title('funzioni trigonometriche')
```



Sottografici

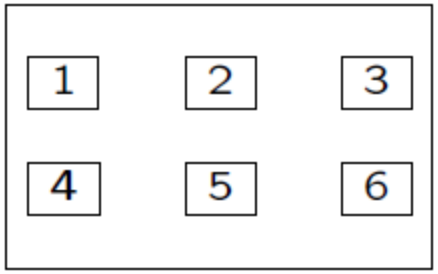
Spesso ci si pone il problema di disegnare diversi grafici separati in una stessa finestra. L'obiettivo può essere raggiunto facilmente utilizzando la funzione subplot la cui sintassi è

```
subplot(Righe, Colonne, Sottofinestra)
subplot(m, n, k)
```

suddivide la finestra dei grafici in una matrice mxn e attiva la k-ma sottofinestra. Il successivo comando plot disegna il grafico nella finestra attivata

Esempio: `subplot(2,3,4)`

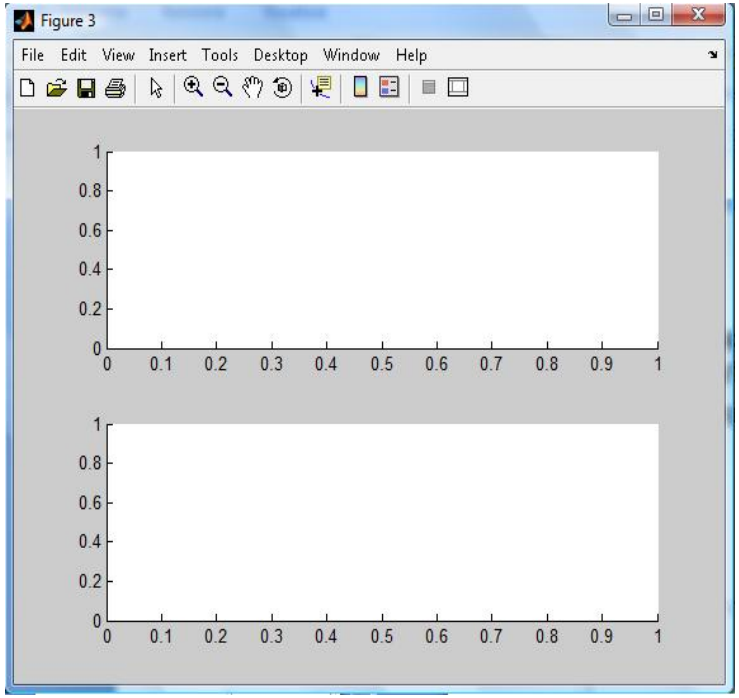
Tale istruzione suddivide la finestra in una matrice 2 × 3 di sottofinestre ed attiva la quarta sottofinestra grafica. Le sottofinestre sono numerate come segue



Sottografici

Esempio

```
>>subplot(2,1,1)
divide la finestra grafica in 2
sottofinestre poste una sotto
l'altra e attiva la prima
>>plot(x1,y1);
>>title('grafico1')
>>subplot(2,1,2);
divide la finestra grafica in 2
sottofinestre poste una sotto
l'altra e attiva la seconda
>>plot(x2,y2); title('grafico2')
```



fplot

```
fplot(funzione,limiti_assi,'opzioni')
```

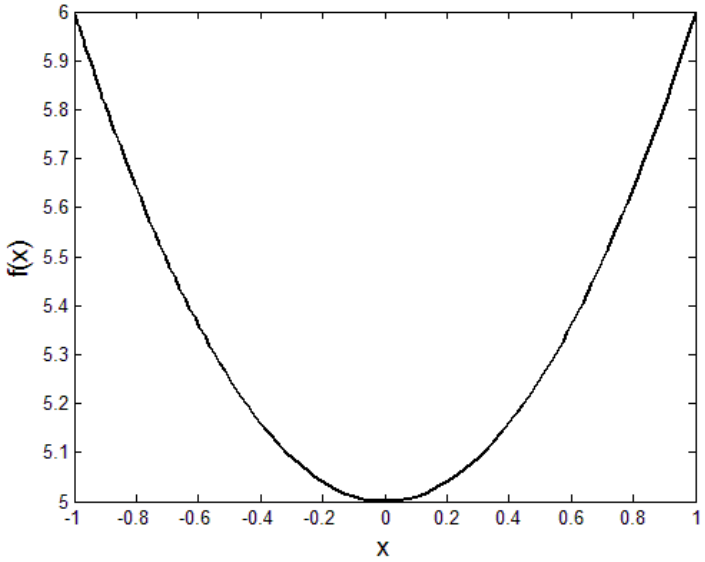
- **`funzione`** puntatore ad una funzione

