



Laboratorio informatico per l'ingegneria elettrica

Università di Pisa
Dipartimento di Sistemi Elettrici e Automazione

Luca Sani

Dipartimento di Sistemi Elettrici e Automazione

Università di Pisa

tel. 050 2217364

email luca.sani@dsea.unipi.it

Sommario

- Importazione di dati
- Le strutture
- Il Symbolic Toolbox
- La GUI

Università di Pisa - Dipartimento di
Sistemi Elettrici e Automazione

Importazione di dati in formato non Matlab

E' possibile importare in Matlab dati prodotti da altri programmi.

Si può operare in due modi:

- con l'import wizard
- con comandi a basso livello

Importazione di dati in formato non Matlab con l'import wizard

Supponiamo di avere un file creato con Microsoft Excel

	A	B	C	D	E
1	x1	x2	x3		
2	1	2	3		
3	1	2	3		
4	1	2	3		
5	1	2	3		
6	1	2	3		
7	1	2	3		
8	1	2	3		
9	1	2	3		
10	1	2	3		
11	1	2	3		
12	1	2	3		
13	1	2	3		
14	1	2	3		
15					
16					
17					

Importazione di dati in formato non Matlab con l'import wizard

Si avvia l'import wizard da menu File->Import Data..

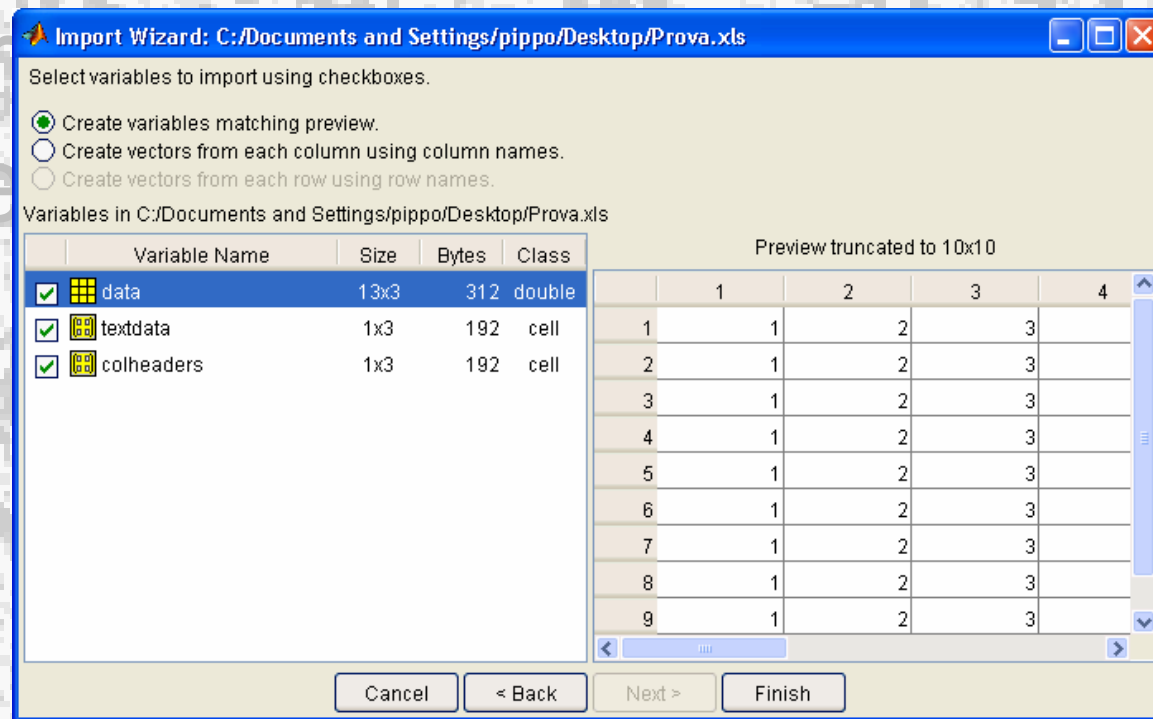
La prima schermata chiede il file da importare.

Una volta immesso, Matlab carica i dati e nella schermata successiva propone il risultato.

Università di Pisa - Dipartimento di
Sistemi Elettrici e Automazione

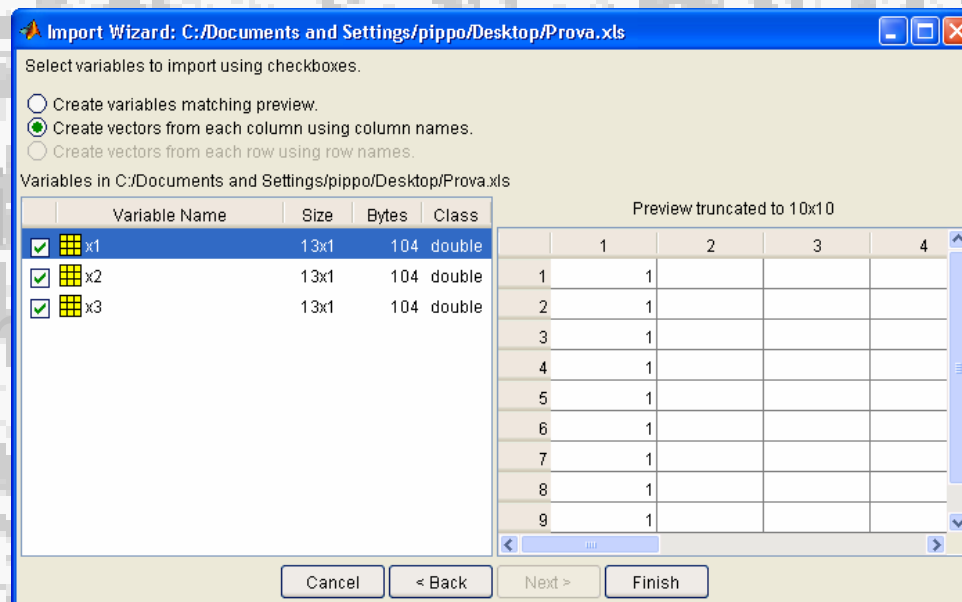
Importazione di dati in formato non Matlab con l'import wizard

Come si vede tra i dati letti (a sinistra), Matlab ha riconosciuto che nel file sono presenti stringhe e numeri



Importazione di dati in formato non Matlab con l'import wizard

Si seleziona l'opzione 'create vectors from each column...'



Premendo il tasto Finish avremo nel workspace di Matlab tre variabili x1, x2 e x3 ognuna corrispondente ad una colonna dei dati letti. Nella Command Window Matlab segnala **Import Wizard created variables in the current workspace.**

Importazione di dati in formato ASCII con istruzioni a basso livello

Supponiamo di avere il seguente file ASCII prova.txt

Esecuzione completata con successo

I risultati sono

x1	x2	x3
----	----	----

0.0	3.2	4.3
-----	-----	-----

0.1	2.8	4.1
-----	-----	-----

0.2	5.9	4.0
-----	-----	-----

.	.	.
---	---	---

.	.	.
---	---	---

.	.	.
---	---	---

Importazione di dati in formato ASCII con istruzioni a basso livello

Per importarlo si può utilizzare la funzione **textread**.

Sintassi

```
nome_var=textread('nomeFile', 'formato_colonna_1 formato_colonna_2 ...', 'headerlines', n)
```

dove

formato_colonna è **%f** per i numeri reali, **%s** per le stringhe, **%d** per gli interi;

n numero di righe di intestazione (comprese quelle bianche) da ignorare.

In questo esempio

```
[x1, x2, x3]=textread('prova.txt', '%f %f %f', 'headerlines',5)
```

crea le tre variabili x1, x2, x3, una per ogni colonna.

Le strutture

Le strutture sono rappresentazioni di dati ben noti ai programmatori di linguaggi ad alto livello quali C e Pascal. Si tratta di elementi che permettono la memorizzazione di dati disomogenei in un'unica variabile (si ricordi che al contrario matrici ed array permettono la memorizzazione solo di elementi dello stesso tipo).

La struttura può essere pensata come una scheda su cui sono memorizzati dati diversi in 'campi' opportuni.

Un dato di tipo struttura può essere creato in due modi:

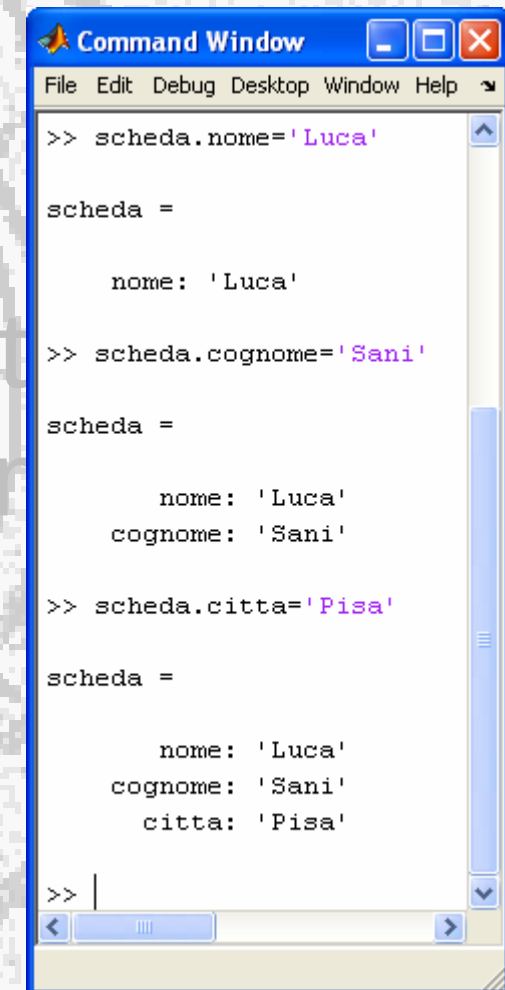
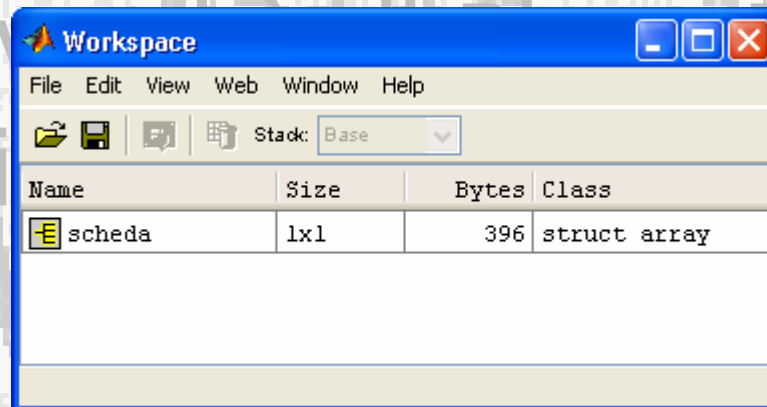
- con comandi di assegnamento
- con il comando struct

Creazione di dati di tipo strutture

Con comandi di assegnamento

Sintassi

nome_var.nome_campo=valore

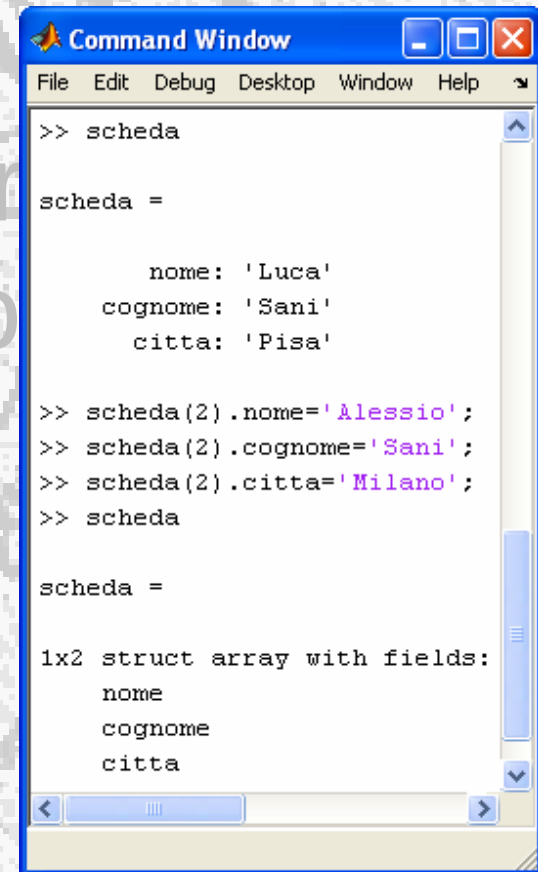


Matlab crea un dato di tipo struct di dimensioni 1x1

Creazione di dati di tipo strutture

Per creare un array di strutture è sufficiente definire il primo elemento e poi quelli successivi specificando la posizione della struttura nell'array mediante indici.

Quando si espande un array di strutture, Matlab assegna in modo automatico una matrice vuota ai campi che non sono specificati. In questo modo tutte le strutture hanno la stessa dimensione.



```
Command Window
File Edit Debug Desktop Window Help
>> scheda

scheda =

    nome: 'Luca'
  cognome: 'Sani'
    città: 'Pisa'

>> scheda(2).nome='Alessio';
>> scheda(2).cognome='Sani';
>> scheda(2).città='Milano';
>> scheda

scheda =

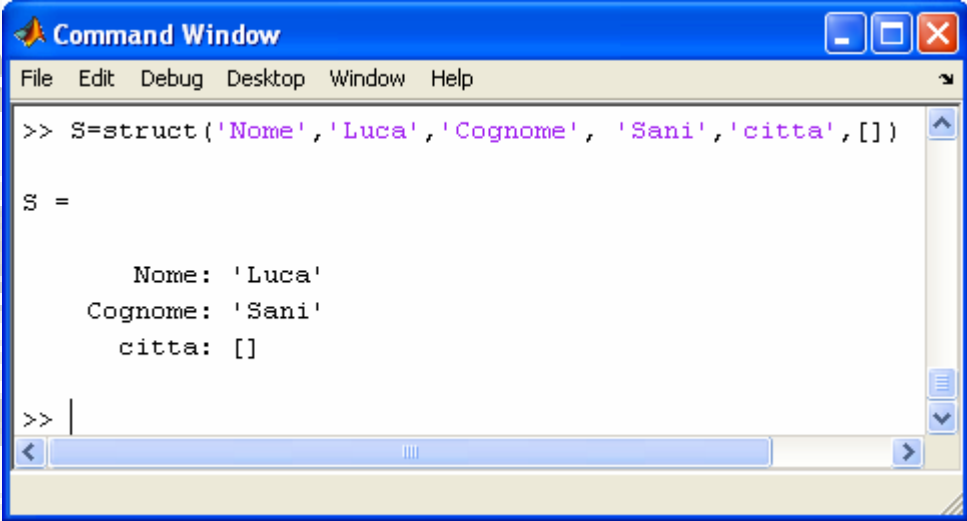
1x2 struct array with fields:
    nome
  cognome
    città
```

Creazione di dati di tipo strutture

Si può creare una struttura con il comando **struct**

Sintassi

nome_var=struct('campo1',valore, 'campo2', valore...)



```
Command Window
File Edit Debug Desktop Window Help
>> S=struct('Nome','Luca','Cognome','Sani','citta',[])

S =

    Nome: 'Luca'
  Cognome: 'Sani'
    citta: []

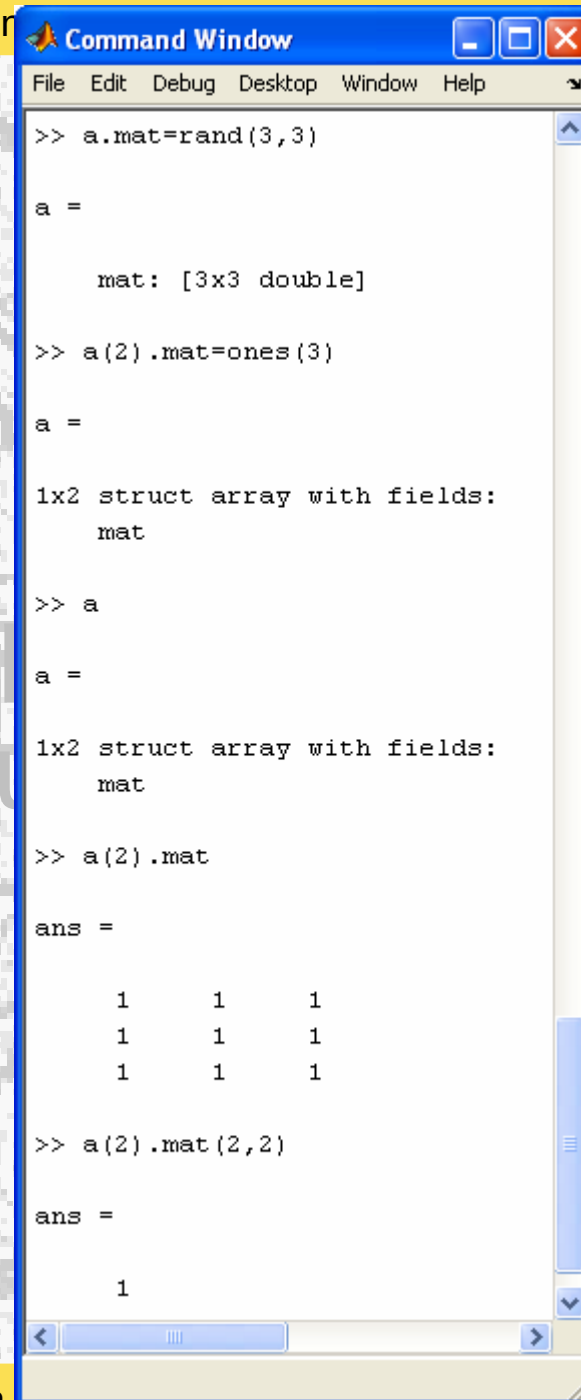
>> |
```

Accesso ai dati di un array di strutture

Utilizzando l'indice dell'array di strutture si può accedere ai valori di ogni campo. Utilizzando anche l'indice del campo si può accedere ai singoli valori di questo ultimo.

Sintassi

var_struc(indice).nomeCampo(indici)



```
>> a.mat=rand(3,3)

a =

    mat: [3x3 double]

>> a(2).mat=ones(3)

a =

1x2 struct array with fields:
    mat

>> a

a =

1x2 struct array with fields:
    mat

>> a(2).mat

ans =

     1     1     1
     1     1     1
     1     1     1

>> a(2).mat(2,2)

ans =

     1
```

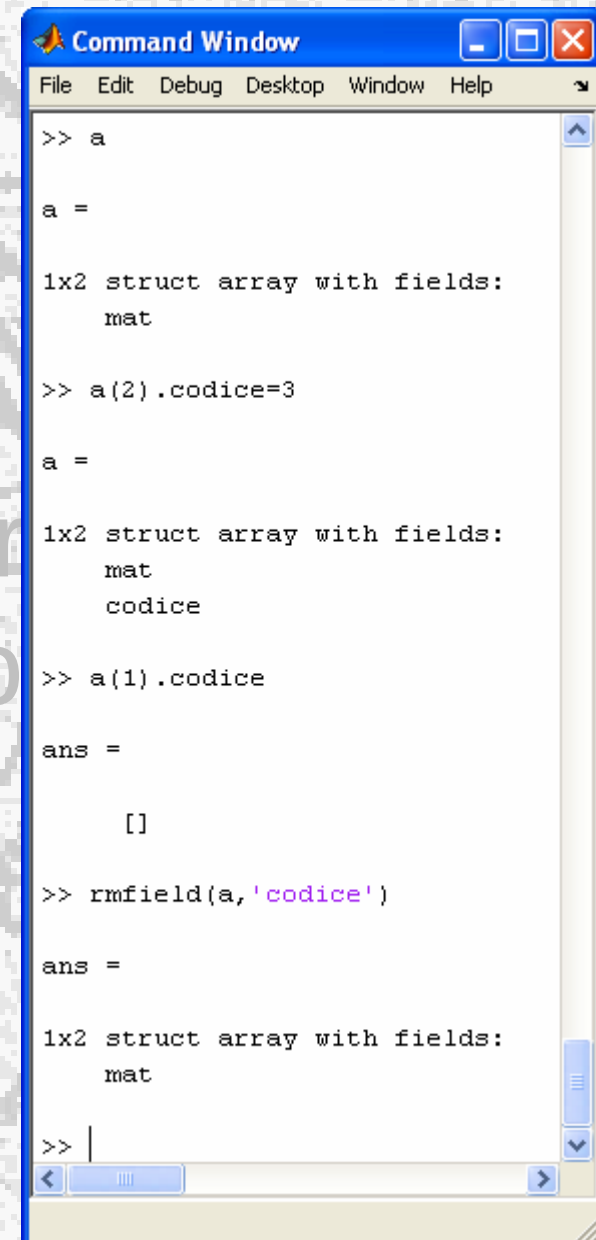
Aggiunta ed eliminazione di un campo ad un array di strutture

Per aggiungere un campo a tutte le strutture di un array è sufficiente aggiungerlo ad una singola struttura. Matlab in modo automatico crea lo stesso campo vuoto per le altre strutture dell'array.

Per eliminare un campo da una struttura che fa parte di un array di strutture si usa la funzione **rmfield**.

