Utilizzo di Matlab per l'analisi di sistemi dinamici

Sistemi dinamici a tempo continuo ed a tempo discreto

Indice del materiale

- Breve **introduzione** a Matlab
  - Descrizione generale di Matlab (v. 6.x)
  - Quadro delle funzioni predefinite
  - Definizione di matrici, vettori e polinomi
  - Rappresentazione grafica dei dati

- **Analisi** e **simulazione** di **sistemi dinamici LTI** in ambiente Matlab
  - Rappresentazione di sistemi dinamici lineari tempo-invarianti nell'ambiente Matlab
  - Analisi e simulazione di sistemi dinamici lineari tempo-invarianti

Indice del materiale (2)

- **Sistemi lineari interconnessi**
  - Sistemi dinamici lineari interconnessi in ambiente Matlab
  - Utilizzo delle istruzioni Matlab FEEDBACK + CONNECT
- Determinazione della **risposta in frequenza** di sistemi dinamici lineari tempo-invarianti nell'ambiente Matlab
  - Esempi di utilizzo
- Conversione da **tempo continuo** a **segnali campionati/ tempo discreto** e viceversa.

Breve introduzione a Matlab

L'ambiente di lavoro,
le istruzioni fondamentali

A cosa serve questa presentazione

- Scopi di questo materiale:
  - fornire le informazioni necessarie per l'uso di **Matlab** in relazione alle esercitazioni del corso;
  - dare una panoramica generale (**tutt'altro che esauriente**) delle potenzialità di **Matlab** per la formulazione e la soluzione di problemi numerici nell'Ingegneria.

Dove trovare altre informazioni?

- Sito web di Mathworks:

  www.mathworks.com

  seguendo i link alla voce "support" è possibile trovare i manuali di Matlab in formato Adobe PDF.

  (http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml)

- Un testo in italiano di introduzione a Matlab e Simulink:

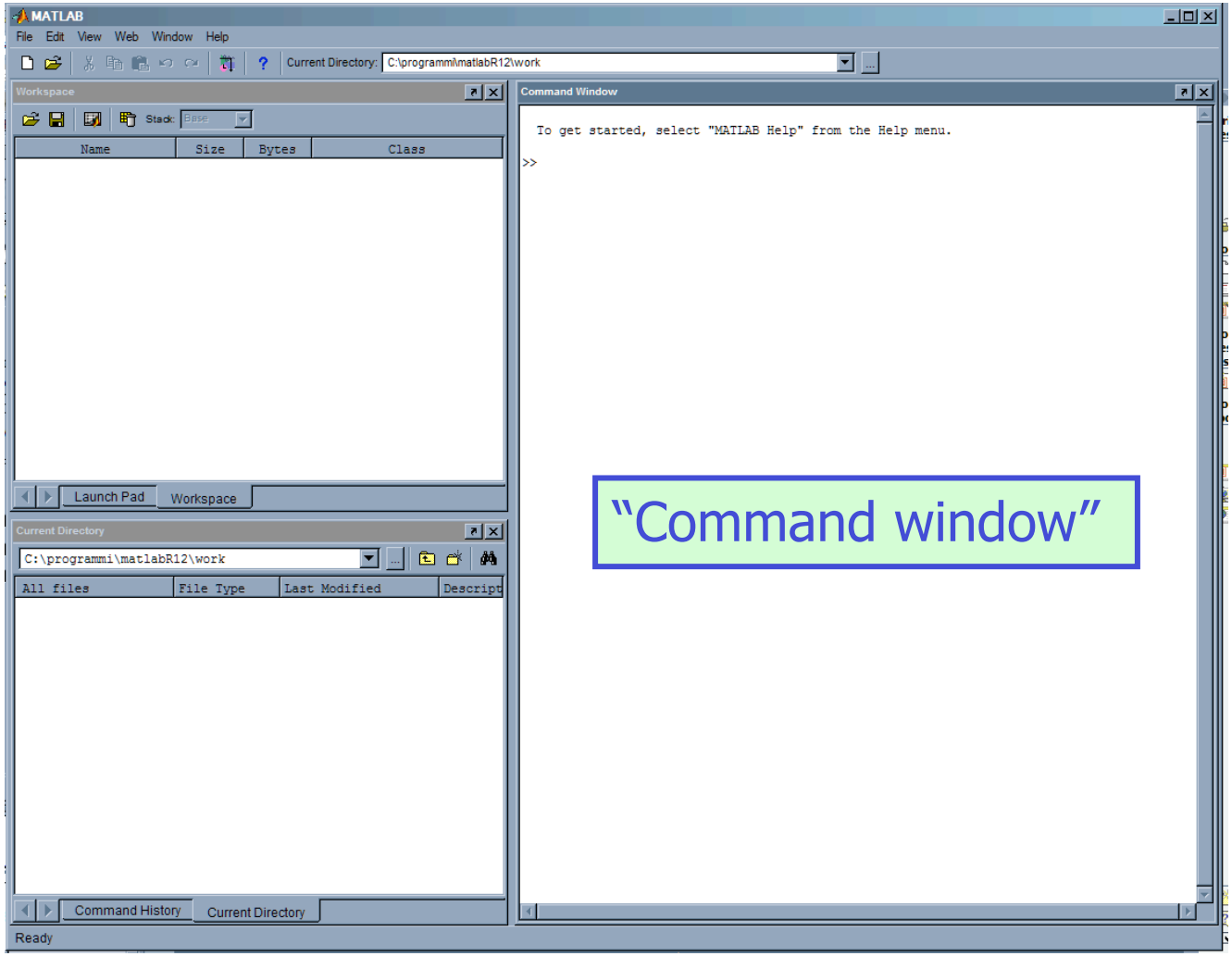
  "***Guida Operativa a MATLAB, SIMULINK e Control Toolbox***" , Alberto Cavallo, Roberto Setola, & Francesco Vasca, Liguori Editore, 1994 → in biblioteca

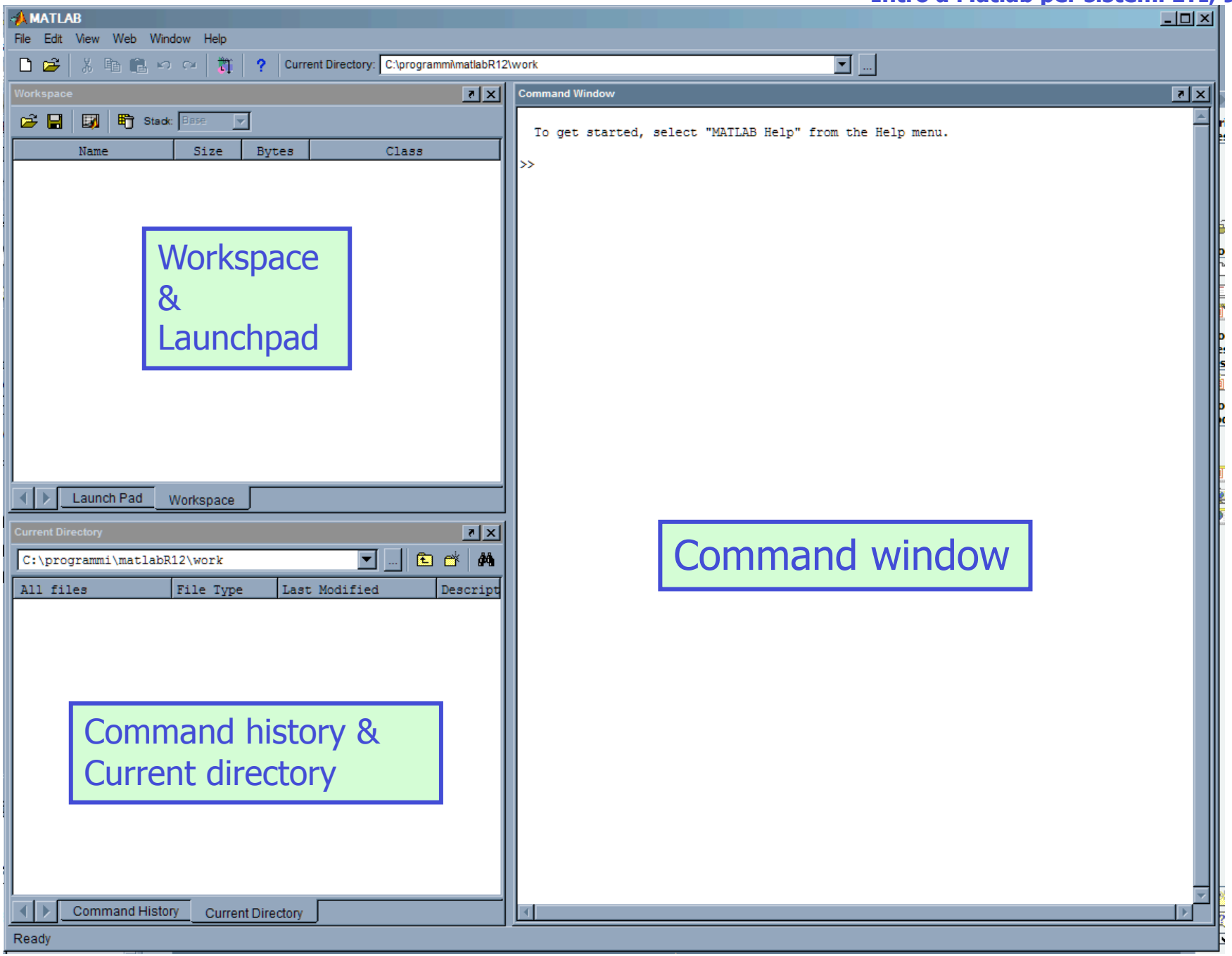
Descrizione generale di Matlab

- MATLAB (= MATrix LABoratory):
  - un **linguaggio di programmazione** per applicazioni scientifiche e numeriche
  - vasto set di **funzioni predefinite**
  - **interprete** di comandi
  - possibilità di scrivere nuove funzioni
  - libreria di **TOOLBOX** per svariate applicazioni; ad es. Signal Processing, Identificazione di modelli, …

L'interfaccia di Matlab

- Interfaccia utente:
  - la ***Command Window*** dà accesso diretto all'interprete
  - scrittura diretta di comandi.




MATLAB
File Edit View Web Window Help
Current Directory: C:\programmi\matlabR12\work
Workspace
Stack: Base
Name Size Bytes Class
Launch Pad Workspace
Current Directory
C:\programmi\matlabR12\work
All files File Type Last Modified Descript
Command History Current Directory
Command Window
To get started, select "MATLAB Help" from the Help menu.
>>
Ready
Workspace & Launchpad
Command history & Current directory
Command window

- **Command Window**: è la finestra dell'interprete di comandi di Matlab. Vi si possono scrivere direttamente comandi o lanciare programmi scritti nel linguaggio di programmazione del Matlab.

- **Command History**: è la finestra che contiene l'elenco dei comandi inseriti nella Command Window.

- **Current Directory**: visualizza in forma grafica il contenuto della cartella di lavoro.

- **Workspace**: è la finestra che visualizza il contenuto dell'area di memoria utilizzata dal Matlab (nomi, tipi e dimensioni delle variabili).

- **LaunchPad**: visualizza l'elenco dei comandi/delle funzioni richiamabili nella Command Window, organizzati secondo l'appartenenza alle varie categorie di comandi/funzioni disponibili (elaborazione immagini, filtraggio, controllo …)

Matlab come calcolatrice...

- La modalità di impiego più "semplice": per valutare espressioni numeriche.
- Esempio: per calcolare $4 + \sqrt{2} - \sin(0.2\pi)^2 + e^2$
  è sufficiente digitare al prompt »

```
»4 + sqrt(2) - sin(0.2*pi)^2 + exp(2)
    ans =
       12.4578
```

- Il risultato viene salvato nella variabile *ans*.

Definizione di variabili

- È possibile definire variabili e espressioni non numeriche più complesse.

- Esempio:

```
» a=4; b=2;
» a*b
ans =
    8
```

- Per cancellare una variabile (es. a):

```
» clear a
```

Il Workspace

- Ogni variabile definita in questo modo viene conservata in memoria, nel ***Workspace***.
- Il comando ***whos*** mostra una lista delle variabili definite:

```
» whos
  Name      Size            Bytes  Class
  a         1x1                 8  double array
  ans       1x1                 8  double array
  b         1x1                 8  double array
Grand total is 3 elements using 24 bytes
```

Lettura e scrittura su file

Mediante i comandi ***load*** e ***save*** è possibile salvare su file le variabili del ***workspace***.

- ***load*** *nomefile variabile1 variabile2* **...** carica dal file *nomefile.mat* le variabili elencate
- ***save*** *nomefile variabile1 variabile2* **...** scrive nel file *nomefile.mat* le variabili elencate.
- ***load*** *nomefile* carica tutte le variabili in nomefile.
- ***save*** *nomefile* salva tutto il **workspace** in *nomefile.mat*

Quindi...

- Esiste un insieme (molto vasto) di funzioni predefinite (come ***sin*** e ***sqrt*** nell'esempio precedente).

- A differenza dei normali linguaggi (C, Pascal...) non occorre *dichiarare* le variabili. L'assegnazione coincide con la dichiarazione.

Operazioni matematiche elementari

- Moltiplicazione $a\,b$ ⟷ **a * b**
- Divisione (divisione "a destra") $\frac{a}{b}$ ⟷ **a / b**
- Divisione "a sinistra" $\frac{b}{a}$ ⟷ **a \ b**
- Elevamento a potenza a^b ⟷ **a ^ b**
- Addizione e sottrazione $a + b$, $a - b$ ⟷ **a + b**, **a - b**

Operazioni elementari: precedenza

- La precedenza nello svolgere le operazioni matematiche elementari in Matlab è quella standard, facendo però attenzione alle operazioni di "divisione a destra" e "divisione a sinistra".
- Infatti valgono le regole:
  - L'**elevamento a potenza** viene valutato prima delle operazioni di **moltiplicazione** e **divisione**, che hanno la medesima priorità,
  - L'operazione di "**divisione a destra**" ha priorità rispetto alla "**divisione a sinistra**";
  - Le operazioni di **addizione** e **sottrazione** hanno la priorità più bassa.
  - Per assegnare una diversa precedenza nella sequenza dei calcoli è necessario **utilizzare le parentesi**.

L'operatore di assegnazione "="

- L'operatore "**=**" è utilizzato in Matlab per **assegnare** un valore (numerico, come il risultato di un'operazione di calcolo, oppure testuale o simbolico [ne parliamo più avanti]) ad una variabile.

- Quindi è appropriato pensare all'operatore "=" come ad un'istruzione con la quale si assegna un valore ad una variabile, in un qualsiasi linguaggio di programmazione (C, Fortran ecc.).

- Se l'assegnazione termina con un "**;**" allora il risultato dell'operazione di assegnazione non viene visualizzato, altrimenti si.

```
» x = 2; y = 4; z = x*y
z = 8
```

L'operatore di assegnazione "=" (2)

- Nel caso di assegnazioni lunghe si può utilizzare il simbolo "**...**" per poi proseguire l'assegnazione nella riga seguente

```
» FirstClassHolders = 72;
» Coach = 121;
» Crew = 8;
» TotalPeopleOnPlane = FirstClassHolders + Coach...
+ Crew

TotalPeopleOnPlane =
201
```

Il formato di visualizzazione

- Il risultato di un'operazione numerica viene visualizzato di solito tramite un numero reale, con sole 4 cifre decimali.
- Questo è il formato di visualizzazione standard.

- Esistono altre possibilità e si scelgono tramite il comando **format**:
  - **format short** 4 cifre decimali
    - **short e** 4 cifre decimali, notazione esponenziale
  - **format long** 15 cifre decimali
    - **long e** 15 cifre decimali, notazione esponenziale
  - …

Il formato di visualizzazione

- **format rat** risultato approssimato tramite la più vicina frazione
- **format bank** formato finanziario, con sole due cifre decimali
- …

```
»format long
» x = 3 + 11/16 + 2^1.2
   x =  5.98489670999407
» format short
» x = 3 + 11/16 + 2^1.2
   x =  5.9849
```

Esempi di funzioni predefinite
(di uso piu' comune)

- Funzioni trigonometriche (***sin***, ***cos***, ***tan***, ***acos***, ***asin***, ***atan***…);
- Esponenziale e logaritmo (***exp***, ***log***, ***log10***, ***sqrt***…);
- Numeri complessi (***abs*** ⇒ modulo, ***angle*** ⇒ fase, ***real*** ⇒ parte reale, ***imag*** ⇒ parte immaginaria…);

Alcuni esempi semplici

- Calcolare il modulo di (2+3i):

```
» abs( 2+3*i )
   ans =
      3.6056
```

- Calcolare $20\log_{10}\left(\left|\frac{2+3i}{4+6i}\right|\right)$

```
» 20*log10( abs( (2+3*i) / (4+6*i) ) )
   ans =
      -6.0206
```

Inf e NaN

- Alcune operazioni numeriche possono dare luogo a risultati non corretti, esprimibili soltanto facendo uso di "forme indeterminate", che vengono definite da Matlab tramite le grandezze **Inf** e **NaN**.

- Esempi:

```
» 5/0
Warning: Divide by zero.
ans =
   Inf
```

```
» 0/0
Warning: Divide by zero.
ans =
   NaN
```

Help! Una funzione fondamentale!

- **help** per vedere la lista dei toolbox installati
- **help *nome_toolbox*** per vedere la lista dei comandi installati in un *toolbox*
- **help *nome_comando*** guida 'on-line' di MATLAB sullo specifico comando
- **ver** info sulla versione di MATLAB
- **helpwin** finestra di help di MATLAB

Definizione di matrici

- Come si definisce una matrice in Matlab?

  Esempio: definire la matrice 2x2

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
» A = [ 1, 2; 3, 4 ]
A =
    1    2
    3    4
```

- Come si accede agli elementi di una matrice:

```
» A( 1, 2 )
ans =
    2
```

Indici (riga e colonna) dell'elemento di interesse

La wildcard :

- Per accedere a intere righe o colonne di una matrice, si usa la *wildcard* **:**

- Es.: selezionare la prima riga di A

```
» A(1,:)
ans =
    1    2
```

- Es.: selezionare la seconda colonna di A

```
» A(:,2)
ans =
    2
    4
```

Selezionare sottomatrici

- Se definiamo

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

Sottomatrice di interesse

- scrivendo

```
» B( 1:2 , 2:3 )
ans =
    2    3
    5    6
```

Indici della sottomatrice di interesse

Operazioni (elementari) sulle matrici

- Sono definiti gli operatori **+**, **-**, * e **^**

```
» A = [ 1 2; 3 4 ]; B = [ 5 6; 7 8 ]
» A+B
ans =
    6    8
   10   12

» B-A
ans =
    4    4
    4    4
```

```
» A*B
ans =
   19   22
   43   50

» A^2
ans =
    7   10
   15   22
```

Operazioni (elementari) sulle matrici (2)

- Sono definiti gli operatori **.*, ./** e **.^**, che si applicano elemento per elemento:

```
» A = [ 1 2; 3 4 ]; B = [ 5 6; 7 8 ];
» A.*B
ans =
     5    12
    21    32
» A.^B
ans =
           1          64
        2187       65536
```

Operazioni (elementari) sulle matrici (3)

- Determinante:

```
» det(A)
ans =
    -2
```

- Autovalori:

```
» eig(A)
ans =
  -0.3723
   5.3723
```

- Matrice inversa:

```
» inv(A)
ans =
  -2.0000   1.0000
   1.5000  -0.5000
```

- Matrice trasposta:

```
» A'
ans =
     1     3
     2     4
```

Operazioni (elementari)

- Funzioni matematiche di base **help elfun**
- Funzioni matematiche specifiche **help specfun**
- Funzioni di manipolazione delle matrici **help elmat**

Altre operazioni

- Osservazione importante: NON occorre definire le dimensioni in modo esplicito!

  Per conoscere le dimensioni di una matrice: ***size***

- Altre operazioni:
  - ***rank*** -> calcolo del rango di una matrice
  - ***trace*** -> calcolo della traccia di una matrice
  - ***norm*** -> calcolo della norma di una matrice

Alcune matrici "speciali"

- ***eye*(n,n)** ⇒ matrice identità n x n;
- ***zeros*(n,m)** ⇒ matrice di "0" n x m;
- ***ones*(n,m)** ⇒ matrice di "1" n x m;
- ***rand*(n,m)** ⇒ matrice n x m con elementi distribuiti uniformemente tra 0 e 1.

Vettori

- I vettori hanno due funzioni fondamentali in Matlab:
  - rappresentazione di **polinomi** (un polinomio è descritto dal vettore dei suoi coefficienti);
  - rappresentazione di **segnali** (un segnale è rappresentato mediante la sequenza dei valori che assume in un insieme di istanti di tempo, quindi mediante un vettore).

Definizione di vettori (1)

- » v=(0:10)

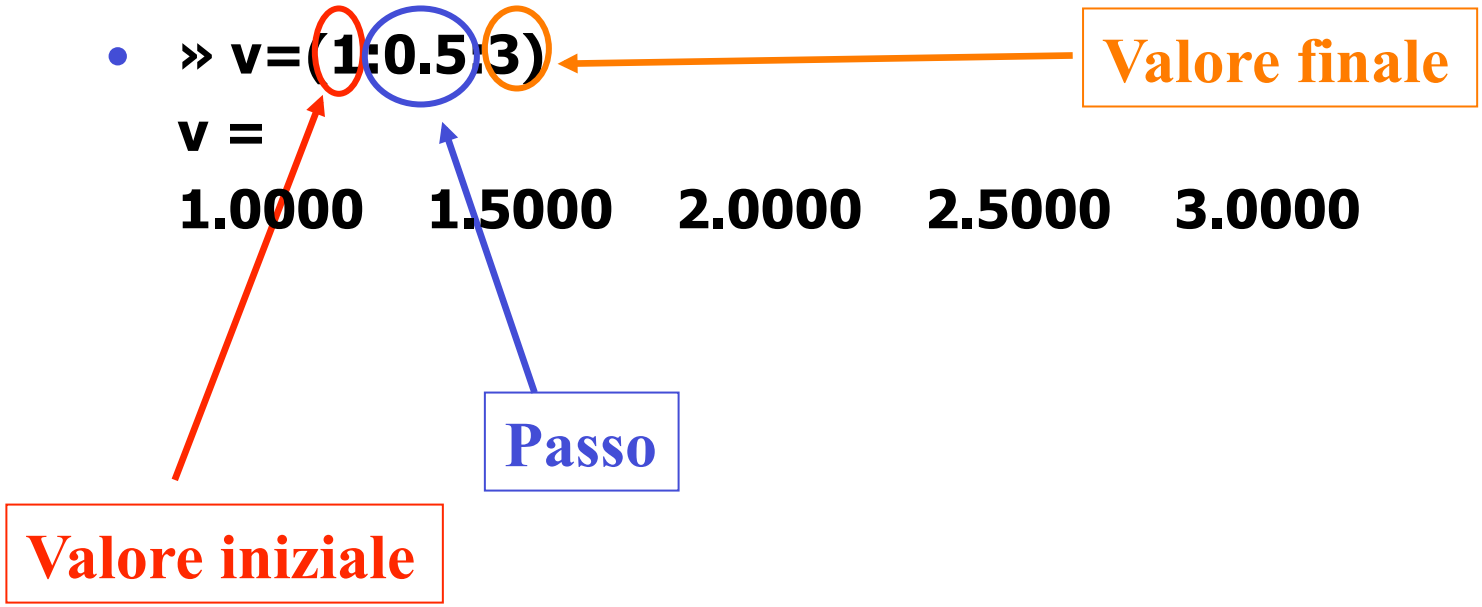
  v =

  0 1 2 3 4 5 6 7 8 9 10

- » v=(1:0.5:3)

  v =

  1.0000 1.5000 2.0000 2.5000 3.0000



Definizione di vettori (2)

- Come matrici riga o colonna:

```
» v = [ 3 6 1 7 ]
v =
   3    6    1    7
```

- Polinomi: sono rappresentati come vettori
  Es.:

```
» pol = [ 3 2 1 ]
pol =
   3    2    1
```

$$p(z) = 3z^2 + 2z + 1$$

Operazioni sui polinomi

- Calcolo delle radici ⇒ ***roots***

```
» roots( pol )
ans =
  -0.3333 + 0.4714i
  -0.3333 - 0.4714i
```

- Valutazione in un punto ⇒ ***polyval***

```
» polyval( pol, 0 )
ans =
     1
```

Operazioni sui polinomi (2)

- Prodotto di polinomi ⇒ ***conv***

  Esempio:

$$(z+1)(z+1) = z^2 + 2z + 1$$

```
» pol1 = [ 1 1 ]; pol2 = [ 1 1 ];
» polprod = conv( pol1, pol2 )
polprod =
     1     2     1
```

Rappresentazione grafica

- Grafici 2D:
  - In **scala lineare** ⇒ ***plot***

    plot(x,y) traccia il grafico dei punti che hanno come ascisse (ordinate) gli elementi del vettore x (y).
  - In **scala semilogaritmica o logaritmica** ⇒ ***semilogx***, ***semilogy***, ***loglog***

    stessa sintassi di *plot*
  - Diagrammi **polari** ⇒ ***polar***

Rappresentazione grafica (2)

- Altre funzioni utili:
  - cambiamenti di scala ⇒ ***axis*([xmin,xmax,ymin,ymax])**
  - sovrapposizione di più grafici ⇒ ***hold***
  - aggiunta di griglia al grafico ⇒ ***grid***
  - titolo al grafico ed etichette agli assi ⇒ ***title('..')***, ***xlabel('..')***, ***ylabel('..')***
  - più grafici in una finestra ⇒ ***subplot***
  - inserimento di testo in una figura ⇒ ***gtext***

Esempio

- Grafici sovrapposti

```
» t=(0:0.01:5);
» y1=sin(t);
» y2=cos(2*t-pi/3);
» figure;
» plot(t,y1,'r',t,y2,'g');
» grid on;
```

Un tipo di dato "speciale": *cell array*

- Il tipo di dato "cell array" sta ad indicare che si vuole creare un oggetto di tipo **array** i cui **elementi** possono essere **di tipo diverso** (valori numerici, testo, altri tipi di dati).

- Operativamente, un dato di tipo "cell array" si crea in modo simile agli array "classici" in Matlab, semplicemente sostituiendo ai delimitatori classici [ ] i delimitatori { }.

- Esempio: per creare un cell array 2-x-2

```
A = {[1 4 3; 0 5 8; 7 2 9], 'Paolino Paperino';3+7i, -pi:pi/4:pi};
```

Cenni alla programmazione in ambiente Matlab

Operazioni con vettori, caratteri e testo

Cicli, operazioni e condizioni logiche

Funzioni e struttura di un programma in Matlab

Operazioni sui vettori e matrici

- I vettori (e le matrici) possono essere manipolati in Matlab non solo per estrarre parti di essi, ma anche per
  - Concatenare più vettori (matrici) assieme

```
» t1 = [1 :10];
» t2 = [ 20:-1:11];
» t3=[t1, t2];
```

  - Modificare le dimensioni di un vettore (matrice)

```
» help reshape
» t3bis = reshape(t3,2,10);
```

– Cancellare/sostituire elementi in un vettore (matrice)

```
» disp(t3bis);
» t3bis(:,4:6)=[];
» disp(t3bis);
» whos
```

```
» disp(t3bis)
» t3bis(1,4:6)=-3;
» whos
```

Operazioni su caratteri e testo

- Le stringhe di caratteri sono viste come particolari array in Matlab, quindi su di esse possono essere applicate le operazioni di manipolazione sui vettori già descritte:

```
» t1 = 'A'
» t2 = 'BCDE'
» t3 = [t1,t2]
» t4 = [t3,' are the first 5 ';...
'characters in the alphabet.']
```

- A volte e' necessario convertire un numero in testo e viceversa: vedere le istruzioni **str2num**, **num2str**, **int2str** e simili.

Cicli, operazioni e condizioni logiche

- Operazioni cicliche:

```
for indice = inizio:passo:fine
    fai_qualcosa;
end
```

- Operazioni logiche, condizioni logiche elementari:
  - "vero" e "falso": **true = 1 false = 0**
  - Confronti logici:
    - **x == 2** x e' uguale a 2?
    - **x ~= 2** x NON e' uguale a 2?
    - **x > 2** x e' maggiore di 2?
    - **x < 2** x e' minore di 2?
    - **x >= 2** x e' maggiore oppure uguale a 2?
    - **x <= 2** x e' minore oppure uguale a 2?

- Attenzione a questo esempio:

```
» x = pi
x =
3.1416
```

Questo non e' un test, ma una assegnazione!

