Scuola Universitaria Professionale
della Svizzera Italiana

Dipartimento
di informatica
ed elettronica

MATLAB e SIMULINK nella regolazione automatica

Ing. Roberto Bucher

7 aprile 2003

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il campo della controllistica è rappresentato in MATLAB da tutta una serie di Toolbox sviluppati appositamente; questi toolbox mettono a disposizione una grande varietà di funzioni che permettono di ottimizzare il lavoro sia con sistemi lineari che con processi non lineari. Negli ultimi anni si stanno anche sviluppando funzioni relative alle nuove tecniche quali la logica fuzzy e i sistemi neurali. Alcuni toolbox permettono inoltre di generare automaticamente codice per dei processori DSP, oppure codice C per programmare dei controller. L'utilizzo parallelamente di SIMULINK non è indispensabile, ma facilita il lavoro, soprattutto nella costruzione grafica del processo da controllare. Noi ci occuperemo in particolare del "Control System Toolbox", oltre che di una serie di funzioni sviluppate presso la nostra scuola. Molte funzioni possono essere richiamate senza specificare i parametri in uscita, ottenendo direttamente dei grafici (p. es. bode), ed inoltre accettano diverse rappresentazioni del processo, dalla funzione di trasferimento a quella nel piano degli stati. È importante famigliarizzarsi con i diversi comandi e con le diverse sintassi mediante un uso frequente del comando "help". Solo in questo modo è possibile arrivare a sfruttare al massimo le potenzialità dei vari comandi a disposizione.

Capitolo 2

Modelli di sistemi

Esistono diversi metodi per descrivere un processo LTI (Linear Time Indipendent) all'interno di MATLAB. È possibile utilizzare una rappresentazione lineare nel piano degli stati, la funzione di trasferimento (Laplace), una rappresentazione mediante zeri-poli-guadagno, oppure si può modellare il processo mediante una S-function di SIMULINK e quindi linearizzarlo. Le funzioni per l'analisi e il design si applicano normalmente su sistemi lineari o linearizzati attorno ad un punto di lavoro. Alcuni toolbox possono agire direttamente anche sui sistemi non lineari.

2.1 Sistemi continui

2.1.1 Spazio degli stati

È forse oggi la rappresentazione più utilizzata, permettendo la descrizione di sistemi non-lineari con entrate ed uscite multiple. Nella forma lineare viene rappresentato mediante 4 matrici A, B, C, D che descrivono il processo mediante un sistema di equazioni differenziali di 1. ordine

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

dove x rappresenta il vettore delle variabili di stato, y il vettore delle uscite e u il vettore delle entrate. In MATLAB è sufficiente descrivere le 4 matrici A, B, C, D. Il sistema descritto dalle equazioni differenziali seguenti

$$\dot{x}_1 = x_2 \tag{2.1}$$

$$\dot{x}_2 = -x_1 - 3x_2 + u \tag{2.2}$$

$$y = x_1 \tag{2.3}$$

diventa in MATLAB

```
>> a=[0,1;-1,-3];
>> b=[0;1];
>> c=[1,0];
>> d=0;
>> plant=ss(a,b,c,d)
```

```
a =
                    x1          x2
          x1         0           1
          x2        -1          -3

b =
                    u1
          x1         0
          x2         1

c =
                    x1          x2
          y1         1           0

d =
                    u1
          y1         0

Continuous-time system.
```

2.1.2 Funzione di trasferimento

Mediante le trasformate di Laplace è possibile determinare la funzione di trasferimento che descrive il processo. In questo caso la relazione entrata-uscita è data da Y(s)=G(s)U(s).

In MATLAB la descrizione della funzione di trasferimento viene fatta mediante il polinomio in "s" al numeratore e quello al denominatore. È possibile in questo modo descrivere anche processi di tipo SIMO.

Esempio 2.1 *La funzione di trasferimento*

$$G(s) = \frac{(s+1)}{s^2+5s+2}$$

viene rappresentata tramite

```
>> g=tf([1 1],[1 5 2])

Transfer function:
    s + 1
-------------
s^2 + 5 s + 2
```

2.1.3 Zeri-Poli-Guadagno

Un ulteriore metodo di rappresentazione di un processo è caratterizzato da un vettore contenente le radici del numeratore (zeri), uno con le radici del denominatore (poli) ed il guadagno. Per la funzione descritta da

$$G(s) = 5\frac{s+1}{(s+3)(s+4)}$$

Si ottiene la rappresentazione

```
>> g=zpk([-1],[-3 -4],5)

Zero/pole/gain:
  5 (s+1)
-----------
(s+3) (s+4)
```

2.1.4 Linmod

Con il comando "linmod" si può determinare il modello linearizzato di un processo dalla sua rappresentazione in SIMULINK (vedi documentazione "Introduzione a SIMULINK"). Questo comando fornisce la rappresentazione nello spazio degli stati.

2.2 Sistemi discreti

Le rappresentazioni dei processi continui in "s" mediante la rappresentazione nello spazio degli stati, la funzione di trasferimento e la forma zero-pole-gain sono valide anche per sistemi discreti e vengono utilizzate nello stesso modo. L'unica differenza consiste nello specificare alla chiamata di una delle funzioni tf, ss o zpk un parametro supplementare che rappresenta il tempo di sampling.

2.2.1 Spazio degli stati

La rappresentazione di sistemi discreti nello spazio degli stati mediante le 4 matrici A, B, C, D descrive il sistema seguente

$$x[n+1] = Ax[n] + Bu[n] \tag{2.4}$$
$$y[n] = Cx[n] + Du[n] \tag{2.5}$$

2.2.2 Funzione di trasferimento

Mediante le trasformate "Z" è possibile determinare la funzione di trasferimento che descrive il processo. In questo caso la relazione entrata-uscita è data da Y(z)=G(z)U(z)

```
>> num=[1,-1];

>> den=[1,-2.3,0.8];

>> gz=tf(num,den,0.1)

Transfer function:
      z - 1
-----------------
z^2 - 2.3 z + 0.8

Sampling time: 0.1
```


2.2.3 Zeri-Poli-Guadagno

Mediante la specificazione di un vettore con gli zeri, un vettore con i poli, e il guadagno del sistema si ottiene la rappresentazione seguente

$$G(z) = K\frac{(z-z_1)(z-z_2)\cdots}{(z-p_1)(z-p_2)\cdots}$$

2.3 Trasformazioni

È possibile passare da una rappresentazione all'altra mediante dei comandi di trasformazione. Per i dettagli sulle varie funzioni è sempre possibile consultare l'help integrato.

2.3.1 Conversione di modelli

Le funzioni di conversione sono rappresentate nella tabella 2.1.

ss	→ spazio degli stati
tf	→ funzione di trasferimento
zpk	→ zeri-poli-guadagno

Tabella 2.1: Funzioni di conversione

2.3.2 Discretizzazione

Il toolbox di controllo mette già a disposizione i metodi per passare da un sistema continuo ad uno discreto e viceversa. Inoltre è possibile anche specificare il metodo di discretizzazione, scegliendo tra il metodo "zoh" (zero order hold), il metodo "foh" (first order hold), il metodo "matched" (zeros-poles matching), o il metodo "tustin" (trasformata di Tustin detto anche trasformata bilineare o trasformata w'). Normalmente viene utilizzato il metodo di default "zero-order-hold". La tabella 2.2 mostra le funzioni di conversione tra continuo e discreto

c2d	conversione continuo → discreto
d2c	conversione discreto → continuo

Tabella 2.2: Trasformazioni continuo-discreto

Capitolo 3

Metodi di analisi

Il "Control System Toolbox" mette a disposizione una serie di comandi per analizzare nel tempo e in frequenza un processo lineare di tipo LTI (Linear Time Indipendent), sia SISO (Single Input Single Output) che MIMO (Multiple Input Multiple Output). Normalmente si pu fare un'analisi sul sistema continuo o sul sistema discreto. Inoltre non è importante la forma in cui è stato memorizzato il modello.

3.1 Risposta nel tempo

Questi metodi permettono di determinare la risposta al transiente di un processo, determinando alcuni parametri come i tempi di risposta (rise time, setting time) sia altri parametri come l'overshooting e l'errore allo stato finito. È possibile determinare la risposta del processo ad un qualsiasi segnale di input, come pure simularlo con entrate casuali.

Esempio 3.1 *Consideriamoo la funzione di trasferimento*

$$G(s) = \frac{s+2}{s^3+2s^2+3s+4}$$

```
>> gs=tf([1,2],[1,2,3,4])

Transfer function:
        s + 2
---------------------
s^3 + 2 s^2 + 3 s + 4

>> gz=c2d(gs,0.5)

Transfer function:
0.1178 z^2 + 0.0736 z - 0.04296
-------------------------------
z^3 - 1.749 z^2 + 1.414 z - 0.3679

Sampling time: 0.5
```

Le figure 3.1, 3.2, 3.3 e 3.4 mostrano il risultato di alcuni comandi su questi processi.

```
>> step(gs)
```

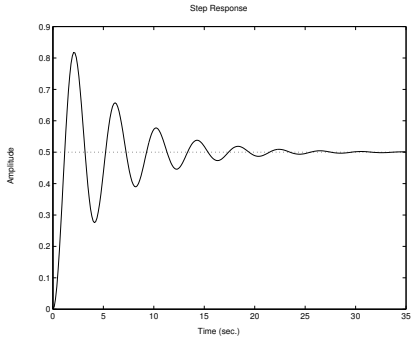


Figura 3.1: Comando *step*

```
>> step(gz)
```

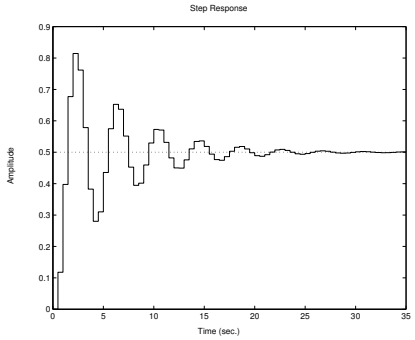


Figura 3.2: Comando *step (discreto)*

```
>> gs_ss=ss(gs);
>> initial(gs,[1,1,1])
```

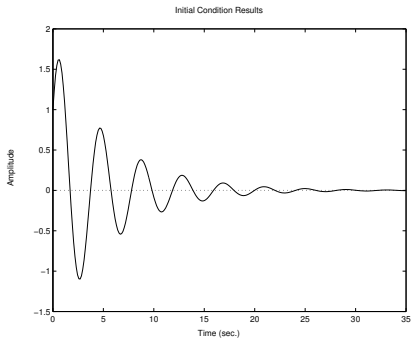


Figura 3.3: Comando *initial*

Nella tabella 3.1 sono riportate le funzioni per l'analisi della risposta nel tempo.

3.2 Analisi in frequenza

La risposta in frequenza di un sistema può essere ottenuta mediante i comandi riportati nella tabella 3.2.