Sintassi

rmfield(array, 'campo')



```
Command Window
File Edit Debug Desktop Window Help
>> a
a =
1x2 struct array with fields:
    mat

>> a(2).codice=3
a =
1x2 struct array with fields:
    mat
    codice

>> a(1).codice
ans =
[]

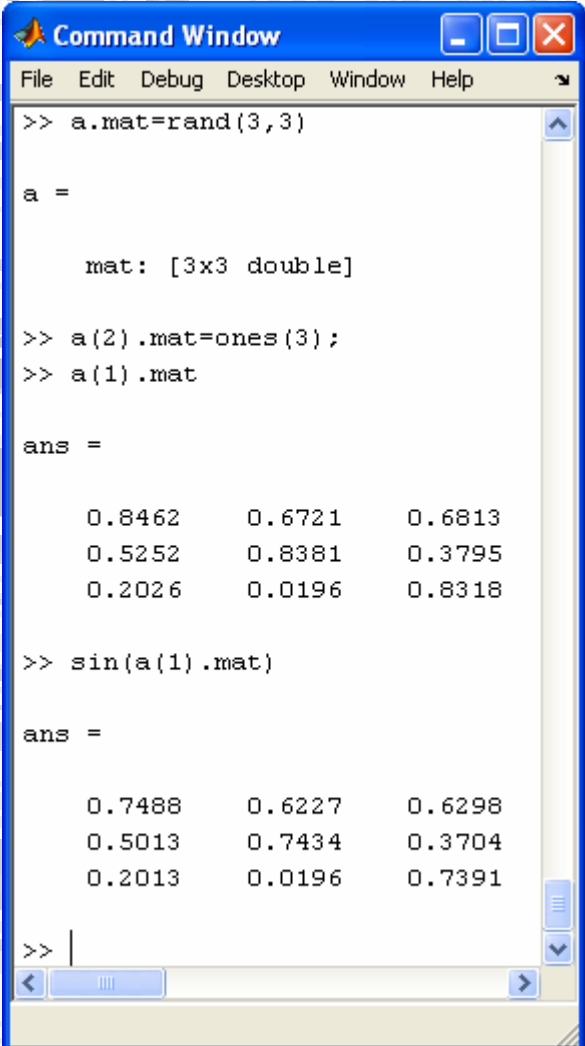
>> rmfield(a, 'codice')
ans =
1x2 struct array with fields:
    mat

>> |
```

Applicazione di funzioni e operatori

E' possibile applicare ai singoli campi di una struttura tutte le funzioni e operatori visti per i dati standard di Matlab.

Si specifica il campo da manipolare con le regole di accesso ad un array di strutture viste in precedenza.



```
Command Window
File Edit Debug Desktop Window Help
>> a.mat=rand(3,3)

a =

    mat: [3x3 double]

>> a(2).mat=ones(3);
>> a(1).mat

ans =

    0.8462    0.6721    0.6813
    0.5252    0.8381    0.3795
    0.2026    0.0196    0.8318

>> sin(a(1).mat)

ans =

    0.7488    0.6227    0.6298
    0.5013    0.7434    0.3704
    0.2013    0.0196    0.7391

>> |
```

IL Symbolic Toolbox

Finora abbiamo utilizzato Matlab soltanto per svolgere operazioni numeriche; in altre parole, i risultati ottenuti sono stati numeri e non espressioni. In questo capitolo utilizzeremo Matlab per svolgere operazioni simboliche i cui risultati si presentano sotto forma di **espressioni**.

Il termine **calcolo simbolico** è utilizzato per indicare che un computer elabora le espressioni matematiche nello stesso modo in cui gli uomini svolgono le operazioni matematiche con carta e penna.

Le funzionalità del calcolo simbolico di Matlab si basano sul pacchetto software Maple V, che è stato sviluppato da Waterloo Maple Software, Inc.

Gli oggetti simbolici

Il Symbolic toolbox definisce un nuovo tipo di dato, detto **oggetto simbolico** o **sym**.

Internamente un oggetto simbolico è un dato di tipo struttura che memorizza una rappresentazione in forma di stringa del simbolo.

Il simbolo può essere una variabile simbolica, un'espressione algebrica, una matrice.

Un oggetto simbolico si definisce con la funzione **sym**

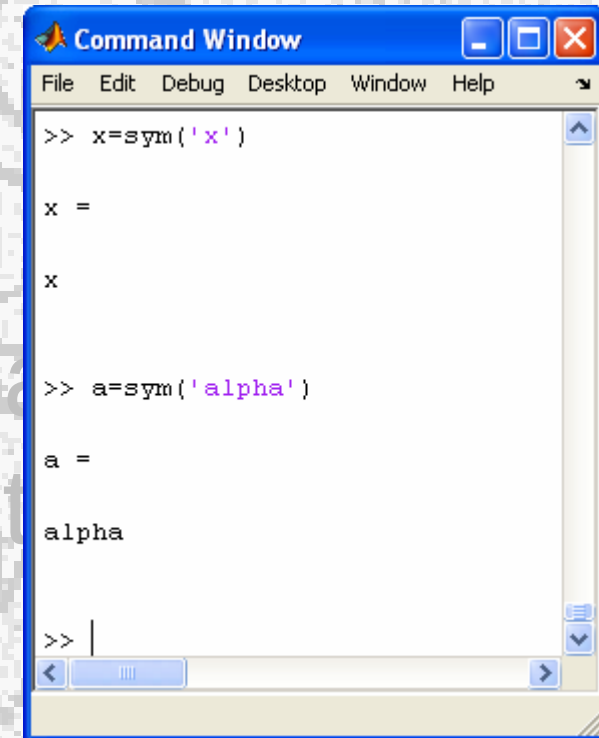
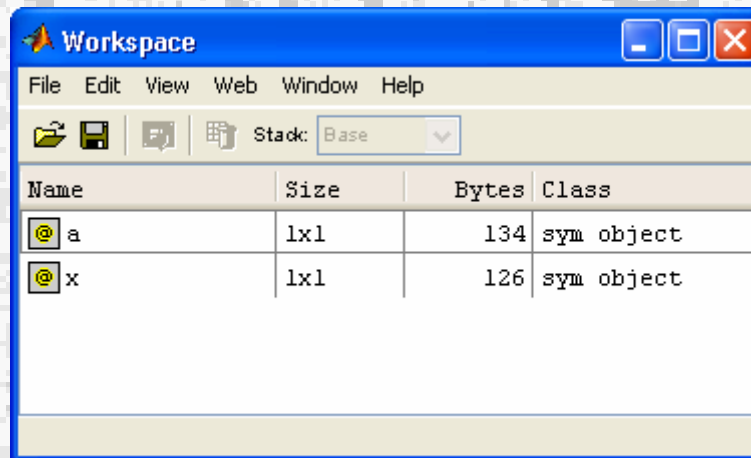
Sintassi

oggetto_simbolico = sym('espressione')

Gli oggetti simbolici

Nell'esempio a lato si definisce un oggetto simbolico x che rappresenta la variabile x , e un secondo oggetto simbolico a che rappresenta la variabile α .

I due oggetti appena definiti appaiono nel workspace.



Differenza tra oggetti simbolici e variabili reali

Matlab opera sugli oggetti simbolici in maniera diversa rispetto alle variabili standard.

La prima istruzione fornisce il valore numerico della radice quadrata di 2. Se questo ultimo è definito come un oggetto simbolico, Matlab, calcola la sua radice quadrata in modo simbolico, con precisione infinita.

Lo stesso accade per frazioni tra numeri o oggetti simbolici.

```
>> sqrt(2)

ans =

    1.4142

>> a=sym(2)

a =

    2

>> sqrt(a)

ans =

    2^(1/2)

>> 5/2

ans =

    2.5000

>> sym(5)/sym(2)

ans =

    5/2
```

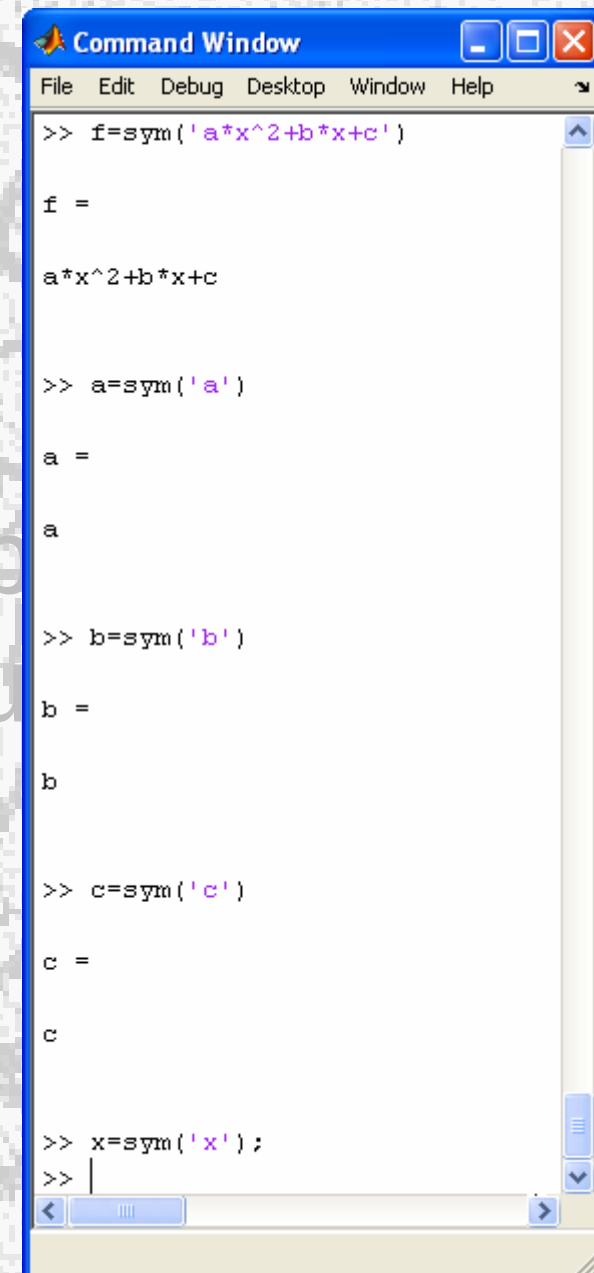
Oggetti simbolici

sym permette di definire un'intera espressione simbolica.

Esempio:

$$f = ax^2 + bx + c$$

Matlab in questo caso non crea però variabili corrispondenti ai termini dell'espressione a, b, c, x. Per eseguire operazioni simboliche (integrazione, derivazione...) occorre creare tali variabili in modo esplicito.



```
Command Window
File Edit Debug Desktop Window Help
>> f=sym('a*x^2+b*x+c')

f =

a*x^2+b*x+c

>> a=sym('a')

a =

a

>> b=sym('b')

b =

b

>> c=sym('c')

c =

c

>> x=sym('x');
>>
```

Oggetti simbolici

E' possibile definire più variabili simboliche contemporaneamente con la funzione **syms**

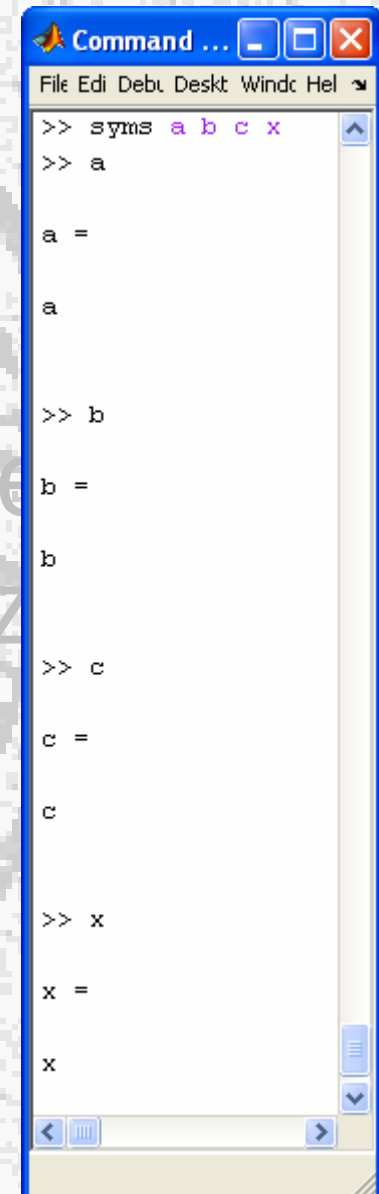
Sintassi

syms var_1 var_2...

è equivalente a

`var_1= sym('var_1')`

`var_2= sym('var_2')`

A screenshot of the MATLAB Command Window. The window title is 'Command ...'. The menu bar includes 'File', 'Edit', 'Debug', 'Desktop', 'Window', and 'Help'. The command history shows: '>> syms a b c x', '>> a', 'a =', 'a', '>> b', 'b =', 'b', '>> c', 'c =', 'c', '>> x', 'x =', 'x'. The output for each variable is displayed on the line immediately following the variable name.

```
>> syms a b c x
>> a
a =
a
>> b
b =
b
>> c
c =
c
>> x
x =
x
```

Oggetti simbolici

I comandi `sym` e `syms` permettono di specificare le proprietà delle variabili simboliche, usando l'opzione **real**

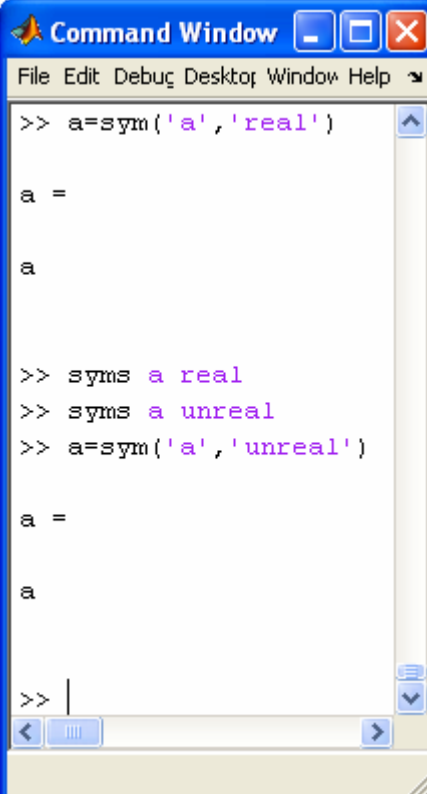
Sintassi

`var = sym('var', real)`

`syms var real`

Se una variabile è definita come **real** significa che ha tutte le proprietà di un numero reale, ad esempio il suo quadrato sarà sicuramente positivo.

Si può annullare tale caratteristica con l'opzione **unreal**.



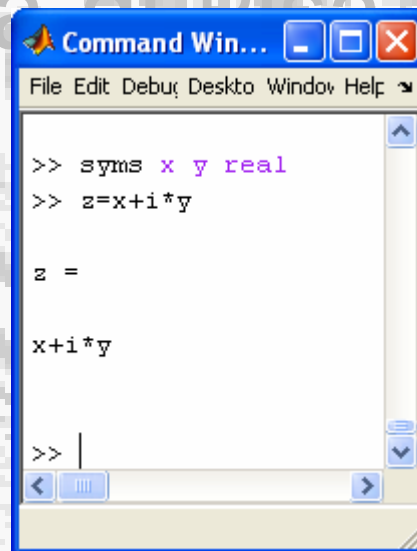
```
Command Window
File Edit Debug Desktop Window Help
>> a=sym('a','real')
a =
a

>> syms a real
>> syms a unreal
>> a=sym('a','unreal')
a =
a

>> |
```

Oggetti simbolici

E' possibile definire una variabile simbolica complessa dopo che sono state definite come variabili simboliche reali la parte reale e la parte immaginaria.

A screenshot of the MATLAB Command Window. The window has a title bar 'Command Win...' and a menu bar with 'File', 'Edit', 'Debug', 'Desktop', 'Window', and 'Help'. The command prompt shows two lines of input: '>> syms x y real' and '>> z=x+i*y'. The output shows 'z =' followed by 'x+i*y' on the next line. The cursor is at the end of the second input line.

```
>> syms x y real
>> z=x+i*y

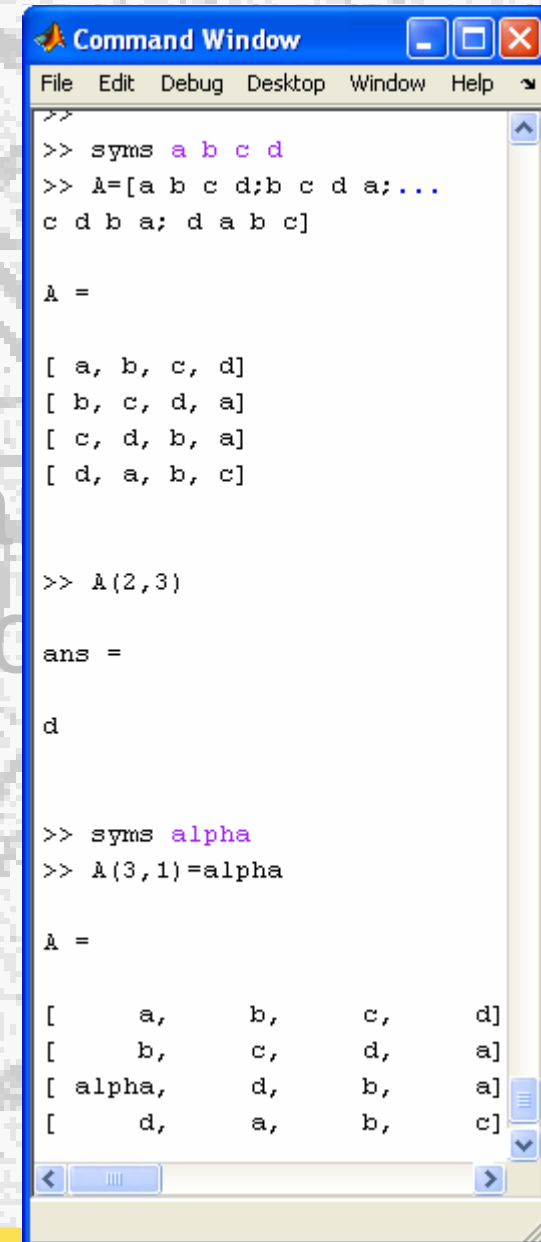
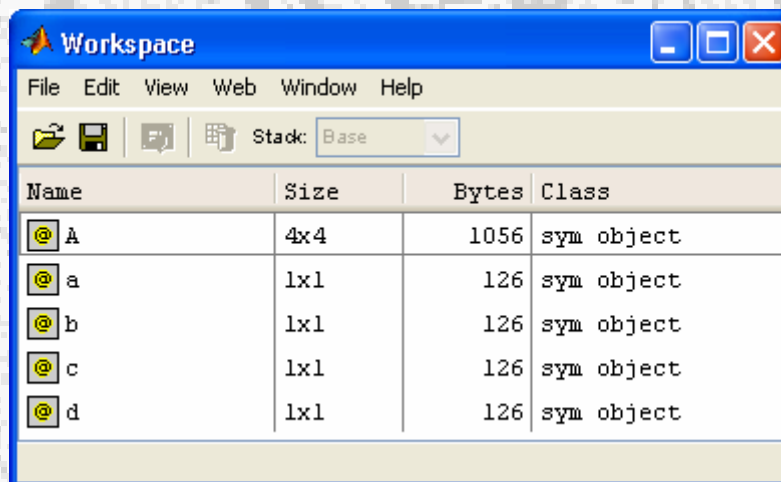
z =
x+i*y

>> |
```

Oggetti simbolici matriciali

È possibile definire una matrice simbolica, una volta definite le singole variabili simboliche.

Una matrice simbolica è trattata in modo analogo a quanto visto per quelle numeriche. Sono valide quindi tutte le considerazioni riguardanti l'indicizzazione.



Variabile indipendente

Prima di eseguire un'operazione su un'espressione simbolica con più variabili occorre stabilire la variabile indipendente. Per default è la 'x', se presente, altrimenti è quella più vicina alla 'x'. La funzione **findsym** restituisce le variabili simboliche presenti in un'espressione

Sintassi

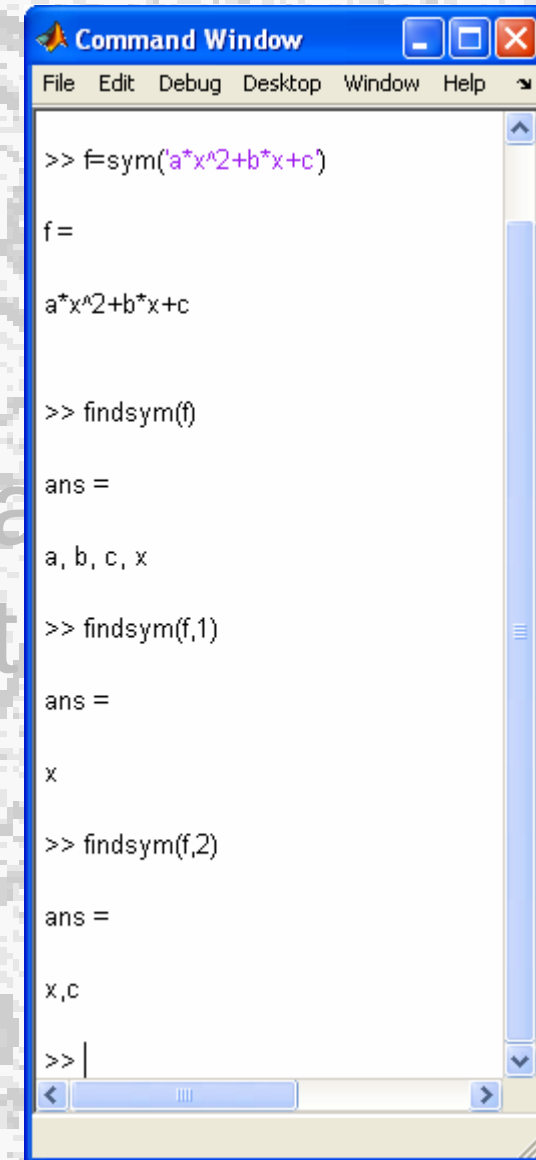
var_str = findsym(E) var_str contiene tutte le variabili simboliche in ordine alfabetico

var_str = findsym(E,n) var_str contiene le prime n variabili simboliche in ordine di vicinanza rispetto alla 'x'

E' sempre possibile nelle operazioni specificare una diversa variabile indipendente.