```
» x ~= 3, x ~= pi
ans =
1
ans =
0
```

Questi sono test logici.

Composizione di operazioni logiche

- Le operazioni logiche elementari si possono comporre con gli operatori logici “and”, “or”, “xor”, “not”:
  - And **&**
  - Or **|**
  - Xor **xor**
  - Not **~**

- Per una descrizione completa: **help relopt**

Cicli "while"

- La struttura del ciclo e' quella classica

while test logico

Comandi_da_eseguire mentre la condizione logica ha valore "true"

end

```
» S = 1; n = 1;
» while S+ (n+1)^2 < 100
    n = n+1; S = S + n^2;
  end
» [n, S]
```

"if" ... "then" ... "else" ...

- La struttura e' quella classica

```
if test_logico_1
   Comandi da eseguire se test1 e' "true"
elseif test_logico_2
   Comandi da eseguire quando test2 e' "true" ma
   test1 e' "false"
...
end
```

Un esempio

- Si vuole risolvere l'equazione $x = \cos(x)$
- La trasformiamo in una successione e cerchiamo di trovar a quale valore converge la successione

$$x_n = \cos(x_{n-1}) \quad n \in \mathbf{N}$$

- Il tutto deve essere uno script Matlab.

```
% risolvi x= cos(x)

% metodo 1
% spreca memoria, memorizza anche passi intermedi
x = zeros(1,20); x(1) = pi/4;
n = 1; d = 1;
 while d > 0.001
    n = n+1; x(n) = cos(x(n-1));
    d = abs( x(n) - x(n-1) );
end
n,x


% metodo 2
% migliore memorizza solo l'ultimo dato
 xold = pi/4; n = 1; d = 1;
 while d > 0.001 & n < 20
    n = n+1; xnew = cos(xold);
    d = abs( xnew - xold );
    xold = xnew;
end
[n, xnew, d]
```

Matlab function

- Un m-file e' una Matlab function **se e solo se** la prima riga del file contiene il seguente testo

  - **function** [elenco variabili in uscita] =
    nome_della_funzione (elenco variabili in ingresso)

- Non e' necessario terminare il file con una parola chiave, anche se esiste l'istruzione "**return**" (vedere **help return**)

Alcuni esempi

- Calcolo dell'area di un triangolo qualsiasi:
  - Vale la formula $$A = \sqrt{s(s-a)(s-b)(s-c)}$$
  $$s = \frac{a+b+c}{2}$$
  - con a,b,c, lunghezze dei lati del triangolo.

- La funzione allora (nella sua versione piu' semplice) avra' 3 parametri in ingresso ed 1 solo parametro in uscita

```
function [A] = area(a,b,c)
% Compute the area of a triangle whose
% sides have length a, b and c.
% Inputs:
% a,b,c: Lengths of sides
% Output:
% A: area of triangle
% Usage:
% Area = area(2,3,4);
% Written by XXX, MM/DD/YY.
s = (a+b+c)/2;
A = sqrt(s*(s-a)*(s-b)*(s-c));
%%%%%%%% end of area %%%%%%%%%%%
```

- Provare a creare il file “my_area.m” e poi a scrivere il comando “help my_area” nella Command Window di Matlab.

- Utilizzo della funzione
  - Assegnando una variabile in uscita

```
» Area = area(10,15,20)
Area =
72.6184
```

  - non assegnando la variabile d’uscita

```
» area(10,15,20)
ans =
72.6184
```

Un esempio: i numeri di Fibonacci

- Calcolare la sequenza:

$$f_n = f_{n-1} + f_{n-2} \quad n = 3, 4, 5 \ldots$$

- Calcolare il valore a cui tende il rapporto tra due termini successivi al crescere dell'indice

$$\frac{f_{n-1}}{f_n} = ?$$

$$f(1) = 0 \qquad f(2) = 1$$

```
F(1) = 0; F(2) = 1;
% inizializzazione
for i = 3:20
    F(i) = F(i-1) + F(i-2);
end
% calcolo della sequenza

% grafici
plot(1:19, F(1:19)./F(2:20),'o' )
hold on, xlabel('n')
plot(1:19, F(1:19)./F(2:20),'-' )
legend('rapporto dei termini  f_{n-1}/f_n')
plot([0 20], (sqrt(5)-1)/2*[1,1],'--')
```

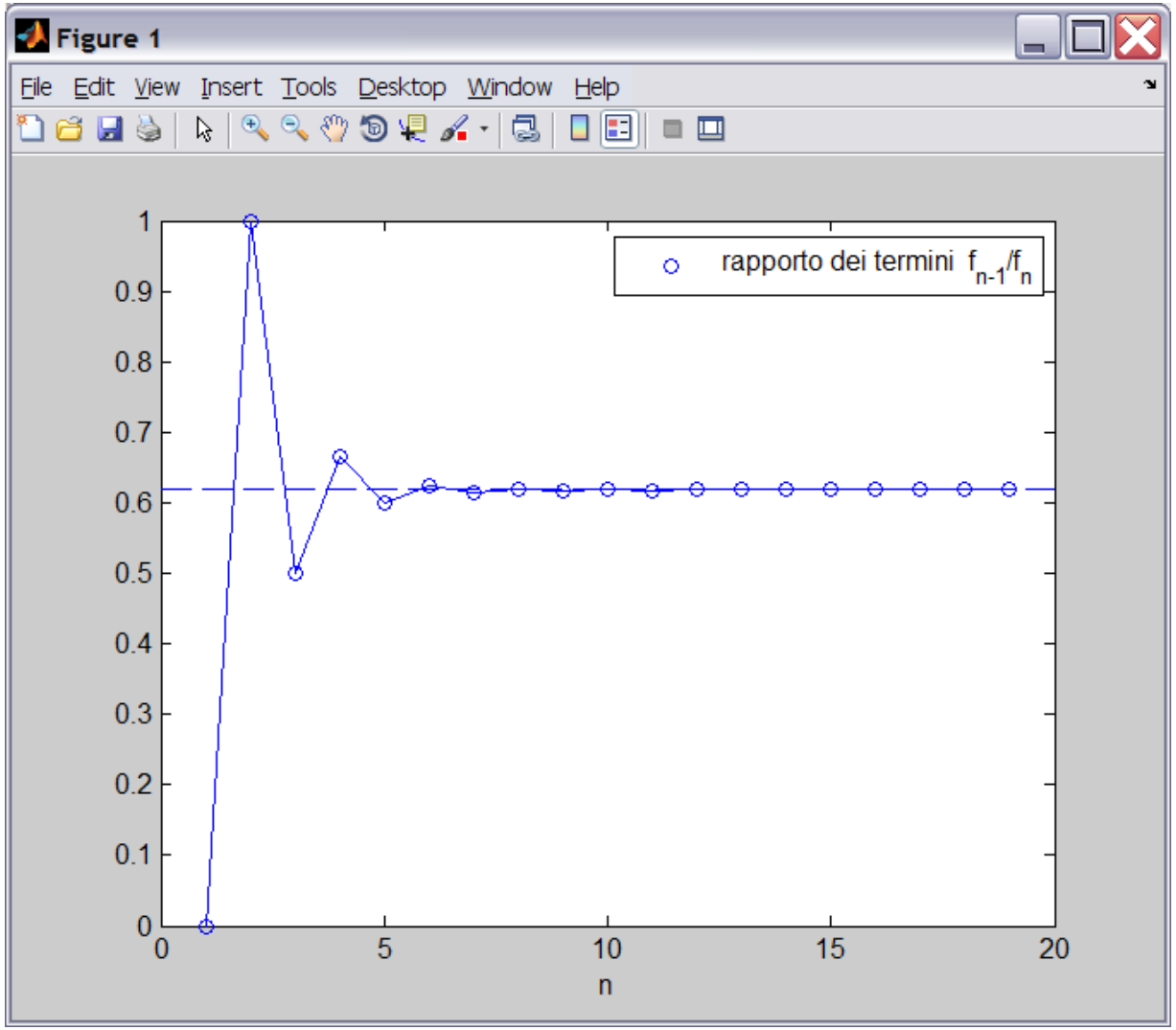


Figure 1
File Edit View Insert Tools Desktop Window Help
rapporto dei termini f_{n-1}/f_n
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
n

Cenni alla "vettorializzazione del codice"

- Per "**calcolo vettorializzato**" si intende un **algoritmo che trae vantaggio dallo sfruttare/eseguire operazioni su interi vettori/matrici** piuttosto che su ciascun singolo componente di essi.

- Si può "vettorializzare" una discreta varietà di strutture di programmazione e tale operazione può portare ad incrementare la velocità d'esecuzione anche di un fattore 10.
- La "vettorializzazione" del codice è **una** delle **pratiche più efficienti** per ottenere **codice** Matlab molto **performante**.

- NB "vettorializzare" NON è sinonimo di "parallelizzare" il codice!

Vettorializzare i cicli

- Vettorializzare in questo caso significa trasformare in operazioni su vettori/matrici le operazioni contenute in cicli for, oppure while.
- Esempio

```
i = 0;
for t = 0:.01:10
  i = i + 1;
  y(i) = sin(t);
end
```

- La versione vettoriale del medesimo codice può essere

```
t = 0:.01:10; y = sin(t);
```

- Quanto si guadagna?

- Lo script "test_ciclo.m" fornisce il seguente risultato

```
risultati
primo ciclo
  7.6309e-005
secondo ciclo
  5.6040e-005
```

Incremento di velocità del 25% circa

- Appare evidente che si riesce ad incrementare la velocità di esecuzione.

- **ATTENZIONE**: non sempre è possibile vettorializzare i cicli. In tal caso si può pensare di precompilare il codice "scalare" per ottenere comunque un certo incremento di prestazioni. **NON** affrontiamo l'argomento.

```
% test_ciclo
% script che serve a eseguire il timing su due cicli,
% uno vettorializzato e l'altro no.
%
tic;
 i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
tempo1 = toc;

% secondo ciclo
clear t y
tic;
t = 0:.01:10;
y = sin(t);
tempo2 = toc;

% risultato
messaggio = sprintf('ecco il risultato: primo ciclo %f secondo ciclo
%f',tempo1, tempo2);
disp(messaggio);
```

Vettorializzazione di algoritmi

- Si può incrementare la velocità di esecuzione vettorializzando anche altre strutture di codice.

- Esempio: versione "scalare"

```
function d = minDistance(x,y,z)
% trova in un insieme di punti di R^3
% quello a distanza minima dall'origine e restituisce la distanza min
nPoints = length(x);
d = zeros(nPoints,1); % prealloca (ottimizzazione tempo di calcolo)
for k = 1:nPoints % calcola la distanza per ogni punto
  d(k) = sqrt(x(k)^2 + y(k)^2 + z(k)^2);
end
d = min(d); % trova la distanza minima
```

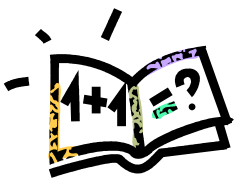
- versione "vettoriale" del medesimo algoritmo

```
function d = minDistance(x,y,z)
% trova la distanza minima dall'origine in R^3
d = sqrt(x.^2 + y.^2 + z.^2); % calcola la distanza per ogni punto
d = min(d); % trova la distanza minima
```

- Si può migliorare ancora:

```
d = sqrt(min(x.^2 + y.^2 + z.^2)))
```

- Provare le tre versioni su di un insieme di punti (es. 10000 punti a caso), misurando le prestazioni in tutti e tre i casi.



1° test (5-10 minuti)

- Risolvere tramite uno script Matlab e/o una Matlab function il problema di algebra matriciale

$$A\,x \;=\; b$$

- Le matrici del problema vengono assegnate all'inizio dello script.
- È necessario prevedere che la matrice A possa essere non quadrata e che nel caso sia matrice quadrata possa essere non invertibile.

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad b_1 = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ -1 & 0 & 10 \\ 4 & 5 & 0 \end{bmatrix} \quad b_2 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ -4 \end{bmatrix}$$

- **Soluzione del test**:
  - Molto semplicemente si sfrutta una peculiare istruzione per il calcolo matriciale
  - in ambiente Matlab il comando **A/B** equivale a risolvere il problema

$$A\,x \;=\; b$$

    sia nel caso di matrice A quadrata, sia nel caso generale di matrice A non quadrata (quindi soluzione ai minimi quadrati).

- Si veda **help mldivide**

Analisi e simulazione di sistemi dinamici LTI in ambiente Matlab

Definizioni
Proprietà

Sistemi dinamici lineari

- Un sistema dinamico **lineare tempo-invariante (LTI system)** può essere descritto:
  - in forma di **equazioni di stato**, assegnando le **matrici A,B,C,D**

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

Sistemi dinamici lineari (...)

- Un sistema dinamico **lineare tempo-invariante (LTI system)** può essere descritto:
  - in forma di **matrice di funzioni di trasferimento**
    - mediante i **polinomi a numeratore e denominatore** delle funzioni di trasferimento

$$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots a_0}$$

$$H(z) = \frac{b_m z^m + b_{m-1} z^{m-1} + \dots b_0}{a_n z^n + a_{n-1} z^{n-1} + \dots a_0}$$

Sistemi dinamici lineari (...)

- Un sistema dinamico **lineare tempo-invariante (LTI system)** può essere descritto:
  - in forma di **matrice di funzioni di trasferimento**
    - assegnando **zeri, poli e guadagno** delle funzioni di trasferimento

$$H(s) = K \frac{\prod_q (s + z_q)}{\prod_r (s + p_r)}$$

$$H(z) = K \frac{\prod_q (z + z_q)}{\prod_r (z + p_r)}$$

Sistemi dinamici lineari (…)

- Nell'ambiente Matlab è possibile definire un sistema dinamico lineare tempo-invariante come ***oggetto*** **di tipo LTI model** a partire da qualsiasi di queste descrizioni (ne esistono anche altre, ma non le analizziamo [si veda il comando **ltimodels**]).

Sistemi dinamici lineari (...)

- Utilizzando questi oggetti di tipo **LTI model** è possibile **analizzare le proprietà** del sistema dinamico corrispondente (stabilità ecc.) ed è possibile **simulare l'evoluzione nel tempo** del sistema dinamico, con condizioni iniziali ed ingressi assegnati.

Sistemi lineari LTI SISO

- Nella descrizione dei comandi Matlab e negli esempi consideriamo soltanto **sistemi SISO.**

Sistemi lineari SISO: dalle equazioni di stato

- partendo dalle **equazioni di stato**

  - definire le matrici **A,B,C,D** nel **workspace**;
  - definire il sistema mediante il comando ***ss***

  - **sintassi** del comando **ss**
    - ***SYS = ss(A,B,C,D)*** crea un sistema dinamico a tempo continuo

    - ***SYS = ss(A,B,C,D,Ts)*** crea un sistema dinamico a tempo discreto, con intervallo di campionamento specificato da Ts [s].
    - Ponendo **Ts** pari a **-1**, si lascia non specificato l'intervallo di campionamento associato al sistema.

Esempi (1)

- Definizione del sistema dinamico a tempo continuo:

$$\begin{cases} \dot{x}_1 = -x_2 + 3u \\ \dot{x}_2 = -3x_1 + 2x_2 \\ y = 4x_1 + 2u \end{cases}$$

```
» A = [ 0 -1;-3 2]; B = [3;0];
    C = [4 0];  D = 2;
» sistema = ss(A,B,C,D)
a =
     x1  x2
 x1   0  -1
 x2  -3   2
b =
     u1
 x1   3
 x2   0
c =
     x1  x2
 y1   4   0
d =
     u1
 y1   2
Continuous-time model.
```

Esempi (2)

- Definizione del sistema dinamico a tempo discreto:

$$\begin{cases} x_1(k+1) = -x_2(k) + 3u(k) \\ x_2(k+1) = -3x_1(k) + 2x_2(k) \\ y(k) = 4x_1(k) + 2u(k) \end{cases}$$

```
» A = [ 0 -1; -3  2]; B = [3;0];
    C = [4 0];  D = 2;

» sistema = ss(A,B,C,D,-1)
a =
     x1  x2
  x1  0  -1
  x2 -3   2
b =
     u1
  x1  3
  x2  0
c =
     x1  x2
  y1  4   0
d =
     u1
  y1  2
Sampling time: unspecified
Discrete-time model.
```

Sistemi lineari SISO: dalla FdT

- partendo dalla **funzione di trasferimento**
  - assegnare i coefficienti dei polinomi a numeratore e denominatore della fdt nel workspace (nel seguito vettori **NUM** e **DEN**);
  - definire il sistema mediante il comando ***tf***
  - **Sintassi** del comando ***tf***
    - ***SYS = tf(NUM,DEN)*** crea un sistema a tempo continuo
    - ***SYS = tf(NUM,DEN,Ts)*** crea un sistema a tempo discreto con intervallo di campionamento specificato da Ts [s].
    - Ponendo **Ts** pari a **-1**, si lascia non specificato l'intervallo di campionamento associato al sistema.

Esempi (3)

- Definizione del sistema tramite la funzione di trasferimento

$$G(s) = \frac{s+1}{s^2+3s+16}$$

```
» num = [ 1 1 ]; den = [ 1 3 16 ];
» sistema = tf( num, den )
 Transfer function:
   s + 1
-------------
s^2 + 3 s + 16
```

Esempi (4)

- Definizione del sistema tramite la funzione di trasferimento

$$G(z) = \frac{z+1}{z^2+3z+16}$$

```
» num = [ 1 1 ]; den = [ 1 3 16 ];
» sistema = tf( num, den,-1 )
 Transfer function:
   z + 1
-------------
z^2 + 3 z + 16
```

Sistemi lineari SISO: dalla FdT (2)

- partendo dalla **funzione di trasferimento**
  - assegnare i vettori degli zeri, dei poli ed il guadagno del sistema nel workspace (nel seguito vettori **Z**, **P** e **K**);
  - definire il sistema mediante il comando ***zpk***.
  - **Sintassi** del comando ***zpk***
    - ***SYS = zpk(Z,P,K)*** crea un sistema a tempo continuo
    - ***SYS = zpk(Z,P,K,Ts)*** crea un sistema a tempo discreto con intervallo di campionamento specificato da Ts [s].
    - Ponendo **Ts** pari a **-1**, si lascia non specificato l'intervallo di campionamento associato al sistema.

Esempi (5)

- Definizione del sistema tramite la funzione di trasferimento

$$G(s) = -5\frac{s-1}{s+1}$$

```
» Z = [ 1 ]; P = [ -1]; K = [-5]
» sistema = zpk( Z,P,K )
» Zero/pole/gain:
-5 (s-1)
--------
 (s+1)
```

Esempi (6)

- Definizione del sistema tramite la funzione di trasferimento

$$G(z) = -5\frac{z-1}{z+1}$$

```
» Z = [ 1]; P = [ -1]; K = [-5]
» sistema =zpk( Z, P, K ,-1)
 Zero/pole/gain:
-5 (z-1)
--------
 (z+1)
 Sampling time: unspecified
```

Simulazione in Matlab di sistemi lineari

- Funzioni disponibili per la simulazione:
  - ***impulse*** ⇒ simulazione risposta all'impulso;
  - ***step*** ⇒ simulazione risposta a scalino;
  - ***initial*** ⇒ simulazione movimento libero;
  - ***lsim*** ⇒ simulazione con ingresso qualsiasi e stato iniziale qualsiasi.

Simulazione in Matlab di sistemi lineari (2)

- Sintassi:

```
» [y,t]=step(sistema);
» [y,t]=step(sistema,t);
```

Vettore dei tempi

Vettore d'uscita

Vettore sequenza d'ingresso

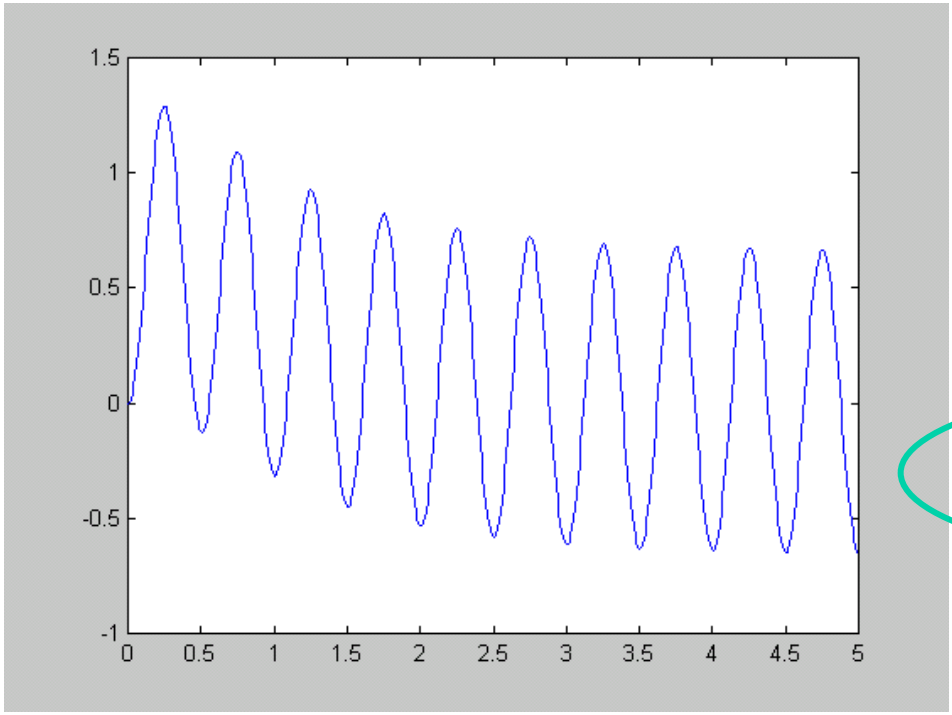
```
» [y,x]=lsim(sistema,u,t);
```

Vettore d'uscita

Vettore di stato

Vettore dei tempi

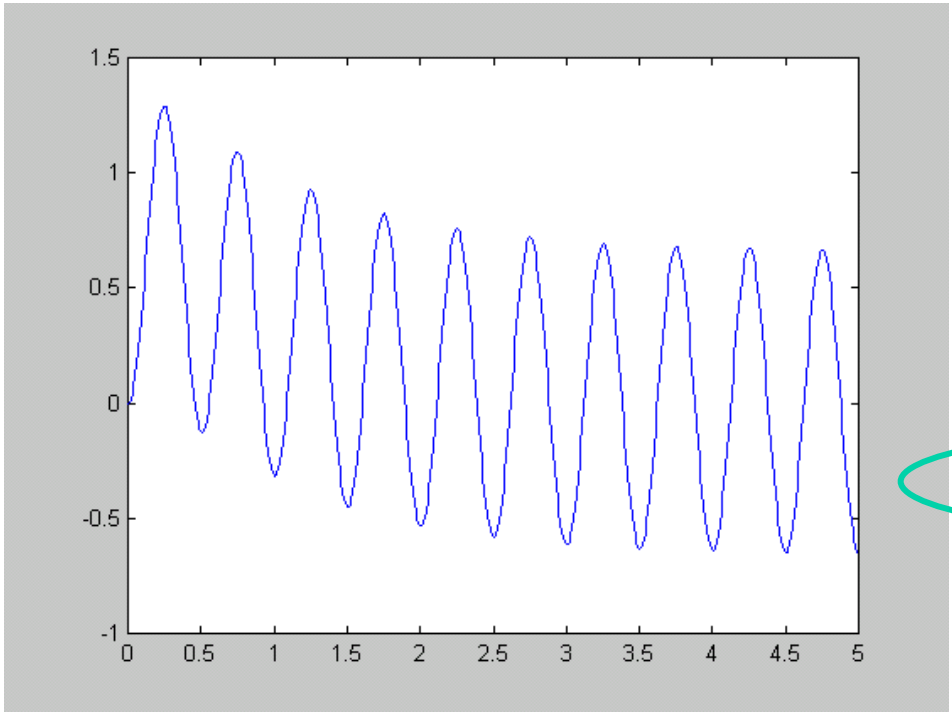
Esempio di utilizzo (1)




```
» a=[-1 ,0;3,-4];
» b=[2;1];c=[1,2];d=0;
» sistema=ss(a,b,c,d);
» t=(0:0.01:5);
» u=2*sin(2*pi*2*t);
» y=lsim(sistema,u,t);
» plot(t,y);
```

Il risultato della simulazione è stato assegnato ad una variabile e successivamente visualizzato in un grafico.

Esempio di utilizzo (2)