Questi comandi disegnano automaticamente il grafico del risultato se chiamati senza parametri in uscita. Tutti questi comandi hanno diverse possibili sintassi. Il processo

```
>> t=0:pi/20:6*pi;
>> u=sin(t);
>> lsim(gs,u,t)
```

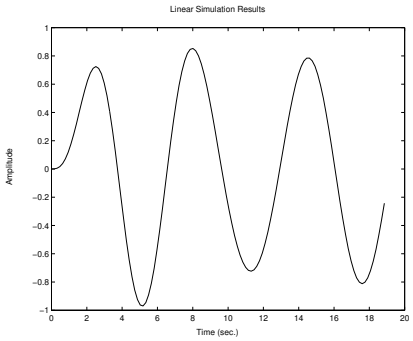


Figura 3.4: Comando *lsim*

covar	risposta con entrata rumore bianco
impulse	risposta ad un impulso di Dirac
initial	risposta naturale a varie condizioni iniziali
step	risposta ad un gradino unitario
lsim	risposta ad un input generico
filter	filtro "z" (vedi help)

Tabella 3.1: Funzioni su processi

bode	diagramma di bode di un sistema
nyquist	diagramma di nyquist
nichols	diagramma di nichols
sigma	diagramma dei valori singolari (sistemi MIMO)

Tabella 3.2: Funzioni per l'analisi in frequenza

può essere passato in qualsiasi rappresentazione(ss, tf o zpk). Il comando *"bode"* può, ad esempio, essere utilizzato nei modi seguenti.

```
>> bode(gs)
>> [mag,phase,w]=bode(gs);
>> [mag,phase]=bode(gs,w);
```

Nel primo caso viene mostrato il grafico del diagramma di bode della funzione descritta mediante "gs", nel secondo il risultato viene messo in tre matrici, senza plot, nel terzo l'utente specifica il vettore delle frequenze interessanti. In quest'ultimo caso, tralasciando i parametri in uscita, si ottiene ugualmente un grafico. Si potrebbe realizzare il plot del grafico di bode anche nel modo seguente:

```
>> [mag,phase,w]=bode(num,den);
>> subplot(211),semilogx(w,20*log10(mag)),grid
>> subplot(212),semilogx(w,phase),grid
```

Nelle figure 3.5,3.6 e 3.7 sono riportati i risultati di questi comandi, su sistemi SISO.

```
>> gs=tf([1],[1,2,1]);
>> bode(gs);
```

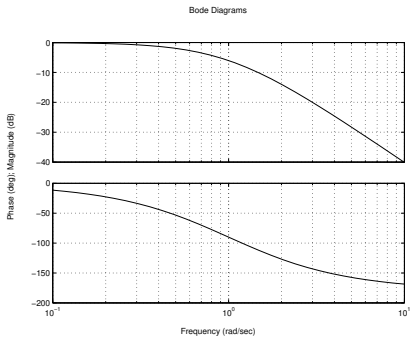


Figura 3.5: Comando *bode*

```
>> gs=tf([1],[1,2,1]);
>> nyquist(gs);
```

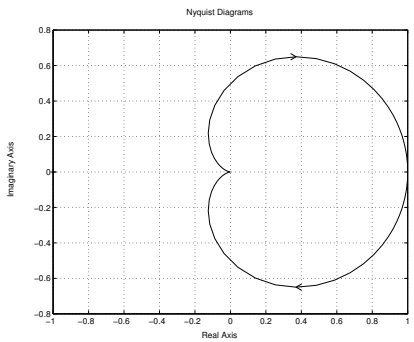


Figura 3.6: Comando *nyquist*

Il comando "margin" permette di determinare il margine di fase, il margine di guadagno e le due frequenze di gain-crossover e phase-crossover. Anche in questo caso esistono le due diverse sintassi:

```
>> gs=tf([1],[1,2,3,1,0]);
>> ngrid('new')
>> nichols(gs);
```

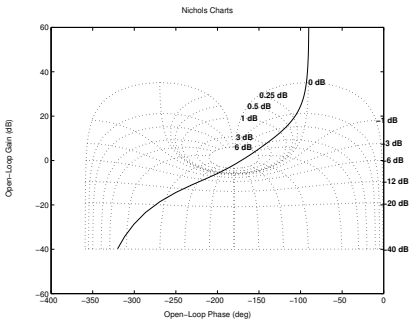


Figura 3.7: Comando *nichols*

```
>> gs=tf([1],[1,2,1,0]);
>> [gm,pm,wpc,wgc]=margin(gs)
Warning: Divide by zero
gm = 2.0000
pm = 21.3864
wpc = 1.0000
wgc = 0.6823
```

oppure

```
>> margin(gs)

Warning: Divide by zero
```

In quest'ultimo caso si ottiene il plot della figura 3.8.

3.3 Analisi con il luogo delle radici

I comandi messi a disposizione dal "Control System Toolbox" sono riportati nella tabella 3.3.

rlocus	diagramma del luogo delle radici
rlocfind	determinare poli e guadagno dal grafico, in modo interattivo
sgrid, zgrid	plot di ξ e ω_n costanti (continuo e discreto)
pzmap	mapping di zeri e poli

Tabella 3.3: Comandi per il luogo delle radici

Anche in questo caso occorre prestare attenzione alle varie sintassi possibili con questi comandi. Per ottenere il grafico del luogo delle radici è sufficiente impostare la funzione di trasferimento o le matrici nel piano degli stati del processo ad anello aperto, e chiamare il comando "rlocus".

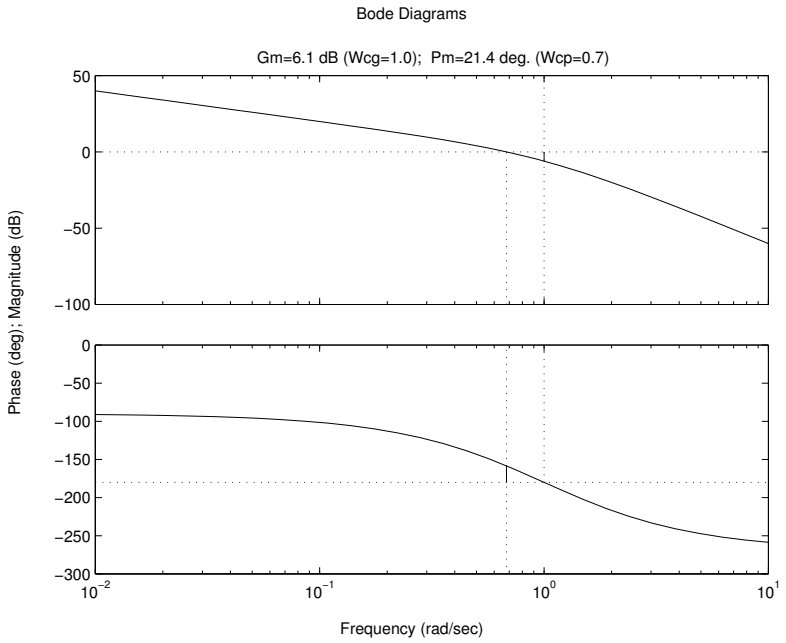


Figura 3.8: Risultato del comando *margin*

```
>> gs=tf([1 3],[1 7 14 8 0]);
>> rlocus(gs)

Warning: Divide by zero

Warning: Divide by zero
```

Si ottiene il grafico della figura 3.9.

Questo grafico può essere ampliato mediante il comando "sgrid", che disegna le linee con fattore di smorzamento ξ costante e quelle di ω_n costante. È possibile specificare i valori di ξ e ω_n in una matrice. Il comando "sgrid" senza parametri mostra le linee dei fattori di smorzamento da 0 a 1 con incremento di 0.1.

```
>> rlocus(gs)

Warning: Divide by zero

Warning: Divide by zero
>> sgrid([0.5],1)
```

Da questi comandi si ottiene il grafico della figura 3.10.

Il comando "rlocfind" permette di risalire ai poli e al guadagno corrispondenti ad un certo punto scelto con il mouse sul grafico. La sintassi di questo comando è

```
>> [k,p]=rlocfind(gs)
```

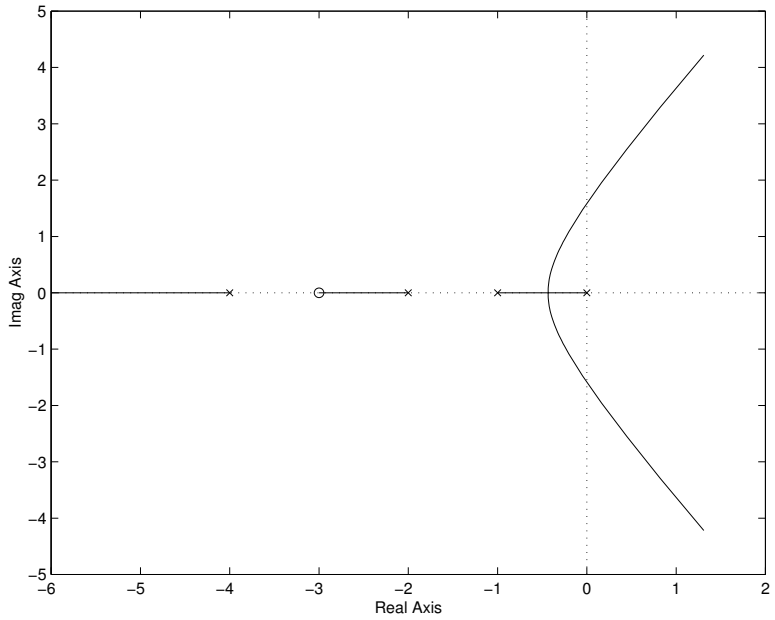


Figura 3.9: Grafico del luogo delle radici

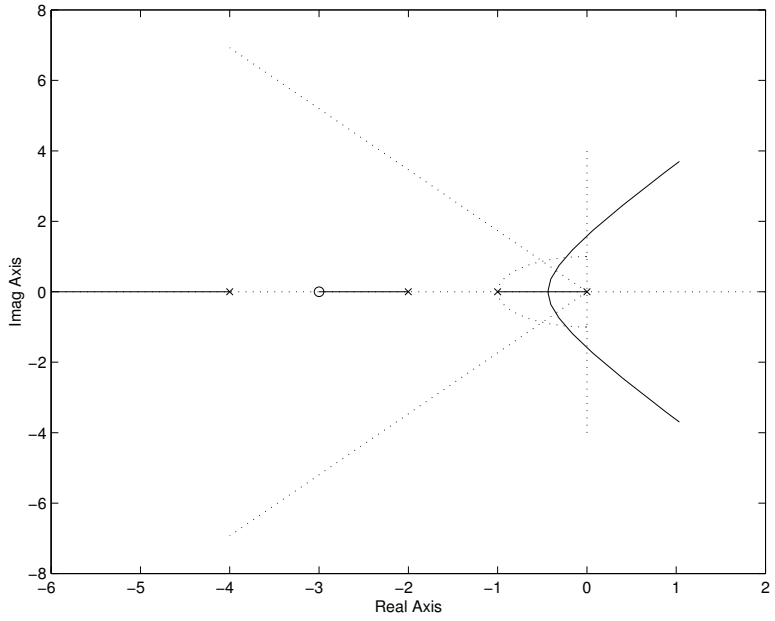


Figura 3.10: Grafico del luogo delle radici con fattori ξ e ω


```
Select a point in the graphics window

selected_point =
  -0.3510 + 0.6804i
k =
    1.7427
p =
  -4.0721
  -2.2602
  -0.3339 + 0.6757i
  -0.3339 - 0.6757i
```

Dopo aver lanciato il comando, viene mostrata la finestra del luogo delle radici, mentre il cursore del mouse diventa una croce. Selezionando un punto sul luogo delle radici, viene subito mostrato il risultato dell'operazione ("selected point") in forma di guadagno "k" e posizione dei poli "p".

