Scuola universitaria professionale
della Svizzera italiana

Dipartimento
Tecnologie
Innovative

SCILAB e SCICOS nella regolazione automatica

Ing. Roberto Bucher

17 settembre 2007

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il campo della controllistica è rappresentato in Scilab da tutta una serie di funzioni specifiche; queste funzioni permettono di ottimizzare il lavoro sia con sistemi lineari che con processi non lineari. Negli ultimi anni si stanno anche sviluppando, attraverso conributi, funzioni relative alle nuove tecniche quali la logica fuzzy e i sistemi neurali. L'utilizzo parallelamente di Scicos non è indispensabile, ma facilita il lavoro, soprattutto nella costruzione grafica del processo da controllare. Molte funzioni accettano diverse rappresentazioni del processo, dalla funzione di trasferimento a quella nello spazio 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.
Alcune funzioni supplementari sono state create per semplificare l'implementazione di parti del controllore. L'appendice A mostra come caricare queste funzioni nell'abmiente Scilab per poterle utilizzare, mentre l'appendice B contiene il codice i queste funzioni.

Capitolo 2

Modelli di sistemi

Esistono diversi metodi per descrivere un processo LTI (Linear Time Invariant) all'interno di Scilab. È possibile utilizzare una rappresentazione lineare nello spazio degli stati o la funzione di trasferimento (Laplace), oppure si può modellare il processo mediante Scicos 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 funzioni possono agire direttamente anche sui sistemi non lineari.
Le principali funzioni necessarie alla costruzione di sistemi sono rappresentati nella tabella 2.1.

syslin	Creazione di un sistema continuo o discreto
./ (comado di feedback)	sistemi in feedback
abcd	estrazione matrici della rappresentazione di stato
dscr	trasformazione da sistema continuo a sistema discreto
ss2tf	spazio degli stati → funzione di trasferimento
tf2ss	funzione di trasferimento → spazio degli stati
ss2ss	trasformazioni da spazio degli stati a spazio degli stati

Tabella 2.1: Funzioni legate ai sistemi

2.1 Sistemi continui

2.1.1 Spazio degli stati

È la rappresentazione più versatile, permettendo anche la descrizione di sistemi nonlineari e sistemi con entrate ed uscite multiple (sistemi MIMO). 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 Scilab 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 Scilab

```
-->a=[0,1;-1,-3];

-->b=[0;1];

-->c=[1,0];

-->d=0;

-->plant=syslin('c',a,b,c,d)
 plant  =


        plant(1)    (state-space system:)

!lss   A   B   C   D   X0   dt   !

        plant(2) = A matrix =

    0.     1.
  - 1.   - 3.

        plant(3) = B matrix =

    0.
    1.

        plant(4) = C matrix =

    1.     0.

        plant(5) = D matrix =

    0.

        plant(6) = X0 (initial state) =

    0.
    0.

        plant(7) = Time domain =

 c
```

2.1.2 Funzione di trasferimento

Mediante un'analisi con le trasformate di Laplace è possibile determinare la funzione di trasferimento che descrive la relazione entrata-uscita per un crto processo. Questa relazione è data da

$$Y(s) = G(s) \cdot U(s)$$

In Scilab l'introduzione della funzione di trasferimento viene fatta sfruttando la variabile $\%s$ È possibile in questo modo descrivere anche processi di tipo MIMO, inserendo in matrici le varie funzioni di trasferimento. È inoltre possibile sostituire la variabile $\%s$ con s, mettendo ad esempio la riga

```
s=%s
```

nello script di inizializzazione di Scilab.
La funzione di trasferimento

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

viene rappresentata tramite

```
-->s=%s
 s  =

    s

-->g=syslin('c',(s+1)/(s^2+5*s+2))
 g  =

      1 + s
   ----------
              2
   2 + 5s + s
```

2.2 Sistemi discreti

I sistemi discreti vengono inizializzati esattamente con la stessa funzione *syslin*, passando però come primo parametro la lettera 'd' (discrete) al posto di 'c' (contimous). Inoltre si può ridefinire la variabile $\%z$ esattamente come già fatto per la variabile $\%s$ ed usarla direttamente nella scrittura delle funzioni di trasferimento.

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

$$x[n+1] = A \cdot x[n] + B \cdot u[n] \tag{2.4}$$
$$y[n] = C \cdot x[n] + D \cdot u[n] \tag{2.5}$$

Per creare un sistema discreto con le 4 matrici della rappresentazione di stato è sufficiente dare il comando

```
-->gssz=syslin('d',A,B,C,D)
 gssz  =
```

```
        gssz(1)    (state-space system:)

!lss   A   B   C   D   X0   dt   !

        gssz(2) = A matrix =

    0.9951666     0.0950041
  - 0.0950041     0.9001625

        gssz(3) = B matrix =

    0.0048334
    0.0950041

        gssz(4) = C matrix =

    1.      0.

        gssz(5) = D matrix =

    0.

        gssz(6) = X0 (initial state) =

    0.
    0.

        gssz(7) = Time domain =

 d
```

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 $Y(z) = G(z) \cdot U(z)$.

```
-->sys=syslin('d',(z-1)/(z^2-2.3*z+0.8))
 sys  =

      - 1 + z
    ------------
                2
    0.8 - 2.3z + z
-->sys.dt=0.1;
```

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. Quando si passa alla rappresentazione di stato, Scilab fornisce una delle infinite possibili rappresentazioni.

2.3.1 Conversione di modelli

Le funzioni di conversione sono rappresentate nella tabella 2.1.

2.3.2 Discretizzazione

la funzione *dscr* permette di trasformare un sistema continuo nel suo equivalente sistema discreto campionato con ZOH e il tempo di campionamento $T_{sampling}$. Questa funzione restituisce sempre il sistema discreto nella forma dello spazio degli stati.

```
-->g=syslin('c',1/(s^2+s+1));

-->gz=dscr(g,0.01);

-->gz=ss2tf(gz)
 gz   =

       0.0000497 + 0.0000498z
    ---------------------------
                              2
    0.9900498 - 1.9899503z + z
```

2.3.3 Alcune utilities

Per poter accedere direttamente a singoli elementi di un sistema (per esempio ad una della 4 matrici, o al denominatore della funzione di trasferimento) si può utilizzare direttamente i campi della variabile che descrive il processo (per esempio *g.den*), oppure utilizzare uno dei comandi messi a disposizione de Scilab. Il comando *abcd* ad esempio estrae le 4 matrici della rappresentazione di stato da un sistema, indipendentemente dalla forma in cui è stato registrato.

```
-->gss.A
 ans  =

    0.      1.
  - 1.    - 1.

-->g.den
 ans  =

                2
    1 + s + s

-->>[A,B,C,D]=abcd(gss)
 D   =

    0.
 C   =

    1.      0.
 B   =

    0.
    1.
 A   =
```

```
    0.      1.
  - 1.    - 1.

-->[A,B,C,D]=abcd(g)
 D  =

    0.
 C  =

    1.      0.
 B  =

    0.
    1.
 A  =

    0.      1.
  - 1.    - 1.

-->g
 g  =

          1
    ----------
                2
    1 + s + s
```

2.3.4 Accoppiamento di sistemi

Sistemi in serie possono essere semplicemente moltiplicati tra di loro, mentre sistemi in parallelo vengono sommati. Per ottenere la funzione di trasferimento ad anello chiuso del sistema rappresentato in figura 2.1

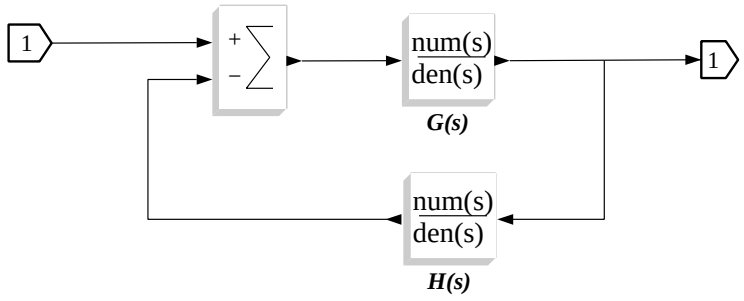

Figura 2.1: Sistema con feedback

possiamo dare i comandi seguenti:

```
-->g=syslin('c',1/(s^2+2*s+1));
-->h=syslin('c',(s+1)/(s+2));
gtot=g/.h;
```

Occore prestare attenzione al caso particolare di feedback unitario in cui

$$H(s) = 1$$

In questo caso occorre dare il comando

```
-->gtot=g/.(1)
```

in modo da non confondere il sistema con funzione 1 dallo scalare 0.1.

Capitolo 3

Metodi di analisi

Scilab mette a disposizione una serie di comandi per analizzare nel tempo e in frequenza un processo lineare di tipo LTI (Linear Time Invariant), sia SISO (Single Input Single Output) che MIMO (Multiple Input Multiple Output). Normalmente si può 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. Il sistema può essere rappresentato sia come funzione di trasferimento che con la rappresentazione di stato. Solo quest'ultima rappresentazione permette però di impostare delle condizioni iniziali particolari.
Consideriamo la funzione di trasferimento

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

```
-->g=syslin('c',(s+2)/(s^3+2*s^2+3*s+4))
 g  =

          2 + s
    ---------------
                2    3
    4 + 3s + 2s  + s

-->gz=ss2tf(dscr(g,0.5))
 gz  =

                                          2
    - 0.0429609 + 0.0735999z + 0.1178093z
    -------------------------------------
                                  2    3
  - 0.3678794 + 1.4142172z - 1.7494412z  + z
```

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

```
-->t=0:0.001:35;
-->y=csim('step',t,g);
```

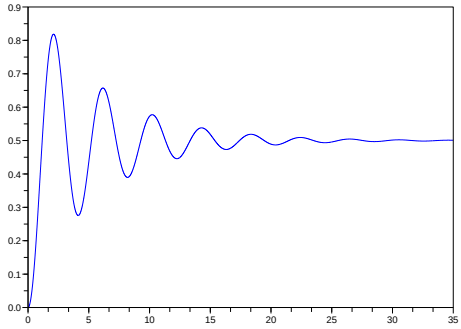

Figura 3.1: Risposta a gradino - Sistema continuo

```
-->t=0:0.5:35;
-->u=ones(1,71);
-->y=dsimul(tf2ss(gz),u);
```

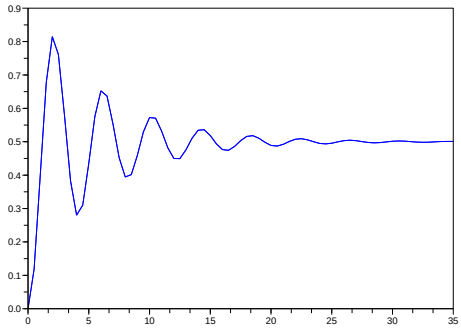

Figura 3.2: Risposta a gradino - Sistema discreto

```
-->t=0:0.001:35;
-->y=csim('impuls',t,g,[1;1;1]);
```

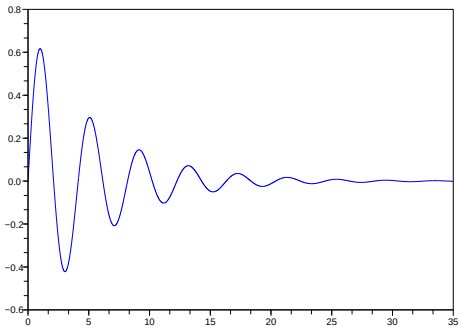

Figura 3.3: Risposta a impulso

```
-->t=0:0.001:20;
-->u=sin(t);
-->y=csim(u,t,g);
```

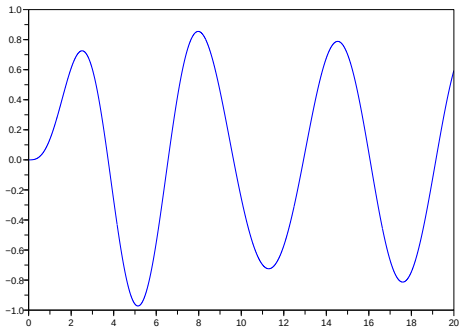

Figura 3.4: Risposta con entrata generica - Sistema continuo

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

csim	risposta con diverse entrate per un sistema continuo
dsimul	risposta con diverse entrate per un sistema discreto
ltitr	risposta degli stati di un sistema discreto
flts	filtraggio di dati discreti