```
>> f = @(x)[x^2+5]
    f =
        @(x)[x^2+5]
```

- **`limiti_assi : [xmin xmax]`** è un vettore contenente i limiti inf e sup dell'intervallo in cui si vuole visualizzare la funzione;

Esempio:

```
>> fplot(f,[-1 1])
>> xlabel('x'), ylabel('f(x)')
```



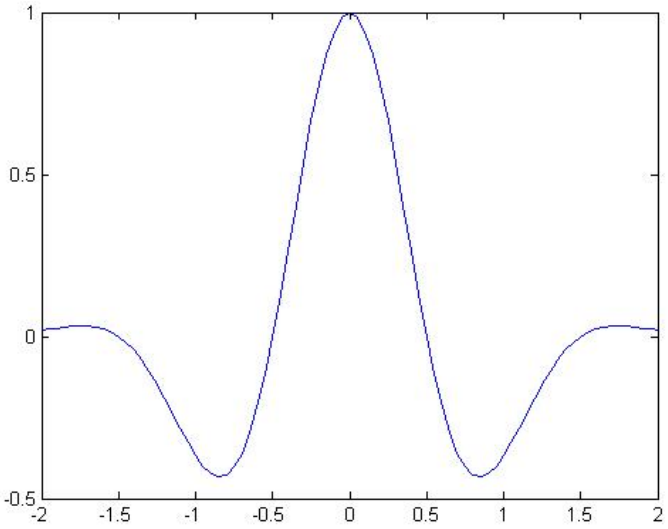
Max

- Trovare il max della funzione

$$f(x) = e^{-x^2}\cos(\pi x)$$

nell'intervallo [-2,2]