Il luogo delle radici può essere utilizzato anche con sistemi discreti. L'unica differenza consiste nel disegno della griglia, che deve essere fatto con il comando "zgrid".

```
>> gs=tf(1,[1 2 1]);
>> gz=c2d(gs,1,'zoh');
>> rlocus(gz),zgrid
```

Da questi comandi si ottiene il grafico della figura 3.11.

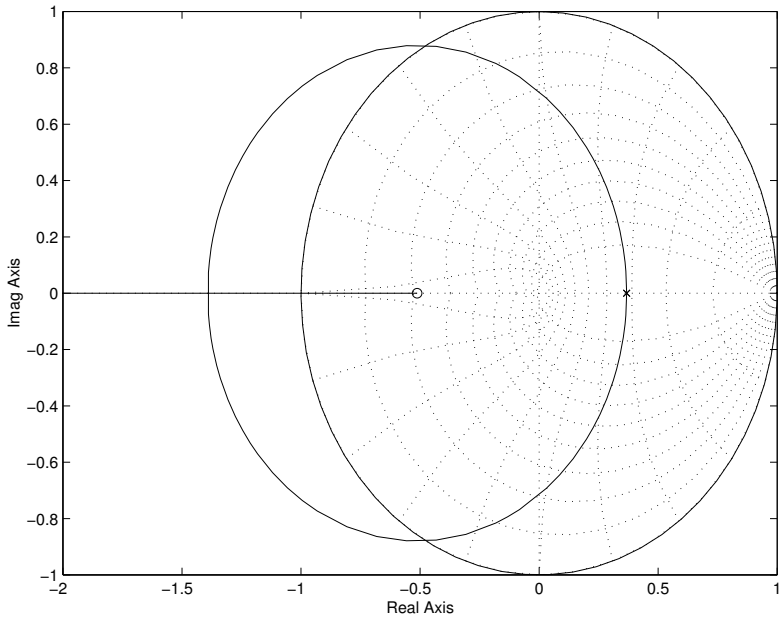


Figura 3.11: Grafico del luogo delle radici di un sistema discreto

Anche in questo caso sono marcate le curve di ξ e ω_n costanti, oltre al cerchio unitario (intervallo di stabilità).

3.4 Proprietà del processo

La tabella 3.4 mostra altre funzioni che permettono di ricavare le proprietà di un sistema.

ctrb	matrice di controllabilità
damp	fattore di smorzamento e frequenza naturale
dcgain	guadagno in dc di un sistema (valore finale)
gram	gramiani di osservabilità e controllabilità
esort	sorting degli autovalori
obsv	matrice di osservabilità
tzero	zeri di trasmissione (guadagno)

Tabella 3.4: Altre proprietà di un sistema

Esempio 3.2 *Consideriamo la funzione di trasferimento*

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

```
>> gs=tf([2],[1,1,1]);
>> dcgain(gs)
ans =
     2
>> damp(gs)
   Eigenvalue         Damping        Freq. (rad/sec)
  -0.5000 + 0.8660i   0.5000             1.0000
  -0.5000 - 0.8660i   0.5000             1.0000
```

3.5 L'applicazione "LTIVIEWER"

Tutte le caratteristiche precedentemente viste, possono essere analizzate anche mediante una finestra interattiva che può essere aperta con il comando "ltiview". All'interno di questa applicazione possono essere visualizzate e determinate quasi tutte le caratteristiche di un processo già viste precedentemente.

Capitolo 4

Costruzione di sistemi

4.1 Connessioni

La costruzione di complessi processi con feedback, diverse funzioni di trasferimento, entrate ed uscite, può essere fatta in modo molto semplice con SIMULINK. L'utilizzo di questo prodotto permette inoltre di modellare anche le diverse nonlinearità del processo.

Anche non avendo a disposizione SIMULINK, è sempre possibile modellare il processo all'interno di MATLAB, sfruttando le proprietà degli oggetti LTI, mediante i comandi riportati nella tabella 4.1.

$+, *, -, \ldots$	parallelo, serie, feedback,...
append	integra due sottosistemi in un unico sistema
augstate	aumenta gli stati di output (per feedback degli stati)
parallel	connessione parallela di due sistemi
series	connessione in serie di due sistemi
feedback	connessione a feedback di due sistemi
connect	costruzione di un modello complesso mediante le sue connessioni
ssselect ssdelete	seleziona sottosistemi elimina input, output o variabili di stato
pade	approssimazione di Padé per time delay

Tabella 4.1: Comandi per interconnettere blocchi

Quale esempio consideriamo il sistema della figura 4.1.

La simulazione può essere fatta direttamente con Simulink, oppure con la sequenza di comandi seguente:

```
>> motore=tf([6.63],[1,101.71,171,0]);
>> regolatore=tf([106.554,258],[0.0518,1]);

>>gtot=feedback(regolatore*motore,1)

Transfer function:
```

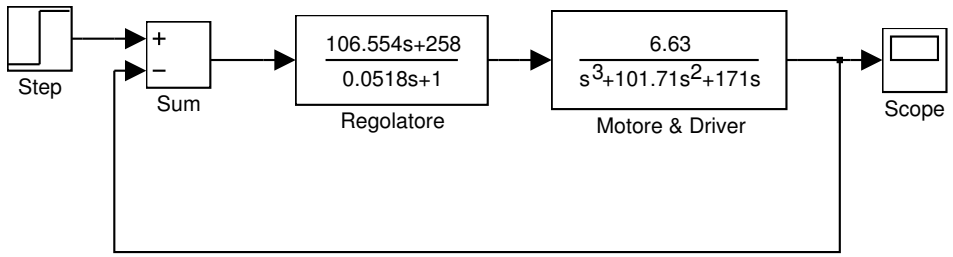


Figura 4.1: Processo e regolatore

```
                  706.5 s + 1711
-------------------------------------------------
0.0518 s^4 + 6.269 s^3 + 110.6 s^2 + 877.5 s + 1711
>> step(gtot),grid
```

Il grafico della simulazione è riportato nella figura 4.2.

4.2 Riduzione

Il toolbox di controllo contiene una serie di funzioni che permettono di ridurre il numero degli stati di un sistema (vedi tabella 4.2).

minreal	cancellazione zeri-poli
balreal	realizzazione bilanciata
modred	riduzione dell'ordine del modello
ssdelete	cancellazione di input, output o stati

Tabella 4.2: Comandi per la riduzione di un processo

Esempio 4.1 *Consideriamo la funzione di trasferimento*

$$G(s) = \frac{(s+10)(s+20.01)}{(s+5)(s+9.9)(s+20.1)}$$

```
>> g=zpk([-10,-20.01],[-5,-9.9,-20.1],1);
>> [sysb,g,t]=balreal(g)

a =
                x1          x2          x3
   x1     -4.96972     0.23990    -0.22617
   x2     -0.23990    -4.27555     9.46710
   x3     -0.22617    -9.46710   -25.75473
```

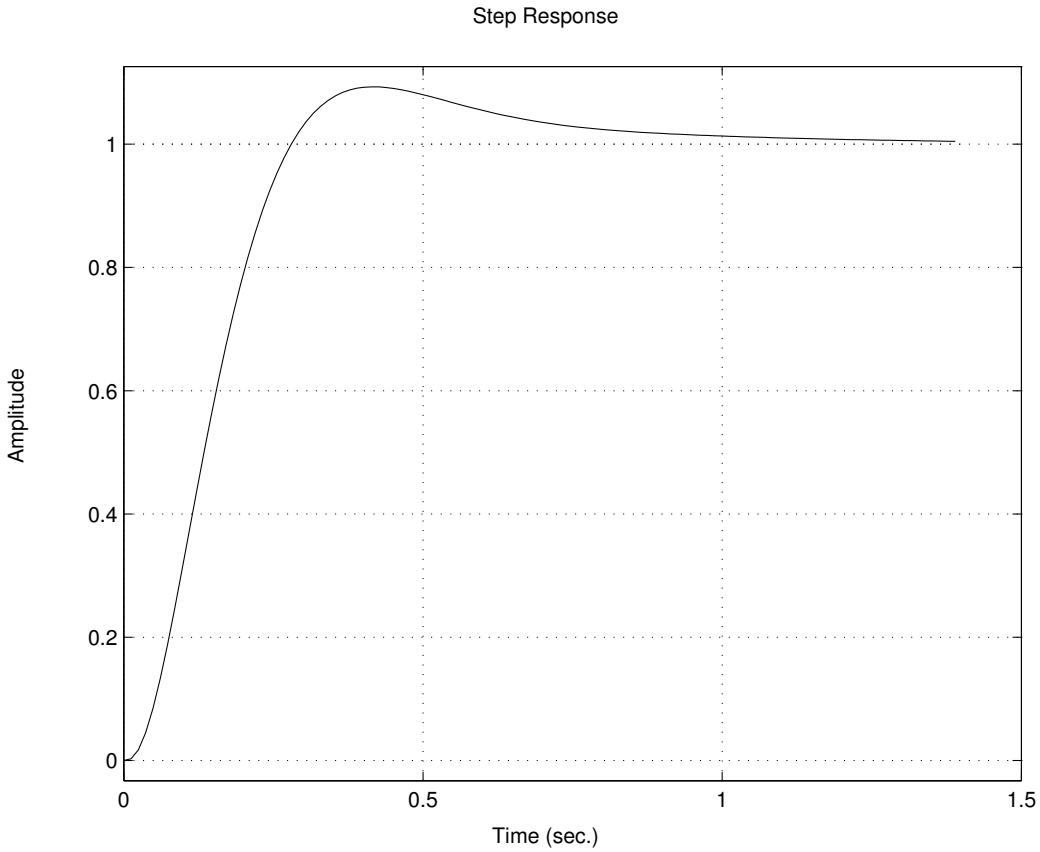


Figura 4.2: Risultato della simulazione

```
b =
                      u1
           x1      1.00003
           x2      0.02412
           x3      0.02276

c =
                      x1           x2           x3
           y1      1.00003     -0.02412      0.02276

d =
                      u1
           y1            0

Continuous-time system.
g =
    0.1006
    0.0001
    0.0000
t =
    1.5860    0.1509    0.1876
  -38.8184    2.4322    3.0126
   64.9259   -4.0522   -5.0486

>> sys2=modred(sysb,[2,3])
a =
                      x1
           x1    -4.98119
b =
                      u1
           x1     1.00119
c =
                      x1
           y1     1.00119
d =
                      u1
           y1    -0.00012

Continuous-time system.

>> g2=zpk(sys2)

Zero/pole/gain:

-0.00011597 (s-8638)
--------------------
     (s+4.981)
```

4.3 Trasformazioni similari

Nello spazio degli stati è possibile trasformare un sistema descritto dalle matrici a, b, c, d, in uno spazio similare mediante alcune funzioni di trasformazione. Le due funzioni più importanti sono "canon" e "ss2ss".

Esempio 4.2 *Consideriamo il sistema*

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

```
>> g=zpk([],[-1,-2,-3],1);
>> plant=ss(g);
> sys2=canon(plant,'modal')

a =
                      x1           x2           x3
           x1     -3.00000            0            0
           x2            0     -2.00000            0
           x3            0            0     -1.00000
 b =
                      u1
           x1      0.50000
           x2     -1.73205
           x3     -1.22474

c =
                      x1           x2           x3
           y1      1.00000      0.57735     -0.40825
d =
                      u1
           y1            0

Continuous-time system.
>> sys2=canon(plant,'companion')

a =
                      x1           x2           x3
           x1            0            0     -6.00000
           x2      1.00000            0    -11.00000
           x3            0      1.00000     -6.00000

b =
                      u1
           x1      1.00000
           x2            0
           x3            0
```

```
c =
                    x1            x2            x3
      y1             0             0       1.00000

d =
                    u1
      y1             0

Continuous-time system.
```

Capitolo 5

Regolatori in cascata

5.1 Regolatori nel continuo

In questo capitolo vedremo i comandi messi a disposizione da MATLAB e del "Control System Toolbox" per aiutare il design di un compensatore, senza entrare in dettaglio nella teoria.