Variabile indipendente

Università di Pisa - Dipartimento
Sistemi Elettrici e Automatici



```
Command Window
File Edit Debug Desktop Window Help

>> f=sym('a*x^2+b*x+c')

f =
a*x^2+b*x+c

>> findsym(f)

ans =
a, b, c, x

>> findsym(f,1)

ans =
x

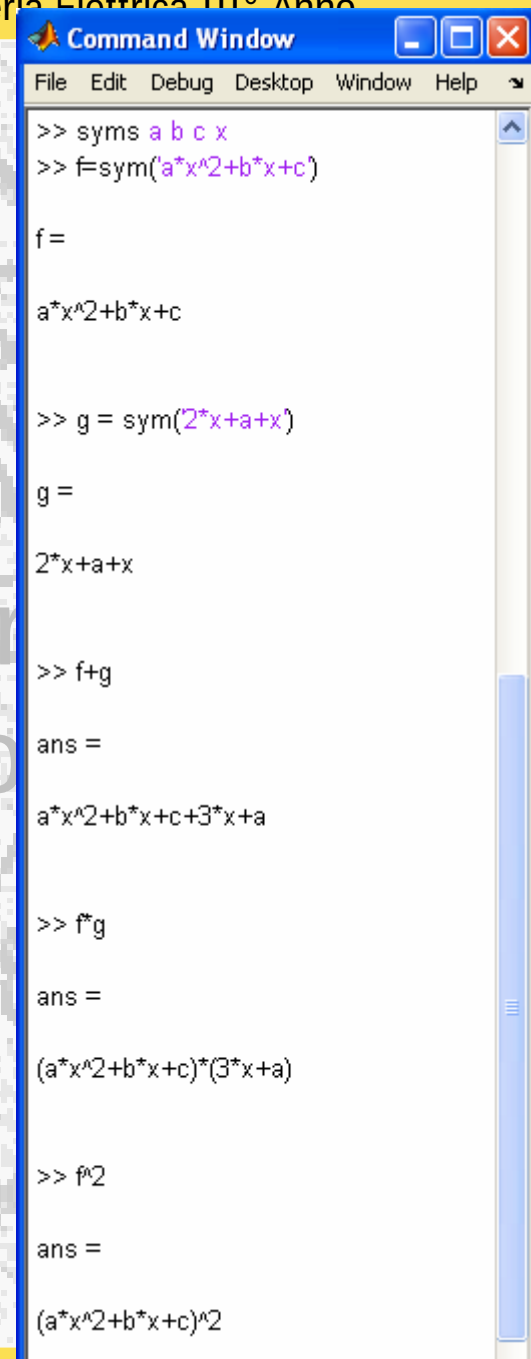
>> findsym(f,2)

ans =
x,c

>> |
```

Operazioni aritmetiche

E' possibile utilizzare gli operatori $+$, $-$, $*$, $/$ e $^$ per ottenere nuove espressioni simboliche.

A screenshot of the MATLAB Command Window interface. The window has a blue title bar with the text 'Command Window' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Debug', 'Desktop', 'Window', and 'Help'. The main area of the window contains a series of MATLAB commands and their outputs. The commands are: 'syms a b c x', 'f=sym('a*x^2+b*x+c')', 'g = sym('2*x+a+x')', 'f+g', 'f*g', and 'f^2'. The outputs are: 'f = a*x^2+b*x+c', 'g = 2*x+a+x', 'ans = a*x^2+b*x+c+3*x+a', 'ans = (a*x^2+b*x+c)*(3*x+a)', and 'ans = (a*x^2+b*x+c)^2'.

```
>> syms a b c x
>> f=sym('a*x^2+b*x+c')

f =

a*x^2+b*x+c

>> g = sym('2*x+a+x')

g =

2*x+a+x

>> f+g

ans =

a*x^2+b*x+c+3*x+a

>> f*g

ans =

(a*x^2+b*x+c)*(3*x+a)

>> f^2

ans =

(a*x^2+b*x+c)^2
```

Manipolazione delle espressioni

Le seguenti funzioni permettono di manipolare le espressioni simboliche definite.

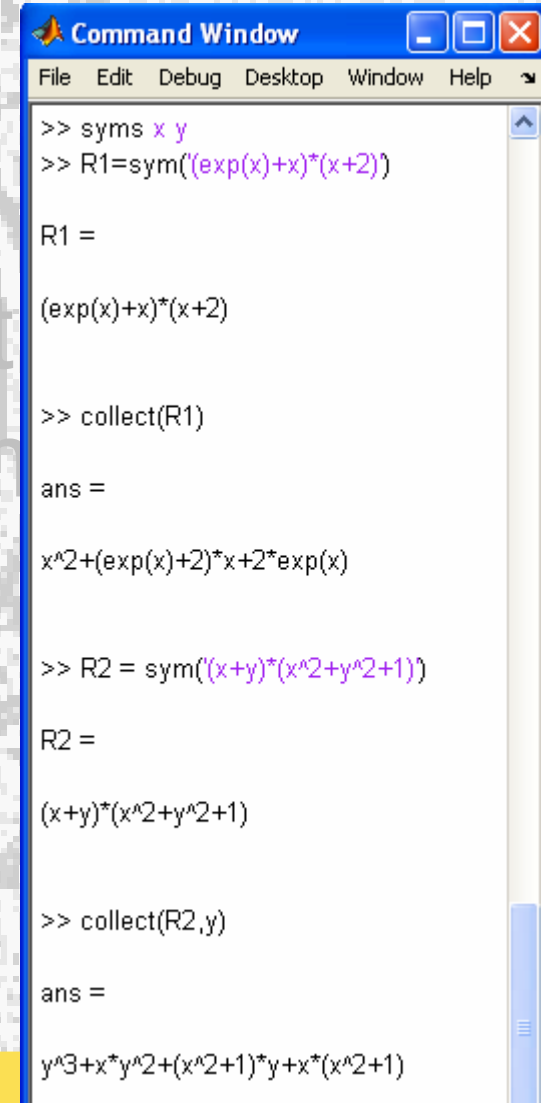
- **collect** raccoglie i coefficienti delle potenze di pari grado rispetto alla variabile indipendente.

Sintassi

collect(E) la variabile ind. è 'x'

collect(E,v) la variabile ind. è 'v'

La E è l'espressione simbolica da manipolare.



```

Command Window
File Edit Debug Desktop Window Help
>> syms x y
>> R1=sym('(exp(x)+x)*(x+2)')

R1 =

(exp(x)+x)*(x+2)

>> collect(R1)

ans =

x^2+(exp(x)+2)*x+2*exp(x)

>> R2 = sym('(x+y)*(x^2+y^2+1)')

R2 =

(x+y)*(x^2+y^2+1)

>> collect(R2,y)

ans =

y^3+x*y^2+(x^2+1)*y+x*(x^2+1)

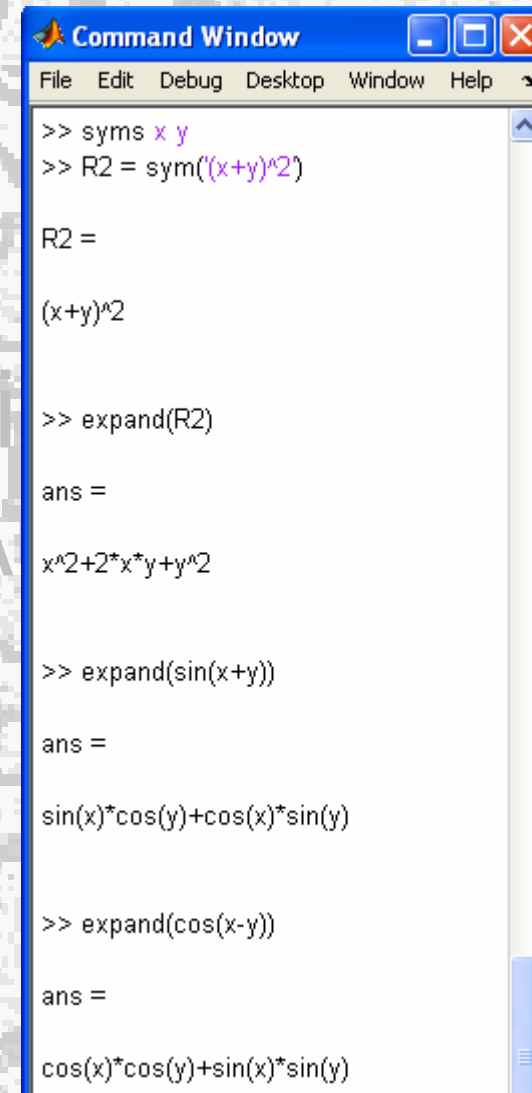
```

Manipolazione delle espressioni

- **expand** sviluppa le potenze contenute nell'espressione.

Sintassi

expand(E)



```
Command Window
File Edit Debug Desktop Window Help
>> syms x y
>> R2 = sym('(x+y)^2')

R2 =

(x+y)^2

>> expand(R2)

ans =

x^2+2*x*y+y^2

>> expand(sin(x+y))

ans =

sin(x)*cos(y)+cos(x)*sin(y)

>> expand(cos(x-y))

ans =

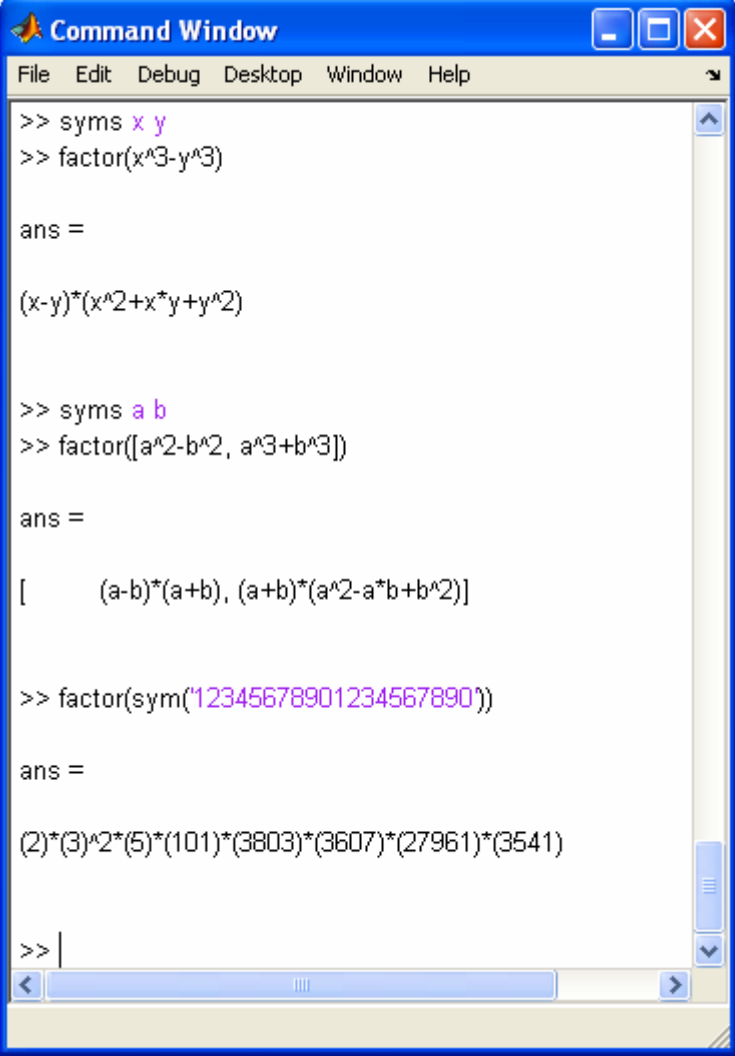
cos(x)*cos(y)+sin(x)*sin(y)
```

Manipolazione delle espressioni

- **factor** trasforma l'espressione in un prodotto di fattori.

Sintassi

factor(E)

A screenshot of the MATLAB Command Window. The window has a blue title bar with the text 'Command Window' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Debug', 'Desktop', 'Window', and 'Help'. The main area shows three commands and their outputs. The first command is 'syms x y' followed by 'factor(x^3-y^3)', resulting in '(x-y)*(x^2+x*y+y^2)'. The second command is 'syms a b' followed by 'factor([a^2-b^2, a^3+b^3])', resulting in a list containing '(a-b)*(a+b)' and '(a+b)*(a^2-a*b+b^2)'. The third command is 'factor(sym('12345678901234567890'))', resulting in a large product of prime factors: '(2)*(3)^2*(5)*(101)*(3803)*(3607)*(27961)*(3541)'.

```
Command Window
File Edit Debug Desktop Window Help
>> syms x y
>> factor(x^3-y^3)

ans =

(x-y)*(x^2+x*y+y^2)

>> syms a b
>> factor([a^2-b^2, a^3+b^3])

ans =

[ (a-b)*(a+b), (a+b)*(a^2-a*b+b^2)]

>> factor(sym('12345678901234567890'))

ans =

(2)*(3)^2*(5)*(101)*(3803)*(3607)*(27961)*(3541)

>> |
```

Manipolazione delle espressioni

- **simple** ricerca la forma più semplice dell'espressione.

Sintassi

simple

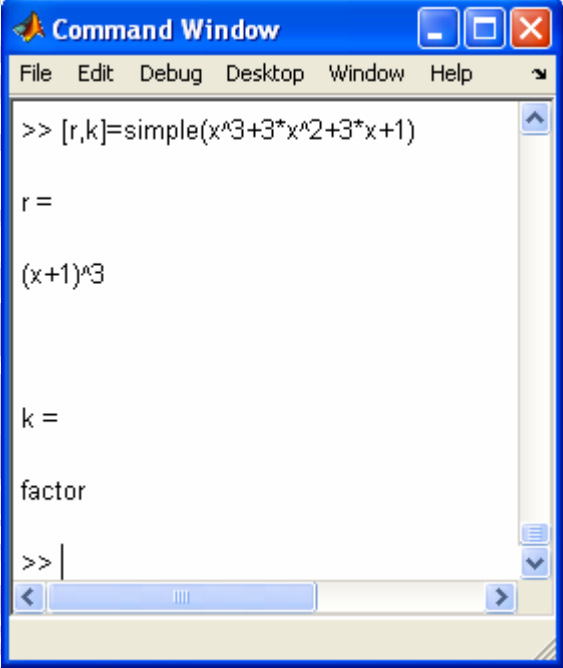
agisce sull'ultima espressione

simple(E)

ricerca l'espressione più semplice e visualizza tutti i passaggi intermedi

[r,k]=simple(E)

l'espressione semplificata è in r e il metodo utilizzato in k. Non sono visualizzati i passaggi intermedi.



```
Command Window
File Edit Debug Desktop Window Help
>> [r,k]=simple(x^3+3*x^2+3*x+1)

r =

(x+1)^3

k =

factor

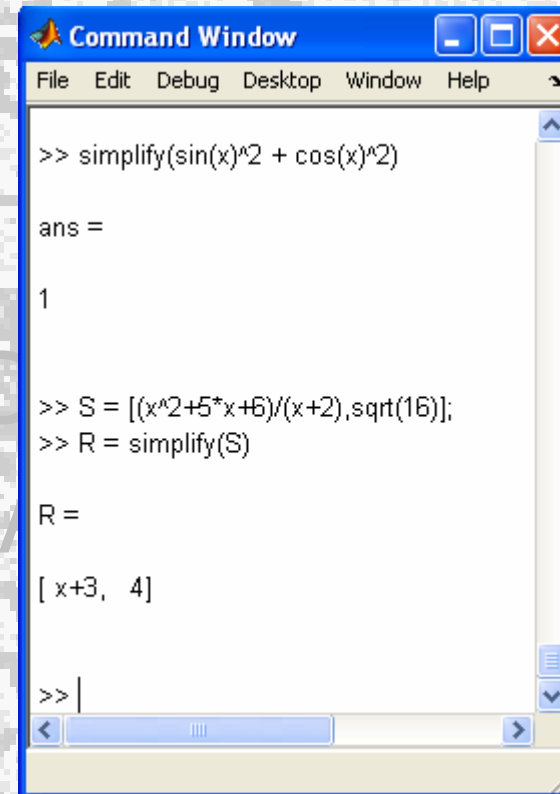
>> |
```

Manipolazione delle espressioni

- **simplify** semplifica l'espressione applicando le regole della semplificazione del software Maple.

Sintassi

simplify(E)



```
Command Window
File Edit Debug Desktop Window Help

>> simplify(sin(x)^2 + cos(x)^2)

ans =

1

>> S := [(x^2+5*x+6)/(x+2),sqrt(16)];
>> R := simplify(S)

R =

[ x+3, 4]

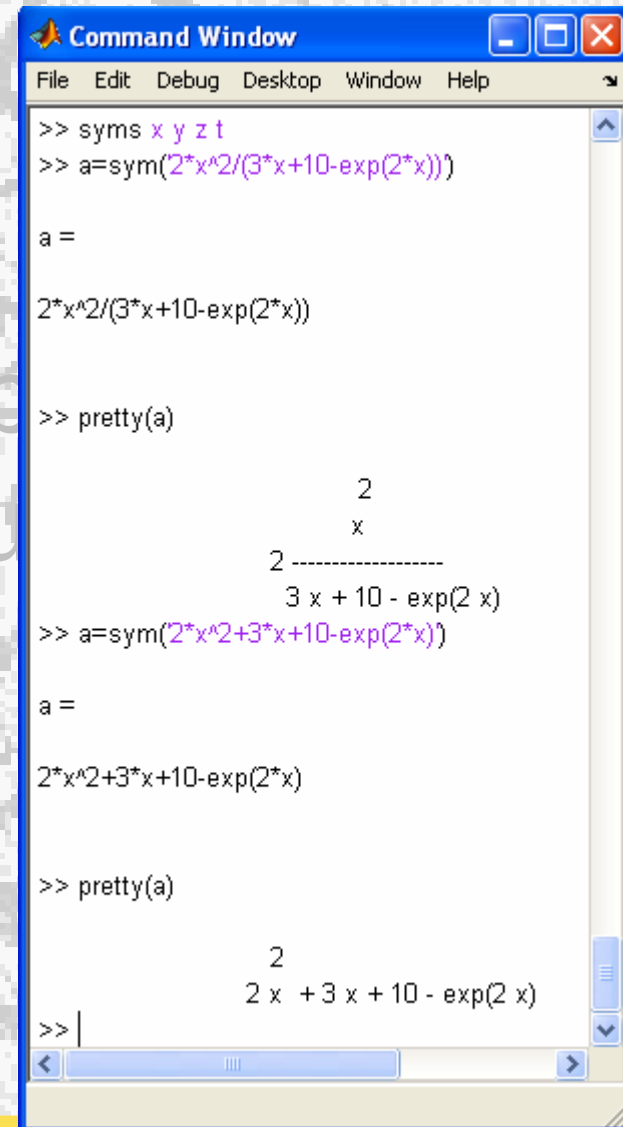
>> |
```

Manipolazione delle espressioni

- **pretty** visualizza l'espressione in una forma che somiglia alla tipica rappresentazione matematica.

Sintassi

pretty(E)



```
Command Window
File Edit Debug Desktop Window Help
>> syms x y z t
>> a=sym('2*x^2/(3*x+10-exp(2*x))')

a =

2*x^2/(3*x+10-exp(2*x))

>> pretty(a)

      2
      x
      2 -----
      3 x + 10 - exp(2 x)

>> a=sym('2*x^2+3*x+10-exp(2*x)')

a =

2*x^2+3*x+10-exp(2*x)

>> pretty(a)

      2
      2 x  + 3 x + 10 - exp(2 x)

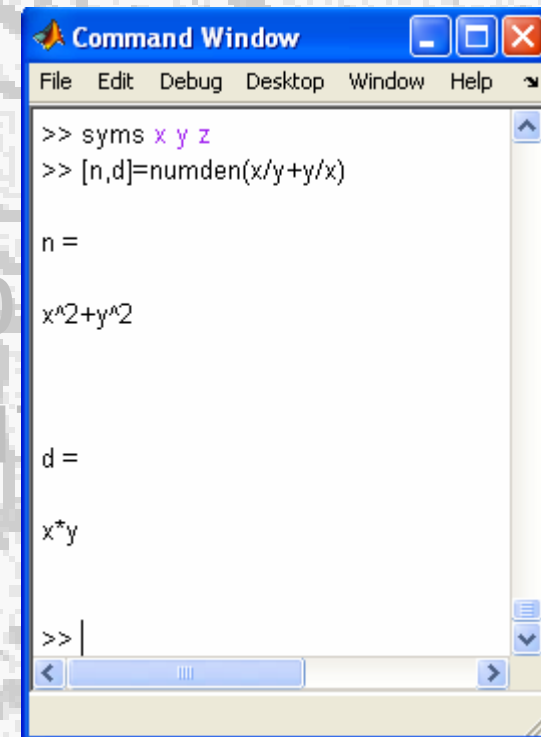
>> |
```

Manipolazione delle espressioni

- **numden** calcola due espressioni simboliche che rappresentano il numeratore e il denominatore della rappresentazione razionale della espressione.