```
» a=[-1 ,0;3,-4];
» b=[2;1];c=[1,2];d=0;
» sistema=ss(a,b,c,d);
» t=(0:0.01:5);
» u=2*sin(2*pi*2*t);
» lsim(sistema,u,t);
```

Utilizzando le funzioni senza assegnare il risultato della simulazione a variabili d'uscita si ottiene direttamente il grafico dell'evoluzione temporale.

Esempi di utilizzo (3)

- Analisi della risposta allo scalino unitario:

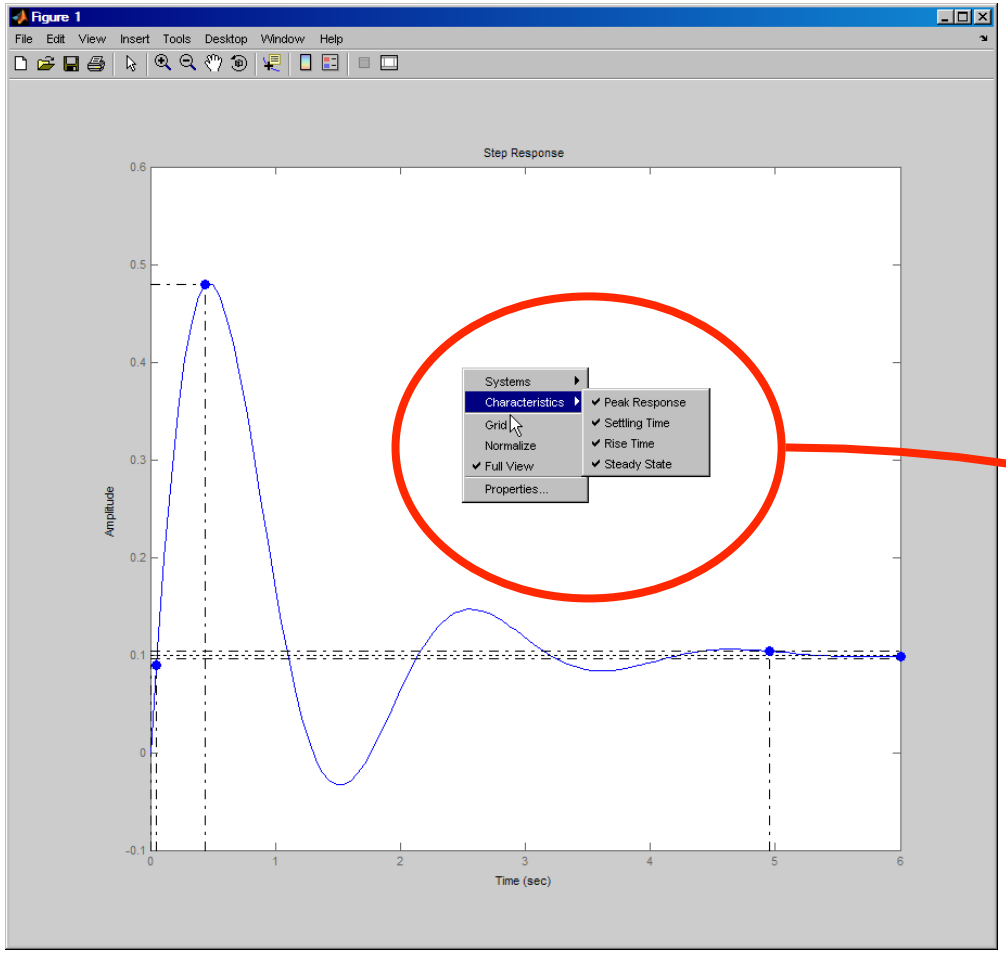
» step(sistema);

- Interagendo tramite il mouse nella finestra che visualizza l'andamento della risposta allo scalino è possibile ottenere informazioni relative a
  - Sovraelongazione della risposta
  - Valore di regime
  - Tempo di salita (*rise time*)
  - Tempo di assestamento (*settling time*)

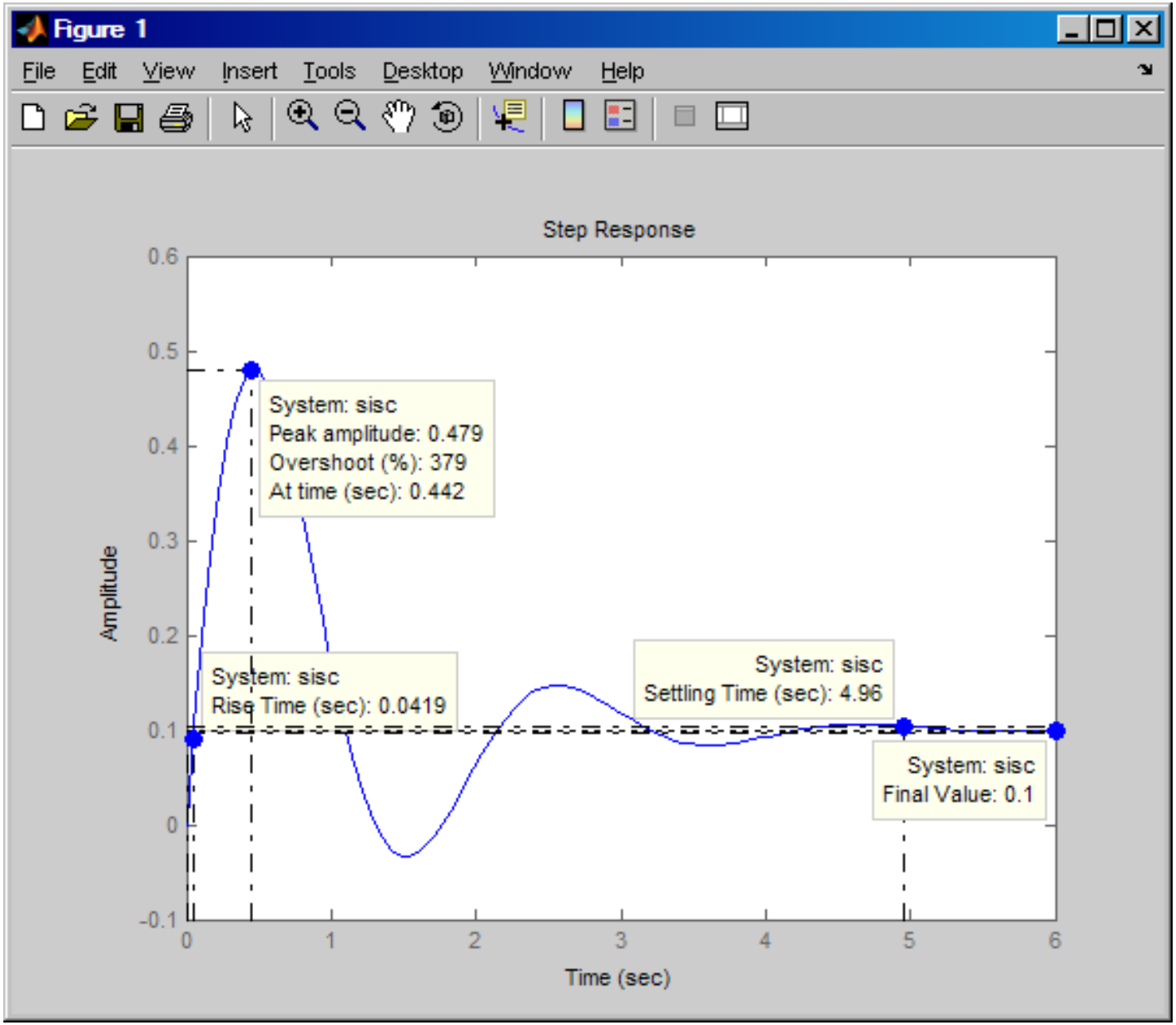
Esempio: analisi della risposta allo scalino

- Il codice Matlab dell'esempio:

```
» % sistema a tempo continuo
» sisc = tf([2 1],[1 2 10]);
»
» % sistema a tempo discreto
» sisd = tf([2 1],[1 0.2 0.5], 0.1);
»
» figure; step(sisc);
» figure;step(sisd);
```

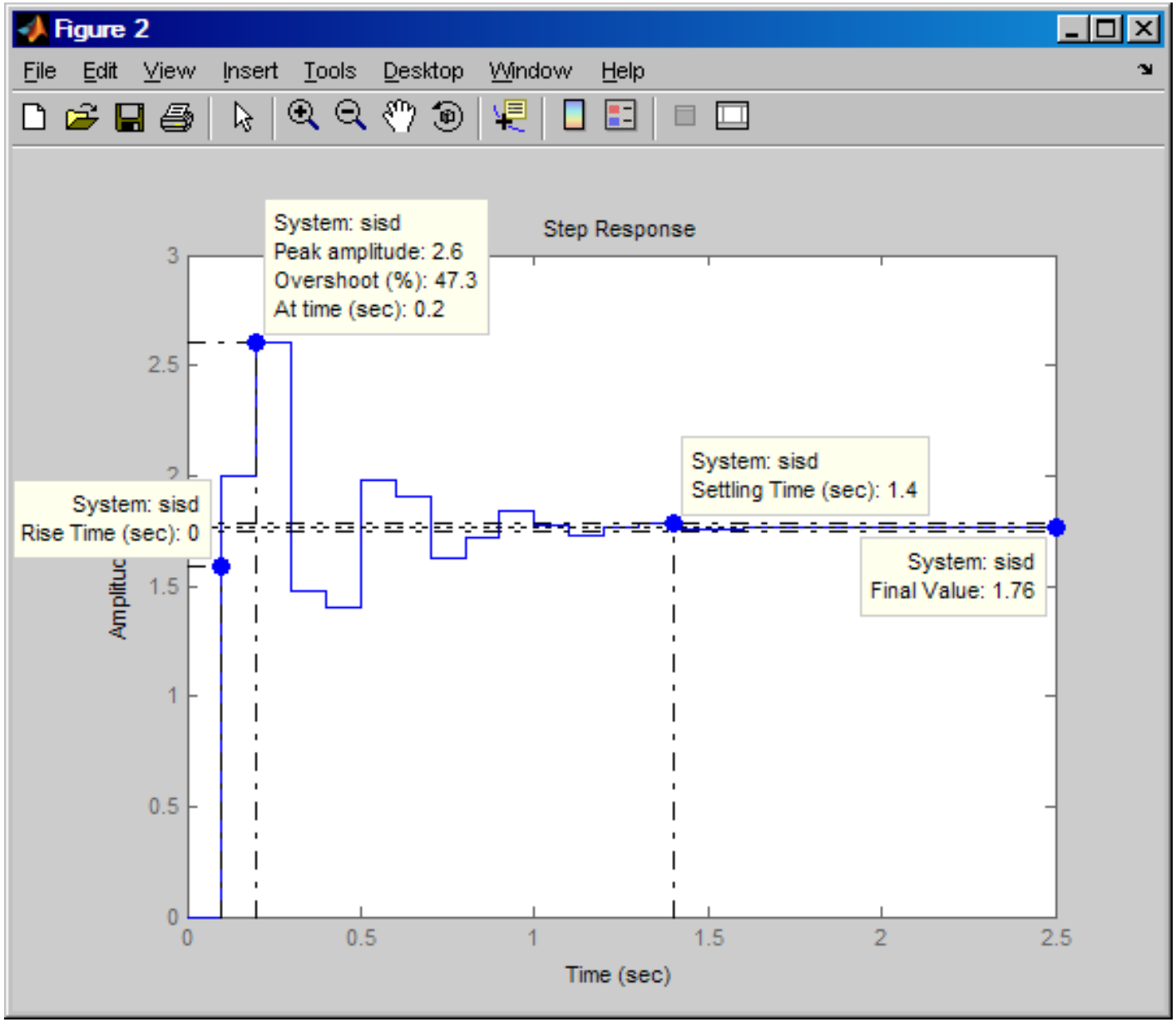


A seguito di un "click" del pulsante destro del mouse compare un menu ...



Esempio: risposta del sistema a tempo continuo.

Esempio: risposta del sistema a tempo discreto.



Considerazioni riassuntive

- Come tutti i risultati di **operazioni di calcolo numerico**, anche i sistemi LTI, descritti con le istruzioni Matlab viste, possono essere affetti da **errori** (errata o mancante cancellazione di termini nella FdT ecc. ...).

- Si vedano l'argomento "**Reliable Computations**" e l'argomento "**Choice of LTI Model**" nella documentazione del **Control Toolbox**.

Sistemi lineari interconnessi

Definizioni, proprietà, applicazioni

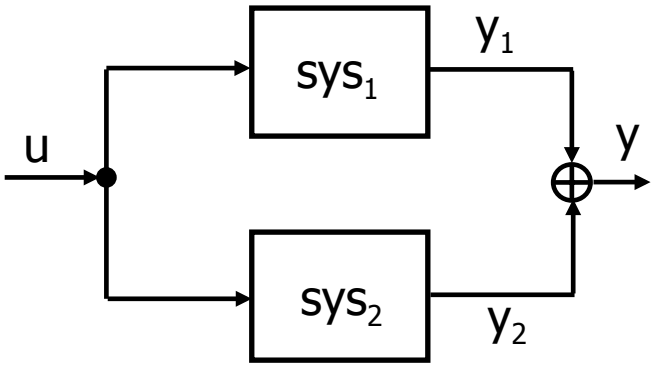
Interconnessione di sistemi

- Agli *oggetti* sistemi lineari si possono applicare i normali operatori **+**,*,**/**, **** con il seguente significato:
  - **+** connessione in **parallelo**;
  - ***** connessione in **serie**;
  - **/, ** usati per definire l'operazione di inversione (a sx, a dx) per **sistemi quadrati**.

Esempi di interconnessioni elementari

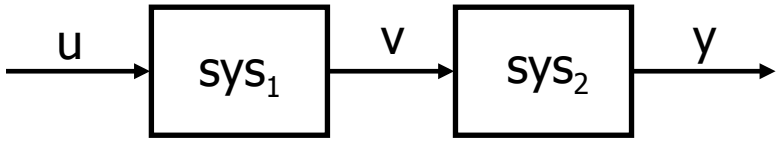
- **Connessione parallelo**

$$sys = sys_1 + sys_2$$



- **Connessione serie**

$$sys = sys_1 \cdot sys_2$$



Altri esempi di operazioni elementari

- **Inversione** (per sistemi quadrati)

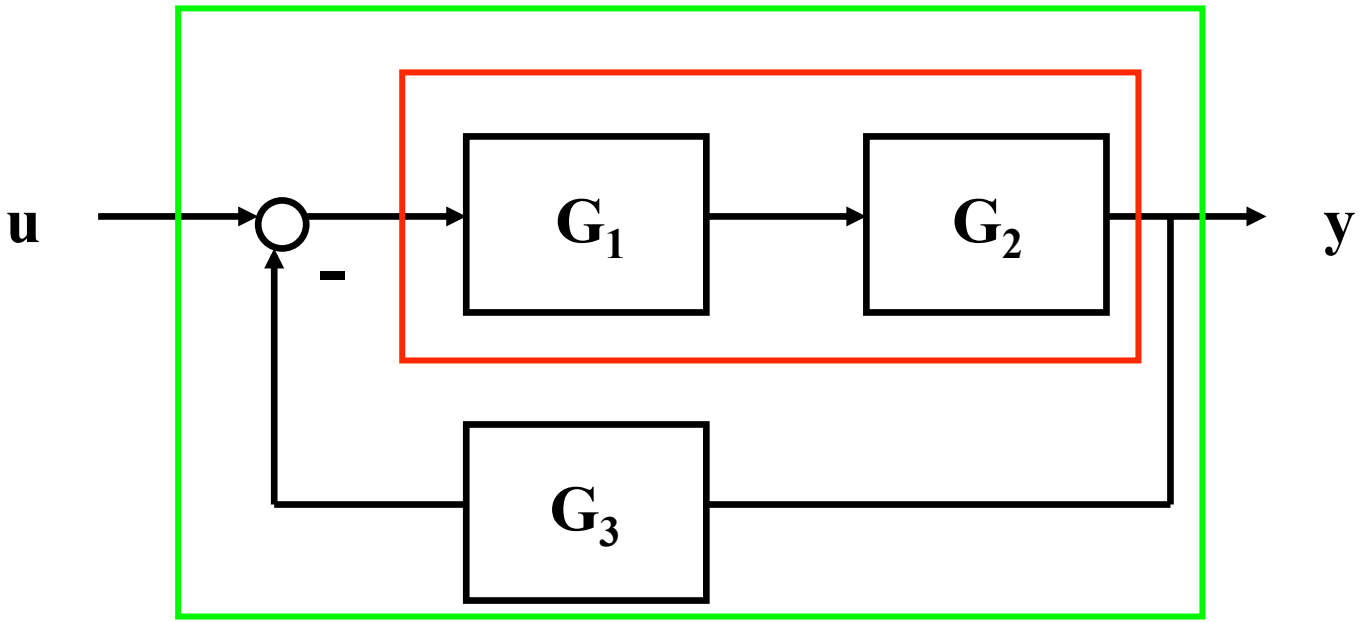
$$sys_1 \setminus sys_2 \quad \Leftrightarrow \quad inv(sys_1) \cdot sys_2$$

$$sys_1 / sys_2 \quad \Leftrightarrow \quad sys_1 \cdot inv(sys_2)$$

- **Trasposizione**

$$sys^T \Leftrightarrow sys.'$$

Esempio di connessione



andata = g1*g2; retroazione = andata/(1+andata*g3)

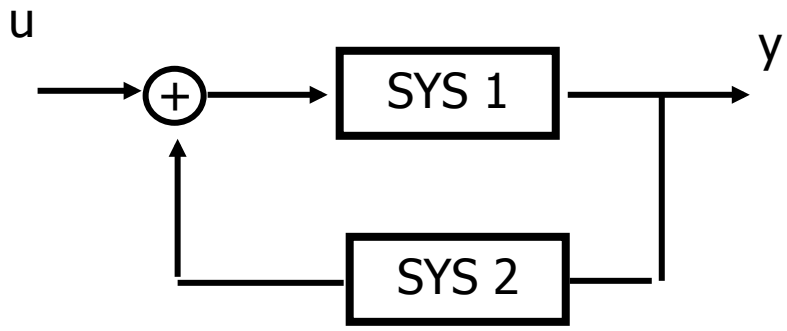
Interconnessioni tra sistemi

- Funzioni che permettono di descrivere interconnessioni tra sistemi dinamici:
  - concatenazione **,** (orizz.) **;** (vert.)
  - connessione diagonale a blocchi **append**
  - connessione parallelo di due blocchi **parallel**
  - connessione serie di due blocchi **series**
  - connessione in retroazione **feedback**
  - connessione generica **connect**

Sintassi del comando feedback

Sintassi semplice del comando

SYS = FEEDBACK(SYS1,SYS2) fornisce il sistema LTI corrispondente al semplice ciclo di reazione seguente

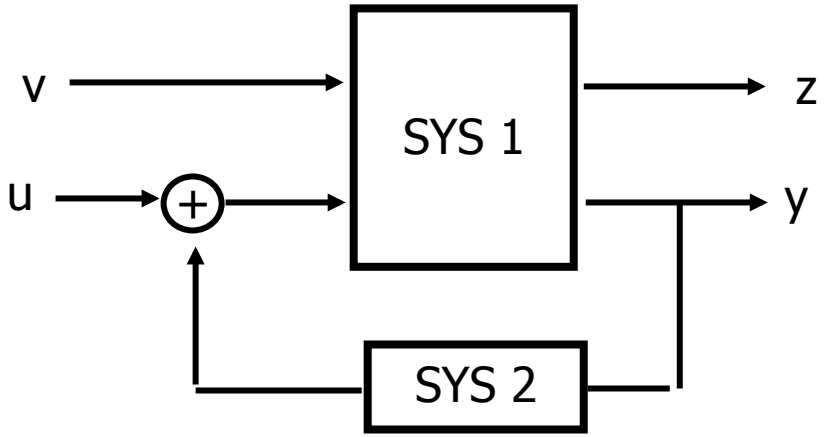


Si presuppone che sia uno schema a **retroazione negativa**. Nel caso in cui si voglia una retroazione positiva, va specificato nel comando

SYS = FEEDBACK(SYS1,SYS2,+1).

Sintassi completa

SYS = FEEDBACK(SYS1,SYS2,FEEDIN,FEEDOUT,SIGN) fornisce il sistema LTI corrispondente alla struttura:



I vettori **FEEDIN** e **FEEDOUT** contengono rispettivamente gli indici degli ingressi e delle uscite del sistema SYS1 coinvolti nella retroazione. La variabile **SIGN** indica se si tratta di retroazione positiva (**SIGN** pari a +1) oppure negativa (**SIGN** pari a –1 oppure non assegnato).

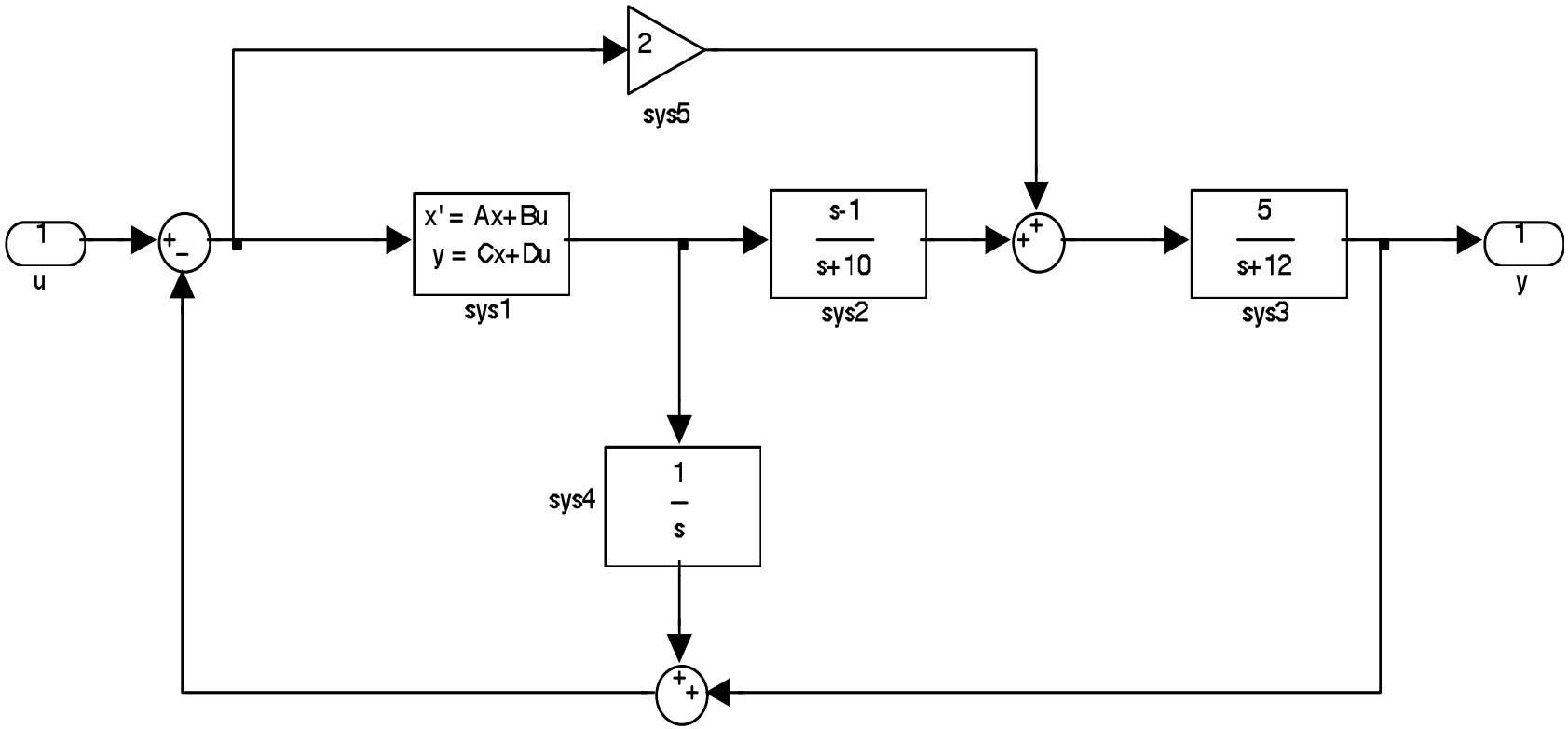
Sintassi del comando connect

L'istruzione **CONNECT** determina un **modello in equazioni di stato** per un sistema descritto da uno schema a blocchi.

SYSc = CONNECT(SYS,Q,INPUTS,OUTPUTS) fornisce un modello in equazioni di stato per il sistema **SYSc**, descritto facendo uso di un **sistema ausiliario SYS** (un sistema MIMO costituito da sottosistemi non interconnessi) che contiene la descrizione di tutti i sottosistemi che compaiono nello schema a blocchi in esame e dalla **matrice di interconnessione Q**. Questa matrice viene costruita in maniera particolare per descrivere come i vari sottosistemi siano connessi tra loro.

Infine i vettori **INPUTS** e **OUTPUTS** indicano quali ingressi e rispettivamente quali uscite dei vari sottosistemi componenti diventano ingressi ed uscite del sistema complessivo SYSc.

Esempio: sistemi interconnessi



Esempio: sistemi interconnessi (2)

- Il sottosistema descritto su base stato è descritto dalle matrici:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -10 & -1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

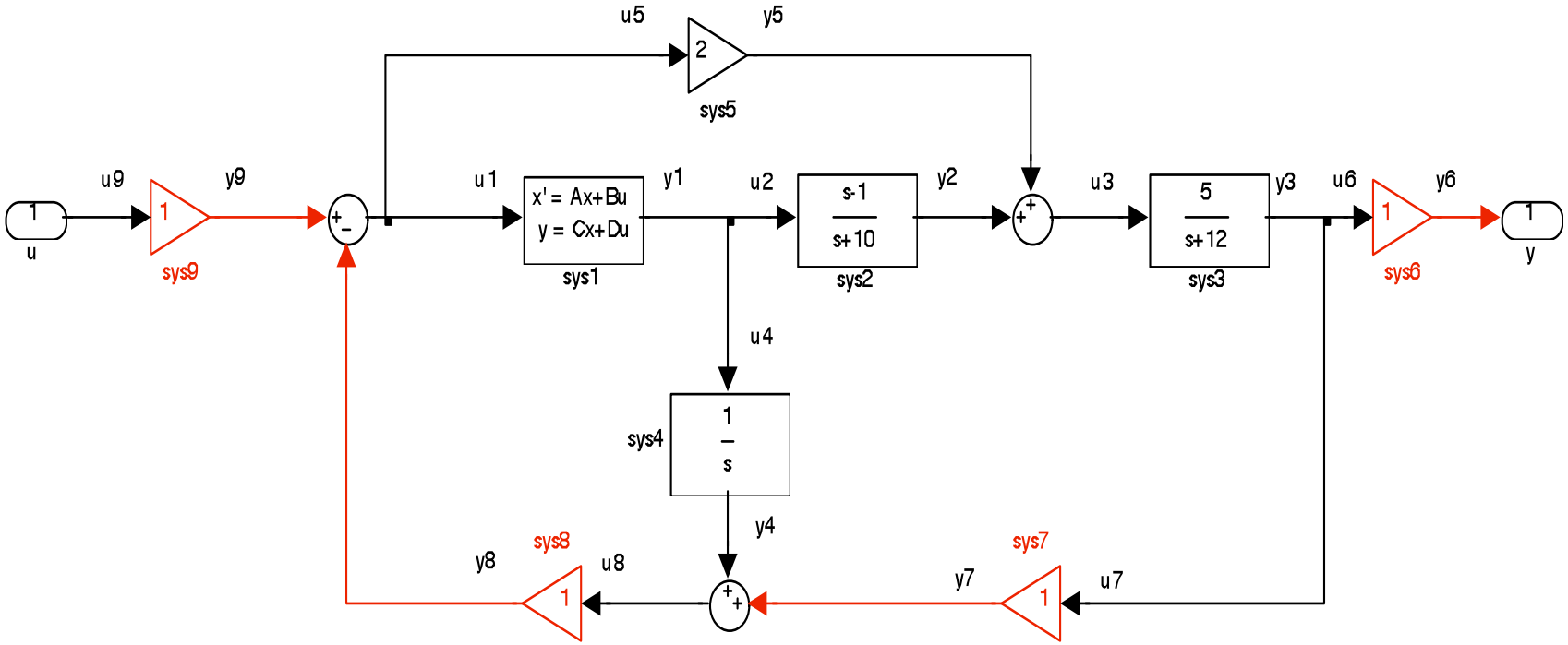
- La funzione di trasferimento del sistema è

$$G_1(s) = \frac{s}{s^2 + s + 10}$$

Esempio: sistemi interconnessi (3)

- Per poter utilizzare l'istruzione **connect** è necessario "etichettare" tutti i rami dello schema a blocchi, evidenziando per ogni sottosistema (dinamico e non) i corrispondenti segnali d'ingresso e di uscita

Esempio: sistemi interconnessi (4)



ecco il sistema dopo l'etichettatura

Esempio: sistemi interconnessi (5)

- Ulteriori passi
  - Assemblaggio di tutti i sottosistemi in un'unica entità, tramite l'istruzione **append** (NB sono ancora sistemi non interconnessi, ma ora sono ordinati!), creando così il sistema ausiliario **SYS**
  - Creazione della matrice di connessione **Q**, secondo lo schema:

Esempio: sistemi interconnessi (5)

- Utilizzare l'istruzione **connect** per ottenere una descrizione **su base stato** del sistema complessivo.

- **Svantaggi**:
  - macchinosa procedura di descrizione delle interconnessioni.

- **Vantaggi**:
  - possibilità di ottenere la realizzazione del sistema complessivo nella forma desiderata (l'ordine di comparizione delle varie variabili di stato è scelto nella fase di "aggregazione" dei sottosistemi con l'istruzione **append**).

Analisi della risposta in frequenza

Risposta in frequenza per sistemi LTI a tempo continuo

- **risposta in frequenza** di un sistema LTI asintoticamente stabile a tempo continuo

$$F(\omega) = G(s)|_{s=j\omega}$$

dove $G(s)$ è la fdt del sistema.

Risposta in frequenza per sistemi LTI a tempo discreto

- **risposta in frequenza** di un sistema LTI asintoticamente stabile a tempo discreto

$$F(\vartheta) = G(z)|_{z=e^{j\vartheta}}$$

dove $G(z)$ è la fdt del sistema.

Grafici della risposta in frequenza

- Comandi Matlab
  - **bode** : rappresentazione della risposta in frequenza tramite **diagrammi di Bode**
  - **nyquist** : rappresentazione della risposta in frequenza tramite **diagramma di Nyquist**
  - **nichols** : rappresentazione della risposta in frequenza tramite **diagramma di Nichols**

Calcolo ed analisi della risposta in frequenza

- Comandi Matlab
  - **freqresp** : **calcolo** di valori della risposta in frequenza di un sistema dinamico;
  - **ltiview** : strumento interattivo che permette **l'analisi** delle risposte nel tempo ed in frequenza di sistemi dinamici LTI.

Sintassi dei comandi (1)

- Sintassi generale dei comandi ***bode***, ***nyquist***, ***nichols***:

  - ***comando(SYS)*** crea il diagramma voluto della risposta in frequenza del sistema ***SYS***, creato tramite le istruzioni ***ss***,***tf***,***zpk***. L'intervallo di frequenza ed il numero di punti utilizzati per i diagrammi sono scelti in modo automatico.

  - ***comando(SYS,{WMIN,WMAX})*** crea il diagramma per le frequenze comprese tra ***WMIN*** and ***WMAX*** [rad/s].

Sintassi dei comandi (2)

- ***comando(SYS,W)*** utilizza il vettore ***W*** specificato dall'utente come vettore delle frequenze (in *rad/s*) per le quali determinare la risposta in frequenza.

- ***[a,b]=comando(SYS,...)*** non visualizza alcun grafico, ma restituisce valori della risposta in frequenza, utilizzabili per tracciare successivamente il diagramma voluto.

  - ***[m,f]=bode(...)*** restituisce modulo e fase (in gradi)
  - ***[m,f]=nichols(...)*** restituisce modulo e fase (in gradi)
  - ***[r,i]=nyquist(...)*** restituisce parte reale ed immaginaria dei punti della risposta in frequenza

Sintassi dei comandi (3)

- Per sistemi LTI *a tempo discreto* con periodo di campionamento $\mathbf{T_s}$ viene utilizzata la relazione

$$z = e^{j\Omega T_s}$$

per mappare la circonferenza unitaria con centro l'origine (in *z*) tramite la pulsazione Ω [rad/s] e la pulsazione massima che compare nei grafici è pari a

$$\Omega_{\text{max}} = \frac{\pi}{T_s} = \Omega_N = \frac{\Omega_S}{2}$$

Se $\mathbf{T_s}$ non è assegnato, viene assunto un periodo di campionamento pari ad ***1 s***.

Sintassi dei comandi (4)

- Calcolo della risposta in frequenza

  – ***H = freqresp(SISTEMA,W)*** calcola la risposta in frequenza ***H*** del sistema dinamico LTI descritto da ***SISTEMA*** in corrispondenza delle pulsazioni assegnate nel vettore ***W***. Queste pulsazioni sono espresse in *rad/s*.

  H è un array di valori complessi.

Sintassi dei comandi (5)

- Analisi nel tempo e in frequenza di un sistema LTI

  – ***ltiview(SISTEMA)*** apre una finestra grafica interattiva, nella quale viene visualizzata, come scelta predefinita, la risposta al gradino unitario di ***SISTEMA***, ma permette di analizzare sia l'evoluzione temporale che la risposta in frequenza di ***SISTEMA***.

Sintassi dei comandi (6)

- ***ltiview(grafico, SISTEMA)*** specifica quale grafico debba venire visualizzato. In particolare grafico può essere una delle seguenti espressioni (oppure una combinazione di esse)

  - ***'step'*** risposta al gradino unitario
  - ***'impulse'*** risposta all'impulso unitario
  - ***'bode'*** diagrammi di Bode
  - ***'bodemag'*** diagramma di Bode del modulo
  - ***'nyquist'*** diagramma di Nyquist
  - ***'nichols'*** diagramma di Nichols
  - ***'pzmap'*** mappa poli/zeri
  - altro ancora (si veda la sintassi completa del comando ...)

Esempi: diagrammi di Bode