```
x = [-2:.05:2];
y = exp(-x.^2).*cos(pi*x);
plot(x,y)
maximo = max(y);
```



Esercizi

- Disegnare la parabola $y = x^2+2x$ nell'intervallo [-10,10] e calcolarne le coordinate del vertice

- Disegnare tra 0 e 2π la funzione sin(x) e le sue approssimazioni di Maclaurin del 1° e 5° grado
  - le 3 curve devono essere rispettivamente a tratto continuo, tratteggiata e marcata con cerchietti

- Disegnare un cerchio

- Sia **`x = 1:10`**, **`y1=atan(x) e y2=log(x)`**. Disegnare y1 e y2 marcandoli opportunamente

Errori

Le sorgenti di errore possono essere, in generale, suddivise in cinque gruppi

- **Errori di impostazione del problema**
  - non sempre le formule matematiche rappresentano modelli esatti dei fenomeni reali; generalmente questi modelli sono più o meno idealizzati. Infatti, nello studio dei vari fenomeni della natura, siamo costretti, al fine di rendere più semplice il problema, ad ammettere certe condizioni e ciò implica una serie di errori.

- **Errori di metodo**
  - talvolta succede che è difficile, se non impossibile, risolvere un problema impostato in termini rigorosi. In questo caso il problema in esame viene sostituito da un altro problema approssimato i cui risultati si distinguono di poco dal problema dato

Errori

- **Errori di troncamento**
  - le funzioni che appaiono in formule matematiche sono spesso date in forma di successioni infinite o di serie e numerose equazioni matematiche non possono essere risolte che attraverso una descrizione di processi infiniti, i cui limiti sono appunto le soluzioni cercate. Siccome un processo infinito non può, in generale, terminare in un numero finito di passi, siamo costretti a interrompere la successione infinita di passi computazionali, necessaria per ottenere un risultato esatto.
- **Errori di rappresentazione dei dati o di Arrotondamento**
  - non tutti i numeri reali possono essere rappresentati in un calcolatore. Se il calcolatore utilizza una rappresentazione con t cifre per la mantissa, tutti i numeri reali compresi tra l'estremo inferiore e l'estremo superiore di F la cui mantissa ha un numero di cifre superiore a t dovranno in qualche modo essere arrotondati a t cifre.

Errori

- **Errori nei calcoli**
  - effettuando calcoli tra numeri macchina approssimati, gli errori di rappresentazione dei dati iniziali si propagano in qualche misura sui risultati di tali calcoli. L'entità della propagazione dipende anche dall'algoritmo utilizzato.

- L'analisi degli errori è fondamentale per comprendere come evitarli, o (qualora non fosse possibile evitarli) ridurli al minimo

Cancellazione numerica

- La cancellazione numerica è la perdita di cifre significative
- E' un fenomeno che si verifica durante l'operazione di **sottrazione** tra due numeri "quasi uguali"
  - se due numeri sono quasi uguali, dove uguali s'intende a meno della precisione macchina, allora è possibile il verificarsi della cancellazione numerica.
- Siano *x1 e x2* due numeri reali. Se *x=x1-x2* è "molto piccolo", l'errore relativo

$$\delta_x = \left| \frac{fl(x1 - x2) - x}{x} \right|$$

può essere molto grande e ciò produce una perdita di cifre significative nel calcolo di *fl(x1-x2)*

- E' sempre preferibile evitare la sottrazione tra numeri macchina "quasi uguali"

Procedimento di calcolo ed accuratezza del risultato

- Consideriamo il calcolo della funzione

$$f(x) = \int_0^1 e^{xt}\,dt \qquad \Rightarrow \qquad \begin{cases} \dfrac{e^x - 1}{x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

supponiamo che x sia dato con precisione $10^{-9} \Rightarrow x = 1.4 \cdot 10^{-9}$

$$f(x) = \frac{1.000000001 - 1}{1.4 \cdot 10^{-9}} = \frac{0.000000001}{1.4 \cdot 10^{-9}} \approx 0.714$$

Si perde l'accuratezza della misura! Dell'ordine di 10^{-1}

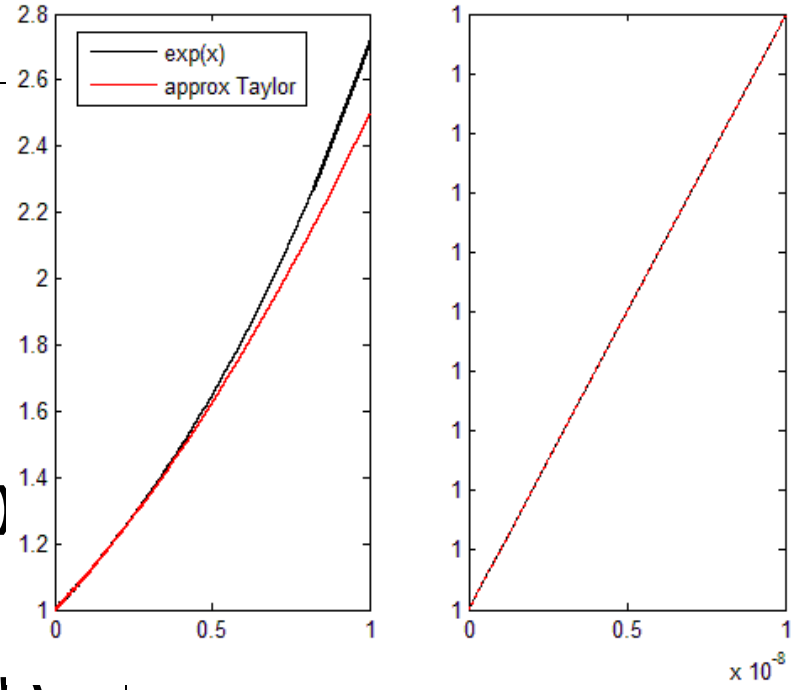
Procedimento di calcolo ed accuratezza del risultato

- Consideriamo lo sviluppo in serie di Taylor di f(x) attorno 0

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}e^{\xi} \qquad 0 \le \xi < x \le 1$$

L'errore dell'approssimazione data dallo sviluppo in serie è dell'ordine di 10^{-28}