Sintassi

[num,den]=numden(E)



```
Command Window
File Edit Debug Desktop Window Help
>> syms x y z
>> [n,d]=numden(x/y+y/x)

n =
x^2+y^2

d =
x*y

>> |
```

Manipolazione delle espressioni

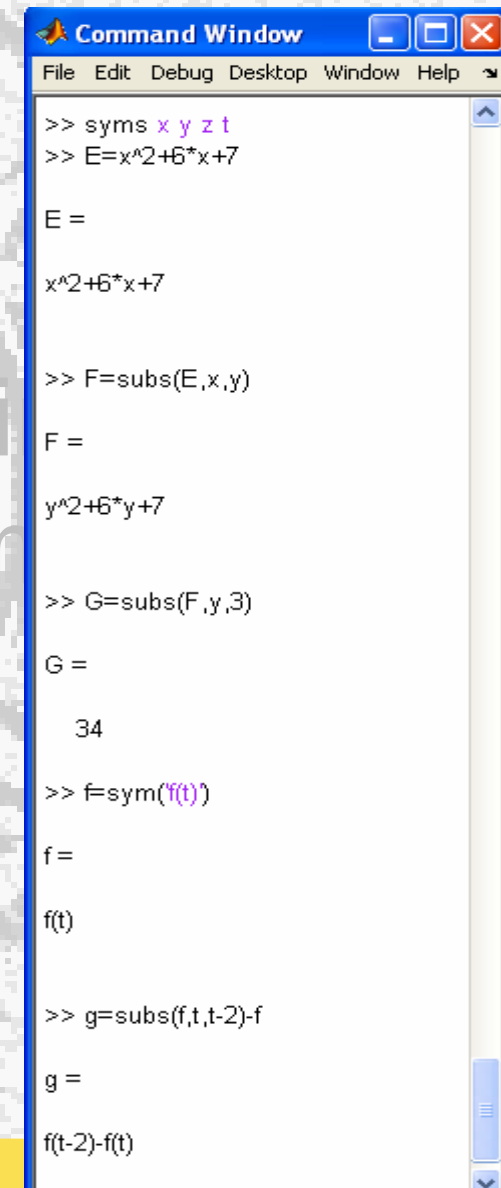
- **subs** simbolica sostituzione in una simbolica espressione.

Sintassi

subs(E, old, new)

old può essere un'espressione o una variabile simbolica;

new può essere un'espressione simbolica, una matrice simbolica, una variabile simbolica o un valore numerico.



```

Command Window
File Edit Debug Desktop Window Help
>> syms x y z t
>> E=x^2+6*x+7

E =

x^2+6*x+7

>> F=subs(E,x,y)

F =

y^2+6*y+7

>> G=subs(F,y,3)

G =

34

>> f=sym('f(t)')

f =

f(t)

>> g=subs(f,t,t-2)-f

g =

f(t-2)-f(t)

```

Manipolazione delle espressioni

- **poly2sym** trasforma un vettore di coefficienti in un polinomio simbolico.

Sintassi

$$E = \text{poly2sym}(p)$$

$E = \text{poly2sym}(p, 'v')$ polinomio in funzione della variabile v .

- **sym2poly** trasforma l'espressione simbolica in un vettore che contiene i coefficienti del polinomio.

Sintassi

$$p = \text{sym2poly}(E)$$

```
>> poly2sym([2 6 4])
ans =
2*x^2+6*x+4
>> pretty(ans)
      2
    2 x + 6 x + 4
>> poly2sym([2 6 4], 'y')
ans =
2*y^2+6*y+4
>> pretty(ans)
      2
    2 y + 6 y + 4
>> sym x
ans =
x
>> p=sym2poly(x^3+x+1)
p =
1 0 1 1
```

Manipolazione delle espressioni

- **char** converte un'espressione simbolica in una stringa.

Sintassi

s = char(E)

- **double** converte l'espressione simbolica in formato numerico.

Sintassi

p = double(E)

La E non deve contenere variabili simboliche.

```
>> syms x y z t
>> a=sym('2*x^2+3*x+10-exp(2*x)')

a =

2*x^2+3*x+10-exp(2*x)

>> s=char(a)

s =

2*x^2+3*x+10-exp(2*x)

>> sq=sym('sqrt(2)')

sq =

sqrt(2)

>> y=6*sq

y =

6*2^(1/2)

>> double(y)

ans =

8.4853
```

Derivazione

La funzione **diff** consente di calcolare le derivate di espressioni simboliche.

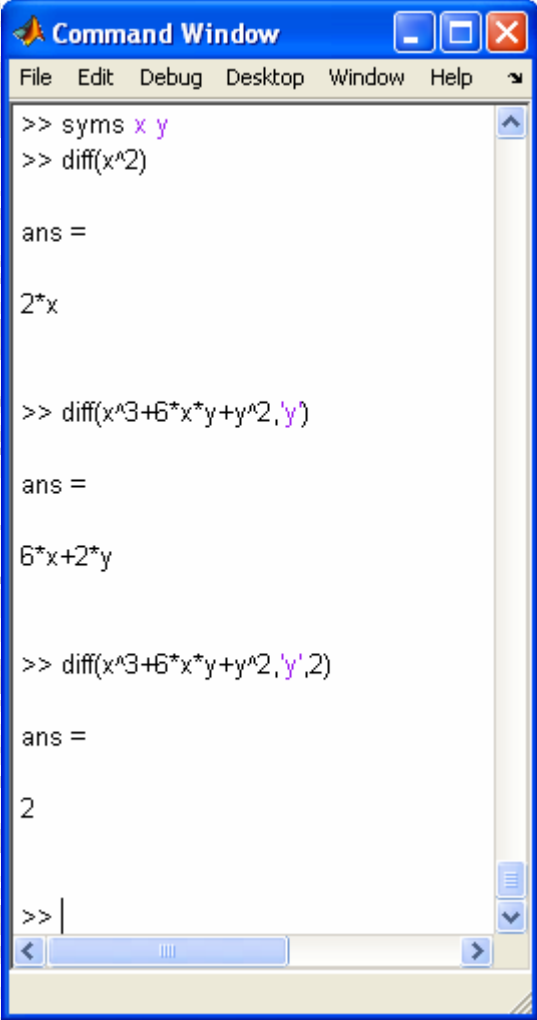
Sintassi

diff(E) derivata di (E) rispetto alla variabile indipendente che può essere identificata con findsym.

diff(E, 'v') derivata di E rispetto a v.

diff(E, n) derivata n-esima di E rispetto alla var. indep.

diff(E, 'v', n) derivata n-esima di E rispetto a v.



```
>> syms x y
>> diff(x^2)

ans =

2*x

>> diff(x^3+6*x*y+y^2,'y')

ans =

6*x+2*y

>> diff(x^3+6*x*y+y^2,'y',2)

ans =

2

>> |
```

Integrazione

La funzione **int** consente di calcolare l'integrale di espressioni simboliche.

Sintassi

int(E) integrale di E rispetto alla variabile indipendente che può essere identificata con findsym.

int(E, 'v') integrale di E rispetto a v.

int(E, a,b) integrale definito rispetto alla var. indep. nell'intervallo [a, b], con a e b grandezze numeriche.

int(E, 'v',a,b) integrale definito di E rispetto a v nell'intervallo [a, b].

```
>> syms x y z t
>> int(-2*x/(1+x^2)^2)

ans =

1/(x^2+1)

>> int(x/(1+z^2),z)

ans =

x*atan(z)

>> int(x*log(1+x),0,1)

ans =

1/4

>> int(2*x, sin(t), 1)

ans =

1-sin(t)^2

>> int(y*log(1+x),y',0,1)

ans =

1/2*log(x+1)
```

Limite

La funzione **limit** consente di calcolare il limite di espressioni simboliche.

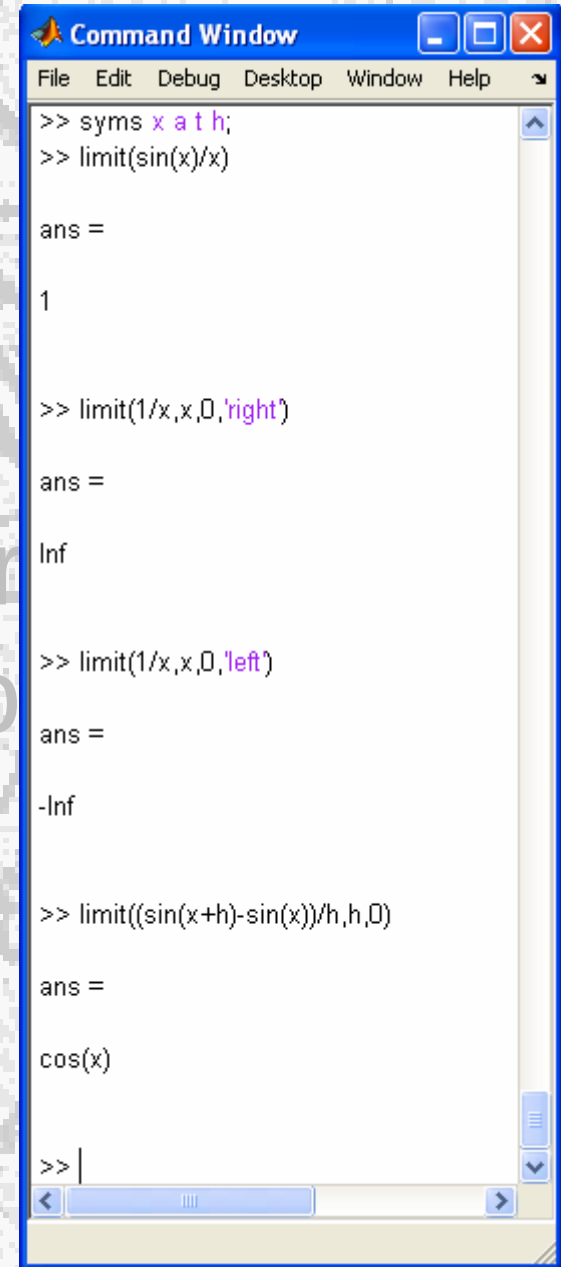
Sintassi

limit(E) limite di E quando la variabile indipendente di default tende a zero.

limit(E, a) limite di E quando la variabile indipendente di default tende ad a.

limit(E, v, a) limite di E quando la variabile v tende ad a.

limit(E, v, a, 'd') limite di E quando la variabile v tende ad a dalla direzione specificata da d: da destra (d=right) o da sinistra (d=left)



```

Command Window
File Edit Debug Desktop Window Help
>> syms x a t h;
>> limit(sin(x)/x)

ans =

1

>> limit(1/x,x,0,'right')

ans =

Inf

>> limit(1/x,x,0,'left')

ans =

-Inf

>> limit((sin(x+h)-sin(x))/h,h,0)

ans =

cos(x)

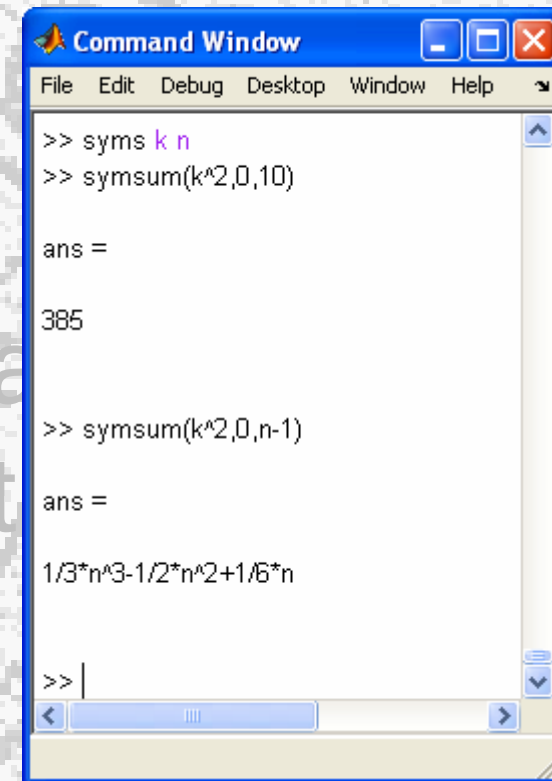
>>
  
```

Serie

La funzione **symsum** consente di calcolare la sommatoria simbolica dell'espressione.

Sintassi

symsum(E, a, b) restituisce la sommatoria dell'espressione E , con la variabile di default che varia tra a e b .



```
Command Window
File Edit Debug Desktop Window Help
>> syms k n
>> symsum(k^2,0,10)

ans =

385

>> symsum(k^2,0,n-1)

ans =

1/3*n^3-1/2*n^2+1/6*n

>> |
```

Taylor

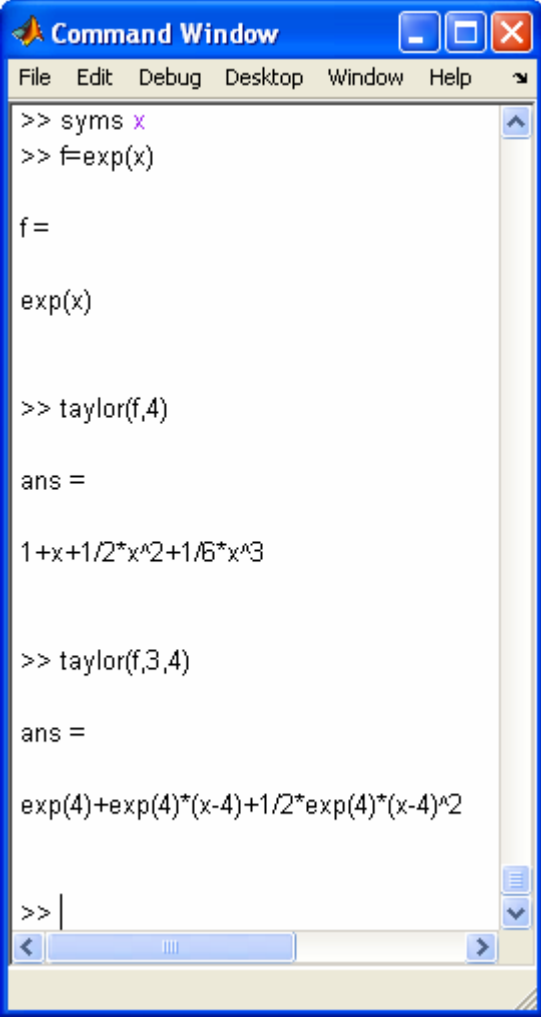
La funzione **taylor** calcola i termini della serie di Taylor dell'espressione simbolica in un punto

Sintassi

taylor(E, n) fornisce i primi $n-1$ termini della serie di Taylor della funzione E per $x=0$.

taylor(E, n, a) fornisce i primi $n-1$ termini della serie di Taylor della funzione E per $x=a$.

taylor(E, n, v, a) fornisce i primi $n-1$ termini della serie di Taylor della funzione E , rispetto alla variabile v , per $v=a$.



```

Command Window
File Edit Debug Desktop Window Help
>> syms x
>> f=exp(x)

f =

exp(x)

>> taylor(f,4)

ans =

1+x+1/2*x^2+1/6*x^3

>> taylor(f,3,4)

ans =

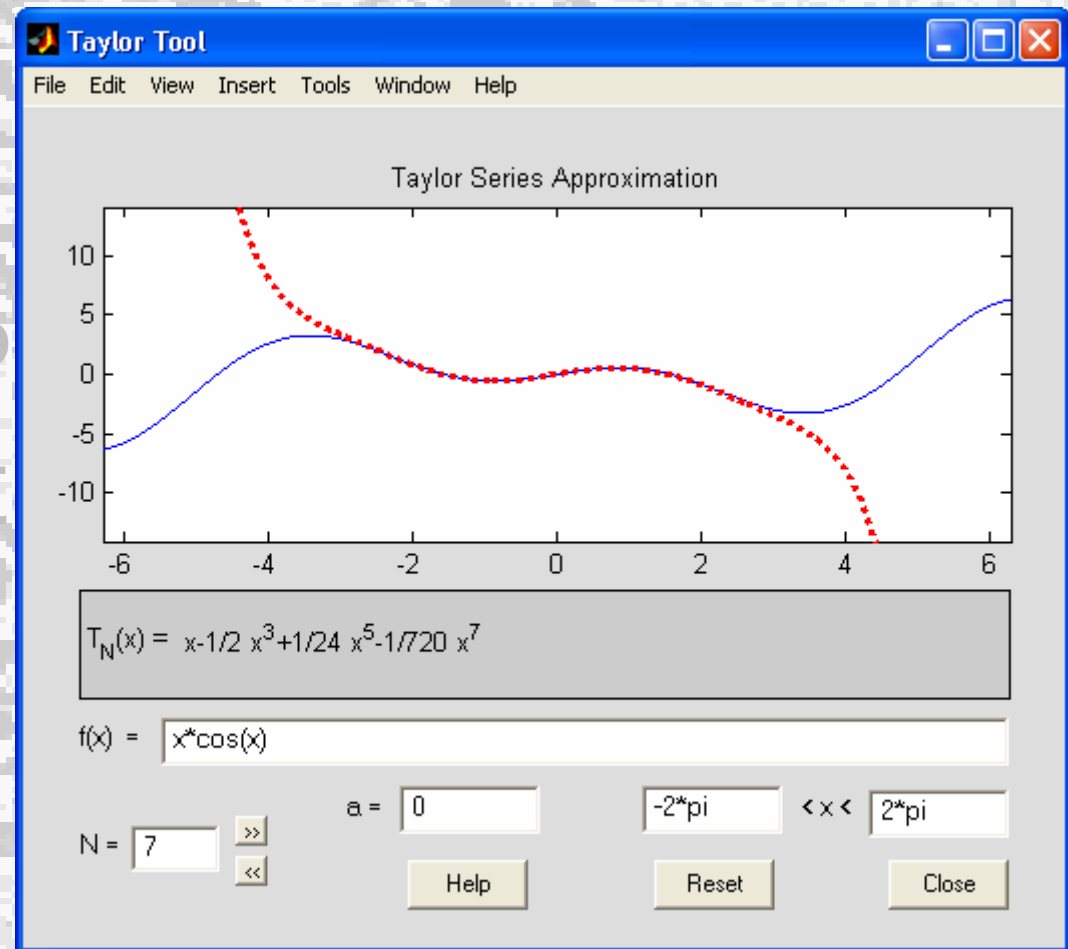
exp(4)+exp(4)*(x-4)+1/2*exp(4)*(x-4)^2

>>
  
```

Taylor

Il toolbox Symbolic mette a disposizione un'interfaccia grafica per il calcolo della serie di Taylor.

Si avvia con il comando **taylortool**



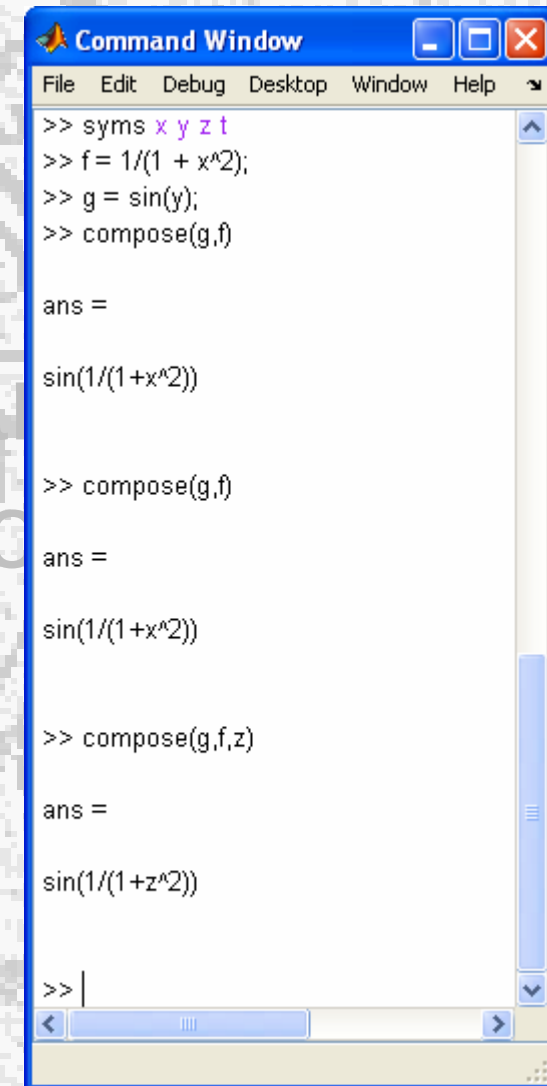
Funzioni composte

La funzione **compose** determina la funzione composta di due funzioni.

Sintassi

compose(f, g) calcola $f(g(y))$ dove $f = f(x)$ e $g = g(y)$. In questo caso x è la variabile indipendente di f e y è la variabile indipendente di g .

compose(f, g, z) calcola $f(g(z))$ dove $f = f(x)$ e $g = g(y)$.



```
Command Window
File Edit Debug Desktop Window Help
>> syms x y z t
>> f = 1/(1 + x^2);
>> g = sin(y);
>> compose(g,f)

ans =

sin(1/(1+x^2))

>> compose(g,f)

ans =

sin(1/(1+x^2))

>> compose(g,f,z)

ans =

sin(1/(1+z^2))

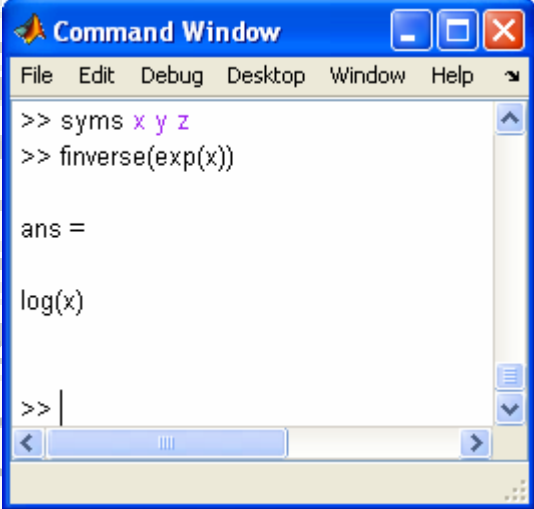
>> |
```

Funzione inversa

La funzione **finverse** determina la funzione inversa.

