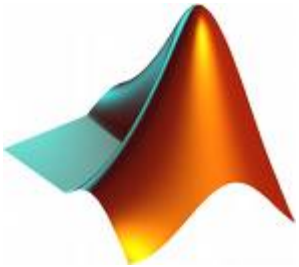


Simulazione dei Sistemi dinamici con Matlab-Simulink

Generalità introduttive

Ing. Alessandro Pisano
apisano@unica.it

Generalità

Matlab è un applicativo per il calcolo scientifico ampiamente utilizzato nell'industria come software per la simulazione dinamica

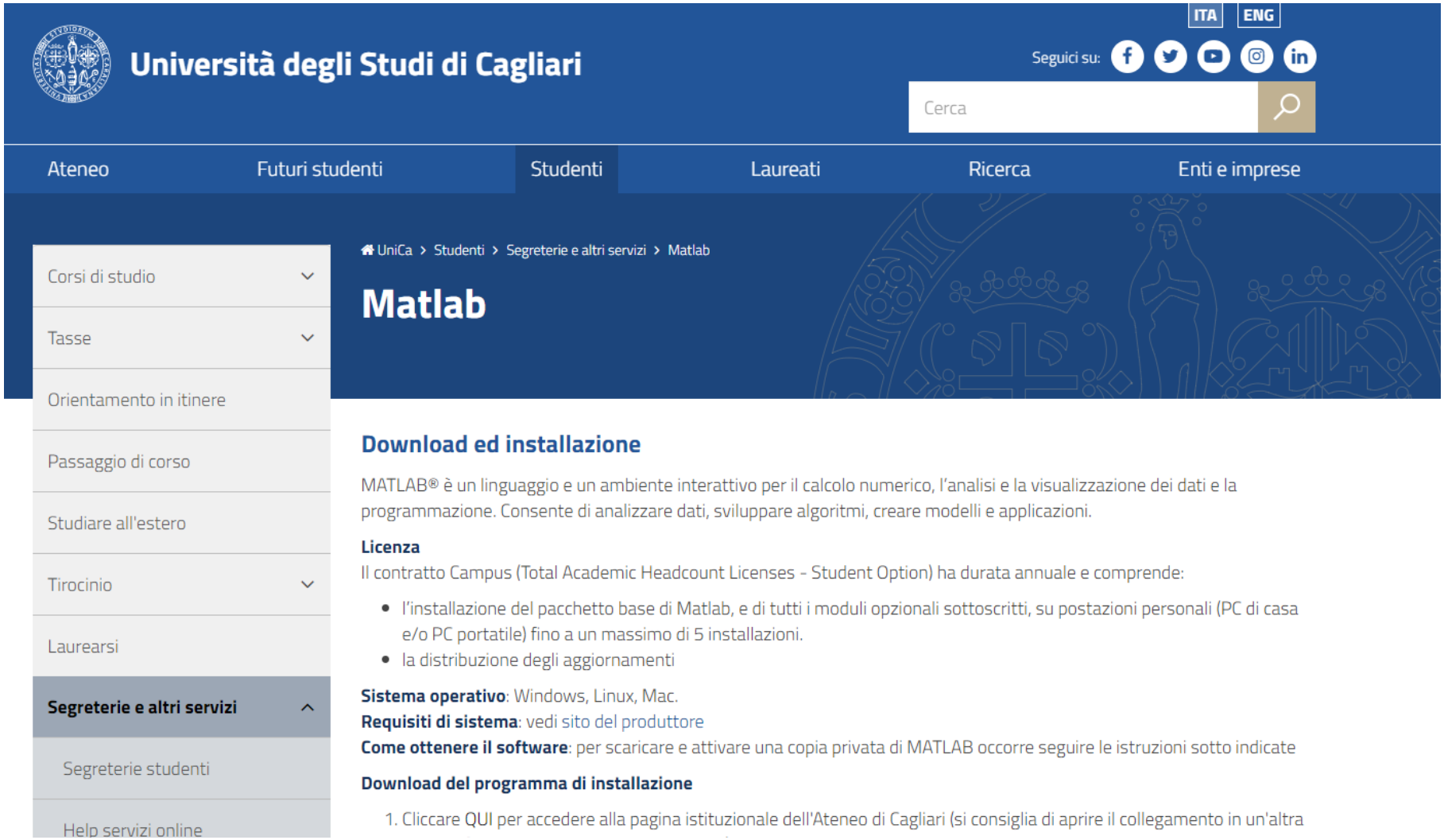
In questo corso faremo una panoramica sulle funzionalità di base.

Impareremo in particolare a creare dei programmi (detti «script») che implementano funzioni matematiche e calcoli avanzati, ciò grazie ad un linguaggio di programmazione di alto livello orientato al calcolo scientifico.

Affronteremo quindi il problema della creazione di grafici di qualità, ed in ultimo ci concentreremo su una particolare interfaccia grafica, denominata Simulink, che consente di modellare e simulare sistemi dinamici complessi in maniera estremamente user-friendly.

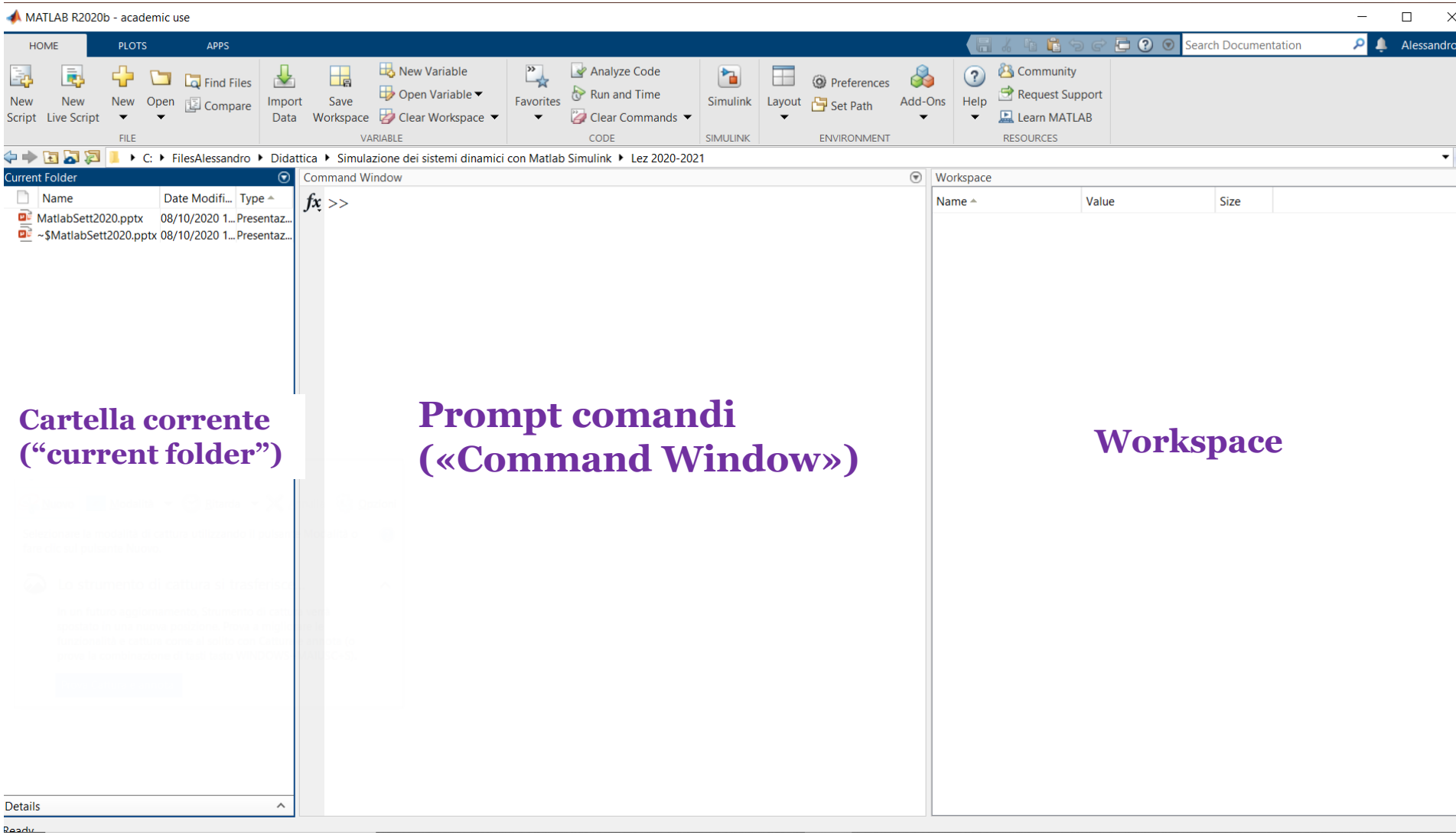
Se non lo avete gia fatto, installate sul vostro laptop l'ultima versione di Matlab utilizzando la licenza Campus di ateneo

`https://unica.it/unica/it/studenti_s08_ss09.page`



Finestra di avvio (v. R2020b)

HOME



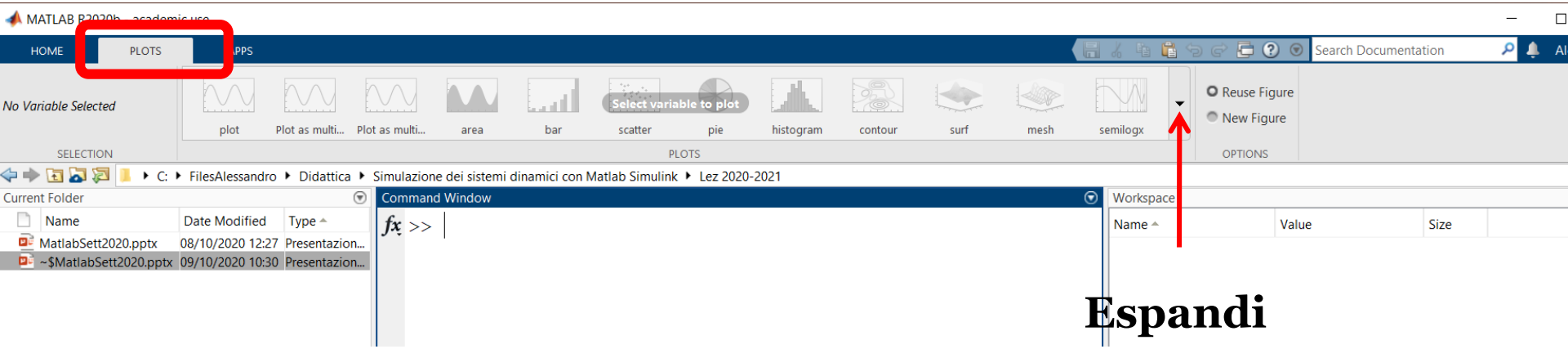
Cartella corrente (“current folder”)

Prompt comandi («Command Window»)