Prendiamo come esempio un sistema formato da un driver di potenza e da un motore dc utilizzato come servomeccanismo per far girare un'antenna parabolica di un certo angolo ϕ. Tralasciando il valore dell'induttanza L dell'armatura del motore, la funzione di trasferimento del sistema ad anello aperto (driver + motore) può essere modellata con la funzione seguente:

$$G(s) = \frac{\Phi(s)}{U(s)} = \frac{6.63}{s(s+1.71)(s+100)}$$

Obiettivo del design del compensatore è quello di ottenere una costante di errore di velocità K_V pari a 10 e un margine di fase (PM) di ca. 60^0. Vogliamo determinare un regolatore di tipo lead che rispetti le specifiche. Il compensatore avrà la forma

$$G_R(s) = K_c \frac{1+\alpha Ts}{1+Ts}$$

Per prima cosa dobbiamo determinare il fattore di amplificazione K_c necessario per soddisfare le specifiche di errore allo stato finito. Possiamo utilizzare la funzione "dcgain" per determinare il valore cui tende il nostro processo.

```
>> mot=zpk([],[0 1.71 100],6.63)
>> [num,den]=tfdata(mot,'v');
>> dcgain(num,den(1:length(den)-1))
ans =
    0.0388
```

Il valore della costante d'errore di velocità è molto più basso di quello desiderato. Conseguentemente occorre amplificare il sistema con un valore di K_c pari a

```
>> Kc=10/ans
Kc =
  257.9186
```

Ora possiamo analizzare il diagramma di Bode del sistema con compensatore proporzionale (vedi figura 5.1)

```
>> bode(Kc*mot)
```

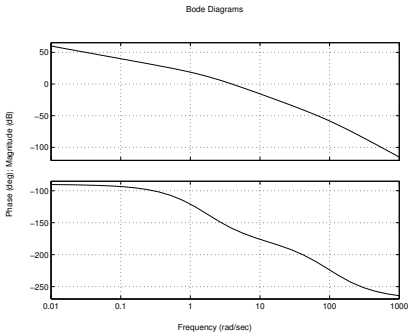


Figura 5.1: Diagramma di bode del motore con regolatore P

Determiniamo ora il margine di fase del sistema con compensatore proporzionale.

```
>> [gm,pm,wpc,wgc]=margin(Kc*mot)

Warning: Divide by zero
gm =
   10.1710
pm =
   21.0839
wpc =
   13.0767
wgc =
    3.9607
```

Il sistema presenta un margine di fase di ca. 21^o ; occorre quindi una fase supplementare di 39^o, che, considerando 5^o supplementari (sicurezza), diventano 44^o.

Sfruttiamo il diagramma di Bode per determinare il valore di α che ci fornisce il supplemento di fase desiderato.

```
>> fi=44/180*pi;
>> alfa=(1+sin(fi))/(1-sin(fi))
alfa =
    5.5500
```

Il valore di α viene ricavato da

$$\alpha = \frac{1+\sin\Phi}{1-\sin\Phi}$$

```
>> mag_g,phas_g,w]=bode(Kc*mot);
>> mag_g_db=20*log10(mag_g);
>> mag_m=-10*log10(alfa);
>> wgc=spline(mag_g_db,w,mag_m)
 wgc =
     6.2267
```

Mediante un'interpolazione "spline" si ricava la nuova ω_{gc}.

```
>> T=1/(wgc*sqrt(alfa))
T =
    0.0682
```

Il valore di T si ricava dalla formula

$$T = \frac{1}{\sqrt{\alpha}\omega_{gc}}$$

```
>> nreg=Kc*[alfa*T 1]
nreg =
   97.5834  257.9186
>> dreg=[T 1]
dreg =
1.0000

>> reg=tf(nreg,dreg)

Transfer function:

   97.58 s + 257.9
   ---------------
     0.06817 s + 1
```

Il compensatore è determinato da nreg/dreg

La simulazione porta al grafico della figura 5.2.

```
>> g=reg*mot;
>> gtot=feedback(g);
>> step(gtot),grid
```

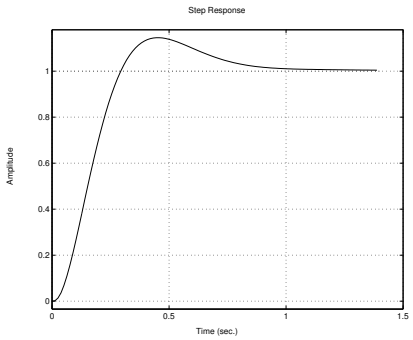


Figura 5.2: Simulazione del sistema compensato

```
>> [n,d]=tfdata(g,v);
>> Kv=dcgain(n,d(1:length(d)-1))
Kv =
   10.0000
>> [gm,pm,wpc,wgc]=margin(g)

Warning: Divide by zero
gm =
   16.4161
pm =
   55.7931
wpc =
   36.8818
wgc =
    6.2267
```

Il controllo delle specifiche mostra che l'errore allo stato finito è compensato, mentre il margine di fase è leggermente inferiore a quanto richiesto (55.79^o).

Il risultato della simulazione può essere ottenuto anche con SIMULINK tramite lo schema

della figura 5.3.

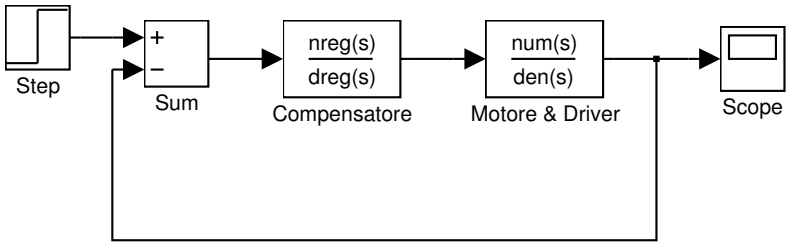


Figura 5.3: Schema a blocchi del motore compensato

5.2 Regolatori nel discreto

La simulazione e analisi nel discreto è analoga a quella nel continuo. Con SIMULINK, in fase di simulazione, si consiglia di utilizzare sempre il sistema continuo preceduto dal mantenitore di ordine zero (Zero-Order-Hold), e non l'equivalente funzione di trasferimento discreta.

Per il calcolo di compensatori discreti si possono utilizzare vari metodi:

- metodi analitici per PID e lead-lag
- metodi grafici (rlocus, dbode)
- determinazione del regolatore nel continuo e trasformazione con o senza l'utilizzo della trasformata w'

La trasformata w' è molto utile quando il tempo di campionamento è piuttosto grande. Questa trasformazione viene fatta utilizzando i comandi "c2d" e "d2c", specificando il metodo "tustin" . Dopo aver fatto una normale trasformazione, per esempio con metodo 'zoh', dal dominio continuo al dominio discreto, si può effettuare la trasformazione

$$z = \frac{2 + Tw'}{2 - Tw'}$$

e ottenere una nuova funzione di trasferimento G(w').

La variabile w' permette di analizzare nel continuo la funzione di trasferimento, potendo utilizzare i metodi di analisi e di design in frequenza (Bode) per ottenere un regolatore. Una volta trovato il regolatore $G_{REG}(w')$ si può ottenere il regolatore D(z) facendo la trasformata w' inversa di $G_{REG}(w')$, o sostituendo

$$w' = \frac{2z - 1}{Tz + 1}$$

MATLAB permette di fare le varie trasformazioni in modo diretto secondo lo schema della figura 5.4.

Esempio 5.1 *Riprendendo il nostro motore possiamo trovare un compensatore con le operazioni seguenti*

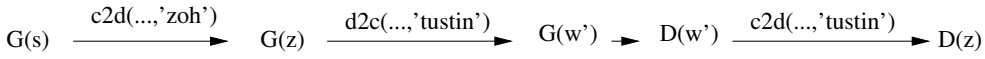


Figura 5.4: Procedimento per determinare il regolatore con trasformata w'

```
>> mot=zpk([],[0 -1.71 -100],6.63)

Zero/pole/gain:
       6.63
------------------
s (s+1.71) (s+100)

>> motd=c2d(mot,0.05)

Zero/pole/gain:
5.4937e-005 (z+1.716) (z+0.05732)
---------------------------------
  (z-0.006738) (z-1) (z-0.9181)

Sampling time: 0.05
>> motw=d2c(motd,'tustin')

Zero/pole/gain:
9.608e-006 (s-151.7) (s+44.86) (s-40)
-------------------------------------
        s (s+39.46) (s+1.709)

>> reg=bdgeo_ld(motw,200,60)

Transfer function:
87.47 s + 200
-------------
0.07323 s + 1

>> dreg=c2d(reg,0.05,tustin)

Transfer function:
941.4 z - 839.6
---------------
   z - 0.491

Sampling time: 0.05
>> step(feedback(dreg*motd,1)),grid
```

Il risultato della simulazione è riportato nella figura 5.5.

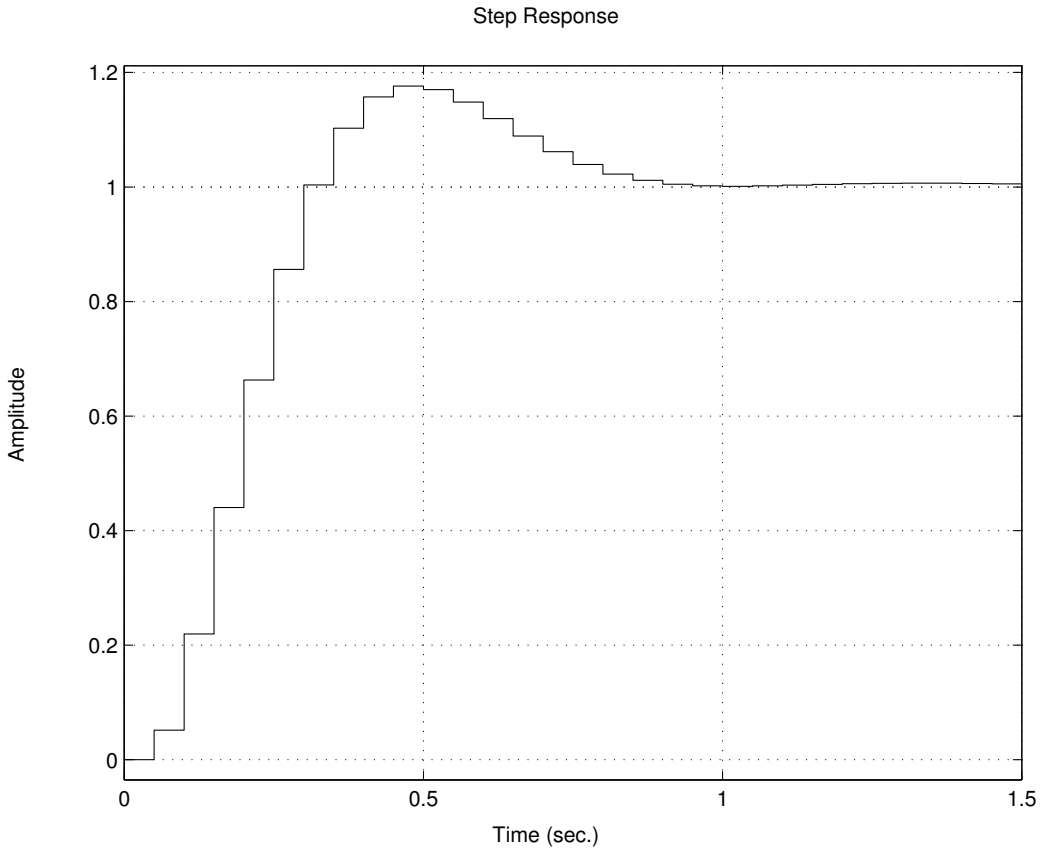


Figura 5.5: Simulazione del sistema discreto (tustin)

Mediante SIMULINK potremmo ottenere un risultato più reale. Costruiamo dapprima il sistema della figura 5.6.