Sintassi

finverse(E)

A screenshot of the MATLAB Command Window. The window has a blue title bar with the text "Command Window" and standard window control buttons. Below the title bar is a menu bar with "File", "Edit", "Debug", "Desktop", "Window", and "Help". The main area shows the following commands and output:

```
>> syms x y z
>> finverse(exp(x))

ans =

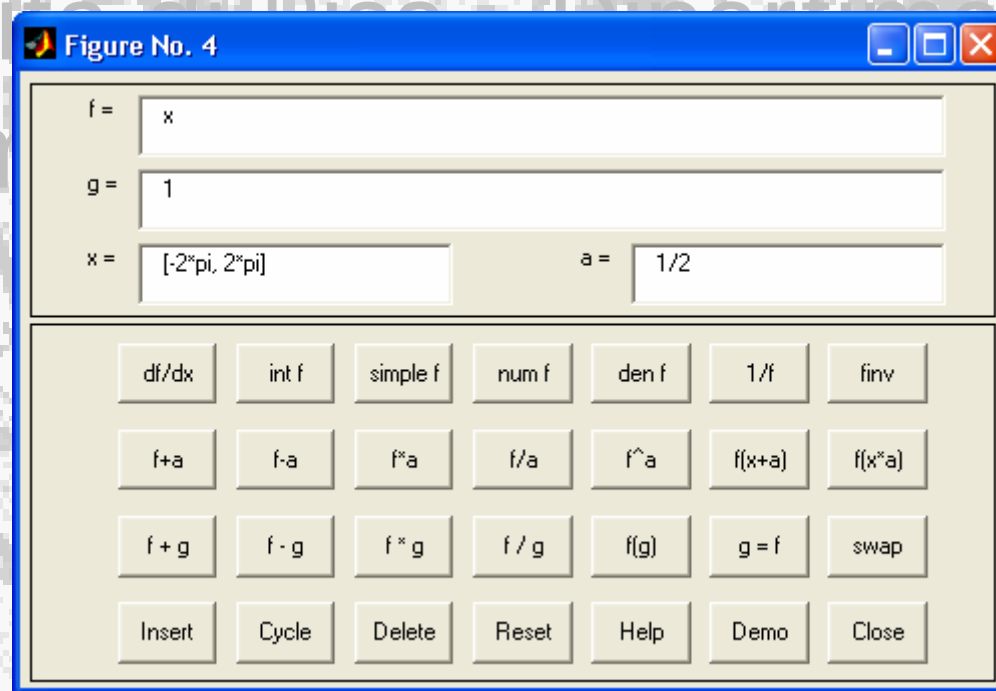
log(x)

>> |
```

Manipolazione di funzioni

Il toolbox Symbolic mette a disposizione un'interfaccia grafica per la manipolazione di funzioni in una variabile.

Si avvia con il comando **funtool**

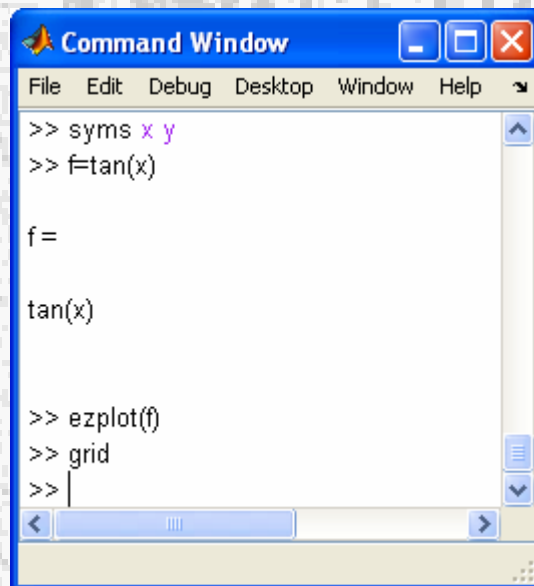


La funzione **ezplot** traccia il grafico della funzione simbolica in una sola variabile.

Sintassi

ezplot(f) traccia il grafico di $f(x)$ per $-2\pi < x < 2\pi$

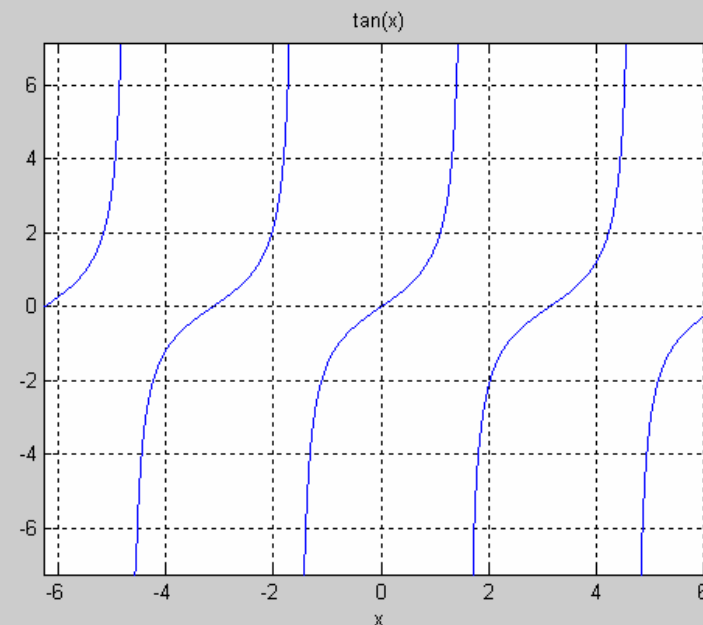
ezplot(f, [xmin, xmax]) traccia il grafico di $f(x)$ per $xmin < x < xmax$



```
Command Window
File Edit Debug Desktop Window Help
>> syms x y
>> f=tan(x)

f=
tan(x)

>> ezplot(f)
>> grid
>> |
```



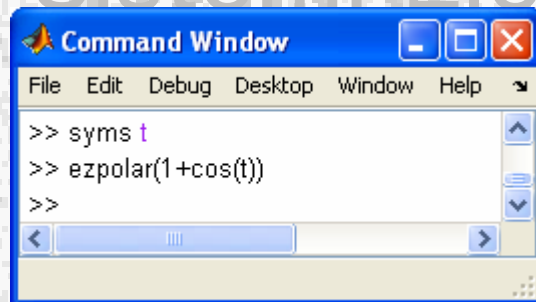
Grafici di funzioni

La funzione **ezpolar** traccia il grafico della funzione simbolica in una sola variabile in coordinate polari.

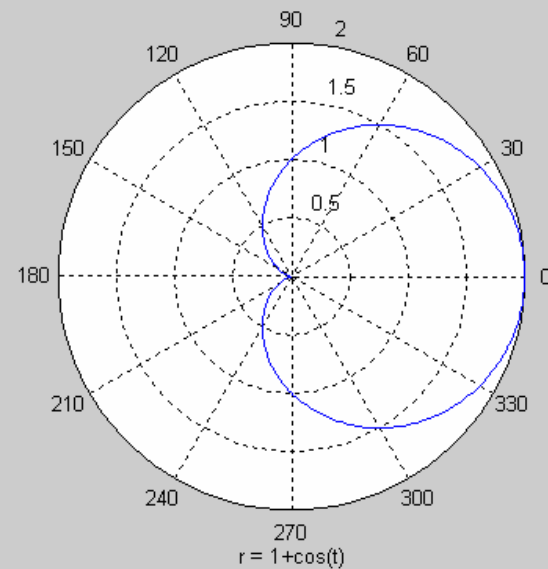
Sintassi

ezpolar(f) traccia il grafico di $f(\theta)$ per $-2\pi < \theta < 2\pi$

ezpolar(f, [a,b]) traccia il grafico di $f(\theta)$ per $a < \theta < b$



```
Command Window
File Edit Debug Desktop Window Help
>> syms t
>> ezpolar(1+cos(t))
>>
```



Grafici di funzioni

La funzione **ezplot3** traccia il grafico 3D di curve parametriche.

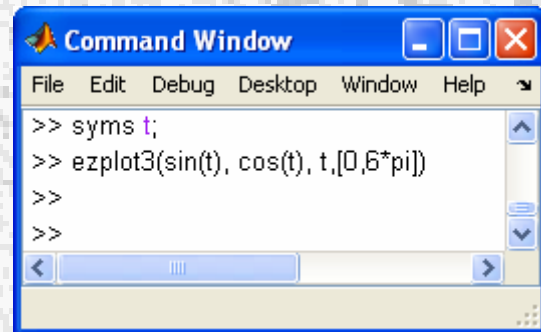
Sintassi

ezplot3(x,y,z) traccia il grafico della funzione descritta da $x=x(t)$, $y=y(t)$ e $z=z(t)$, per $0 < t < 2\pi$

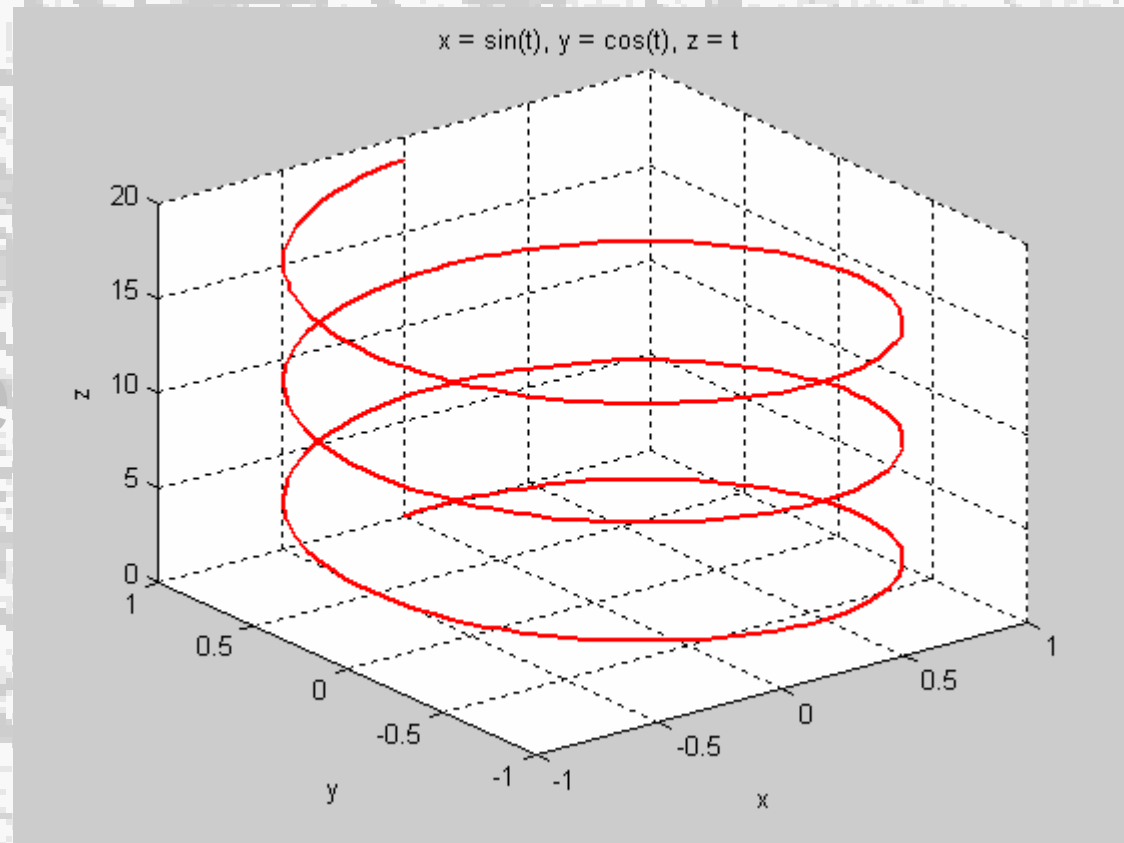
ezplot3(x,y,z, [tmin, tmax]) traccia il grafico della funzione descritta da $x=x(t)$, $y=y(t)$ e $z=z(t)$, per $tmin < t < tmax$

ezplot3(x,y,z,..., 'animate') traccia il grafico animato della funzione descritta da $x=x(t)$, $y=y(t)$ e $z=z(t)$.

Grafici di funzioni



```
>> syms t;  
>> ezplot3(sin(t), cos(t), t,[0,6*pi])  
>>  
>>
```



Grafici di funzioni

La funzione **ezmesh** traccia il grafico 3D di una funzione in due variabili.

La funzione **ezsurf** traccia il grafico 3D di una funzione in due variabili. Il grafico è una superficie 3D colorata.

La funzione **ezcontour** traccia le linee di livello di una funzione in due variabili.

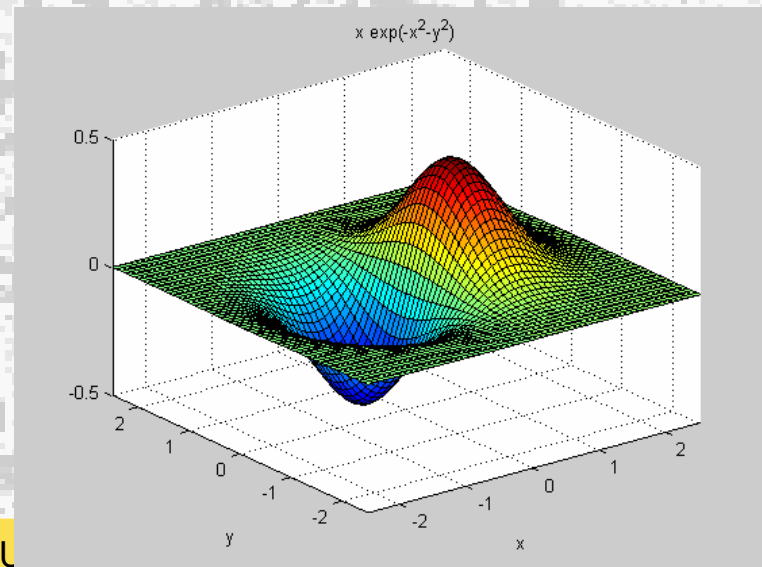
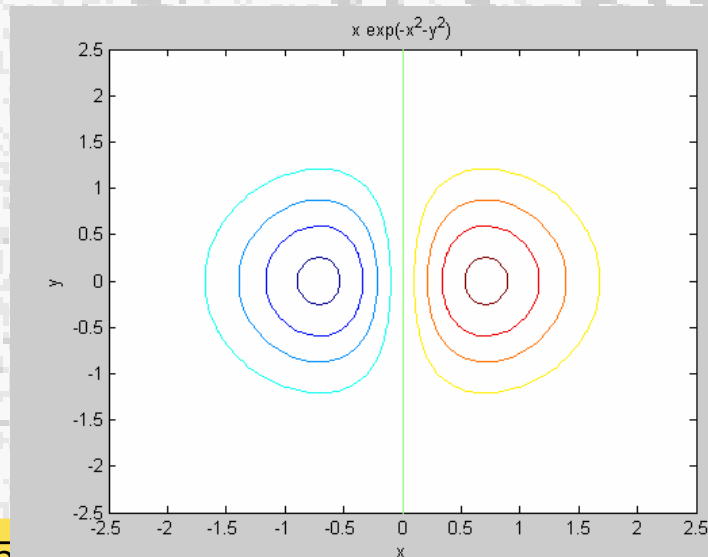
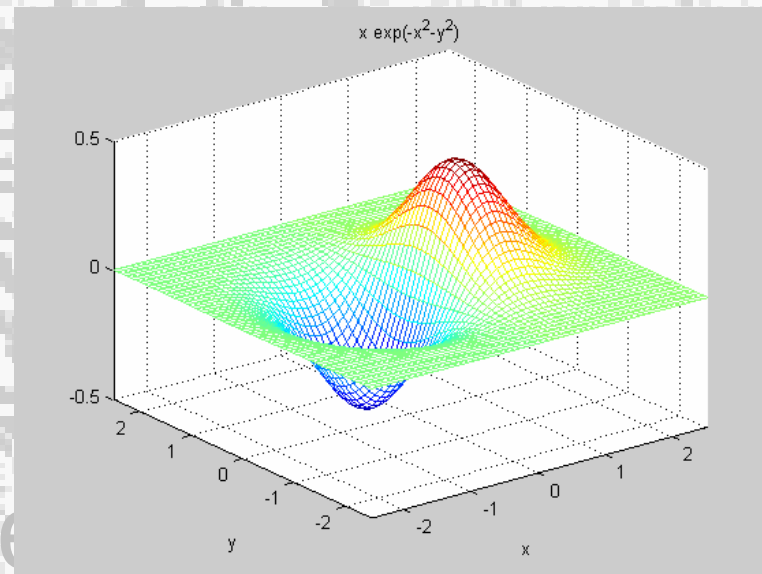
Sintassi

funzione_plot(f) traccia il grafico della funzione $f(x,y)$ con $-2\pi < x < 2\pi$ e $-2\pi < y < 2\pi$

funzione_Plot(f,d) traccia il grafico della funzione $f(x,y)$ sul dominio $d = [xmin, xmax, ymin, ymax]$

Grafici di funzioni

```
Command Window
File Edit Debug Desktop Window Help
>> syms x y
>> ezmesh(x*exp(-x^2-y^2),[-2.5,2.5])
>> ezsurf(x*exp(-x^2-y^2),[-2.5,2.5])
>> ezcontour(x*exp(-x^2-y^2),[-2.5,2.5])
>>
```



Equazioni algebriche

La funzione **solve** risolve un'equazione simbolica.

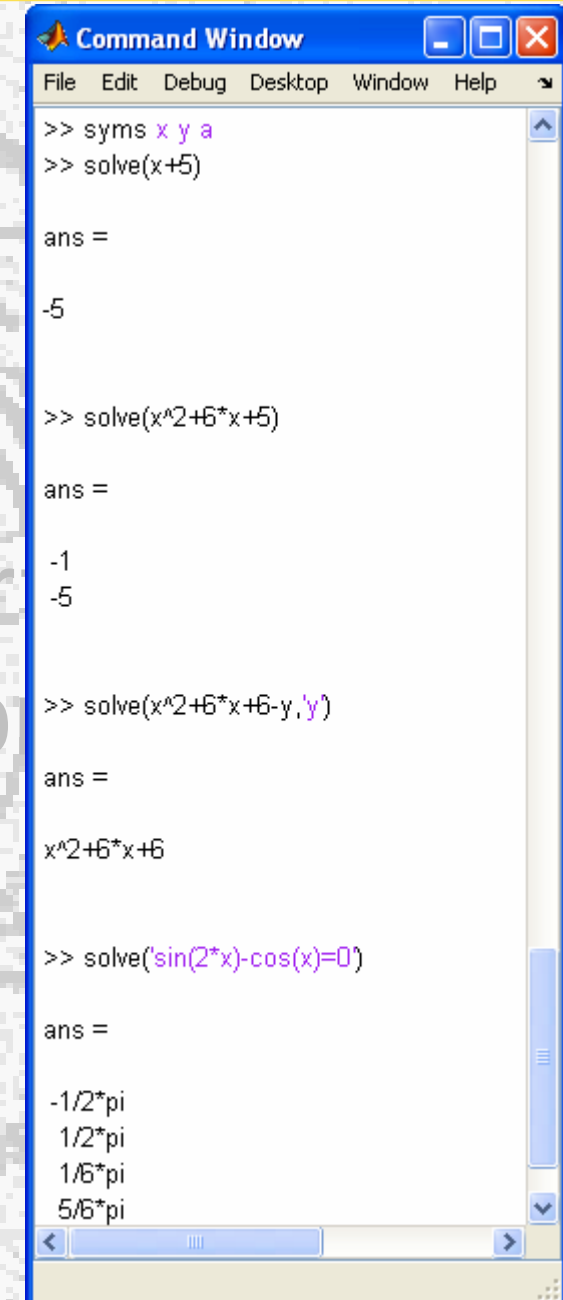
Sintassi

solve(E) risolve l'equazione rispetto alla variabile di default. Se E contiene il segno di uguale(=) deve essere racchiusa tra apici.

Se E non contiene il segno di uguale, non deve essere racchiusa tra apici e viene risolta l'equazione 'E=0'.

solve(E, 'v') come sopra, rispetto alla variabile v.

In un'operazione di assegnamento il risultato è memorizzato in una struttura.



```
Command Window
File Edit Debug Desktop Window Help
>> syms x y a
>> solve(x+5)

ans =

-5

>> solve(x^2+6*x+5)

ans =

-1
-5

>> solve(x^2+6*x+6-y, y)

ans =

x^2+6*x+6

>> solve('sin(2*x)-cos(x)=0')

ans =

-1/2*pi
1/2*pi
1/6*pi
5/6*pi
```

Sistemi di equazioni

La funzione **solve** può risolvere un sistema di equazioni.

Sintassi

solve(E1, E2, ... En) risolve il sistema di equazioni E1, E2, ... En rispetto alle prime n variabili determinate da findsym.

Per ognuna delle espressioni valgono le stesse considerazioni sul segno di uguale.

solve(E1, E2, ... En, 'var1', 'var2', ... 'varn') risolve il sistema di equazioni E1, E2, ... En rispetto alle variabili var1, var2, ... varn.

Sistemi di equazioni

Esempio: intersezione tra due cerchi.

