

# PROGETTO DI LABORATORIO DI AUTOMATICA

Antonio Paonessa, 213428

## ESERCIZIO 1

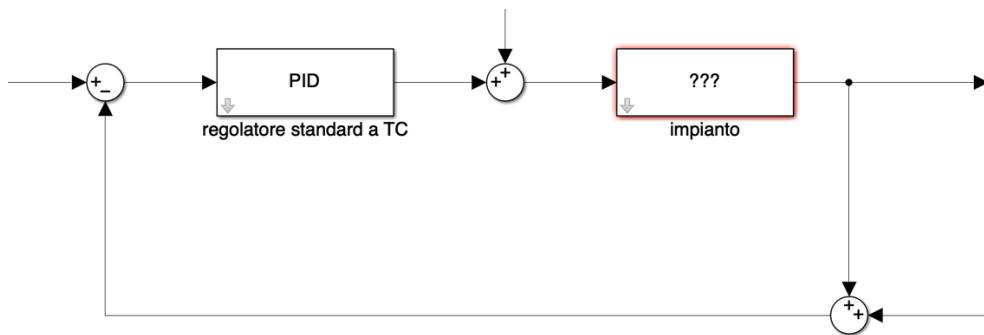
Dato il sistema descritto dalla seguente funzione di trasferimento:

$$G(s) = \frac{4}{s(s + 0.2)(s + 3)}$$

Progettare un regolatore standard digitale in grado di garantire stabilità a ciclo chiuso ed errore nullo rispetto ad un riferimento a gradino.

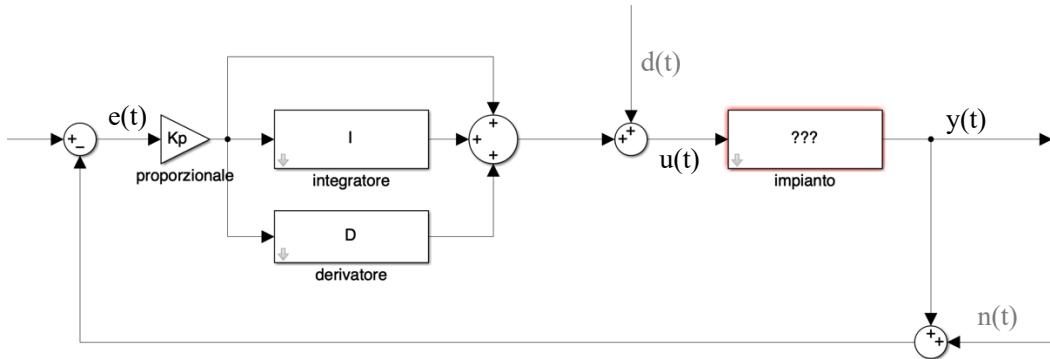
L'idea è quella di progettare un controllore standard a tempo continuo e successivamente *discretizzarlo*.

L'attenzione si focalizzerà sul tarare i parametri dei regolatori standard del tipo P, PI e PID, con **l'ipotesi di non conoscere il modello dell'impianto**.



*Schema di controllo*

Tale regolatore è caratterizzato dalle azioni di tipo **proporzionale, integrale e derivativa**.



*Regolatore standard componenti*

Supponendo  $n(t) = d(t) = 0$ , il **segnale di controllo** sarà dato dalla somma dei contributi:

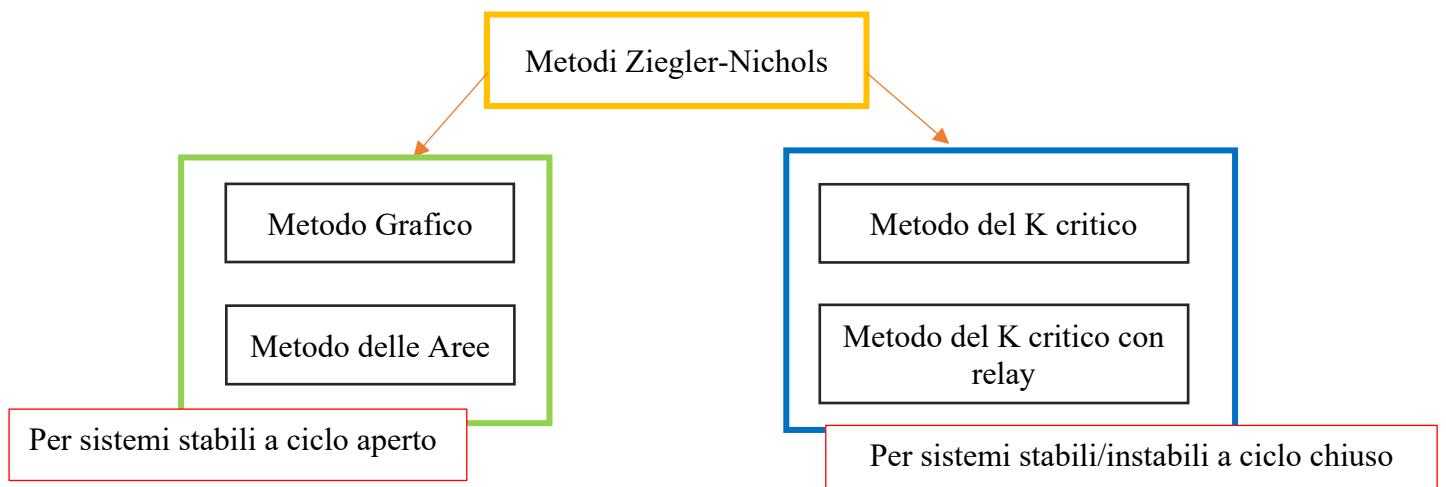
$$u(t) = K_p e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{d}{dt}(e(t))$$

Che nel dominio della trasformata di Laplace:

$$U(s) = K_p + \frac{1}{T_i s} + T_d s$$

La procedura di *tuning* consiste nello scegliere valori ottimali di  $K_p$ ,  $T_i$  e  $T_d$ .

Con l'ipotesi di non conoscere il modello fisico/matematico dell'impianto, allora si ricerca un opportuno metodo di taratura nei cosiddetti *metodi model-free* / *metodi di Ziegler-Nichols*.



Si inizia con una valutazione della risposta al gradino del sistema a ciclo chiuso e aperto

*>> folder Esercizio1 >> script.m*

```

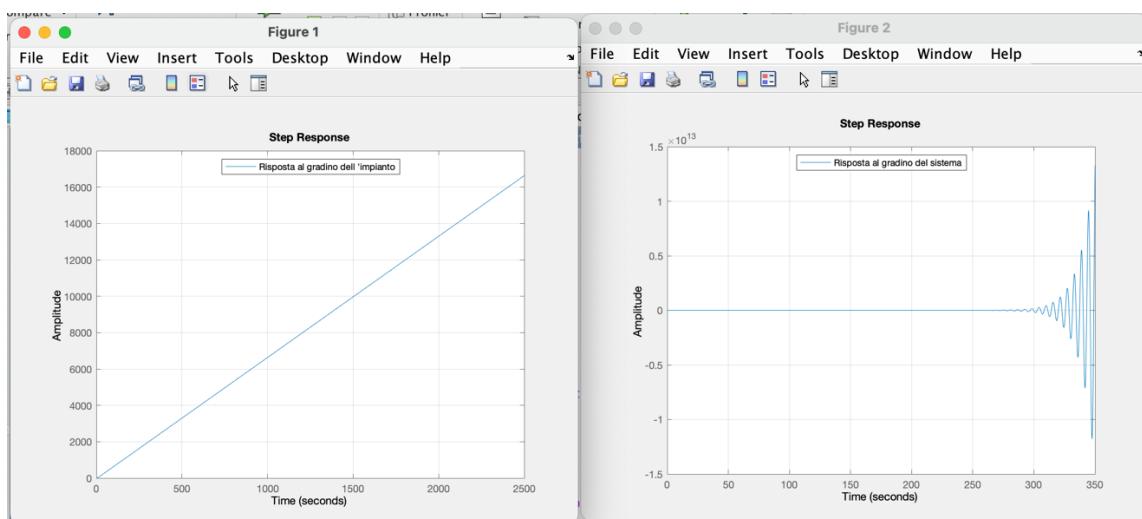
clear
close all

%
% G(s) = 4
%           s(s+0.2)(s+3)

s=zpk('s');
G=4/(s*(s+0.2)*(s+3))

figure(1)
step(G)
grid
legend('Risposta al gradino dell ''impianto'', 'Location', 'north')

figure(2)
T=feedback(G,1);
step(T);
grid
legend('Risposta al gradino del sistema', 'Location', 'north')
  
```

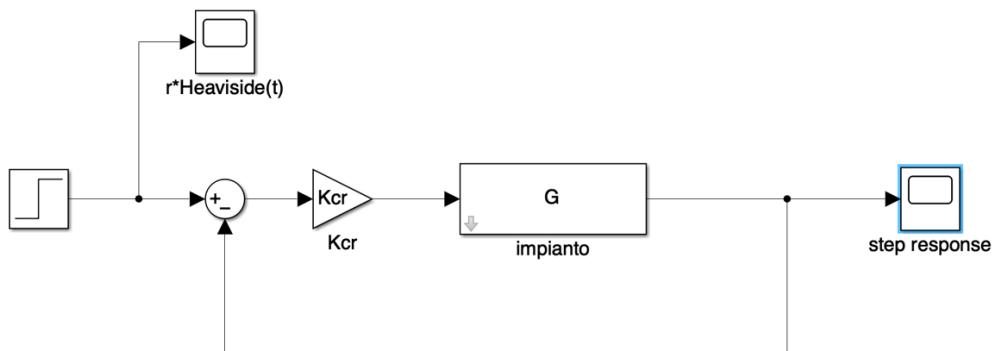


Risulta evidente l'instabilità del sistema, quindi si passa alla fase di taratura dei parametri.

## Metodo del K critico

Si considera il seguente schema:

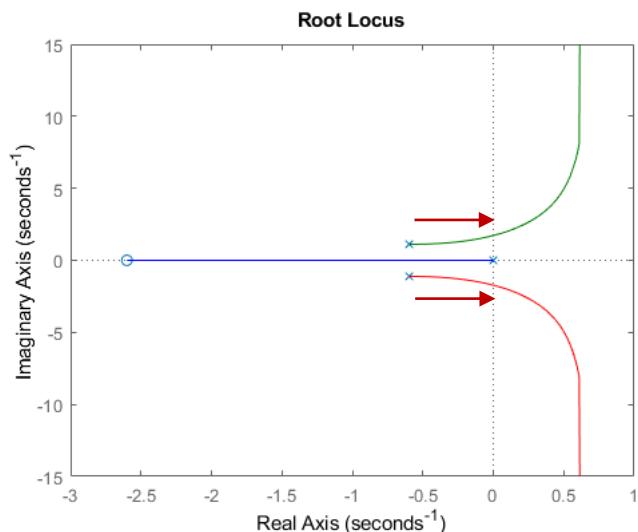
>> Esercizio1 >> valutazione\_K\_cr.slx



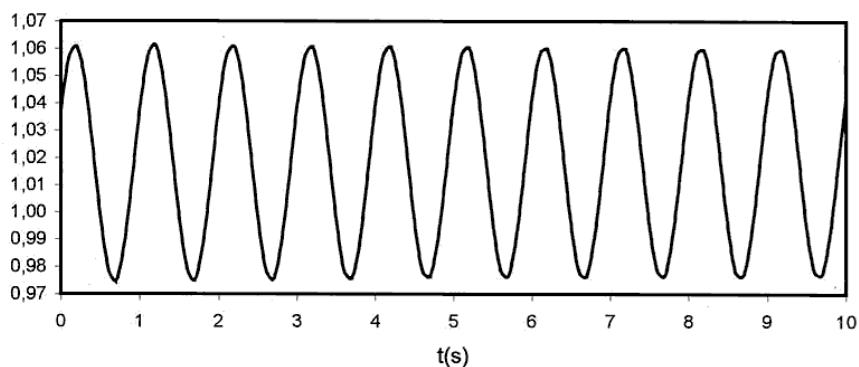
Metodo di Ziegler-Nichols ad anello chiuso

Si determina sperimentalmente per quale valore del guadagno  $K_{cr}$  la risposta del sistema retro-azionato genera oscillazioni autosostenute e permanenti.

Tarando il guadagno  $K_{cr}$  in modo di ‘avvicinare’ il più possibile i poli dominati all’asse immaginario



Con l’obiettivo di arrivare al limite di stabilità e di avere oscillazioni permanenti nella risposta.



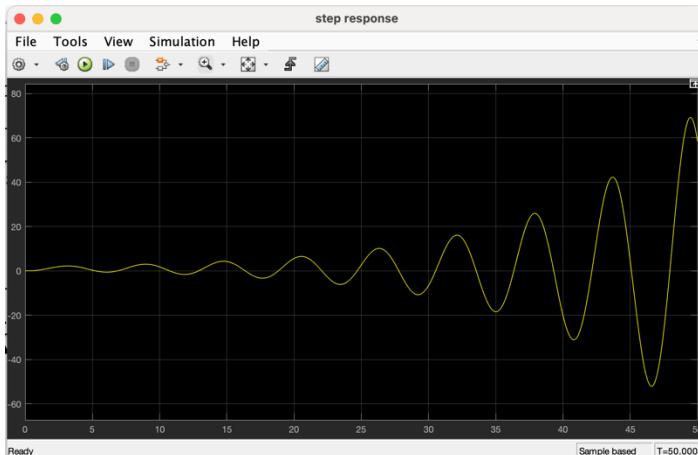
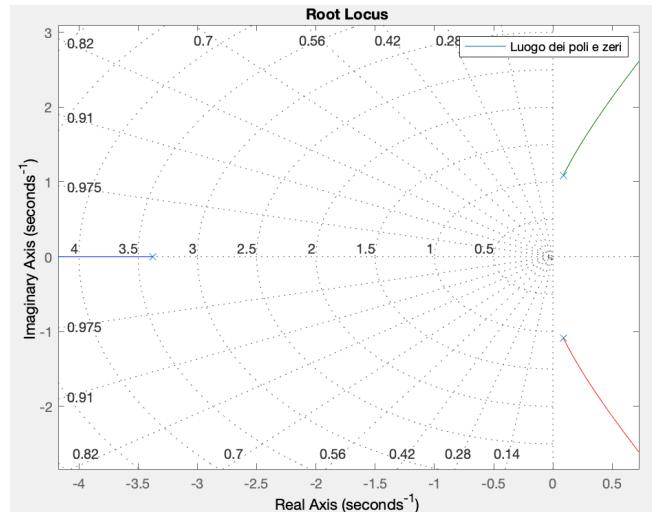
In questo caso si inizia con un guadagno critico unitario:

```
% Ampiezza del gradino
r=1;

%% Metodo del K-Critico
fprintf('\nMetodo del K critico con:\n')

Kcr=1;
fprintf('#Kcr: %5.2f \n',Kcr)

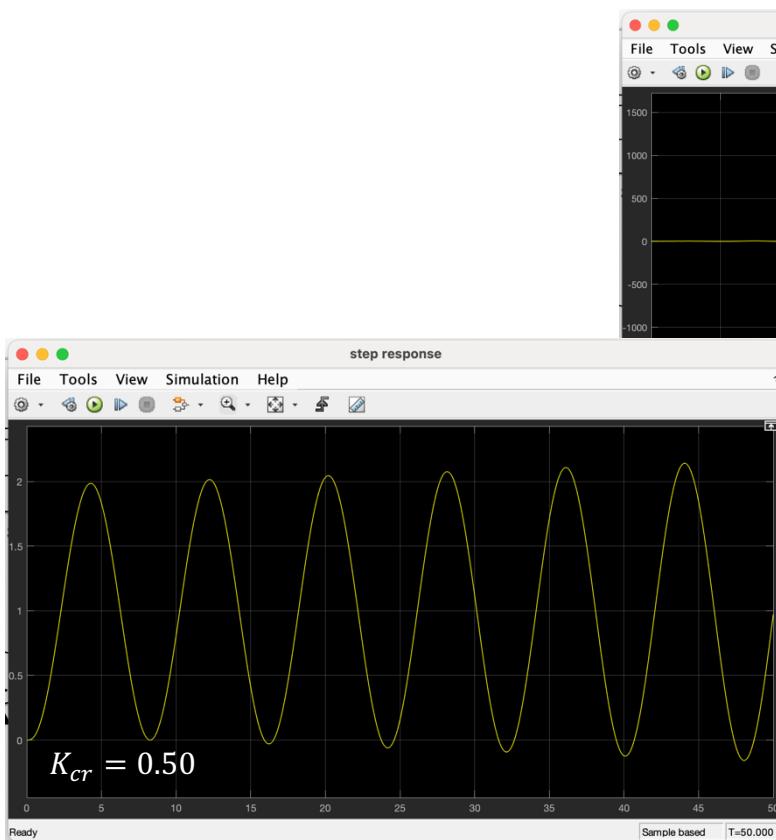
[num,den]=tfdata(feedback(Kcr*G,1));
figure(3)
rlocus(num,den)
legend('Luogo dei poli e zeri')
grid
```



Valutando la risposta al gradino e il luogo delle radici, ci si rende conto di essere molto vicini da 'destra' all'asse immaginario.

Quindi ci si aspetta di dover **diminuire** il guadagno critico per raggiungere ottenere la configurazione di interesse.

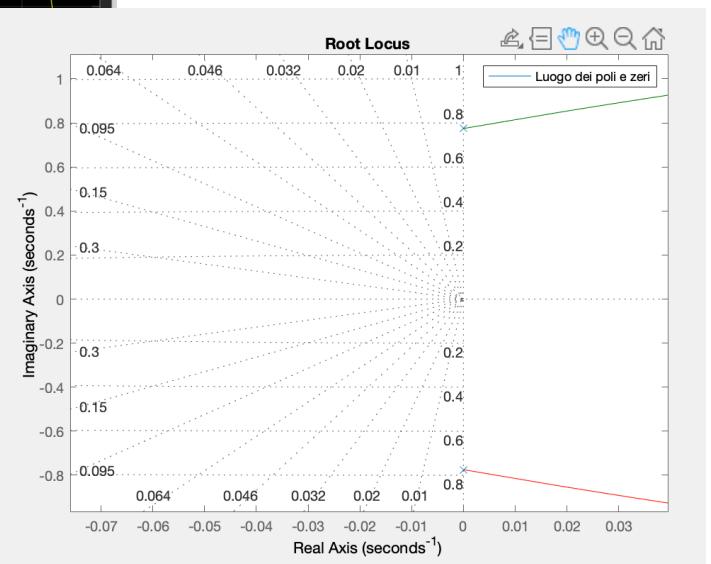
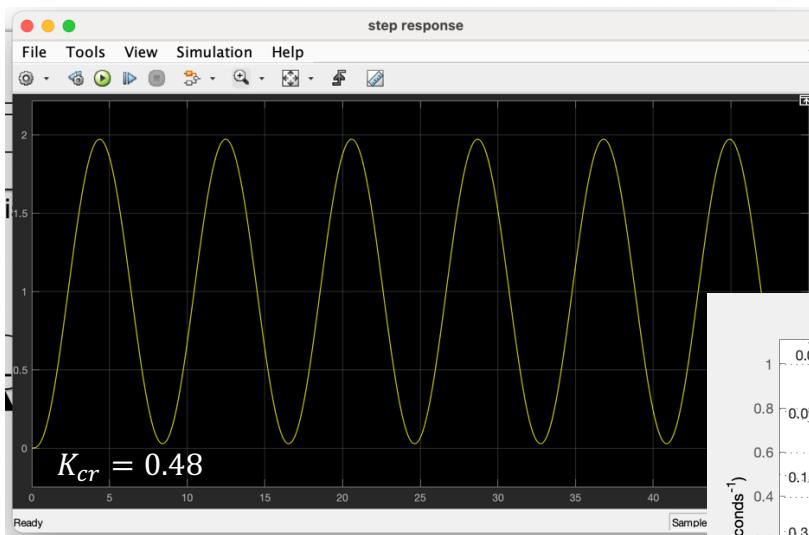
Per verificare tale osservazione si effettua un test prima per  $K_{cr} = 1.50$  e poi con  $K_{cr} = 0.50$ .