```
» num=conv([1 0 4],[1 0 4]);
» den=[1 1 4 1 0 0];
» fdt=tf(num,den);
» figure;bode(fdt)
```

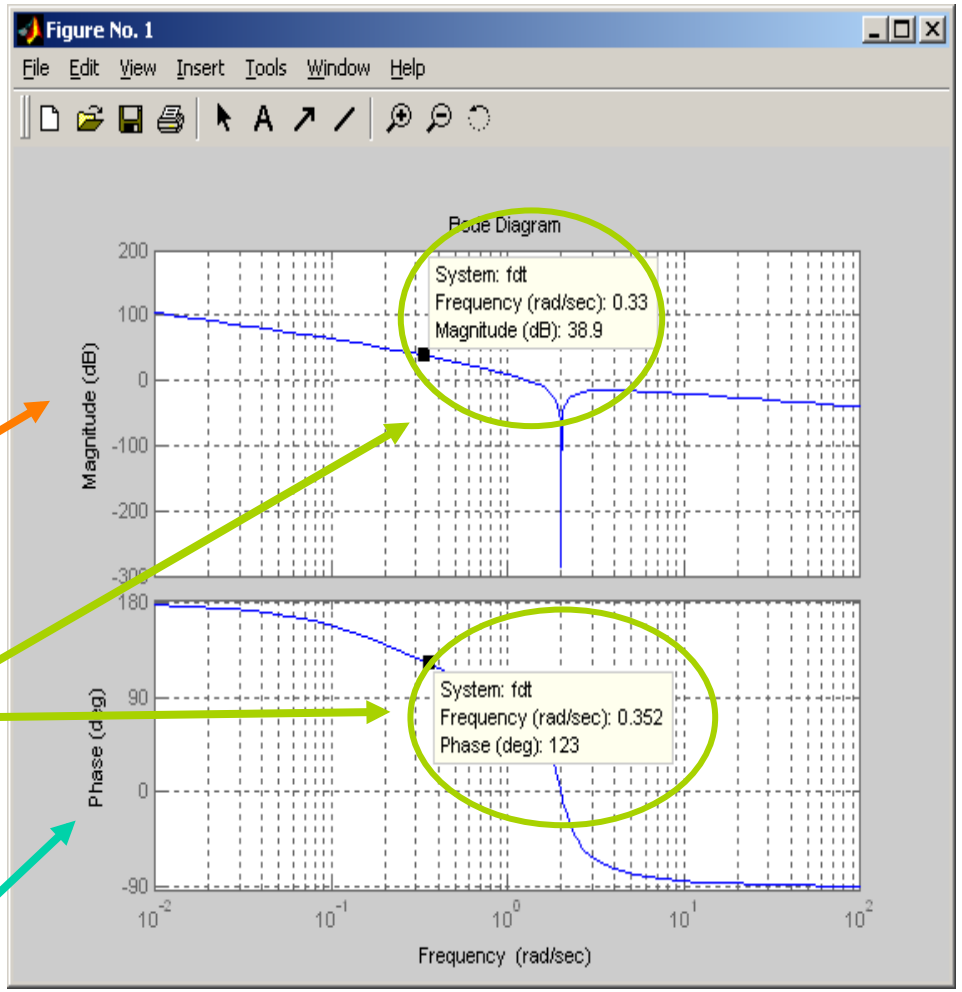


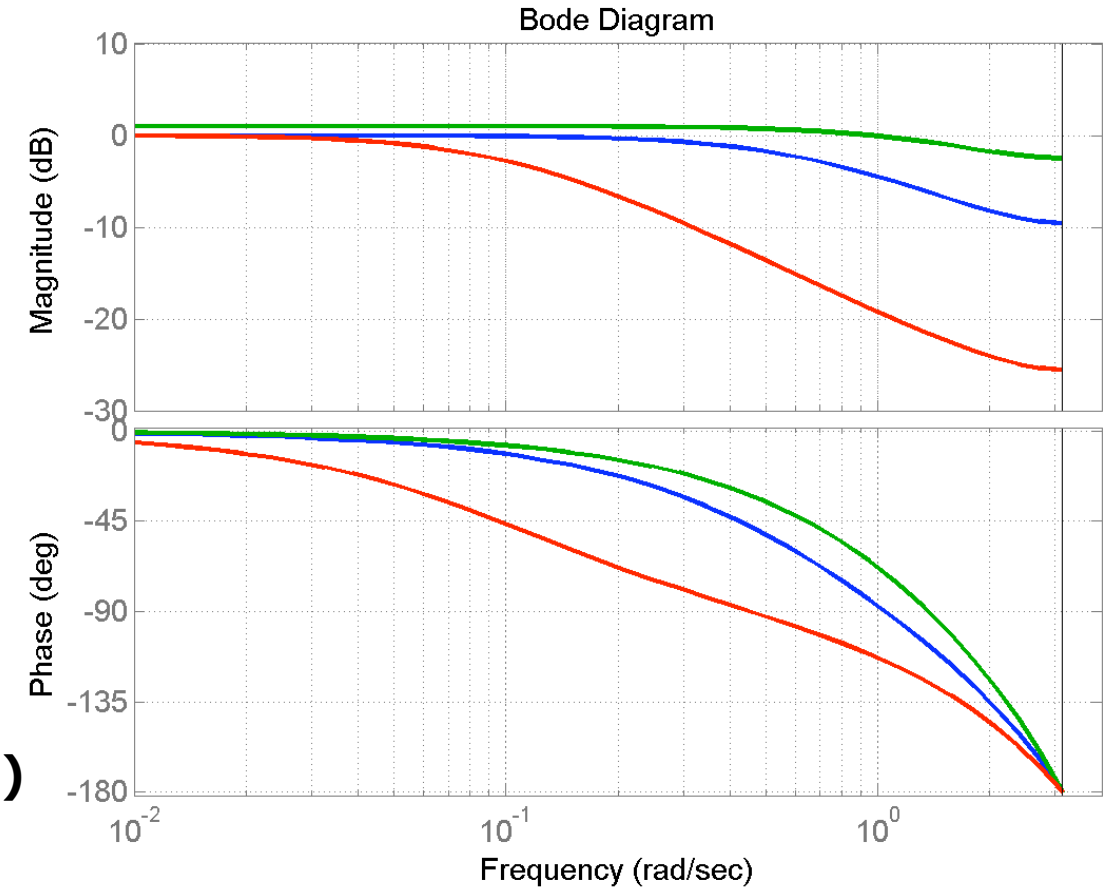
Diagramma del modulo

Esplorazione del diagramma con il mouse (pulsante sx premuto)

Diagramma della fase

Esempi: diagrammi di Bode

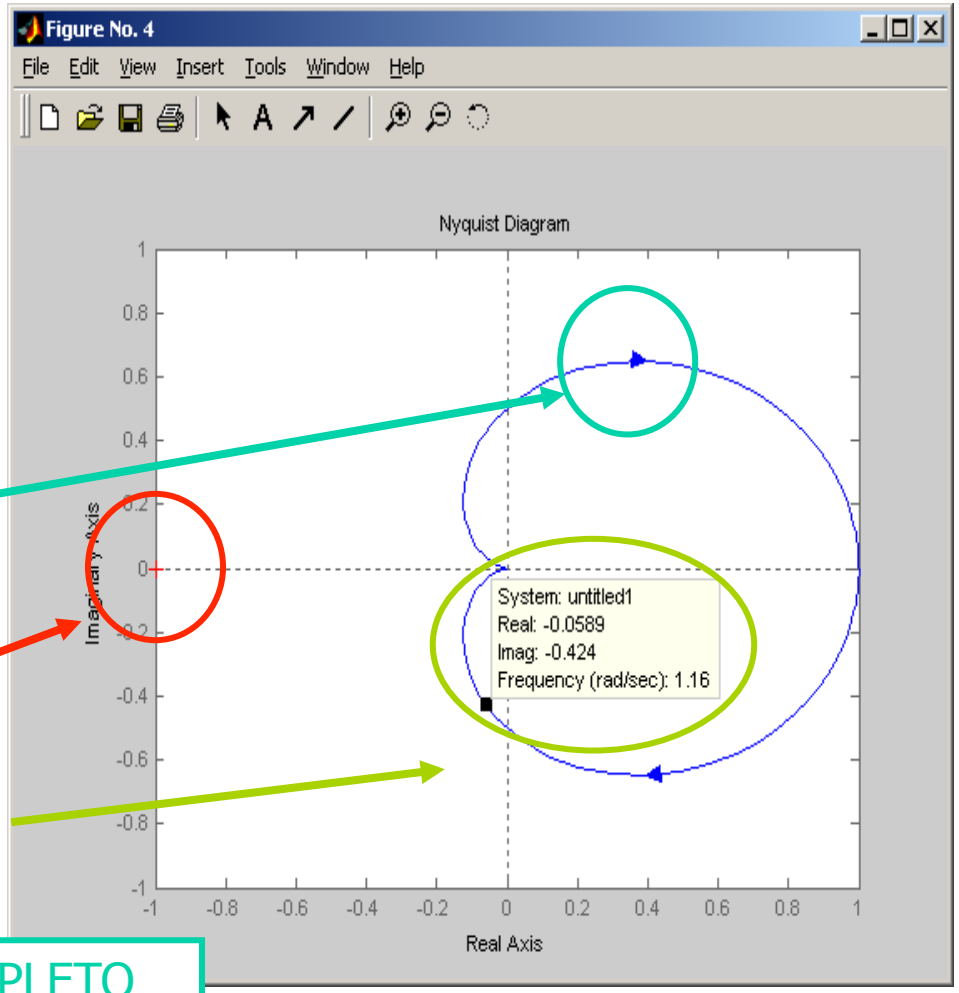
```
» num = [0.5]
» den=[1 -0.5];
» fdt1=tf(num,den,-1);
» num = [0.9]
» den=[1 -0.2];
» fdt2=tf(num,den,-1);
» num = [0.1]
» den=[1 -0.9];
» fdt3=tf(num,den,-1);
» figure;bode(fdt1,fdt2,fdt3)
```



Diagrammi di Bode per sistemi LTI a tempo discreto

Esempi: diagrammi di Nyquist

```
» num1=1
» den1=[1 2 1]
» figure;
» nyquist(tf(num1,den1))
```



Verso di percorrenza sul diagramma

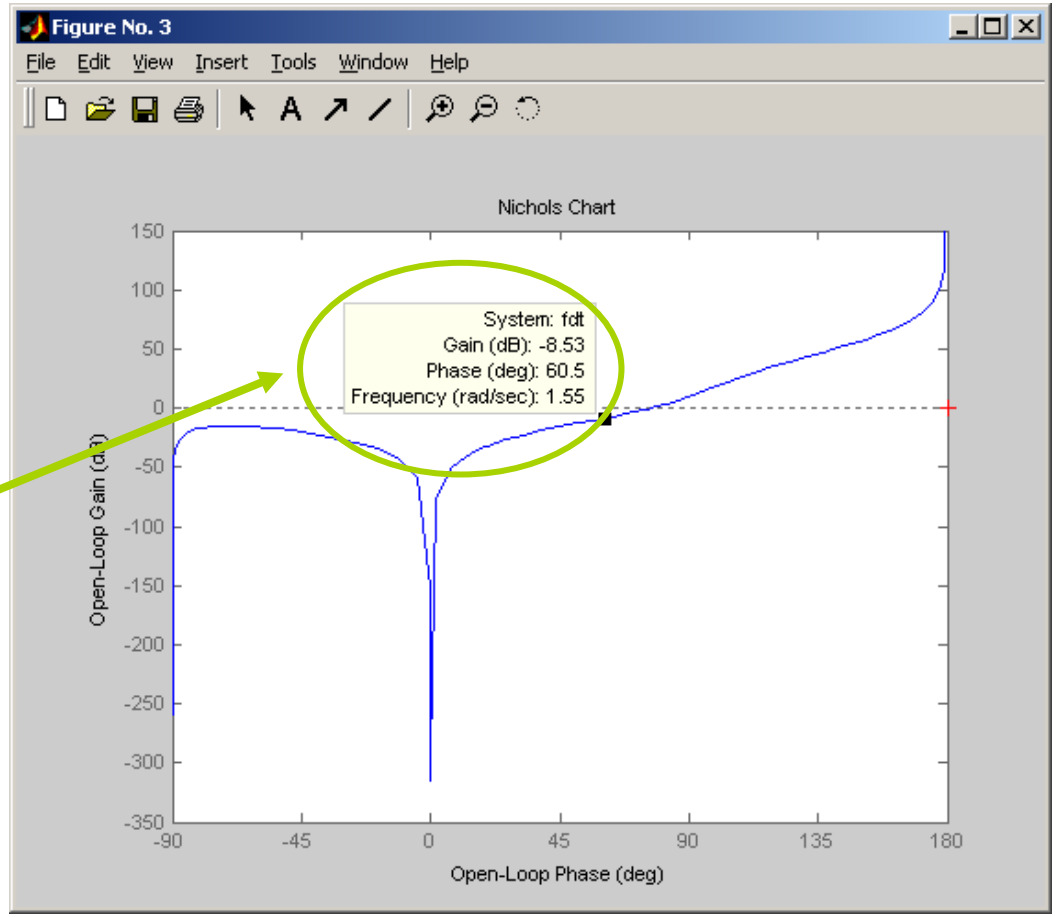
In evidenza il punto –1+j0

Esplorazione del diagramma con il mouse (pulsante sx premuto)

Diagramma di Nyquist COMPLETO

Esempi: diagrammi di Nichols

```
» num=conv([1 0 4],[1 0 4]);
» den=[1 1 4 1 0 0];
» fdt=tf(num,den);
» figure;nichols(fdt)
```



Esplorazione del diagramma con il mouse (pulsante sx premuto)

Diagrammi di Nyquist: casi particolari

- **Attenzione!** Il comando ***nyquist(sistema)*** traccia il diagramma completo di Nyquist della risposta in frequenza del sistema in esame.

- Ciò può portare ad una ***visualizzazione non ottimale*** dell'andamento del diagramma in condizioni particolari

Diagrammi di Nyquist: casi particolari

- nel caso di **sistema LTI a tempo continuo**
  - presenza di poli (semplici o multipli) nell'origine;
  - presenza di poli (semplici o multipli) immaginari puri;

- nel caso di **sistema LTI a tempo discreto**
  - Presenza di poli (semplici o multipli) complessi, di modulo unitario

Esempi: diagrammi di Nyquist sistemi a tempo continuo

$$F(s) = \frac{\left(s^2+4\right)^2}{s^2\left(s^3+s^2+4s+1\right)}$$

Zeri imm. puri

Dovrà essere F(jω) ! 0 per ω ! 2.

Polo doppio in 0

Dovrà essere F(jω) ! 0 per ω ! 1, poiché c'è un eccesso di poli su zeri.

Dovrà essere F(jω) ! 1 per ω ! 0 ed essendo doppio il polo nell'origine non ci sarà alcun asintoto nel diagramma di Nyquist.

Esempi: diagrammi di Nyquist sistemi a tempo continuo

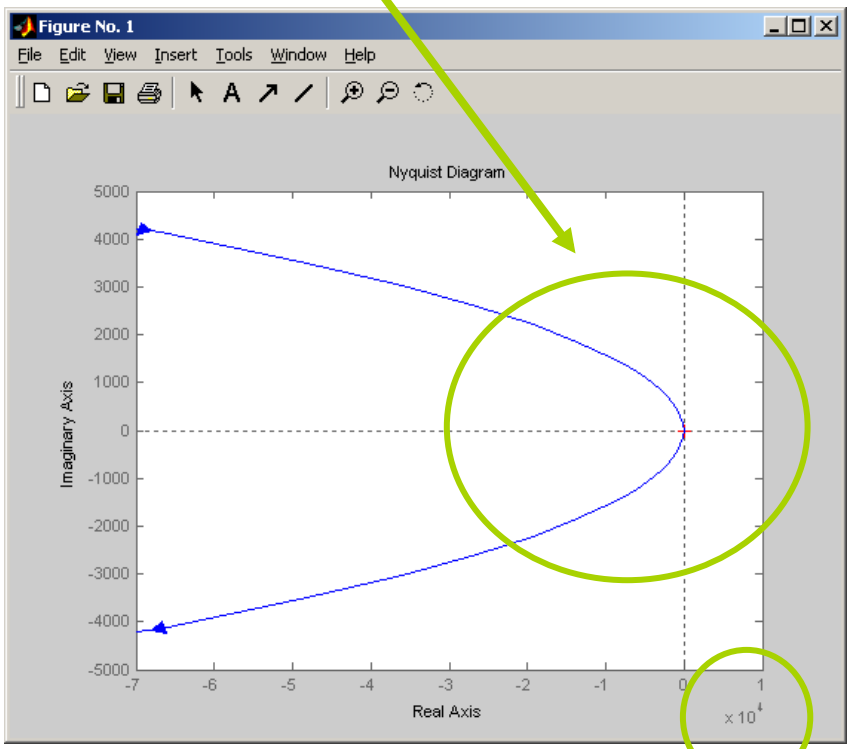
```
» num=conv([1 0 4],[1 0 4]);
» den=[1 1 4 1 0 0];
» fdt=tf(num,den);
» zpk(fdt)

Zero/pole/gain:
          (s^2  + 4)^2
-----------------------------------
s^2 (s+0.2627) (s^2  + 0.7373s +
    3.806)

» figure; nyquist(fdt);
```

Ed il comportamento locale per ω ! 1?
E per ω ! 2?

Zeri imm. puri

Polo doppio in 0

Diagramma di Nyquist COMPLETO



NB! 10^4

Esempi: diagrammi di Nyquist sistemi a tempo continuo

È possibile selezionare una modalità di presentazione del grafico che metta in evidenza la zona attorno al punto *(–1+j0)*

[mouse, pulsante dx premuto]

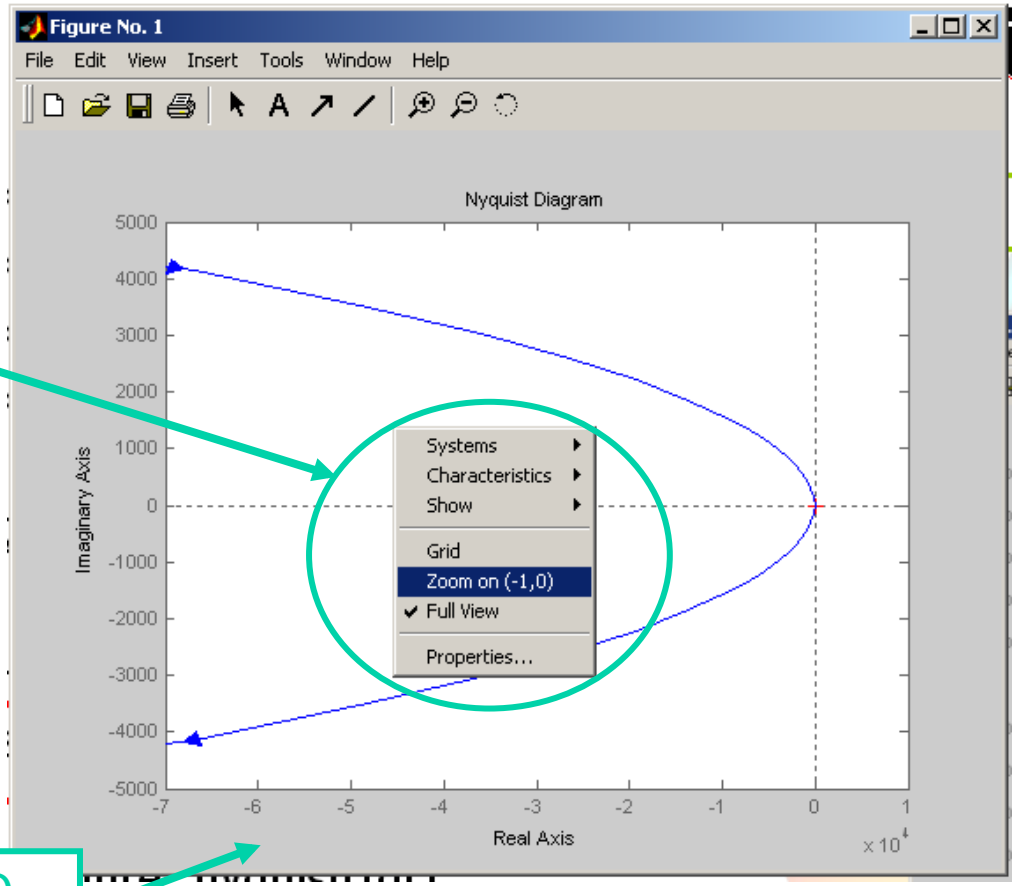


Diagramma di Nyquist COMPLETO

Esempi: diagrammi di Nyquist sistemi a tempo continuo

Ecco ciò che si ottiene.

In evidenza il punto –1+j0

Comportamento per ω ! 2

ed anche per ω ! 1

(si vede ancora poco …)

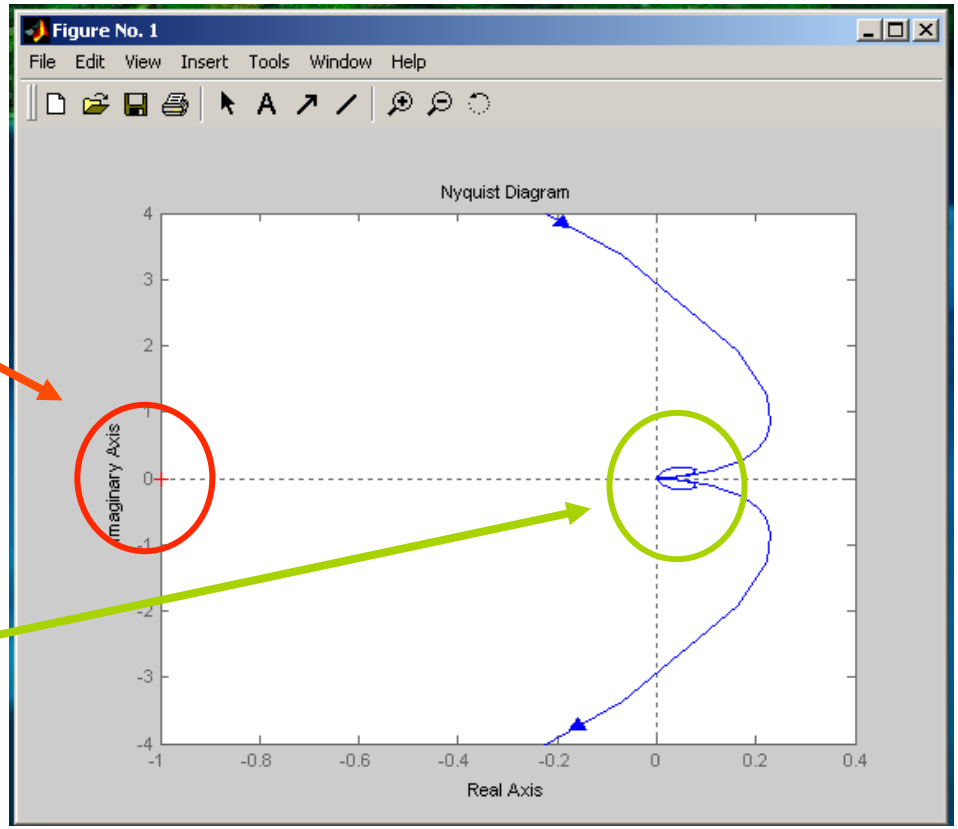


Diagramma di Nyquist: comportamento per ω ! 0

Esempi: diagrammi di Nyquist sistemi a tempo continuo