Workspace

Finestra di avvio

PLOTS

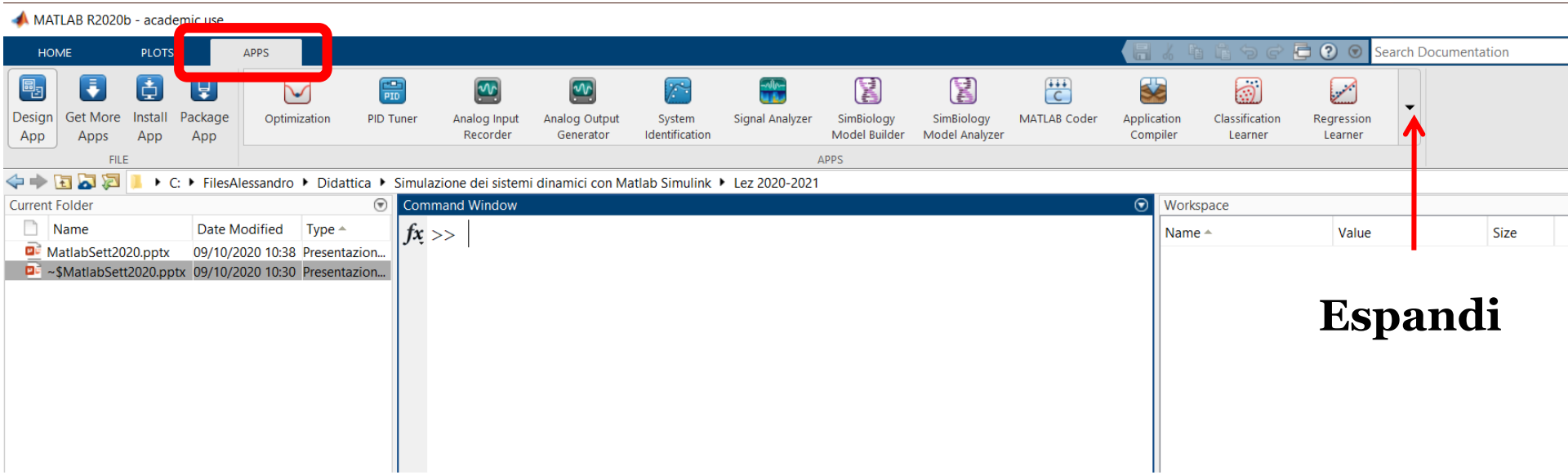


Espandi

Pulsanti per la creazione rapida di grafici

Finestra di avvio

APPS



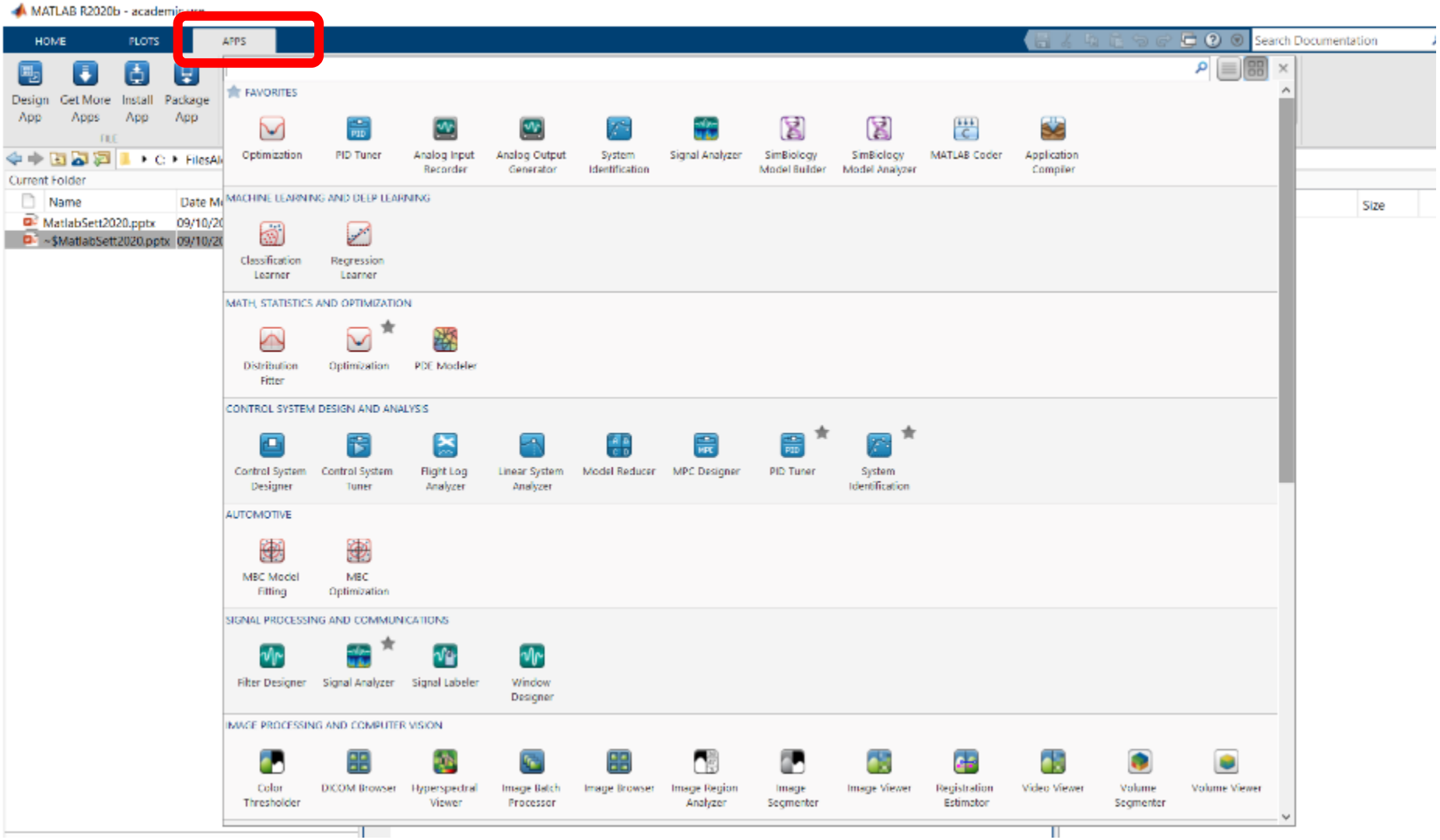
Espandi

Applicazioni dedicate a task particolari

La dotazione di apps dipende dalla particolare installazione, in particolare dai pacchetti (Toolboxed e Toolkit) installati.

Finestra di avvio

APPS

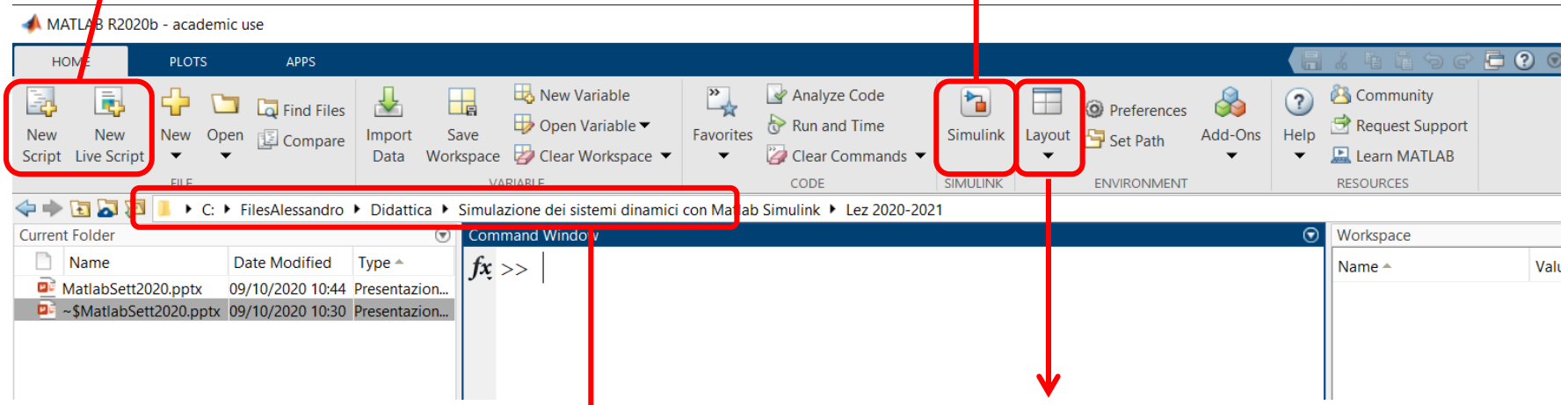


Versione “estesa”

Finestra di avvio

Editor per «Script» e «Live Script»

Avvio SIMULINK

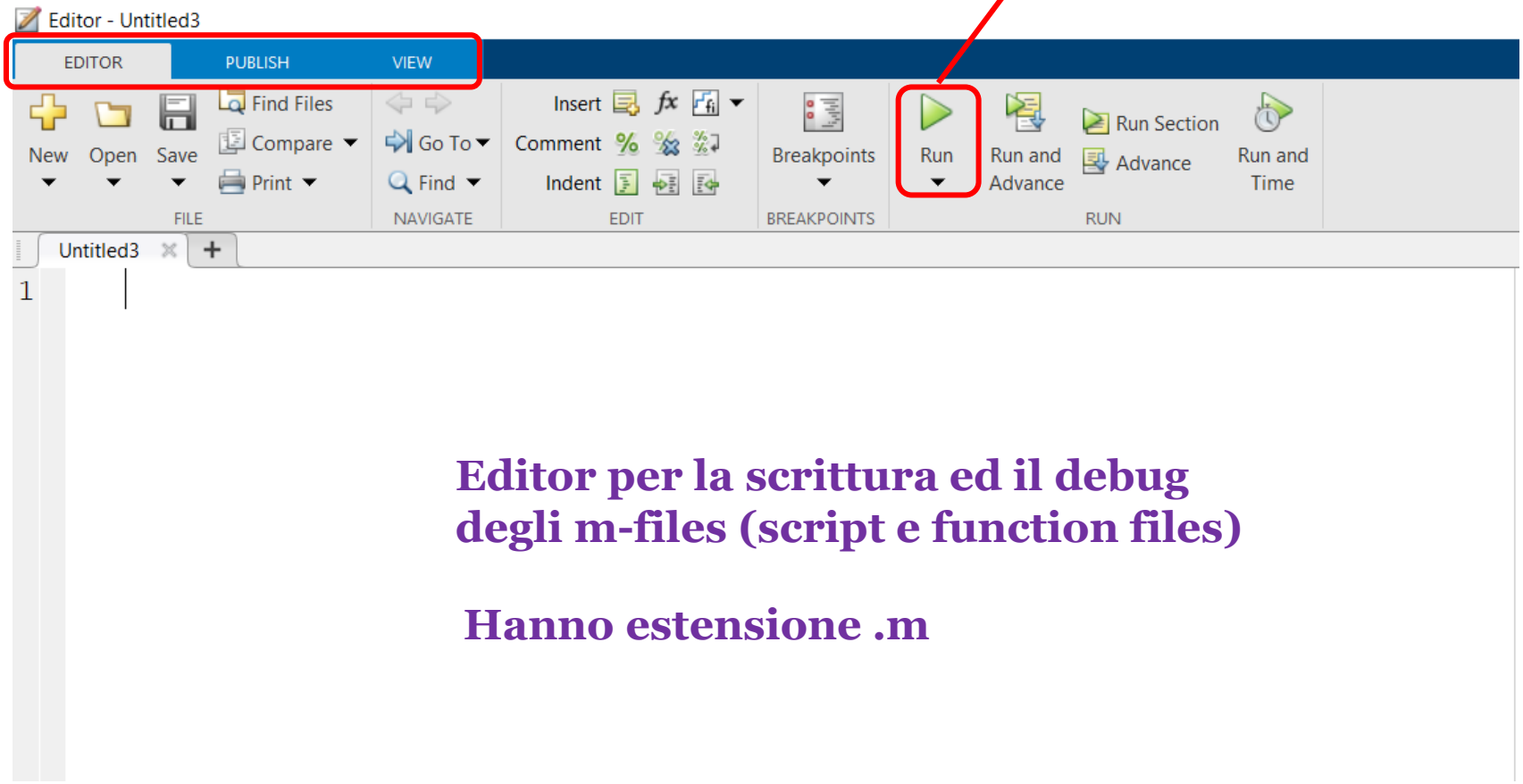


Cartella corrente

Layout della finestra di avvio del programma

Editor di testo per gli Script

Save and run

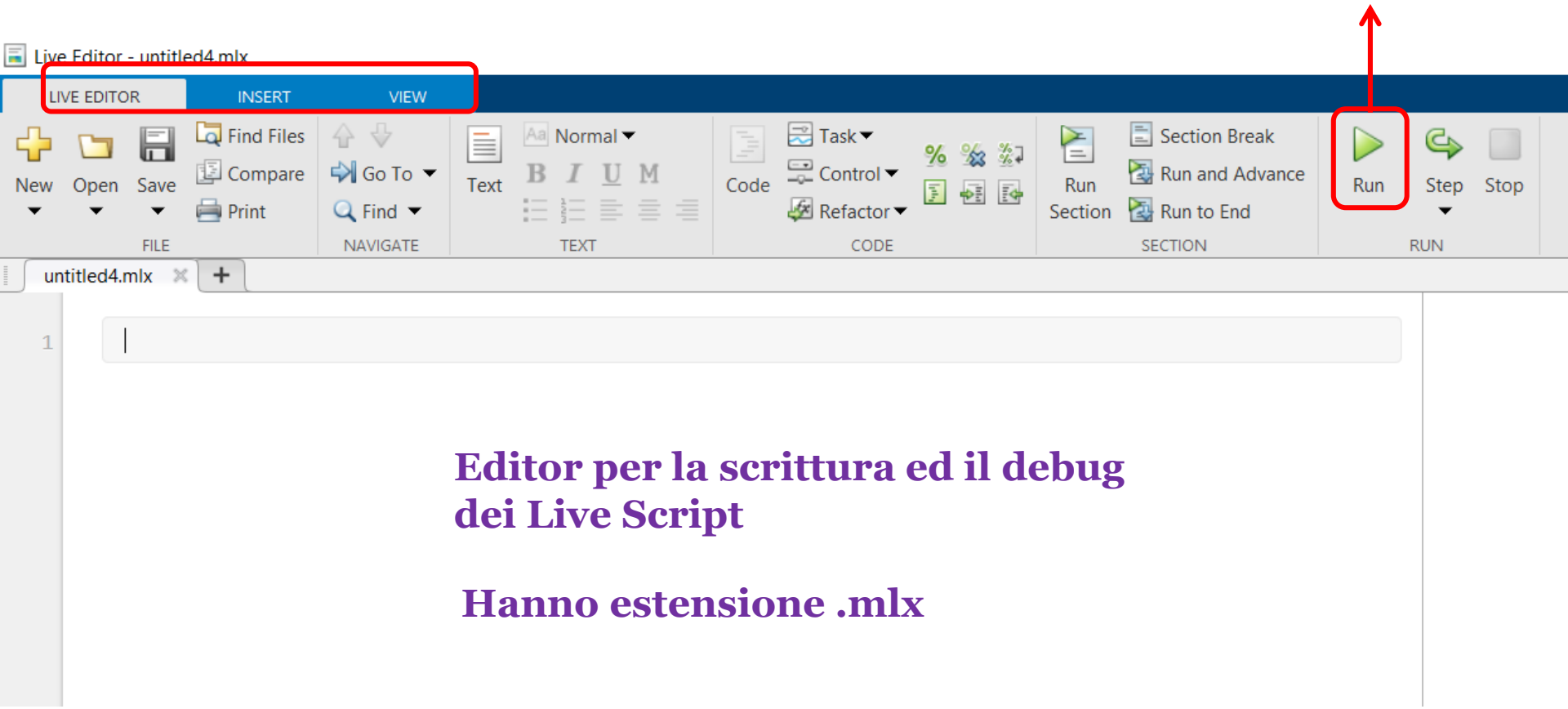


Editor per la scrittura ed il debug degli m-files (script e function files)

Hanno estensione .m

Editor di testo per i live Script

Save and run



Editor per la scrittura ed il debug dei Live Script

Hanno estensione .mlx

I Live Script sono dei files di calcolo che inglobano al proprio interno descrizioni testuali e immagini

Esempio di Live Script

C:\Users\pisan\Documents\MATLAB\Examples\R2020b\matlab\sunspots

Live Editor - C:\Users\pisan\Documents\MATLAB\Examples\R2020b\matlab\sunspots\sunspots.mlx

LIVE EDITOR | INSERT | VIEW

New | Open | Save | Find Files | Compare | Print | Go To | Find | Text | Title | Code | Task | Control | Refactor | Section Break | Run and Advance | Run to End | Run Section | Run | Step | Stop

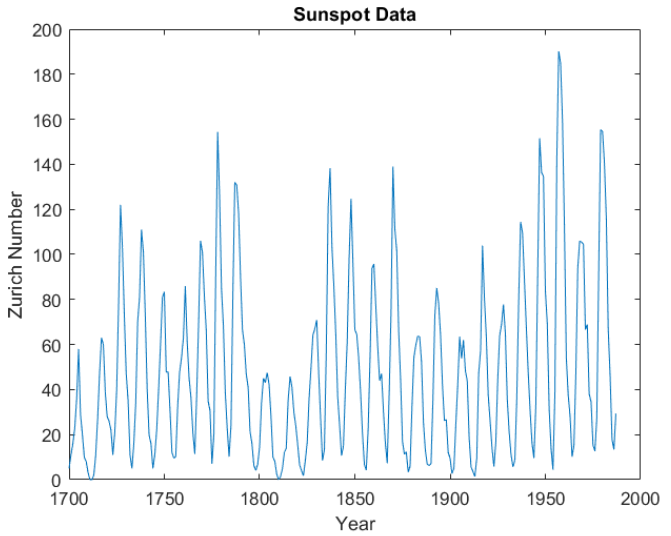
sunspots.mlx

Analyzing Cyclical Data with FFT

You can use the Fourier transform to analyze variations in data, such as an event in nature over a period time.