Come ipotizzato si ottengono oscillazioni permanenti per  $K_{cr} = 0.50$ .

Ma si può fare di meglio.

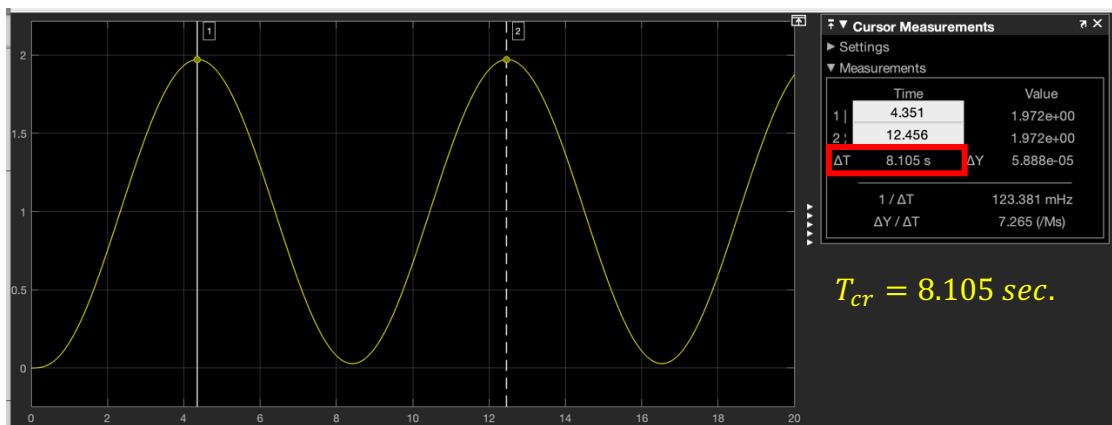
Infatti, per:



Si ottiene la configurazione di interesse.

Successivamente si determina il secondo parametro necessario per la taratura del PID, cioè il cosiddetto *periodo di oscillazione critico*  $T_{cr} = \frac{2\pi}{\omega_{cr}}$ , in cui  $\omega_{cr}$  è la pulsazione delle oscillazioni autosostenute.

In questo caso valuto in periodo di oscillazione sul grafico della risposta al gradino:



Caricati nello script  $K_{cr}$  e  $T_{cr}$ , si valutano i cosiddetti valori *di primo tentativo* dalle [formule di taratura di Ziegler-Nichols](#):

	$K_p$	$T_i$	$T_d$
'P'	$0.5 K_{cr}$		
'PI'	$0.45 K_{cr}$	$0.75 T_{cr}$	
'PID'	$0.6 K_{cr}$	$0.5 T_{cr}$	$0.12 T_{cr}$

```

%% Metodo del K-Critico
fprintf('\nMetodo del K critico con:\n')

Kcr=0.480;
fprintf('#Kcr: %5.2f \n',Kcr)

[num,den]=tfdata(feedback(Kcr*G,1));
figure(1)
rlocus(num,den)
legend('Luogo dei poli e zeri')
grid

pause

Tcr=8.105;
fprintf('#Tcr: %5.2f \n',Tcr)

```

Command window

```

Metodo del K critico con:
#Kcr: 0.48
#Tcr: 8.11

```

Script per il calcolo dei valori di primo tentativo mediante la tabella di Ziegler-Nichols.

*taratura\_k\_critico.m*

```

function [Kp,Ti,Td] = taratura_k_critico(K_cr,T_cr,string);

fprintf('\nTaratura del regolatore %s\n',string)
Kp=0;
Ti=0;
Td=0;

if strcmp(string,'P')==1
    Kp=0.50*K_cr;
    fprintf(' 1)Kp=%3.3f \n',Kp)
    return
end

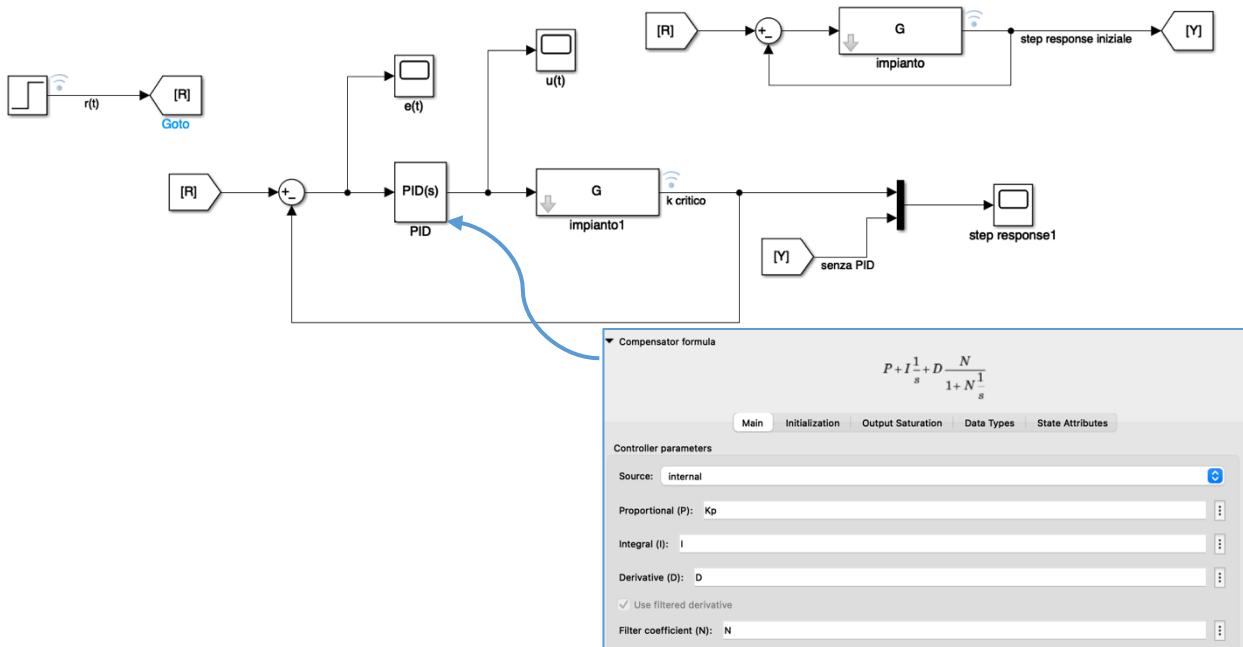
if strcmp(string,'PI')==1
    Kp=0.45*K_cr;
    fprintf(' 1)Kp=%3.3f \n',Kp)
    Ti=0.75*T_cr;
    fprintf(' 2)Ti=%3.3f \n',Ti)
    return
end

if strcmp(string,'PID')==1
    Kp=0.60*K_cr;
    fprintf(' 1)Kp=%3.3f \n',Kp)
    Ti=0.50*T_cr;
    fprintf(' 2)Ti=%3.3f \n',Ti)
    Td=0.12*T_cr;
    fprintf(' 3)Td=%3.3f \n',Td)
    return
end

```

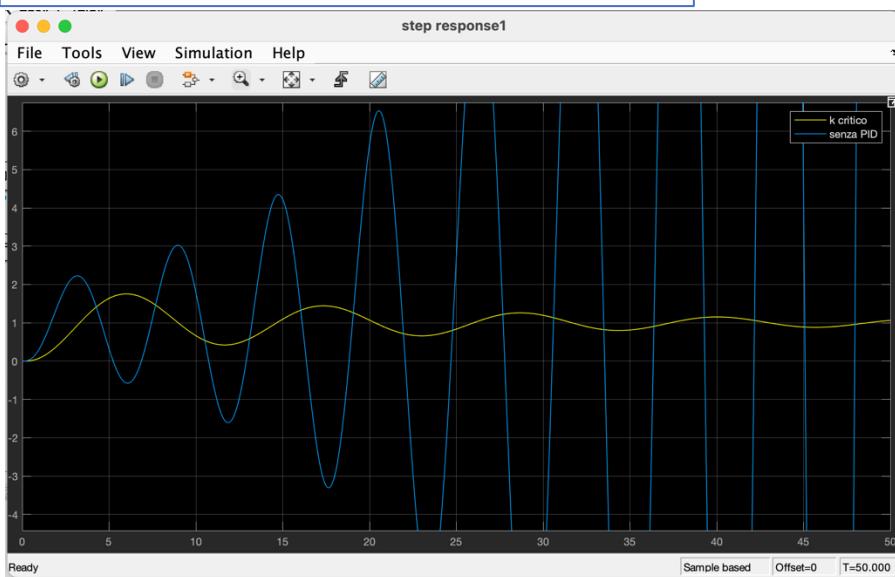
Si passa alla fase di valutazione della risposta del sistema con i valori di primo tentativo con il seguente schema:

>> Esercizio1 >> modello\_K\_easy\_relay.slx



Calcolo e valutazione dei regolatori 'P', 'PI' e 'PID':

```
[Kp,Ti,Td] = taratura_k_critico(Kcr,Tcr,'P');
```

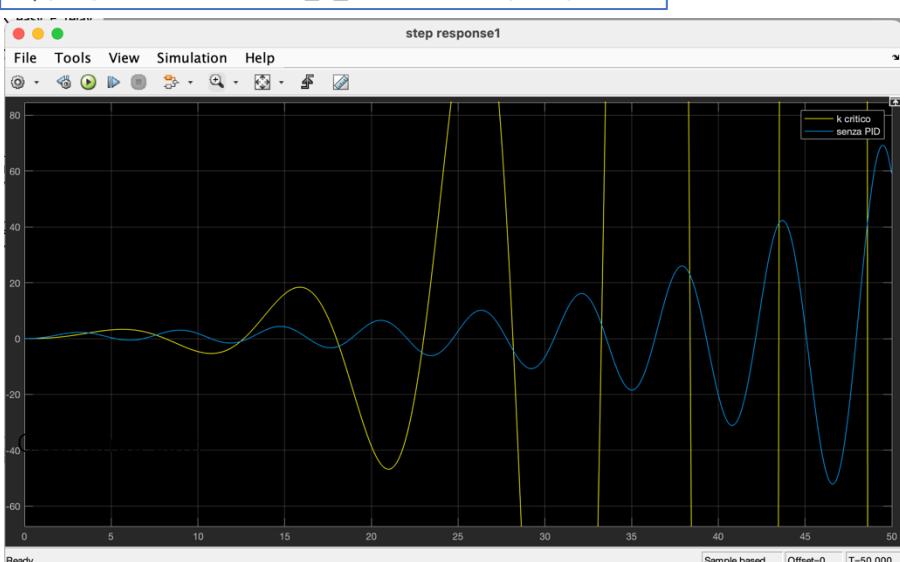


>> script

Metodo del K critico con:  
#Kcr: 0.48  
#Tcr: 8.11

Valori di primo tentativo  
Taratura del regolatore P  
1)Kp=0.240

```
[Kp,Ti,Td] = taratura_k_critico(Kcr,Tcr,'PI');
```



>> script

Metodo del K critico con:  
#Kcr: 0.48  
#Tcr: 8.11

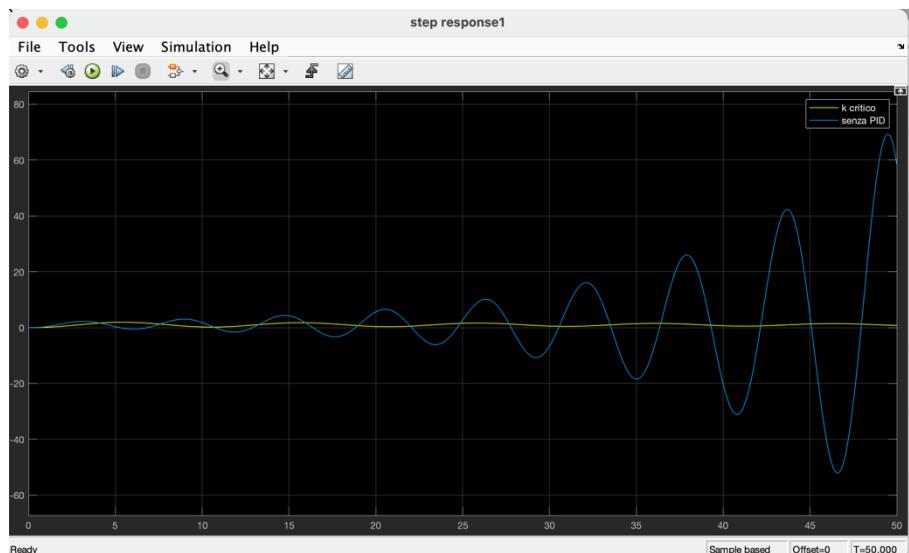
Valori di primo tentativo  
Taratura del regolatore PI  
1)Kp=0.216  
2)Ti=6.079

Per migliorare la risposta si manipolano i valori di primo tentativo.

Diminuendo il guadagno integrale  $I = \frac{1}{T_i}$  ed aumentando leggermente l'azione proporzionale:

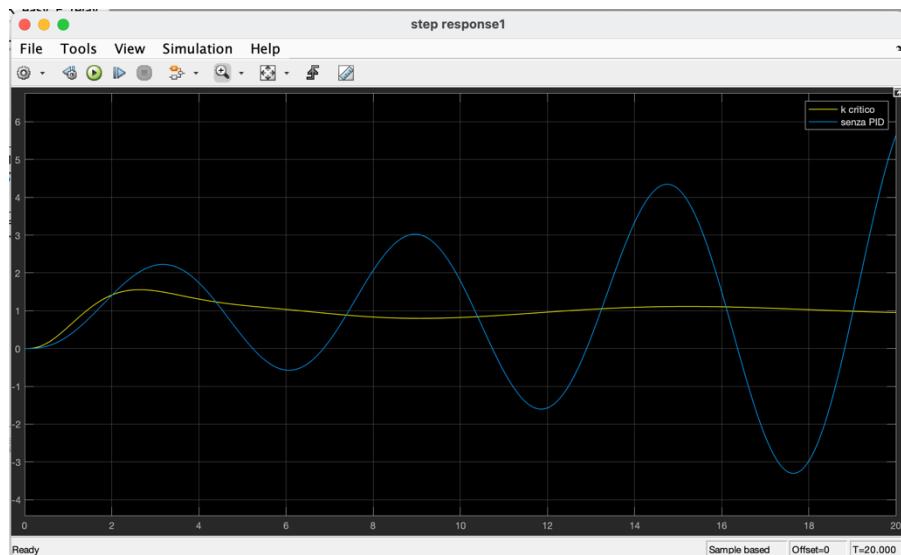
Per  $T_i = 100$  e  $K_p = 0.3$  si ottiene la seguente risposta:

Essendo i parametri che si calcolano dalla tabella di Ziegler-Nichols dei valori di primo tentativo, essi sono presi come riferimento per migliorare o soddisfare le specifiche statiche e dinamiche.



Infine, si determina il regolatore standard completo:

```
[Kp,Ti,Td] = taratura_k_critico(Kcr,Tcr,'PID');
```



```
>> script
Metodo del K critico con:
#Kcr: 0.48
#Tcr: 8.11

Valori di primo tentativo
Taratura del regolatore PID
1)Kp=0.288
2)Ti=4.053
3)Td=0.973
```

Come ci si aspetta la combinazione di tutte le azioni del regolatore standard, anche se con valori di primo tentativo, danno un risultato migliore rispetto ai regolatori 'parziali' P e PI.

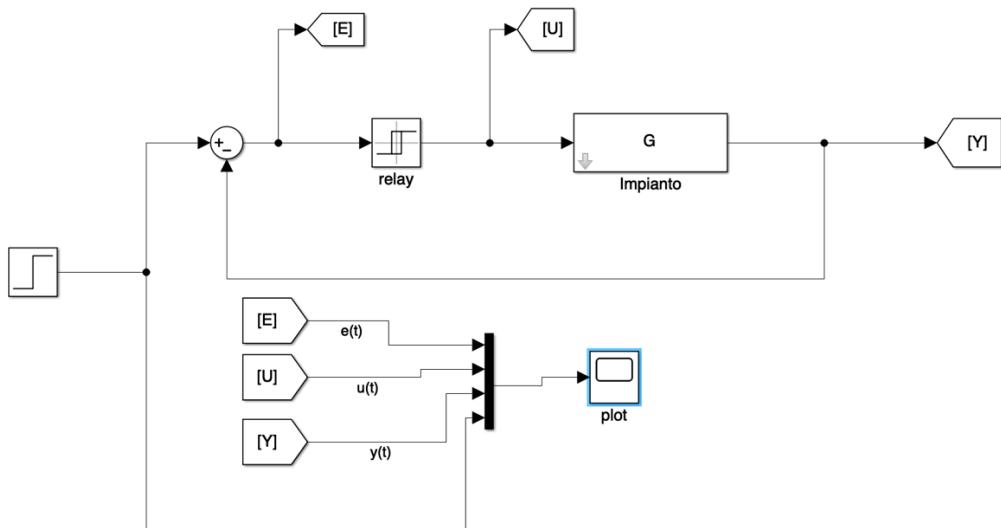
Tuttavia, anche se 'P', 'PI' e 'PID' soddisfano la specifica statica e la stabilità richiesta dall'esercizio, risulta evidente come tale approccio rischia di compromettere seriamente il sistema che ipotizziamo di non conoscere.

Un'alternativa al *metodo del k critico* è con l'uso del cosiddetto **relè**.