Trovare i punti di intersezione tra

$$(x-3)^2 + (y-5)^2 = 4$$

e

$$(x-5)^2 + (y-3)^2 = b^2$$

```
>> syms x y b
>> e1=(x-3)^2+(y-5)^2-4;
>> e2=(x-5)^2+(y-3)^2-b^2;
>> s=solve(e1,e2)

s =

    x: [2x1 sym]
    y: [2x1 sym]

>> s.x

ans =

9/2-1/8*b^2+1/8*(-16+24*b^2-b^4)^(1/2)
9/2-1/8*b^2-1/8*(-16+24*b^2-b^4)^(1/2)

>> s.y

ans =

7/2+1/8*b^2+1/8*(-16+24*b^2-b^4)^(1/2)
7/2+1/8*b^2-1/8*(-16+24*b^2-b^4)^(1/2)

>> subs(s.x,b,1)

ans =

    4.7057
    4.0443

>> subs(s.y,b,1)

ans =

    3.9557
    3.2943
```

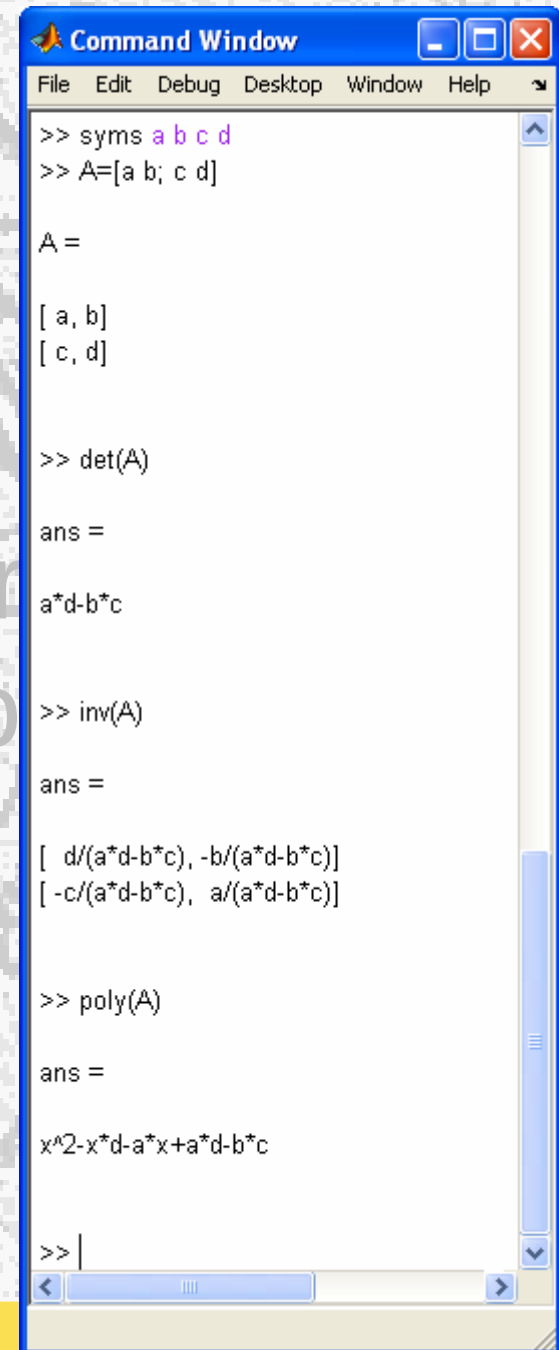
Operazioni su matrici

Alcune funzioni disponibili per le matrici simboliche sono:

- **det** determinante
- **eig** autovalori e autovettori
- **expm** esponenziale
- **inv** inversa
- **jordan** forma di Jordan
- **poly** polinomio caratteristico
- **rank** rango

Sintassi

var = nomeFunzione(matrice simbolica)



```
Command Window
File Edit Debug Desktop Window Help
>> syms a b c d
>> A=[a b; c d]

A =

[ a, b]
[ c, d]

>> det(A)

ans =

a*d-b*c

>> inv(A)

ans =

[ d/(a*d-b*c), -b/(a*d-b*c)]
[ -c/(a*d-b*c), a/(a*d-b*c)]

>> poly(A)

ans =

x^2-x*d-a*x+a*d-b*c

>> |
```

Trasformata di Laplace

La funzione **laplace** calcola la trasformata di Laplace di un'espressione simbolica.

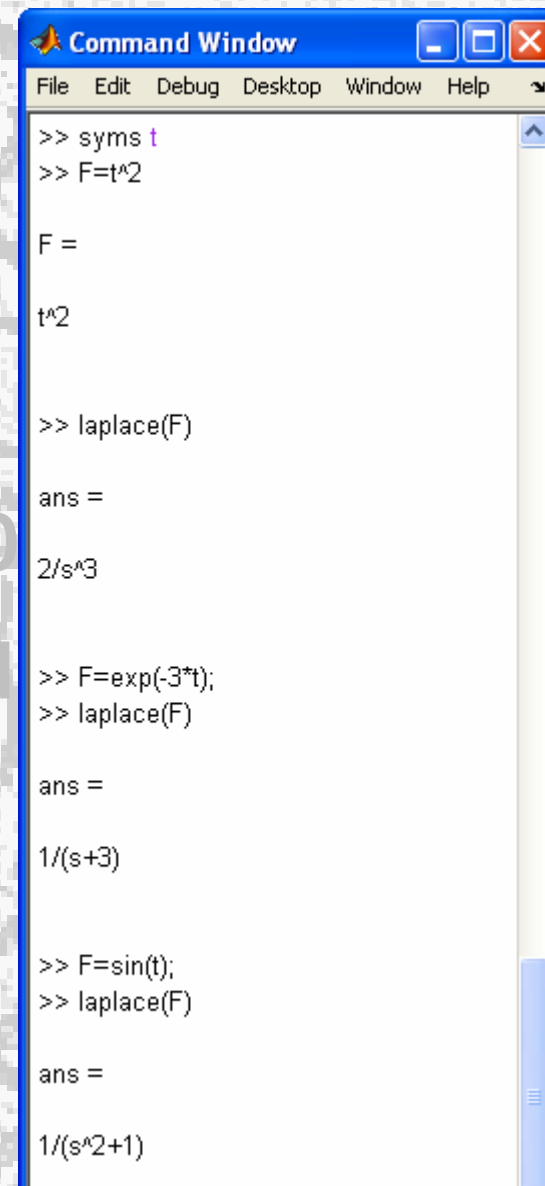
$$L(s) = \int_0^{\infty} F(t) e^{-st} dt$$

Sintassi

$$L = \text{laplace}(F)$$

F ha come variabile indipendente t.

L ha come variabile indipendente s

A screenshot of the MATLAB Command Window. The window has a blue title bar with the text 'Command Window' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Debug', 'Desktop', 'Window', and 'Help'. The main area contains the following text:

```
>> syms t
>> F=t^2

F =

t^2

>> laplace(F)

ans =

2/s^3

>> F=exp(-3*t);
>> laplace(F)

ans =

1/(s+3)

>> F=sin(t);
>> laplace(F)

ans =

1/(s^2+1)
```

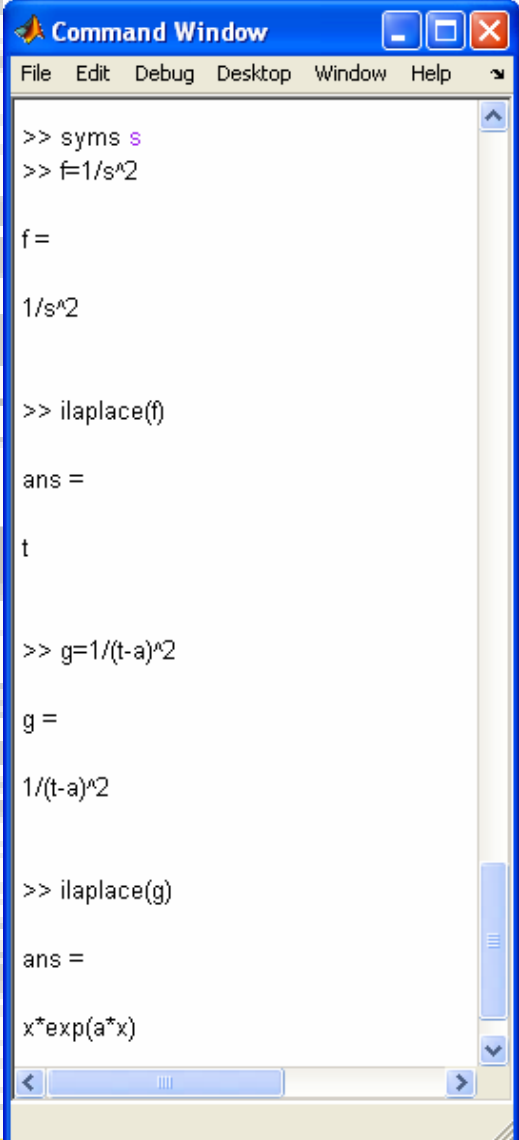
Trasformata inversa di Laplace

La funzione **ilaplace** calcola la trasformata inversa di Laplace di un'espressione simbolica.

Sintassi

$$F = \text{ilaplace}(L)$$

F ha come variabile indipendente t, L ha come variabile indipendente s



```
Command Window
File Edit Debug Desktop Window Help
>> syms s
>> f=1/s^2

f =

1/s^2

>> ilaplace(f)

ans =

t

>> g=1/(t-a)^2

g =

1/(t-a)^2

>> ilaplace(g)

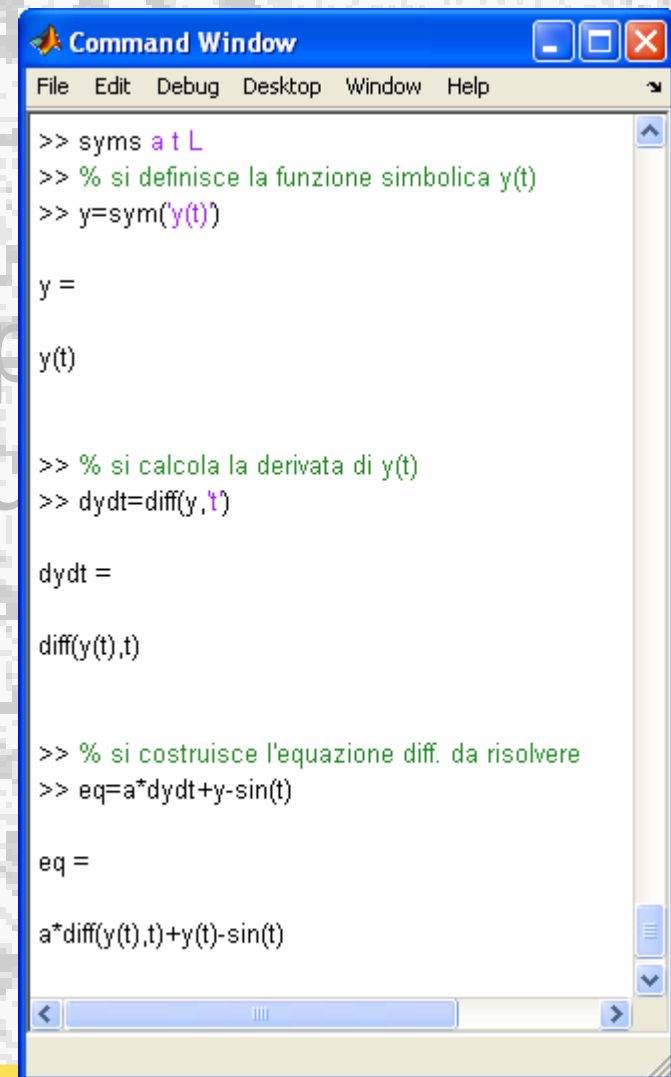
ans =

x*exp(a*x)
```

Risoluzioni di equazioni differenziali con la trasformata di Laplace

Si vuole risolvere la seguente equazione differenziale:

$$a \frac{dy(t)}{dt} + y(t) = \sin(t)$$



```
Command Window
File Edit Debug Desktop Window Help
>> syms a t L
>> % si definisce la funzione simbolica y(t)
>> y=sym('y(t)')

y =
y(t)

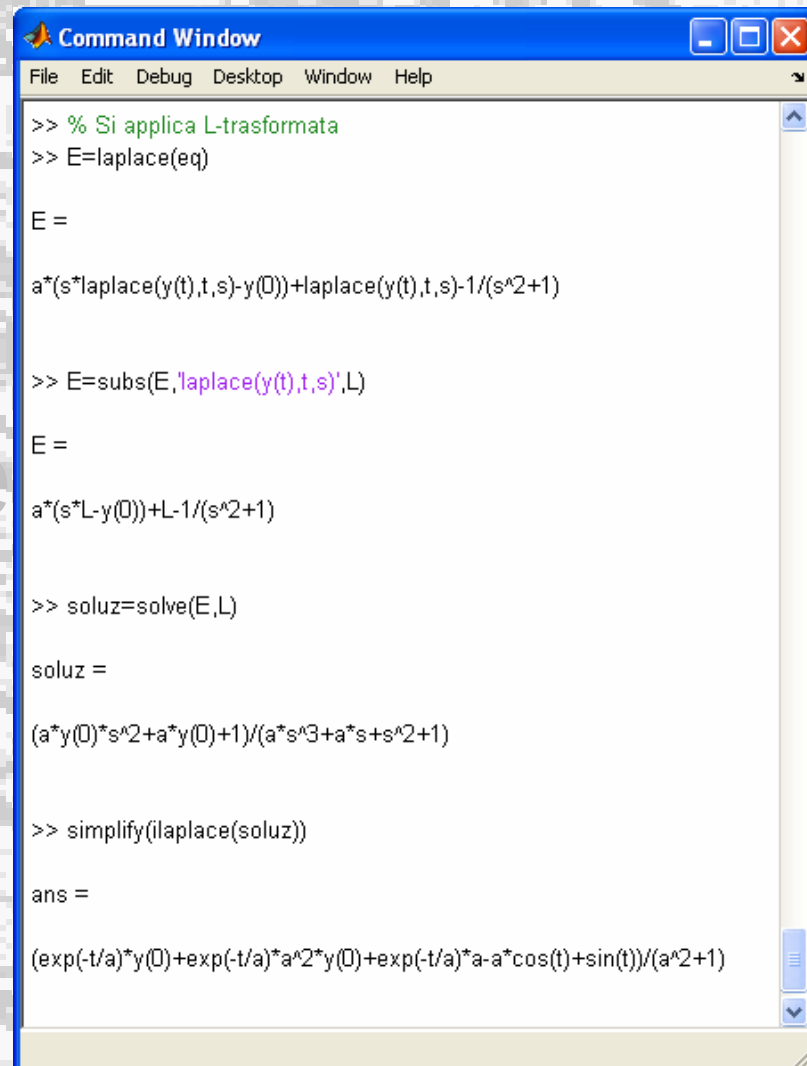
>> % si calcola la derivata di y(t)
>> dydt=diff(y,t)

dydt =
diff(y(t),t)

>> % si costruisce l'equazione diff. da risolvere
>> eq=a*dydt+y-sin(t)

eq =
a*diff(y(t),t)+y(t)-sin(t)
```

Risoluzioni di equazioni differenziali con la trasformata di Laplace



```
Command Window
File Edit Debug Desktop Window Help

>> % Si applica L-trasformata
>> E=laplace(eq)

E =

a*(s*laplace(y(t),t,s)-y(0))+laplace(y(t),t,s)-1/(s^2+1)

>> E=subs(E,'laplace(y(t),t,s)',L)

E =

a*(s*L-y(0))+L-1/(s^2+1)

>> soluz=solve(E,L)

soluz =

(a*y(0)*s^2+a*y(0)+1)/(a*s^3+a*s+s^2+1)

>> simplify(ilaplace(soluz))

ans =

(exp(-t/a)*y(0)+exp(-t/a)*a^2*y(0)+exp(-t/a)*a-a*cos(t)+sin(t))/(a^2+1)
```

Trasformata Z

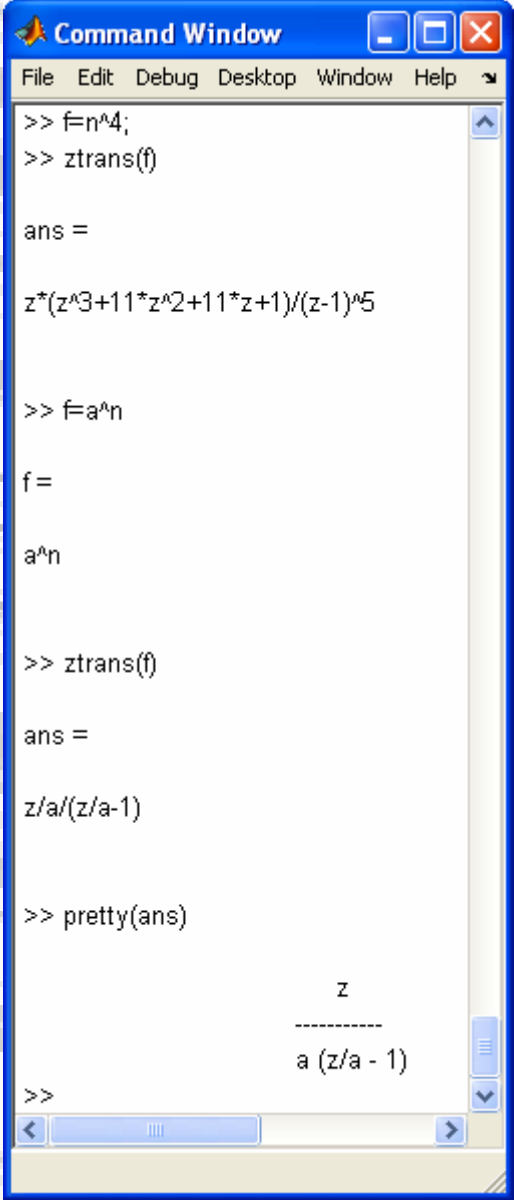
La funzione **ztrans** calcola la trasformata di Z di un'espressione simbolica.

$$G(z) = \sum_{n=0}^{\infty} z^{-n} F(n)$$

Sintassi

$$G = \text{laplace}(F)$$

F ha come variabile indipendente n
G ha come variabile indipendente z



```

>> f=n^4;
>> ztrans(f)

ans =

z*(z^3+11*z^2+11*z+1)/(z-1)^5

>> f=a^n
f =

a^n

>> ztrans(f)

ans =

z/a/(z/a-1)

>> pretty(ans)

      z
-----
a (z/a - 1)
  
```

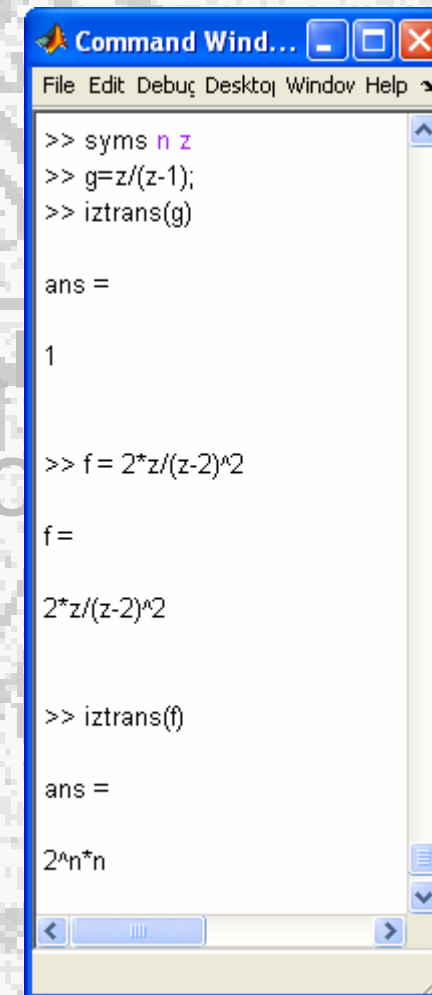
Inversa della trasformata

La funzione **iztrans** calcola la trasformata inversa di Z di un'espressione simbolica.

Sintassi

$$F = \text{iztrans}(G)$$

F ha come variabile indipendente n G ha come variabile indipendente z



```
Command Window
File Edit Debug Desktop Window Help
>> syms n z
>> g=z/(z-1);
>> iztrans(g)

ans =

1

>> f= 2*z/(z-2)^2
f =

2*z/(z-2)^2

>> iztrans(f)

ans =

2^n*n
```

LA GUI

Una GUI (**G**raphical **U**ser **I**nterface) è un'interfaccia costruita con oggetti grafici, come bottoni, campi testo, menu.

Si tratta di un'interfaccia tra l'utente e il programma.



La GUI permette di utilizzare il programma senza dover conoscere quali siano le istruzioni che vanno immesse nella Command Window, oppure quali devono essere i parametri di ingresso e di uscita delle funzioni.

Un'applicazione che presenta una GUI è più semplice da usare.

LA GUI

Una GUI ben progettata dovrebbe essere intuitiva da utilizzare.

Matlab implementa una GUI come un oggetto di tipo **figure** (è lo stesso che viene creato con il comando plot) contenete vari controlli, detti **uicontrol objects**, come tasti campi editabili, barre di scorrimento, assi.

Università di Pisa - Dipartimento di
Sistemi Elettrici e Automazione

LA GUI

Creare una GUI significa:

- Costruire l'interfaccia

occorre progettare e realizzare l'aspetto grafico della GUI e quindi definire quali oggetti inserire e come disporli.

- Programmare ogni oggetto

per ogni oggetto occorre definire l'azione (sequenza di comandi), detta **callback**, che deve essere eseguita quando l'oggetto viene attivato

Il tool GUIDE

Matlab mette a disposizione un ambiente grafico, chiamato **GUIDE**, GUI Design Environment (ambiente di progettazione GUI), che rappresenta un semplice strumento per la realizzazione di interfaccia grafiche.