For almost 300 years, astronomers have tabulated the number and size of sunspots using the Zurich sunspot relative number. Plot the Zurich number over approximately the years 1700 to 2000.

```
load sunspot.dat
year = sunspot(:,1);
relNums = sunspot(:,2);
plot(year,relNums)
xlabel('Year')
ylabel('Zurich Number')
title('Sunspot Data')
```

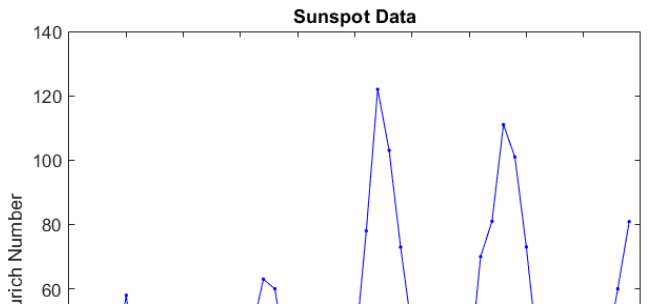


To take a closer look at the cyclical nature of sunspot activity, plot the first 50 years of data.

```
plot(year(1:50),relNums(1:50),'b.-');
xlabel('Year')
ylabel('Zurich Number')
title('Sunspot Data')
```

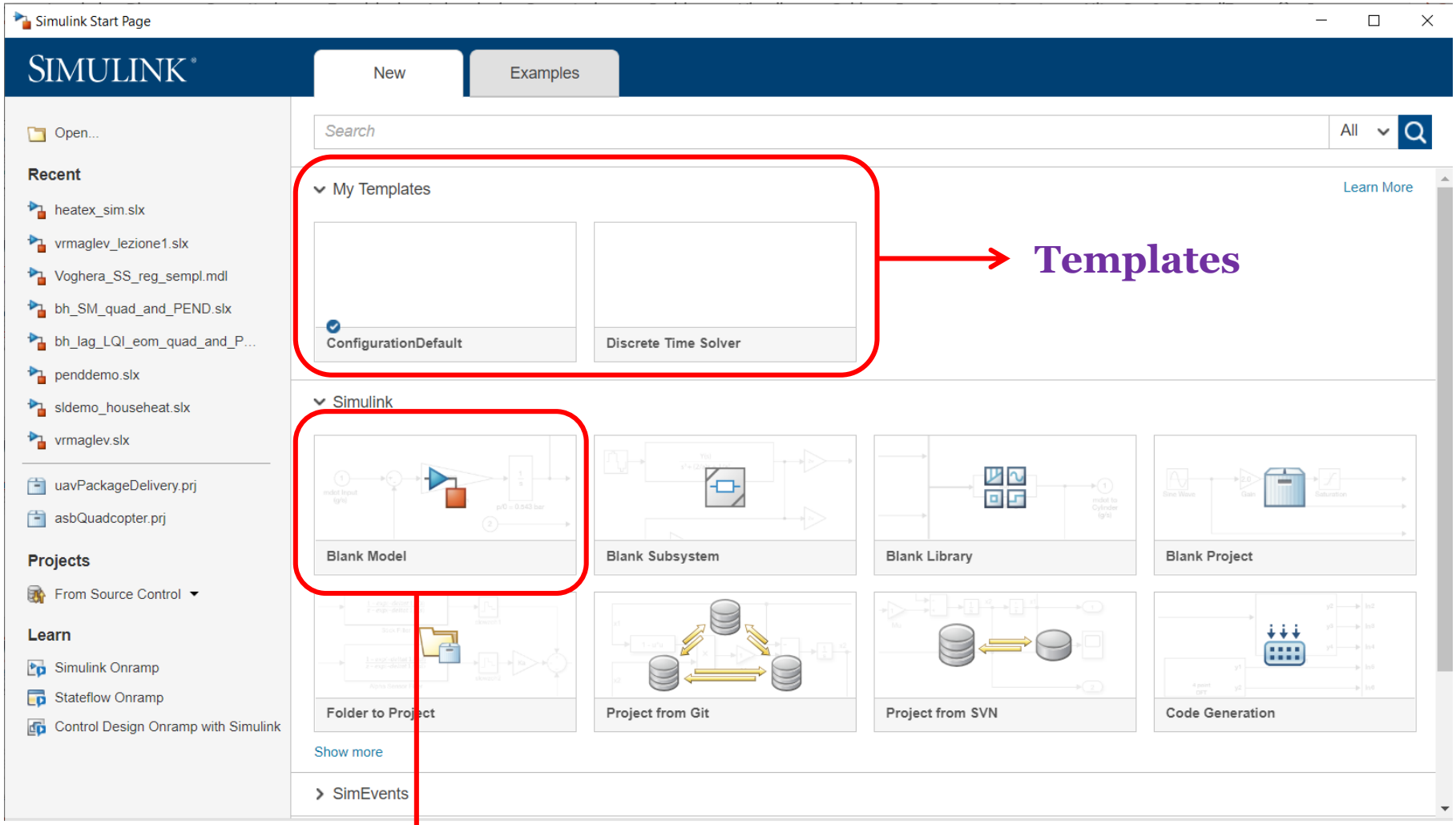
The Fourier transform is a fundamental tool in signal processing that identifies frequency components in data. Using the `fft` function, take the Fourier transform of the Zurich data. Remove the first element of the output, which stores the sum of the data. Plot the remainder of the output, which contains a mirror image of complex Fourier coefficients about the real axis.

```
y = fft(relNums);
y(1) = [];
plot(y,'ro')
xlabel('real(y)')
ylabel('imag(y)')
title('Fourier Coefficients')
```



Fourier coefficients on their own are difficult to interpret. A more meaningful measure of the coefficients is their magnitude

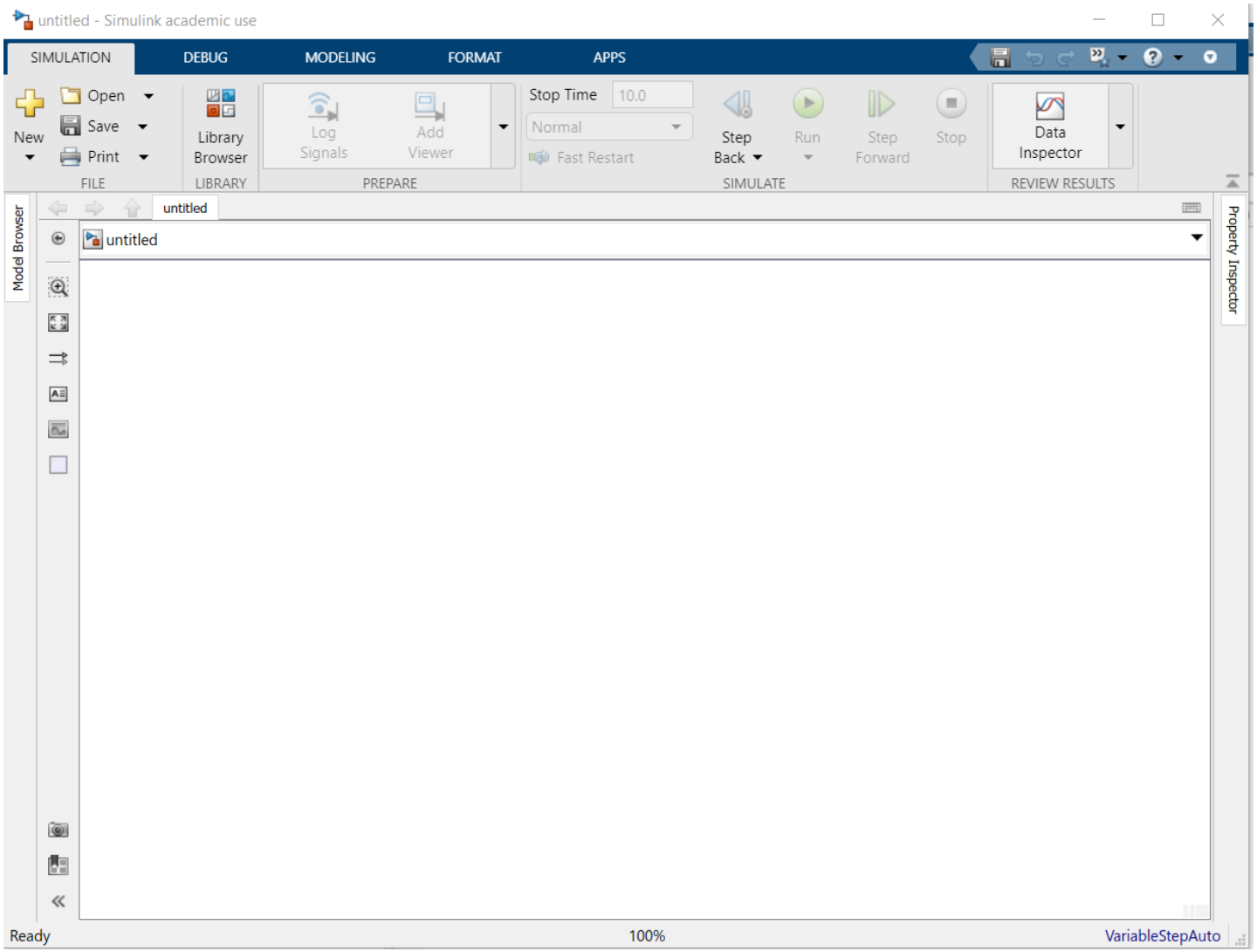
Finestra di avvio SIMULINK



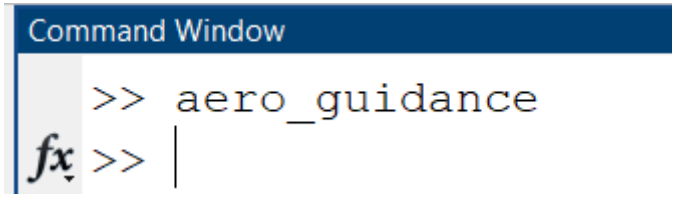
Templates

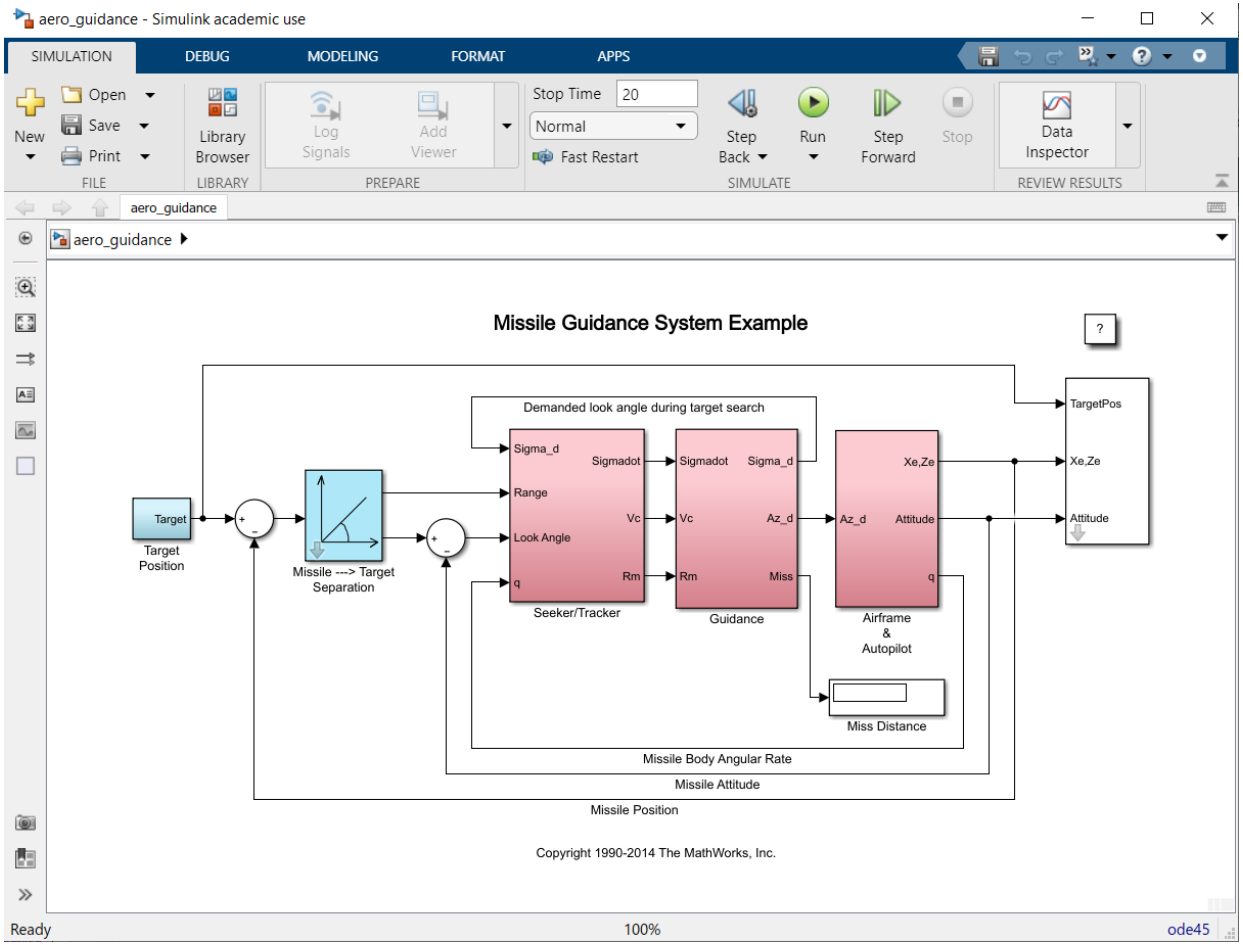
Apre un modello Simulink vuoto avente le proprietà di default

Modello Simulink in bianco

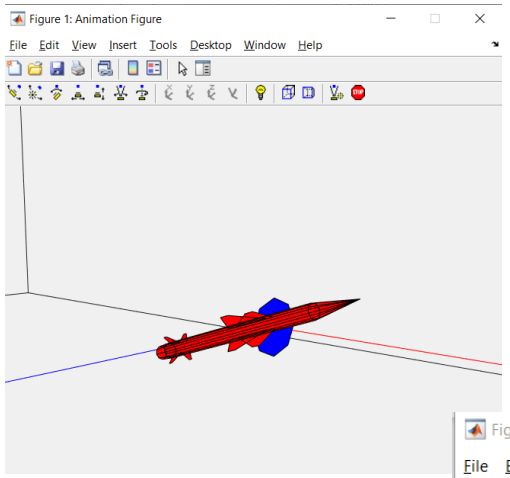


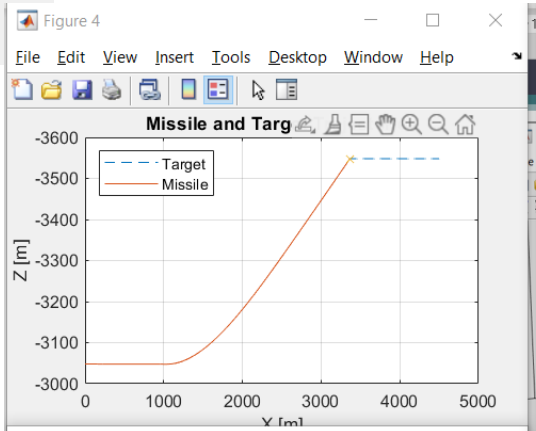
Modello Simulink di esempio con creazione automatica di grafici ed animazione 3D