## Metodo del K critico con relè

Si considera il seguente schema:

`>> Esercizio1 >> valutazione_K_cr_Tcr.slx`

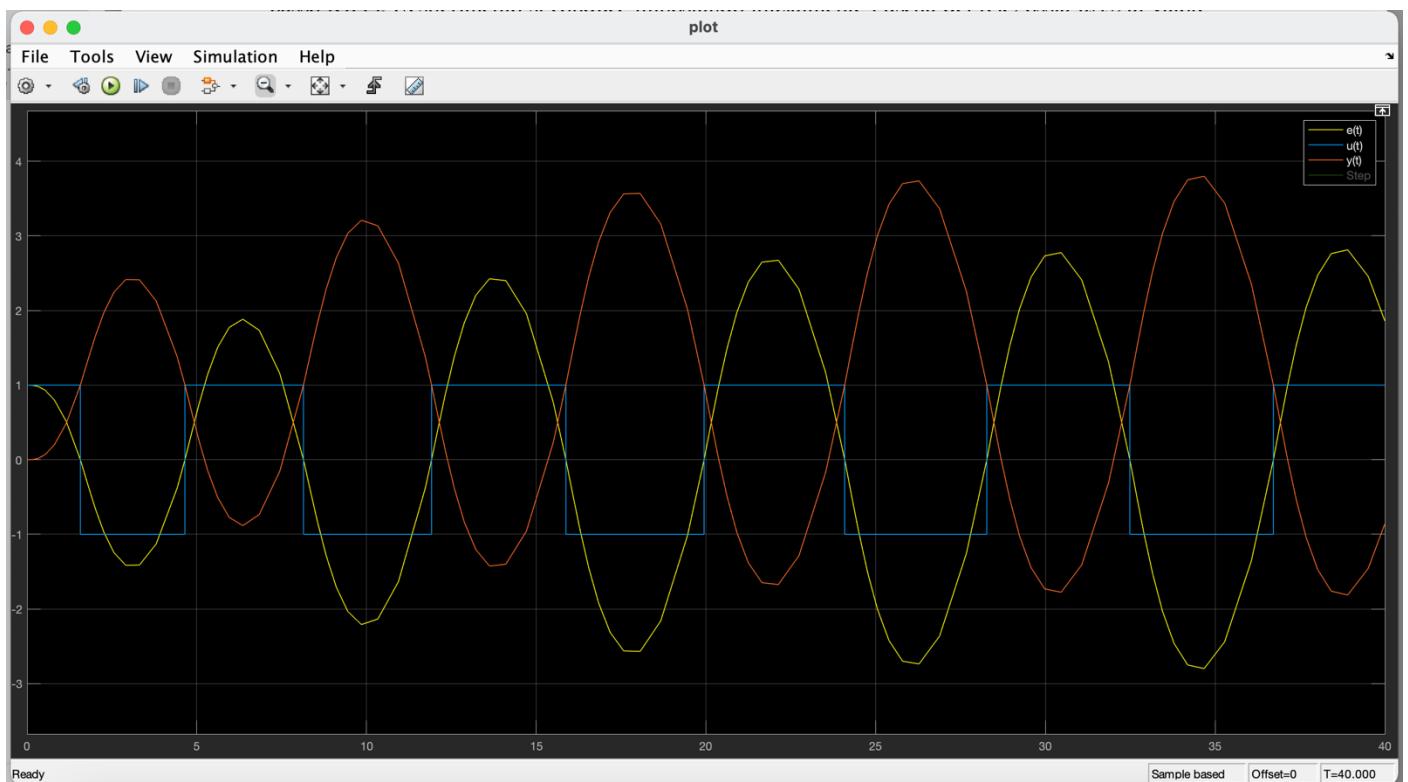


Metodo di Ziegler-Nichols ad anello chiuso con relay

Il contributo guadagno critico viene sostituito da un **relè** a due stati ON-OFF e la logica di funzionamento di tale componente che ha come ingresso  $e(t)$  e uscita  $u(t)$  è la seguente:

$$\begin{cases} \text{per } e(t) > 0 & u(t) = +r \\ \text{per } e(t) < 0 & u(t) = -r \end{cases}$$

Imponendo un'oscillazione permanente con ampiezza controllata per  $y(t)$ , simulando, in altre parole, il comportamento del sistema ai margini della stabilità.

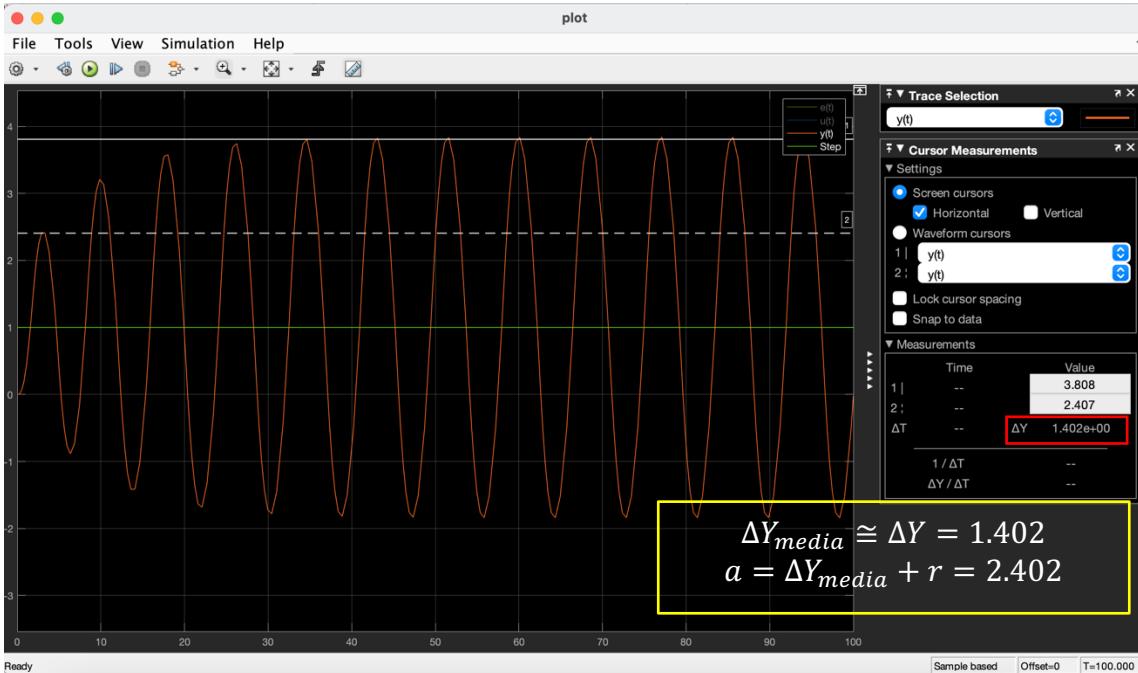
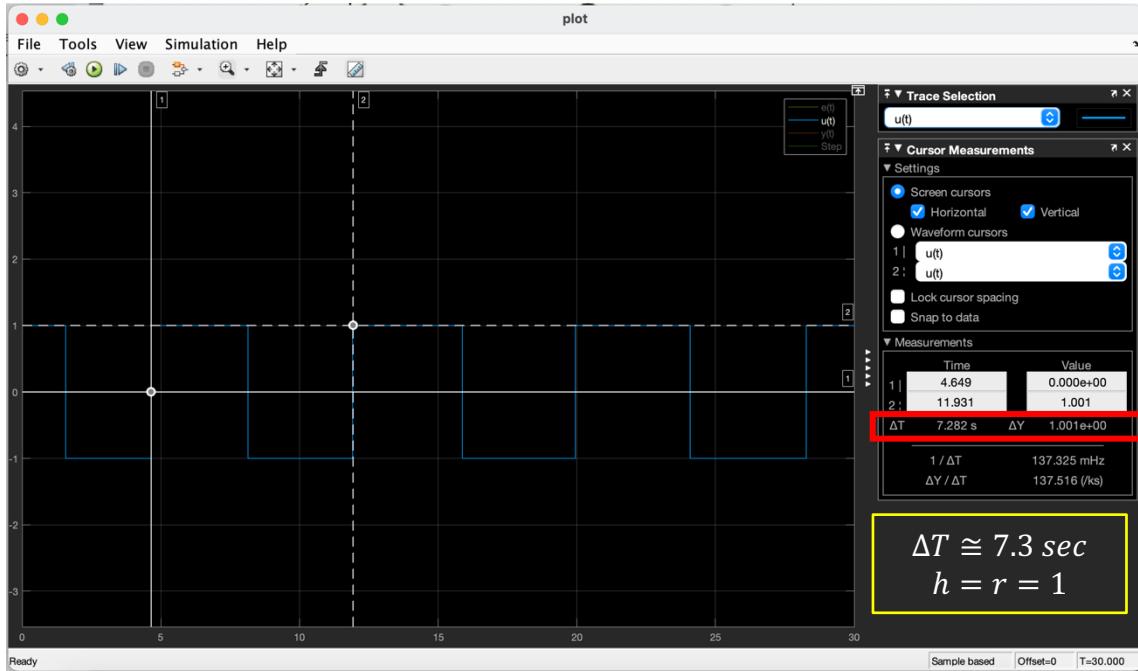


Si ricorda che  $r$  è rappresentata l'ampiezza del gradino di riferimento!

I parametri  $K_{cr}$  e  $T_{cr}$  si determinano dai valori:

$$\left\{ \begin{array}{l} \Delta T : \text{periodo dell'onda quadra } u(t) \\ h : \text{ampiezza dell'onda quadra } u(t) \\ a : \text{ampiezza dell'oscillazione controllata } y(t) - \text{valore di regime } r \end{array} \right.$$

Nel caso in analisi si hanno:



Da questi si ricavano:

$$\left\{ \begin{array}{l} T_{cr} = \Delta T \\ K_{cr} = \frac{4h}{a\pi} \end{array} \right.$$

E i valori di primo tentativo si determinano mediante la tabella di Ziegler-Nichols.

>> Esercizio 1 >> script.m e modello\_K\_easy\_E\_relay.slx

```

%% Metodo del K-Critico con relay
fprintf('\nMetodo del K critico + Relay con:\n')
a=2.782;
T=7.328;
h=r;

Kcr_relay=(4*h)/(a*pi);
fprintf('#Kcr: %5.2f \n',Kcr_relay)

Tcr_relay=T;
fprintf('#Tcr: %5.2f \n',Tcr_relay)

pause

[Kp_relay,Ti_relay,Td_relay] = taratura_k_critico(Kcr_relay,Tcr_relay,'PID');

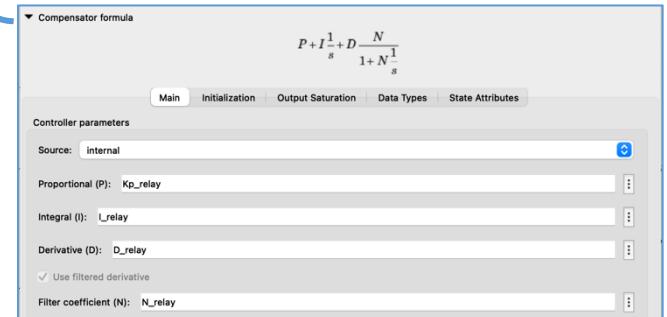
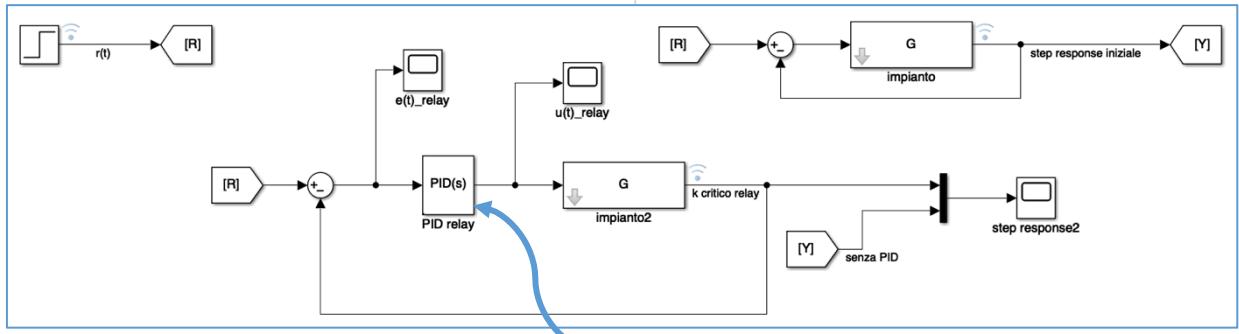
```

```

if Ti_relay~=0
    I_relay=1/Ti_relay;
else
    I_relay=0;
end
D_relay=Td;
N_relay=3;

```

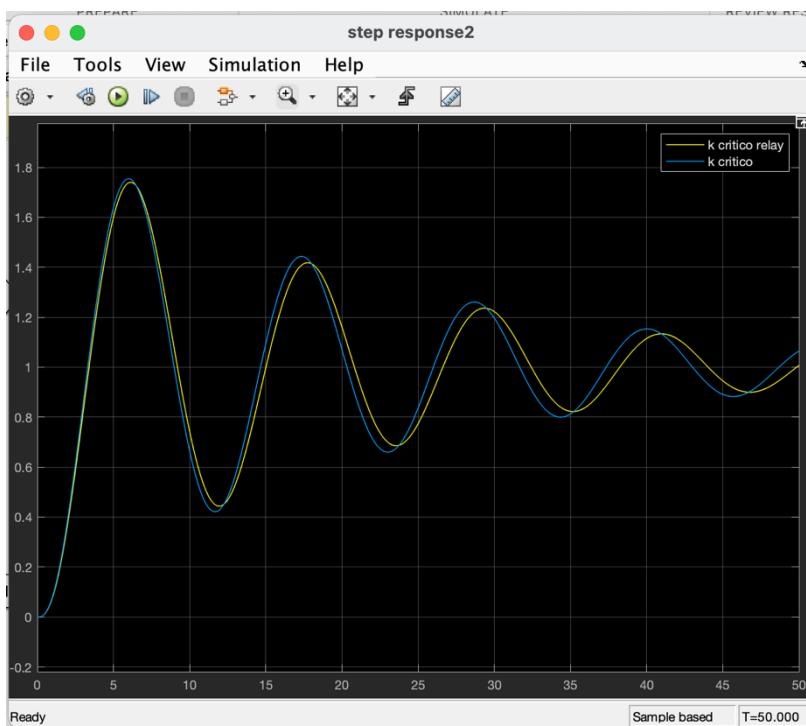
Metodo del K critico + Relay con:  
#Kcr: 0.46  
#Tcr: 7.33



Si passa alla costruzione dei relativi regolatori 'P', 'PI' e 'PID', effettuando un confronto con i regolatori discussi precedentemente.

```
[Kp,Ti,Td] = taratura_k_critico(Kcr,Tcr,'P');
```

```
[Kp_relay,Ti_relay,Td_relay] = taratura_k_critico(Kcr_relay,Tcr_relay,'P');
```



```

>> script

Metodo del K critico con:
#Kcr: 0.48
#Tcr: 8.11

Valori di primo tentativo
Taratura del regolatore P
1)Kp=0.240

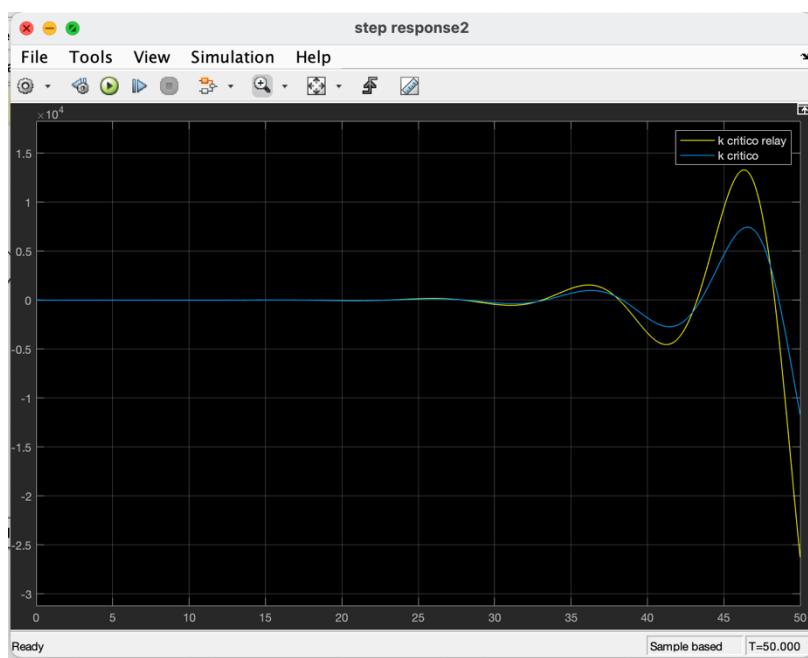
Metodo del K critico + Relay con:
#Kcr: 0.46
#Tcr: 7.33

Taratura del regolatore P
1)Kp=0.229

```

Dai guadagni proporzionali, si intuisce che le due risposte del sistema saranno molto simili; tuttavia, risulta leggermente migliore la risposta col k-critico senza relay (avendo un'azione proporzionale maggiore).

```
[Kp,Ti,Td] = taratura_k_critico(Kcr,Tcr,'PI');
[Kp_relay,Ti_relay,Td_relay] = taratura_k_critico(Kcr_relay,Tcr_relay,'PI');
```



```
>> script
Metodo del K critico con:
#Kcr: 0.48
#Tcr: 8.11

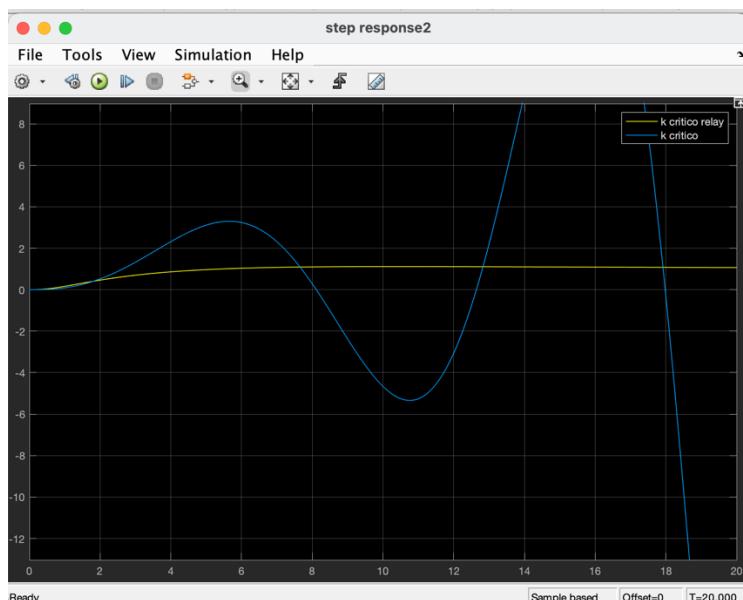
Valori di primo tentativo
Taratura del regolatore PI
1)Kp=0.216
2)Ti=6.079

Metodo del K critico + Relay con:
#Kcr: 0.46
#Tcr: 7.33

Taratura del regolatore PI
1)Kp=0.206
2)Ti=5.496
```

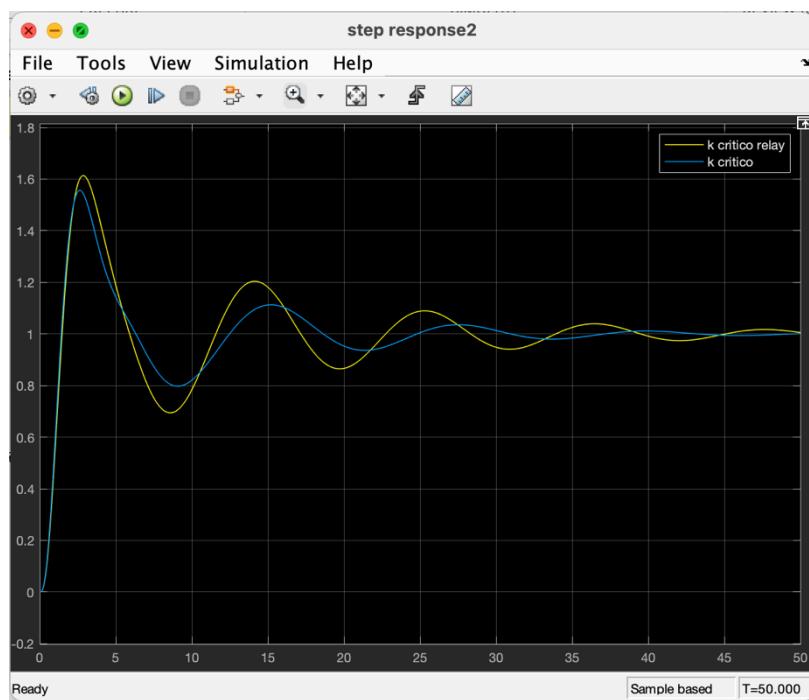
Come nel caso del regolatore 'PI' discusso con il metodo del k-critico senza relay, si procede con una taratura sperimentale dei valori di partenza.