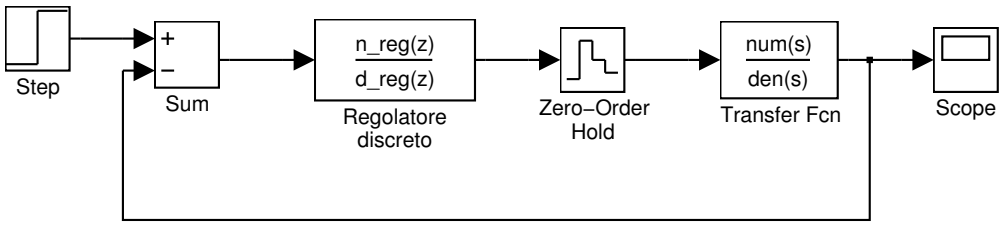


Figura 5.6: Simulazione del sistema digitale (tustin)

La simulazione mediante SIMULINK ci fornisce il grafico della figura 5.7.

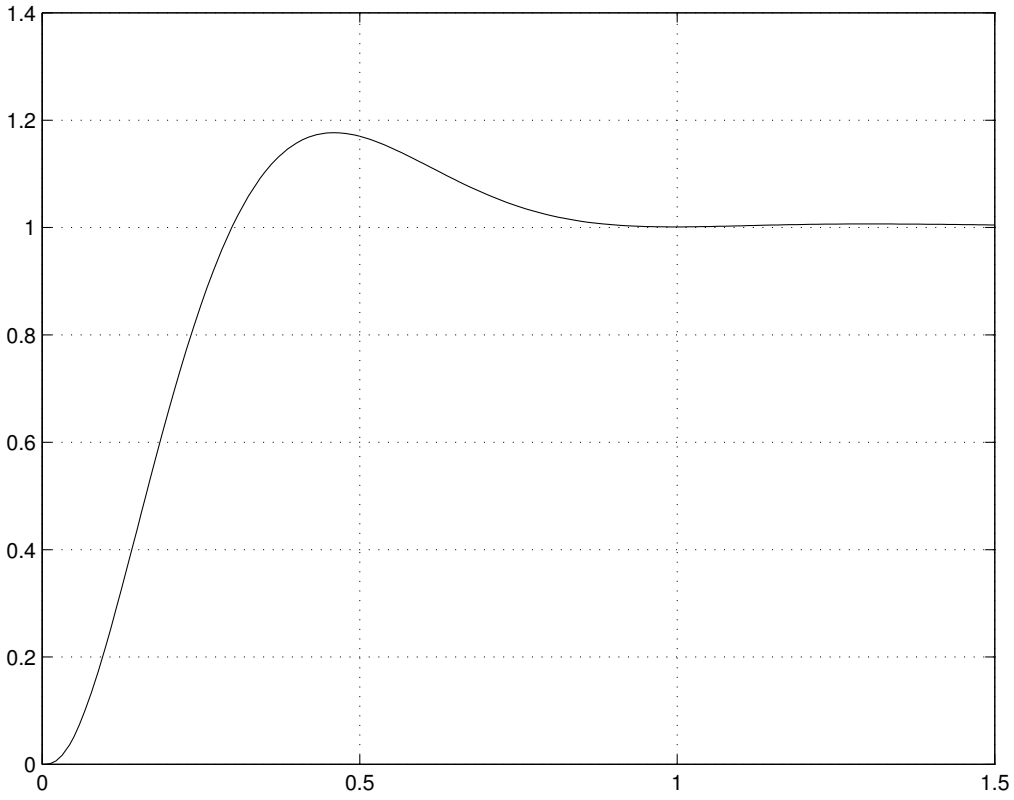


Figura 5.7: Simulazione del sistema discreto da SIMULINK (tustin)

5.3 Funzioni implementate presso la STS

Presso la SUPSI sono state implementate diverse funzioni che permettono un design automatizzato di regolatori. Queste funzioni sono riportate nelle tabelle 5.1 e 5.2.

os2xi	da %OS a ξ
pm2xi	da margine di fase a ξ
ts2wn	da Tsetting e ξ a ω_n
xi2os	da ξ a %OS
xi2pm	da ξ a margine di fase
xi2th	da ξ a θ (luogo delle radici)
xw2s	da ξ e ω_n a un polo dominante s
init_par	da %OS e Tsetting a ξ, ω_n e polo s dominante
s2k	da polo s nel luogo delle radici a amplificazione K corrispondente

Tabella 5.1: Funzioni di trasformazione

bdana_ld	regolatore lead con bode (metodo analitico)
bdgeo_ld	regolatore lead con bode (metodo geometrico)
bdgeo_lg	regolatore lag con bode (metodo geometrico)
rlana_ld	regolatore lead con luogo delle radici (metodo analitico)
rlgeo_ld	regolatore lead con luogo delle radici (metodo geometrico)
deadbeat	regolatore deadbeat discreto
pid	regolatore PID analitico
ziegnich	regolatore PID secondo Ziegler-Nichols, con il metodo dell'amplificazione limite e quello della funzione di trasferimento
h2d	regolatore discreto D(z) da funzione ad anello chiuso desiderata H(z)

Tabella 5.2: Funzioni di design

Grazie a queste funzioni è possibile determinare molto velocemente un regolatore per un determinato processo.

Esempio 5.2

$$G(s) = \frac{6.63}{s(s+1.71)(S+100)}$$

$$e_\infty(rampa) < 0.1 \quad (5.1)$$

$$Tsetting < 2s \quad (5.2)$$

$$\%OS < 10\% \quad (5.3)$$

```
>> mot=tf([6.63],[1,101.71,171,0]);
>> Kc=258;
>> Tsett=2
Tsett =
     2
>> os=10
os =
    10
>> [xi,wn,s]=init_par(os,Tsett)
xi =
    0.5912
wn =
    3.4906
s =
  -2.0635 + 2.8154i
>> pm=xi2pm(xi)
pm =
   58.5931
>> reg=bdgeo_ld(mot,Kc,pm)

Transfer function:

95.94 s + 258
-------------
0.07195 s + 1
```

Il risultato della simulazione è visibile nella figura 5.8.

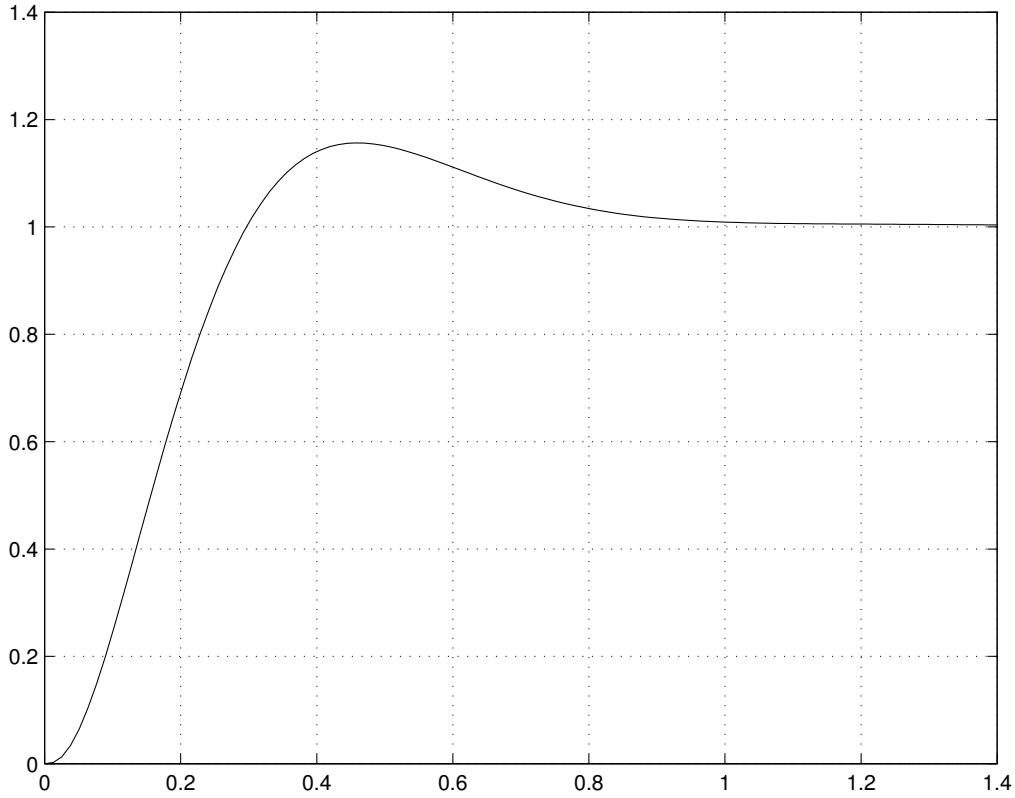


Figura 5.8: Simulazione del sistema

Capitolo 6

Regolazione nel piano degli stati

6.1 Controllabilità e osservabilità

Il toolbox di controllo mette a disposizione delle funzioni che permettono di costruire le matrici di controllabilità ed osservabilità, per poterne poi determinare il rango (vedi tabella 6.1).

ctrb	matrice di controllabilità
obsv	matrice di osservabilità
obsvf	scomposizione in parte osservabile/non osservabile
ctrbf	scomposizione in parte controllabile/non controllabile

Tabella 6.1: Funzioni per controllabilità e osservabilità

6.2 Piazzamento dei poli

La regolazione nel piano degli stati presenta grossi vantaggi ma anche grosse difficoltà, legate soprattutto alla misurazione degli stati o l'utilizzo di osservatori. Il toolbox di controllo mette a disposizione una grande quantità di funzioni predefinite per agevolare lo sviluppo in questo ambito.