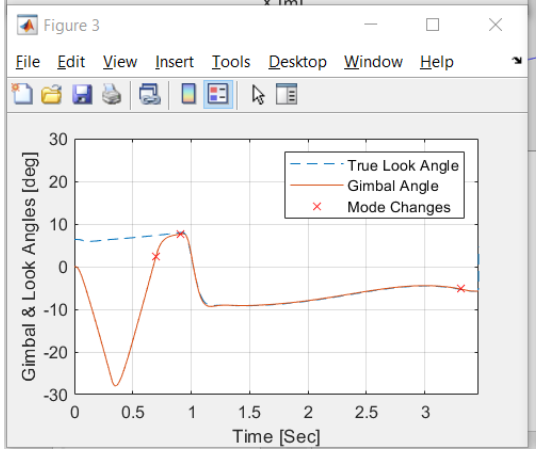


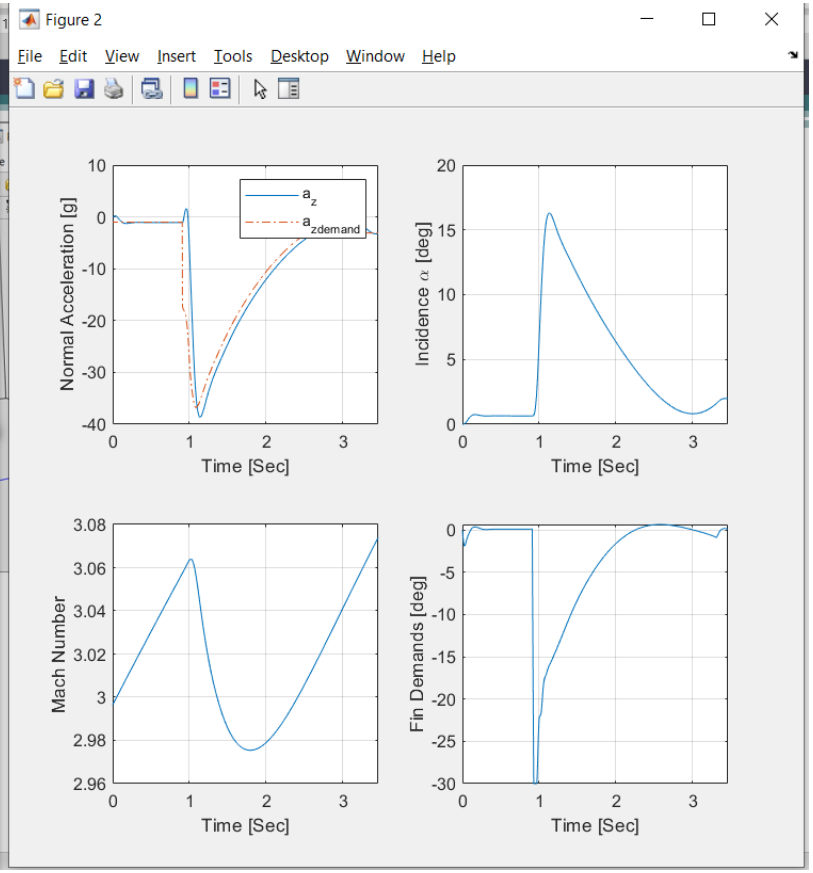


Output del modello

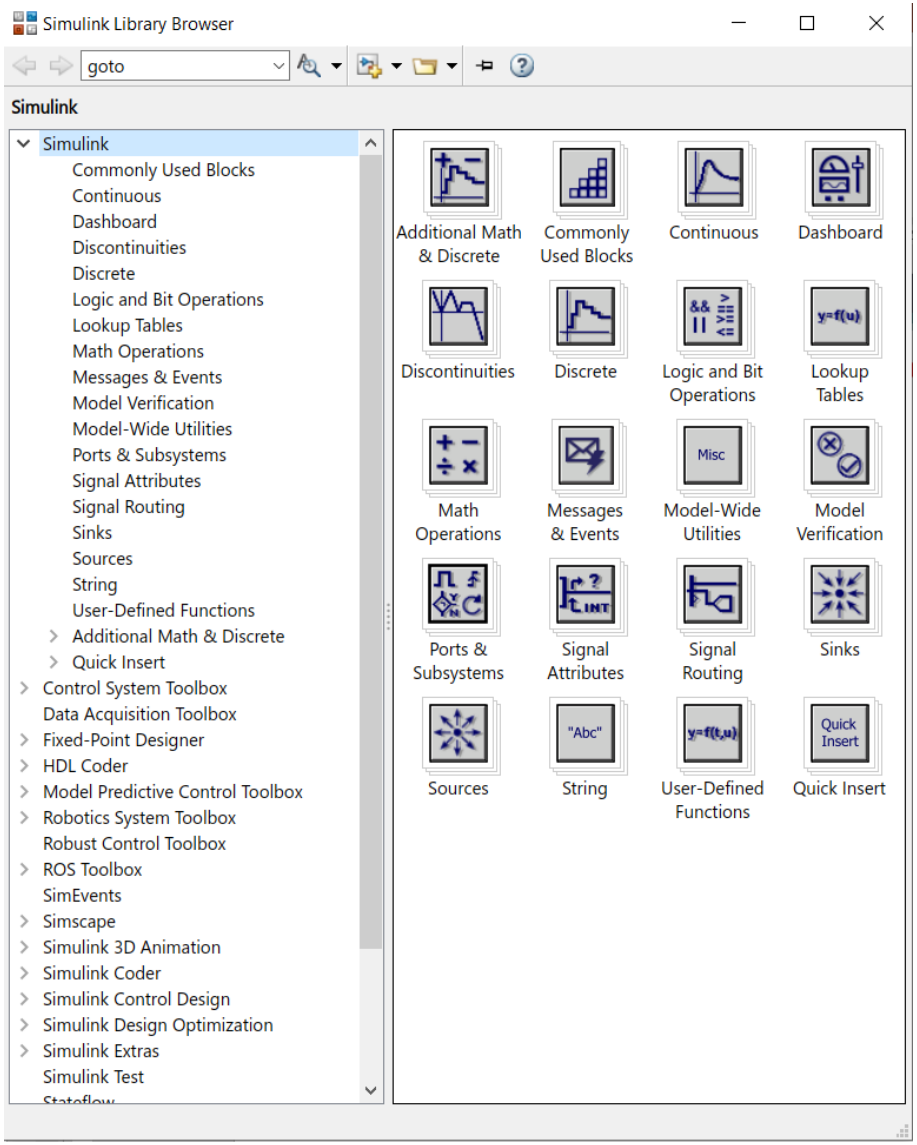




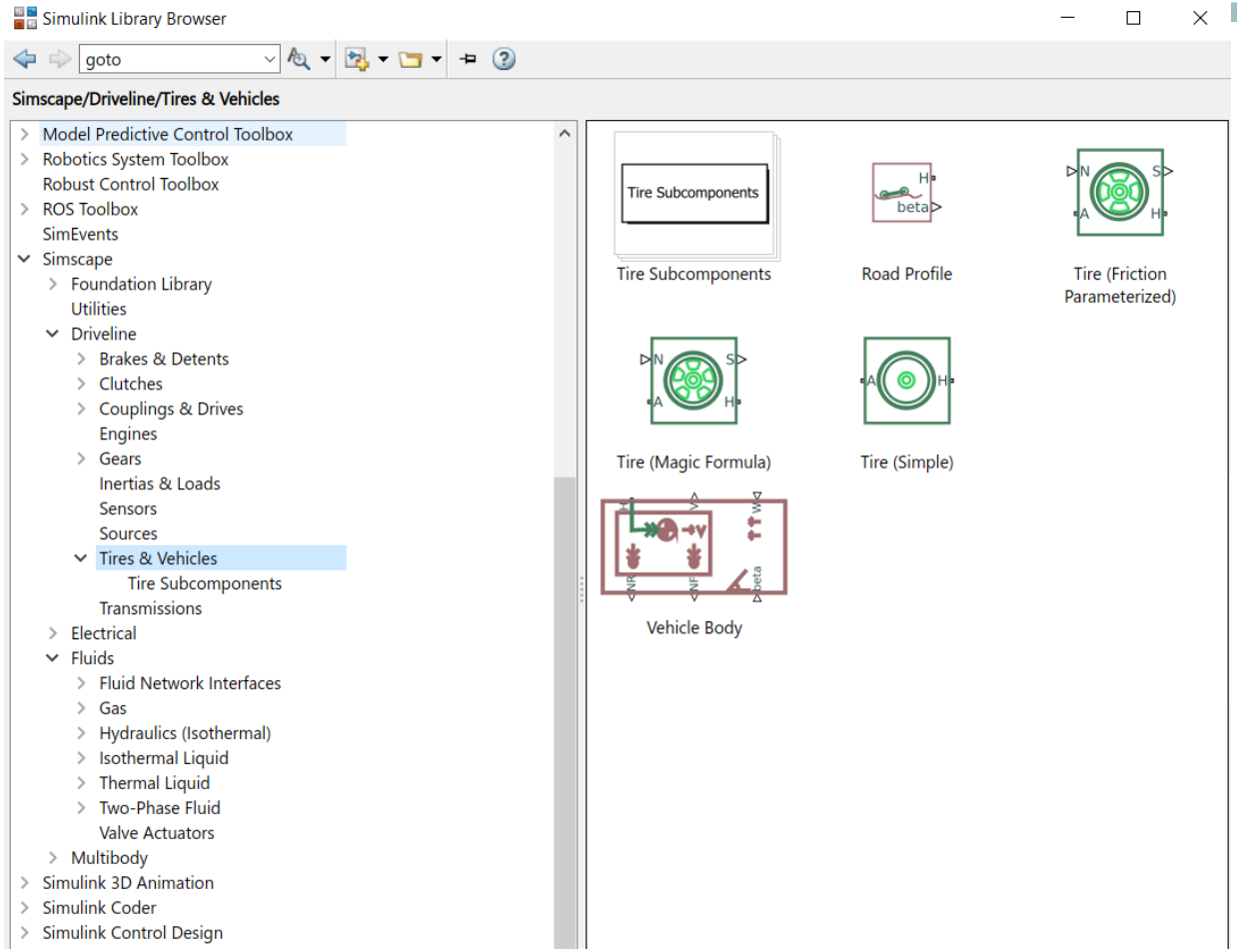




Library browser



Contiene le librerie di blocchi elementari SIMULINK, da quelli di base fino a quelli più sofisticati orientati a particolari applicazioni



Libreria «Tires and vehicles» per la simulazione dinamica di sistemi automotive

Preliminari

```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
>> (log(1.25)*3.45+sin(pi/2))/2

ans =

    0.8849

fx >>
```

Il prompt dei comandi ("command window") può essere usato come una "calcolatrice".

Il risultato di ciascuna operazione viene memorizzato in una variabile temporanea denominata "ans". Tale variabile viene sovrascritta continuamente ad ogni operazione.

Operatori base

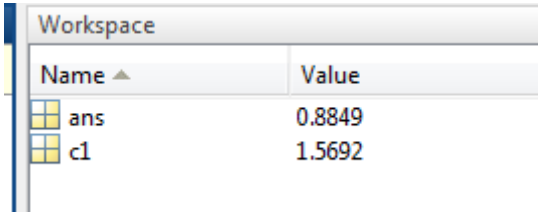
+	-	*	/	^

```
>> c1=sqrt(2)^1.3

c1 =

    1.5692
```

Il risultato delle operazioni può essere **memorizzato in variabili.**

Il contenuto del workspace viene aggiornato con la nuova variabile appena definita

Il **punto e virgola** alla fine di una istruzione disabilita la visualizzazione del risultato.



Funzioni matematiche principali

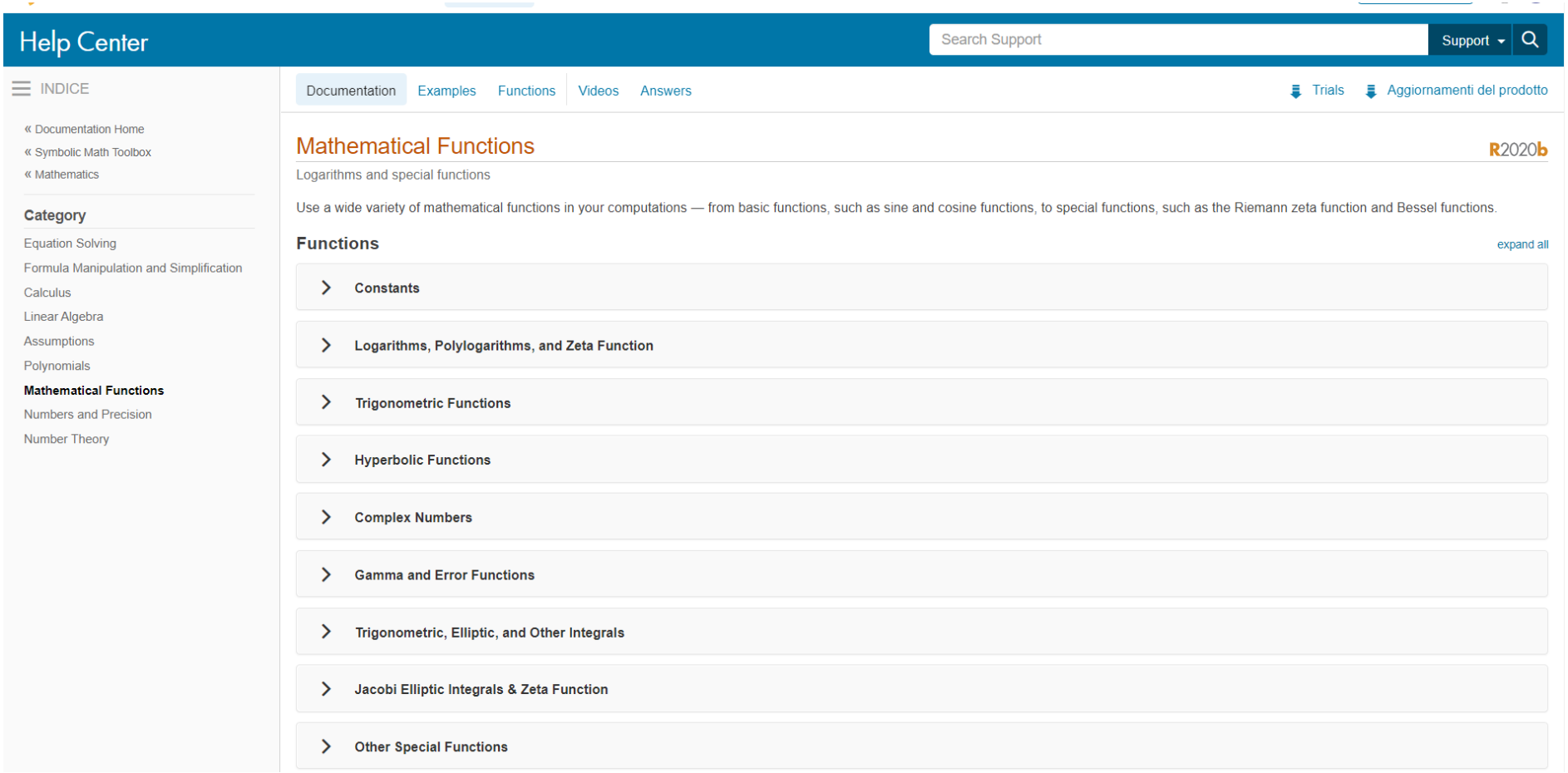
Trigonometric.	
sin	Sine.
sinh	Hyperbolic sine.
asin	Inverse sine.
asinh	Inverse hyperbolic sine.
cos	Cosine.
cosh	Hyperbolic cosine.
acos	Inverse cosine.
acosh	Inverse hyperbolic cosine.
tan	Tangent.
tanh	Hyperbolic tangent.
atan	Inverse tangent.
atan2	Four quadrant inverse tangent.
atanh	Inverse hyperbolic tangent.
sec	Secant.
sech	Hyperbolic secant.
asec	Inverse secant.
asech	Inverse hyperbolic secant.
csc	Cosecant.
csch	Hyperbolic cosecant.
acsc	Inverse cosecant.
acsch	Inverse hyperbolic cosecant.
cot	Cotangent.
coth	Hyperbolic cotangent.
acot	Inverse cotangent.
acoth	Inverse hyperbolic cotangent.