GUIDE è prima di tutto un ambiente grafico per costruire in modo semplice la struttura della GUI.

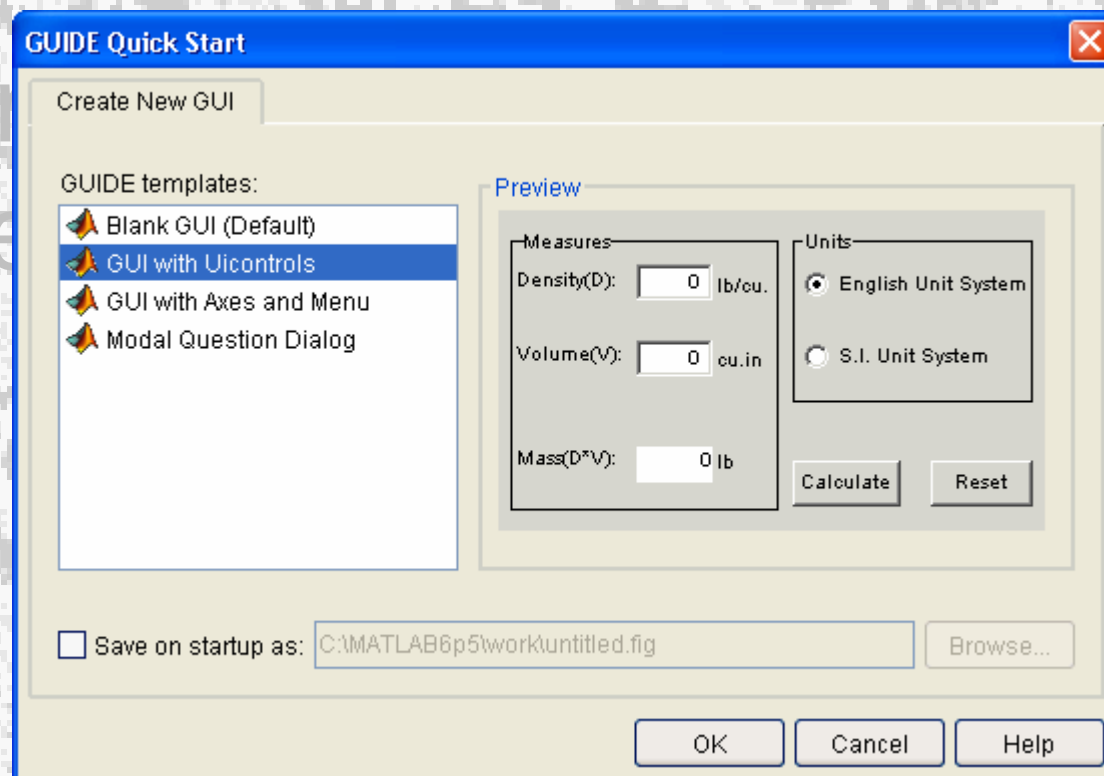
Al momento dell'avvio, Matlab genera in modo automatico due file:

- un Fig-file (file con estensione .fig) che contiene la completa descrizione grafica della GUI;
- un M-file che contiene le funzioni che avviano e gestiscono la GUI e le callbacks relative a tutti i controlli presenti nella GUI. In particolar modo per ogni controllo è presente l'intestazione della funzione che si deve avviare quando si genera l'evento relativo al controllo in esame. Il programmatore non deve fare altro che inserire il codice voluto all'interno della callback.

Il tool GUIDE

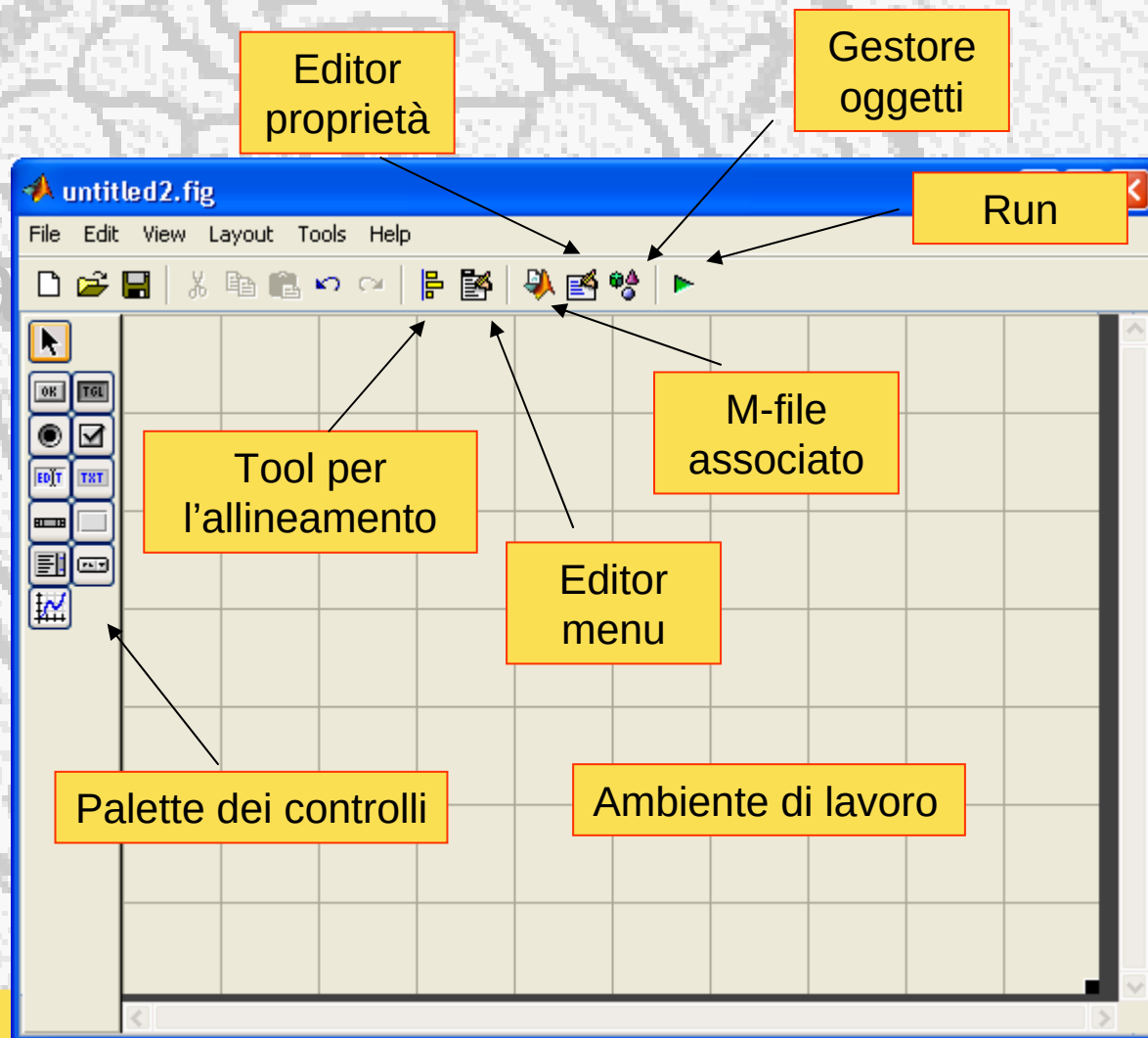
GUIDE si avvia dal prompt dei comandi con il comando **guide**.

All'avvio GUIDE mette a disposizione quattro diversi modelli di GUI che l'utente può modificare in modo da creare la propria GUI.



Il tool GUIDE

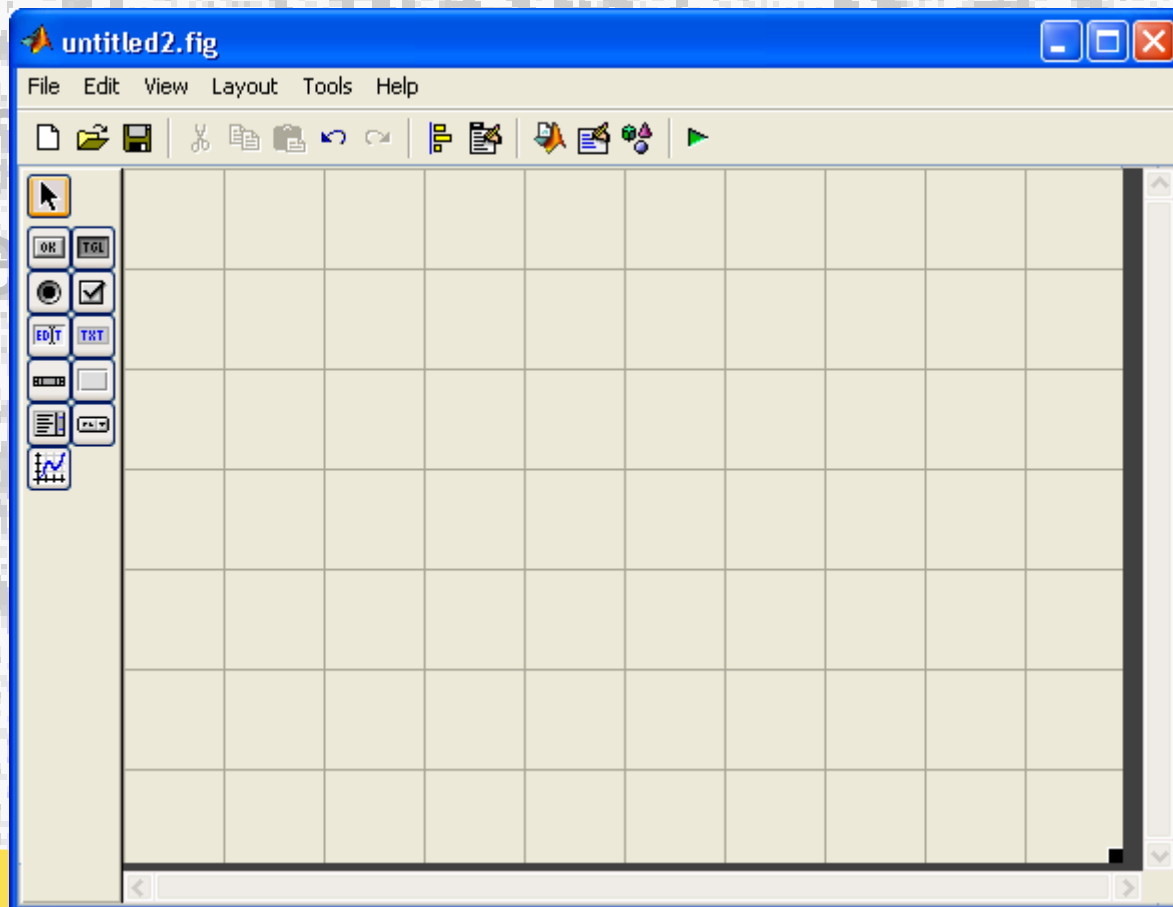
Una volta selezionato il modello si avvia il tool



Il tool GUIDE II Layout editor

Il GUIDE si compone di vari tool:

- il **Layout editor** permette di selezionare i componenti nella palette degli oggetti e di costruire in modo grafico il layout della GUI



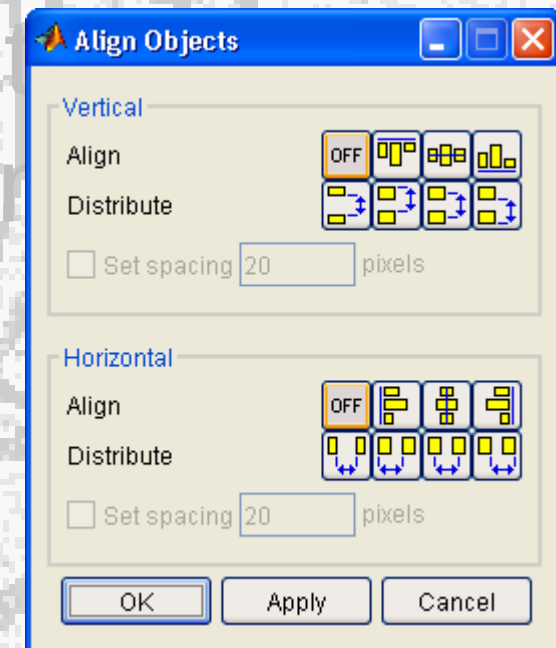
Il tool GUIDE L'Alignment tool

- L'**Alignment tool** consente di posizionare i vari oggetti in modo appropriato.

Si avvia dalla barra di comando selezionando la relativa icona



oppure dal menu **Tool->Align Object.**

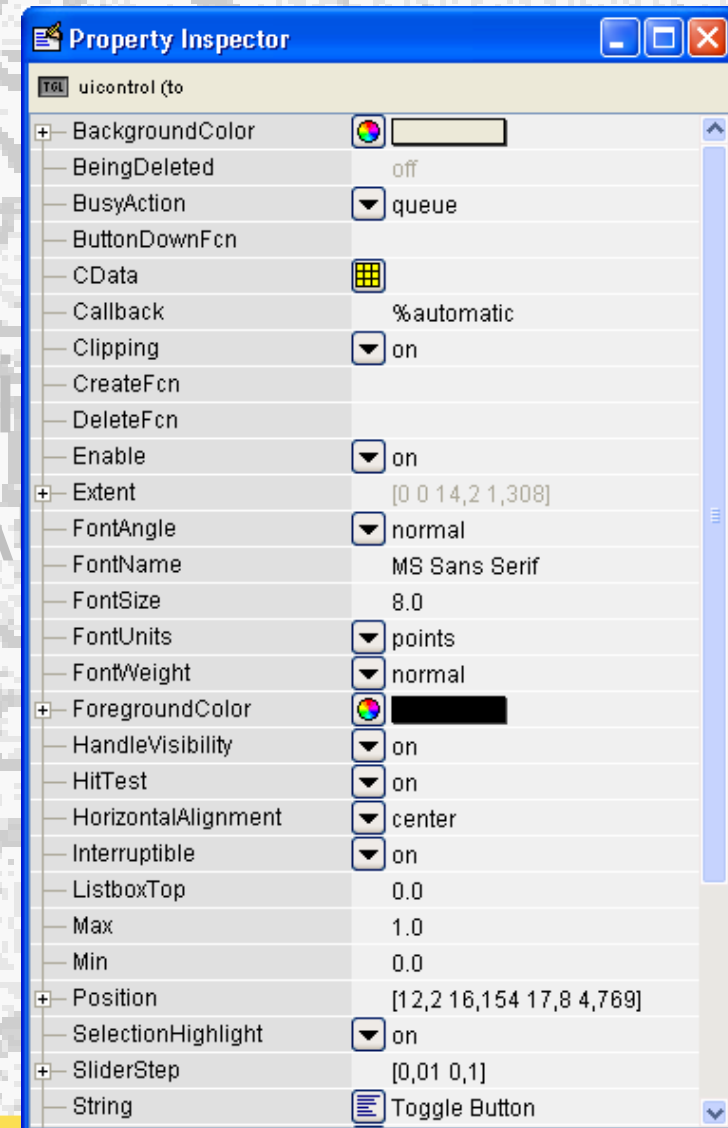


Il tool GUIDE Il Property Inspector

- Il **Property Inspector** permette di accedere alle proprietà dell'oggetto selezionato e di modificarle. Si avvia dalla barra di comando selezionando la relativa icona

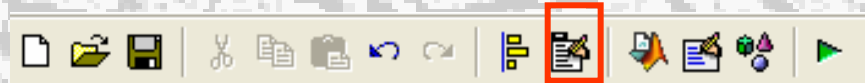


oppure selezionando l'oggetto e con il tasto destro del mouse si sceglie la voce Property Inspector. Si può avviare anche dal menu **View->Property Inspector**

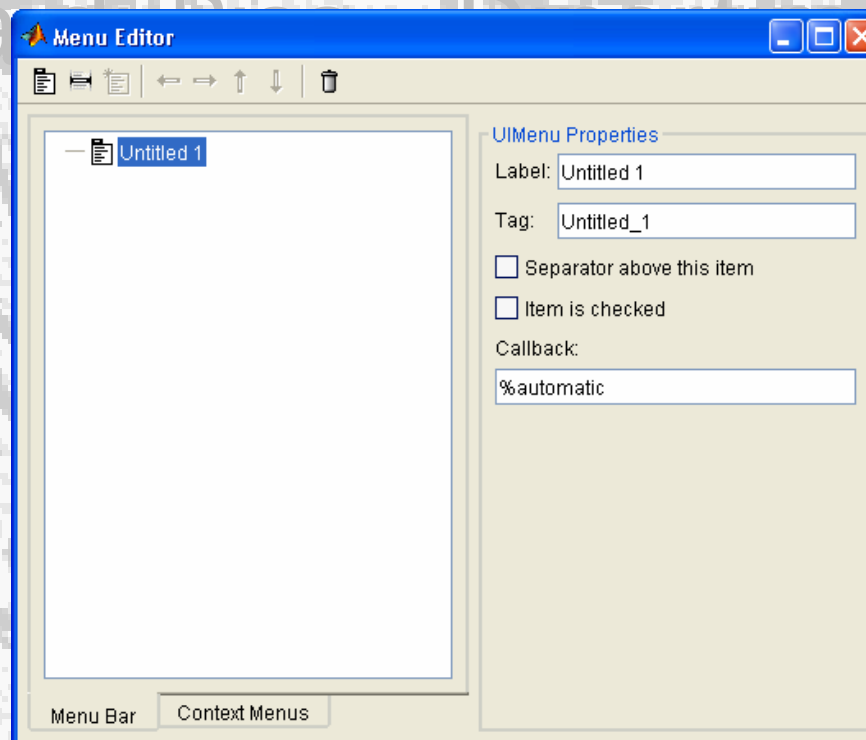


Il tool GUIDE Il menu editor

- Il **menu editor** permette di creare i menu della GUI. Si avvia dalla barra di comando selezionando la relativa icona



oppure dal menu **Tool->Menu editor**.

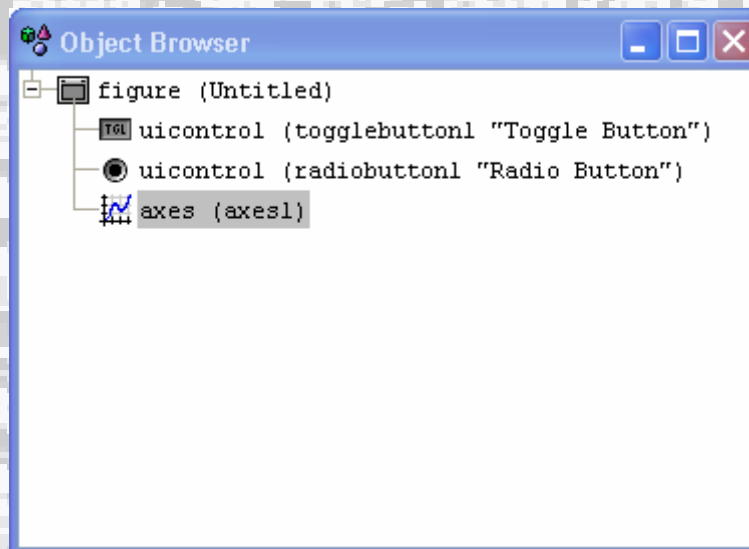


Il tool GUIDE L' Object browser

- L'**Object browser** mostra la lista degli oggetti presenti nella GUI, indicando per ognuno il grado di parentela con gli altri. Si avvia dalla barra di comando selezionando la relativa icona



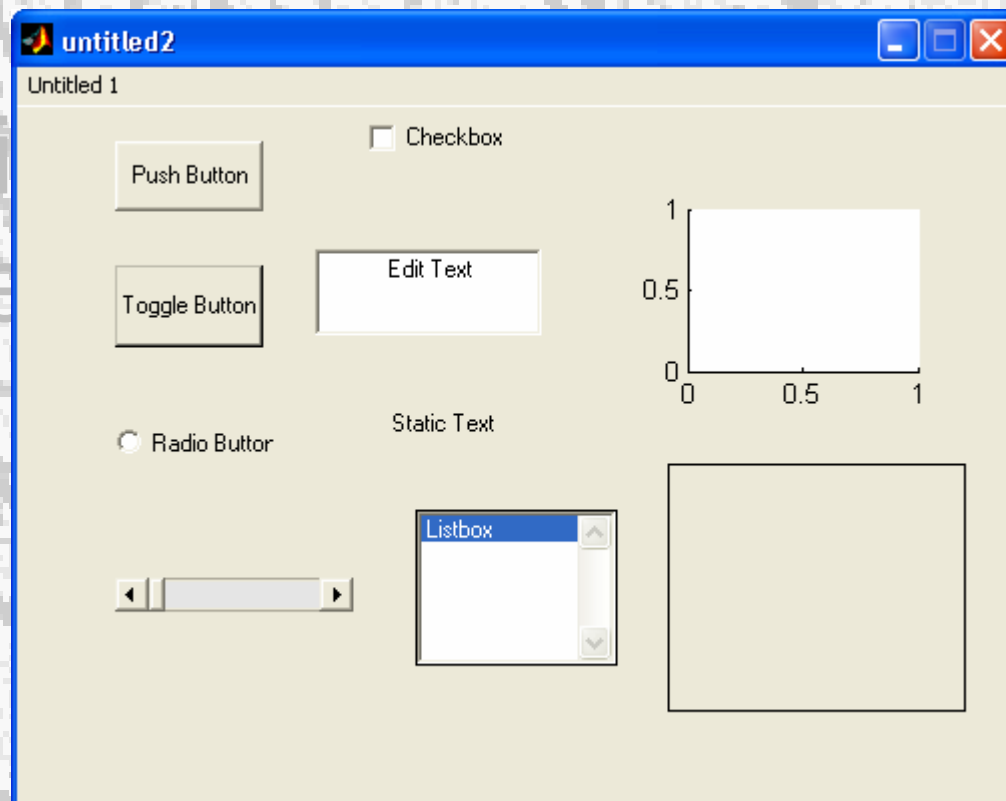
oppure dal menu **View-> Object browser**.