```
» figure;
» omega = linspace(1.5,200,10000);
» nyquist(fdt1,omega);
» axis([-0.1 0.4 -.4 0.4]);
» % cambia la visualizzazione
```

Selezione dell'intervallo di pulsazioni da utilizzare: $1.5 \cdot \omega \cdot 200$ rad/s.

In evidenza il ramo del diagramma che corrisponde a pulsazioni positive

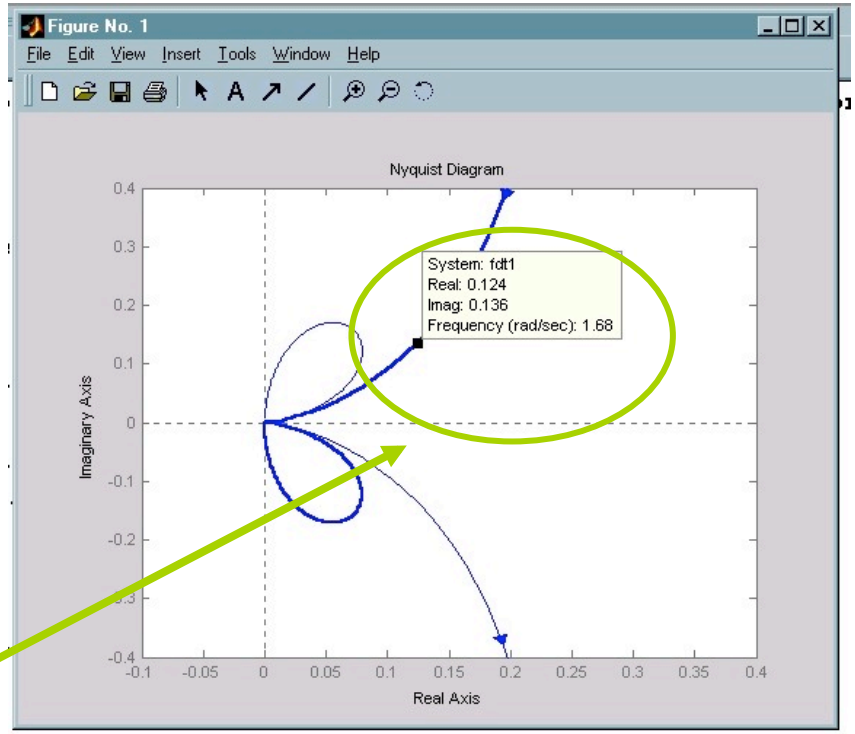


Diagramma di Nyquist ottenuto

Esempi: diagrammi di Nyquist sistemi a tempo continuo

```
» num=[10 20];
» den=conv([1 0 4],[1 1]);
» fdt2=tf(num,den)

Transfer function:
     10 s + 20
------------------
s^3 + s^2 + 4 s + 4
»  zpk(fdt2)

Zero/pole/gain:
   10 (s+2)
---------------
(s+1) (s^2  + 4)
```

$$F(s) = 10\,\frac{(s+2)}{(s+1)\left(s^2+4\right)}$$

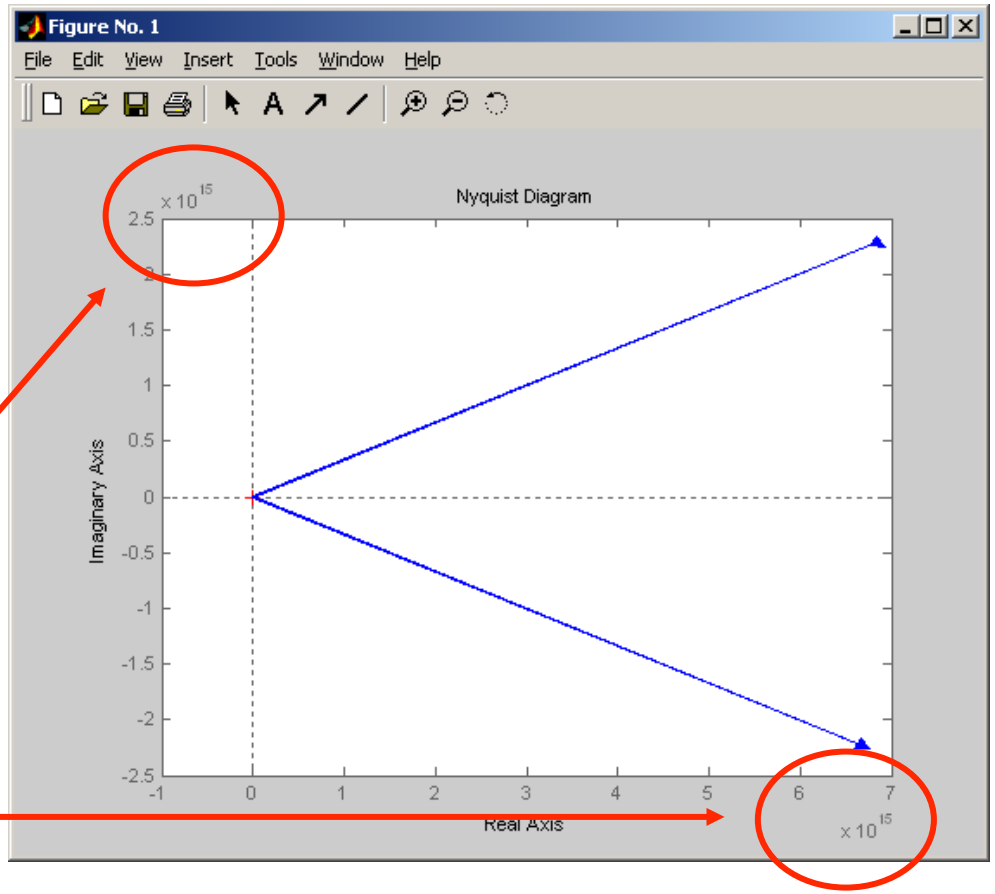
Poli immaginari puri e di molteplicità pari a 1: la curva presenta un asintoto per ω ! 2

Esempi: diagrammi di Nyquist sistemi a tempo continuo

```
» num=[10 20];
» den=conv([1 0 4],[1 1]);
» fdt2=tf(num,den)
» figure;nyquist(num,den)
```

Questo è ciò che si ottiene, ma è il grafico corretto?

NB! 10^{15}



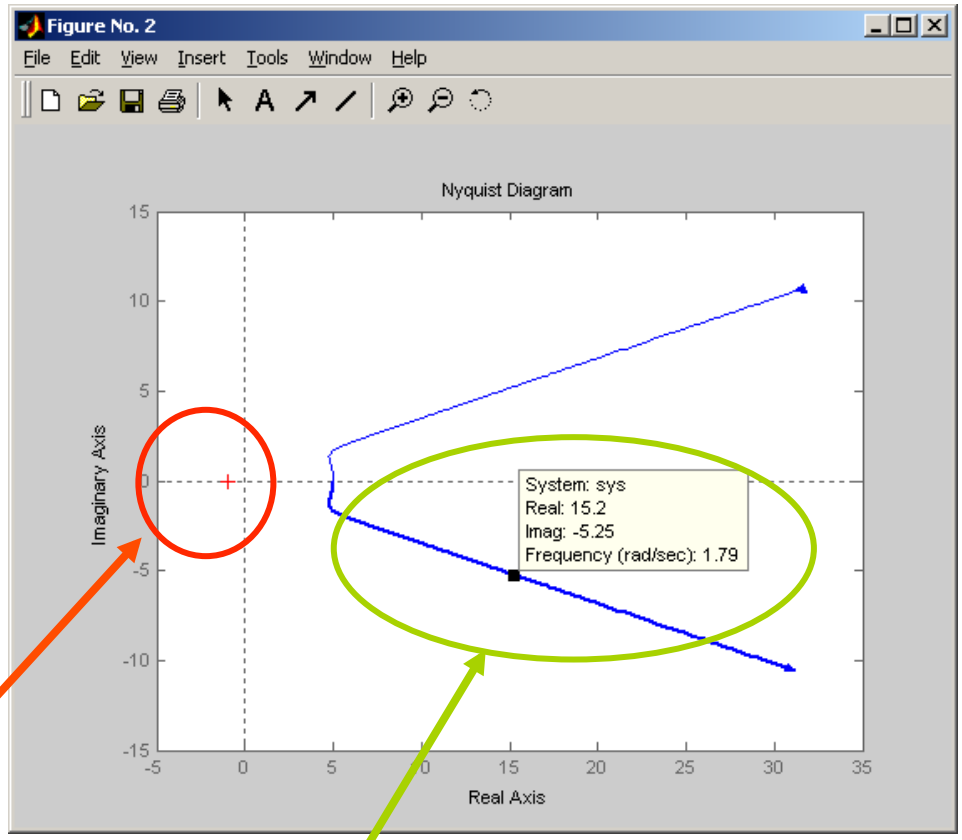
Esempi: diagrammi di Nyquist sistemi a tempo continuo

Studio per $\omega \cdot 2$

```
» num=[10 20];
» den=conv([1 0 4],[1 1]);
» fdt2=tf(num,den)
» figure;
»nyquist(num,den, ...
          linspace(0,1.95,1000))
```

Selezione dell'intervallo di pulsazioni da utilizzare: $0 \cdot \omega \cdot 1.95$ rad/s.

In evidenza il punto $-1+j0$



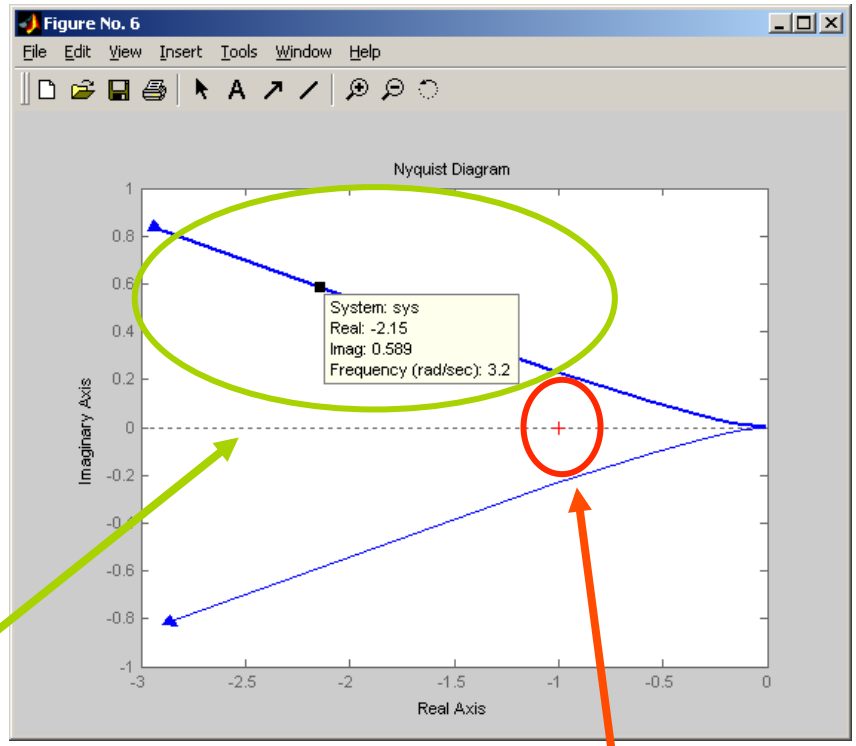
In evidenza il ramo corrispondente a pulsazioni ω ¸ 0

Esempi: diagrammi di Nyquist sistemi a tempo continuo

Studio per $\omega \geq 2$

```
» num=[10 20];
den=conv([1 0 4],[1 1]);
fdt2=tf(num,den)

figure;
nyquist(num,den,linspace(2.8,100.1000))
```

Selezione dell'intervallo di pulsazioni da utilizzare: $2.8 \cdot \omega \cdot 100$ rad/s.

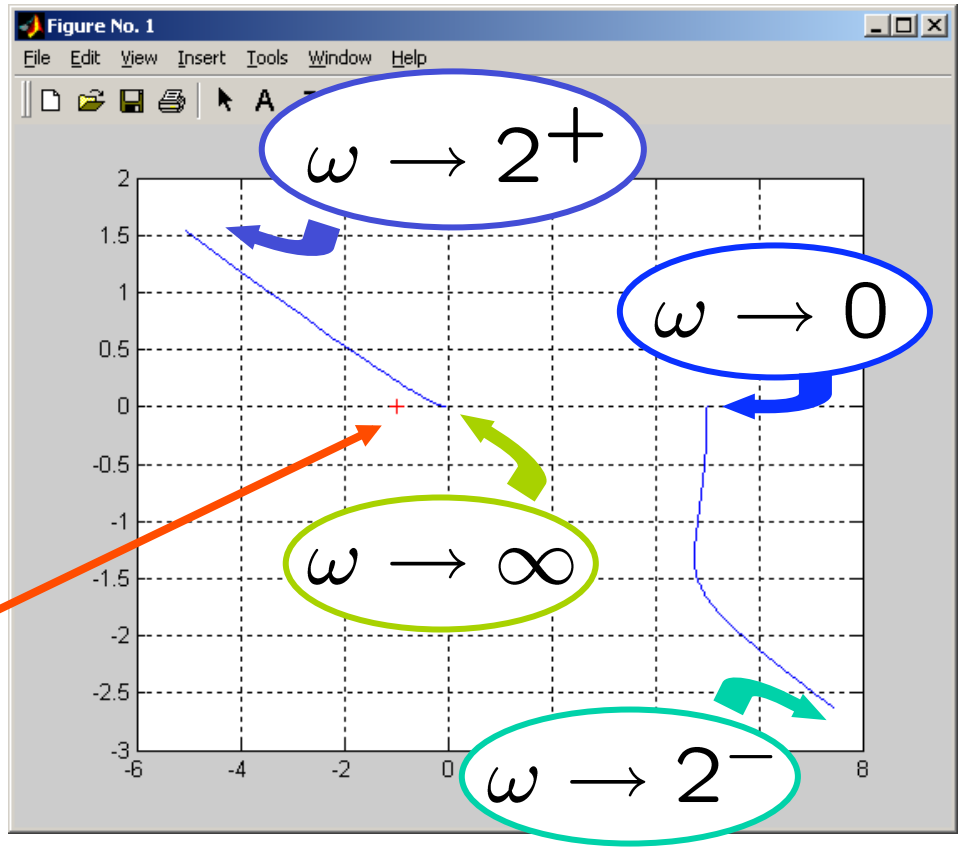


In evidenza il ramo corrispondente a pulsazioni $\omega \geq 0$

In evidenza il punto –1+j0

Esempi: diagrammi di Nyquist sistemi a tempo continuo

```
» num = [10 20];
» den = conv([1 0 4] , [1 1]);
» fdt2 = tf(num , den)
» % estrae i dati. NB sono vettori tridimensionali!!
» [ r1 i1] = nyquist(fdt2 , linspace(0 , 1.5 , 1000));
» [ r2 i2] = nyquist(fdt2 , linspace(2.5 , 100 , 1000));
» % ridiventano vettori bidimensionali
» % (obbligatorio per il comando PLOT)
» r1 = reshape(r1 , 1 , 1000); i1 = reshape(i1 , 1 , 1000);
» r2 = reshape(r2 , 1 , 1000); i2 = reshape(i2 , 1 , 1000);
» % diagramma di Nyquist della sola parte per
» % pulsazioni positive
» figure; plot(r1 , i1 , r2 , i2);
» hold on; plot(-1 , 0 , '+r'); grid on
» % con in evidenza il punto (-1+j0)
```



In evidenza il punto –1+j0

Diagramma esatto; soltanto $\omega \neq 0$

Esempi: diagrammi di Nyquist sistemi a tempo continuo

```
» num = [10 20];
» den = conv([1 0 4] , [1 1]);
» fdt2 = tf(num , den)
» % estrae i dati. NB sono vettori tridimensionali!!
» [ r1 i1] = nyquist(fdt2 , linspace(0 , 1.5 , 1000));
» [ r2 i2] = nyquist(fdt2 , linspace(2.5 , 100 , 1000));
» % ridiventano vettori bidimensionali
» % (obbligatorio per il comando PLOT)
» r1 = reshape(r1 , 1 , 1000); i1 = reshape(i1 , 1 , 1000);
» r2 = reshape(r2 , 1 , 1000); i2 = reshape(i2 , 1 , 1000);
» % diagramma di Nyquist della sola parte per
» % pulsazioni positive
» figure; plot(r1 , i1 , r2 , i2);
» hold on; plot(-1 , 0 , '+r'); grid on
» % con in evidenza il punto (-1+j0)
```

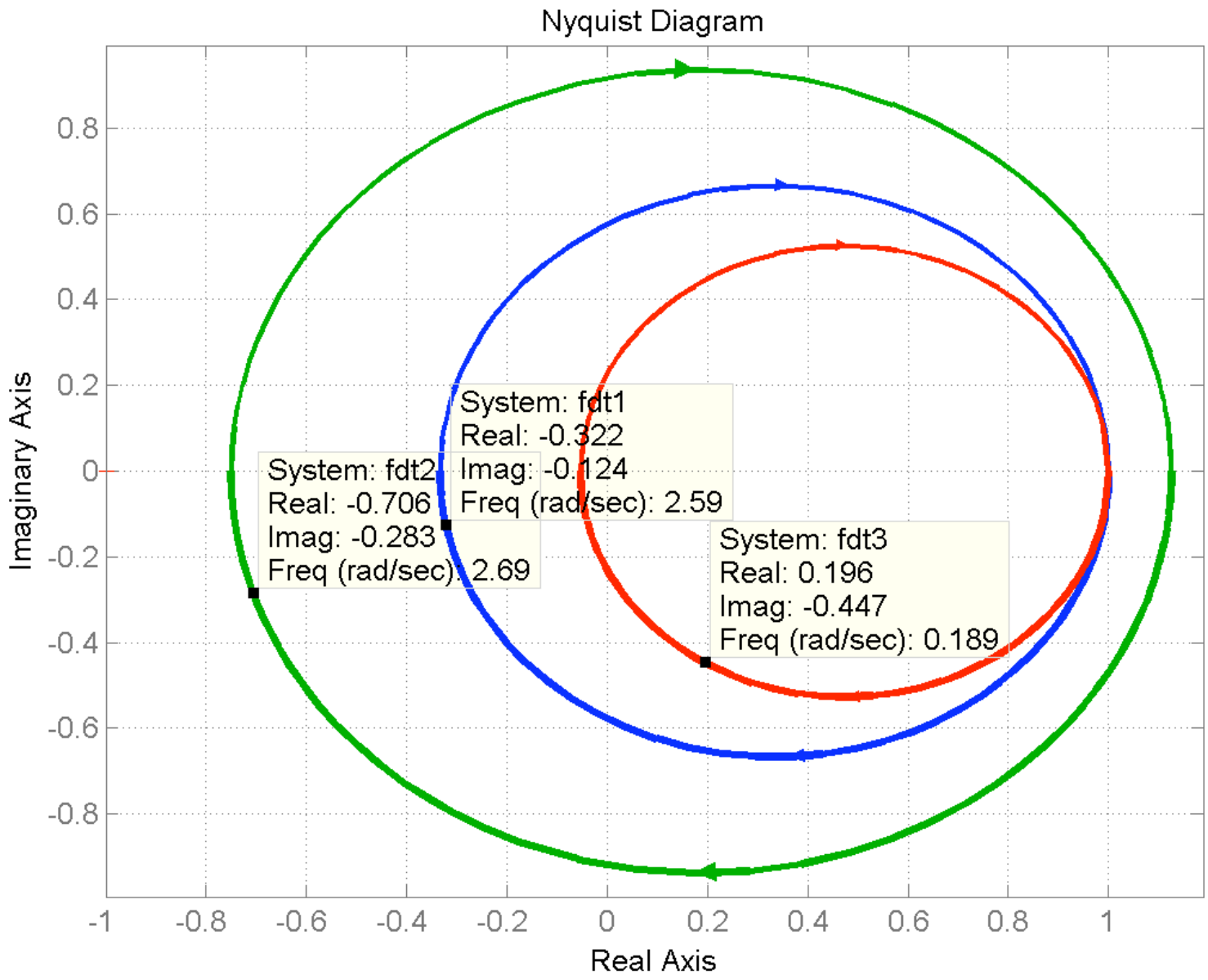
Esempi: diagrammi di Nyquist sistemi a tempo discreto

```
» num = [0.5]
» den=[1 -0.5];
» fdt1=tf(num,den,-1);
» num = [0.9]
» den=[1 -0.2];
» fdt2=tf(num,den,-1);
» num = [0.1]
» den=[1 -0.9];
» fdt3=tf(num,den,-1);
» figure;bode(fdt1,fdt2,fdt3)
» figure;nyquist(fdt1,fdt2,fdt3);
```

$$\mathrm{FdT}_1(z) = \frac{0.5}{z - 0.5}$$

$$\mathrm{FdT}_2(z) = \frac{0.9}{z - 0.2}$$

$$\mathrm{FdT}_3(z) = \frac{0.1}{z - 0.9}$$

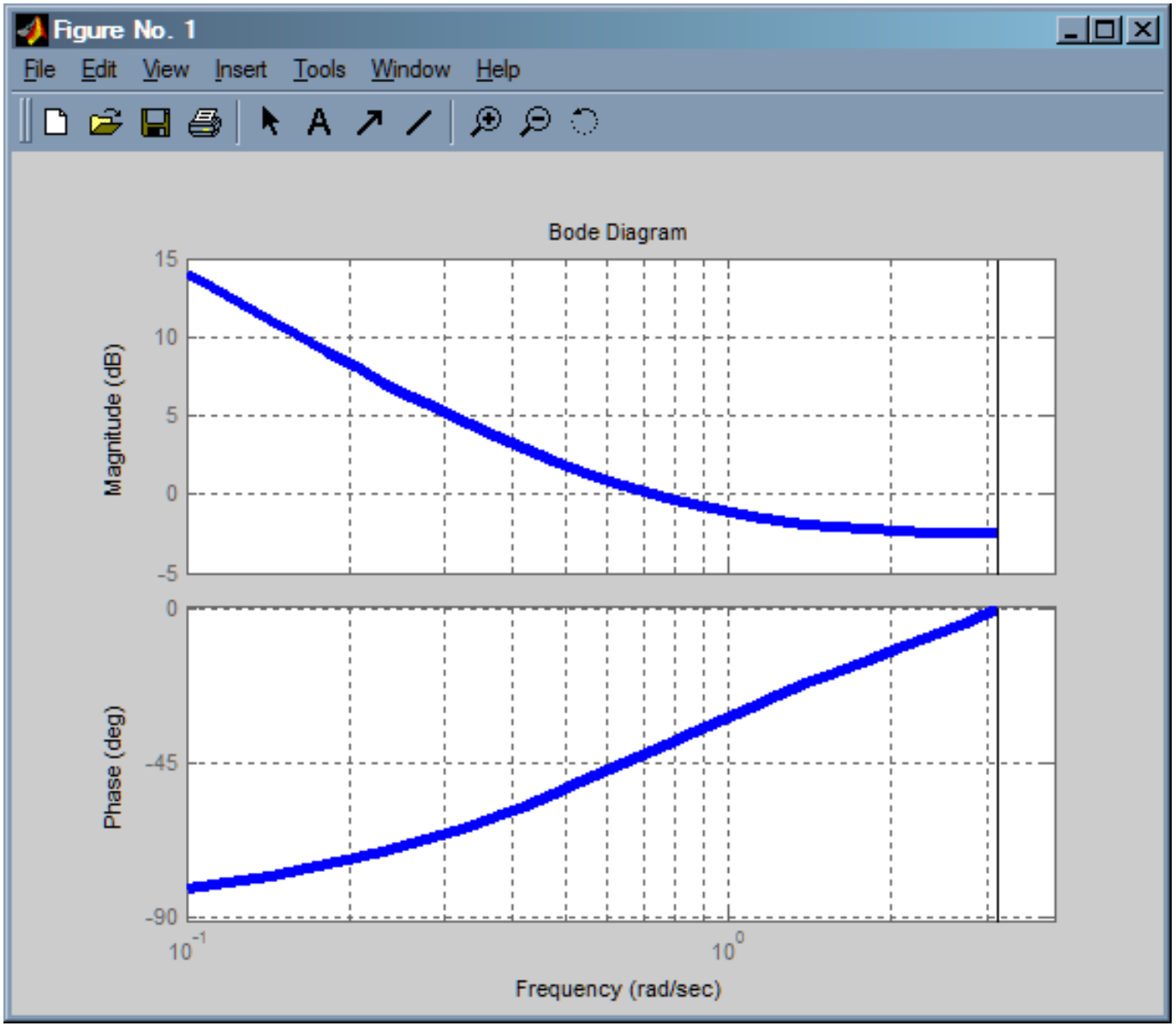

Nyquist Diagram
Imaginary Axis
Real Axis
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
System: fdt1
Real: -0.322
Imag: -0.124
Freq (rad/sec): 2.59
System: fdt2
Real: -0.706
Imag: -0.283
Freq (rad/sec): 2.69
System: fdt3
Real: 0.196
Imag: -0.447
Freq (rad/sec): 0.189

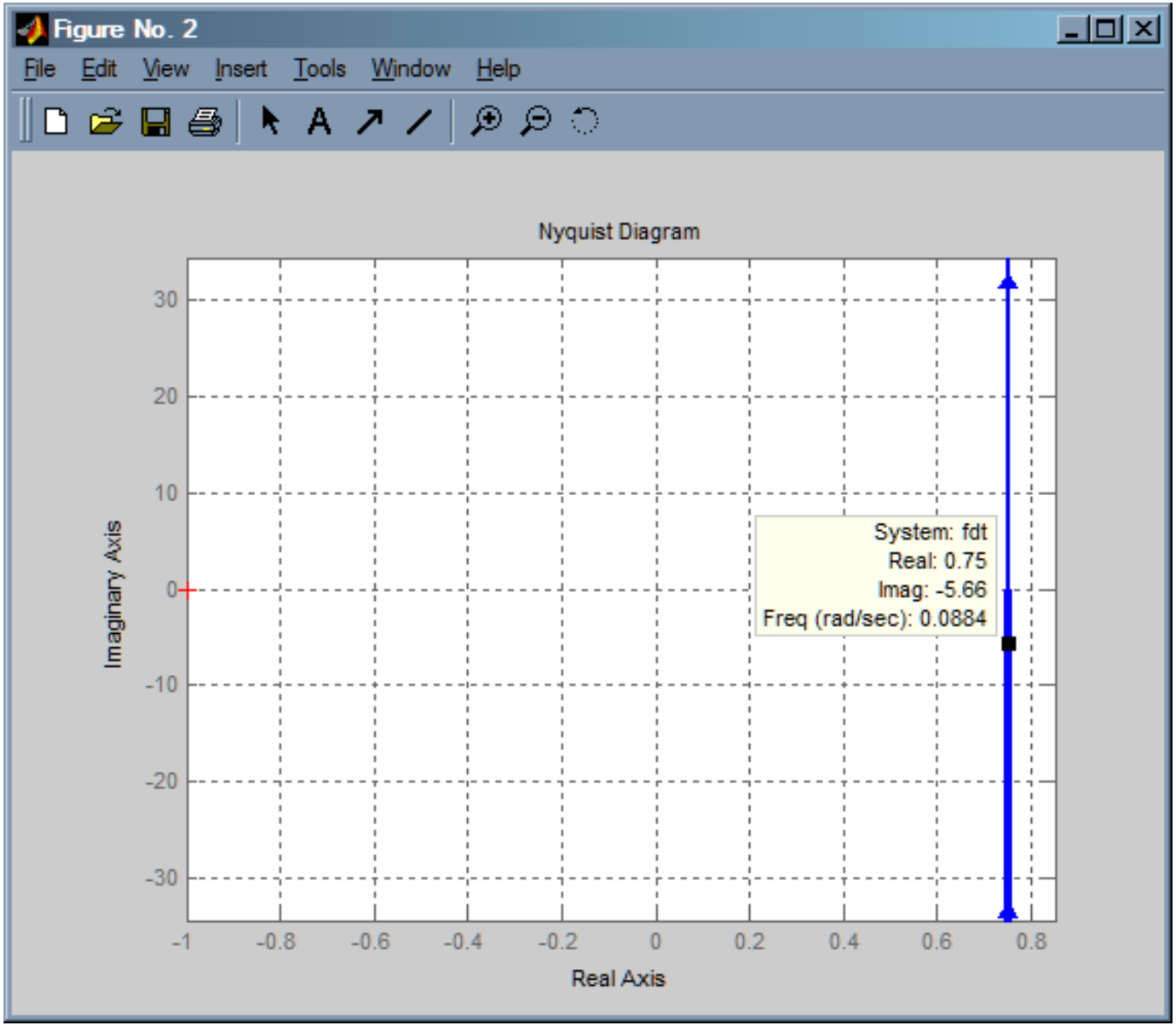
Esempi: diagrammi di Nyquist sistemi a tempo discreto