Exponential.	
exp	Exponential.
log	Natural logarithm.
log10	Common logarithm.
sqrt	Square root.

Complex.	
abs	Absolute value.
angle	Phase angle.
conj	Complex conjugate.
imag	Complex imaginary part.
real	Complex real part.

Numeric.	
fix	Round towards zero.
floor	Round towards minus infinity.
ceil	Round towards plus infinity.
round	Round towards nearest integer.
rem	Remainder after division.
sign	Signum function.

https://it.mathworks.com/help/symbolic/mathematical-functions.html



Variabili e costanti speciali

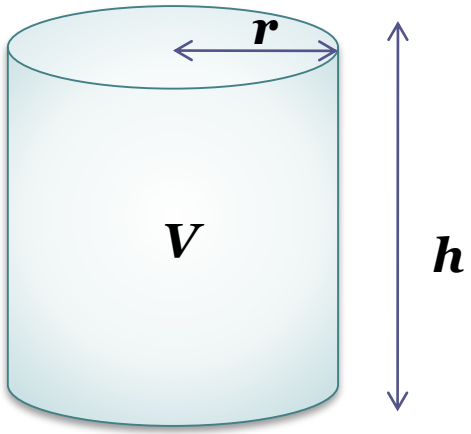
- **ans:** variabile temporanea che contiene il risultato della operazione più recente;
- **eps:** il più piccolo numero reale che addizionato ad 1 crea un numero maggiore di 1 (≅2.2E-16);
- **realmin** il più piccolo numero reale positivo che può essere utilizzato (≅2.2E-308)
- **realmax** il più grande numero reale che può essere utilizzato (≅1.8E308)
- **i, j**: unità immaginaria;
- **Inf**: infinito;
- **NaN**: risultato numerico indefinito; (ad es. 0/0)
- **pi**: indica il numero π .

I nomi delle variabili devono iniziare con un carattere alfabetico, e successivamente si possono avere caratteri alfanumerici o underscores.
La lunghezza del nome di una variabile non deve eccedere 19 caratteri.

Se si attribuisce ad una variabile il nome di una costante speciale (ad es. pi=4) la costante speciale non puo essere piu utilizzata finche la variabile avete il suo stesso nome non venga rimossa dal workspace.

Un semplice esercizio

Un serbatoio di forma cilindrica (serbatoio 1) è alto 15 metri ed ha un raggio di 8 metri. Si desidera realizzare un secondo serbatoio che abbia un volume maggiore del 20% e che abbia la stessa altezza del serbatoio 1. Quale raggio dovrà avere il secondo serbatoio ?

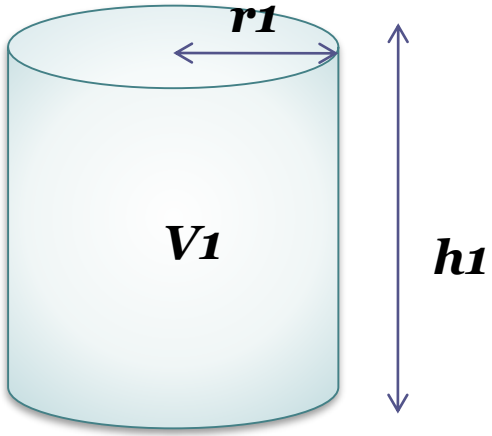


$$V = \pi h r^2$$

$$r = \sqrt{\frac{V}{\pi h}}$$

Un semplice esercizio

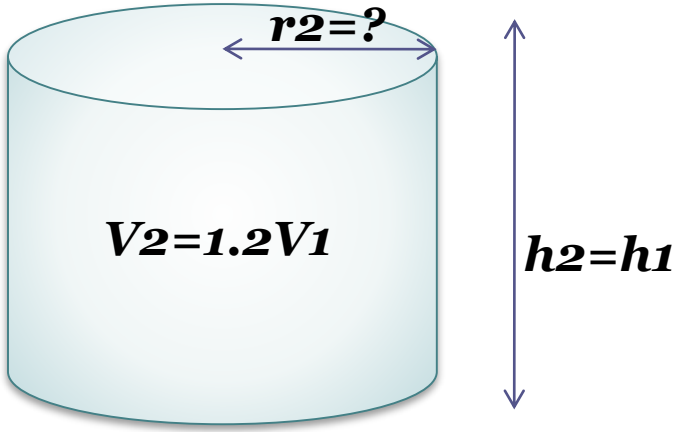
Un serbatoio di forma cilindrica (serbatoio 1) è alto 15 metri ed ha un raggio di 8 metri. Si desidera realizzare un secondo serbatoio che abbia un volume maggiore del 20% e che abbia la stessa altezza del serbatoio 1. Quale raggio dovrà avere il secondo serbatoio ?



ALGORITMO RISOLUTIVO

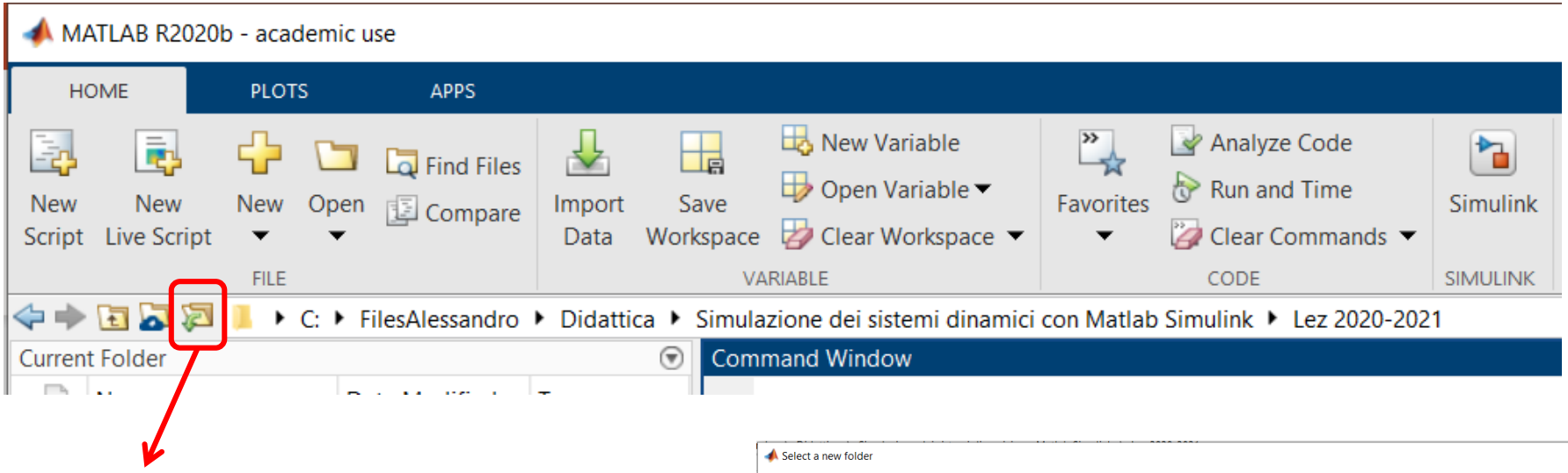
1. Definiamo due variabili, h1 ed r1, che denotano altezza e raggio del serbatoio 1, alle quali assegneremo i valori di 15 ed 8.
2. Calcoliamo il volume V1 del serbatoio 1: $V_1 = \pi h_1 r_1^2$
3. Calcoliamo il volume V2 del serbatoio 2 moltiplicando V1 per 1.2
4. Per mezzo della formula inversa $r_2 = \sqrt{\frac{V_2}{\pi h_1}}$

   calcoliamo il raggio r2 del secondo serbatoio

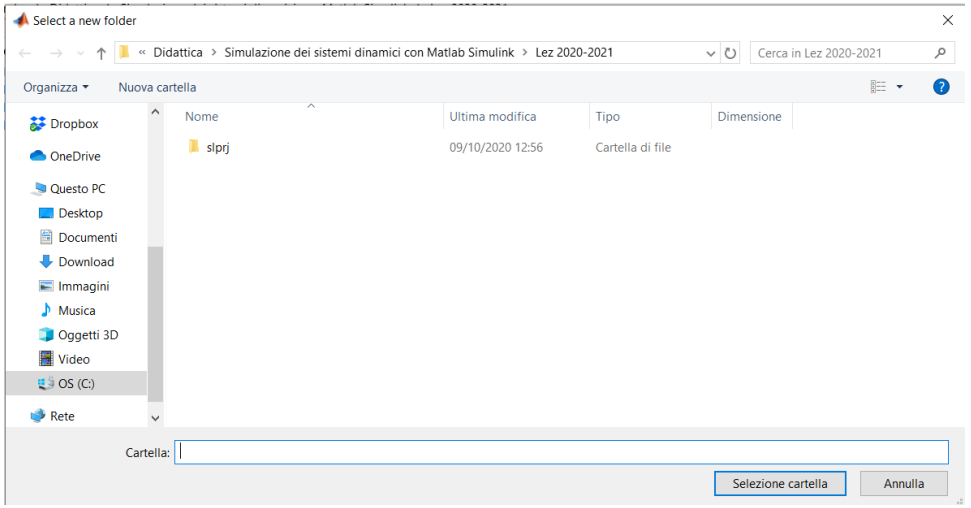


Implementiamo questo algoritmo per mezzo di un file script

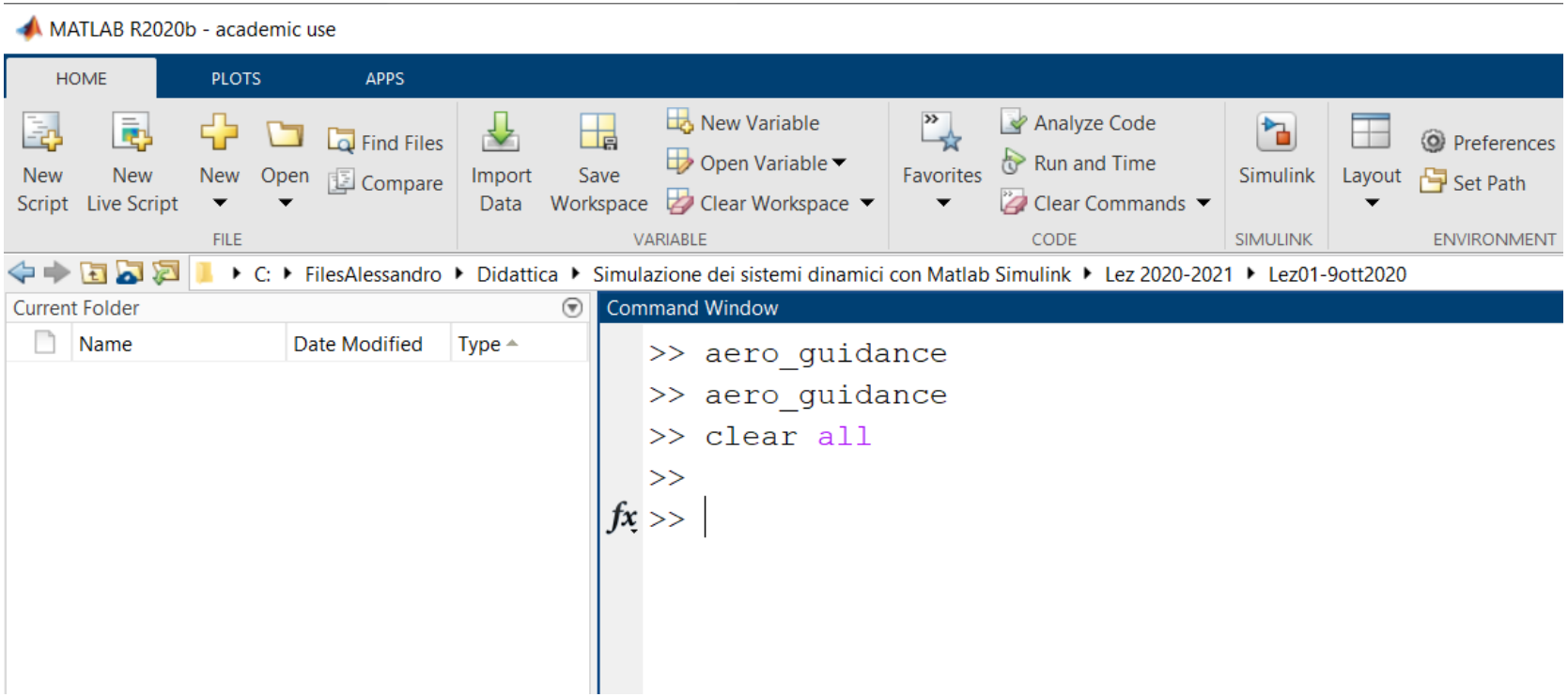
Scegliere una “current folder” nel PC, all’interno della quale verrà salvato il fil.e



Browse for folder



Si apre la finestra “Select a new folder”. Navigando tra le cartelle del proprio PC sceglierne una (o crearla ex-novo) e poi premere “Selezione cartella”

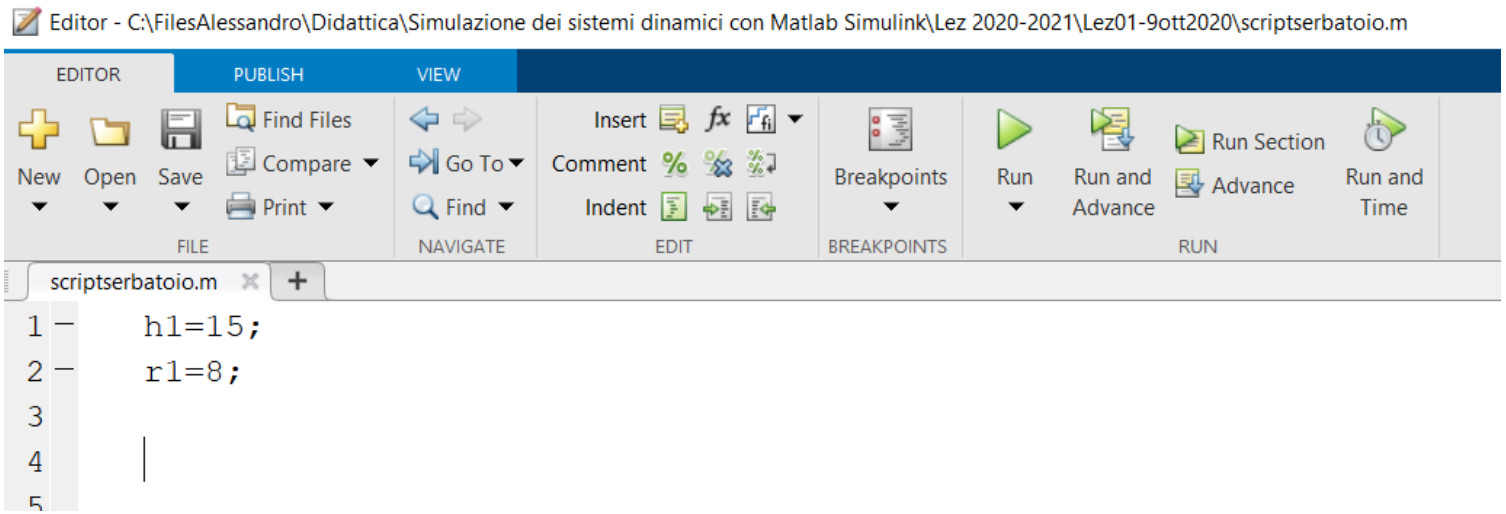


Ho creato, e scelto come «Current Folder», la cartella «Lez01-9ott2020»
Il path compare nella apposita casella.
Nella sottofinestra «Current Folder» non ci sono ovviamente files, in quanto la cartella è stata appena creata

Ora dalla Home premere il pulsante “New script” per aprire il relativo editor di testo

Iniziamo ad inserire le istruzioni necessarie.