Per  $T_{i,relay} = 333$  e  $K_{p,relay} = 0.07$



```
[Kp,Ti,Td] = taratura_k_critico(Kcr,Tcr,'PID');
```

```
[Kp_relay,Ti_relay,Td_relay] = taratura_k_critico(Kcr_relay,Tcr_relay,'PID');
```



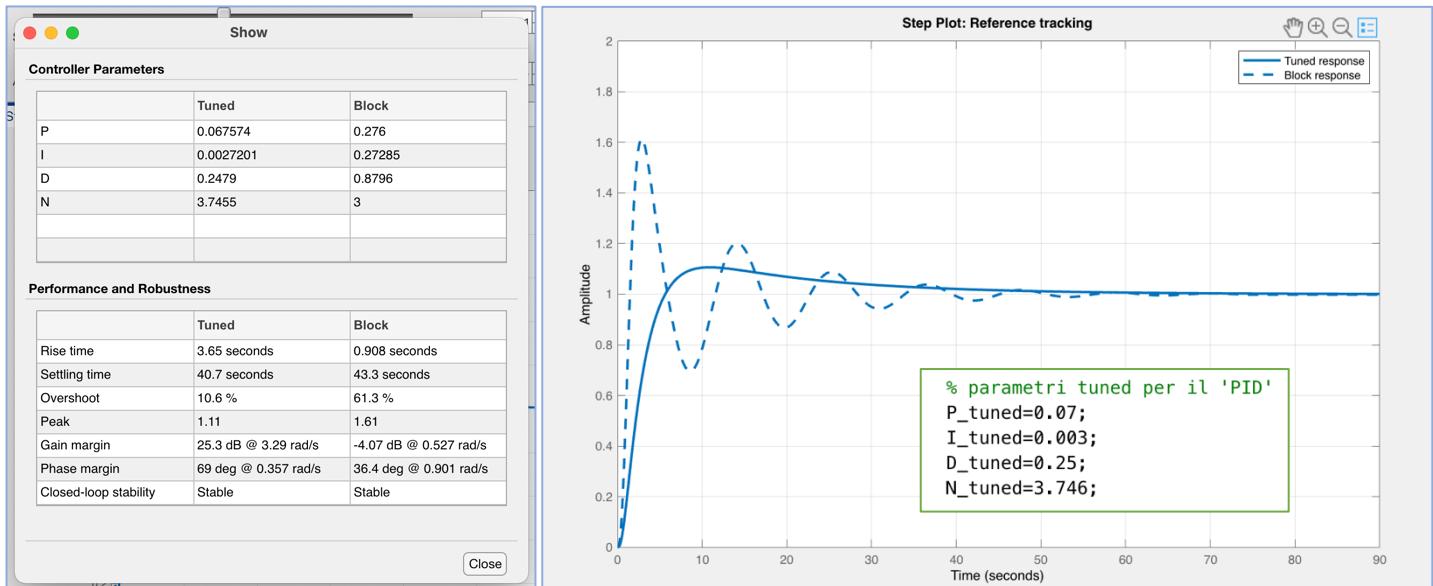
```
>> script
Metodo del K critico con:
#Kcr: 0.48
#Tcr: 8.11

Valori di primo tentativo
Taratura del regolatore PID
1)Kp=0.288
2)Ti=4.053
3)Td=0.973

Metodo del K critico + Relay con:
#Kcr: 0.46
#Tcr: 7.33

Taratura del regolatore PID
1)Kp=0.275
2)Ti=3.664
3)Td=0.879
```

Con l'uso dell'Automated Tuning di matlab, si ottengono i seguenti parametri e risposta al gradino



Facendo un confronto tra 'P', 'PI' e 'PID' e 'PID\_tuned', è evidente come l'uso di tutti e tre i contributi (proporzionale, integrale e derivativo) riduca sovra elongazione, tempo di assestamento e tempo di salita, considerando i valori di primo tentativo.

```
% confronto tra P,PI,PID e PID tuned
P=0; TI=0; TD=0;
str=["P","PI","PID"];
figure(1)
hold on
for i=1:3
    if strcmp(str(:,i),'PI')==1 % PI tuned
        P=0.3;
        I=1/100;
    else
        [P, TI, TD] = taratura_k_critico(Kcr_relay, Tcr_relay, str(:,i));
    end
    C=regolatore(P, TI, TD, N_relay);
    T=feedback(series(C, G), 1);
    step(T)
end
C=regolatore(P_tuned,(1/(I_tuned*P_tuned)),D_tuned/P_tuned,N_tuned)
T=feedback(series(C, G), 1);
step(T)
legend('P','PI','PID','PID_tuned','Location','NorthEast')
grid
```

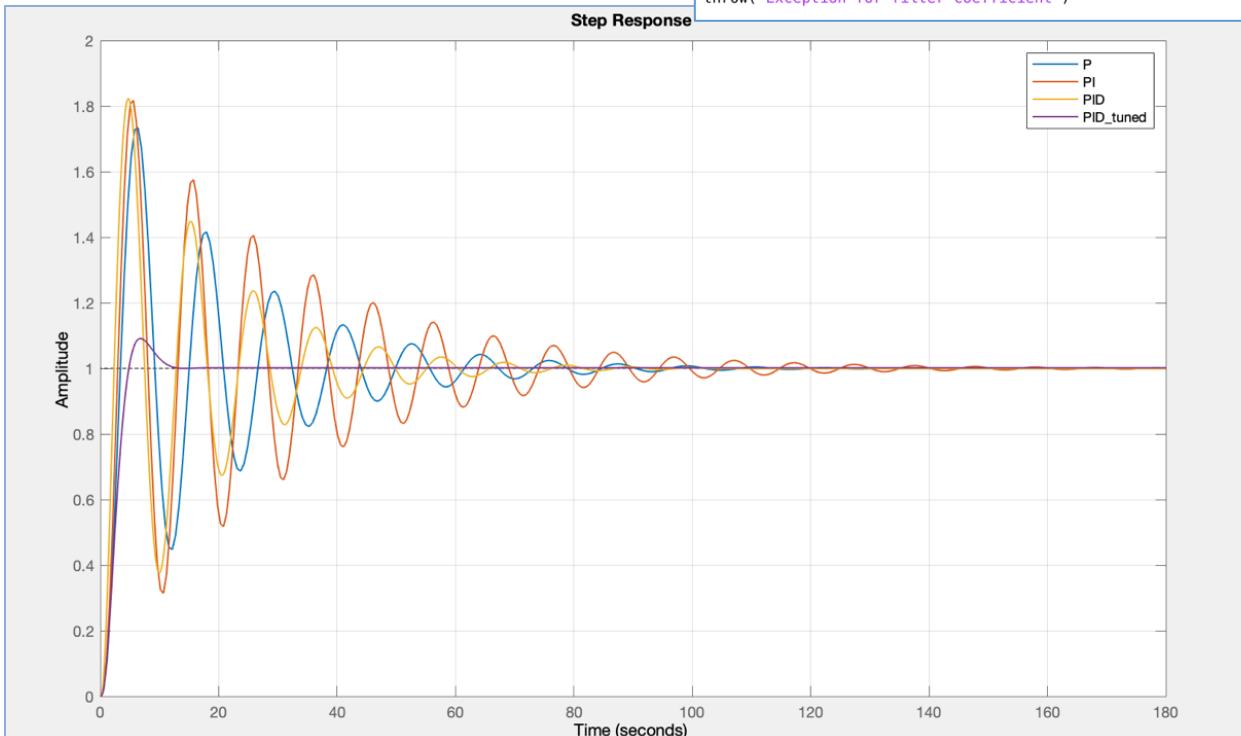
regolatore.m

```
function C=regolatore(Kp,Ti,Td,N);
s=zpk('s');
if N~=0
    if Ti~=0 || Td~=0
        C=Kp + (Kp)/(Ti*s) + (Kp*Td*s)/(1+(Td*s)/N); % derivatore realizzabile
        return
    end

    if Ti~=0 & Td==0
        C=Kp + (Kp)/(Ti*s)
        return
    end

    if Ti==0 & Td~=0
        C=Kp + (Kp*Tds)/(1+(Td*s)/N);
        return
    end

    if Ti==0 & Td==0
        C=Kp;
        return
    end
end
throw('Exception for filter coefficient')
```



Si passa alla fase di discretizzazione del controllore standard, andando a considerare il PID i cui contributi proporzionale, integrale e derivativo coincidano con quelli *tuned*, precedentemente ricavati e memorizzati nel *work-space* di matlab.

```
%% Discretizzazione
fprintf('\nValori tuned:\n')
fprintf(' P=%3.4f\n',P_tuned)
fprintf(' I=%3.4f\n',I_tuned)
fprintf(' D=%3.4f\n',D_tuned)
fprintf(' N=%3.4f\n',N_tuned)

fprintf('\nControllore a tempo continuo\n')
C=regolatore(P_tuned,(1/(I_tuned*P_tuned)),D_tuned/P_tuned,N_tuned);
```

Per prima cosa è necessario scegliere un opportuno *tempo di campionamento*  $T_c$ , che risulta essere un parametro fondamentale, la cui scelta influenza le prestazioni del sistema di controllo a tempo discreto.

Nel seguente caso, si determina il tempo di campionamento mediante il *tempo di salita del sistema*  $t_r$ .

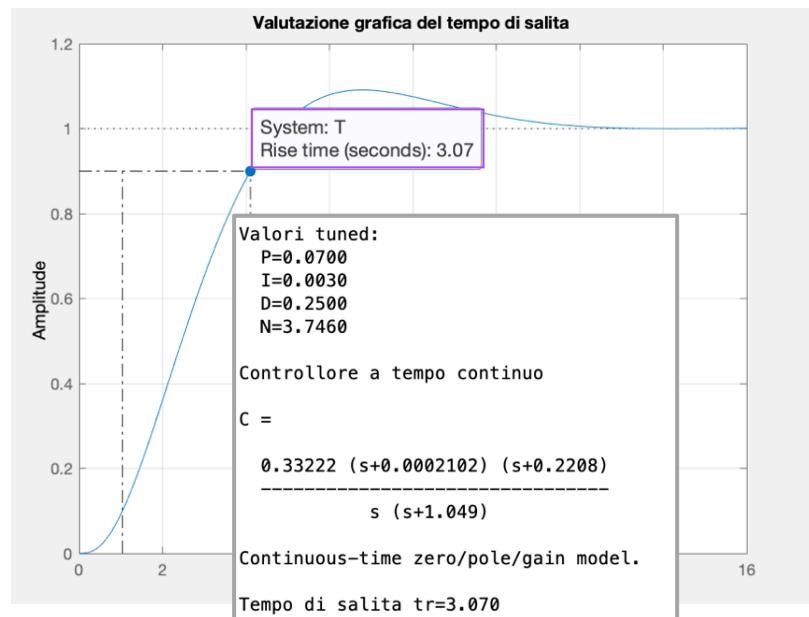
$$\frac{t_r}{20} \leq T_c \leq \frac{t_r}{10}$$

```
T=feedback(series(C,G),1);

figure(1)
step(T)
title('Valutazione grafica del tempo di salita')
grid

pause

info=stepinfo(T);
tr=info.RiseTime;
fprintf('Tempo di salita tr=%3.3f',tr)
```



Quindi:

```
fprintf('\nIl tempo di campionamento è un valore compreso tra: %3.3f e %3.3f\n',(tr/20),(tr/10))
Tc=0.31;
fprintf('\nSi è scelto Tc=%3.3f\n',Tc)
```

Il tempo di campionamento è un valore compreso tra: 0.153 e 0.307  
 Si è scelto Tc=0.200

Per la discretizzazione si farà uso del seguente schema:

$$s = \frac{2z - 1}{T_c z + 1}$$

Che in letteratura è nota come *Trasformazione di Tustin*, in cui il semipiano sinistro della *trasformata di Laplace* si mappa nel piano della *trasformata Z* all'interno di una circonferenza centrata nell'origine e di raggio unitario.

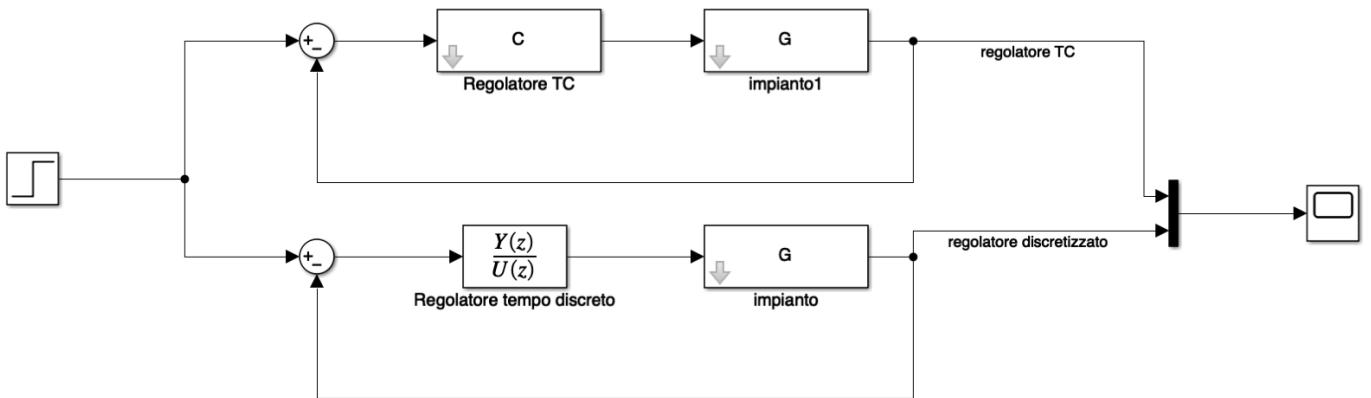
```
fprintf('\nDiscretizzazione del regolatore\n')
C_z=c2d(C,Tc,'tustin')
C_z=tf(C_z);
[numer,denom]=tfdata(C_z);
```

**Discretizzazione del regolatore**

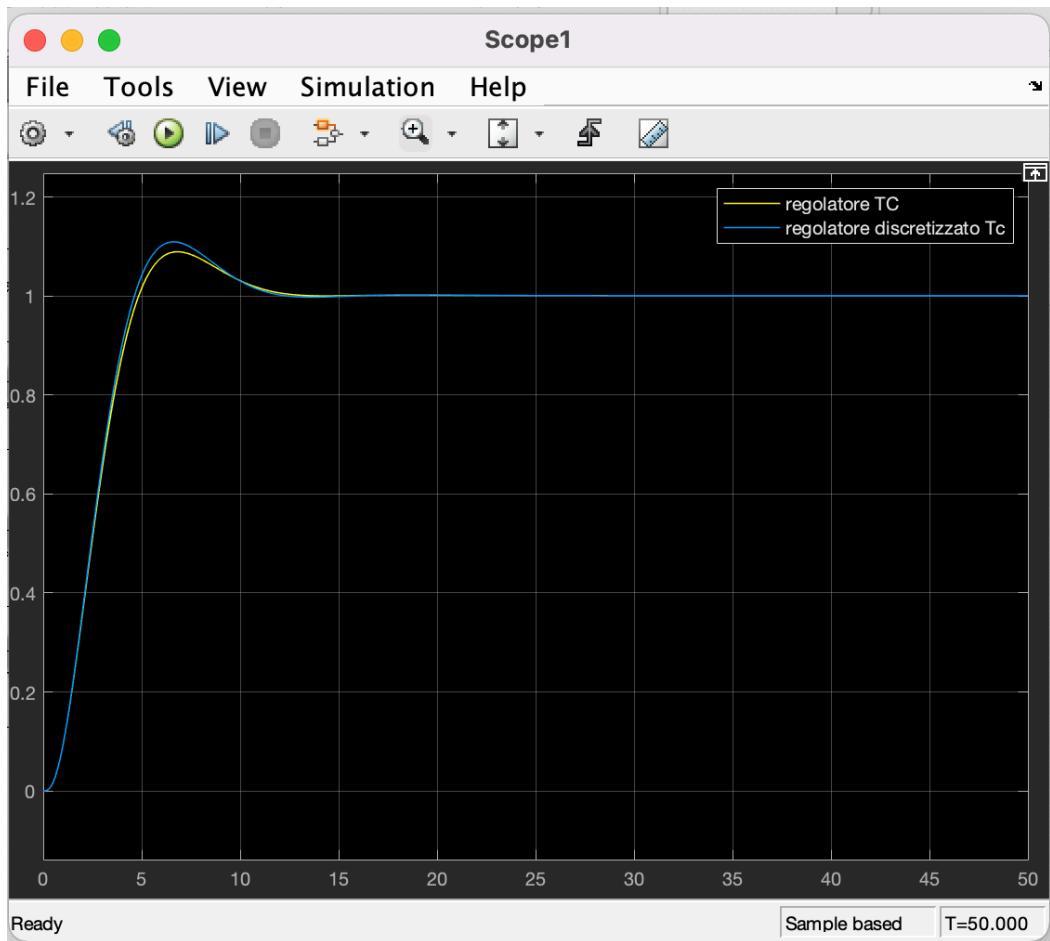
$$C_z = \frac{0.30733 (z-1) (z-0.9568)}{(z-1) (z-0.8101)}$$

Sample time: 0.2 seconds  
Discrete-time zero/pole/gain model.

>> Esercizio 1 >> modello\_discretizzato.slx



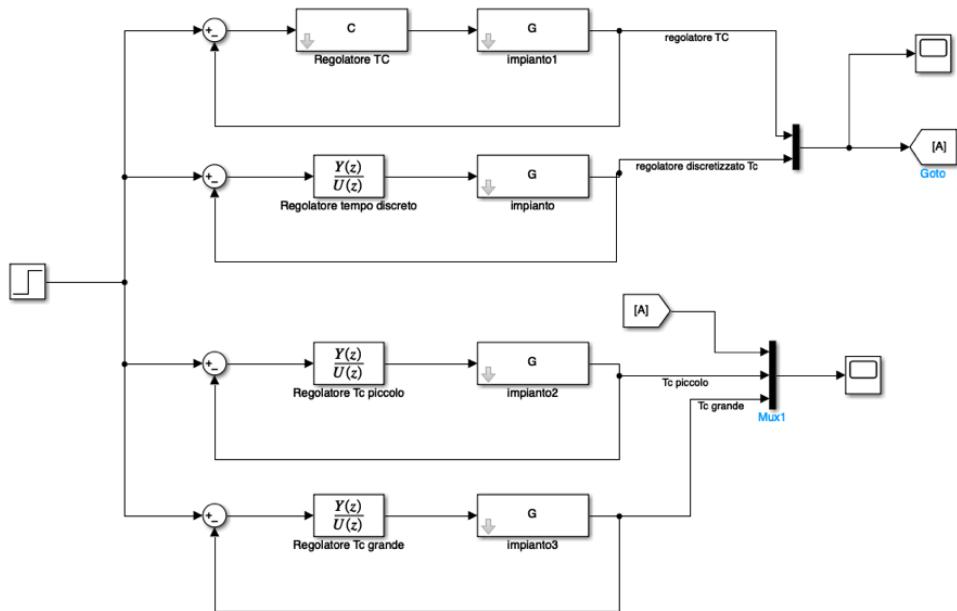
Confronto tra *regolatore standard a tempo continuo* e *regolatore discretizzato*.