Tabella 3.1: Funzioni su processi

3.2 Analisi in frequenza

Scilab mette a disposizione tutta una serie di comandi utili per il calcolo della risposta in frequenza.

repfreq calcolo della risposta in frequenza

```
[ [frq,] repf]=repfreq(sys,fmin,fmax [,step])
[ [frq,] repf]=repfreq(sys [,frq])
[ frq,repf,splitf]=repfreq(sys,fmin,fmax [,step])
[ frq,repf,splitf]=repfreq(sys [,frq]) Parameters
```

dbphi calcolo del quadagno (in dB) e della fase (in gradi)

```
[db,phi] =dbphi(repf)
```

g_margin margine di guadagno del sistema h

```
[gm [,fr]]=g_margin(h)
```

p_margin margine di fase del sistema h

```
[phm,fr]=p_margin(h)
phm=p_margin(h)
```

I principali grafici per l'analisi in frequenza sono rappresentati nelle figure 3.5, 3.6 e 3.7. Occorre prestare particolarmente attenzione al fatto che Scilab rappresenta il diagramma di Bode in funzione di una frequenza tarata in Hz e non in rad/s come altri sistemi (ad esempio MATLAB).

```
-->g=syslin('c',1/(s^2+s+1));
-->bode(g)
```

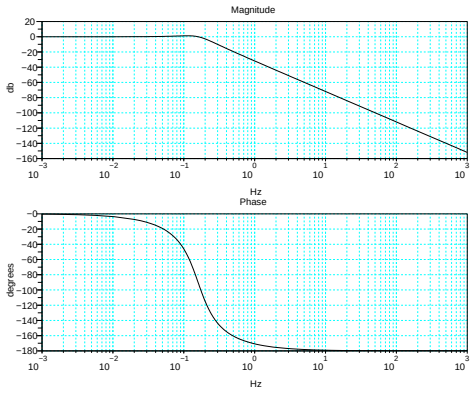

Figura 3.5: Diagramma di Bode

```
-->g=syslin('c',1/(s^2+2*s+1));
-->nyquist(g)
```

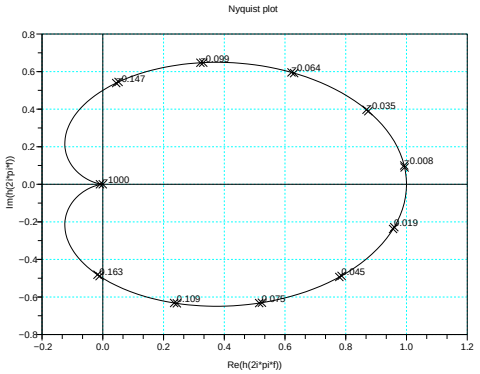

Figura 3.6: Diagramma di Nyquist

3.3 Analisi tramite le radici

I comandi messi a disposizione da Scilab per l'analisi con il luogo delle radici sono riportati nella tabella 3.2.

evans	diagramma del luogo delle radici
kpure	determina il guadagno necessario a portare i poli sull'asse immaginario (limite di guadagno)
krac2	guadagno necessario ad avere due poli reali uguali

Tabella 3.2: Comandi per il luogo delle radici

La figura 3.8 mostra il risultato del comando

```
-->g=syslin('c',1/(s^4+2*s^3+3*s^2+s));
-->black(g)
-->chart()
```

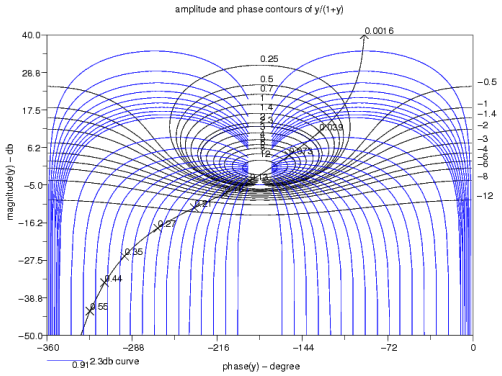

Figura 3.7: Diagramma di Nichols

```
-->g=syslin('c',(s+3)/(s^4+7*s^3+14*s^2+8*s));

-->evans(g)
```

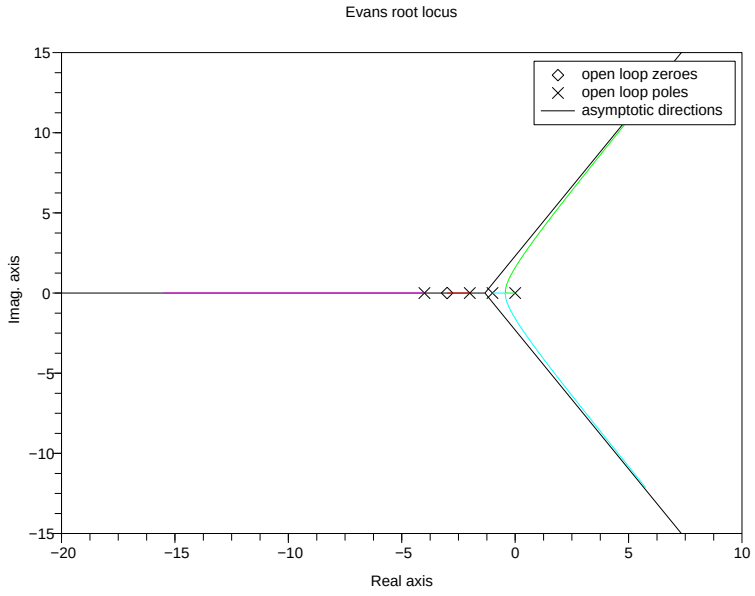

Figura 3.8: Grafico del luogo delle radici

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.
Analogamente si può determinare il diagramma del luogo delle radici per un sistema discreto.

```
-->g=syslin('c',1/(s^2+2*s+1));
-->gz=ss2tf(dscr(g,0.1));
```

```
-->evans(gz)
-->zgrid
```

Da questi comandi si ottiene il grafico della figura 3.9.

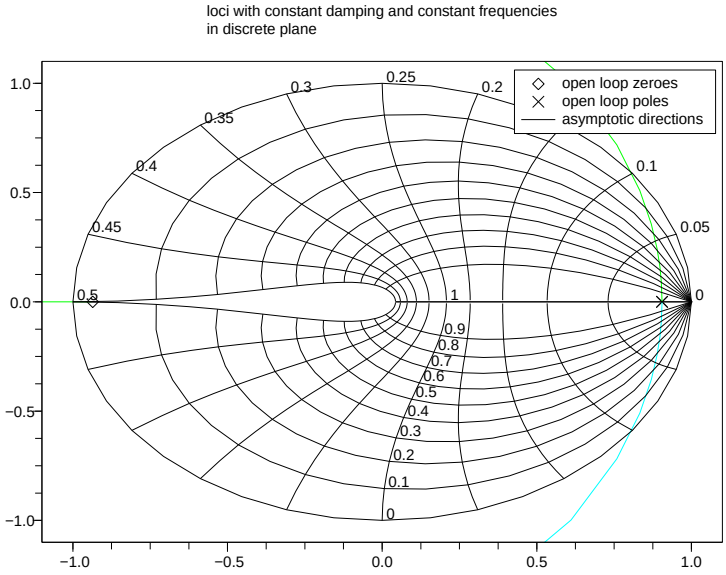

Figura 3.9: Grafico del luogo delle radici di un sistema discreto

3.4 Analisi di stabilità con Routh-Hurwitz

Scilab contiene un comando che permette di costruire automaticamente la tabella di Routh di un determinto polinomio, e di effettuare quindi anche l'analisi di stabilità utilizzando il criterio di Routh-Hurwitz.

```
-->g=syslin('c',1/((s+2)*(s-3)*(s+4)*(s+5)))
 g  =

                    1
    ------------------------------
                      2     3    4
  - 120 - 74s + 5s + 8s + s

-->routh_t(g.den)
 ans  =

    1.            5.      - 120.
    8.          - 74.       0.
    14.25       - 120.      0.
  - 6.6315789     0.        0.
  - 120.          0.        0.
```

Qui possiamo vedere che il sistema è instabile, essendoci un cambiamento di segno nella prima colonna della tabella, ciò che significa che il sistema ha un polo nel semipiano destro (polo a +3 nella definizione della funzione di trasferimento).
Nel caso avessimo la funzione di traferimento ad anello aperto, potremmo anche determinare la stabilità del sistema ad anello chiuso, in funzione di una certa amplificazione k. In questo caso la funzione *routh_t* viene chiamata passando un parametro supplementare k.

```
-->k=poly(0,'k')
 k  =

    k

-->g=syslin('c',1/(s^3+2*s^2+3*s))
 g  =

          1
    ------------
              2    3
    3s + 2s + s

-->routh_t(g,k)
 ans  =

    1             3

    2             k

    6 - k         0

             2
    6k - k        0
```

In questo caso possiamo determinare facilmente i valori di k che non fanno variare il segno nella 1. colonna della tabella ($0 < k < 6$).

Capitolo 4

Costruzione di sistemi

4.1 Connessioni

La costruzione di processi complessi con diverse funzioni di trasferimento, feedback, più entrate ed uscite, può essere fatta in modo molto semplice con Scicos. L'utilizzo di questo prodotto permette inoltre di modellare anche le diverse nonlinearità del processo.
Anche non avendo a disposizione Scicos, è sempre possibile modellare il processo all'interno di Scilab, sfruttando le proprietà degli oggetti LTI, mediante i comandi riportati nella tabella 4.1 (vedi anche sezione 2.3.4).

$S_1 + S_2$	Sistemi in parallelo
$S_1 \cdot S_2$	sistemi in serie
$S_1/.S_2$	Sistemi in feedback

Tabella 4.1: Comandi per interconnettere blocchi

Quale esempio consideriamo il sistema della figura 4.1.

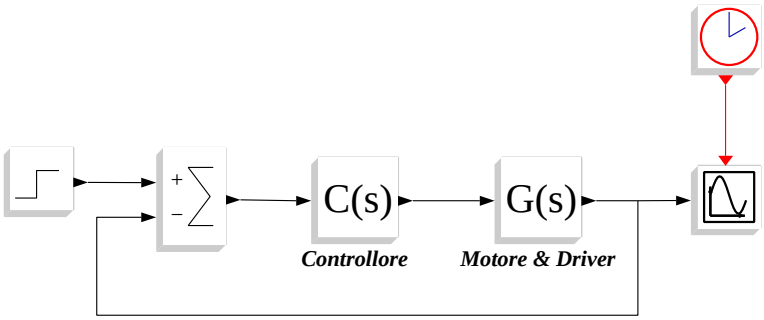

Figura 4.1: Processo e regolatore

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

```
-->g=syslin('c',6.63/(s^3+101.71*s^2+171*s))
```



```
 g  =

          6.63
    ------------------
                  2    3
    171s + 101.71s + s

-->c=syslin('c',(106.554*s+258)/(0.0518*s+1))
 c  =

    258 + 106.554s
    --------------
     1 + 0.0518s

-->gtot=c*g/.(1)
 gtot  =

                      1710.54 + 706.45302s
    ------------------------------------------------------
                                  2             3          4
    1710.54 + 877.45302s + 110.5678s + 6.268578s + 0.0518s

-->t=0:0.001:1.5;
-->y=csim('step',t,gtot);
-->plot(t,y),xgrid(4)
```

Il grafico della simulazione è riportato nella figura 4.2.