1. Definiamo due variabili, h1 ed r1, che denotano altezza e raggio del serbatoio 1, alle quali assegneremo i valori di 15 ed 8.

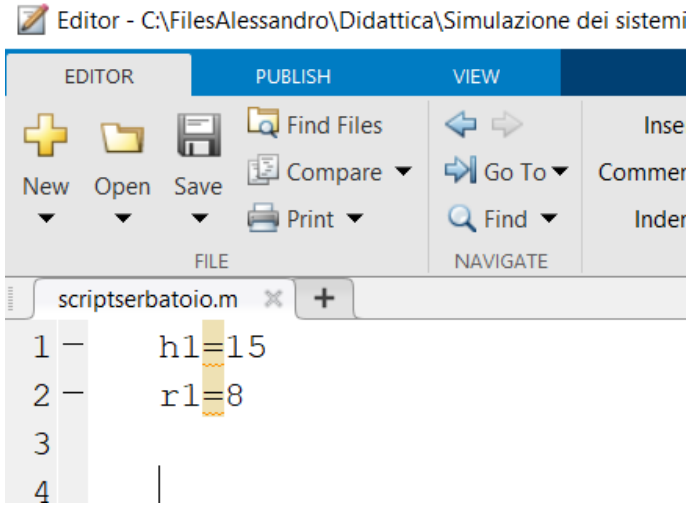


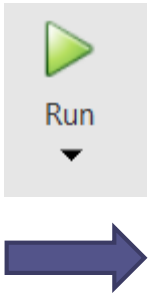
Salviamo il file nella Current Folder (pulsante «Save»).

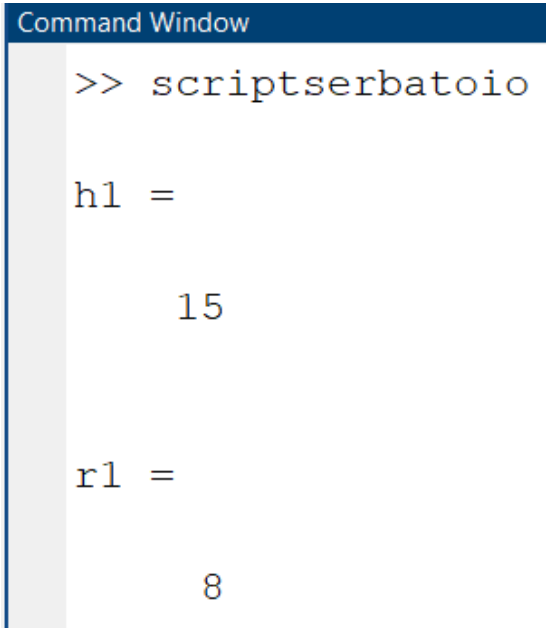
Che succede se mandiamo in run questo script ? Controllare le variabili presenti nel workspace di Matlab prima e dopo il run.

Se viene rimosso il punto e virgola come carattere finale delle istruzioni il risultato dell'operazione viene visualizzato nel prompt successivamente alla esecuzione del file.

Ciò può essere utile in fase di debug, ma in generale è da evitarsi a meno che non si tratti di variabili delle quali desideriamo visualizzare il valore.







2. Calcoliamo il volume V1 del serbatoio 1: $V_1 = \pi h_1 r_1^2$

Calcoliamo il volume V2 del serbatoio 2 moltiplicando V1 per 1.2

scriptserbatoio.m

```
1 —    h1=15;
2 —    r1=8;
3 —    V1=pi*h1*r1^2;
4 —    V2=(1.2)*V1;
5
6
```

3. Per mezzo della formula inversa $r_2 = \sqrt{\frac{V_2}{\pi h_1}}$ calcoliamo il raggio r2 del secondo serbatoio

scriptserbatoio.m

```
1 -    h1=15;
2 -    r1=8;
3 -    V1=pi*h1*r1^2;
4 -    V2=(1.2)*V1;
5      %istruzione corretta
6 -    r2=sqrt(V2/(pi*h1))
7
8      %istruzione con sintassi errata
9 -    r2_errata=sqrt(V2/pi*h1)
0
1
```

$$r_2 = \sqrt{\frac{V_2}{\pi\, h_1}}$$

$$r_2 = \sqrt{\frac{V_2}{\pi} h_1}$$

I commenti si inseriscono con il simbolo %

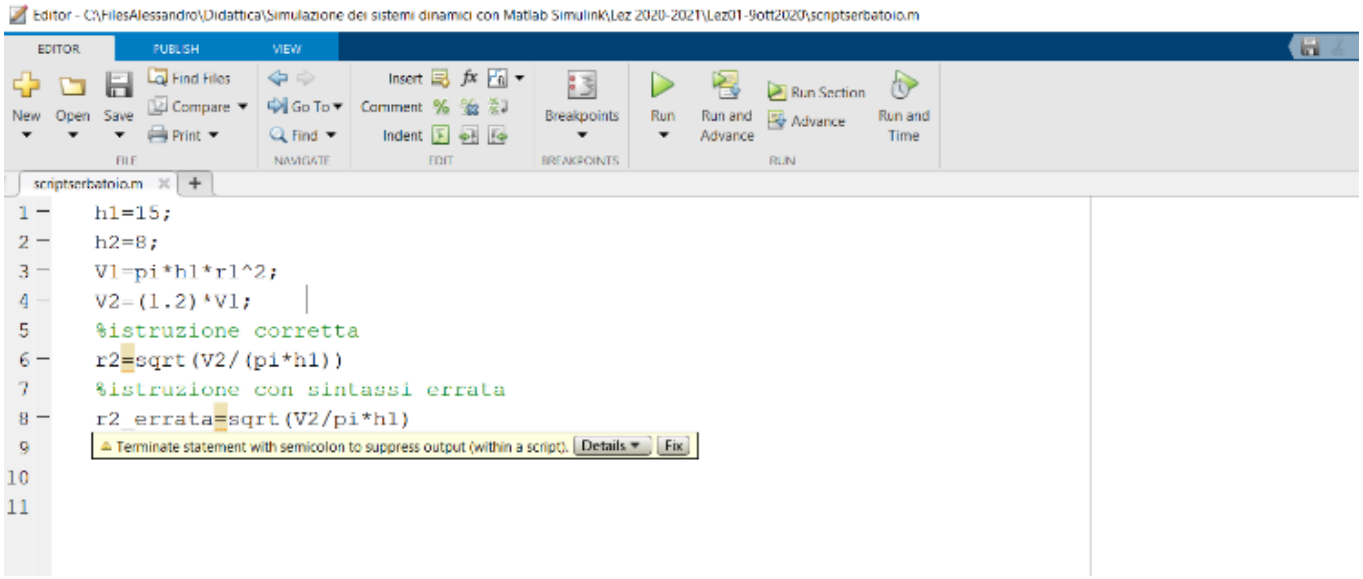
Output

Command Window

```
>> scriptserbatoio

r2 =

    8.7636


r2_errata =

  131.4534

>>
```

L'editor di testo fornisce potenti funzioni di debug.

Si noti come i segni di "=" nelle ultime due istruzioni dello script siano evidenziati in colore giallo.
Si provi a portare il mouse sopra uno dei simboli evidenziati



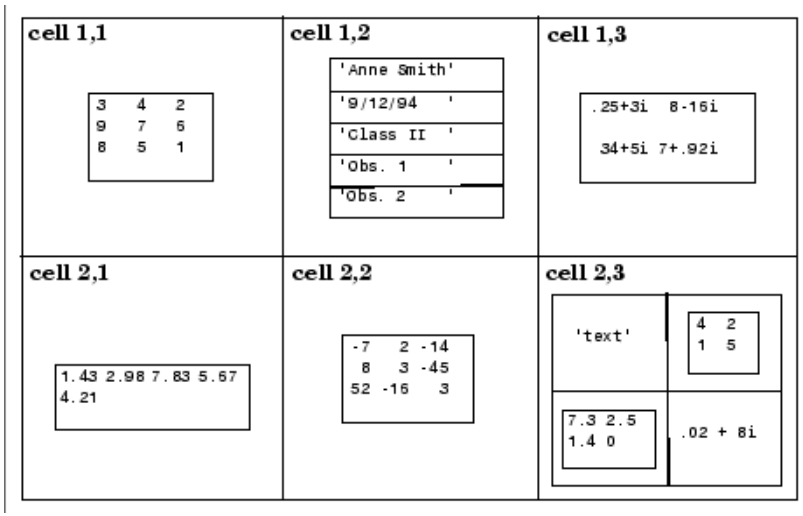
E un warning con cui Matlab ci "avvisa" del fatto che non è stata inserito il punto e virgola alla fine della istruzione, cosa che talvolta si dimentica e che può produrre indesiderati output lunghissimi nel prompt.

Tipi di variabili

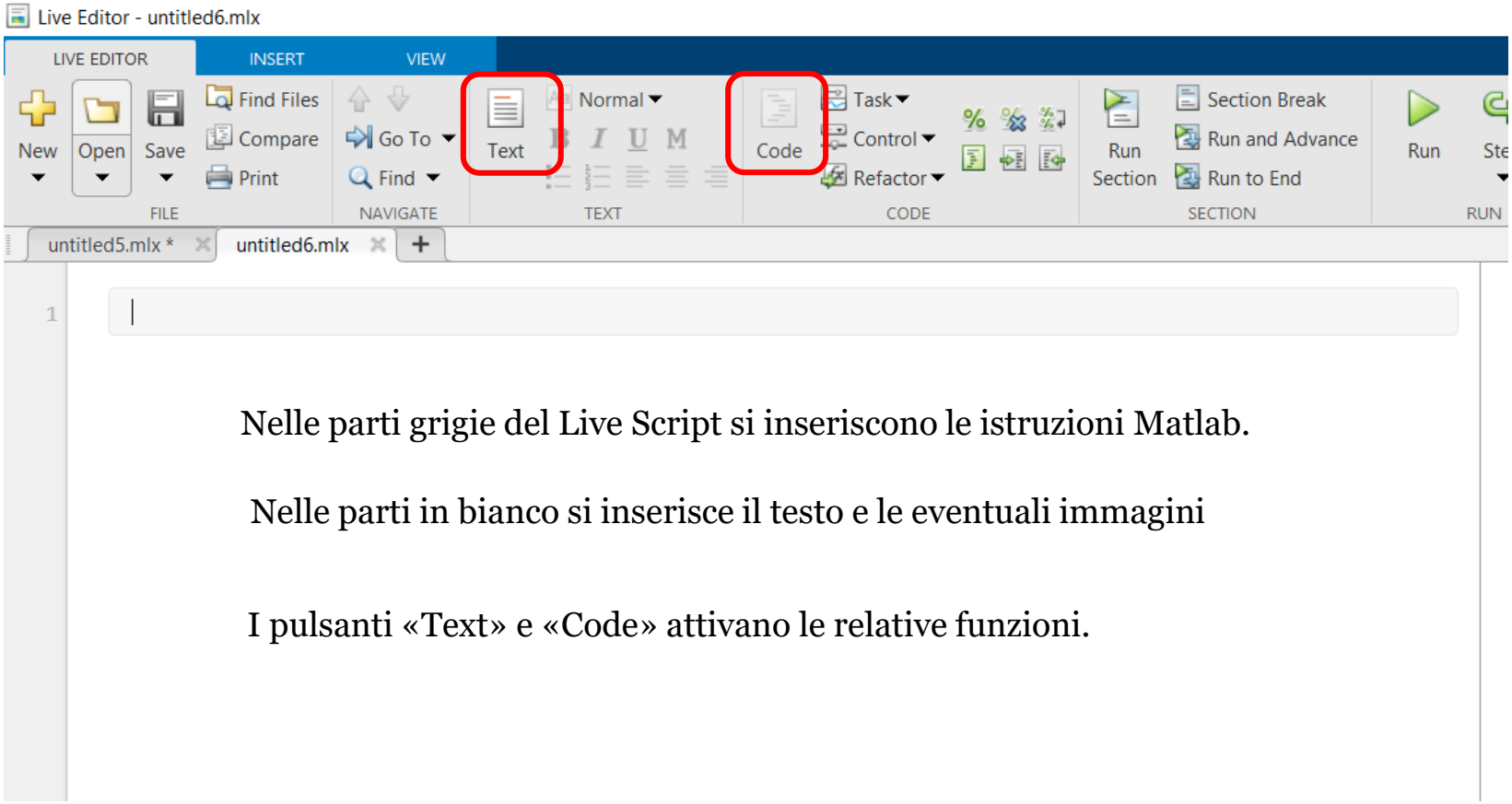
Il tipo di dato più importante in Matlab è il tipo **Array**.
Il tipo di Array più utilizzato è in particolare **l'Array bidimensionale** (n righe ed m colonne). Esso include gli scalari (1x1), i vettori riga e colonna (1xn , nx1), e le matrici quadrate e rettangolari.

In moltissimi problemi di simulazione e calcolo scientifico il tipo di dato Array bidimensionale per tutte le variabili è sufficiente a coprire le esigenze del problema. Da qui a breve studieremo nel dettaglio la creazione e la manipolazione di Array bidimensionali.