```
>> f = @(x)[exp(x)];
>> f2 = @(x)[1+x+x.^2/2];
>> figure,
>> subplot(1,2,1),
>> fplot(f,[0 1],'k'), hold on,
>> fplot(f2,[0 1],'r'),
>> legend('exp(x)','approx Taylor')
>> subplot(1,2,2),
>> fplot(f,[0 10^-8],'k:'),
>> hold on, fplot(f2,[0 10^-8],'r:')
```



Procedimento di calcolo ed accuratezza del risultato

- L'errore sulla misura di f(x) ha lo stesso ordine di grandezza dell'errore sulla misura di x

$$\Rightarrow \qquad f(x) = \frac{e^x - 1}{x} \approx 1 + \frac{x}{2} \approx 1 + \frac{1.4}{2} 10^{-9} = 1 + 0.7 \cdot 10^{-9}$$

- Il secondo algoritmo è più **stabile**
- Uno dei modi per evitare la cancellazione numerica è quella di trovare un'espressione più stabile tale da non far aumentare gli errori introdotti dalla formulazione del problema

Esercizio

- Considerare tre forme equivalenti di uno stesso polinomio:

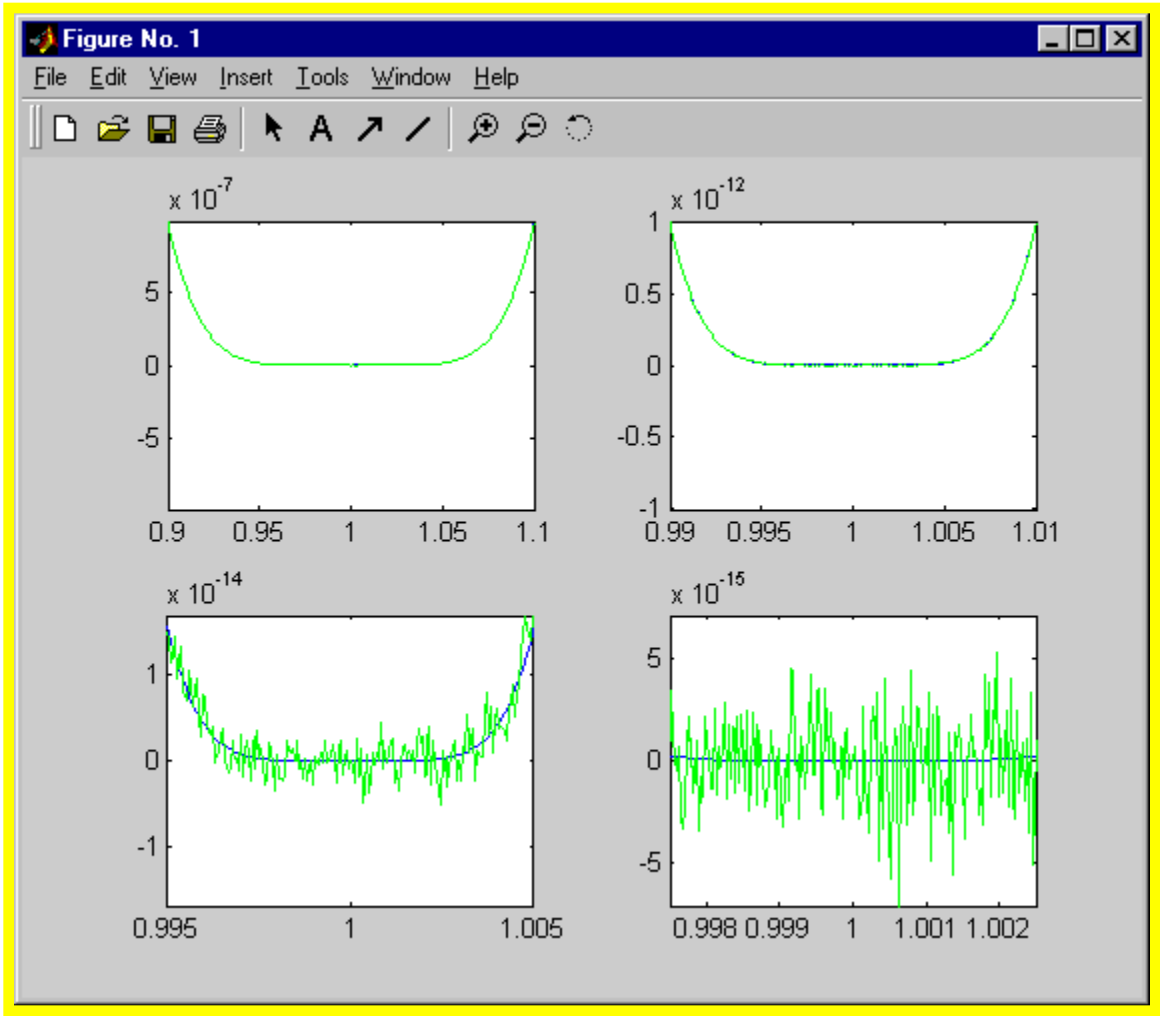
$$\begin{aligned} p(x) &= (x-1)^6 \\ &= x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1 \\ &= 1 + x(-6 + x(15 + x(-20 + x(15 + x(-6 + x))))) \end{aligned}$$

- Stabilire se, valutate al calcolatore in uno stesso punto, forniscono lo stesso risultato

Esempio di cancellazione numerica