4.2 Riduzione

Scilab 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

Tabella 4.2: Comandi per la riduzione di un processo

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 principali sono *canon* e *ss2ss*.
Consideriamo il sistema

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

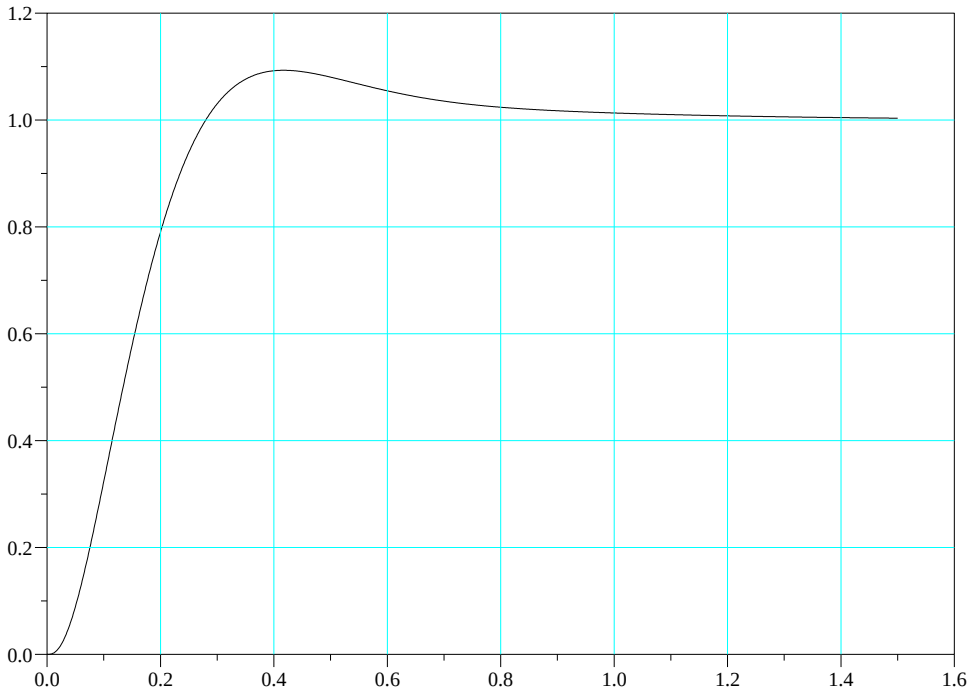

Figura 4.2: Risultato della simulazione


```
-->g=syslin('c',1/((s+1)*(s+2)*(s+3)))
 g  =

             1
    ----------------
                 2    3
    6 + 11s + 6s + s

-->gss=tf2ss(g);

-->canon(gss.A,gss.B)
 ans  =

  - 6.   - 11.   - 6.
    1.     0.     0.
    0.     1.     0.
```

Capitolo 5

Regolatori in cascata

5.1 Introduzione

La ricerca di una regolazione fatta secondo le regole classiche si applica normalmente a sistemi descritti mediante la funzione di trasferimento. Si può lavorare con metodi che operano sui poli, come pure con metodi basati sull'analisi in frequenza. La rappresentazione classica di un sistema retroazionato con regolatore è rappresentato nella figura 5.1.

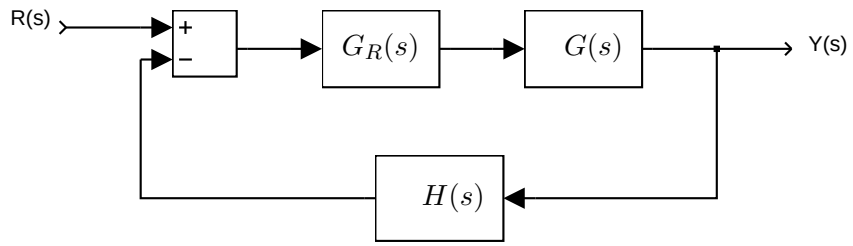

Figura 5.1: Sistema retroazionato

$G_R(s)$ rappresenta la funzione di trasferimento del regolatore che viene inserito nel circuito per migliorarne le caratteristiche.
Una regolazione è necessaria quando:

- Il sistema è instabile
- Il sistema è stabile ma il suo comportamento statico (errore allo stato finito) è insoddisfacente.
- Il sistema è stabile, l'errore allo stato finito è buono, il comportamento dinamico non è soddisfacente.
- Il sistema è stabile ma né il comportamento statico, né il comportamento dinamico sono soddisfacenti.

In tutti questi casi è necessario intervenire sul sistema per modificarne le caratteristiche in uscita.

5.2 Compensatori classici

Esistono diverse configurazioni che possono essere utilizzate per la compensazione. Esistono compensatori in serie, in feedback e combinazioni di entrambi. Il compensatore in serie è quello maggiormante utilizzato.
Una delle tecniche di compensazione è quella di cancellare poli del sistema mediante gli *zeri* del regolatore e sostituirli con i suoi poli. Questo metodo può essere utilizzato unicamente per modificare la posizione di poli che si trovano già nel semipiano sinistro, mai per eliminare quelli del semipiano destro. Il problema legato a questo metodo è che normalmente noi non sappiamo dove si trovano esattamente i poli del sistema, poiché noi conosciamo unicamente un modello di esso. Un altro problema è dato dal fatto che i poli del sistema, malgrado vengano eliminati nella funzione di trasferimento ad anello chiuso, restano attivi per quel che riguarda ad esempio i disturbi sul segnale di comando. Un uso di questa tecnica è possibile unicamente dopo un'accurata analisi di sensibilità che dia informazioni esatte sulla dipendenza dei risulatati rispetto alla variazione degli zeri e dei poli.
Le tecniche di compensazione più usate restano ancora il controllore *proporzionale-integrale-derivativo (PID)* e il compensatore *lead-lag*.
Le funzioni di trasferimento di queste funzioni sono riportate nella tabella 5.1

PID	$K_P + K_D s + \frac{K_I}{s}$	
Lead	$K\frac{s+a}{s+b}$	$0 < a < b$
Lag	$K\frac{s+a}{s+b}$	$a > b > 0$
Lead-Lag	Combinazione dei due compensatori precedenti	

Tabella 5.1: Regolatori classici

5.3 Controllore Proporzionale-Integrale-Derivativo (PID)

Ci sono diverse realizzazione del regolatore PID.

$$G_{PID} = K_P + K_D \cdot s + K_I \frac{1}{s}$$

oppure

$$G_{PID} = K_P(1 + \frac{1}{s \cdot T_I} + T_D \cdot s)$$

A seconda di quali parametri vengono impostati si possono avere diverse varianti di questo controllore. Nela caso della prima rappresentazione si può realizzare un controllore proporzionale P ($K_I = 0$, $K_D = 0$), proporzionale- derivativo PD ($K_I = 0$), proporzionale-integrale PI ($K_D = 0$) o proporzionale-integrale-derivativo PID.
Il regolatore proporzionale è il più semplice; esso permette però unicamente di soddisfare al massimo una specifica di progetto ad anello chiuso (p.es. GM, PM, e_∞ ecc.). L'aggiunta di una parte derivativa aumenta lo smorzamento del sistema ad anello chiuso, mentre la parte integrale aumenta il tipo del sistema e di conseguenza migliora l'errore allo stato finito. Con un controllore di tipo PID è possibile rispettare più specifiche di design. È il sistema più usato industrialmente ed esiste in diverse forme (analogico, digitale e adattivo). Per il calcolo dei vari parametri esistono diversi metodi, empirici, analitici e basati sull' analisi dei poli o in frequenza.

5.3.1 Metodi euristici (Metodi di Ziegler-Nichols)

5.3.1.1 Regolatore P, PI, PID dall'amplificazione limite di stabilità

Per utilizzare questo metodo occorre conoscere l'amplificazione limite per la stabilità e la freuenza di oscillazione del sistema con questo valore. (vedi Tabella 5.2).

	K_P	K_I	K_D
P	$0.5K_m$		
PI	$0.45K_m$	$\frac{K_P\omega_m}{1.66\pi}$	
PID	$0.6K_m$	$\frac{K_P\omega_m}{\pi}$	$\frac{K_P\pi}{4\omega_m}$

Tabella 5.2: Parametri di Ziegler-Nichols

K_m e ω_m possono essere determinati sperimentalmente, oppure determinati dal luogo delle radici, il diagramma di Bode, o tramite la tabella di Routh- Hurwitz. Nel luogo delle radici il valore di K per il quale il grafico taglia l'asse immaginario è uguale a K_m, mentre il valore sull'asse immaginario equivale a ω_m. Se invece utilizziamo il diagramma di Bode otteniamo direttamente K_m dal margine di guadagno e ω_m dalla frequenza in cui la fase taglia il grafico della fase a $-180°$.
Si tratta di un metodo puramente empirico, largamente usato industrialmente.
Prendiamo un processo con funzione di trasferimento pari a

$$G(s) = \frac{V_{out}}{V_{in}} = \frac{6.63K}{s(s+1.71)(s+100)} \tag{5.1}$$

Utilizzando la funzione *kpure* (vedi tabella 3.2) possiamo determinare subito il guadagno limite che porta il sistema ad anello chiuso in oscillazione.

```
-->g=syslin('c',6.63/(s*(s+1.71)*(s+100)))
 g  =

          6.63
    ---------------------
                  2    3
    171s + 101.71s + s

-->K=kpure(g)
 K  =

    2623.2896     0.

-->hf=g/.K(1);

-->roots(hf.den)
 ans  =

    5.011E-17 + 13.076697i
    5.011E-17 - 13.076697i
  - 101.71
```

Le radici sull'asse immaginario sono a $\pm 13.08j$.
Il calcolo dei parametri fornisce il risultato seguente

$$K_P = 1573.97 \simeq 1574 \tag{5.2}$$
$$K_D = 94.51 \simeq 95 \tag{5.3}$$
$$K_I = 6553.23 \simeq 6553 \tag{5.4}$$

La simulazione del sistema con questo regolatore PID è riportata nella figura 5.2.
Possiamo vedere che il tempo di setting T_S è notevolmente migliorato, ma abbiamo un $\%OS$ decisamente elevato. Inoltre il sistema oscilla diverse volte prima di stabilizzarsi all'equilibrio.

5.3.1.2 Regolatore P, PI, PID dopo approssimazione del processo

Una grande quantità di processi industriali non sono rappresentabili con un modello matematico. Può essere sufficiente avere a disposizione un'approssimazione del processo che ci permette in seguito di costruire un regolatore semplice. Il metodo di Ziegler-Nichols in questo caso parte dalla risposta ad anello aperto del processo ad un'entrata gradino e produce un modello semplice del processo. In seguito esistono formule che permettono di determinare velocemente i parametri per il regolatore PID.
Partendo dalla risposta del sistema ad anello aperto ad un'entrata gradino unitario, si ottengono in genere due tipi di curve (figura 5.3 e figura 5.4).