Esempio 6.1 *Viene data la funzione di trasferimento*

$$G(s) = 20\frac{s+5}{s^3+5s^2+4s}$$

Determinare il feedback degli stati che dia un %OS di 9.48 e un TSetting di 0.74 s.

```
>> g=tf(20*[1,5],[1,5,4,0]);
>> [x,w,s]=init_par(9.48,0.74)

x =
    0.6000
```

```
w =
    9.3139
s =
  -5.5880 + 7.4514i
```

Scegliamo i due poli dominanti s e conj(s), mentre il 3. polo lo mettiamo a -5, in corrispondenza dello zero del sistema.

```
>> p=[-5,s,conj(s)]
p =
  -5.0000      -5.5880 + 7.4514i  -5.5880 - 7.4514i
```

Per il calcolo della matrice di feedback occorre trasformare il sistema nello spazio degli stati, e quindi risolvere con il comando "place".

```
>> [a,b,c,d]=ssdata(g);
>> k=place(a,b,p)
k =
    2.7940    8.6643    13.5545
```

Ora è possibile simulare il risultato (vedi figura 6.1).

```
>> g2=ss(a-b*k,b,c,d);
>> step(g2),grid
```

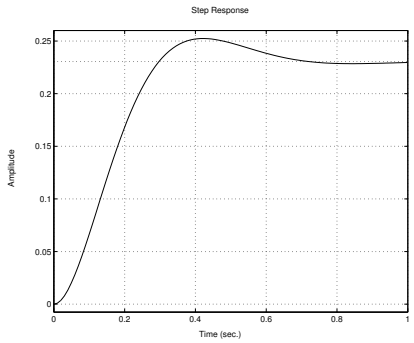


Figura 6.1: Simulazione del sistema con feedback degli stati

6.3 Introduzione di un integratore

Appena si incomincia ad introdurre una parte integrativa, diventa subito più complesso l'utilizzo dei comandi in linea nello shell di MATLAB. Conviene a questo punto utilizzare SIMULINK.

Per il calcolo dei coefficienti della matrice di feedback K all'interno di MATLAB è indispensabile che il modello nello spazio degli stati coincida con quello del blocco di SIMULINK.

Vediamo ora di introdurre una parte integrale nell'esempio visto precedentemente, e di voler trovare un regolatore discreto.

Lo schema di SIMULINK è raffigurato nella figura 6.2.

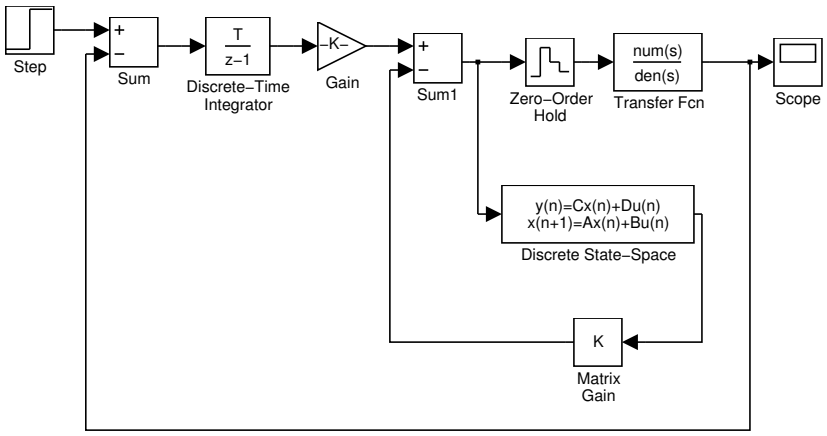


Figura 6.2: Schema del processo con osservatore e feedback

```
>> g=zpk([-5],[0 -1 -4],20)
```

Dati del processo e dell'osservatore

```
Zero/pole/gain:
  20 (s+5)
-------------
s (s+1) (s+4)

>> ts=0.1
ts =
    0.1000
>> gz=c2d(g,ts)

Zero/pole/gain:
0.099698(z+0.9993)(z-0.6065)
----------------------------
 (z-0.6703) (z-1) (z-0.9048)

Sampling time: 0.1
>> [ad,bd,cd,dd]=ssdata(gz);
>> cd=eye(3);
>> dd=zeros(3,1);
```

```
>>[x,w,s]=init_par(9.48,0.74)
```

Specifiche di progetto

```
x =
    0.6000
w =
    9.3139
s =
  -5.5880 + 7.4514i
```



```
>> p=[-20,-5,s,conj(s)]
p =
 -20.0000 -5.0000
-5.5880 + 7.4514i  -5.5880 -
7.4514i
```

poli desiderati nel piano "s"

```
>> pz=exp(p*ts)
pz =
   0.1353             0.6065
0.4203 + 0.3878i   0.4203 -
0.3878i
```

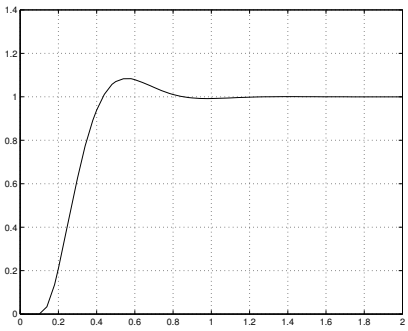
corrispondenti poli in "z"

$$pz = e^{p}T_s$$

```
 k=k_full(gz,pz)
place: ndigits= 15
k =
 3.6421 2.2448 -1.4013 -21.0979
```

calcolo dei coefficienti di feedback (la funzione k_full è stata implementata presso la nostra scuola)

Simulazione



Questi risultati si potevano ottenere anche utilizzando le funzioni "sys_full" e "ss_full" implementate presso la nostra scuola, che permettono di ottenere il sistema ad anello chiuso con integratore dal processo ad anello aperto. Vedremo come utilizzare questa funzione nel caso di sistemi LQR.

6.4 Sistemi LQR

Le problematiche legate all'ottimizzazione dei parametri in funzione di una cifra di merito, possono portare ad ottimizzare una funzione del tipo

$$J = \int_0^T (x^T Q x + u^T R u) dt$$

Questo regolatore viene chiamato "Regolatore LQR" ("linear quadratic regulator").

I fattori Q e R all'interno dell'integrale servono a dare peso alle varie componenti del sistema, in modo da poter ottimizzare il sistema privilegiando il consumo energetico o l'equilibrio degli stati.

Dalla teoria, la ricerca del feedback degli stati ottimale

$$u = -Kx$$

è determinato con la soluzione dell'equazione di Riccati

$$A^T P + PA + Q - PBR^{-1}B^T P = 0$$

e la matrice K si ricava da

$$K = R^{-1}B^T P$$

Le funzioni messe a disposizione per il calcolo dei regolatori LQR sono descritte nella tabella 6.2.

lqr, dlqr	regolatore LQR (continuo e discreto)
lqrd	regolatore LQR discreto da J continua
lqry	regolatore LQR con peso su u e y
care, dare	risoluzione dell'equazione di Riccati (continua e discreta)

Tabella 6.2: Funzioni LQR

Queste funzioni permettono di determinare il feedback ottimale K.

Esempio 6.2 *Il motore dc già visto precedentemente viene controllato mediante un sistema LQR, in cui la cifra di merito viene descritta da*

$$J = \int_0^T (x^T Q x + u^T R u) dt, Q = 1, R = 0.001$$

Il sistema da compensare comprende anche un integratore per portare a 0 l'errore allo stato finito con entrata gradino, come già visto con il feedback degli stati.

Q viene applicato solo alla variabile di stato creata con il feedback supplementare tra entrata e uscita necessario per la parte integrale. In questo modo viene data molta importanza all'errore tra entrata e uscita.

```
>> mot=zpk([],[0 -1.71 -100],6.63);
>> Q=[0,0;0,1];
>> R=0.001;
>> mot_full=sys_full(mot);
>> k=lqry(mot_full,Q,R)
k =
    0.4071    1.1588    1.4230  -31.6228
>> mot_reg=ss_full(mot,k);
>> step(mot_reg),grid
```

Il risultato di questa simulazione è mostrato nella figura 6.3.

6.5 Sistemi LQG

Industrialmente si preferisce l'utilizzazione di sistemi LQG e dei filtri di Kalman-Bucy per effettuare la stima degli stati di un sistema, per poi in seguito fare un feedback degli stati. Questo è dovuto principalmente al fatto che la misura degli stati, anche se possibile,

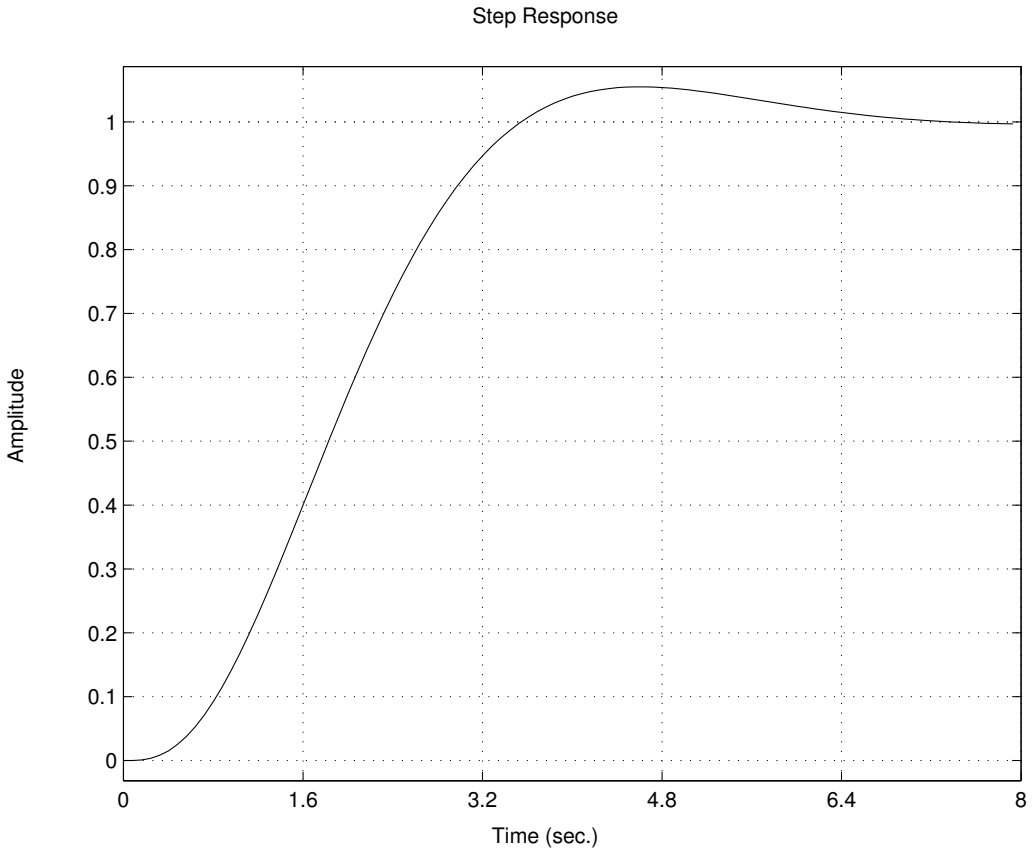


Figura 6.3: Simulazione del sistema con regolatore LQR

presenta sempre un certo rumore di fondo. Occorre quindi ottenere, con l'osservatore, la migliore stima degli stati da un set di misure corrotte dal rumore di fondo. Il problema può essere formulato nel modo seguente

$$\dot{x} = Ax + Bu + \Gamma\omega$$

$$y = Cx + \nu$$

dove ω rappresenta un'entrata di rumore di fondo casuale e ν rappresenta il rumore di fondo dei sensori. Entrambi i segnali di disturbo sono da considerare gaussiani con media 0 e covarianza conosciuta. Valgono le uguaglianze seguenti

$$E\{\omega(t)\} = 0$$

$$E\{\nu(t)\} = 0$$

$$E\{\omega(t)\omega(t+\tau)'\} = Q_0\delta(t-\tau)$$

$$E\{\nu(t)\nu(t+\tau)'\} = R_0\delta(t-\tau)$$

$$E\{\omega(t)\nu(t+\tau)'\} = 0, \forall t, \tau$$

Il problema è quello di ottenere una stima dei valori di x(t) basandosi sulle misure disturbate, in modo che la varianza dell'errore sia minima. La cifra di merito viene definita come

$$J_0 = E[(x(t) - \hat{x}(t))'(x(t) - \hat{x}(t))]$$

e rappresenta la varianza dell'errore, dove x(t) è il valore misurato e $\hat{x}(t)$ il valore stimato. L'osservatore ottimale è quello descritto dalle equazioni

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{y})$$

$$L = \sum C'R^{-1}$$

dove Σ si trova risolvendo l'equazione di Riccati

$$A\Sigma + \Sigma A' + \Gamma Q_0 \Gamma' - \Sigma C' R_0^{-1} C\Sigma = 0$$

Per risolvere questo tipo di problemi MATLAB mette a disposizione i comandi riportati nella tabella 6.3.

kalman	Osservatore di Kalman
kalmd	Osservatore di Kalman discreto da sistema continuo
lqgreg	Crea il regolatore dal guadagno e dall'osservatore di Kalman

Tabella 6.3: Funzioni LQG

Nella SUPSI sono stati implementati alcuni moduli per rappresentare in SIMULINK l'osservatore di Kalman discreto.