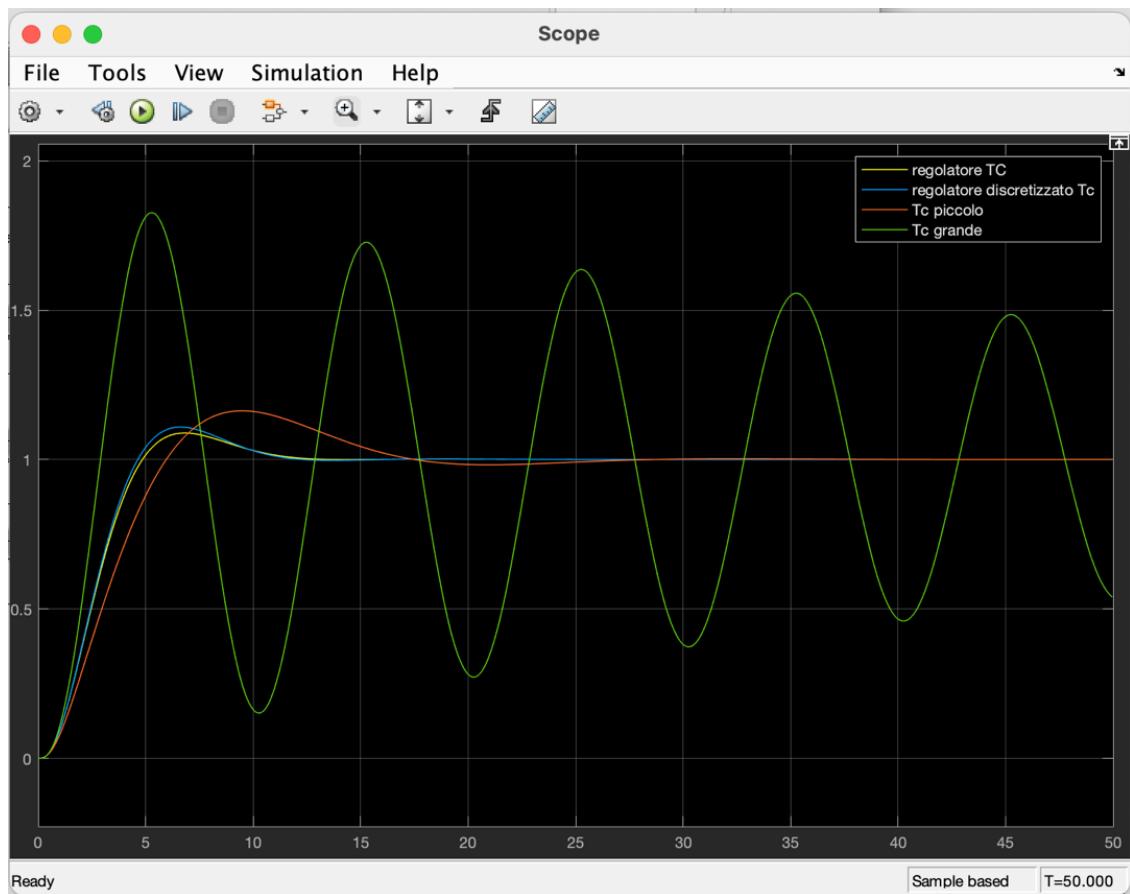
Si noti come la scelta del tempo di campionamento pari a  $T_c = 0.20$ , mediante il criterio del *tempo di salita*  $t_r$ , risulti essere ‘azzeccata’.

Successivamente si dimostrerà l’importanza del *tempo di campionamento* nella fase di discretizzazione.

Facendo un confronto per  $T_c = 0.01$  (relativamente piccolo) e per  $T_c = 0.99$  (relativamente grande)

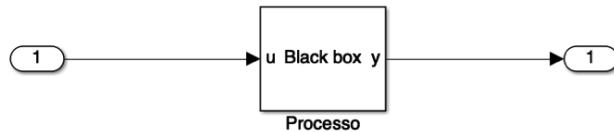


Si evidenzia come il tempo di campionamento determini le *performance* del regolatore discretizzato.



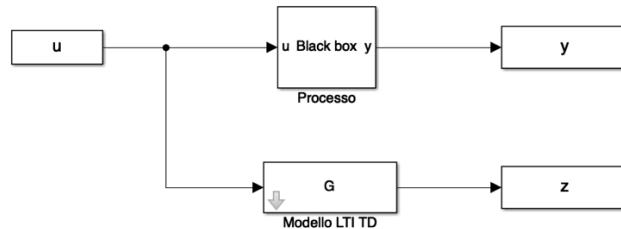
## ESERCIZIO 2

Dato il seguente processo 'black box'



- Approssimare la dinamica del processo con un modello ARMA di ordine 4 identificandone opportunamente i parametri.

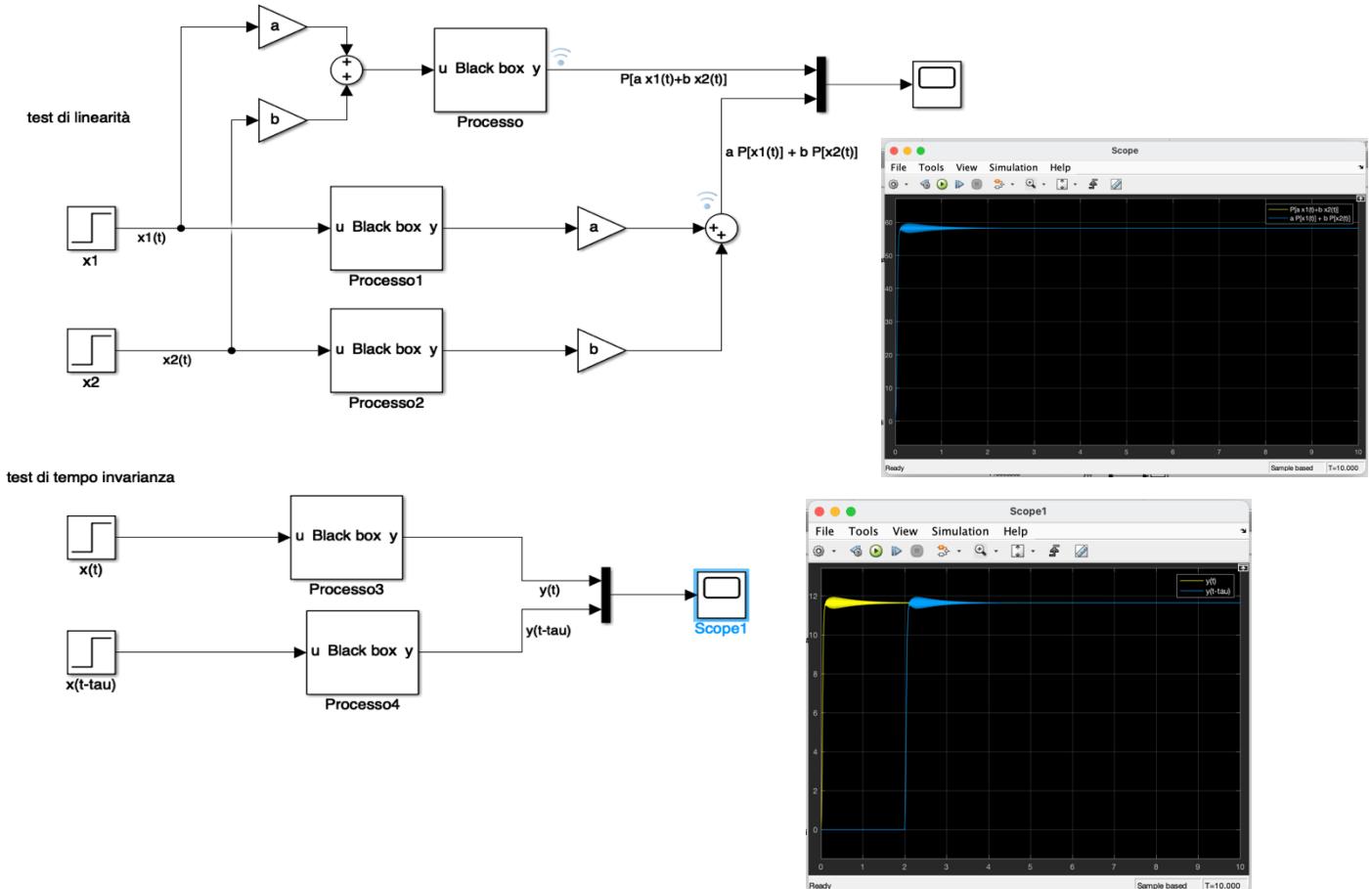
Per l'identificazione del processo si segue un approccio cosiddetto *data driven*, basato sulla costruzione di modelli mediante la raccolta di dati, senza lo studio della matematica/fisica del processo in esame.



Innanzitutto, si verifica se il processo soddisfa:

- Il principio di sovrapposizione degli effetti*
- La proprietà di tempo invarianza*

>> folder Esercizio 2 >> script.m e test\_LTI.slx



Dalle osservazioni precedenti, si deduce che il processo può essere descritto mediante un modello LTI.

## ○ FASE DI DEFINIZIONE E DI IDENTIFICAZIONE DEI PARAMETRI DEL MODELLO

L'idea è di approssimare la dinamica del processo mediante un modello LTI a tempo discreto.

In questo caso, il modello scelto per la descrizione del processo è il seguente:

$$z(k) + \alpha_1 z(k-1) + \alpha_2 z(k-2) + \cdots + \alpha_n z(k-n) = \beta_0 u(k) + \beta_1 u(k-1) + \cdots + \beta_m u(k-m)$$

In forma chiusa diventa:

$$z(k) = -\alpha_1 z(k-1) - \alpha_2 z(k-2) - \cdots - \alpha_n z(k-n) + \beta_0 u(k) + \beta_1 u(k-1) + \cdots + \beta_m u(k-m)$$

Con l'ipotesi di  $n \geq m$ , che l'ordine dell'equazione alle differenze è  $n = 4$ , l'obiettivo sarà di determinare i parametri:  $\alpha_1, \alpha_2, \dots, \alpha_n, \beta_0, \beta_1, \dots, \beta_m$ .

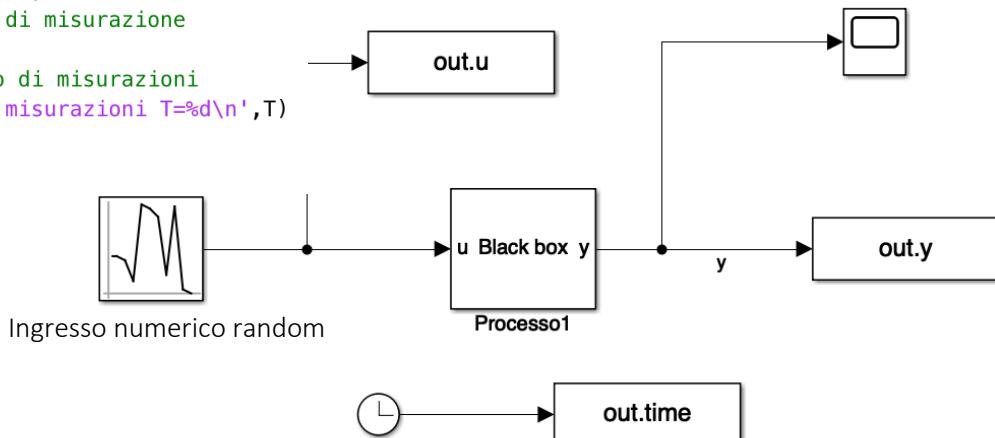
## ○ FASE DI MISURAZIONE

La fase di raccolta dei dati risulta essere cruciale, poiché è alla base della metodologia *data-driven* per l'identificazione dei modelli black box.

Fissato il numero di misurazioni  $T$  che si desiderano effettuare, allora si collezionano i campioni raccolti relativi all'ingresso  $u(k)$  e alla corrispondente uscita  $y(k)$  con  $k = 0, \dots, T$ .

>> Esercizio 2 >> misurazione1.slx

```
data=sim('misurazione');
t=data.time; % tempo di misurazione
T=length(t); % numero di misurazioni
fprintf('\nNumero di misurazioni T=%d\n',T)
u=data.u; % u(k)
y=data.y; % y(k)
```



## ○ FASE DI ELABORAZIONE DEI PARAMETRI

L'obiettivo è di approssimare il più possibile la risposta del processo con la risposta del modello LTI a tempo discreto,  $y(k) \approx z(k) \forall k = 0, \dots, T$ .

Dalla forma dell'equazione alle differenze finite con ritardo e dalla condizione  $n \geq m$ , riportate nella fase di definizione, si può dedurre come il primo istante di tempo (campione) utile per l'identificazione dei parametri corrisponde a  $k = n$ .

$$z(n) \approx y(n) = -\alpha_1 y(n-1) - \alpha_2 y(n-2) - \cdots - \alpha_n y(0) + \beta_0 u(n) + \beta_1 u(n-1) + \cdots + \beta_m u(n-m)$$

Per  $k = n, \dots, T$  si costruisce il seguente sistema di equazioni:

$$\Sigma: \begin{cases} y(n) = -\alpha_1 y(n-1) - \alpha_2 y(n-2) - \cdots - \alpha_n y(0) + \beta_0 u(n) + \beta_1 u(n-1) + \cdots + \beta_m u(n-m) \approx z(n) \\ y(n+1) = -\alpha_1 y(n) - \alpha_2 y(n-1) - \cdots - \alpha_n y(1) + \beta_0 u(n+1) + \beta_1 u(n) + \cdots + \beta_m u(n-m+1) \approx z(n+1) \\ \vdots \\ y(T) = -\alpha_1 y(T-1) - \alpha_2 y(T-2) - \cdots - \alpha_n y(T-n) + \beta_0 u(T) + \beta_1 u(T-1) + \cdots + \beta_m u(T-m) \approx z(T) \end{cases}$$

In forma compatta il sistema  $\Sigma$  lo si può esprimere come  $Y = \phi \theta$ , in cui:

1.  $Y$  è il vettore dei *termini noti*

$$Y = \begin{bmatrix} y(n) \\ y(n+1) \\ \vdots \\ y(T) \end{bmatrix} \in \mathbb{R}^{(T-n) \times 1}$$

2.  $\phi$  è la cosiddetta *matrice dei regressori*

$$\phi = \begin{bmatrix} -y(n-1) & \cdots & -y(0) & u(n) & \cdots & u(n-m) \\ -y(n) & \cdots & -y(1) & u(n+1) & \cdots & u(n-m+1) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -y(T-1) & \cdots & -y(T-2) & u(T) & \cdots & u(T-m) \end{bmatrix} \in \mathbb{R}^{(T-n) \times (n+m+1)}$$

3.  $\theta$  è il vettore delle *incognite* del sistema  $\Sigma$

$$\theta = [\alpha_1 \ \cdots \ \alpha_n \ \beta_0 \ \cdots \ \beta_m] \in \mathbb{R}^{1 \times (n+m+1)}$$

Si procede con la costruzione del vettore  $Y$ , della matrice  $\phi$  e al calcolo di  $\theta$  mediante uno script matlab:

*>> calcolo\_parametri.m*

```
function theta=calcolo_parametri(y,u,n,m,T)

Y=y(n+1:T); % termini noti

PHI=[]; % inizializzazione matrice dei regressori
% costruzione matrice dei regressori
for k=n:T-1
    phi=zeros(1,n+m+1); % k-esima row della PHI
    for j=1:n
        phi(j) = -y(k-j+1);
    end
    for j=1:m+1
        phi(n+j) = u(k-j+2);
    end
    PHI=[PHI;phi];
end
```

Si noti come la matrice dei regressori  $\phi \in \mathbb{R}^{(T-n) \times (n+m+1)}$  risulti essere una matrice rettangolare, allora  $\nexists \phi^{-1}: \theta = \phi^{-1} Y$ , per questo motivo ci si accontenterà di una *soluzione approssimata* per  $\Sigma$ :

$$\theta = (\phi^T \phi)^{-1} \phi^T Y = [\alpha_1 \ \cdots \ \alpha_n \ \beta_0 \ \cdots \ \beta_m]$$

```
theta=regress(Y,PHI);
%theta=inv(PHI.'*PHI)*PHI.'*Y;
```

Nel main si ha:

```
theta=calcolo_parametri(y,u,n,m,T);
fprintf('\nVettore delle incognite: [');
fprintf('%3.2f, ', theta(1:end-1));
fprintf('%3.2f]\n', theta(end));
```

Numero di misurazioni T=201
Con n=4 e m=1
Vettore delle incognite theta: [-1.22, -0.30, 1.26, -0.61, -0.03, 0.42]

## ○ FASE DI VALIDAZIONE

Una prima validazione dei parametri ottenuti la si effettua costruendo la f.d.t. del modello in *forma filtro*:

$$G_z(z^{-1}) = \frac{\beta_0 + \beta_1 z^{-1} + \cdots + \beta_m z^{-m}}{1 + \alpha_1 z^{-1} + \cdots + \alpha_n z^{-n}}$$

I cui coefficienti sono le componenti di  $\theta = [\alpha_1 \ \cdots \ \alpha_n \ \beta_0 \ \cdots \ \beta_m] = [alpha \ beta]$

```
alpha=theta(1:n);
beta=theta(n+1:end);

fprintf('\nFunzione di trasferimento del modello:\n')
Gz=tf(beta',[1 alpha'],Ts,'Variable','z^-1')
```

Funzione di trasferimento del modello:

$G_z =$

$$\frac{-0.02501 + 0.4194 z^{-1}}{1 - 1.225 z^{-1} - 0.2996 z^{-2} + 1.264 z^{-3} - 0.6076 z^{-4}}$$

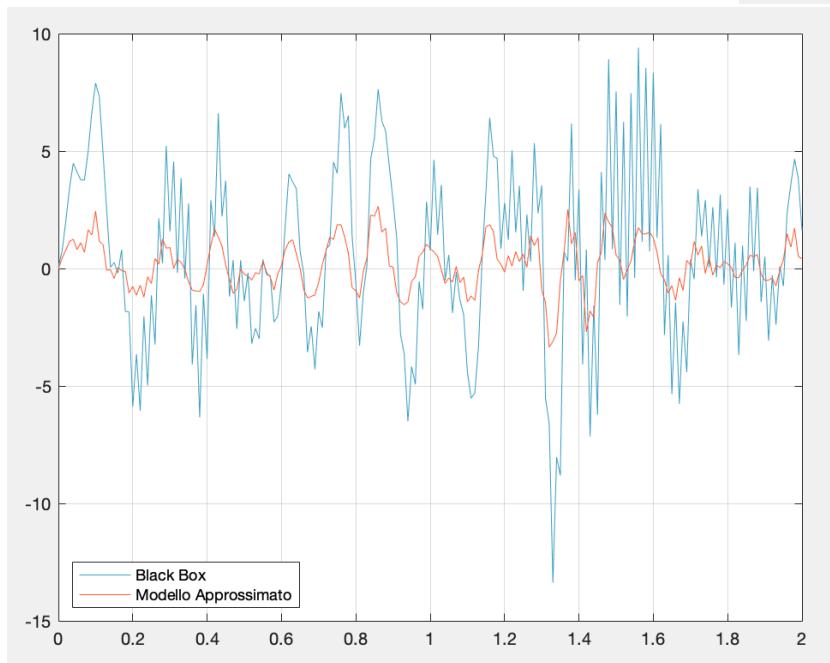
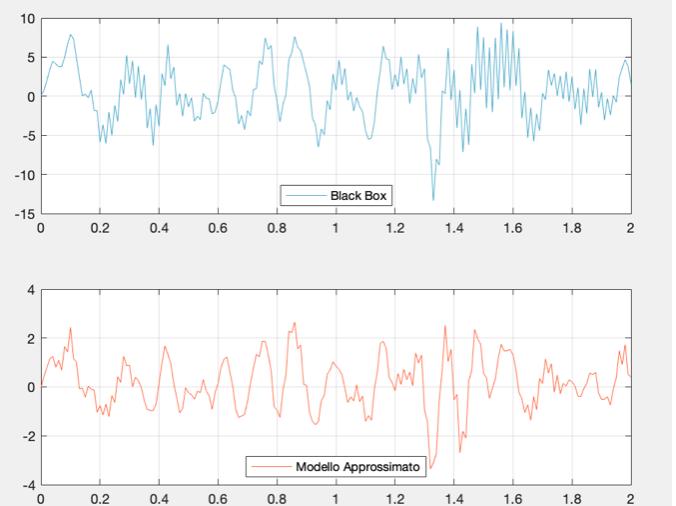
Sample time: 0.01 seconds

Discrete-time transfer function.