$$G(s) = \frac{Ke^{-sT_1}}{1 + sT_2} \tag{5.5}$$

mentre la seconda ha una funzione di trasferimento approssimata pari a

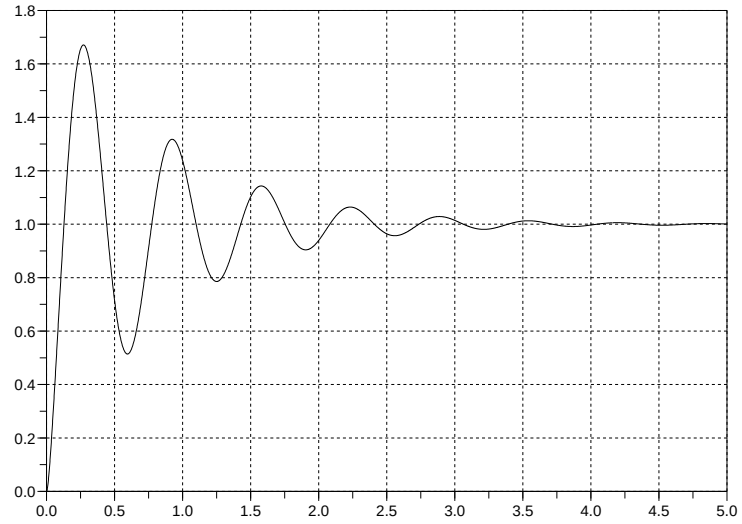

Figura 5.2: Risposta al gradino con PID empirico

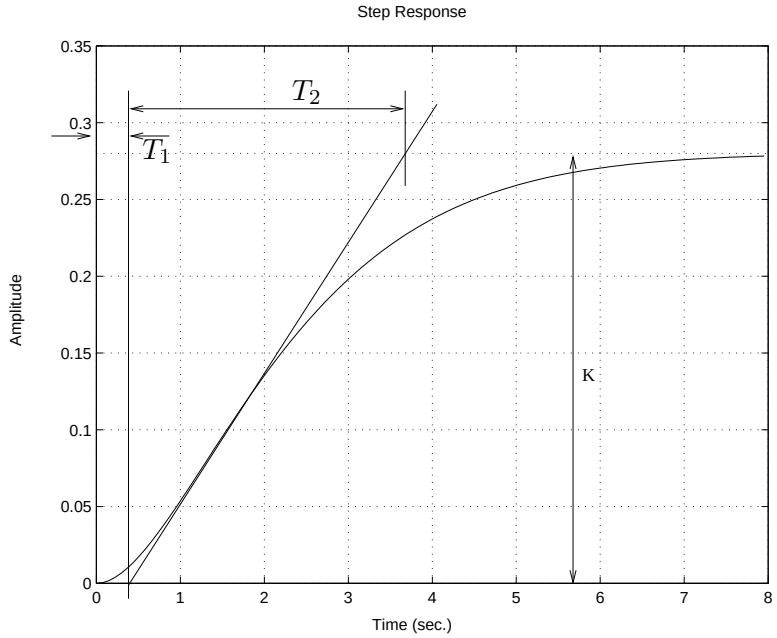

Figura 5.3: Approssimazione di un sistema di tipo 0

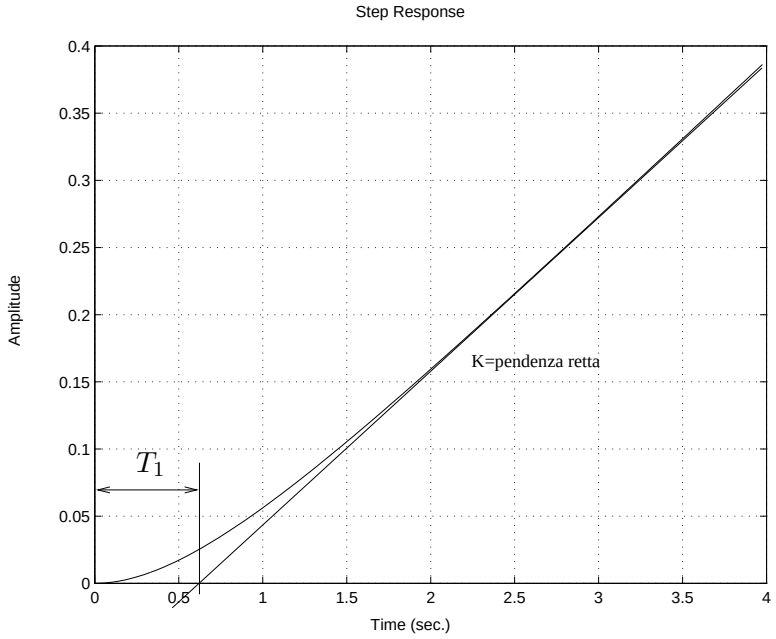

Figura 5.4: Approssimazione di un sistema di tipo 1

$$G(s) = \frac{Ke^{-sT_1}}{s} \tag{5.6}$$

Se applichiamo le regole di Ziegler-Nichols al primo sistema otteniamo un regolatore PID con la funzione di trasferimento definita da

$$G_{PID} = K_c \left(1 + \frac{1}{T_I s} + T_D s\right) \tag{5.7}$$

con

$$K_c = \frac{1.2T_2}{KT_1} \tag{5.8}$$
$$T_I = 2T_1 \tag{5.9}$$
$$T_D = 0.5T_1 \tag{5.10}$$

Nel caso ad esempio di un regolatore PI abbiamo

$$G_{PI} = K_c \left(1 + \frac{1}{T_I s}\right) \tag{5.11}$$

con

$$K_c = \frac{0.9T_2}{KT_1} \tag{5.12}$$
$$T_I = 2T_1 \tag{5.13}$$

Infine con un regolatore P otteniamo

$$K_c = \frac{T_2}{KT_1} \tag{5.14}$$

5.3.2 Metodo grafico

Con i metodi grafici possiamo analizzare e capire l'influsso che i vari regolatori hanno sulla rappresentazione del diagramma di Bode ad anello aperto, e capire subito in che modo influenzeranno la risposta del sistema ad anello chiuso.

5.3.2.1 Controllore P

Con un controllore di tipo proporzionale si possono corregere solo singole specifiche di progetto. L'errore allo stato finito impone spesso valori di amplificazione che determinano delle sovraelongazioni elevate o, spesso, anche instabilità nel sistema. Con il valore di K_P si può influenzare il punto di ω_{gc}, con conseguente $T_{setting}$ e e_∞ spesso imposti.
Prendiamo ad esempio un processo con funzione di trasferimento pari a

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

Si vuole avere un errore allo stato finito pari al 5% e un PM di ca. 60^o
L'analisi dell'errore allo stato finito porta all'equazione

$$e_\infty = \lim_{s\to 0} \frac{1}{1+G(s)} = 0.05 = \frac{1}{1+\frac{K}{5}}$$

da cui si ricava che K vale 95, che equivalgono a ca. 40dB. Per questo valore di K il margine di fase risulta essere

```
-->180+p_margin(95*g)
 ans  =

    35.067815
```

valore che può anche essere letto dal diagramma di bode della figura 5.5 e che è insufficiente per rispettare le specifiche.

5.3.2.2 Controllore PI

La funzione di trasferimento di un controllore PI può essere descritta come

$$G_{PI}(s) = K_P(1 + \frac{1}{sT_I}) = K_P\frac{1+sT_I}{sT_I}$$

Il diagramma di Bode di questa funzione di trasferimento è mostrato nella figura 5.6.
Si può subito vedere come con questo regolatore non sia possibile migliorare l'andamento della fase del sistema compensato. È unicamente possibile correggere l'errore allo stato finito, grazie all'aumento del tipo del sistema ad anello aperto, e scegliere in seguito l'amplificazione K_P per impostare il valore di ω_{gc} dove si ha il margine di fase desiderato.
Per costruire il regolatore si può procedere nel modo seguente

- Disegnare il diagramma di Bode del processo da compensare
- Determinare la frequenza dove si ha il margine di fase desiderato con 5 gradi supplementari.

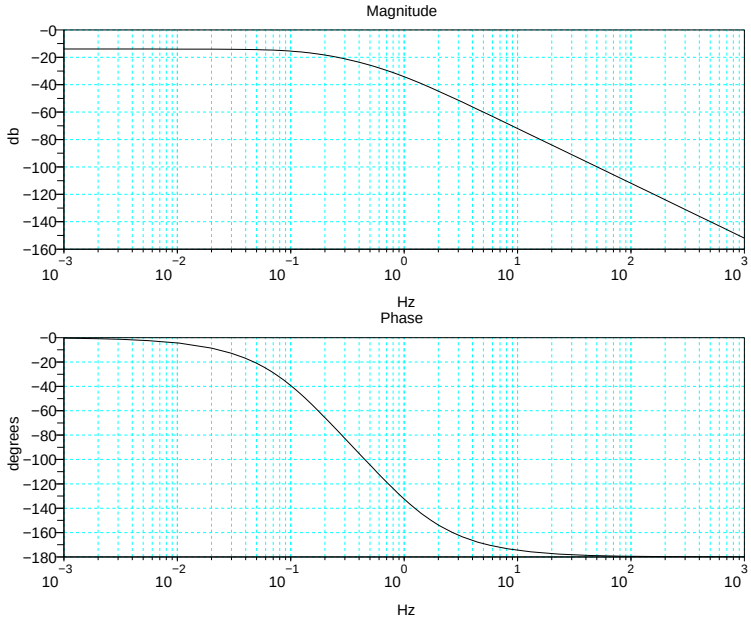

Figura 5.5: Diagramma di Bode del sistema

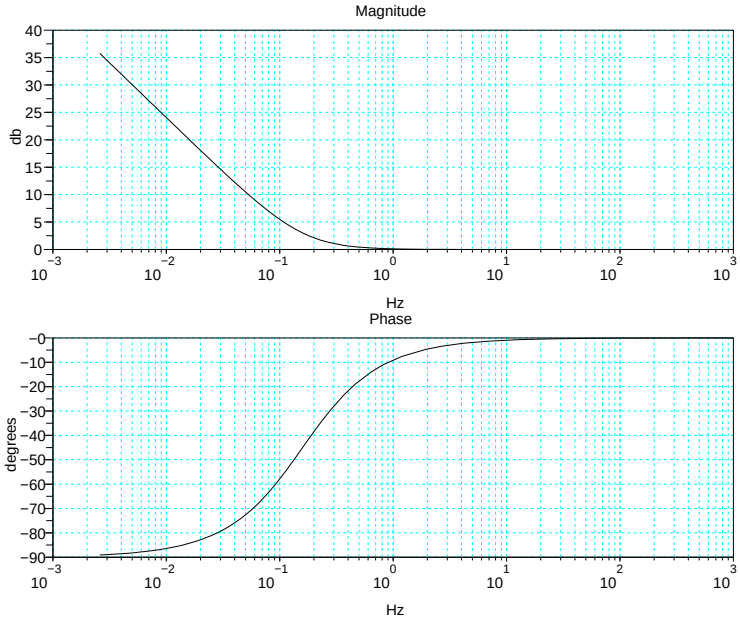

Figura 5.6: Diagramma di Bode di un regolatore PI con $T_I = 1$ e $K_P = 1$

- Per non influenzare l'andamento della fase del sistema compensato, la frequenza del regolatore PI pari a $1/T_I$ va scelta almeno 10 volte più piccola della frequenza (in radianti) trovata precedentemente.
- Disegnare il diagramma di Bode del sistema compensato con $K_P = 1$.
- Determinare la frequenza in cui si avrebbe il PM desiderato, determinare l'ampiezza in dB di questo punto e calcolare l'amplificazione necessaria per portare questo punto a 0dB. Questo valore equivale a K_P.
- Controllare il sistema compensato (diagramma di Bode, risposta nel tempo).

Per il sistema con funzione di trasferimento pari a

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

vediamo che con un regolatore di tipo PI l'errore allo stato finito viene annullato. Le specifiche statiche sono quindi rispettate. Dal diagramma di Bode del sistema non compensato (figura 5.5) si può vedere come il margina di fase di $60^o + 5^o$ sia presente alla frequenza di

```
-->f=bode2freq(g,-115,0.1,10,'phi')
 f  =

    0.6424341
```

e conseguentemente il valore di T_I vale

```
-->Ti=10/(2*pi*f)
 Ti  =

    2.477374
```

Ora si può disegnare il diagramma di Bode del sistema compensato con $K_P = 1$.
Dal nuovo diagramma di Bode si ricava la frequenza in cui si ha il margine di fase desiderato di 60^o, che corrisponde a

```
-->gr=syslin('c',(1+s*Ti)/(s*Ti))
 gr  =

    1 + 2.4773737s
    --------------
      2.4773737s

-->bode(gr*g)

-->bode(gr*g)

-->f=bode2freq(g,-120,0.1,10,'phi')
 f  =

    0.7258226
```

A questo punto occorre determinare il guadagno necessario a portare questa frequenza a $0dB$.

```
-->f=bode2freq(gr*g,-120,0.1,10,'phi')
 f  =

    0.6297755

-->repf=repfreq(g*gr,f);

-->[db,phi]=dbphi(repf);

-->db
 db  =

  - 28.262663
```

$$K_P = 10^{\frac{28}{20}} \simeq 25$$

La figura 5.7 mostra la risposta nel tempo del processo compensato.