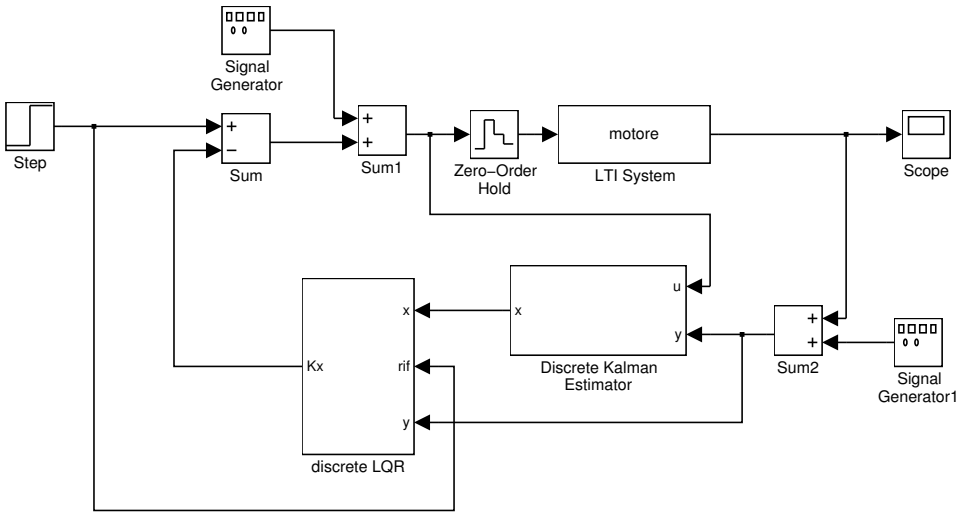


Figura 6.4: Sistema con regolatore LQG

Continous plant	motore2
Qn	0.01
Rn	0.01
Sample time	0.1

Tabella 6.4: Parametri del filtro di Kalmann

Esempio 6.3 *Riprendendo il nostro motore dc, implementiamo l'osservatore di Kalman e un regolatore LQR per il feedback degli stati. Costruiamo lo schema di figura 6.4.*

Le impostazioni da fornire al modulo "Discrete Kalman Estimator" sono riportate nella tabella 6.4.

Il modulo "motore2" è stato costruito da "motore" sostituendo la matrice b con $[b, \Gamma]$*;* Γ *qui è stata posta uguale a b.*

Il modulo "discrete kalman Estimator" contiene il blocco raffigurato nella figura 6.5.

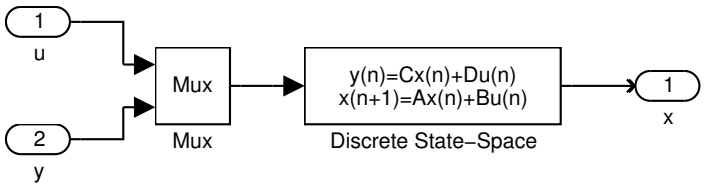


Figura 6.5: Osservatore

Questo sistema viene inizializzato da un programma di inizializzazione contenuto nella maschera del blocco:

```
sys=ss(sys);
[a,b,c,d]=ssdata(sys);
[kest,L,P,M,Z]=kalmd(sys,Qn,Rn,Ts);
[ae,be,ce1,de1]=ssdata(kest);
n=length(a);
[ny,nu]=size(d);
ce=ce1(ny+1:ny+n,:);
de=de1(ny+1:ny+n,:);
```

Per quanto riguarda il modulo "discrete LQR", pure implementato presso la nostra scuola, i parametri che devono essere passati sono quelli riportati nella tabella 6.5.

Continous plant	motore
Q	[0,0;0,10]
R	0.001
Sample time	0.1

Tabella 6.5: Parametri per il regolatore LQR

Occorre prestare un'attenzione particolare alla matrice Q, di tipo 2x2; il sistema modificato possiede 2 uscite, quella del processo y(t) e quella creata con l'integrazione supplementare che non è altro che l'errore tra segnale di riferimento ed uscita.

L'ottimizzazione viene fatta mediante il comando lqry. Come nell'esempio precedente si è voluto dare maggiore peso alla variabile errore, piuttosto che all'uscita del processo.

All'interno del modulo viene fatta l'inizializzazione dei parametri con i comandi

```
sys=ss(sys);
```

```
sysd=c2d(sys,Ts);
[ad,bd,cd,dd]=ssdata(sysd);
[m1,n1]=size(ad);
[m2,n2]=size(bd);
[m3,n3]=size(cd);
aa=[ad,zeros(m1,1);-cd*Ts,1];
bb=[bd;zeros(1,n2)];
cc=[cd,zeros(m3,1);zeros(1,n3),1];
dd=[dd;zeros(1,n2)];
sysd2=ss(aa,bb,cc,dd,Ts);
[k,s,e]=lqry(sysd2,Q,R);
```

Il risultato della simulazione è riportato nella figura 6.6.

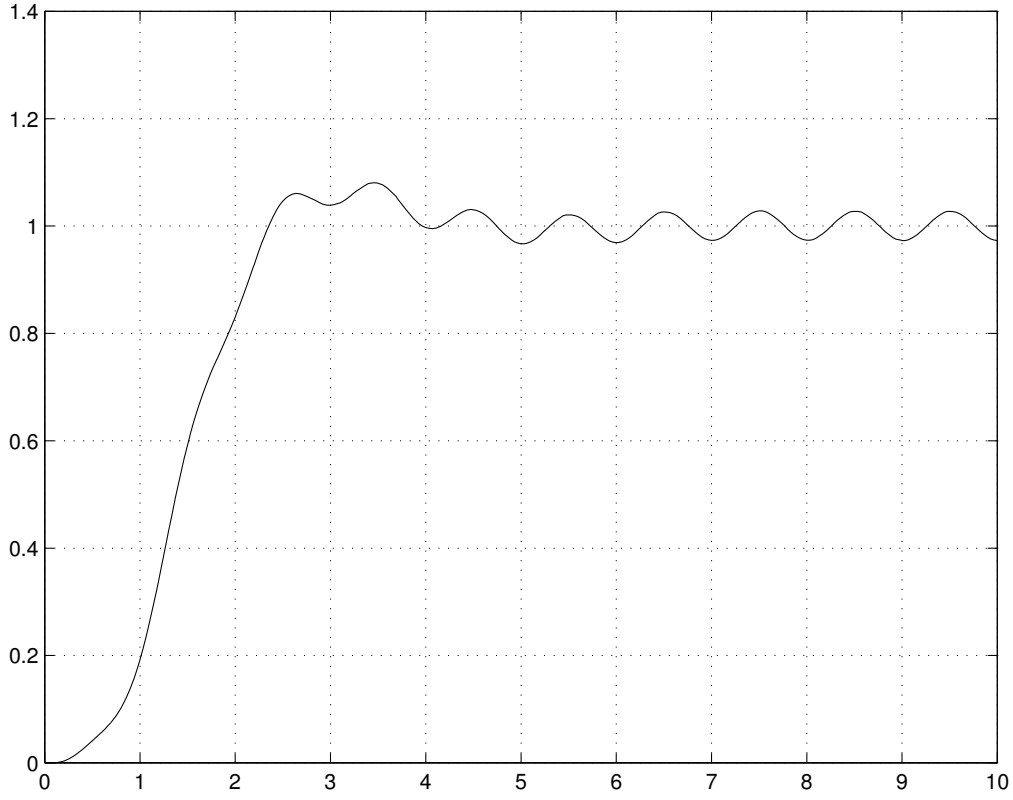


Figura 6.6: Simulazione del sistema LQG

Capitolo 7

Controllo robusto

7.1 Introduzione

Pur esistendo un toolbox specifico per il controllo robusto, alcune operazioni di base sono possibili anche utilizzando unicamente il "Control System Toolbox" e le operazioni di base di MATLAB. Le operazioni più importanti sono riportate nella tabella 7.1.

svd	decomposizione in valori singolari di una matrice, utile con sistemi di tipo MIMO
sigma	diagramma di bode dei valori singolari

Tabella 7.1: Operazioni per il controllo robusto

Esempio 7.1 *Vogliamo vedere il diagramma dei valori singolari del sistema MIMO descritto dalla funzione di trasferimento*

$$H(s) = \begin{bmatrix} 0 & \dfrac{3s}{s^2+s+10} \\ \dfrac{s+1}{s+5} & \dfrac{2}{s+6} \end{bmatrix}$$

Costruiamo dapprima lo schema di SIMULINK *riportato nella figura 7.1.*
Ora determiniamo il modello del sistema

```
>> [a,b,c,d]=linmod('syst')
a =
    -1   -10     0     0
     1     0     0     0
     0     0    -5     0
     0     0     0    -6
```

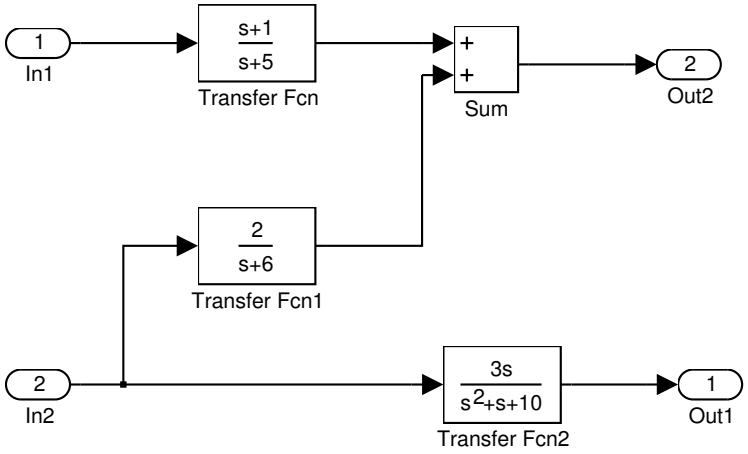
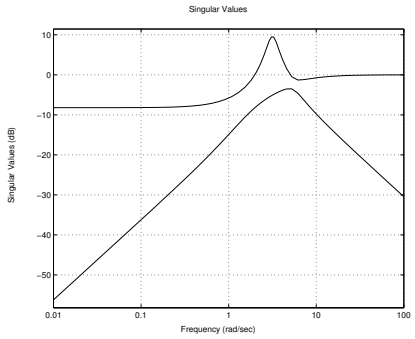