Sono disponibili anche altri tipi di dati, ad esempio gli array multidimensionali, il tipo `struct`, analogo al tipo di dato struct del linguaggio C (una struttura ripetitiva con dei sottocampi eterogenei), o il tipo `cell array` (v. figura)



Onde impratichirci con le istruzioni di costruzione di vettori e matrici, realizziamo un Live Script



Nelle parti grigie del Live Script si inseriscono le istruzioni Matlab.

Nelle parti in bianco si inserisce il testo e le eventuali immagini

I pulsanti «Text» e «Code» attivano le relative funzioni.

Creazione di vettori

CreaVettori.mlx

Sintassi per la creazione di vettori

Creazione di un vettore riga: gli elementi del vettore si inseriscono fra parentesi quadre, separati da spazi.

```
v=[1 2 3 4 5]
```

```
v = 1×5
     1     2     3     4     5
```

Gli elementi del vettore possono anche essere separati da virgole. Una sintassi equivalente è la seguente:

```
v=[1,2,3,4,5]
```

```
v = 1×5
     1     2     3     4     5
```

Creazione di un vettore colonna.

```
v=[1;2;3;4;5]
```

```
v = 5×1
     1
     2
     3
     4
     5
```

Sintassi alternativa

```
v=[1 2 3 4 5]'
```

```
v = 5×1
     1
     2
     3
     4
     5
```

Creazione di vettori utilizzando variabili simboliche

```
a=1;
x=[ a 1+a  5 4-a]
```

```
x = 1×4
     1     2     5     3
```

Esempio che mostra come le variabili Matlab siano case-sensitive

```
a=2;
A=5;
z=[a A]
```

```
z = 1×2
     2     5
```

CreaVettori.mlx

Creazione di matrici

CreaMatrici.mlx

Sintassi per la creazione di matrici

Creazione di una matrice: gli elementi si inseriscono fra parentesi quadre, separati da spazi o da virgole. Il punto e virgola rappresenta il passaggio alla riga successiva della matrice.

Es. Creazione di una matrice 3x3:

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

Sintassi alternativa per la creazione della medesima matrice 3x3

```
2  A=[1,2,3;4,5,6;7,8,9]
```

Es: creazione di una matrice 2x3

```
3  B=[1 2 3;4 5 6]
```

Una matrice può essere creata anche "accorpando" più matrici che devono avere ugual numero di righe.

Per creare ad esempio la matrice:

$$M = \begin{bmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 5 & 6 & 4 & 5 & 6 \\ 7 & 8 & 9 & 7 & 8 & 9 \end{bmatrix}$$

che vede "affiancate" due matrici 3x3 identiche si può utilizzare la seguente sintassi

```
4  M=[A A]
```

L'"accorpamento" puo essere fatto anche in senso verticale. La matrice $M=\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ può ad esempio essere generata impiegando la seguente sintassi:

```
5  M=[A;A]
```

```
A = 3x3
     1     2     3
     4     5     6
     7     8     9

A = 3x3
     1     2     3
     4     5     6
     7     8     9

B = 2x3
     1     2     3
     4     5     6

M = 3x6
     1     2     3     1     2     3
     4     5     6     4     5     6
     7     8     9     7     8     9

M = 6x3
     1     2     3
     4     5     6
     7     8     9
     1     2     3
     4     5     6
     7     8     9
```

CreaMatrici.mlx

Comandi per gestire una sessione di lavoro con Matlab

clc	cancella il contenuto della finestra dei comandi;
clear all	elimina tutte le variabili dal workspace
clear v1 v2	elimina le variabili v1 e v2 dal workspace
save 15ott2020	salva tutte le variabili del workspace in un file binario 15ott2020.mat
load 15ott2020	carica nel workspace tutte le variabili salvate nel un file binario 15ott2020.mat
...	l'istruzione continua nella riga successiva.

1.2e-4	notazione esponenziale per il numero $1.2\ 10^{-4}$

Precedenze

Le parentesi tonde prevalgono su tutti gli operatori.

L'elevazione a potenza (^) ha precedenza superiore al prodotto (*) ed alla divisione (/)

Istruzioni speciali per la creazione di matrici e vettori

istrspecialivettmatrici.mlx

Istruzioni speciali per la creazione di matrici e vettori

Crezione di vettori ordinati con incremento unitario.

La seguente sintassi crea un vettore riga il cui primo elemento è pari a 2, l'ultimo elemento è pari a 7, e gli elementi intermedi hanno incremento unitario

```
1  x=2:7
```

```
x = 1×6
     2     3     4     5     6     7
```

Crezione di vettori ordinati con incremento arbitrario e elementi iniziali e finali prefissati

La sintassi generale è

x=elemento_iniziale:incremento:elemento_finale

La seguente sintassi crea un vettore riga il cui primo elemento è nullo, l'ultimo elemento è pari a 0.5, e gli elementi intermedi hanno incremento pari a 0.1

```
2  x=0:0.1:0.5
```

```
x = 1×6
         0    0.1000    0.2000    0.3000    0.4000    0.5000
```

Si faccia attenzione al fatto che l'ultimo elemento del vettore potrebbe non coincidere con "elemento_finale" se l'incremento non è scelto correttamente. Il seguente esempio lo evidenzia

```
3  x=0:0.15:0.5
```

```
x = 1×4
         0    0.1500    0.3000    0.4500
```

Istruzioni speciali per la creazione di matrici e vettori

Una sintassi alternativa per la creazione di vettori equispaziati prevede l'impiego della funzione

```
linspace(elemento_iniziale,elemento_finale,dimensione_vettore)
```

Si analizzi il seguente esempio di codice

```
x=linspace(1,2,7)
```

che crea un vettore il cui primo elemento è pari ad 1, l'ultimo elemento è pari a 2, e la spaziatura viene determinata automaticamente in modo che il vettore sia composto da 7 elementi. Se alla funzione linspace vengono passati solo due argomenti di ingresso

```
linspace(x1,x2)
```

viene generato un vettore equispaziato di 100 elementi il cui primo elemento è x1 ed il cui ultimo elemento è x2.

```
x=linspace(1,2)
```

La funzione logspace(a,b,n) crea un array di elementi intervallati logaritmicamente, dove n é il numero dei punti fra 10^a e 10^b. Se n viene omesso, vengono generati 50 elementi

```
x=logspace(-1,1,7)
```

```
x = 1×7
    1.0000    1.1667    1.3333    1.5000    1.6667    1.8333    2.0000

x = 1×100
    1.0000    1.0101    1.0202    1.0303    1.0404    1.0505    1.0606 ..

x = 1×7
    0.1000    0.2154    0.4642    1.0000    2.1544    4.6416   10.0000
```

Prodotto scalare e prodotto vettoriale

```
x1=[1 2 3];x2=[4 5 6];

xscalprod=x1*x2'

xscalprod =
    32

xvecprod=cross(x1,x2)

xvecprod =
    -3     6    -3
```

Prodotto scalare

Prodotto vettoriale: funzione cross

```
A1=[1 2 3;4 5 6;7 8 9]; x1=[1 2 3];
A1*x1
??? Error using ==> mtimes
Inner matrix dimensions must agree.

A1*x1'

ans =
    14
    32
    50
```

Il prodotto fra matrici deve rispettare i vincoli dimensionali sul numero di righe e di colonne degli operandi

Altre istruzioni per la creazione di matrici

size(A)	estrae la dimensione della matrice ***A***
length(b)	estrae la lunghezza del vettore riga o colonna ***b***
eye(n)	crea una matrice identità (***n*****n***n***)
eye(size(A))	crea una matrice identità con la stessa dimensione della matrice A
ones(n)	crea una matrice (***n*****n***n***) i cui elementi sono pari a 1
ones(m,n)	crea una array (***m*****n***n***) i cui elementi sono pari a 1
ones(size(A))	crea una matrice di elementi pari a 1 avente la stessa dimensione della matrice A
zeros(n)	crea una matrice quadrata (***n*****n***n***)i cui elementi sono pari a 0
zeros (m,n)	crea una array (***m*****n***n***)i cui elementi sono pari a 0
zeros(size(A))	crea una matrice di elementi pari a 0 avente la stessa dimensione della matrice A
rand(n)	crea una matrice (n*n) di elementi casuali compresi tra [0,1]
rand (m,n)	crea una matrice (m*n) di elementi casuali compresi tra [0,1]

Esempi

```
>> eye(3)

ans =

     1     0     0
     0     1     0
     0     0     1

>> zeros(2,3)

ans =

     0     0     0
     0     0     0
>> rand(3)

ans =

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

```
>> size(A)

ans =

     3     3
```

Accesso agli array

V(i) *identifica l'elemento* ***i****-esimo del vettore* ***V***

A(n,m) *identifica l'elemento che occupa la riga* ***n*** *e la colonna* ***m*** *nella matrice* ***A***

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

A =

     1     2     3
     4     5     6
     7     8     9

>> A(3,3)=20

A =

     1     2     3
     4     5     6
     7     8    20
```

Ridefinizione di un singolo elemento della matrice

Accesso agli array

vettori:

- ***v(n1:n2)*** *rappresenta tutti gli elementi del vettore con indice compreso tra n1 ed n2*
- ***v(n2:end)*** *rappresenta tutti gli elementi del vettore con indice superiore o uguale ad n2*

matrici:

- ***M(:,n)*** *identifica tutti gli elementi della n-esima colonna della matrice **M***
- ***M(:,end)*** *identifica tutti gli elementi della ultima colonna della matrice M*
- ***M(:,n1:n2)*** *identifica tutti gli elementi con indice di colonna compreso tra n1 ed n2*
- ***M(3:5,1:6)*** *identifica tutti gli elementi con indice di riga compreso tra 3 ed 5 e con indice di colonna compreso tra 1 ed 6*

Se si assegna un valore ad una componente di un vettore con indice più elevato della dimensione del vettore (ad esempio, se si assegna un valore al quinto elemento x(5) del vettore bidimensionale x=[1 2]) la dimensione del vettore viene corrispondentemente aumentata e tutti i nuovi valori aggiunti, eccetto l'ultimo x(5), sono posti di default pari a zero.

"end" rappresenta l'indice dell'ultima componente del vettore

```
M=rand(3)
```

```
M = 3×3
    0.7922    0.0357    0.6787
    0.9595    0.8491    0.7577
    0.6557    0.9340    0.7431
```

```
M(:,3)
```

Estrae dalla matrice la terza colonna

```
ans = 3×1
    0.6787
    0.7577
    0.7431
```

```
M(1:2,:)
```

Estrae dalla matrice le prime due righe

```
ans = 2×3
    0.7922    0.0357    0.6787
    0.9595    0.8491    0.7577
```

Funzioni speciali per matrici

det(A)	Calcola il determinante della matrice quadrata A
inv(A)	Calcola l'inversa della matrice quadrata A
rank(A)	Calcola il rango della matrice A
pinv(A)	Calcola la pseudoinversa della matrice A
eig(A)	Calcola gli autovalori della matrice A

```
>> A=[1 2 3;4 5 6;7 8 10];B=[2 4 6;1 1 3];
>> det(A)

ans =
    -3

>> inv(A)

ans =
   -0.6667   -1.3333    1.0000
   -0.6667    3.6667   -2.0000
    1.0000   -2.0000    1.0000
```

```
>> det(B)
??? Error using ==> det
Matrix must be square.

>> inv(B)
??? Error using ==> inv
Matrix must be square.

>> eig(A)

ans =

   16.7075
   -0.9057
    0.1982
```

Si supponga che gli autovalori della matrice A siano da utilizzarsi all'interno del codice per successive analisi.

L'istruzione eig(A) non crea nessuna variabile che immagazzini il valore degli autovalori.

E' opportuno, in tale situazione, creare una variabile che vada a contenere gli autovalori della matrice A

```
clear all
Clc

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

autovA=eig(A);

autovA(1)
autovA(2)
autovA(3)
```

Calcolo degli autovalori.

Accesso agli elementi del vettore `autovA`

Funzioni di vettori

Le funzioni Matlab scalari (es. sin, sqrt, abs, sign) sono tali che se ricevono in ingresso un vettore restituiscono un vettore di pari dimensione che contiene il risultato della operazione applicata via via a ciascun elemento:

```
>> x1=[1 2 -3 4 5];
>> abs(x1)

ans =

     1     2     3     4     5

>> x2=[1 2 3 4 5]; sqrt(x2)

ans =

    1.0000    1.4142    1.7321    2.0000    2.2361

>> sin(x2)

ans =

    0.8415    0.9093    0.1411   -0.7568   -0.9589
```

```
>> length(x)

ans =

     5
```

Funzioni di vettori

Alcuni operatori non si prestano ad essere applicati a vettori.
Ad esempio l'elevazione al quadrato.

```
>> x=[1 2 3 4 5];
>> x^2
??? Error using ==> mpower
Matrix must be square.

>>
```

Per applicare a tutti gli elementi x_i di un vettore, ad esempio, l'operazione $sqrt(x_i)/x_i$ incontriamo simili problemi. Il rapporto tra vettori non è una operazione consentita (non viene interpretata in Matlab come il rapporto tra le relative componenti, ma origina un messaggio di errore)

Per realizzare tali funzionalità si utilizzano gli **operatori elemento per elemento**

Operazioni elemento per elemento

+	*somma fra array A e scalare b*	***A+b***
-	*sottrazione array A e scalare b*	***A-b***
+	*somma di array*	***A+B***
-	*sottrazione fra array*	***A-B***
.^	*elevazione a potenza tra array A e scalare b*	***A.^b***
.*	*moltiplicazione tra array (dimensioni omologhe)*	***A.*B***
./	*divisione tra array (dimensioni omologhe)*	***A./B***
.^	*elevazione a potenza tra array (dimensioni omologhe)*	***A.^B***

*Le operazioni **elemento per elemento** vengono svolte tra gli elementi che occupano posizioni omologhe.*

Il prodotto standard è consentito solo tra vettori o matrici dimensionalmente coerenti, cioè un vettore riga per un vettore colonna, e definisce il prodotto scalare o il prodotto matriciale standard

```
v1=[1 2 3 4];
v2=[3 4 5 6];
v1.*v2
```

```
ans = 1×4
     3     8    15    24
```

```
v1*v2
```

```
Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the number of rows in the second matrix. To perform
elementwise multiplication, use '.*'.

Related documentation
```

```
v1*v2'
```

```
ans = 50
```

```
v1.^2
```

```
ans = 1×4
     1     4     9    16
```

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

```
M = 3×3
     1     2     3
     4     5     6
     7     8     9
```

```
M+10
```

```
ans = 3×3
    11    12    13
    14    15    16
    17    18    19
```

```
M.^2
```

```
ans = 3×3
     1     4     9
    16    25    36
    49    64    81
```

```
M^2
```

```
ans = 3×3
    30    36    42
    66    81    96
   102   126   150
```