```
fdt=tf([1 -0.5],[1 -1],-1); % un polo in +1
figure;bode(fdt)
figure;nyquist(fdt)
```

$$\mathrm{FdT}(z) = \frac{z - 0.5}{z - 1}$$

$$\mathrm{FdT}(z) = \frac{z - 0.5}{z - 1}$$





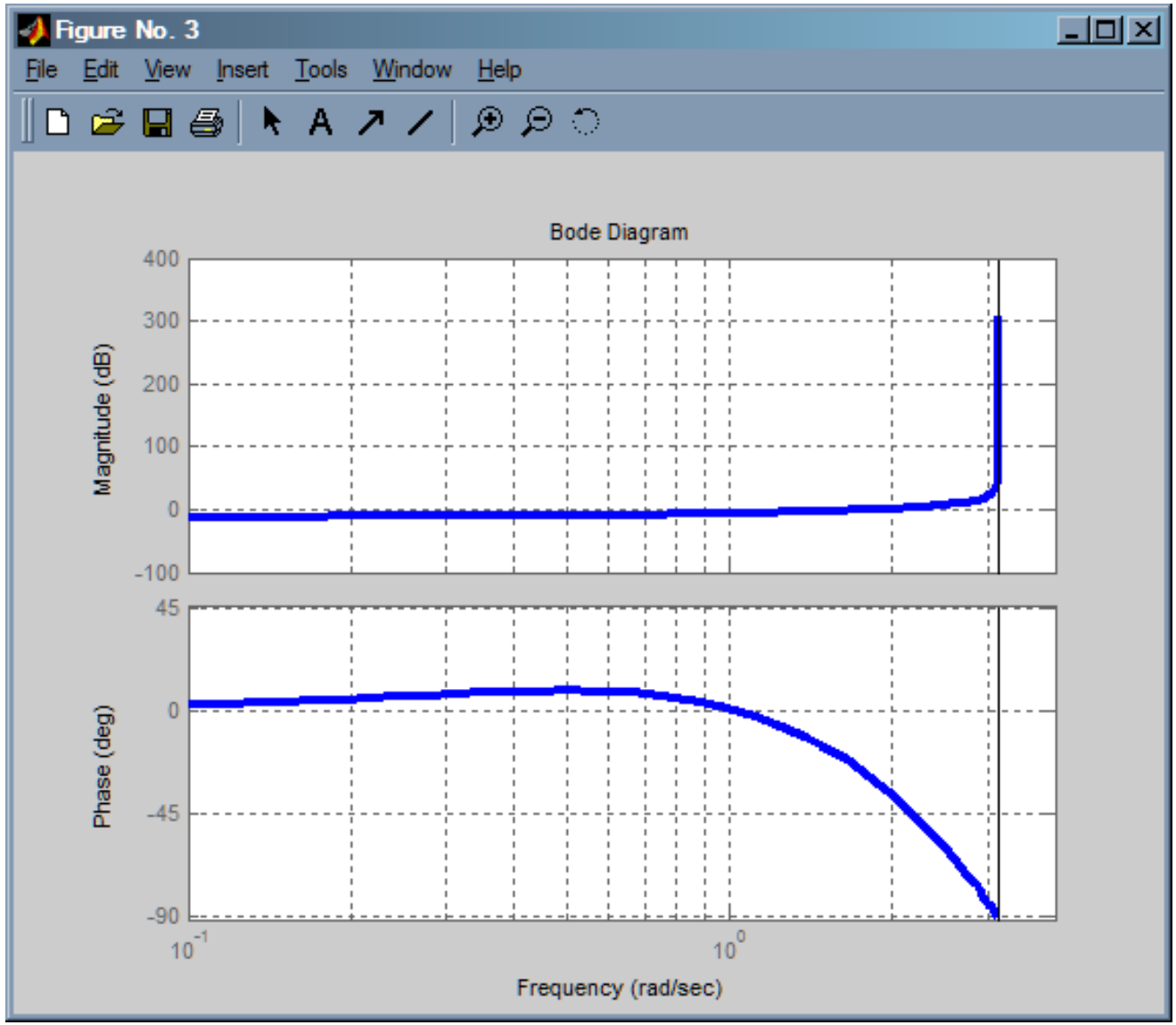
$$\mathrm{FdT}(z) = \frac{z - 0.5}{z - 1}$$

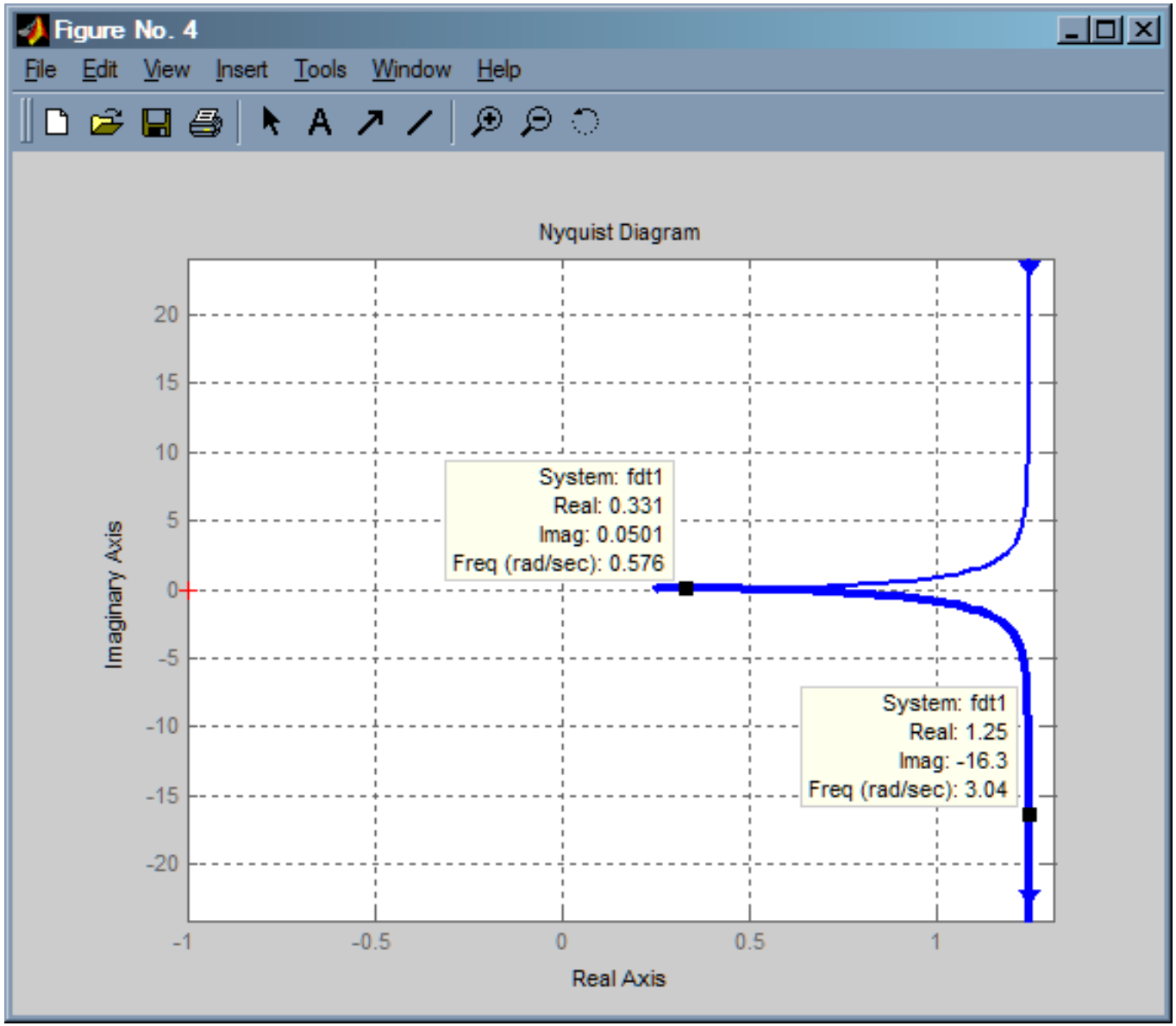
Esempi: diagrammi di Nyquist sistemi a tempo discreto

```
fdt1=tf([1 -0.5],[1 1 0],-1); % un polo in -1 ed uno un 0
figure;bode(fdt1);
figure;nyquist(fdt1)
figure;nyquist(fdt1,{0.01,pi/2}) % metto in evidenza il
% grafico per angoli piccoli
```

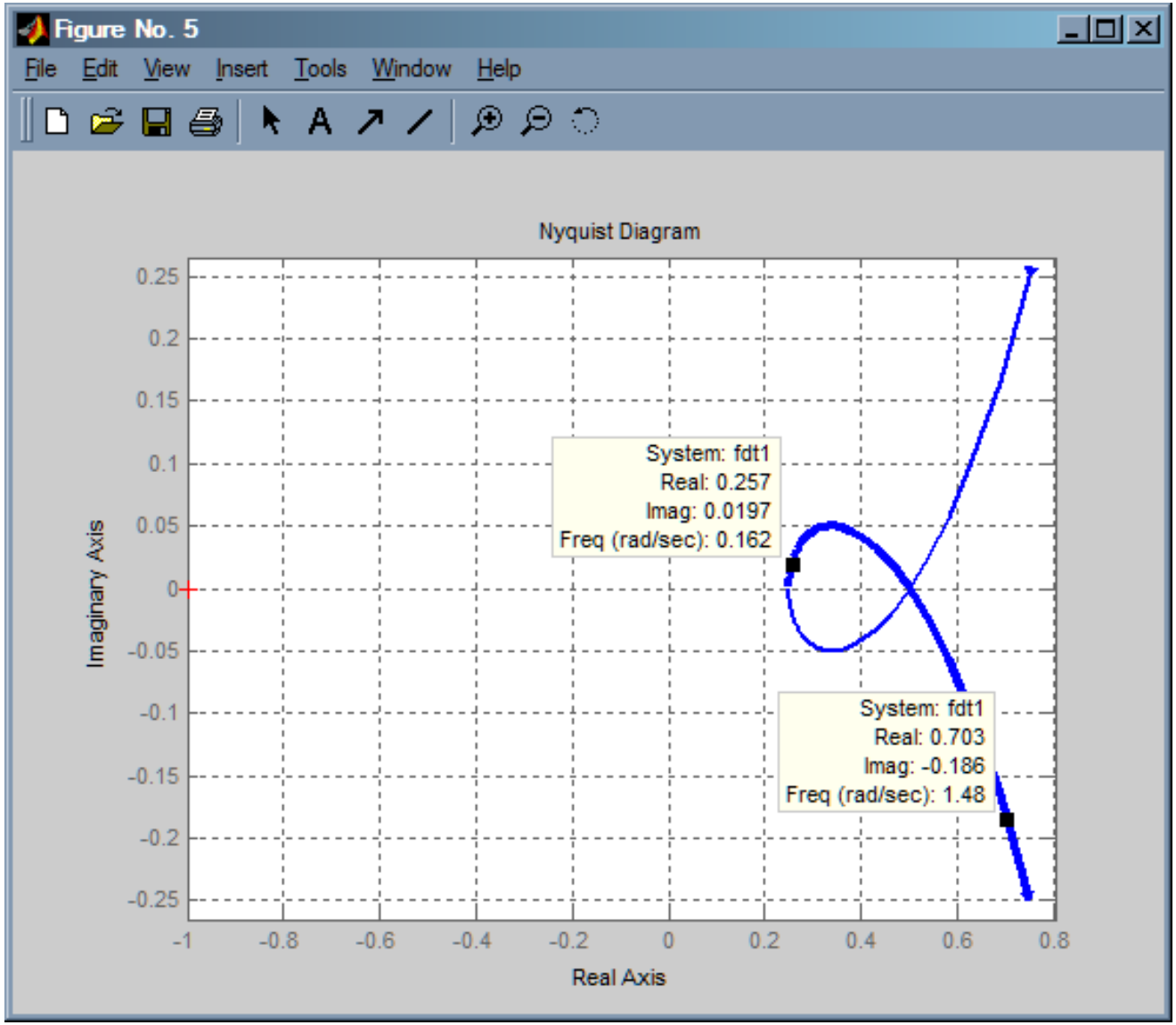
$$\mathrm{FdT}_1(z) = \frac{z - 0.5}{z^2 + z}$$

$$\mathrm{FdT}_1(z) = \frac{z - 0.5}{z^2 + z}$$





$$\mathrm{FdT}_1(z) = \frac{z - 0.5}{z^2 + z}$$



$$\mathrm{FdT}_1(z) = \frac{z - 0.5}{z^2 + z}$$

un particolare del diagramma di Nyquist

Esempi: diagrammi di Nyquist sistemi a tempo discreto

```
fdt2=tf([1 -0.5],conv([1 1 1],[2 1]),-1) % un polo in 1/2 e due compl.con. di modulo 1
figure;bode(fdt2);
figure;nyquist(fdt2);
% in evidenza il comportamento per valori angolari piccoli (da 0 a pi/3)
figure;nyquist(fdt2,{0.001,pi/3});
% in evidenza il comportamento vicino all'asintoto in 2pi/3
angolo_asintoto = 2*pi/3;
figure;nyquist(fdt2,{0.0001, angolo_asintoto-0.001});
figure;nyquist(fdt2,{angolo_asintoto+0.001, angolo_asintoto+0.1});

% estrae i dati. NB sono vettori tridimensionali!!
[ r1 i1] = nyquist(fdt2 , linspace(0 , angolo_asintoto-0.001 , 1000));
[ r2 i2] = nyquist(fdt2 , linspace(angolo_asintoto+0.001 , pi , 1000));
% ridiventano vettori bidimensionali
% (obbligatorio per il comando PLOT)
r1 = reshape(r1 , 1 , 1000); i1 = reshape(i1 , 1 , 1000);
r2 = reshape(r2 , 1 , 1000); i2 = reshape(i2 , 1 , 1000);
% diagramma di Nyquist della sola parte per
% pulsazioni positive
figure; plot(r1 , i1 ,'-b', r2 , i2,'-c');
hold on; plot(-1 , 0 , '+r'); grid on; axis([-5 5 -20 20]);
% con in evidenza il punto (-1+j0)
```

$$\mathrm{FdT}_2 = \frac{z - 0.5}{(2z - 1)\left(z^2 + z + 1\right)}$$

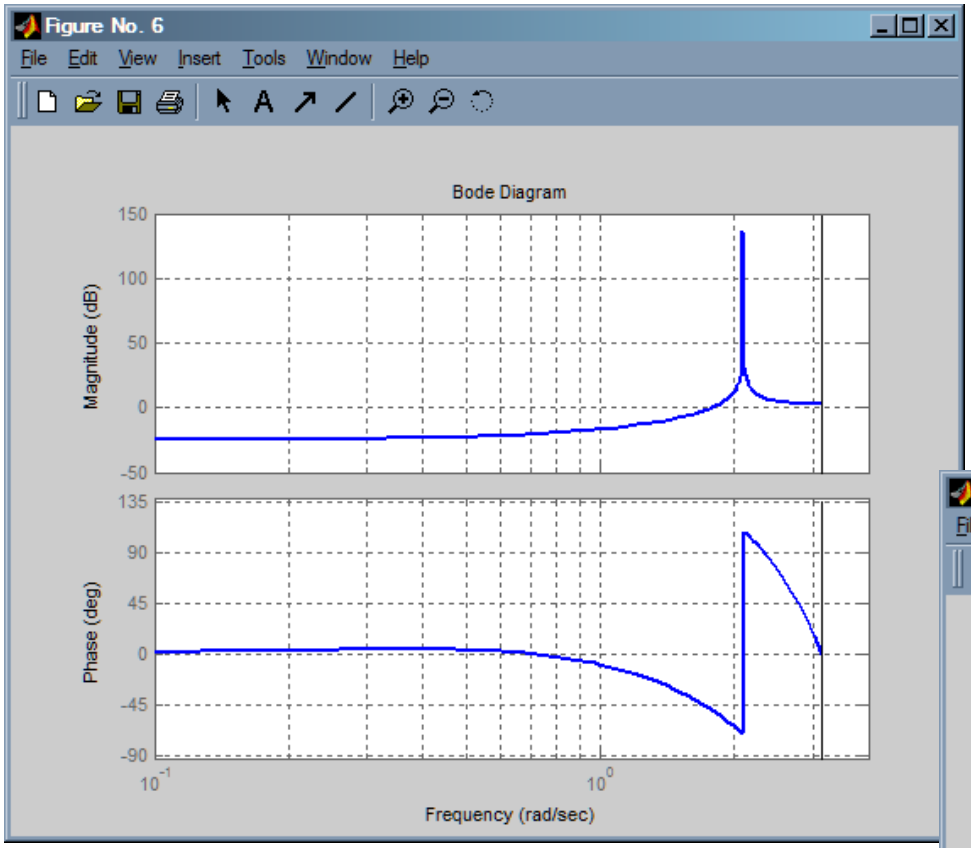


Figure No. 6
File Edit View Insert Tools Window Help
Bode Diagram
Magnitude (dB)
150
100
50
0
-50
Phase (deg)
135
90
45
0
-45
-90
10^{-1}
10^{0}
Frequency (rad/sec)

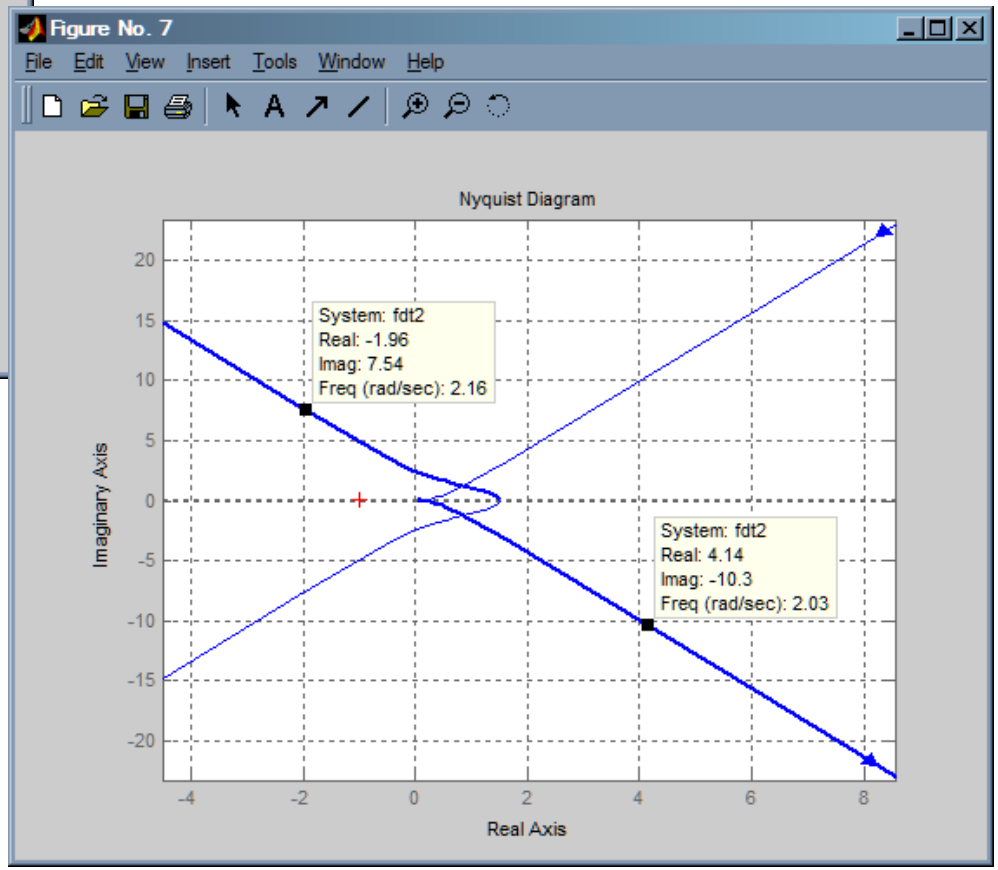
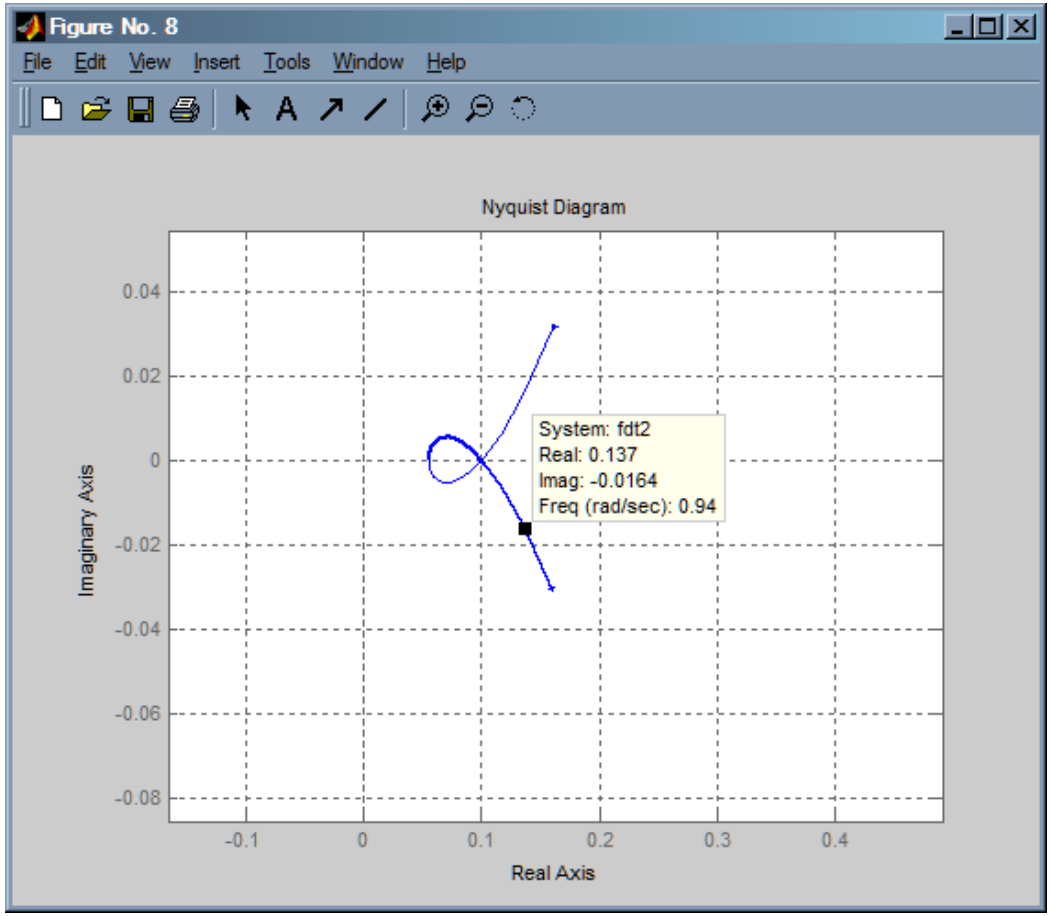


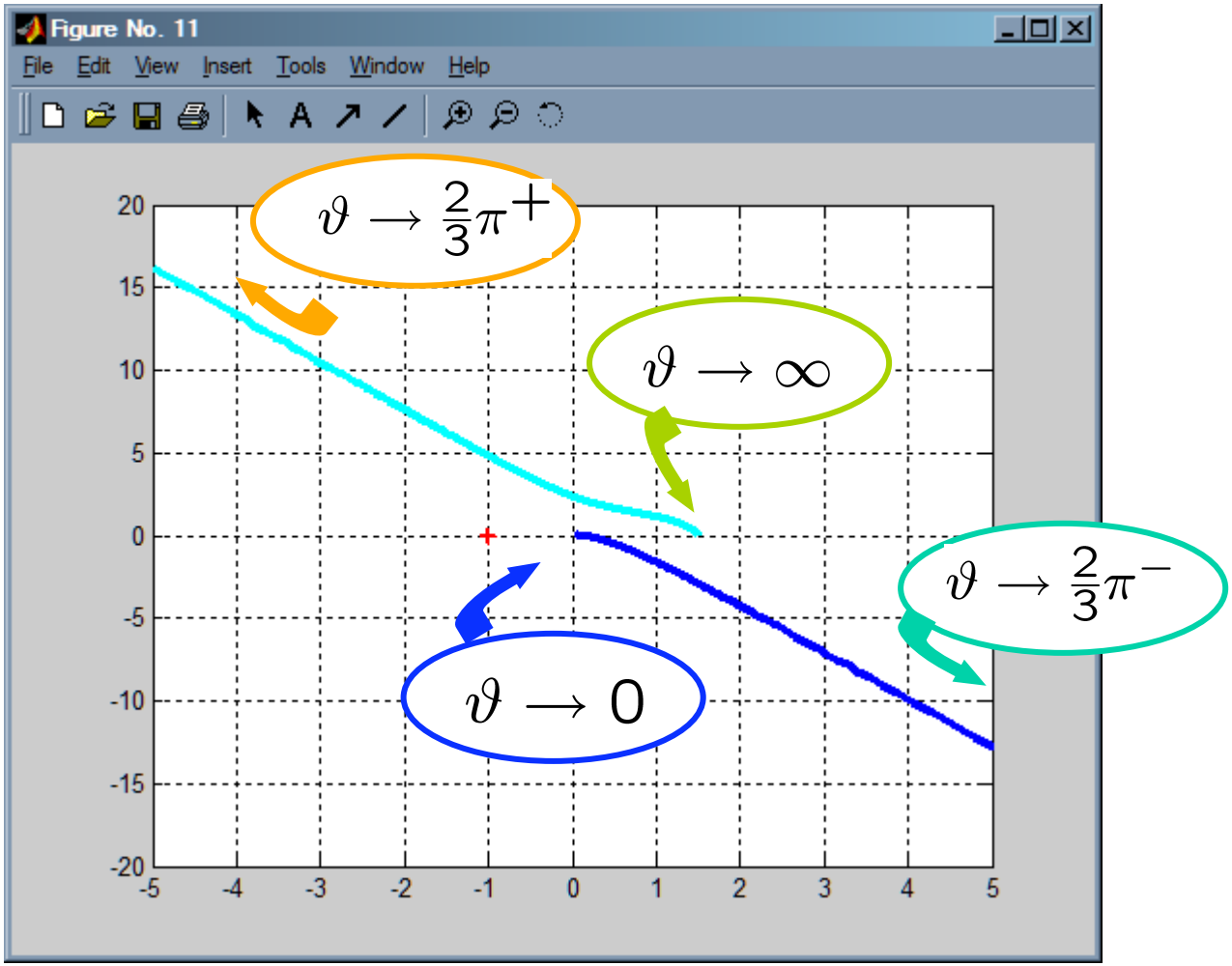
Figure No. 7
File Edit View Insert Tools Window Help
Nyquist Diagram
System: fdt2
Real: -1.96
Imag: 7.54
Freq (rad/sec): 2.16
System: fdt2
Real: 4.14
Imag: -10.3
Freq (rad/sec): 2.03
Imaginary Axis
20
15
10
5
0
-5
-10
-15
-20
-4
-2
0
2
4
6
8
Real Axis

un particolare del diagramma di Nyquist