Un primo confronto tra *processo* e *modello approssimato* lo si può fare usando come valori di ingresso al modello approssimato la collezione di valori  $u(k)$  determinati nella fase di misurazione:

```
% Prima Validazione
figure(1)
subplot(2,1,1),plot(t,data.y,'Color','#3EA2C6'),grid
legend('Black Box','Location','south')
subplot(2,1,2),plot(t,lsim(Gz,data.u,t),'Color','#FF5733'),grid
legend('Modello Approssimato','Location','south')

figure(2)
plot(t,y,'Color','#3EA2C6')
hold on
plot(t,lsim(Gz,u,t),'Color','#FF5733')
grid
legend('Black Box','Modello Approssimato','Location','southwest')
hold off
```



È evidente che si può migliorare.

Fissato  $n$  si può aumentare  $m$  rispettando il vincolo di  $n \geq m$ .

Si effettua un confronto tra i vari modelli:

```
% Confronto
figure(3)
Color=["#3EA2C6","#57C784","#F061FA","#F26D6D","blue"];
pos=1;
for m=0:n
    theta=calcolo_parametri(y,u,n,m,T);

    alpha=theta(1:n);
    beta=theta(n+1:end);

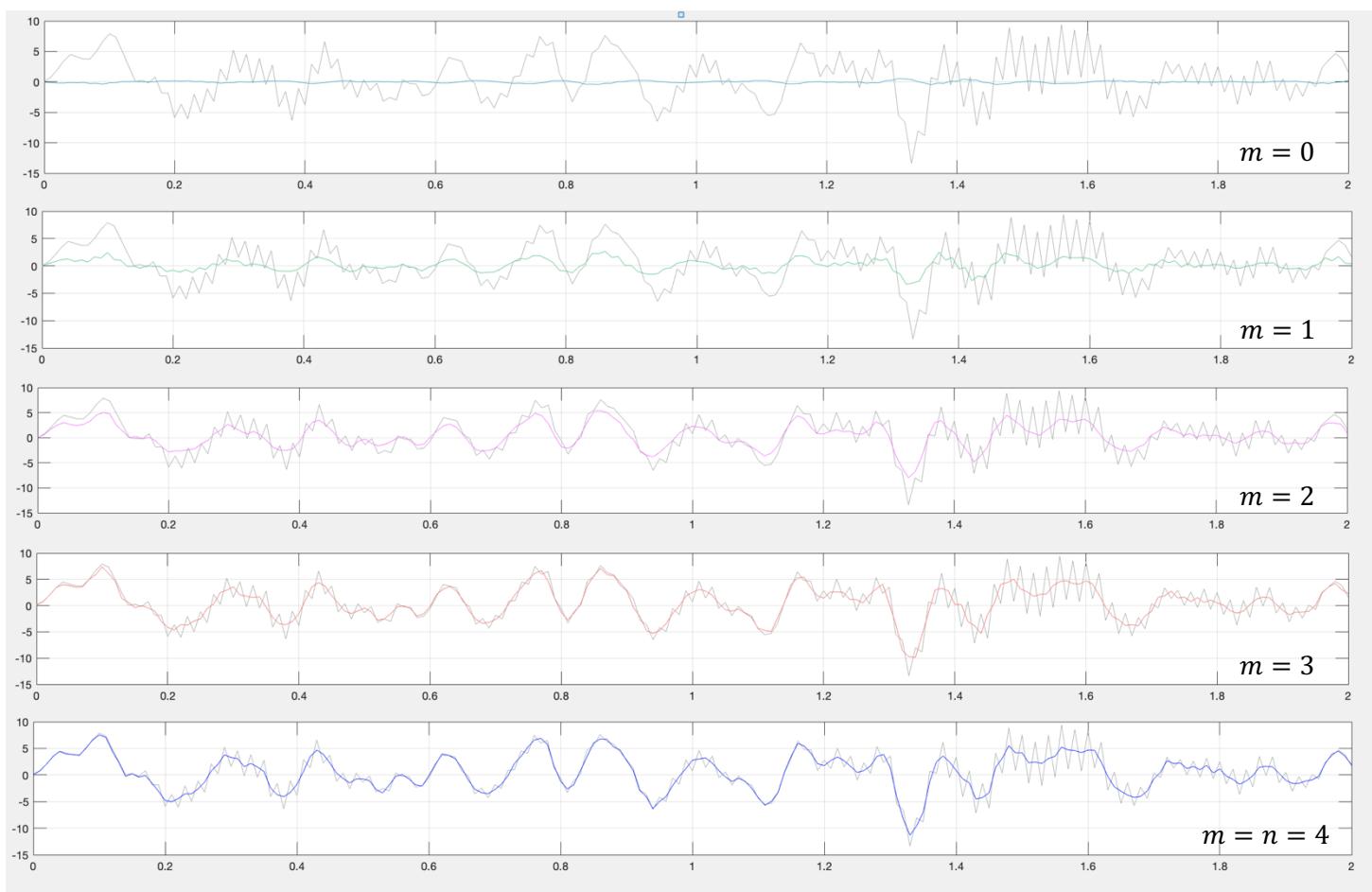
    Gz=tf(beta',[1 alpha'],Ts,'Variable','z^-1');

    subplot(5,1,pos)
    plot(t,y,'Color', '#A6ACA9'), hold on, grid
    plot(t,Lsim(Gz,u,t),'Color',Color(m+1)), hold off
    pos=pos+1;
    pause
end
```

Risposta del processo black box

Risposte dei modelli per n=4 e m variabile

Ottenendo:



Si può notare dai vari grafici e dal modello, che con  $m \rightarrow n$ :

$$z(k) = -\alpha_1 z(k-1) - \alpha_2 z(k-2) - \cdots - \alpha_n z(k-n) + \beta_0 u(k) + \beta_1 u(k-1) + \cdots + \beta_m u(k-m)$$

Aumentano i campioni di ingresso  $u(k)$ , e aumenta la dimensione del vettore delle incognite:

$$\theta = [\alpha_1 \ \cdots \ \alpha_n \ \beta_0 \ \cdots \ \beta_m]$$

Di conseguenze, per  $m \rightarrow n$  si ha:

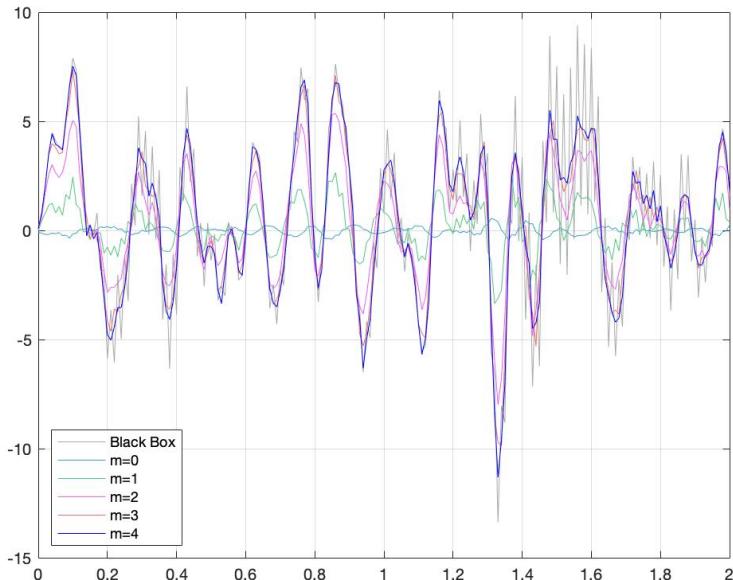
- un significativo diminuire dell'errore di approssimazione del processo black-box
- un aumentare della complessità spaziale, essendo la *matrice dei regressori*  $\phi$  nel worst-case di dimensioni

$$(T - n)x(n + m + 1) \Rightarrow (T - n)x(2n + 1)$$

e quindi un aumentare della complessità di elaborazione del vettore  $\theta = (\phi^T \phi)^{-1} \phi^T Y$ .

Di seguito, vi è un ulteriore confronto dei vari modelli ma sullo stesso grafico:

```
figure(4)
plot(t,y,'Color','#A6ACAA'), hold on, grid
pos=1;
for m=0:n
    theta=calcolo_parametri(y,u,n,m,T);
    alpha=theta(1:n);
    beta=theta(n+1:end);
    Gz=tf(beta',[1 alpha'],Ts,'Variable','z^-1');
    plot(t,lsim(Gz,u,t),'Color',Color(pos))
    pos=pos+1;
end
legend('Black Box','m=0','m=1','m=2','m=3','m=4','Location','southwest')
```



Nel seguito sarà analizzato e discusso il modello LTI con  $n = 4$  e  $m = 4$ .

### ○ FASE DI TESTING

```
%% Test-bench
fprintf('\nConsidero la f.d.t. con n=%d e m=%d\n',n,m);

fprintf('\nVettore dei parametri alpha: [');
fprintf('%3.2f, ', alpha(1:end-1));
fprintf('%3.2f]\n', alpha(end));

fprintf('\nVettore dei parametri beta: [');
fprintf('%3.2f, ', beta(1:end-1));
fprintf('%3.2f]\n', beta(end));

Gz=tf(beta',[1 alpha'],Ts,'Variable','z^-1')
```

Considero la f.d.t. con n=4 e m=4

Vettore dei parametri alpha: [-0.37, -0.76, 0.54, -0.06]

Vettore dei parametri beta: [0.07, 0.63, 1.09, 1.22, 0.87]

Gz =

$$0.06887 + 0.6283 z^{-1} + 1.085 z^{-2} + 1.223 z^{-3} + 0.8683 z^{-4}$$


---

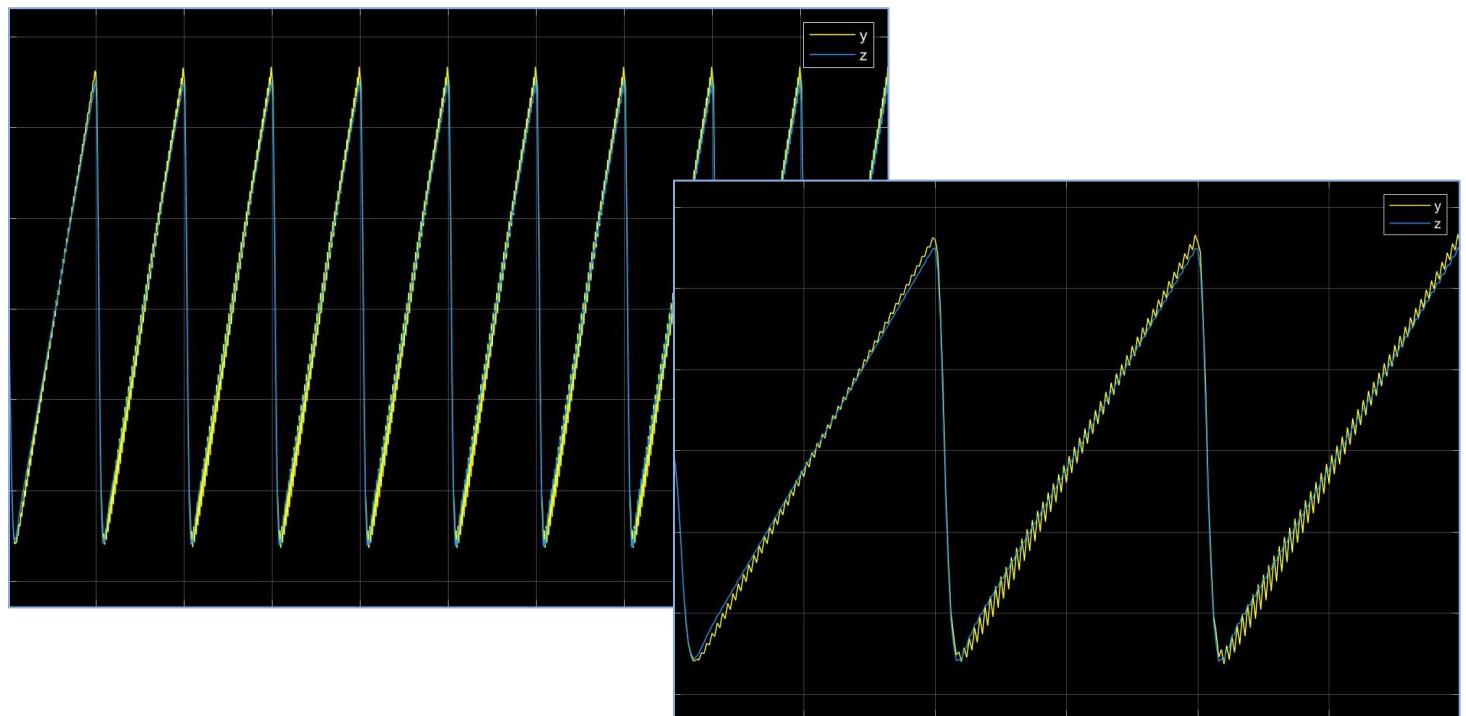
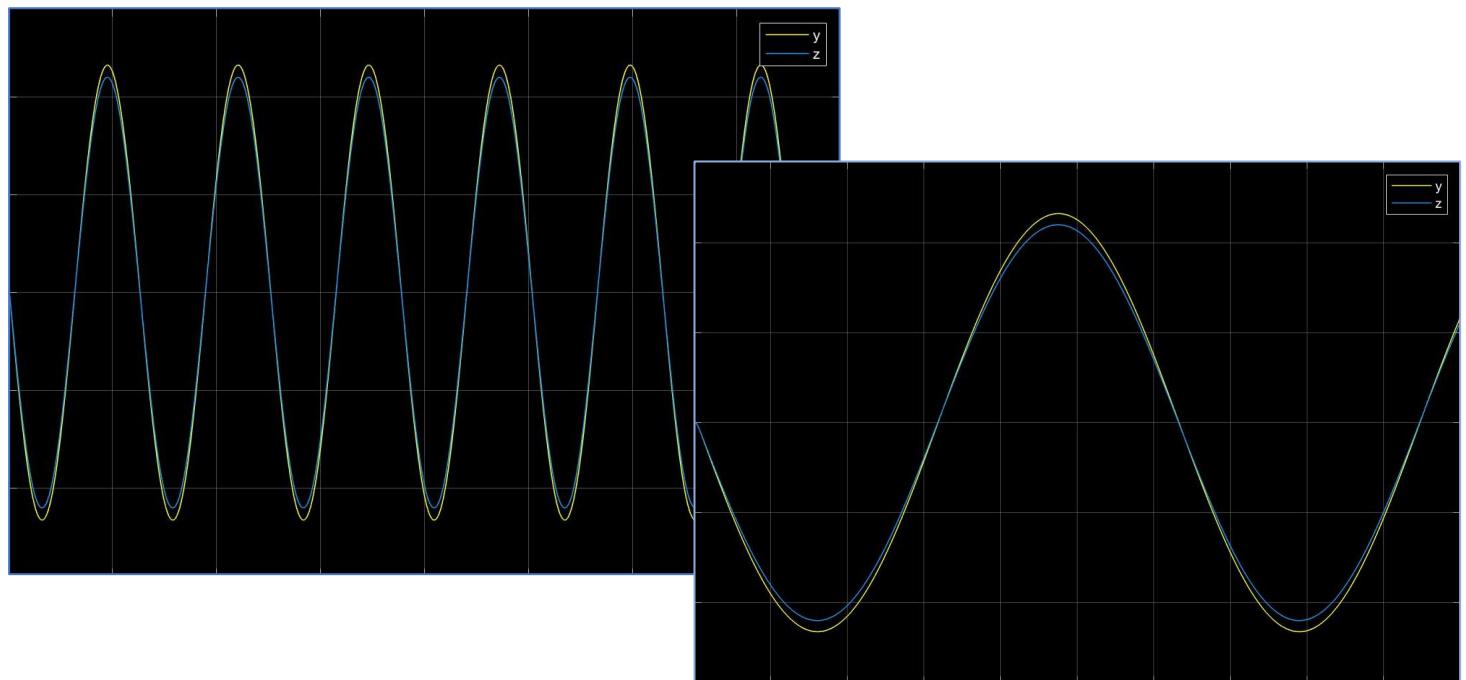
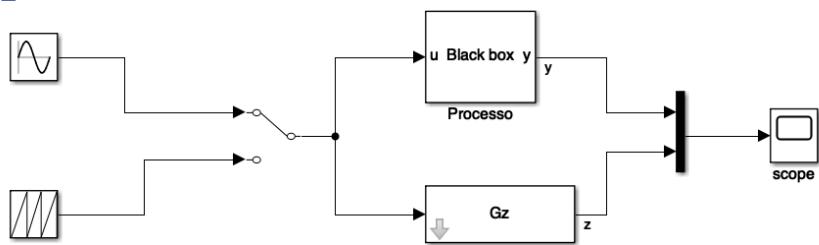

$$1 - 0.3703 z^{-1} - 0.7582 z^{-2} + 0.5429 z^{-3} - 0.06282 z^{-4}$$

Sample time: 0.01 seconds

Discrete-time transfer function.

Effettuando dei test con ingressi differenti, ci si rende conto di come il modello LTI approssimi quasi fedelmente il processo Black-Box.