```
%Calcola (1-x)^6 con le due formule:
%y1 = (1-x)^6
%y2 = 1-6x +15x^2 -20x^3 +15x^4 -6x^5 +x^6
%e confronta y1 e y2 in un intorno di uno
%
k = 0
for delta = [0.1, 0.01, 0.005, 0.0025]
  h = delta/100;
  x = 1-delta:h:1+delta;
  y1 = (1-x).^6;
  y2 = 1 -6*x +15*x.^2 -20*x.^3 +15*x.^4 -6*x.^5 + x.^6;
  k = k+1;
  subplot(2,2,k)
     plot(x,y1)
     hold
     plot(x,y2,'g')
     axis([1-delta 1+delta -max(abs(y2)) max(abs(y2)) ])
end
```

L'output dello script precedente è:



Esercizi

- Assegnare alla variabile **x** il vettore costituito dai primi 20 numeri naturali. Estrarne il quarto elemento e moltiplicarlo per il quindicesimo
- Costruire il vettore v di 40 elementi **`v = [1,2,…,20,20,19,…,1]`**
- Creare un vettore **x** che ha 6 componenti con valori compresi tra 0 e 10, estremi inclusi, con incremento costante.
- Assegnati i vettori **`u = [1; 0; 2; -3] e v = [3; 0; 2; 1]`**
  - calcolarne il prodotto scalare; cosa fornisce invece il prodotto **`v*u`**?
  - calcolare i vettori colonna **`z;w;y`** definiti, componente per componente, da

```
zi = ui*vi; wi = ui^vi; yi = zi=wi
```

Esercizi

- Consideriamo per ogni $k \in \mathrm{R}$ la funzione seguente

$$f(x) = kx^3 + \sqrt{k}\,x$$

Disegnare il grafico di $f(x)$ nell'intervallo [0,2] e trovare il punto di minimo (nell'intervallo [0,2]) con un'approssimazione di 0.02

- Come evitare il problema della cancellazione numerica nel calcolo delle radice

$$\sqrt{x+\delta} - \sqrt{x}, \quad \delta \to 0$$