Figura 7.1: Schema del processo

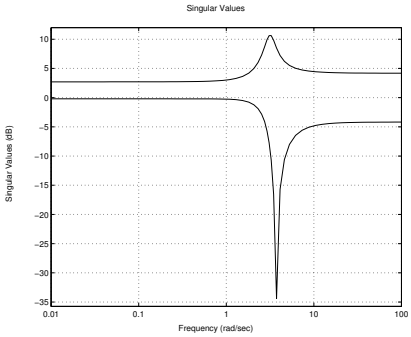
```
b =
     0     1
     0     0
     1     0
     0     1
c =
     3     0     0     0
     0     0    -4     2
d =
     0     0
     1     0

>> sys=ss(a,b,c,d);
```

Costruiamo ora il plot di H(s) e quello di I+H(s)

```
>> sigma(sys),grid
```



```
>> sigma(sys,[],2),grid
```



Per semplici analisi di robustezza su incertezze additive e moltiplicative, sono state sviluppate presso la nostra scuola due funzioni, riportate nella tabella 7.2.

rob_mult	analisi di robustezza con incertezze moltiplicative
rob_add	analisi di robustezza con incertezze additive

Tabella 7.2: Funzioni per l'analisi di robustezza

7.2 Criterio di stabilità robusta

L'incertezza moltiplicativa tra funzione di trasferimento approssimata $G(s)$ e funzione reale $G_m(s)$ è definita come

$$G_m(s) = G(s)\,[1 + M(s)]$$

e il criterio di stabilità robusta vale

$$|M(j\omega)| < \left|1 + \frac{1}{G(j\omega)}\right| = \frac{1}{|T(j\omega)|}$$

e deve essere valido per tutti gli ω

Esempio 7.2 *Consideriamo il processo descritto da*

$$G(s) = \frac{170000(s + 0.1)}{s(s + 3)(s^2 + 10s + 10000)}$$

Il sistema è instabile con un regolatore proporzionale pari a 1*, ma può essere stabilizzato con una riduzione di* K *a* 0.5*.*

Nel modello nominale è stato trascurato un polo a -50*. Ciò significa che l'incertezza moltiplicativa è determinata da*

$$[1 + M(s)] = \frac{50}{s + 50}$$

Risolvendo per M(s) si trova

$$M(s) = -\frac{s}{s + 50}$$

Ora, utilizzando la funzione rob_mult, si può controllare se il criterio di stabilità robusta è rispettato.

Come si può vedere dalla figura 7.2 il criterio di stabilità robusta non è rispettato, poiché il grafico di $M(j\omega)$ *passa sopra a quello di* $1/T(j\omega)$.

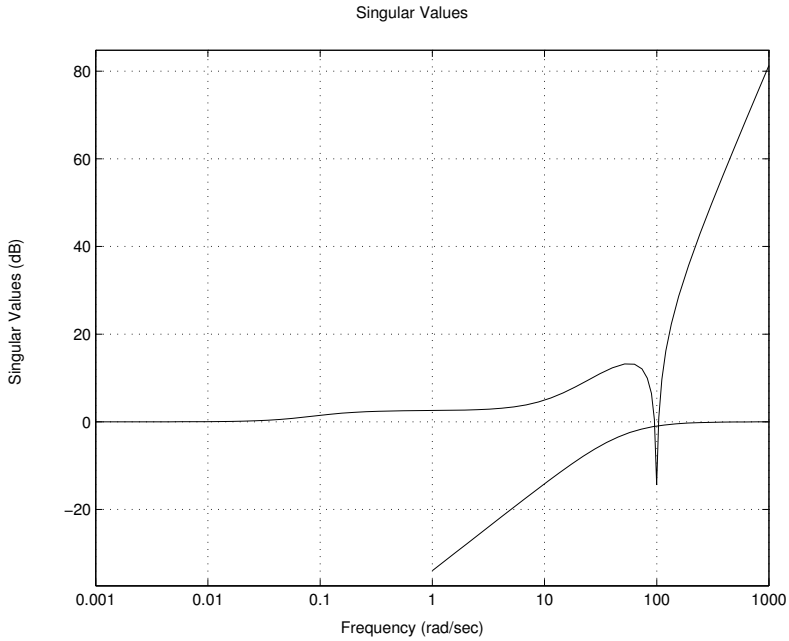


Figura 7.2: Controllo di robustezza con regolatore proporzionale

Se al posto del regolatore proporzionale usiamo un regolatore lag del tipo

$$G_{REG} = 0.15\frac{s+25}{s+2.5}$$

per modificare il sistema compensato alle frequenze

$$2 < \omega < 25$$

otteniamo il grafico di figura 7.3, che mostra che il criterio di stabilità robusta viene rispettato.

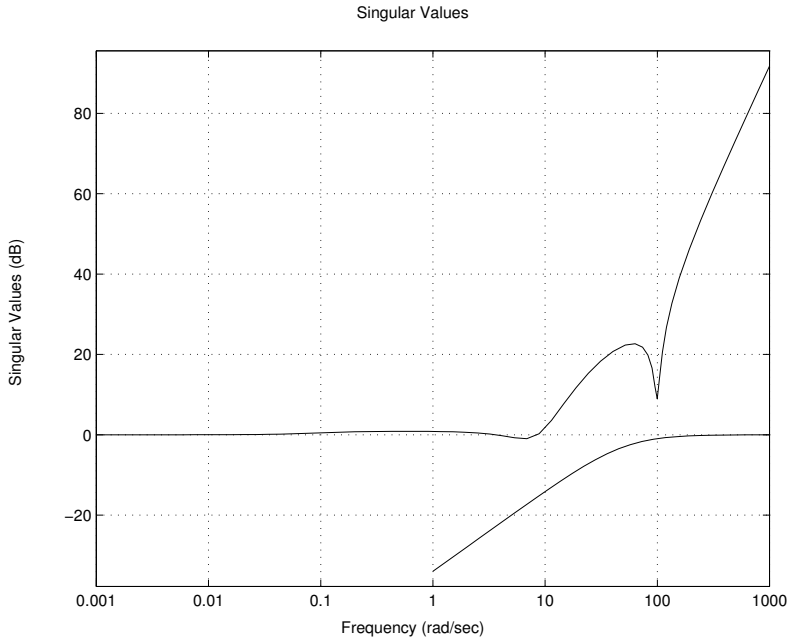


Figura 7.3: Controllo di robustezza con regolatore lag

Appendice A

Comandi di regolazione

A.1 Creation of LTI models.

```
ss          - Create a state-space model.
zpk         - Create a zero/pole/gain model.
tf          - Create a transfer function model.
dss         - Specify a descriptor state-space model.
filt        - Specify a digital filter.
set         - Set/modify properties of LTI models.
ltiprops    - Detailed help for available LTI properties.
```

A.2 Data extraction.

```
ssdata      - Extract state-space matrices.
zpkdata     - Extract zero/pole/gain data.
tfdata      - Extract numerator(s) and denominator(s).
dssdata     - Descriptor version of SSDATA.
get         - Access values of LTI model properties.
```

A.3 Model characteristics.

```
class       - Model type ('ss', 'zpk', or 'tf').
size        - Input/output dimensions.
isempty     - True for empty LTI models.
isct        - True for continuous-time models.
isdt        - True for discrete-time models.
isproper    - True for proper LTI models.
issiso      - True for single-input/single-output systems.
isa         - Test if LTI model is of given type.
```

A.4 Conversions.

```
ss          - Conversion to state space.
zpk         - Conversion to zero/pole/gain.
tf          - Conversion to transfer function.
```

```
c2d         - Continuous to discrete conversion.
d2c         - Discrete to continuous conversion.
d2d         - Resample discrete system or add input delay(s).
```

A.5 Overloaded arithmetic operations.

```
+ and -     - Add and subtract LTI systems (parallel connection).
*           - Multiplication of LTI systems (series connection).
\           - Left divide -- sys1\sys2 means inv(sys1)*sys2.
/           - Right divide -- sys1/sys2 means sys1*inv(sys2).
          - Pertransposition.
.         - Transposition of input/output map.
[..]        - Horizontal/vertical concatenation of LTI systems.
inv         - Inverse of an LTI system.
```

A.6 Model dynamics.

```
pole, eig   - System poles.
tzero       - System transmission zeros.
pzmap       - Pole-zero map.
dcgain      - D.C. (low frequency) gain.
norm        - Norms of LTI systems.
covar       - Covariance of response to white noise.
damp        - Natural frequency and damping of system poles.
esort       - Sort continuous poles by real part.
dsort       - Sort discrete poles by magnitude.
pade        - Pade approximation of time delays.
```

A.7 State-space models.

```
rss,drss    - Random stable state-space models.
ss2ss       - State coordinate transformation.
canon       - State-space canonical forms.
ctrb, obsv  - Controllability and observability matrices.
gram        - Controllability and observability gramians.
ssbal       - Diagonal balancing of state-space realizations.
balreal     - Gramian-based input/output balancing.
modred      - Model state reduction.
minreal     - Minimal realization and pole/zero cancellation.
augstate    - Augment output by appending states.
```

A.8 Time response.

```
step        - Step response.
impulse     - Impulse response.
initial     - Response of state-space system with given initial state.
lsim        - Response to arbitrary inputs.
```

```
ltiview     - Response analysis GUI.
gensig      - Generate input signal for LSIM.
stepfun     - Generate unit-step input.
```

A.9 Frequency response.

```
bode        - Bode plot of the frequency response.
sigma       - Singular value frequency plot.
nyquist     - Nyquist plot.
nichols     - Nichols chart.
ltiview     - Response analysis GUI.
evalfr      - Evaluate frequency response at given frequency.
freqresp    - Frequency response over a frequency grid.
margin      - Gain and phase margins.
```

A.10 System interconnections.

```
append      - Group LTI systems by appending inputs and outputs.
parallel    - Generalized parallel connection (see also overloaded +).
series      - Generalized series connection (see also overloaded *).
feedback    - Feedback connection of two systems.
star        - Redheffer star product (LFT interconnections).
connect     - Derive state-space model from block diagram description.
```

A.11 Classical design tools.

```
rlocus      - Evans root locus.
rlocfind    - Interactive root locus gain determination.
acker       - SISO pole placement.
place       - MIMO pole placement.
estim       - Form estimator given estimator gain.
reg         - Form regulator given state-feedback and estimator gains.
```

A.12 LQG design tools.

```
lqr,dlqr    - Linear-quadratic (LQ) state-feedback regulator.
lqry        - LQ regulator with output weighting.
lqrd        - Discrete LQ regulator for continuous plant.
kalman      - Kalman estimator.
kalmd       - Discrete Kalman estimator for continuous plant.
lqgreg      - Form LQG regulator given LQ gain and Kalman estimator.
```

A.13 Matrix equation solvers.

```
lyap        - Solve continuous Lyapunov equations.
dlyap       - Solve discrete Lyapunov equations.
care        - Solve continuous algebraic Riccati equations.
```

```
dare        - Solve discrete algebraic Riccati equations.
```

A.14 Demonstrations.

```
ctrldemo   - Introduction to the Control System Toolbox.
jetdemo    - Classical design of jet transport yaw damper.
diskdemo   - Digital design of hard-disk-drive controller.
milldemo   - SISO and MIMO LQG control of steel rolling mill.
kalmdemo   - Kalman filter design and simulation.
```