```
% in evidenza il comportamento per
% valori angolari piccoli (da 0 a pi/3)
figure;nyquist(fdt2,{0.001,pi/3});
```

un particolare del diagramma di Nyquist

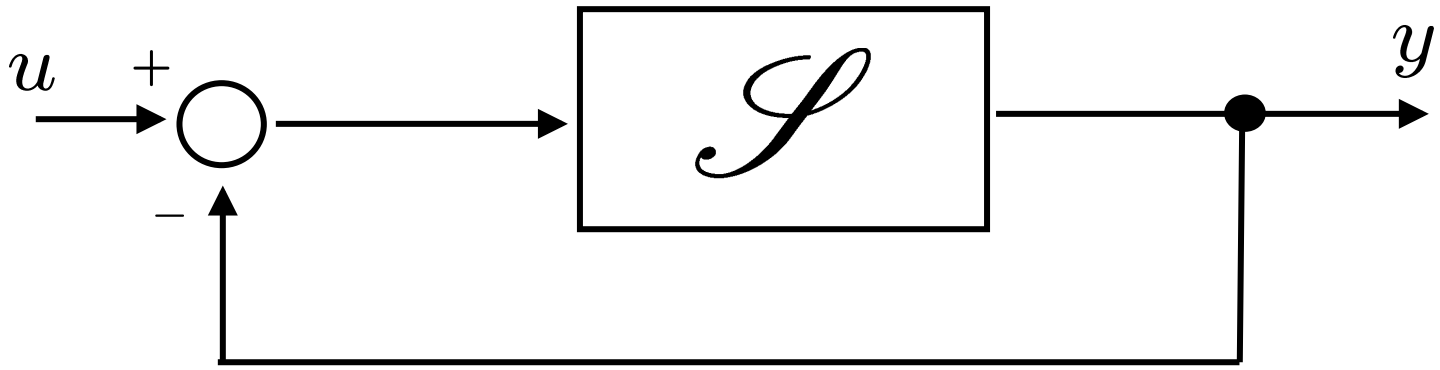


Il codice Matlab corrispondente

```
% estrae i dati. NB sono vettori tridimensionali!!
[ r1 i1] = nyquist(fdt2 , linspace(0 , angolo_asintoto-0.001 , 1000));
[ r2 i2] = nyquist(fdt2 , linspace(angolo_asintoto+0.001 , pi , 1000));
% ridiventano vettori bidimensionali
% (obbligatorio per il comando PLOT)
r1 = reshape(r1 , 1 , 1000); i1 = reshape(i1 , 1 , 1000);
r2 = reshape(r2 , 1 , 1000); i2 = reshape(i2 , 1 , 1000);
% diagramma di Nyquist della sola parte per
% pulsazioni positive
figure; plot(r1 , i1 ,'-b', r2 , i2,'-c');
hold on; plot(-1 , 0 , '+r'); grid on; axis([-5 5 -20 20]);
% con in evidenza il punto (-1+j0)
```

Margini di guadagno e di fase

- Esiste un comando del Control Toolbox che permette di analizzare la stabilità a ciclo chiuso di un sistema LTI di tipo SISO determinando (anche visualizzando) i margini di guadagno e di fase del sistema di ciclo aperto.



Margini di guadagno e di fase

- Sintassi del comando ***margin*** :
- ***[Gm,Pm,Wcg,Wcp] = MARGIN(SYS)*** determina i margini di guadagno ***Gm***, di fase ***Pm***, le pulsazioni corrispondenti ***Wcg***, ***Wcp*** per il sistema a ciclo aperto SYS.
- ***SYS*** è un sistema LTI a tempo continuo oppure a tempo discreto.
- Il comando ***MARGIN(SYS)*** (usato senza richiedere variabili in uscita) visualizza il diagramma di Bode della risposta in frequenza del sistema a ciclo aperto SYS ed in esso visualizza i margini di stabilità e le corrispondenti pulsazioni.

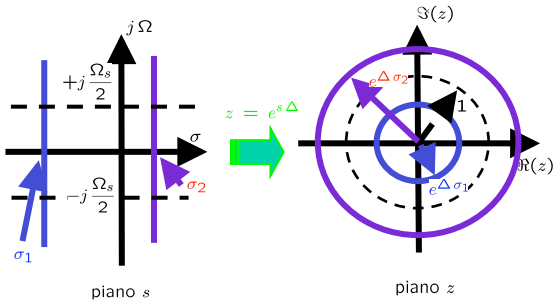
Luoghi caratteristici in s e corrispondenti in z

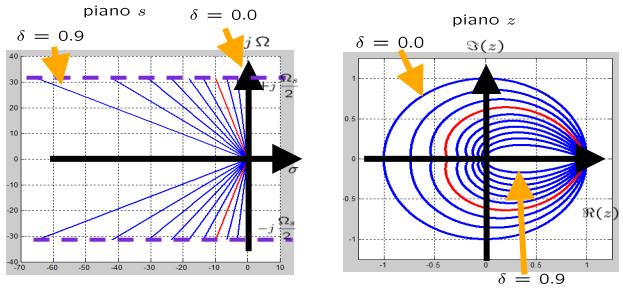
Luoghi a modulo costante, a smorzamento costante, a pulsazione naturale costante ...

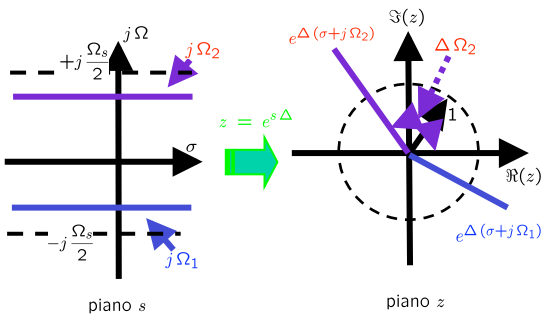
Luoghi in s e corrispondenti in z: richiami

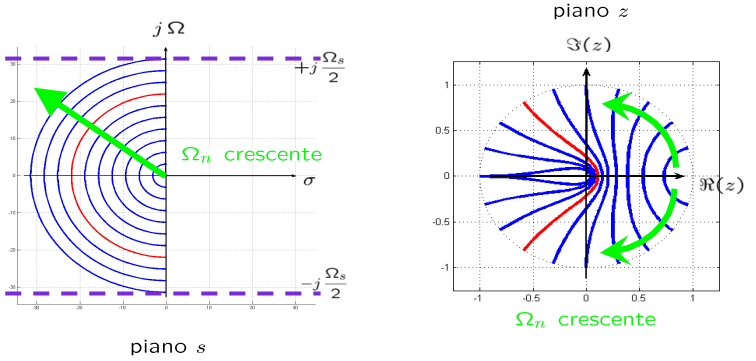
- Data la relazione del campionamento $z = e^{s\,\Delta}$

  come si modificano i luoghi a smorzamento costante, quelli a pulsazione costante, quelli a pulsazione naturale costante oppure quelli a decadimento esponenziale costante?









Ed in Matlab?

- Tramite le istruzioni **sgrid** e **zgrid** è possibile disegnare, rispettivamente nel piano *s* di una FdT *F(s)* oppure nel piano *z* di una FdT *F(z)*, un insieme di curve dei luoghi a smorzamento costante/a pulsazione naturale costante.

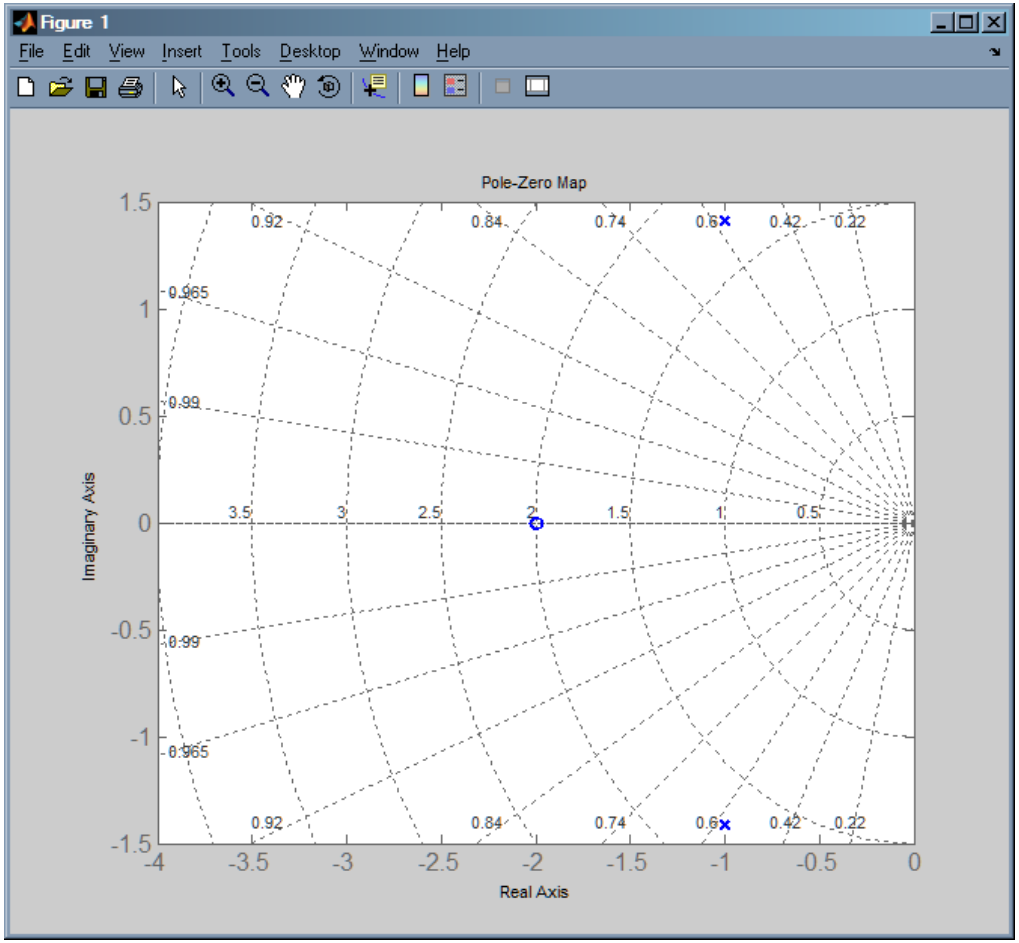
- In particolare **sgrid(Z, Wn)** / **zgrid(Z, Wn)** disegna
  - l'insieme dei **luoghi a smorzamento costante** individuati dai valori di smorzamento assegnati nel **vettore Z**,
  - L'insieme dei **luoghi a pulsazione naturale costante** individuati dai valori di pulsazione contenuti nel **vettore Wn**.
  - Se non presenti i vettori Z, Wn, vengono disegnati i luoghi per valori predefiniti di smorzamento e pulsazione naturale.

Esempi

- Un primo esempio per un sistema a tempo continuo definito da una FdT:

  » ***fdt = tf([1 2],[1 2 3]);***

  » ***figure;pzmap(fdt);sgrid;***

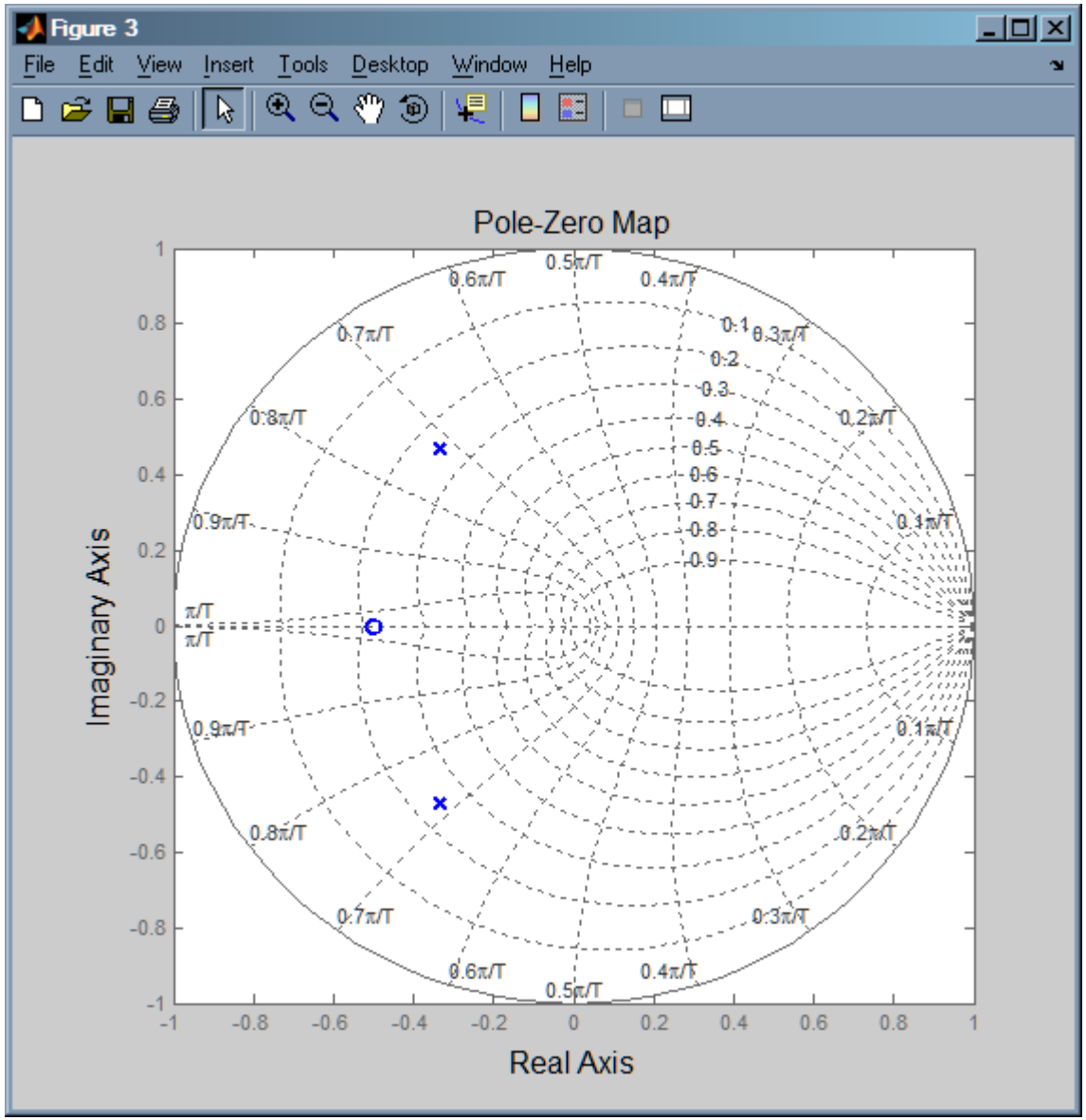


Esempi

- Un secondo esempio per un sistema a tempo discreto descritto da una FdT:

» ***fdt = tf([2 1],[3 2 1],0.5);***

» ***figure;pzmap(fdt);zgrid;***



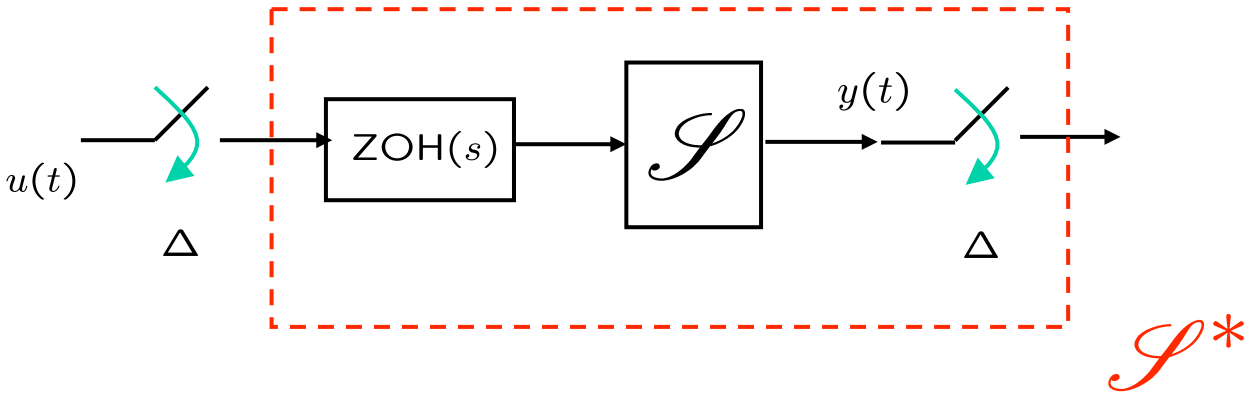
Conversione da tempo—continuo a tempo —discreto in MATLAB

Come ottenere una rappresentazione a segnali campionati di un sistema dinamico a tempo continuo e viceversa.

Conversione da tempo continuo a tempo discreto: richiami

- **Su base stato**: il sistema a tempo continuo è descritto da

$$\mathscr{S} \quad \Longleftrightarrow \quad \begin{cases} \dot{x}(t) &= A_c\, x(t) + B_c\, u(t) \\ y(t) &= C_c\, x(t) + D_c\, u(t) \end{cases}$$

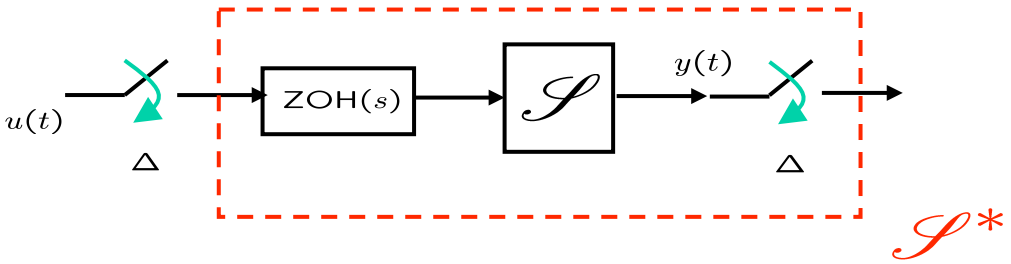


Conversione da tempo continuo a tempo discreto: richiami

- Il sistema a **segnali campionati** (**a tempo discreto**) ottenuto per campionamento è

$$\begin{cases} x_{i+1} = A\,x_i + B\,u_i \\ y_i = C\,x_i + D\,u_i \end{cases}$$

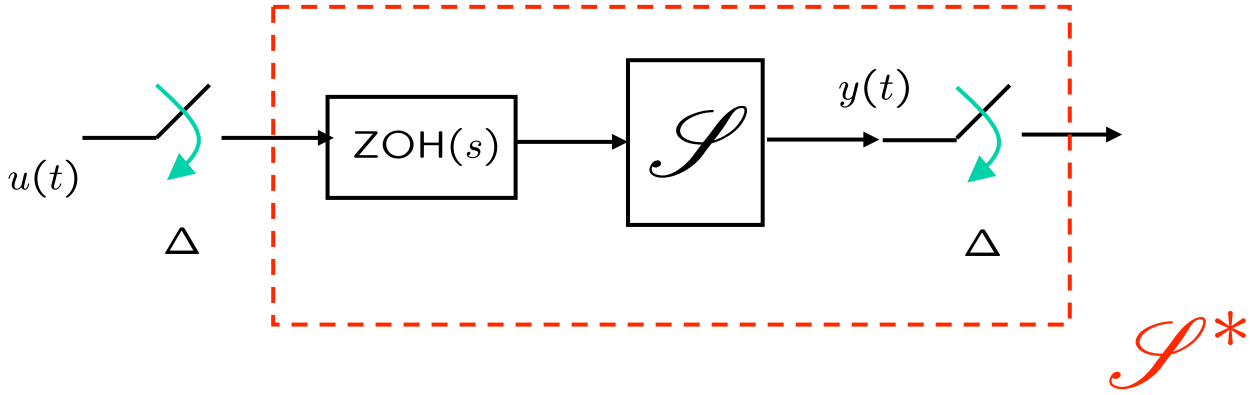
$$\begin{cases} A = e^{A_c \Delta} \;,\; B = \displaystyle\int_0^{\Delta} e^{A_c r}\, B_c\, dr \\ C = C_c \;,\; D = D_c \end{cases}$$



Conversione da tempo continuo a tempo discreto: richiami

- Su **base Funzione di Trasferimento**: il sistema a tempo continuo è descritto da

$$\mathscr{S} \quad \Longleftrightarrow \quad G(s) = \frac{\beta_0 s^m + \ldots + \beta_m}{s^n + \alpha_1 s^{n-1} + \ldots + \alpha_0}$$



Conversione da tempo continuo a tempo discreto: richiami

- Il sistema a **segnali campionati** (**a tempo discreto**) ottenuto per campionamento è

$$G(z) = \left(1 - z^{-1}\right) \mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{G(s)}{s}\right\}\right\}$$

$$G(z) = \left(1 - z^{-1}\right) \sum_{\text{poli di } \left[\frac{G(\lambda)}{\lambda}\right]} \text{Res} \left[\frac{\frac{G(\lambda)}{\lambda}}{1 - e^{\lambda \Delta} z^{-1}}\right]$$

Conversione da tempo continuo a tempo discreto in MATLAB

- Deve essere assegnato un **sistema dinamico a tempo continuo** come oggetto ***LTI system***
- Il sistema può essere descritto in uno qualsiasi dei modi a disposizione per gli oggetti LTI system: su base stato, su base FdT ecc.

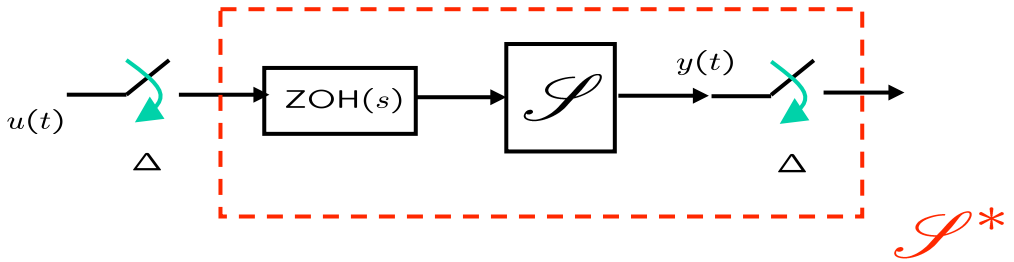
- Il comando che esegue la conversione continuo → discreto è il comando ***c2d***

- ***c2d*** è una funzione che fa parte del pacchetto "***Control Toolbox***".

Sintatti del comando *c2d*

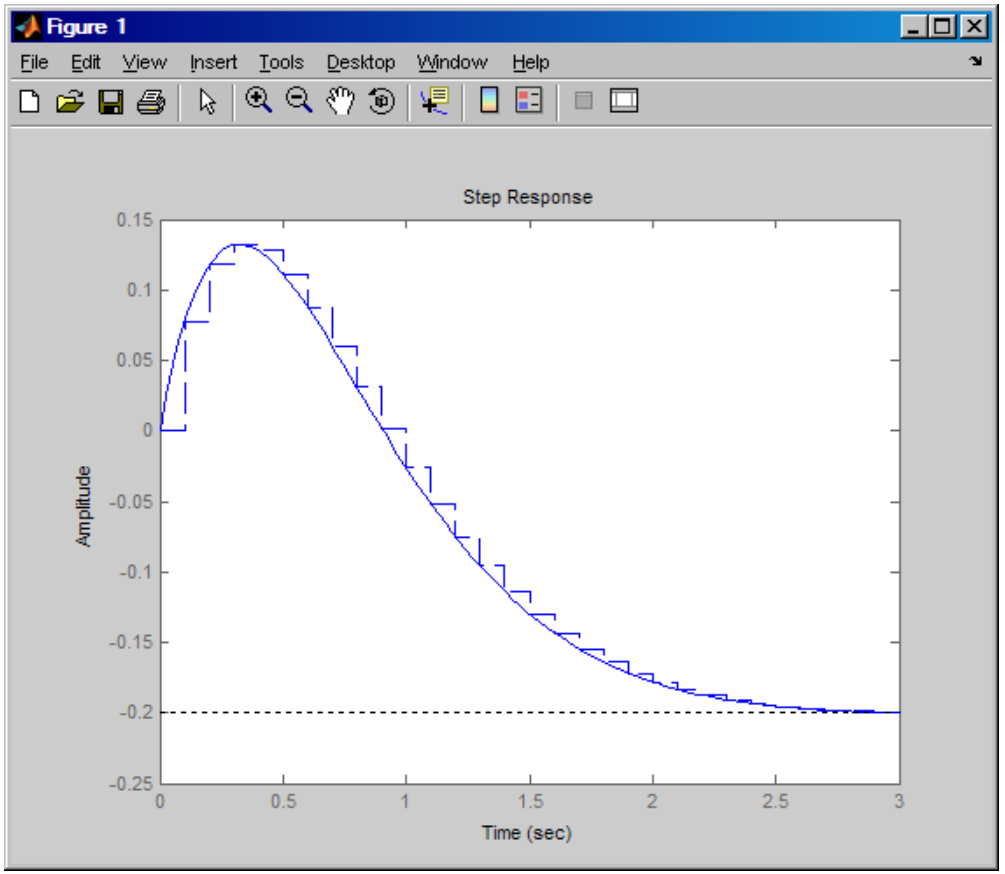
- ***SYSD = c2d(SYSC, Ts)***: esegue la conversione del sistema a tempo continuo descritto dall'oggetto LTI ***SYSC*** nel sistema a segnali campionati ***SYSD***, con periodo di campionamento ***Ts***.

- Il metodo di conversione utilizzato è quello "**ad invarianza della risposta al gradino**" o "**con mantenitore di ordine zero (ZOH) in ingresso**" cioè quello dello schema



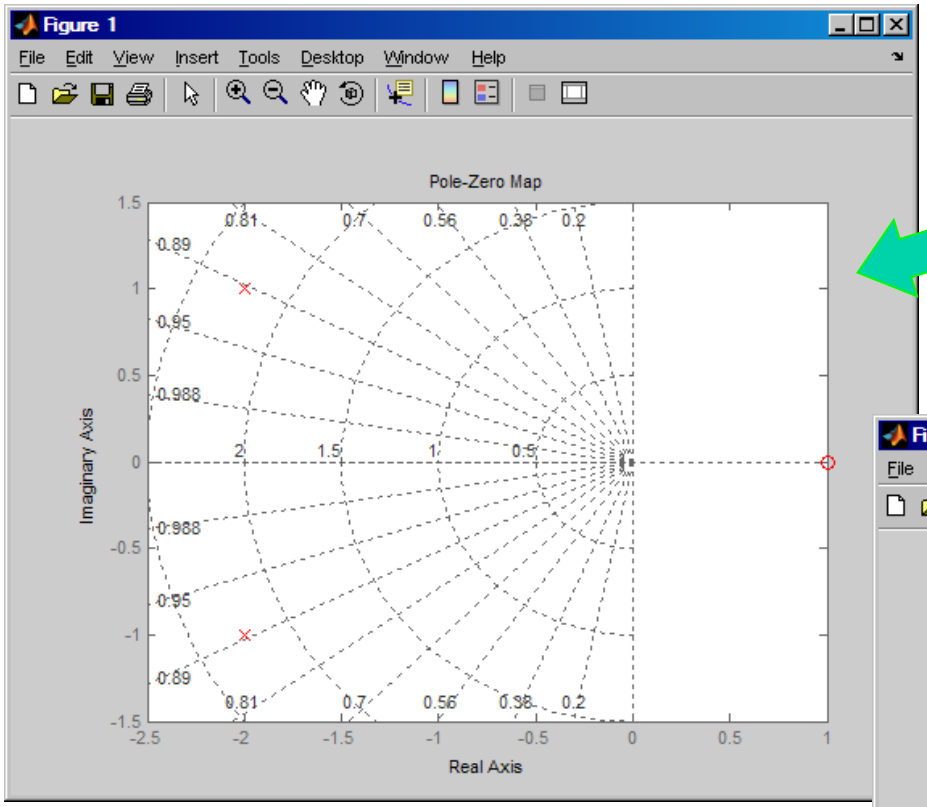
Esempi