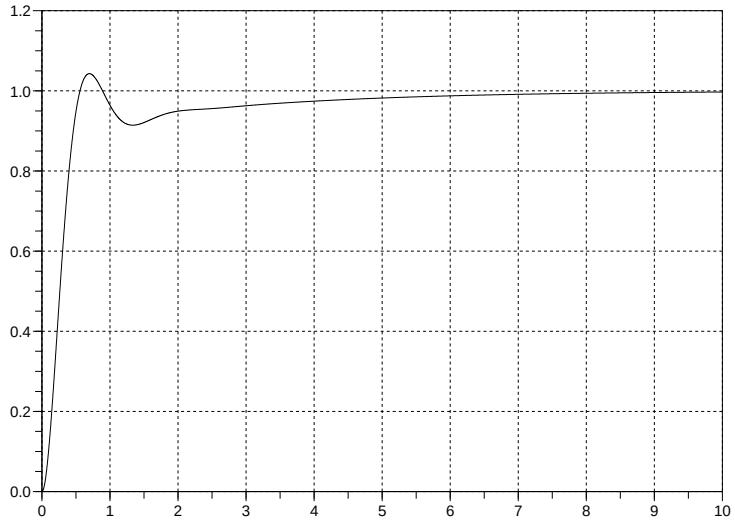

Figura 5.7: Risposta nel tempo del sistema compensato

5.3.3 Regolatore lead dal diagramma di Bode

L'idea di base nel design attraverso Bode è quella di portare la funzione di trasferimento ad anello aperto ad avere il guadagno in bassa frequenza desiderato (per es. in funzione di un certo errore statico) una certa frequenza di gain-crossover ω_{gc} (per la velocità della risposta) e un'adeguata stabilità attraverso il margine di fase PM. La parte derivativa del regolatore PID non dovrebbe essere realizzata tramite un derivatore puro, ma viene realizzata tramite un regolatore lead (rete anticipatrice).
La funzione di trasferimento di un regolatore di tipo lead vale

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

con $\alpha > 1$.
Dapprima viene scelto il valore di K_c che soddisfa le specifiche statiche. Poi si cerca di soddisfare il PM alla frequenza di taglio desiderata. Per fare questo dobbiamo iniziare a conoscere meglio la funzione di trasferimento del regolatore con il guadagno $K_c = 1$.
Il diagramma di Bode del regolatore con guadagno unitario e $\alpha = 10$ è rappresentato nella figura 5.8.

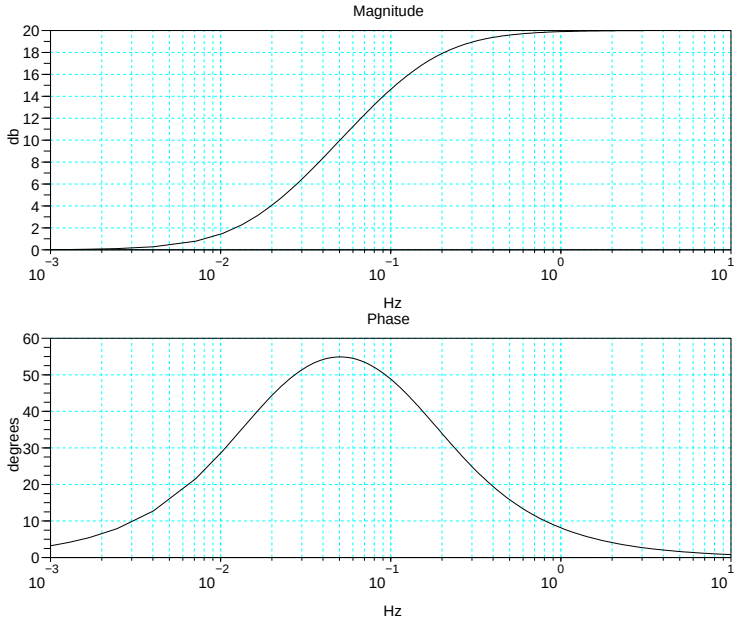

Figura 5.8: Diagramma di Bode di un regolatore lead

La fase supplementare fornita dal compensatore è data da

$$\Phi = \angle G_{R1}(j\omega) = \tan^{-1} \alpha T\omega - \tan^{-1} T\omega \tag{5.15}$$

da cui si ricava

$$\sin \Phi = \frac{\alpha - 1}{\sqrt{\alpha^2 + 2\alpha + 1}} = \frac{\alpha - 1}{\alpha + 1} \tag{5.16}$$

o meglio ancora

$$\alpha = \frac{1 + \sin \Phi}{1 - \sin \Phi} \tag{5.17}$$

Applicando le formule trovate possiamo determinare i parametri del compensatore con i passi seguenti

1. Scegliere K_c per soddisfare l'errore statico.
2. Disegnare il diagramma di Bode di $K_c G(j\omega)$ e determinare il PM.

3. Determinare la fase supplementare necessaria.
4. Aggiungere ca $5^o - 15^o$ alla fase per trovare il valore di Φ
5. Calcolare $\alpha = \frac{1+\sin\Phi}{1-\sin\Phi}$.
6. Trovare la frequenza alla quale il guadagno di $K_c G(j\omega) = -10\log\alpha$. Questa frequenza sarà la nuova ω_{gc} del sistema compensato.
7. Determinare $T = \frac{1}{\sqrt{\alpha}\omega_{gc}}$.
8. Disegnare il diagramma di Bode completo per controllare il design.
9. Determinare la risposta ad anello chiuso del sistema compensato.

Da notare che con questo metodo non possiamo scegliere anticipatamente il valore di ω_{gc}. Se otteniamo un valore inaccettabile per la frequenza di gain-crossover, dobbiamo modificare il PM o l'errore statico.
Vediamo ora di sviluppare un regolatore lead per il processo descritto dalla funzione di trasferimento

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

Si vuole avere un errore allo stato finito pari al 5% e un PM di ca. 60^o
L'analisi dell'errore allo stato finito porta all'equazione

$$e_\infty = \lim_{s\to 0} \frac{1}{1+G(s)} = 0.05 = \frac{1}{1+\frac{K}{5}}$$

da cui si ricava che K vale 95, che equivalgono a ca. 40dB. Per questo valore di K il margine di fase risulta essere

```
-->[phase,frq]=p_margin(95*g)
 frq  =

    1.447828
 phase  =

  - 144.93218

-->PM=180+phase
 PM  =

    35.067815
```

La frequenza *frq* rappresenta la frequenza minima di taglio a $0dB$ per il sistema compensato.
Il PM è insufficiente e per arrivare almeno ad avere 60^o occorre aggiungere 25^o, che con il margine di sicurezza diventano $25^o + 15^o = 40^o$.
Il valore di α diventa quindi

```
-->Phi=40/180*pi;

-->alfa=(1+sin(Phi))/(1-sin(Phi))
 alfa  =

    4.5989099
```

Dobbiamo ora determinare il punto del diagramma di Bode che al momento con solo il gudagno K_c si trova a $-10 \log \alpha$.

```
-->val=-10*log10(alfa)
 val  =

  - 6.626549

-->f=bode2freq(95*g,val,1,10,'db')
 f  =

    2.2000329
```

Possiamo subito controllare se a questa frequenza (che sarà la nuova frequenza di gain-crossover) l'aggiunta di 35^o è sufficiente per avere alla fine il PM desiderato:

```
-->rep=repfreq(95*g,f)
 rep  =

  - 0.4259162 - 0.1898374i

-->[db,phi]=dbphi(rep)
 phi  =

  - 155.97673
 db  =

  - 6.626549
```

Il PM finale sarà di ca. 64^o, rispettando le specifiche.
Scilab ci fornisce la costante di tempo del regolatore lead

```
-->Tl=1/(2*pi*f*sqrt(alfa))
 Tl  =

    0.0337337
```

Possiamo costruire il regolatore lead e controllare se il margine di fase desiderato è rispettato

```
-->Gl=syslin('c',95*(1+alfa*s*Tl)/(1+s*Tl))
 Gl  =

    95 + 14.738117s
    ---------------
    1 + 0.0337337s

-->[phase,f]=p_margin(Gl*g)
 f  =

    2.2000329
```

```
 phase  =

  - 115.97673

-->PM=180+phase
 PM  =

    64.02327
```

La figura 5.9 mostra il diagramma di Bode del sistema compensato con il lead e solo proporzionale, mentre la figura 5.10 mostra la risposta delle due compensazioni.

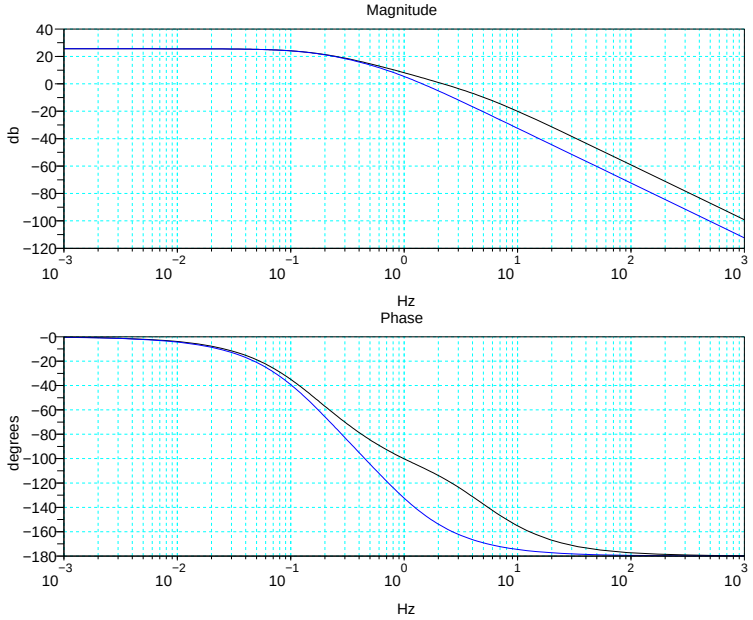

Figura 5.9: Diagramma di Bode con e senza lead

5.4 Regolatore PID reale

Il regolatore con il PID reale può essere realizzato mettendo in serie un regolatore PI e un regolatore lead.

$$G_{PID} = K_c \cdot \frac{1 + s \cdot T_I}{s \cdot T_I} \cdot \frac{1 + \alpha \cdot s \cdot T_{lead}}{1 + s \cdot T_{lead}}$$

Una volta scelta la larghezza di banda che si vuole ottenere è sufficiente eseguire i passi seguenti:

1. Scegliere la costante di tempo della parte PI del regolatore in modo da non peggiorare in modo irrcuperabile il PM del sistema alla frequenza di gain-crossover scelta. Occorre tenere presente che la parte lead può fornire un supplemento di fase $< 90^o$!

2. Costruire il regolatore PI e determinare la fase del sistema con questo compensatore, alla frequenza di gain-crossover scelta.

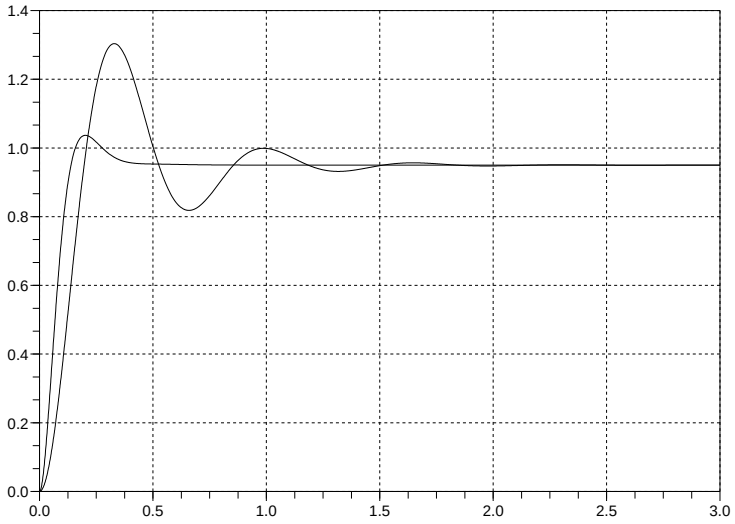

Figura 5.10: Risposta a gradino con e senza lead

3. Costruire la parte lead (α e T_{lead}) in modo da fornire tutta la fase mancante alla frequenza di gain-crossover scelta.
   - $\alpha = \frac{1+\sin\Phi}{1-\sin\Phi}$
   - $T_{lead} = \frac{1}{\sqrt{\alpha}\cdot\omega_{gc}}$ [Attenzione: ω_{gc} in rad/s!]
4. Determinare il valore dell'amplificazione K_c in modo da portare il guadagno alla frequenza di gain-crossover scelta a $0dB$.