Operazioni elemento per elemento

Es. a partire da un vettore generico x=[x1, x2, …, xn], determinare il vettore z la cui generica componente z_i ha la seguente espressione

$$z_i = \frac{\sqrt{|x_i|}}{x_i}$$

```
x=1:10;
z=sqrt(abs(x))./x
```

```
z =

  Columns 1 through 6

    1.0000    0.7071    0.5774    0.5000    0.4472    0.4082

  Columns 7 through 10

    0.3780    0.3536    0.3333    0.3162
```

NUMERI COMPLESSI

abs(x)	*calcola il valore assoluto di x se x è reale, ed il suo modulo se x è un numero complesso*
angle(x)	*calcola la fase di un numero complesso*
conj(x)	*calcola il numero complesso coniugato di x*
imag(x)	*restituisce la parte immaginaria di un numero complesso x*
real(x)	*restituisce la parte reale di un numero complesso x*

Esempi

```
>> z1=1+i

z1 =

   1.0000 + 1.0000i

>> abs(z1)

ans =

    1.4142

>> angle(z1)

ans =

    0.7854
```

```
>> z1c=conj(z1)

z1c =

   1.0000 - 1.0000i

>> imag(z1)

ans =

     1

>> real(z1)

ans =

     1
```

Prodotto e divisione

```
>> z1=1+i;  z2=2-3i;
>> z1*z2

ans =

   5.0000 - 1.0000i

>> z1/z2

ans =

  -0.0769 + 0.3846i
```

APPROSSIMAZIONI

ceil(x) approssima x al numero intero più vicino verso infinito
fix(x) approssima x al numero intero più vicino verso lo zero
floor(x) approssima x al numero intero più vicino verso - infinito
round(x) approssima x al numero intero più vicino
sign(x) calcola il segno di x e restituendo 0 se x = 0, 1 se x > 0, -1 se x < 0

Esempi

```
>> x=1.23;
>> ceil(x)

ans =
    2

>> fix(x)

ans =
    1

>> floor(x)

ans =
    1
```

```
>> round(x)

ans =
    1

>> sign(x)

ans =
    1
```

Istruzioni utili per gli script

`return` Interrompe l'esecuzione dello script.
Spesso inserito in costrutti condizionali IF

`pause` Interrompe l'esecuzione dello script finche non viene premuto un tasto qualunque

CTLR + C Digitato durante l'esecuzione di uno script, ne interrompe l'esecuzione.
Utile quando ad esempio si compie un errore di programmazione che instaura una condizione di stallo o di loop infinito per il programma.

Polinomi

Rappresentazione di polinomi per mezzo di vettori

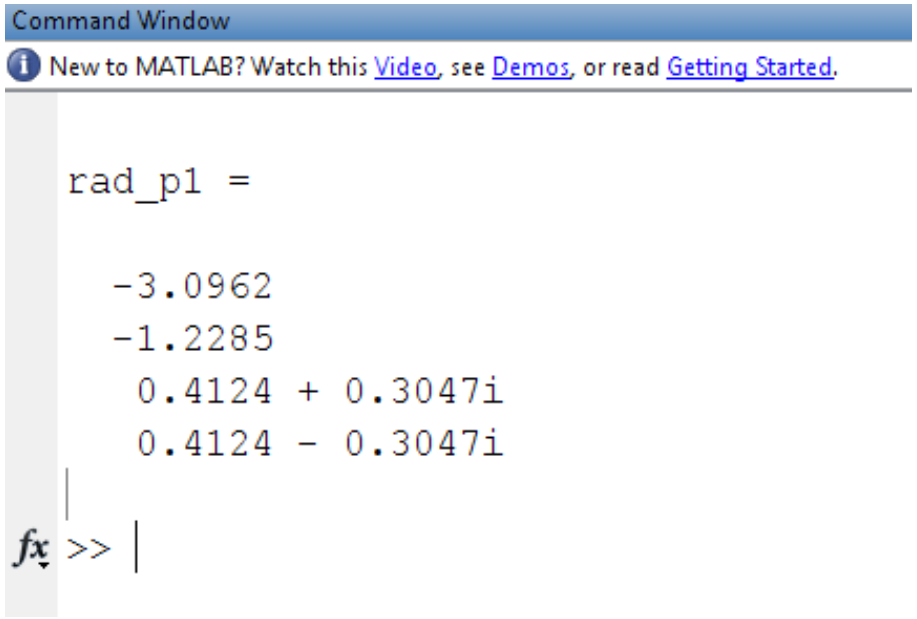
$$p1(x) = 2x^4 + 7x^3 + x^2 - 4x + 2 \qquad vp1 = [2 \quad 7 \quad 1 \quad -4 \quad 2]$$

$$p2(x) = 3x^3 + 1 \qquad vp2 = [3 \quad 0 \quad 0 \quad 1]$$

Radici di un polinomio

Funzione "**roots**"

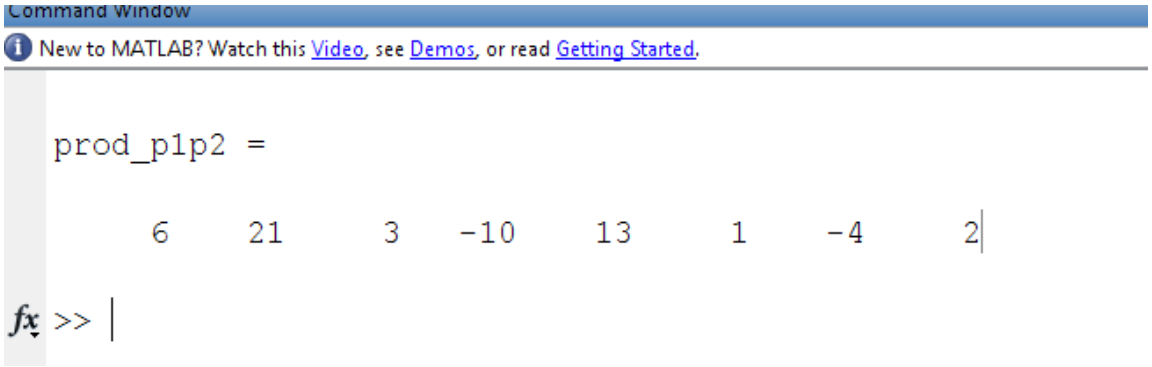
```
clear all
clc
p1=[2 7 1 -4 2];
p2=[3 0 0 1];
rad_p1=roots(p1)
```



Prodotto tra polinomi

Funzione "**conv**"

```
clear all
clc
p1=[2 7 1 -4 2];
p2=[3 0 0 1];
prod_p1p2=conv(p1,p2)
```

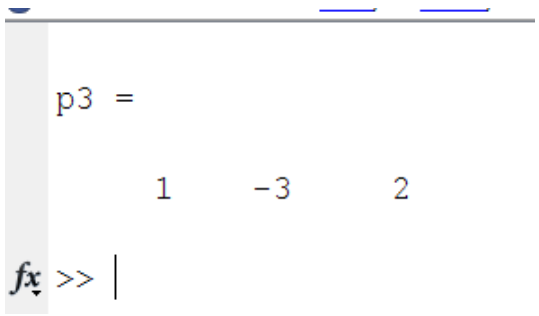


Polinomio di radici assegnate

Funzione "**Poly**"

$$p3 = poly([z1 \quad z2 \quad \ldots \quad zn]) \Rightarrow p3(x) = (x - z1)(x - z2)\ldots(x - zn)$$

```
clear all
clc
p3=poly([1 2])
```



Analisi dei dati.

Operazioni fondamentali

`max`	`Componente massima.`
`min`	`Componente minima.`
`mean`	`Valor medio.`
`std`	`Deviazione standard.`
`sort`	`ordina`
`sum`	`Somma degli elementi.`
`prod`	`Prodotto degli elementi.`
`cumsum`	`Somma cumulativa degli elementi.`
`cumprod`	`Prodotto cumulativo degli elementi.`

Differenze finite

`diff`	`Differenze prime`
`gradient`	`Gradiente approssimato`

Analisi spettrale

`fft`	`Fast Fourier Transform`

Vediamo alcuni esempi di impiego di tali funzioni applicate a vettori

```
v=[1 2 0 3.5 5 2];
x1=max(v)
x2=min(v)
x3=mean(v)
X4=std(v)
```

```
x1 =

     5


x2 =

     0
```

```
x3 =

    2.2500


x4 =

    1.7819
```

```
v=[1 2 0 3.5 5 2];
x5=sum(v)
x6=prod(v)
x7=cumsum(v)
x8=cumprod(v)
```

```
x5 =

   13.5000
```

```
x6 =

     0
```

```
x7 =

    1.0000    3.0000    3.0000    6.5000   11.5000   13.5000


x8 =

     1     2     0     0     0     0
```

```
v=[1 2 0 3.5 5 2];
x9=sort(v)
x10=sort(v,'descend')
```

```
x9 =

         0    1.0000    2.0000    2.0000    3.5000    5.0000


x10 =

    5.0000    3.5000    2.0000    2.0000    1.0000         0
```

Le funzioni `max` e `min` possono anche restituire, in aggiunta all'elemento massimo o minimo, la posizione all'interno del vettore di tale elemento. Tale funzionalità risulta utile, e la impiegheremo in vari futuri esempi.

```
v=[1 2 0 3.5 7.5 7.5]
[massimo posiz_massimo]=max(v)
[minimo posiz_minimo]=min(v)
```

```
v =

    1.0000    2.0000         0    3.5000    7.5000    7.5000
```

```
massimo =

    7.5000


posiz_massimo =

     5
```

```
minimo =

     0


posiz_minimo =

     3
```

Quando tali funzioni operano su matrici anziché su vettori restituiscono i rispettivi valori di uscita riferiti alle singole colonne della matrice

```
v=[1 2 0 3.5 5 2;0 1 4 7 2 0.5;4 1 3 2 9 6.5]
x1=max(v)
x2=min(v)
x3=sum(v)
```

```
v =

    1.0000    2.0000         0    3.5000    5.0000    2.0000
         0    1.0000    4.0000    7.0000    2.0000    0.5000
    4.0000    1.0000    3.0000    2.0000    9.0000    6.5000


x1 =

    4.0000    2.0000    4.0000    7.0000    9.0000    6.5000


x2 =

         0    1.0000         0    2.0000    2.0000    0.5000


x3 =

    5.0000    4.0000    7.0000   12.5000   16.0000    9.0000
```

Operatori relazionali

OperatoriRelazionali.mlx

< *minore*

<= *minore o uguale*

> *maggiore*

>= *maggiore o uguale*

== *uguale*

~= *diverso*

Gli operatori relazionali si applicano tra scalari,tra vettori (o matrici) di dimensioni analoghe, oppure tra un vettore (o matrice) ed uno scalare. Gli operatori testano elemento per elemento il soddisfacimento della relazione e restituiscono un vettore di variabili logiche 0/1.

```
a=3; b=1;
c=a>b
```

```
c = logical
   1
```

```
d=a==b
```

```
d = logical
   0
```

Vediamo come si comportano gli operatori relazioni se applicati a vettori di uguale dimensione

```
v=[1 4 0];
w=[0 7 0];
v>=w
```

```
ans = 1×3 logical array
   1   0   1
```

```
v==w
```

```
ans = 1×3 logical array
   0   0   1
```

E' possibile confrontare tutti gli elementi di un vettore, o di una matrice, con il medesimo numero scalare semplicememente applicando l'operatore relazionale al vettore/matrice ed al numero scalare.

```
M=[1 2 3;4 5 6;7 8 9];
M<5
```

```
ans = 3×3 logical array
   1   1   1
   1   0   0
   0   0   0
```

```
M==5
```

```
ans = 3×3 logical array
   0   0   0
   0   1   0
   0   0   0
```

I primi quattro operatori verificano solo la parte reale, gli ultimi due (**==** e **~=**) verificano anche la parte immaginaria.

Funzioni logiche

any *(**x**)*	*1 se almeno un elemento di x è non nullo, zero altrimenti.*
all *(**x**)*	*1 se tutti gli elementi di x sono non nulli, zero altrimenti*
find *(**x**)*	*restituisce gli indici degli elementi non nulli del vettore in ingresso*
isnan *(**x**)*	*vettoriale, 1 se l'elemento con quell'indice è NaN (es. 0/0), zero altrimenti*
isinf *(**x**)*	*vettoriale, 1 se l'elemento con quell'indice è infinito (es. 1/0), zero altrimenti*
isempty *(**x**)*	*scalare, vale 1 se x è una matrice vuota (A=[]), zero altrimenti*
isstr *(**x**)*	*scalare, vale 1 se x è una stringa di testo, zero altrimenti.*

```
x=[4  1  0  2  6  0]

 a1=any(x)
 a2=all(x)
 a3=find(x)
```

```
a1 =

  logical

    1


a2 =

  logical

    0


a3 =

     1     2     4     5
```

0	62.3800
0.1000	62.3900
0.2000	62.4000
0.3000	62.4000
0.4000	62.4100
0.5000	62.4200
0.6000	62.4300
0.7000	62.4300
0.8000	62.4400
0.9000	62.4500
1	62.4600
1.1000	62.4700
1.2000	62.4800
1.3000	0
1.4000	62.4900
1.5000	62.5000
1.6000	62.5100
1.7000	62.5200
1.8000	62.5300
1.9000	62.5400
2	62.5500
2.1000	62.5600
2.2000	62.5700
2.3000	62.5800
2.4000	62.5800
2.5000	0
2.6000	62.6000
2.7000	62.6100
2.8000	62.6200
2.9000	62.6300
3	62.6400

Rimozione da uno stream di dati al di sotto di un valore di soglia

Carichiamo nel workspace, con il comando LOAD, il file stream.mat

Esso provoca il caricamento nel WS di due vettori, time e data.

Desideriamo rimuovere dallo stream le misure inferiori a 10 bar.

```
clear all, clc
load stream

soglia=10;
z1=data<soglia;
indici=find(z1);

data_f=data;
time_f=time;
data_f(indici)=[];
time_f(indici)=[];

figure(1)
plot(time,data),grid
title('Dati grezzi')
figure(2)
plot(time_f,data_f),grid
title('Dati processati')
```

z1 è un vettore di elementi booleani, che contiene 1 in corrispondenza dei valori dei dati che stanno al di sotto della soglia, e zero altrove

Con la funzione find estraiamo gli indici associati alle posizioni di tali misure sottosoglia.

Esempio Trovare in un vettore a l'elemento che risulta più prossimo ad un certo valore v, e visualizzare tale elemento

```
clc
a=1:1:10;
v=4.9;
z=abs(a-v);
zmin=min(z);
indice=find(z==zmin)
elemento=a(indice)
```

```
z =

  Columns 1 through 7

    3.9000    2.9000    1.9000    0.9000    0.1000    1.1000    2.1000

  Columns 8 through 10

    3.1000    4.1000    5.1000


zmin =

    0.1000


indice =

     5


elemento =

     5
```

Se si mette come soglia
4.5 anziche 4.9 ci sono due elementi equidistanti, e lo script li indentifica entrambi