```
» H = tf( [1 -1], [1 4 5] );
» Hd = c2d( H, 0.1 );
» step( H, '-', Hd, '--' )
```

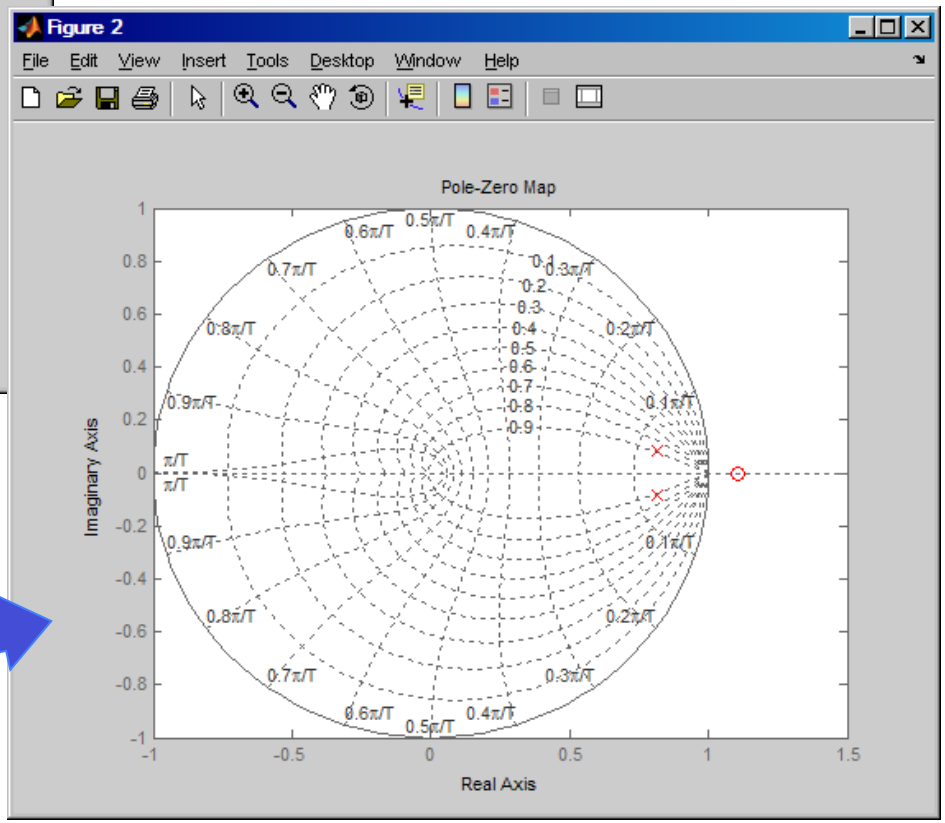


Ancora esempi

```
» a=[-4 -2.5;2 0];b=[1;0];c=[1 -0.5];d=0;
» sABCD = ss(a,b,c,d)
» figure;pzmap(sABCD);
»
» sD = c2d(sABCD, 0.1)
» figure;pzmap(sD);

» figure; impulse (sABCD,'-', sD, '--');
» figure; step (sABCD,'-', sD,'--');
```

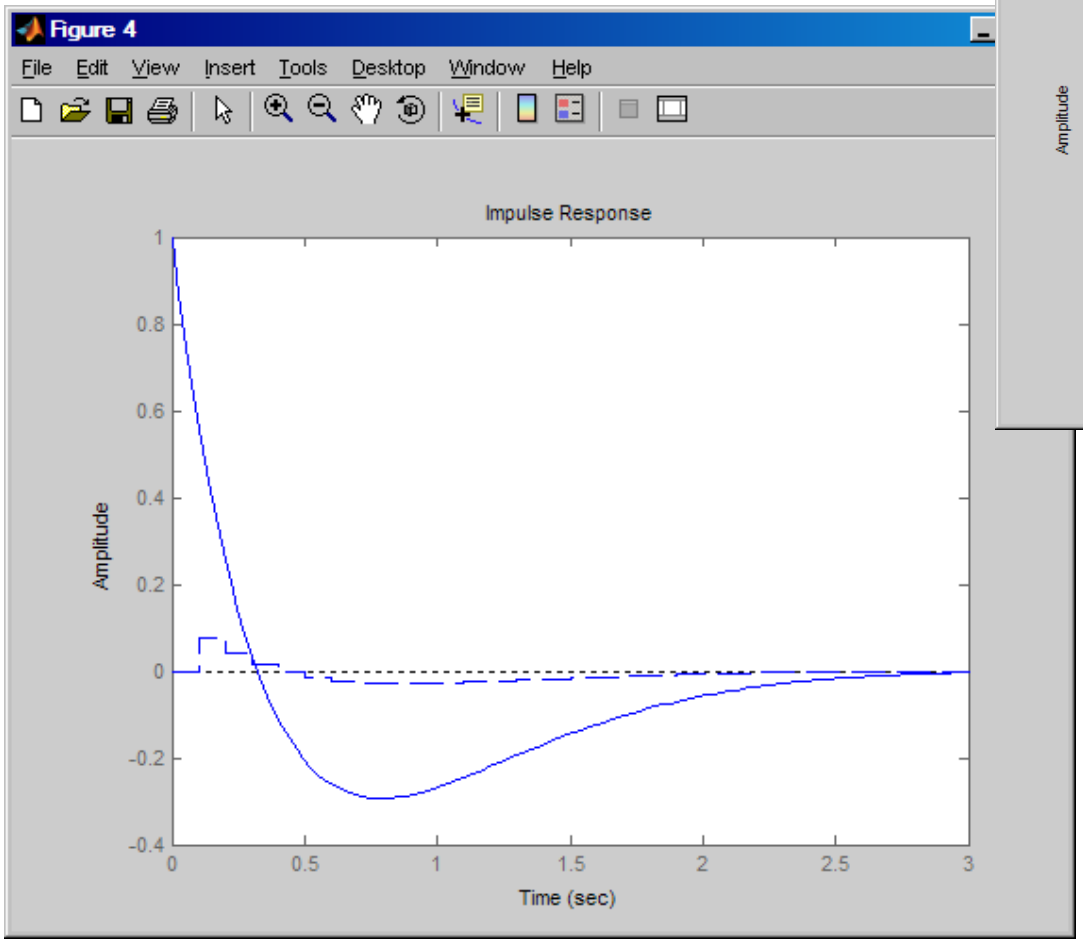


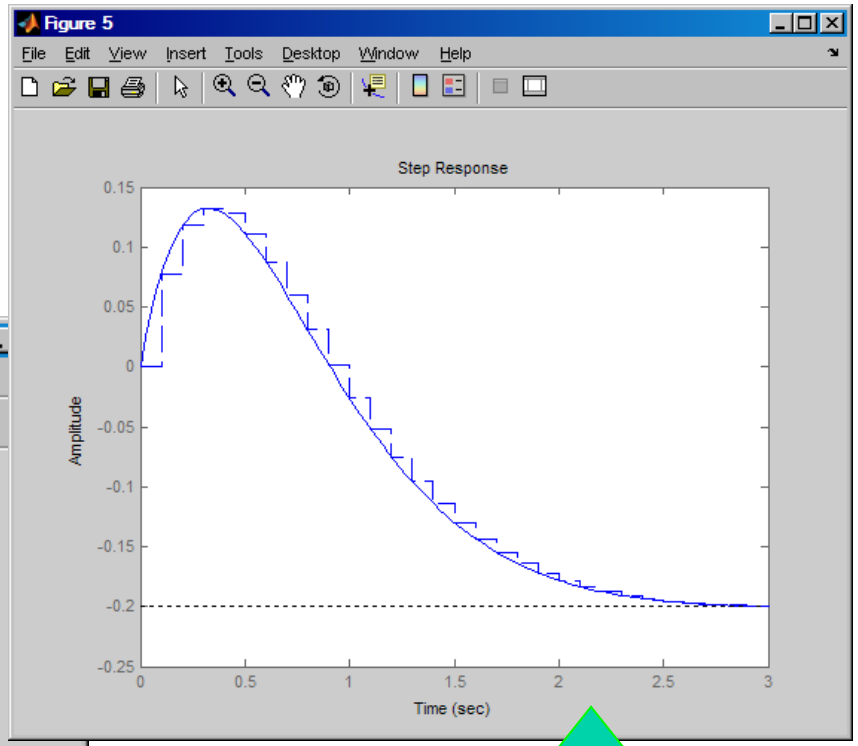
mappa zeri/poli del sistema a tempo continuo



mappa zeri/poli del sistema a tempo discreto

Confronto tra le risposte all'impulso





Confronto tra le risposte allo scalino

Sintatti completa del comando *c2d*

- ***SYSD = c2d(SYSC, Ts, METHOD)***: il sistema a tempo continuo ***SYSC*** viene convertito nel sistema a segnali campionati ***SYSD***, con periodo di campionamento ***Ts***.

- ***METHOD*** identifica il metodo di conversione da utilizzare
  - ***'zoh'*** : conversione con ZOH in ingresso o "invarianza della risposta allo scalino". È il metodo predefinito
  - '**imp**' : conversione per "invarianza della risposta all'impulso"
  - ***'tustin'*** : metodo che fa uso della trasformazione di Tustin
  - ***'prewarp'*** : trasformazione di Tustin con predistorsione in frequenza. Come quarto parametro viene specificata proprio la pulsazione critica Ω_c (in rad/s)

    SYSD = c2d(SYSC ,Ts ,'prewarp' , Wc)

Sintassi completa del comando *c2d*

- Esistono ancora altri possibili metodi di discretizzazione, che non vengono però trattati nel corso di "Controllo Digitale".

- Si rimanda quindi alla documentazione del comando ***c2d*** ed alla bibliografia del corso per approfondimenti su tali metodi.

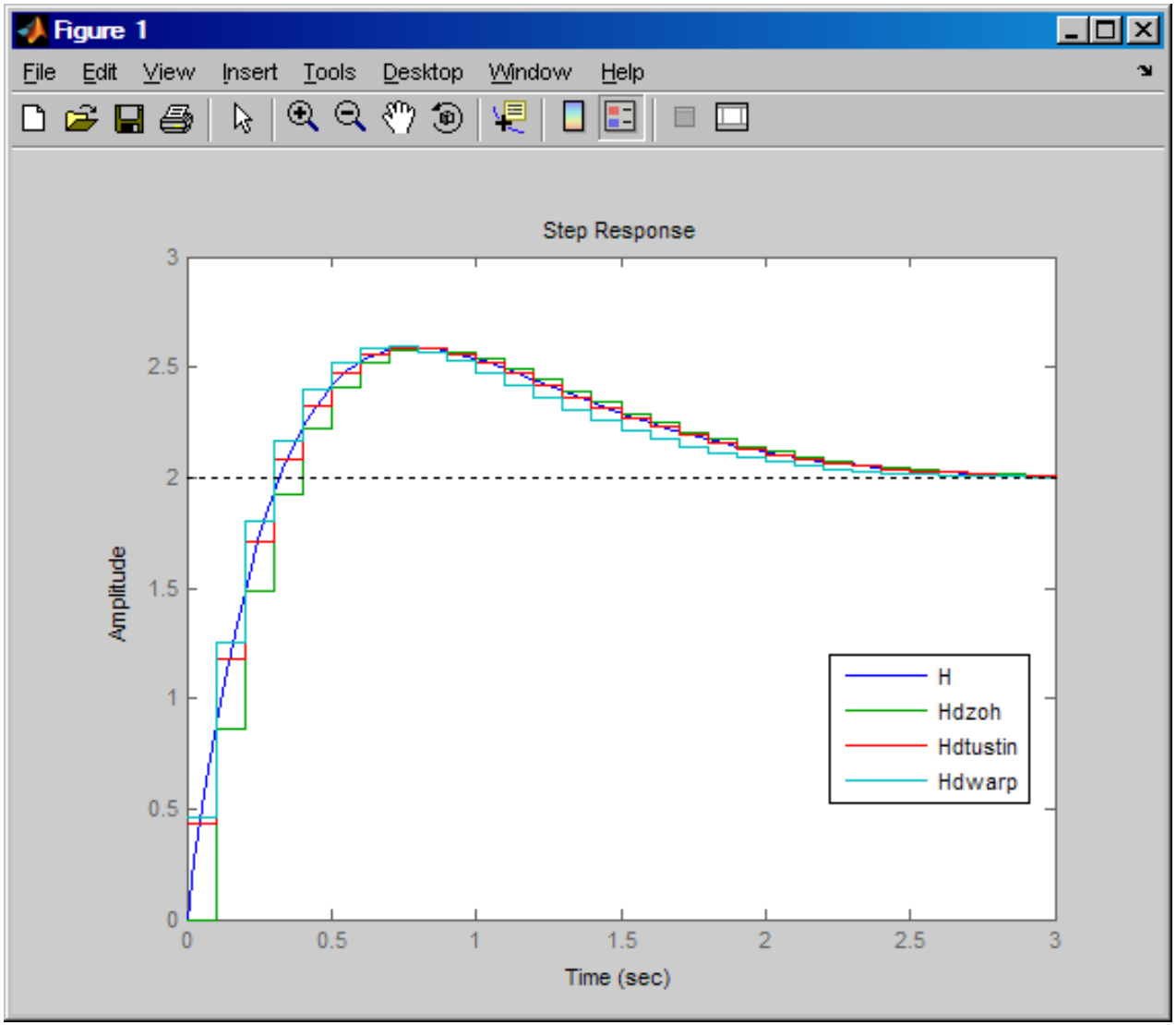
Esempio

```
»  %  sistema LTI a tempo continuo
»  H = 10*tf( [1  1], [1 4 5] );
»
»  % periodo di campionamento: 0.1
»  % metodo di conversione: standard, con ZOH
»  Ts = 0.1; % intervallo di campionamento
»  Hdzoh = c2d( H, Ts,'zoh');
»
»  Hdtustin = c2d( H, Ts,'tustin');
»  % discretizzato con la trasformazione bilineare di Tustin
»
»  Wc = 9.752; % pulsazione critica
»  % e' la pulsazione in cui il sistema continuo ha margine di fase 108 deg
»  % (si provi l'istruzione margin )
»  Hdwarp = c2d(H, Ts, 'prewarp',Wc);
»  % discretizzazione con la trasformazione bilineare di Tustin predistorta
```

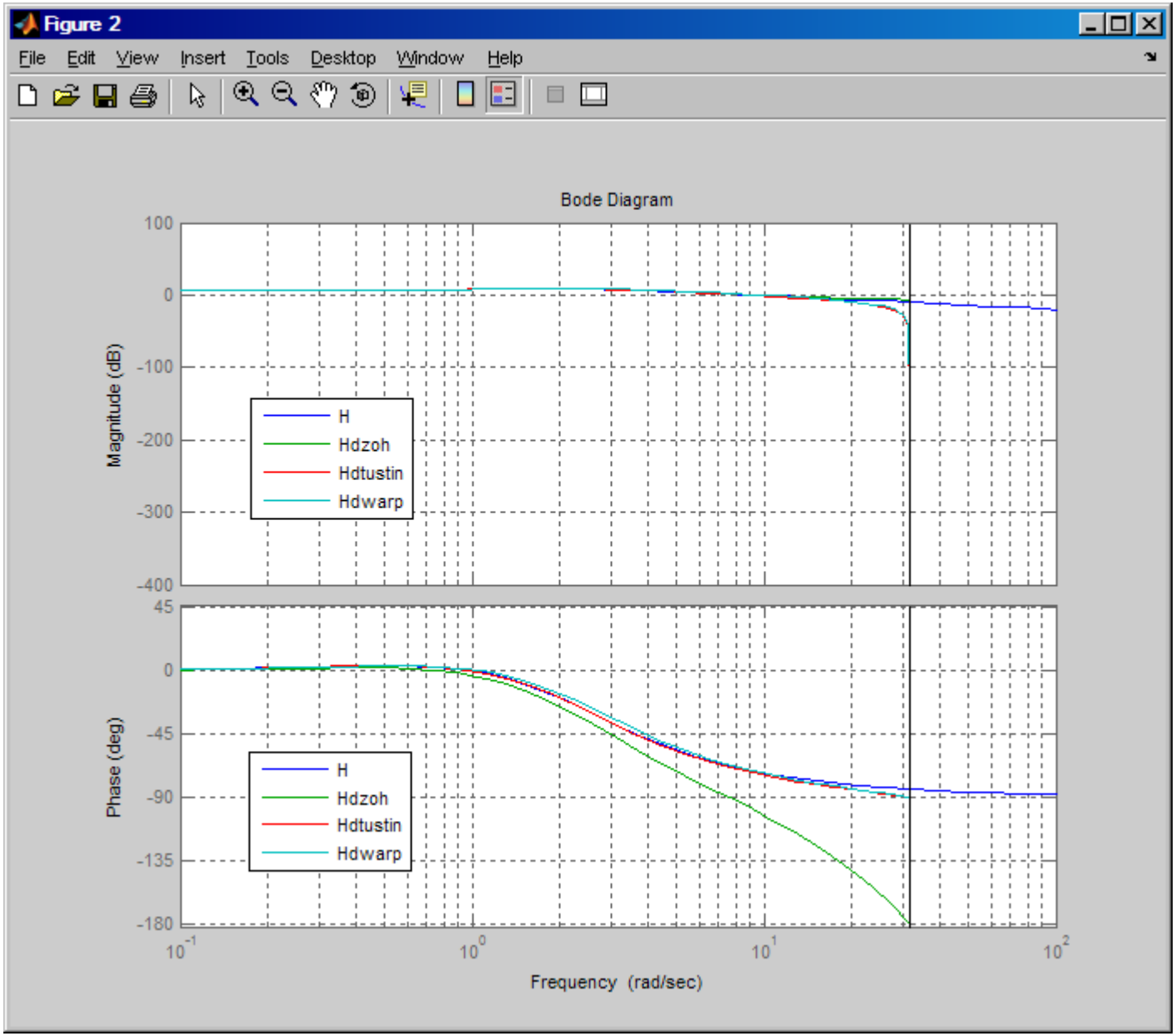
Esempio (continua …)

```
» % ora visualizzo la risposta allo scalino unitario del sistema originario
» % e del sistema a segnali campionati
» step( H, Hdzoh, Hdtustin, Hdwarp);
»
» % ora invece confrontiamo la risposta in frequenza
» figure;bode(H ,Hdzoh, Hdtustin, Hdwarp );
```

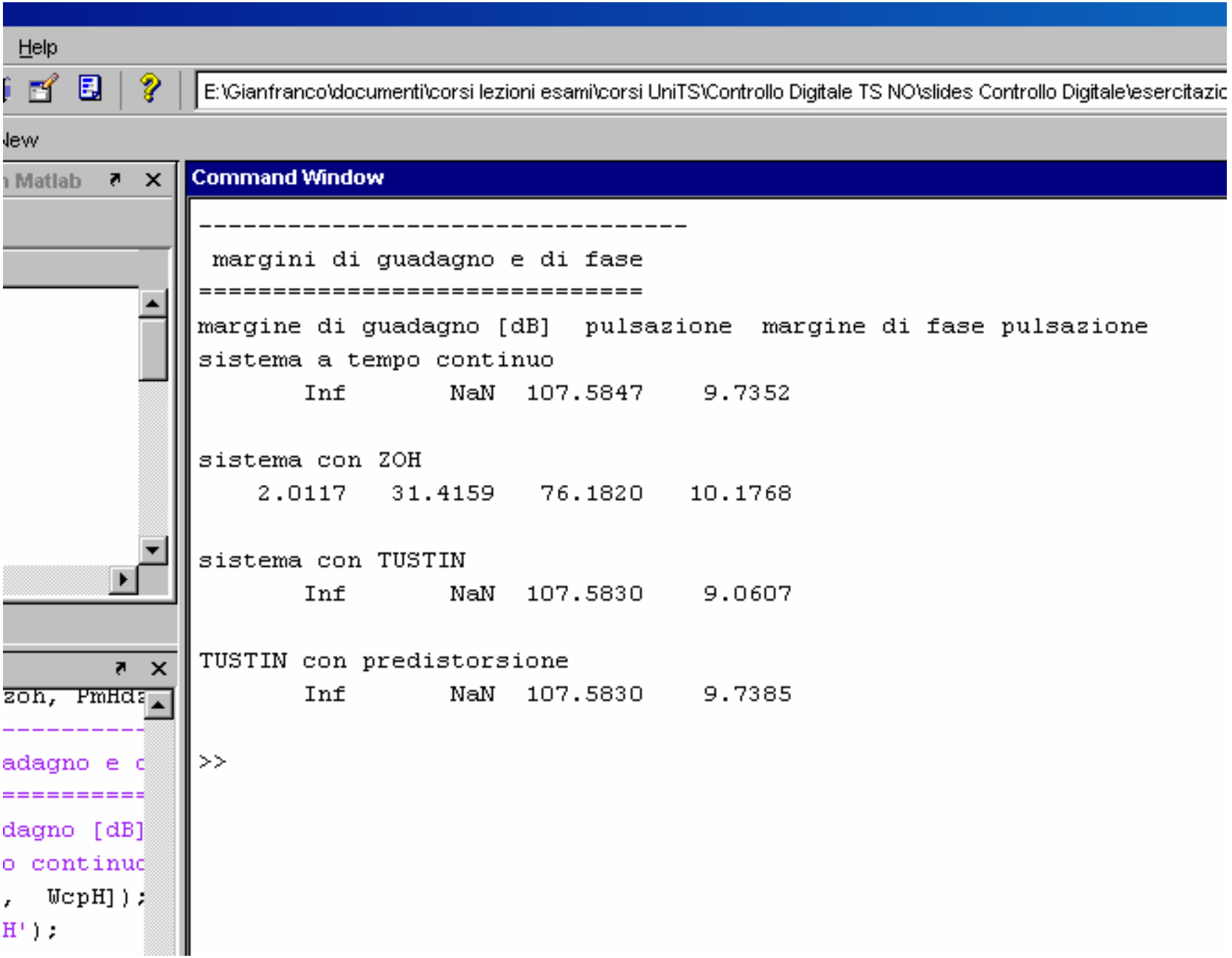
Risposta allo scalino



Risposta in frequenza: diagrammi di Bode



```
» % la stabilita'
» [GmH, PmH, WcgH, WcpH ] = margin(H);
» [GmHdzoh, PmHdzoh, WcgHdzoh, WcpHdzoh ] = margin(Hdzoh);
» [GmHdtustin, PmHdtustin, WcgHdtustin, WcpHdtustin ] =
  margin(Hdtustin);
» [GmHdwarp, PmHdwarp, WcgHdwarp, WcpHdwarp ] =
  margin(Hdwarp );
»
»
» disp('margine di guadagno [dB]  pulsazione  margine di fase
  pulsazione')
» disp('sistema a tempo continuo')
» disp([GmH, WcgH, PmH,  WcpH]);
»
» disp('sistema con ZOH');
» disp([GmHdzoh, WcgHdzoh, PmHdzoh,  WcpHdzoh ]);
» % notare la perdita di margine di guadagno e di fase
»
» disp('sistema con TUSTIN');
» disp([GmHdtustin, WcgHdtustin, PmHdtustin,  WcpHdtustin ]);
» % piccole differenze col sistema a tempo continuo
»
» disp('TUSTIN con predistorsione');
» disp([GmHdwarp, WcgHdwarp, PmHdwarp, WcpHdwarp ]);
» % ancora minori differenze
```


Help
E:\Gianfranco\documenti\corsi lezioni esami\corsi UniTS\Controllo Digitale TS NO\slides Controllo Digitale\esercitazic
Command Window
---
margini di guadagno e di fase
margine di guadagno [dB] pulsazione margine di fase pulsazione
sistema a tempo continuo
Inf NaN 107.5847 9.7352
sistema con ZOH
2.0117 31.4159 76.1820 10.1768
sistema con TUSTIN
Inf NaN 107.5830 9.0607
TUSTIN con predistorsione
Inf NaN 107.5830 9.7385
>>