Per il sistema descritto dalla funzione di trasferimento

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

vogliamo costruire un regolatore PID che permetta di ottenere un $PM \geq 60^o$ e una larghezza di banda del sistema ad anello chiuso di $4Hz$.
Seguendo lo schema precedente possiamo iniziare a costruire la parte PI del regolatore

```
-->g=syslin('c',1/((s+1)*(s+5)))
 g  =

          1
    ----------
              2
    5 + 6s + s

-->Ti=10/(4*2*pi)
 Ti  =

    0.3978874
```

```
-->Gpi=syslin('c',(1+s*Ti)/(s*Ti))
 Gpi  =

    1 + 0.3978874s
    --------------
      0.3978874s
```

Occorre ora determinare il PM a $4Hz$.

```
-->repf=repfreq(Gpi*g,4)
 repf  =

  - 0.0015447 - 0.0002121i

-->[db,phi]=dbphi(repf)
 phi  =

  - 172.18034
 db  =

  - 56.141824

-->PM=180+phi
 PM  =

    7.8196573
```

Si può vedere come questo punto abbia momentaneamente $PM = 7.8^o$. Per arrivare ad avere almeno un $PM \geq 60^o$ occorre fornire 53^o supplementari tramite la parte lead.

```
-->Phi=53/180*pi
 Phi  =

    0.9250245

-->alfa=(1+sin(Phi))/(1-sin(Phi))
 alfa  =

    8.9322378

-->Tl=1/(2*pi*4*sqrt(alfa))
 Tl  =

    0.0133131

-->Gl=syslin('c',(1+alfa*s*Tl)/(1+s*Tl))
 Gl  =

    1 + 0.118916s
    -------------
    1 + 0.0133131s
```

Al momento il guadagno del regolatore vale 1. Possiamo scegliere noi il valore di K_c in modo da portare il guadagno a $4Hz$ a $0dB$.

```
-->repf=repfreq(Gl*Gpi*g,4)
 repf  =
```

```
  - 0.0022720 - 0.0040686i

-->[db, phi]=dbphi(repf)
 phi  =

  - 119.18034
 db  =

  - 46.632221

-->Kc=10^(-db/20)
 Kc  =

    214.59078
```

Determiniamo ora se il PM soddisfa le specifiche.

```
-->[phase, frq]=p_margin(Kc*Gl*Gpi*g)
 frq  =

    4.
 phase  =

  - 119.18034

-->PM=180+phase
 PM  =

    60.819657
```

Le specifiche sono rispettate e possiamo vedere la risposta al gradino del sistema compensato (figura 5.11).

```
-->gt=Kc*Gl*Gpi*g/.(1)
 gt  =

                                                    2
                  214.59078 + 110.90124s + 10.1534s
    ------------------------------------------------------------------
                                        2         3          4
    214.59078 + 112.89067s + 12.567209s + 0.4296701s + 0.0052971s

-->bb_step(gt,2)
```

5.5 Regolatori nel discreto

La simulazione ed analisi nel discreto è analoga a quella nel continuo.
Con Scicos, 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.

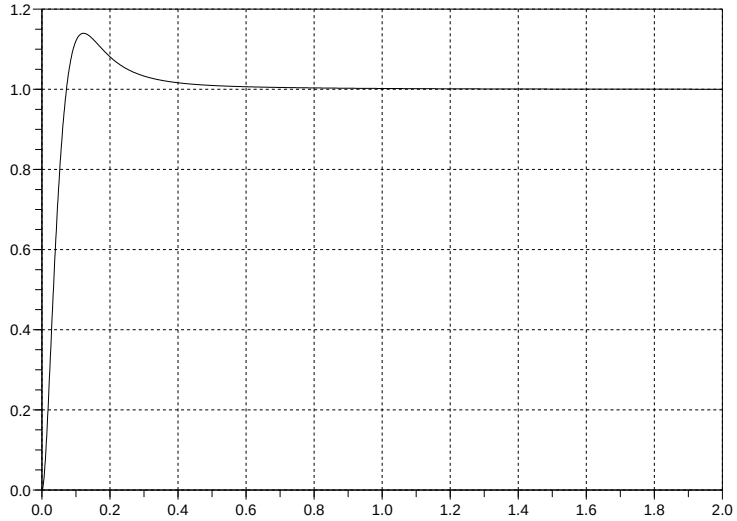

Figura 5.11: Risposta a gradino con regolatore PID

Capitolo 6

Regolazione nello spazio degli stati

6.1 Introduzione

La regolazione nello spazio degli stati permette l'analisi e il design di sistemi non lineari e di sistemi con entrate e uscite multiple. Il vantaggio principale del controllo nello spazio degli stati è quello di avere a disposizione un numero di parametri pari all'ordine del processo da controllare, parametri che possono essere utilizzati per modificare il comportamento del sistema. Se con un regolatore in cascata si cerca di *imitare* il comportamento di un sistema di 2. ordine, nel piano degli stati possiamo andare a posizionare tutti i *poli* in posizioni desiderate, a condizione che siano esauditi certi requisiti (controllabilità). Questo risultato viene raggiunto moltiplicando ogni stato per un certo guadagno, e riportando questo segnale all'entrata del processo. Con il controllore di stato non è però possibile influenzare la posizione degli *zeri* del sistema globale. I guadagni dei feedback degli stati possono essere calcolati utilizzando diverse metodologie:

- Piazzamento dei poli in posizioni desiderate (pole placement)
- Minimizzazione di una funzione di costo (sistemi LQR)

6.2 Controllabilità e osservabilità

Il controllo di stato è possibile unicamente se tutte le grandezze di stato possono essere controllate dal segnale in entata $u(t)$. Questa caratteristica è chiamata controllabilità e può essere testata tramite una specifica matrice chiamata matrice di controllabilità, il cui rango deve essere uguale al numero di stati del sistema.
Scilab mette a disposizione alcune funzioni che permettono di analizzare la controllabilità di un sistema. (vedi tabella 6.1).

cont_mat	matrice di controllabilità
contr	controllabilità e sottospazi controllabili

Tabella 6.1: Funzioni per controllabilità

6.3 Piazzamento dei poli

Tramite funzioni messe a disposizione da Scilab, il piazzamento dei poli di un sistema controllabile risulta molto semplice.
Partendo da un sistema descritto da

$$\dot{x} = A \cdot x + B \cdot u$$

possiamo quindi sostituire l'entrata $u(t)$ con

$$u = rif - K \cdot x$$

Conseguentemente la nuova equazione differenziale del sistema diventa

$$\dot{x} = A \cdot x + B \cdot (rif - K \cdot x) = (A - B \cdot K) \cdot x + B \cdot rif$$

I poli del sistema, coincidenti con gli autovalori della matrice $(A - B \cdot K)$, possono quindi essere portati in una qualsiasi posizione, scegliendo in modo appropriato i valori della matrice di feedback K.
Nei casi reali, la difficoltà maggiore sta nel riuscire ad *estrarre* tutti gli stati del sistema per poter effettuare questo feedback. Poiché non è possibile, oppure risulta troppo oneroso, misurare tutti gli stati, viene realizzato un osservatore, che permette di stimare gli stati del sistema ed usare questi valori per effettuare il feedback. La condizione per poter effettuare un controllore di stato è che il sistema sia *controllabile*, mentre per realizzare un osservatore occorre che il sistema sia *osservabile*.
L'utilizzo di un ambiente quale Scilab permette di sviluppare velocemente controllori di stato. Nell'ambiente SCICOS è inoltre possibile effettuare un test completo integrando anche la parte di osservatore (completo o ridotto).
Viene data la funzione di trasferimento

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

Determinare il feedback degli stati che dia $\%OS = 9.48\%$ e $T_{Setting} = 0.74s$. Utilizziamo la funzione supplementare*init_par* per determinare la posizione dei due poi dominanti di un sistema con le caratteristiche date.

```
-->g=syslin('c',20*(s+5)/(s^3+5*s^2+4*s))
 g  =

     100 + 20s
    ------------
          2    3
    4s + 5s + s

-->[xi,wn,p]=init_par(9.48,0.74)
 p  =

  - 5.5880198 + 7.4513529i
 wn  =

    9.3138942
 xi  =

    0.599966
```

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

```
-->poli=[-5,p,conj(p)]
 poli  =

  - 5.  - 5.5880198 + 7.4513529i  - 5.5880198 - 7.4513529i
```

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

```
-->[K]=ppol(gss.A,gss.B,poli)
 K  =

  - 41.308092  - 4.7542289     0.0715565
```

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

```
-->g2=syslin('c',gss.A-gss.B*K,gss.B,gss.C,gss.D);

-->bb_step(g2,1)
```

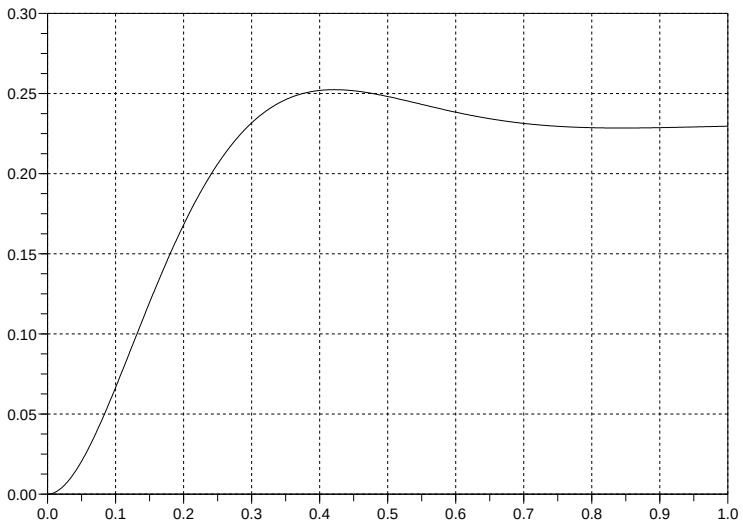

Figura 6.1: Simulazione del sistema con feedback degli stati

6.4 Osservatore

Con un controllore di stato è possibile modificare il comportamento di un sistema riportando in entrata le variabili di stato moltiplicate per un certo fattore di guadagno. Non sempre però queste variabili possono essere misurate, o perchè fisicamente non raggiungibili, o magari perché la scelta di un sensore appropriato sarebbe troppo costosa. In

questo caso è possibile *stimare* i valori della variabili di stato ed utilizzare questo valore per il feedback del controllo. La stima di questi valori viene fatta utilizzando un modello matematico del sistema da controllare che riceve in entrata lo stesso valore di controllo $u(t)$.
Partendo da un processo descritto da

$$\begin{cases} \dot{x} &= A \cdot x &+B \cdot u \\ y &= C \cdot x &+D \cdot u \end{cases} \tag{6.1}$$

e assumendo un modello perfetto, l'osservatore avrebbe la forma

$$\begin{cases} \dot{\hat{x}} &= A \cdot \hat{x} &+B \cdot u \\ \hat{y} &= C \cdot \hat{x} &+D \cdot u \end{cases}$$

Siccome possono esserci differenze tra i due valori in uscita (dovuti ad esempio a condizioni iniziali diverse), si cerca di portare a 0 il più svelto possibile questa differenza, sfuttando l'errore tra le due uscite e riportandolo in entrata sugli stati moltiplicandolo per guadagni specifici (matrice L).
Pertanto il modello dell'osservatore diventa

$$\begin{cases} \dot{\hat{x}} &= A \cdot \hat{x} + B \cdot u + L \cdot (y - \hat{y}) \\ \hat{y} &= C \cdot \hat{x} + D \cdot u \end{cases} \tag{6.2}$$

Sottraendo l'equazione 6.2 all'equazione 6.1 otteniamo

$$\begin{cases} \dot{x} - \dot{\hat{x}} &= A \cdot (x - \hat{x}) - L \cdot (y - \hat{y}) \\ y - \hat{y} &= C \cdot (x - \hat{x}) \end{cases}$$

che diventa in seguito

$$\begin{cases} \dot{x} - \dot{\hat{x}} &= A \cdot (x - \hat{x}) - L \cdot C \cdot (x - \hat{x}) \\ y - \hat{y} &= C \cdot (x - \hat{x}) \end{cases}$$

Sostituendo ora $e_y = y - \hat{y}$ e $e_x = x - \hat{x}$ arriviamo ad ottenere

$$\begin{cases} \dot{e_x} &= (A - L \cdot C) \cdot e_x \\ e_y &= C \cdot e_x \end{cases}$$

Quest'ultima è una equazione omogenea che possiamo far tendere a 0 il più velocemente possibile, piazzando in modo appropriato gli autovalori della matrice $A - L \cdot C$. In genere questo risultato viene raggiunto posizionando questi poli molto più a sinistra rispetto ai poli del sistema controllato.
Il comando

```
-->L_ppol = ppol(A',C',poli_obs);
-->L = L_ppol';
```

permette di determinare la matrice L di feedback per l'osservatore.
Siccome alcuni stati sono conosciuti o possono essere calcolati dai valori in uscita del processo da controllare, è possibile creare un osservatore che in uscita ha gli stati misurati e la stima unicamente degli stati mancanti. Questo osservatore è detto *osservatore ridotto*.