>> Esercizio 2 >>test\_bench.slx



2. Determinare successivamente il grafico della risposta a gradino attraverso le tre forme canoniche.

Per  $n = 4$  e  $m = 4$

```
% Risposta al gradino delle 3 forme canoniche
[b,a]=tfdata(Gz,'v');

fprintf('\nCon n=%d e m=%d\n',n,m);

fprintf('\nVettore a: [');
fprintf('%3.2f, ', a(1:end-1));
fprintf('%3.2f]\n', a(end));

fprintf('\nVettore b: [');
fprintf('%3.2f, ', b(1:end-1));
fprintf('%3.2f]\n', b(end));

a=a(2:end);

R=1; % ampiezza gradino

data=sim('forme_canoniche');
t=data.time;
```

Gz =

$$\frac{0.06887 + 0.6283 z^{-1} + 1.085 z^{-2} + 1.223 z^{-3} + 0.8683 z^{-4}}{1 - 0.3703 z^{-1} - 0.7582 z^{-2} + 0.5429 z^{-3} - 0.06282 z^{-4}}$$

Sample time: 0.01 seconds  
Discrete-time transfer function.

Con n=4 e m=4

Vettore a: [1.00, -0.37, -0.76, 0.54, -0.06]

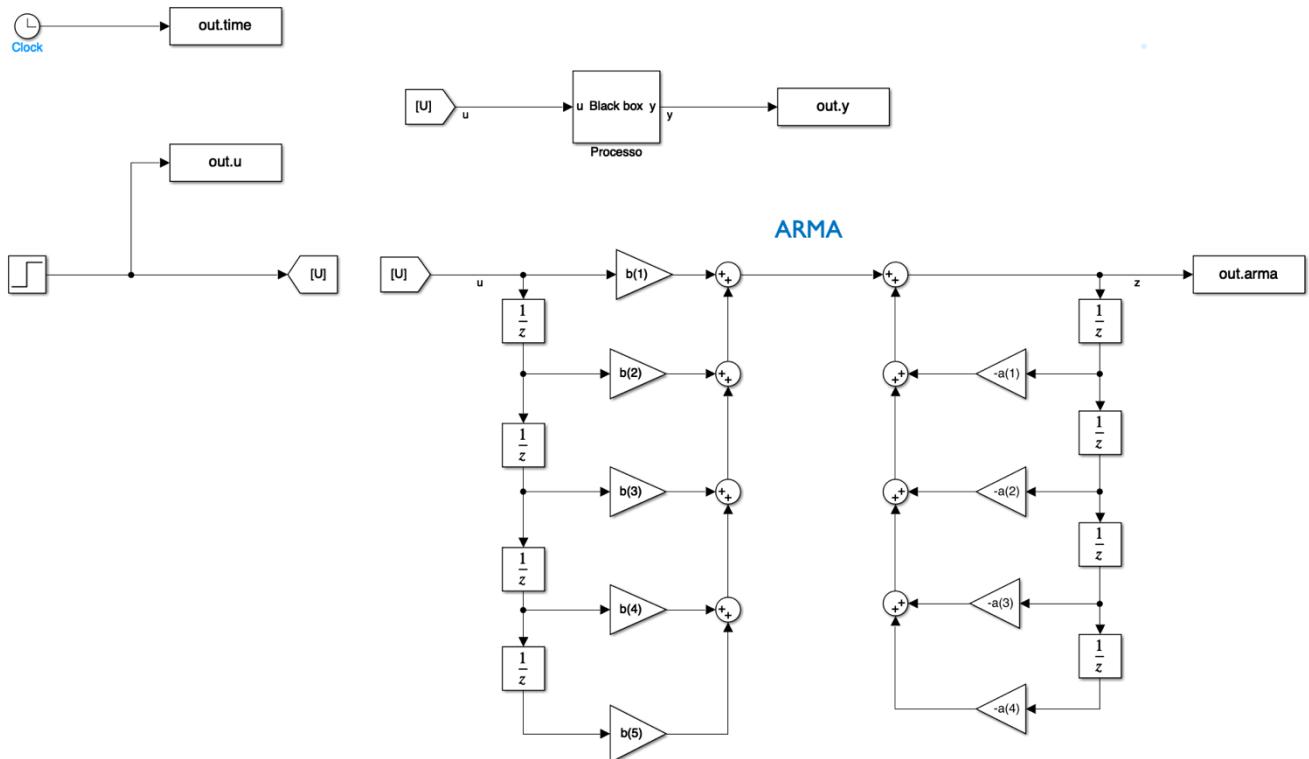
Vettore b: [0.07, 0.63, 1.09, 1.22, 0.87]

Rappresentazione grafica mediante schema a blocchi:

>> forme\_canoniche.slx

- Modello ARMA

$$z(k) = -\alpha_1 z(k-1) - \alpha_2 z(k-2) - \alpha_3 z(k-3) - \alpha_4 z(k-4) + \beta_0 u(k) + \beta_1 u(k-1) + \beta_2 u(k-2) + \beta_3 u(k-3) + \beta_4 u(k-4)$$



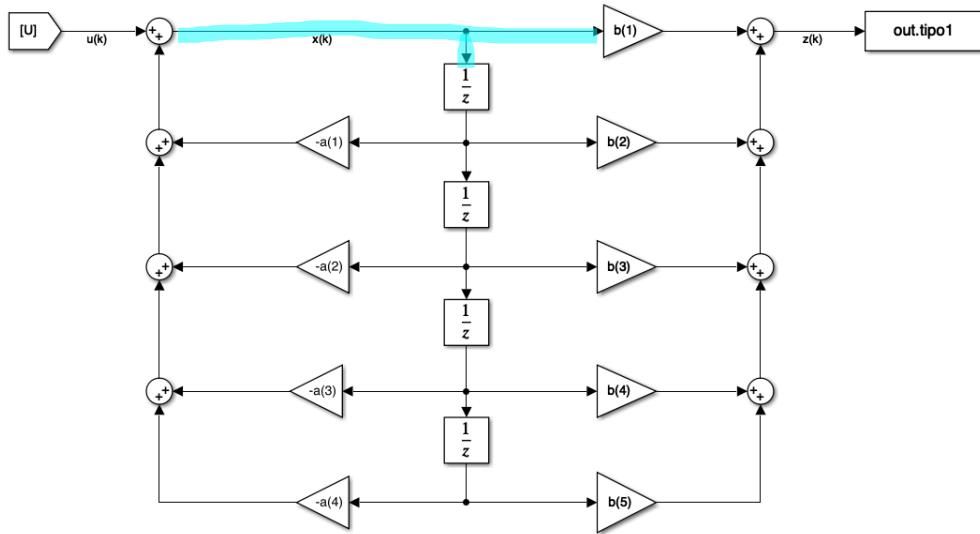
Numero di registri totali: con  $n = m \Rightarrow 2 * n$  registri

- Forma canonica di tipo 1

$$\begin{cases} z(k) = \beta_0 x(k) + \beta_1 x(k-1) + \beta_2 x(k-2) + \beta_3 x(k-3) + \beta_4 x(k-4) \\ x(k) = u(k) - \alpha_1 x(k-1) - \alpha_2 x(k-2) - \alpha_3 x(k-3) - \alpha_4 x(k-4) \end{cases}$$

In cui  $x$  è una cosiddetta *variabile di stato*

**TIPO UNO**



Numero di registri totali: con  $n = m \Rightarrow n$  registri

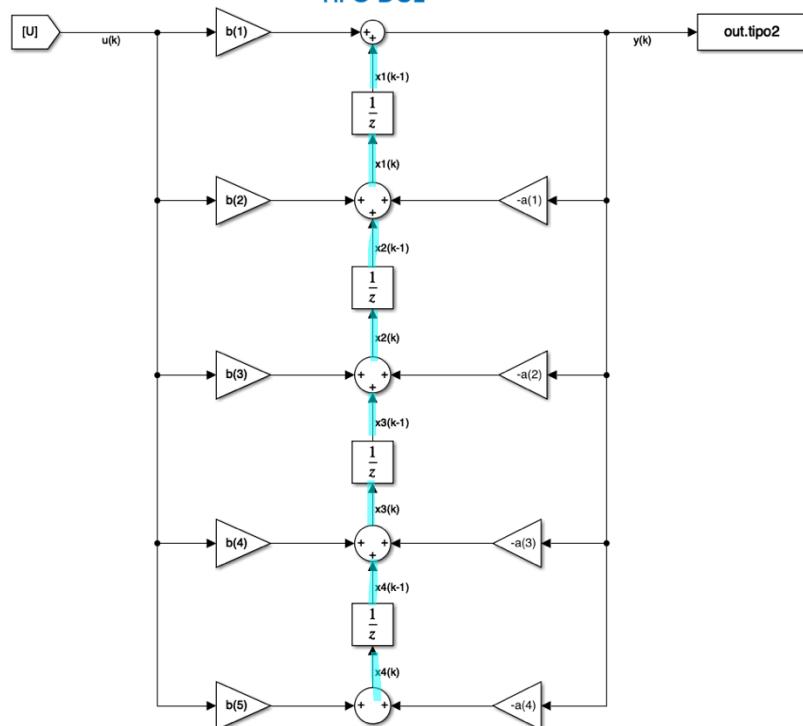
Rispetto allo schema a blocchi del modello ARMA, vi è un dimezzamento del numero di registri utilizzati.

- Forma canonica di tipo 2

$$\begin{cases} z(k) = \beta_0 u(k) + x_1(k-1) \\ x_1(k) = \beta_1 u(k) - \alpha_1 z(k) + x_2(k-1) \\ x_2(k) = \beta_2 u(k) - \alpha_2 z(k) + x_3(k-1) \\ x_3(k) = \beta_3 u(k) - \alpha_3 z(k) + x_4(k-1) \\ x_4(k) = \beta_4 u(k) - \alpha_4 z(k) \end{cases}$$

In cui  $x_1, x_2, x_3, x_4$  sono le *variabili di stato*

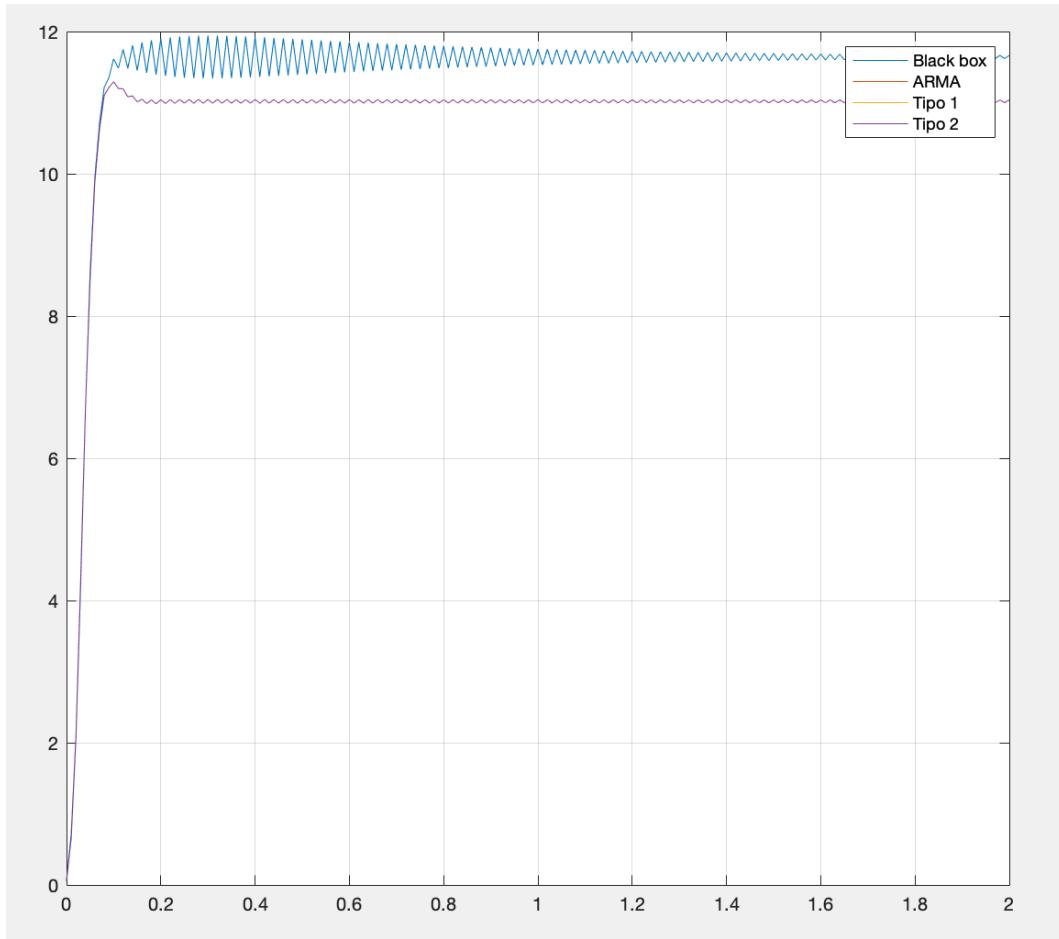
**TIPO DUE**



Numero di registri totali: con  $n = m \Rightarrow n$  registri.

Si determina il grafico della risposta al gradino attraverso le tre forme canoniche:

```
% step-response del processo  
y=data.y;  
  
% step-response del modello ARMA  
ARMA=data.arma;  
  
% step-response del modello TIPO 1  
TIP01=data.tipo1;  
  
% step response del modello TIPO 2  
TIP02=data.tipo2;  
  
figure(1)  
plot(t,y), hold on, grid  
pause  
plot(t,ARMA)  
pause  
plot(t,TIP01)  
pause  
plot(t,TIP02)  
legend('Black box', 'ARMA', 'Tipo 1', 'Tipo 2')
```



### ESERCIZIO 3

Sia dato il seguente segnale periodico

$$x(t) = \sin\left(2\pi 4 t + \frac{\pi}{6}\right) + \cos\left(2\pi 20 t - \frac{\pi}{3}\right)$$

Scrivere uno script matlab che, dopo opportuna scelta del passo di campionamento e dell'intervallo di osservazione, mostri le ampiezze e le fasi del segnale mediante l'algoritmo DFT.

La *trasformata discreta di Fourier* è un particolare tipo di *trasformata di Fourier* che converte una collezione finita di campioni equi spaziati nel tempo di una funzione in una collezione di coefficienti di una combinazione lineare di sinusoidi complesse, ordinate al crescere delle frequenze e/o pulsazioni.

La funzione in input alla *trasformata discreta di Fourier (DFT)* deve essere definita su dominio limitato e discreto, i cui campioni hanno durata limitata.

**DEFINIZIONE** – Data una successione di  $N$  campioni di una funzione  $x(t)$

$$x_1 = x(t_1), x_2 = x(t_2) \dots, x_N = x(t_N)$$

È trasformata nella successione di esattamente  $N$  numeri complessi  $\hat{x}_1, \hat{x}_2, \dots, \hat{x}_N$  dalla DFT secondo la seguente formula:

$$\hat{x}_k = \sum_{i=1}^N x_i e^{-j f_k t_i} = \sum_{i=1}^N x(t_i) e^{-j f_k t_i} \quad k = 1, \dots, N$$

In cui la quantità  $f_k = k \frac{f_s}{N}$  rappresenta il  $k$ -esimo campione frequenziale, mentre  $\frac{f_s}{N}$  prende il nome di *frequenza fondamentale*.

Per il calcolo della DFT su matlab:

*>> Esercizio 3 >> script.m*

```
close all
clear

Fs=100; % Intervallo di frequenze che si vogliono rappresentare

N=Fs/2; % numero di campioni
if mod(N,2)~=0
    N=N+1;
end
```

È importante che il numero di campioni sia pari; per  $N$  dispari c'è il rischio di discretizzare erroneamente il dominio delle frequenze.

È importante definire gli intervalli limitati e discretizzati per il tempo, le frequenze e le pulsazioni.

```
% dominio del tempo
tstep=1/Fs; % tempo di campionamento
t=0:tstep:(N-1)*tstep;

% dominio delle frequenze
fstep=Fs/N; % frequenza fondamentale
freq=-Fs/2:fstep:Fs/2-fstep;

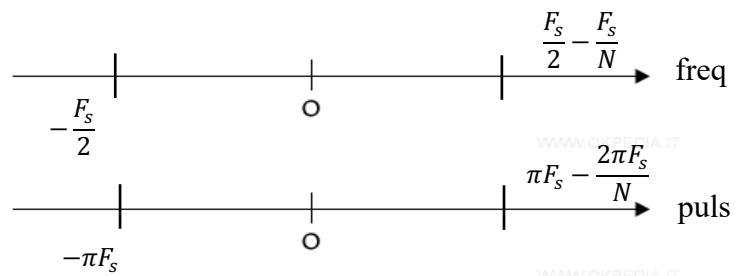
% dominio delle pulsazioni
puls=2*pi*freq;
```

$$\{x_1, x_2, \dots, x_N\} \rightarrow \hat{x}_k$$

si individuano le quantità:

- $t_i = i * tstep$  con  $i = 0, \dots, (N - 1) * tstep$
- $f_k = k * fstep$

In cui il dominio delle *frequenze f* e *pulsazioni ω* è rappresentato in questo modo:



Con la conoscenza del segnale da elaborare,  $x(t) = \sin\left(2\pi 4 t + \frac{\pi}{6}\right) + \cos\left(2\pi 20 t - \frac{\pi}{3}\right)$ , che è somma di due contributi sinusoidali, per la *proprietà di linearità* della DFT:

Dato il segnale:

$$x(t) = \sin\left(2\pi 4 t + \frac{\pi}{6}\right) + \cos\left(2\pi 20 t - \frac{\pi}{3}\right) = x_1(t) + x_2(t)$$

Rappresentando la trasformata discreta di Fourier con  $\mathcal{F}[\dots]$ , si ha che:

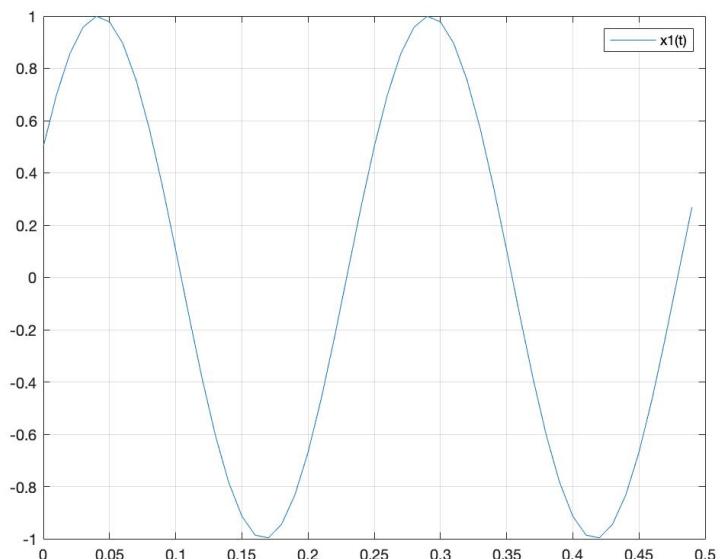
$$X = \mathcal{F}[x(t)] = \mathcal{F}[x_1(t) + x_2(t)] = \mathcal{F}[x_1(t)] + \mathcal{F}[x_2(t)] = X_1 + X_2$$

Di conseguenza, si procede con la definizione dei due segnali:

```
% parametri del segnale da processare
A=[1 1];
f=[4 20]; %Hz
phi=[pi/6 -pi/3];
```

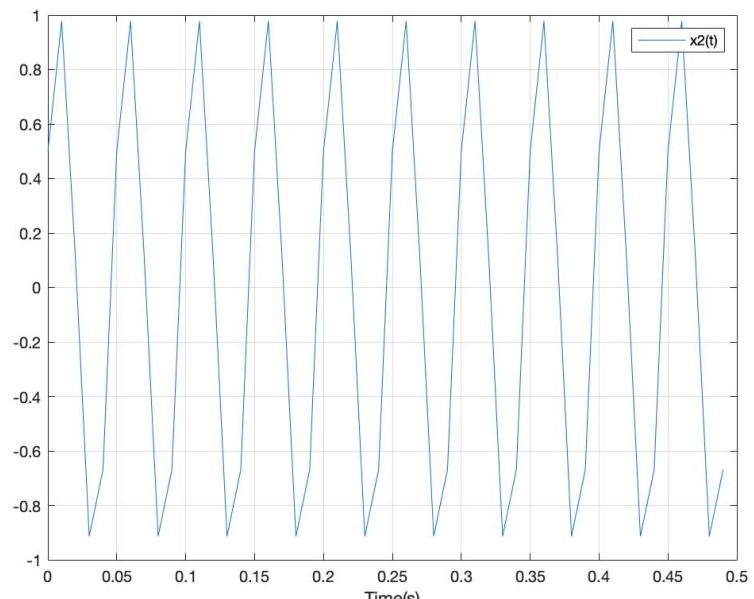
```
%% Linearità
%%% 1.
x1=A(1)*sin(2*pi*f(1)*t+phi(1));
```

```
figure(1)
plot(t,x1,'Color','#1776B8')
grid
xlabel('Time(s)')
legend('x1(t)')
```



```
%%% 2.
x2=A(2)*cos(2*pi*f(2)*t+phi(2));
```

```
figure(4)
plot(t,x2,'Color','#1776B8')
grid
xlabel('Time(s)')
legend('x2(t)')
```



Sviluppando:

$$\hat{x}_k = \sum_{i=1}^N x_i e^{-j f_k t_i}$$

Si costruisce il seguente sistema di equazioni in forma matriciale per  $k = 1, \dots, N$

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_N \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{-j f_2 t_2} & \dots & e^{-j f_N t_N} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-j f_N t_2} & \dots & e^{-j f_N t_N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

La matrice dei coefficienti prende il nome di *matrice della DFT*.