Il tool GUIDE Gli oggetti

Gli oggetti (guicontrol) che possono essere inseriti in una GUI sono raccolti nella palette.

- Push Button
- Toggle Button
- Radio Button
- Check Boxes
- Edit Text
- Static Text
- Slide
- Frames
- List Boxes
- Pop-Up Menu
- Axes



Il tool GUIDE Le proprietà di un oggetto

Ogni oggetto possiede un certo numero di proprietà. Queste possono essere esaminate nel Property Inspector. Tutte le proprietà sono modificabili e acquisibili in fase di progettazione della GUI (proprio con il property inspector), oppure anche durante il funzionamento della GUI con le funzioni **get** e **set**.

Get permette di acquisire il valore di una proprietà

Sintassi

var=get(puntatore_oggetto, 'proprietà')

Set permette di impostare il valore di una proprietà

Sintassi

set(puntatore_oggetto, 'proprietà', valore)

Il tool GUIDE Le proprietà di un oggetto

Il puntatore dell'oggetto è una variabile che contiene l'indirizzo di memoria in cui l'oggetto è memorizzato. I puntatori di tutti gli oggetti che compongono la GUI sono memorizzati in modo automatico in una struttura chiamata **handles**. Si ha un campo per il puntatore di ogni oggetto.

Il nome di tale campo coincide con la stringa contenuta nella proprietà **Tag** dell'oggetto. Quindi il puntatore ad un generico oggetto è sempre contenuto in una variabile definita nel seguente modo:

handles.stringa_Tag

Il tool GUIDE L'M-file

Una volta costruito il layout della GUI si può accedere al M-file per poter inserire il codice nelle callback associate ai vari controlli. Si accede all'editor del M-file dalla relativa icona:



oppure dal menu **View->M-file editor**

Le sezioni editabili sono:

- la funzione di apertura della GUI (*opening*)
viene eseguita al momento che la GUI è avviata
- la funzione di chiusura della GUI (*closing*)
viene eseguita al momento che la GUI è chiusa
- le funzioni associate ai controlli presenti nella GUI

Il tool GUIDE L'M-file

```

1 function varargout = untitled2(varargin)
2 % UNTITLED2 M-file for untitled2.fig
3 % UNTITLED2, by itself, creates a new UNTITLED2 or raises the existing
4 % singleton*.
5 %
6 % H = UNTITLED2 returns the handle to a new UNTITLED2 or the handle to
7 % the existing singleton*.
8 %
9 % UNTITLED2('CALLBACK',hObject,eventData,handles,...) calls the local
10 % function named CALLBACK in UNTITLED2.M with the given input arguments.
11 %
12 % UNTITLED2('Property','Value',...) creates a new UNTITLED2 or raises the
13 % existing singleton*. Starting from the left, property value pairs are
14 % applied to the GUI before untitled2_OpeningFunction gets called. An
15 % unrecognized property name or invalid value makes property application
16 % stop. All inputs are passed to untitled2_OpeningFcn via varargin.
17 %
18 % *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
19 % instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help untitled2
24
25 % Last Modified by GUIDE v2.5 06-Mar-2005 15:41:39
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
  
```

Il tool GUIDE L'M-file

Tutte le funzioni presentano la seguente intestazione:

```
function tag_oggetto_Callback(hObject, eventdata, handles)
```

hObject puntatore dell'oggetto

eventdata riservato – per future versioni di Matlab

handles struttura con puntatori e dati

Il tool GUIDE Esempio

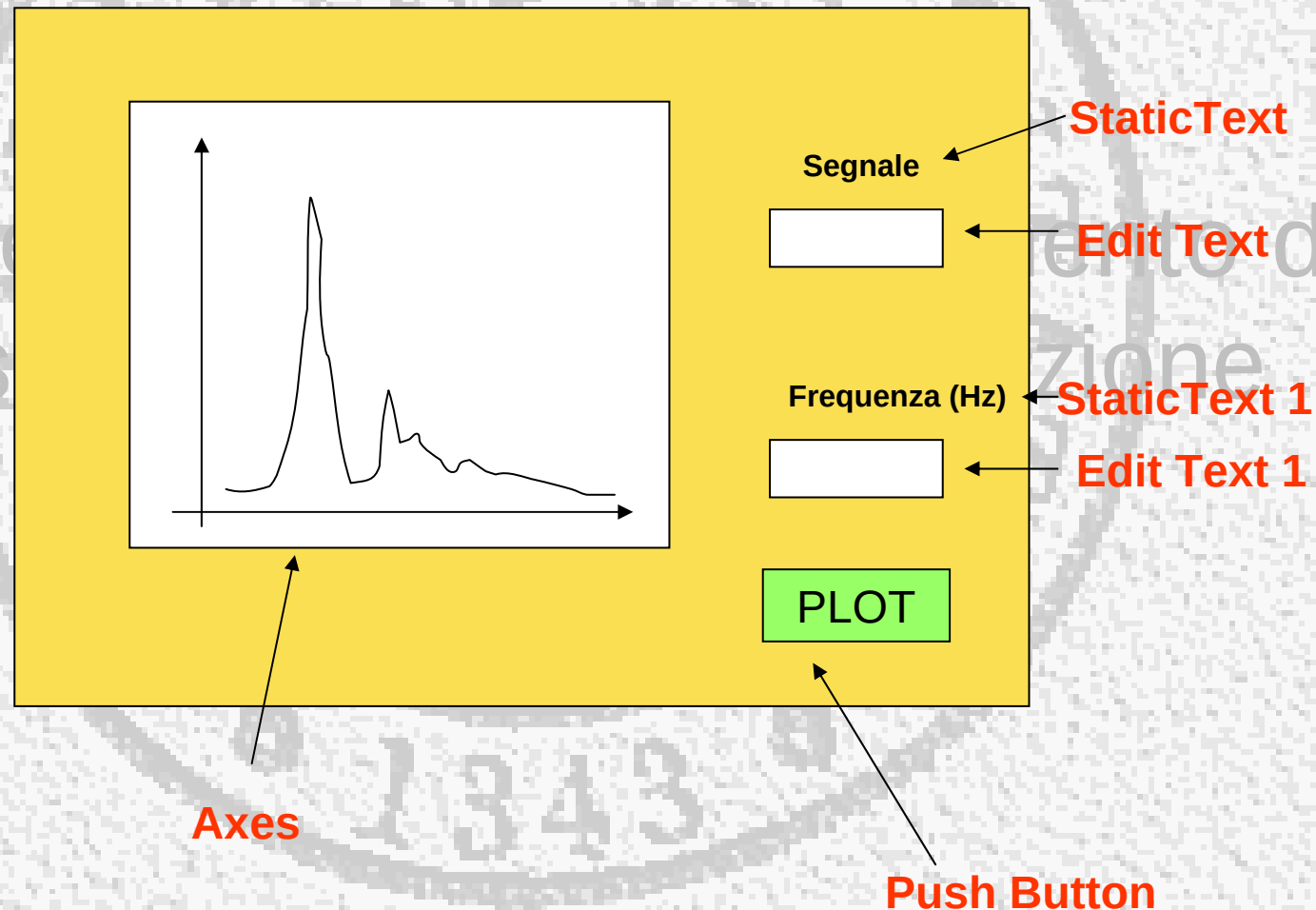
Si vuol realizzare la GUI di un analizzatore di spettro.

Funzioni:

- si seleziona la variabile nel workspace di cui si vuole lo spettro;
- si deve specificare la frequenza di campionamento;
- con questi dati si fa disegnare lo spettro di ampiezza del segnale.

Il tool GUIDE Esempio

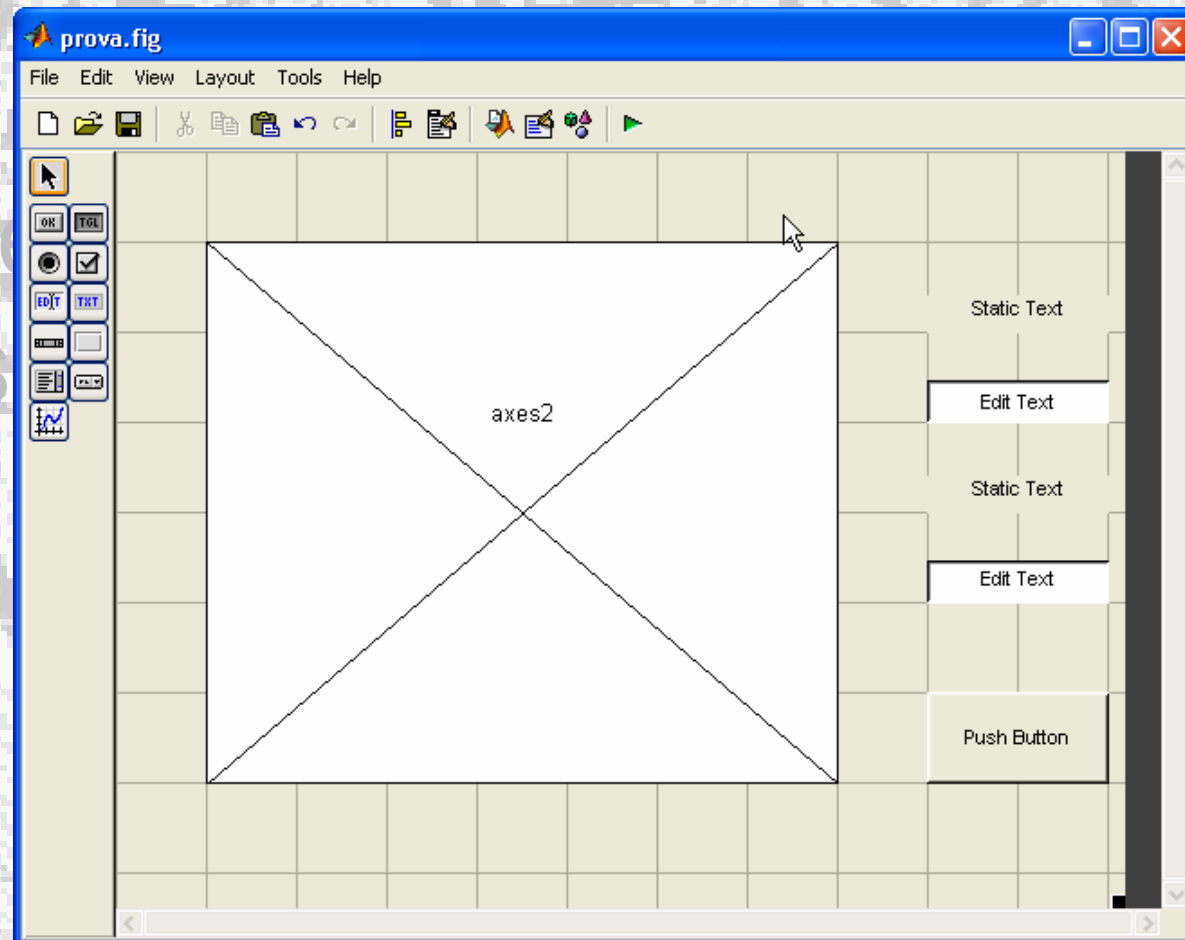
La GUI si dovrà presentare nel seguente modo:



Il tool GUIDE Esempio

Si avvia il GUIDE tool e si seleziona il modello Blank

Utilizzando il Layout Editor si costruisce la GUI

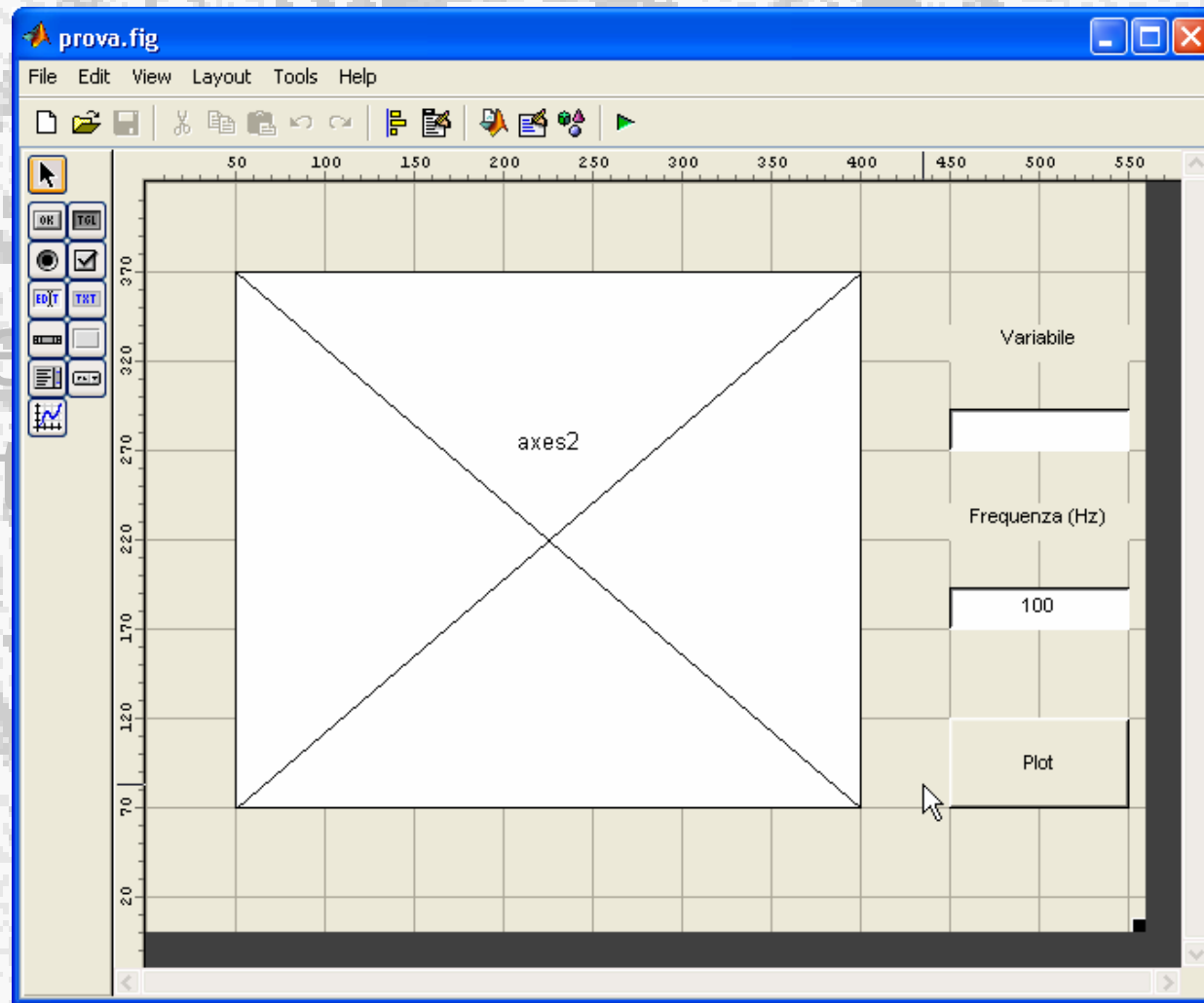


Il tool GUIDE Esempio

Con il Property Inspector si definiscono le proprietà dei vari oggetti:

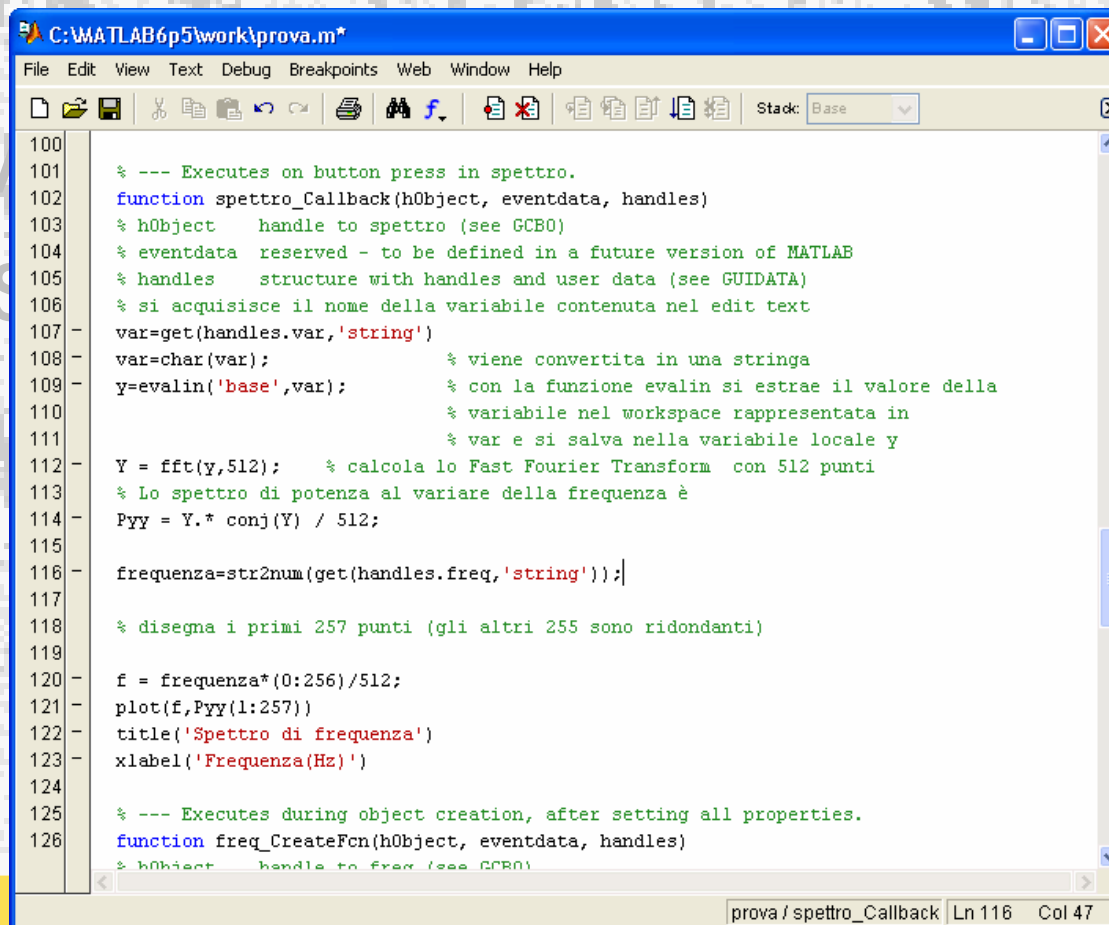
- per l'oggetto axes si fissa Tag=assi
- per l'oggetto Push Button String= Plot
Tag = spettro
- per l'oggetto static text String = Variabile
- per l'oggetto Edit Text String = 'Automazione'
Tag = var
- per l'oggetto static text 1 String = Frequenza (Hz)
- per l'oggetto Edit Text 1 String = 100
Tag = freq

Il tool GUIDE Esempio



Il tool GUIDE Esempio

Si salva il Fig. file con il nome prova.fig e si avvia l'M-file editor associato prova.m. Il codice va aggiunto nella callback associata al push button con tag spettro

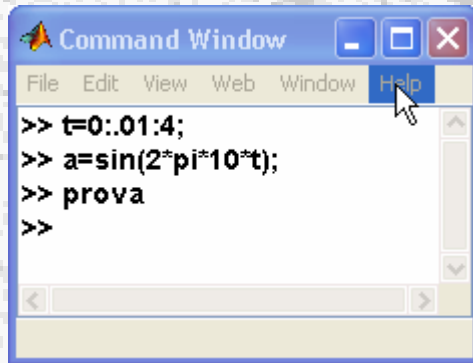


```

C:\MATLAB6p5\work\prova.m*
File Edit View Text Debug Breakpoints Web Window Help
[Icons] Stack: Base
100
101 % --- Executes on button press in spettro.
102 function spettro_Callback(hObject, eventdata, handles)
103 % hObject    handle to spettro (see GCBO)
104 % eventdata  reserved - to be defined in a future version of MATLAB
105 % handles    structure with handles and user data (see GUIDATA)
106 % si acquisisce il nome della variabile contenuta nel edit text
107 var=get(handles.var,'string')
108 var=char(var);           % viene convertita in una stringa
109 y=evalin('base',var);    % con la funzione evalin si estrae il valore della
110                          % variabile nel workspace rappresentata in
111                          % var e si salva nella variabile locale y
112 Y = fft(y,512);          % calcola lo Fast Fourier Transform con 512 punti
113 % Lo spettro di potenza al variare della frequenza è
114 Pyy = Y.* conj(Y) / 512;
115
116 frequenza=str2num(get(handles.freq,'string'))/;
117
118 % disegna i primi 257 punti (gli altri 255 sono ridondanti)
119
120 f = frequenza*(0:256)/512;
121 plot(f,Py(1:257))
122 title('Spettro di frequenza')
123 xlabel('Frequenza(Hz)')
124
125 % --- Executes during object creation, after setting all properties.
126 function freq_CreateFcn(hObject, eventdata, handles)
127 % hObject    handle to freq (see GCBO)
  
```

prova / spettro_Callback Ln 116 Col 47

Il tool GUIDE Esempio



```
>> t=0:0.01:4;  
>> a=sin(2*pi*10*t);  
>> prova  
>>
```