6.5 Controllori con osservatore e integratore

Quando vogliamo simulare sistemi considerando anche tutta la parte legata agli osservatori, è meglio lavorare sotto Scicos. La parte di controllo viene quindi realizzata costruendo un blocco compatto che riceve come ingressi il segnale di riferimento e l'uscita (o le uscite) del processo, e ha in uscita il segnale di controllo da applicare al processo (vedi figura 6.2). Due funzioni *comp_form* e *comp_form_i*) permettono di calcolare questo controllore compatto sia per il caso senza integrazione che per il caso in cui l'integrazione è presente. È possibile inoltre, una volta determinata la forma compatta con integratore, costruire il controllore introducendo un meccanismo anti-windup.

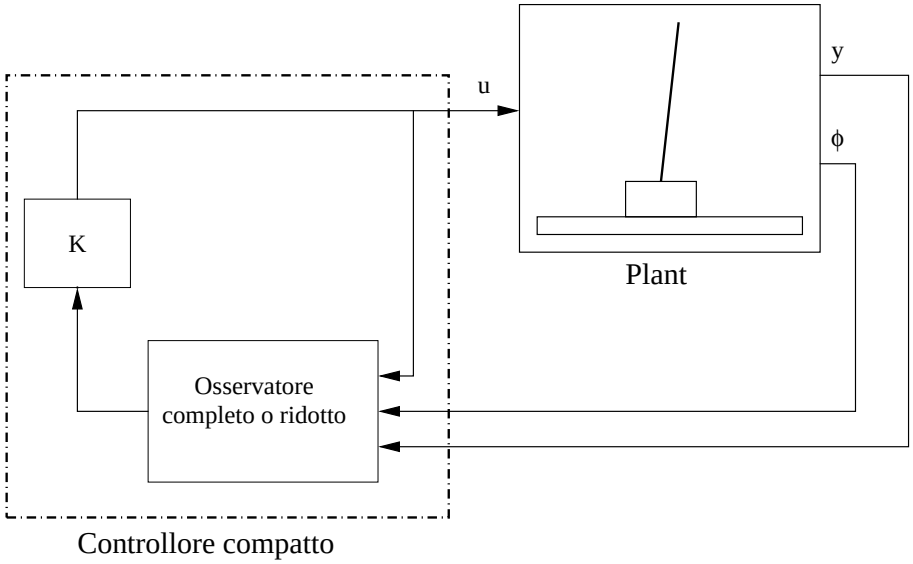

Figura 6.2: Controllore e osservatore in forma compatta

Il prossimo esempio mostra i calcoli di diverse configurazioni, con controllore di stato e precompensazione, controllore di stato con parte integrale e controllore di stato con integratore e sistema anti-windup. Il controllore con parte integrale è realizzato completamente in forma compatta, e in un secondo tempo viene realizzato con la parte anti-windup.
Un processo con doppio integratore è stato ralizzato con resistenze e condensatori come da figura 6.3. Lo schema a blocchi per Scicos è rappresentato nella figura 6.4. Il controllore discreto ha un tempo di campionamento di $1ms$.
Una volta che abbiamo introdotto lo schema di figura 6.4 in un *superblock* di Scicos si può semplicemente determinare il numero dell'oggetto dello schema Scicos tramite il comando *Object→Get Info* e cliccando sul superblocco. Occorre mettere a *yes* il campo *Others* del dialogo che compare e, in seguito, arriva una finestra con tutte le informazioni sul blocco (Figura 6.5).
In questa seconda finestra possiamo andare a vedere il numero di blocco che è stato assegnato da Scicos al nostro superblocco e in seguito possiamo effettuare le operazioni seguenti

```
-->load models.cos
```

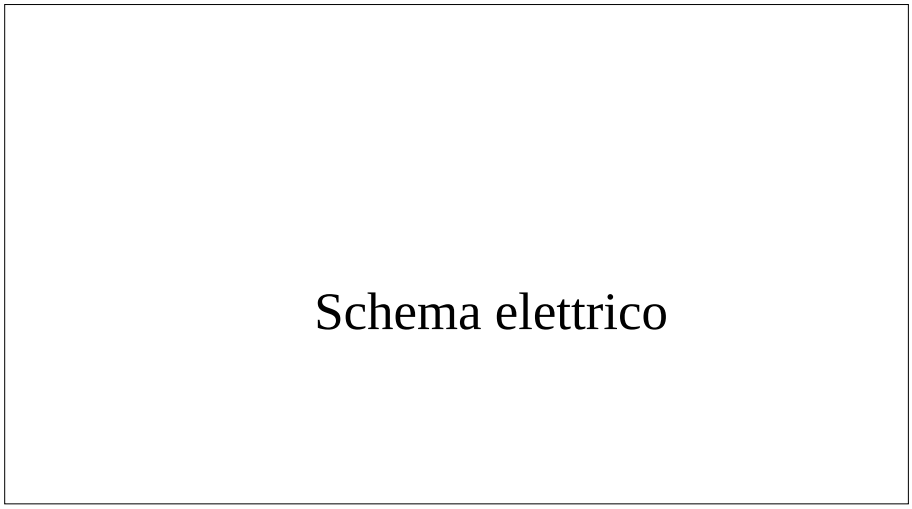

Figura 6.3: Schema elettrico del sistema da regolare

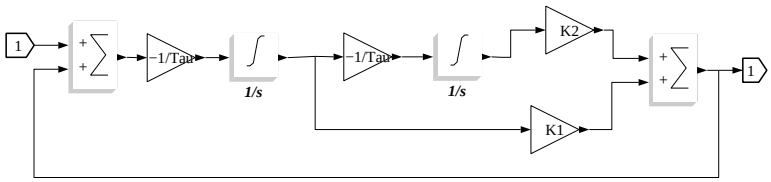

Figura 6.4: Diagramma SCICOS del sistema da regolare, contenuto del superblocco

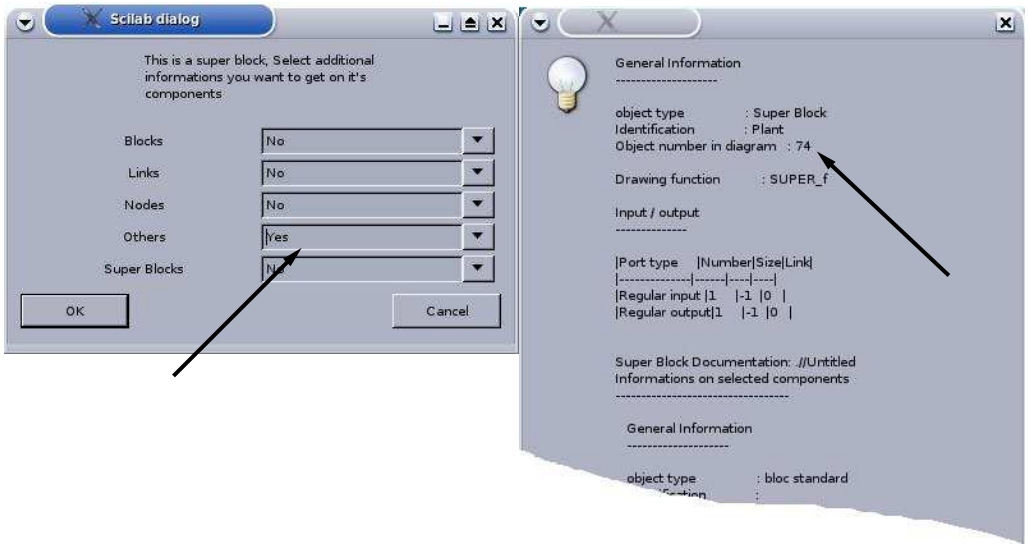

Figura 6.5: Informazioni sul superblocco

```
-->scs_m=scs_m.objs(74).model.rpar;

-->sys=lincos(scs_m);

-->ss2tf(sys)
 ans  =

    - 100 - 0.5s
    -------------
                  2
    100 + 0.5s + s
```

Possiamo anche introdurre a mano il modello nel modo seguente

```
-->Ri = 100E3 ; //** Integrator resistor 100kOhms FIXED
-->Ci = 1.0E-6 ; //** Integrator capacitor 1.0 microFarad FIXED
-->Tau = Ri*Ci ; //** Int.Time.Const = Int.Gain = 10
-->K1 =  0.05 ; //** First gain (Variable +/-1),
-->K2 = -1.0 ; //** Second gain (Variable +/-1)
-->Vsat = 10 ;  // Controller saturation

-->//** System Matrix

-->A = [-K1/(Ri*Ci) -K2/(Ri*Ci);
-->       -1/(Ri*Ci)     0          ];
-->B = [ -1/(Ri*Ci) ;
-->        0           ] ;
-->C = [K1 K2];
-->D = 0;
-->sys = syslin("c",A,B,C,D);
-->//** Sampling time definition
-->Ts = 10E-3; //** 1000 Hz, 6280rad/s
-->sys_d = dscr (sys, Ts); //** discretization
-->[Ad,Bd,Cd,Dd] = abcd(sys_d); // split the discrete system
```

A questo punto sono state introdotte le matrici della rappresentazione di stato e si è determinato il modello discreto del processo.
Per il controllo si vogliono portare tutti i poli nella posizione -4 (sistema continuo).
Procediamo ora con lo sviluppo del controllore senza integratore. Lo schema Scicos della simulazione è lo stesso sia nel caso che si utilizzi un osservatore completo, sia nel caso si utilizzi un osservatore ridotto (Figura 6.6).
I valori dei feedback degli stati si trova direttamente con un piazzamento di poli nel discreto.

```
-->pc1 = -4 ;
-->pc2 = -4 ;
-->pc=[pc1,pc2];
-->pd = exp(pc*Ts);

-->// Pole placement for state feedback
-->F = ppol(Ad, Bd, pd) ;

-->Adcl = Ad - Bd * ( F ) ;

-->//  Simulate the controlled system
-->Adnew = Adcl;
-->Bdnew = Bd;
```

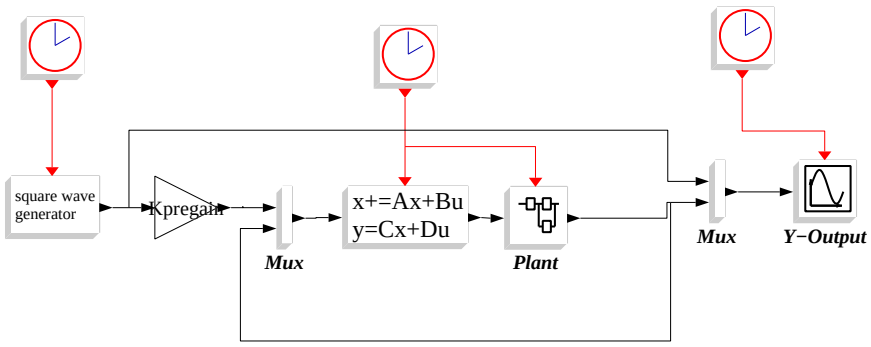

Figura 6.6: Schema scicos per il regolatore di stato senza integratore

```
-->Cdnew = Cd;
-->Ddnew = Dd;
-->Gzctr = syslin('d',Adnew,Bdnew,Cdnew,Ddnew);

-->// Set the input signal to ""step" and simulate
-->// the system output
-->u = ones(1:1000);
-->y = dsimul(Gzctr,u);
-->// Get the value of the precompensation
-->Kpregain = 1/y($);
```