Per  $X = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_N\}$  esprimendo i numeri complessi  $\hat{x}_k$  in forma polare, si ottengono ampiezza  $A_k$  e la fase  $\phi_k$  delle sinusoidi dal modulo e argomento di  $\hat{x}_k$ :

$$A_k = |\hat{x}_k| = \sqrt{\operatorname{Re}(\hat{x}_k)^2 + \operatorname{Im}(\hat{x}_k)^2} \quad \text{e} \quad \phi_k = \arg(\hat{x}_k) = \operatorname{atan}\left(\frac{\operatorname{Im}(\hat{x}_k)}{\operatorname{Re}(\hat{x}_k)}\right)$$

Ottenendo  $|X| = \{|x_1|, |x_2|, \dots, |x_N|\}$  e  $\arg X = \{\arg x_1, \arg x_2, \dots, \arg x_N\}$ .

Dalla seguente function si ricavano lo spettro delle ampiezze e delle fasi delle due sinusoidi:

*>> Esercizio 3 >> modulo\_fase.m*

```
function X=modulo_fase(freq,puls,X,fig,N)

% rappresentazione spettro ampiezze e fase
% input : dominio delle frequenze, dominio delle pulsazioni,
%         fft del segnale x, vettore per le figure, il numero di campioni
color=["#EC6A24","#259AD5"];

X=fftshift(X);
X_mag=abs(X/(N/2));
X_ph=deg2rad(angle(X));

figure(fig(1))
subplot(2,1,1),stem(freq,X_mag,'Color',color(1)),xlabel('Frequency(Hz)'),title('Modulo','Color',color(1)),grid
subplot(2,1,2),stem(freq,X_ph,'Color',color(2)),xlabel('Frequency(Hz)'),title('Fase','Color',color(2)),grid

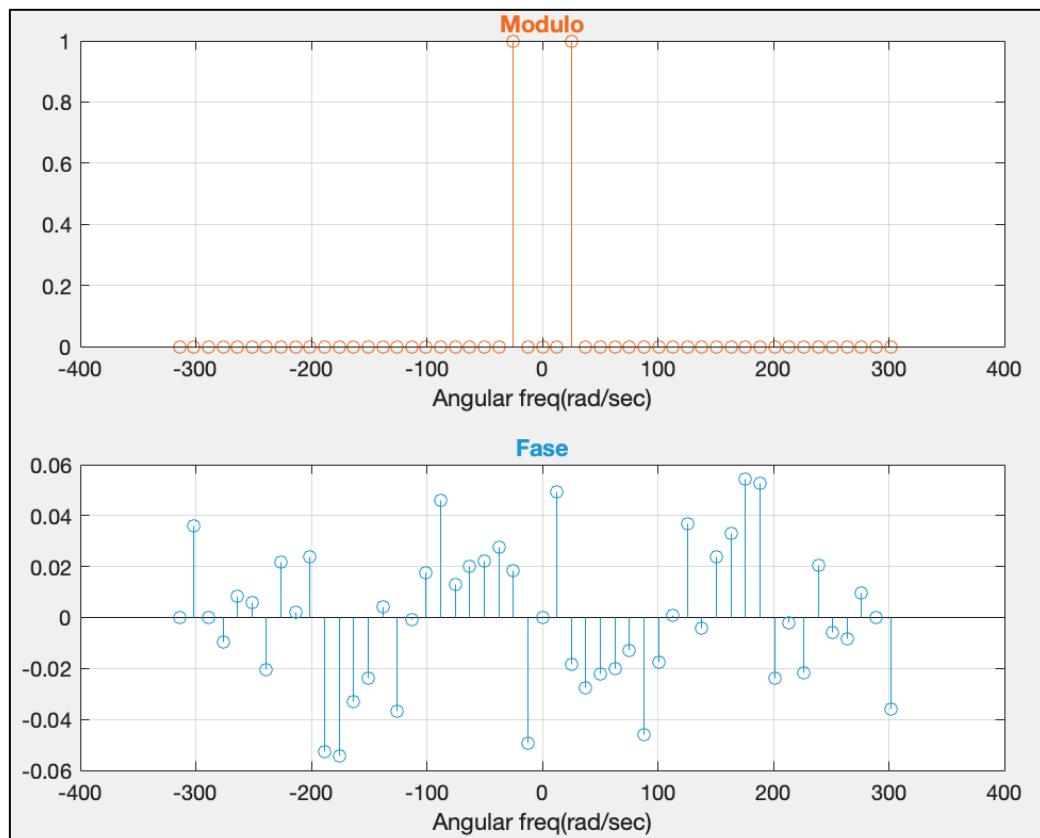
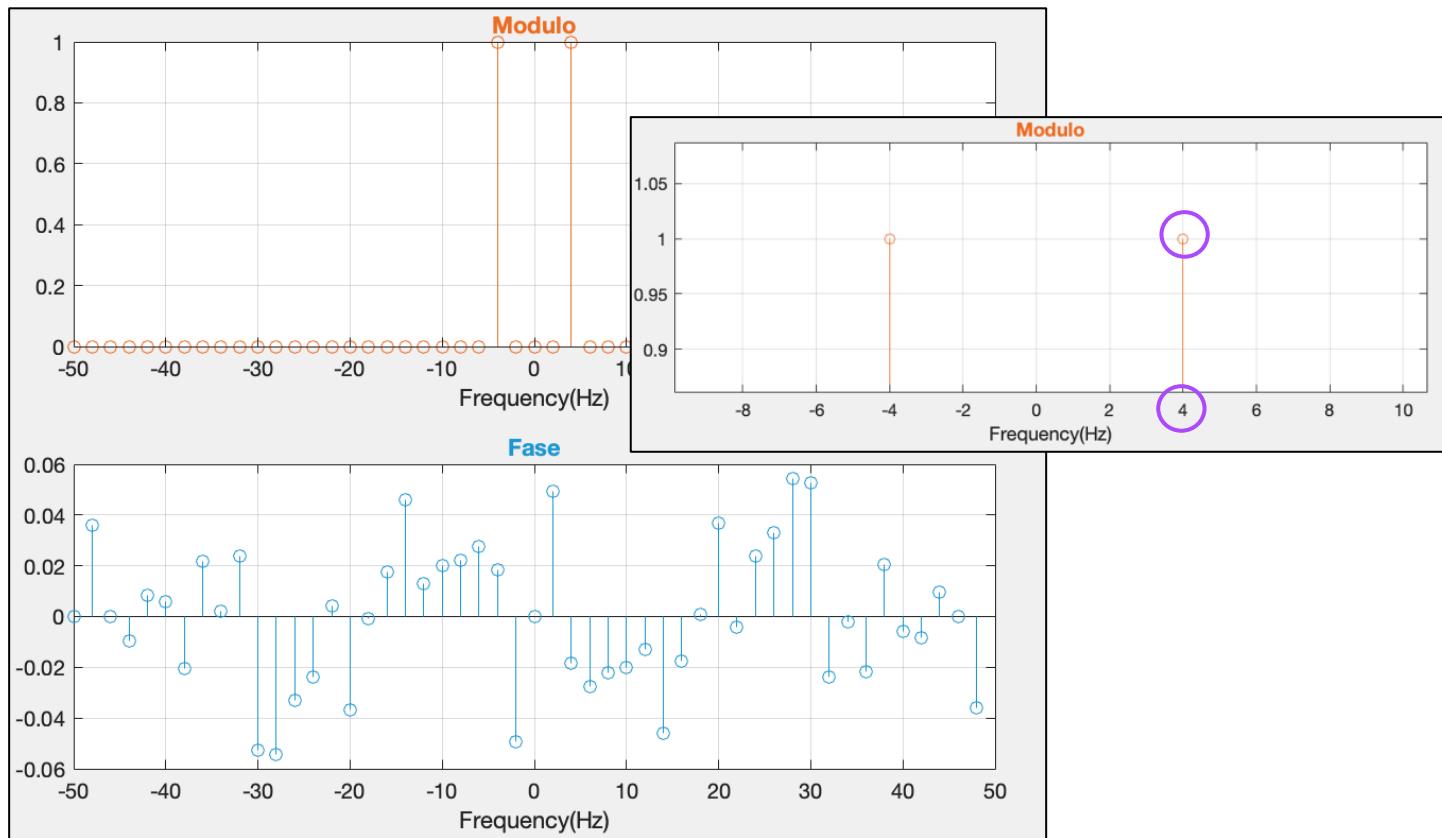
figure(fig(2))
subplot(2,1,1),stem(puls,X_mag,'Color',color(1)),xlabel('Angular freq(rad/sec)'),title('Modulo','Color',color(1)),grid
subplot(2,1,2),stem(puls,X_ph,'Color',color(2)),xlabel('Angular freq(rad/sec)'),title('Fase','Color',color(2)),grid

end
```

Di seguito lo spettro delle ampiezze e delle fasi del segnale sul dominio delle frequenze e pulsazioni:

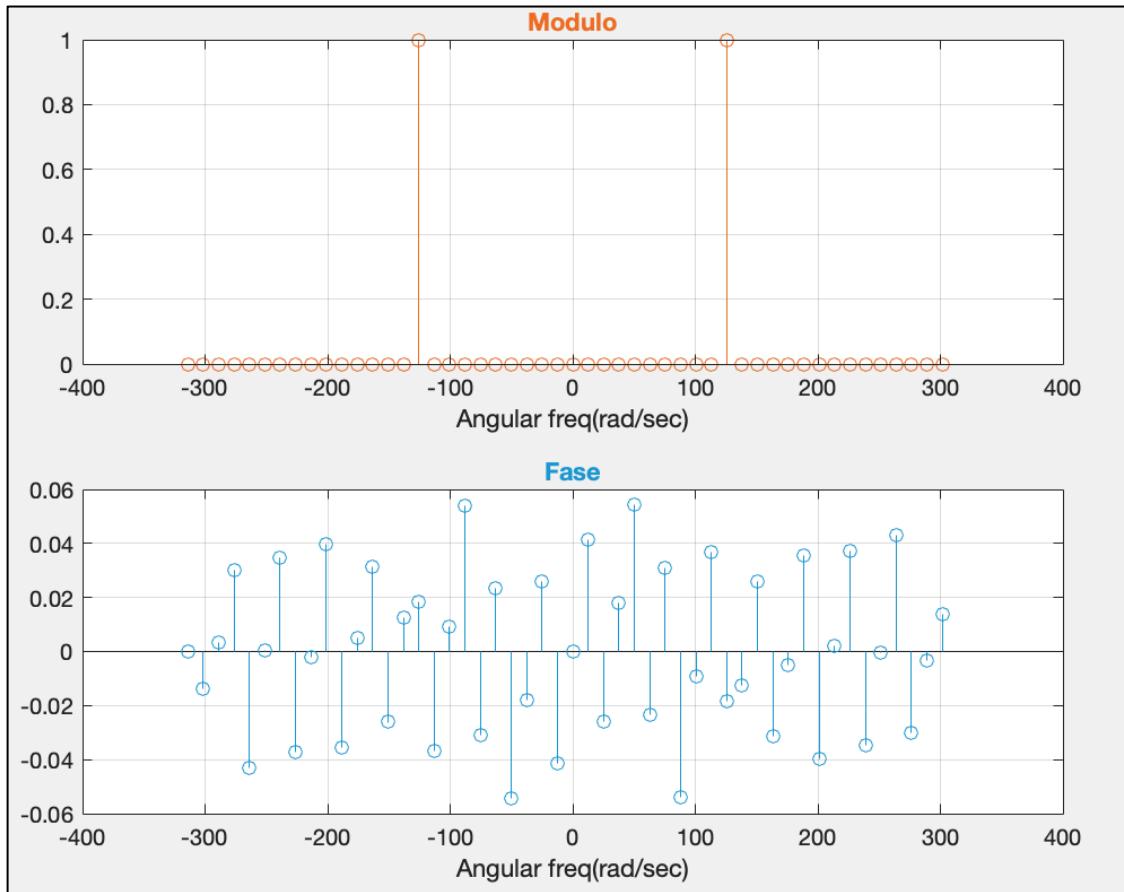
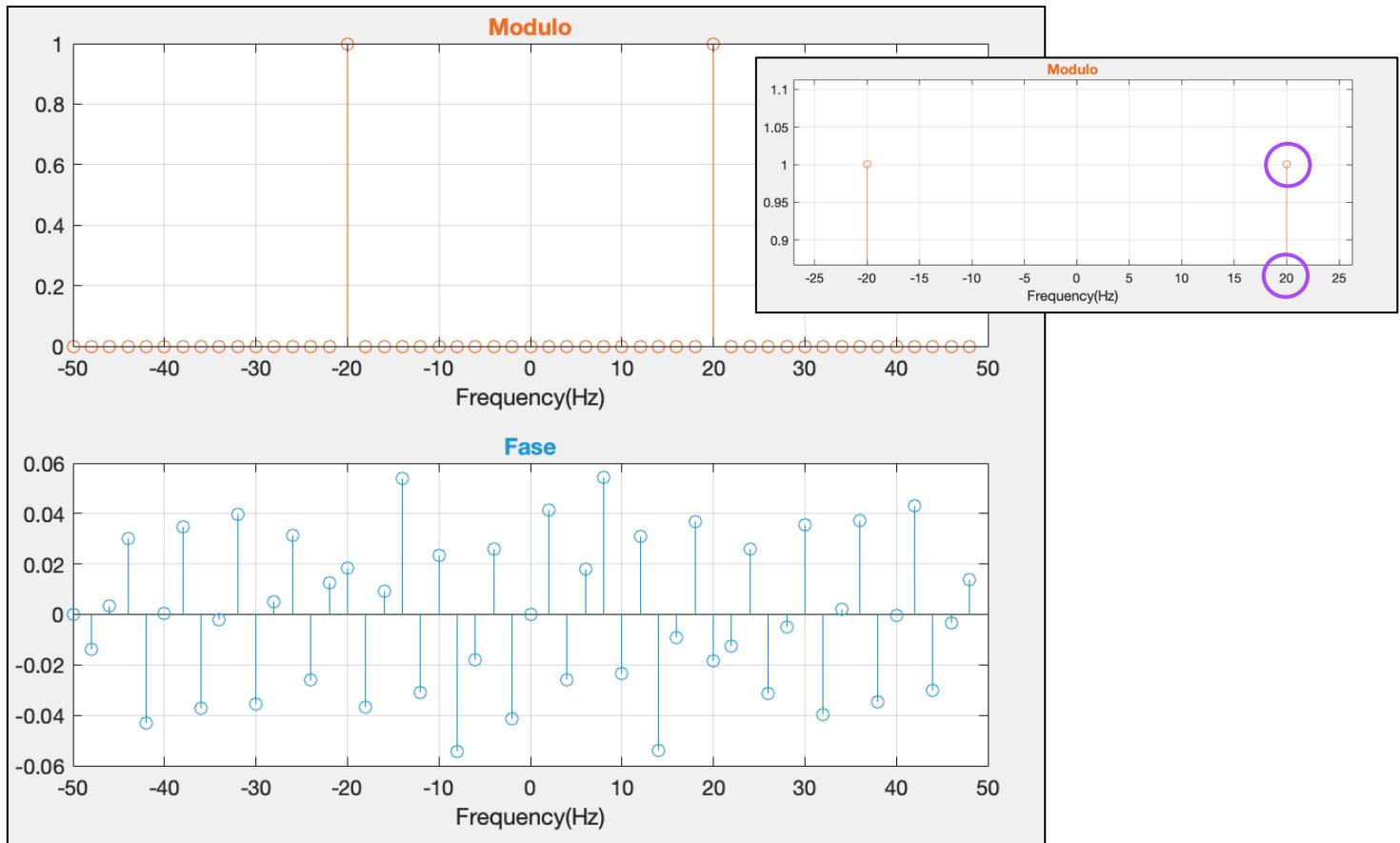
$$1. \quad x_1(t) = \sin\left(2\pi 4 t + \frac{\pi}{6}\right)$$

```
% dft x1(t)
fig=[2 3];
X1=fft(x1);
X1=modulo_fase(freq,puls,X1,fig,N);
```



$$2. \quad x_2(t) = \cos\left(2\pi 20 t - \frac{\pi}{3}\right)$$

```
% dft x2(t)
fig=[5 6];
X2=fft(x2);
X2=modulo_fase(freq,puls,X2,fig,N);
```



Applicando la proprietà di linearità della DFT, si ottiene lo spettro delle ampiezze e fasi del segnale completo

$$x(t) = \sin\left(2\pi 4 t + \frac{\pi}{6}\right) + \cos\left(2\pi 20 t - \frac{\pi}{3}\right) = x_1(t) + x_2(t)$$

%% 3. Segnale completo  
x=x1+x2;

```
figure(1)
fig=[2 3];
plot(t,x,'Color','#1776B8','LineWidth',1.0)
hold on
plot(t,x1,'Color','#64D9CE'),plot(t,x2,'Color','#54CA9A'),grid
xlabel('Time(s)')
legend('x(t)', 'x1(t)', 'x2(t)')

X1=fft(x1);
X2=fft(x2);

X=X1+X2;
X=modulo_fase(freq,puls,X,fig,N);
```