Si può ora passare al calcolo dell'osservatore completo e alla realizzazione del controllore in forma compatta.

```
-->// Full Observer -> Controller is calculated in compact form
-->//                  with input "ref" and "y" and output "u"
-->pc_obs = 10*pc;
-->pd_obs = exp(pc*Ts) ;
-->L_ppol = ppol(Ad',Cd',pd_obs);
-->L = L_ppol';

-->// Full observer
-->Aobs1 = Ad-L*Cd;
-->Bobs1 = [Bd-L*Dd,L];
-->Cobs1 = eye(2,2);
-->Dobs1 = [0,0;0,0];

-->// Compact controller
-->Contr1 = comp_form(Aobs1,Bobs1,Cobs1,Dobs1,Ts,F);
```

Nel caso invece che si voglia utilizzare un osservatore ridotto, possiamo sfruttare la funzione *redobs* (vedi allegato B) che permette di determinare direttamente le matrici di stato dell'osservatore. In paticolare, la matrice T deve essere scelta in modo che la matrice quadrata $[C;T]$ sia invertibile.

```
-->// Reduced Observer -> Controller is calculated in
-->//                     compact form with input
-->//                     "ref" and "y" and output "u"
-->T=[1,0];
```

```
-->[Aobs2,Bobs2,Cobs2,Dobs2]=redobs(Ad,Bd,Cd,Dd,T,pd_obs(1));
-->Contr2 = comp_form(Aobs2,Bobs2,Cobs2,Dobs2,Ts,F);
```

Per il calcolo del controllore con parte integrale viene introdotto uno stato supplementare e quindi occorre aggiungere un polo prima di calcolare i feedback degli stati. Scegliamo anche per questo polo la posizione -4 (sistema contnuo).
Dovendo tenere conto dello stato supplementare dovuto all'integratore, nel caso di un sistema discreto, le nuove matrici diventano:

```
-->pc1 = -4 ;
-->pc2 = -4 ;
-->pc3 = -4 ;
-->pc=[pc1,pc2,pc3];
-->[m1,n1]=size(Ad);
-->[m2,n2]=size(Bd);
-->Ad_f=[Ad,zeros(m1,1);-Cd(1,:)*Ts,1];
-->Bd_f=[Bd;zeros(1,n2)];
-->pd = exp(pc*Ts);

-->F = ppol(Ad_f, Bd_f, pd) ;
```

Il calcolo dell'osservatore (completo o ridotto) viene fatto esattamente come per il controllore senza integratore.
Possiamo ora creare un controllore compatto comprendente anche la parte integrale, usando la funzione *comp_form_i* dell'allegato B e in seguito (se ne abbiamo la necessità) trasformare questo controllore in due parti distinte per implementare il meccanismo anti-windup.

```
-->Contr3 = comp_form_i(Aobs1,Bobs1,Cobs1,Dobs1,Ts,F);
```

Il meccanismo anti-windup viene calcolato nel modo seguente

```
-->g_contr=ss2tf(Contr3)
 g_contr  =

          column 1
                                        2
    - 0.0055836 + 0.0116229z - 0.0060486z
    ------------------------------------------
                                   2    3
  - 0.8228347 + 2.6417361z - 2.8189014z + z

          column 2
                                        2
    - 0.1209912 + 0.2447576z - 0.1237571z
    ------------------------------------------
                                   2    3
  - 0.8228347 + 2.6417361z - 2.8189014z + z

-->gctr_r=g_contr(:,1);

-->gctr_y=g_contr(:,2);

-->g_r=gctr_r.num/z^3;
-->g_y=gctr_y.num/z^3;
-->g_s=1-gctr_r.den/z^3;
-->gin=[g_r,g_y];
```



```
-->gss_in3=tf2ss(gin);
-->gss_fbk3 = tf2ss(g_s);
```

La figure 6.7 mostra lo schema del sistema completo, mentre la figure 6.8 mostra il contenuto del superblocco *controllore* con la parte anti-windup.

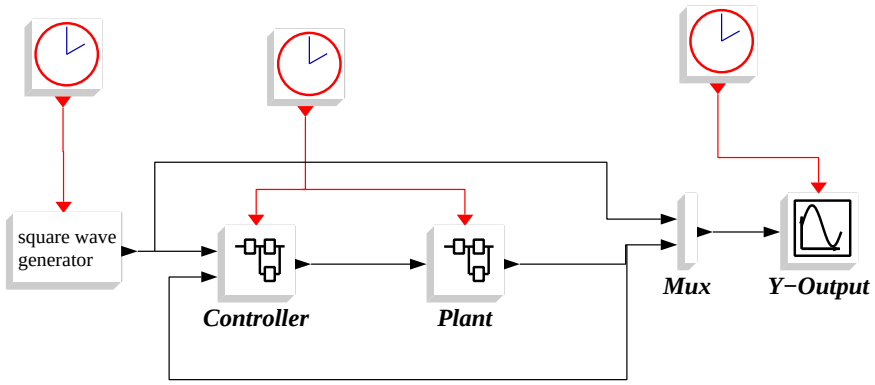

Figura 6.7: Sistema controllato con parte integrale

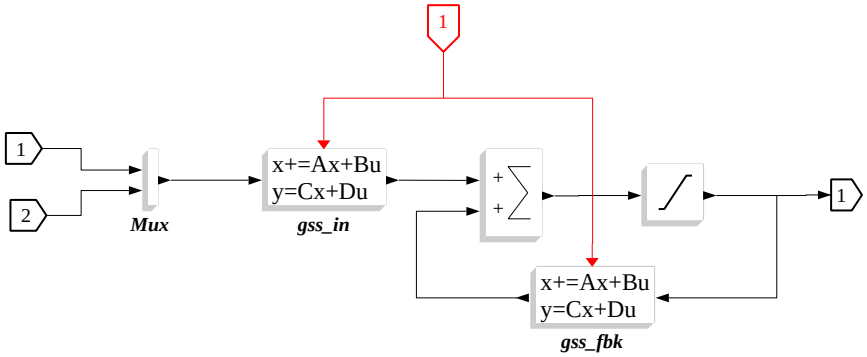

Figura 6.8: Meccanismo anti-windup

6.6 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

$$u = -Kx$$

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	regolatore LQR (continuo e discreto)
lqe	linear quadratic estimator (filtro di Kalman)
gcare	Equazione di Riccati
gfare	Filtro di Riccati

Tabella 6.2: Funzioni LQR

L'utilizzo della funzione *lqr* non è immediato in Scilab; pertanto sonno state sviluppate due nuove funzioni *bb_lqr* e *bb_dlqr* per semplificare il calcolo.
L'esempio classico di controllo LQR è qui dato dal pendolo inverso della SUPSI. Dopo aver determinato i vari parametri del sistema, viene impostato il sistema linearizzato nello spazio degli stati. Le variabili *num* e *den* contengono numeratore e denominatore della funzione di rasferimento del motore che muove il carrello del pendolo, identificata dalla risposta a gradino (senza il pendolo).

```
-->rp=0.75/26.17585;      // raggio puleggia

-->m=0.08377+0.10834;     // massa pendolo

-->Tosc=29/21;            // frequenza di oscillazione del pendolo

-->r=0.026+.297;          // distanza sul centro di massa del pendolo

-->num=g.num;             // numeratore G(s) motore-carrello-cinghie

-->den=g.den;             // denominatore G(s) motore-carrello-cinghie

-->g=9.81;

-->Theta=m*g*r*Tosc^2/(4*pi^2); // Inerzia pendolo

-->kt=60.3e-6;                  // Costante di coppia motorw

-->k_encp = -1;                 // rapporto msura encoder-angolo pendolo

-->A=[   0 1 0 0
      m*r*(M+m)*g/(Theta*(M+m)-m*m*r*r) 0 0 -m*r*d/(Theta*(M+m)-m*m*r*r)
      0 0 0 1
      m*m*r*r*g/(Theta*(M+m)-m*m*r*r) 0 0 -Theta*d/(Theta*(M+m)-m*m*r*r)];
```

```
-->B=[0
      m*r*kt/(rp*(Theta*(M+m)-m*m*r*r))
      0
      kt*Theta/(rp*(Theta*(M+m)-m*m*r*r))];

-->C=[1 0 0 0; 0 0 1 0];

-->D=[0;0];
```

A questo punto si può determinare il controllore LQR, impstando le matrici Q e R con i pesi.

```
-->sys=syslin('c',A,B,C,D);

-->sysd=dscr(sys,Ts);

-->[Ad,Bd,Cd,Dd]=abcd(sysd);

-->// Controllore LQR discreto

-->// weights
-->  Q=diag([500,1,500,1]);

-->R=[1];

-->k_lqr=bb_dlqr(Ad,Bd,Q,R);
```

Ora non resta che determinare l'osservatore (in questo caso ridotto) e il controllore in forma compatta.

```
-->preg=max(abs(spec(A)));

-->// Osservatore ridotto
-->poli_oss=exp([-preg,-preg]*10*Ts);

-->T=[0,0,0,1;0,1,0,0];

-->[Ao,Bo,Co,Do]=redobs(Ad,Bd,Cd,Dd,T,poli_oss);

-->Contr=comp_form(Ao,Bo,Co,Do,Ts,k_lqr);
```

Capitolo 7

Identificazione

7.1 Identificazione dalla risposta

Conoscendo la forma della funzione di trasferimento (con alcuni parametri sconosciuti) di un certo processo, e la risposta al gradino (dati registrati), è possibile trovare la sua antitrasformata di Laplace e tramite un'analisi di minimi quadrati si può in seguito risalire all'identificazione parametrica del processo.
Se prendiamo un motore dc che muove un carrello su un binario, dalla teoria è possibile determinare (principi primi) la sua funzione di trasferimento tra la corrente in entrata e l'angolo in uscita che risulta essere:

$$\frac{Kt}{s\ (r^2\, s\, M + Jm\, s + d\, r^2 + Dm)}$$

dove d rappresenta l'attrito del carrello, D_m l'attrito del motore, M la massa del carrello, J_m l'inerzia del motore e r il raggio del perno che muove il carrello.
Questa funzione di trasferimento può essere generalizzata in

$$\frac{\Phi_m(s)}{I_a(s)} = \frac{K}{s \cdot (s + \alpha)}$$

L'antitrasformata di Laplace di questa funzione con entrata gradino risulta essere

$$\varphi(t) = -\frac{K \cdot I_a}{\alpha^2} + \frac{K \cdot I_a}{\alpha} \cdot t + \frac{K \cdot I_a}{\alpha^2} \cdot e^{-\alpha \cdot t}$$

Se ora si applica una corrente pari a I_a ad un motore dc e si registrano i valori dell'angolo in uscita (lettura ad esempio tramite un encoder e trasformazione in radianti) si può utilizzare lo script seguente per determinare in modo sufficientemente preciso i valori di K e α del motore, e avere quindi la sua funzione di trasferimento.

```
x=read('mot.dat',-1,2);
t=x(:,1);
y=x(:,2);

t=t(400:$);     // Step starts after 2 sec.
t=t-t(1);
y=y(400:$);
t=t(1:400);     // Use only the first 2 sec of registered signal
y=y(1:400);
```

```
xbasc()
plot(t,y)

ts=5e-3;        // Sampling Time

function z=fun(p,t,y)
z=y+p(1)/p(2)^2-p(1)/p(2)*t-p(1)/p(2)^2*exp(-p(2)*t);
endfunction

Ia=400;      // mA
p0=[1,4];

[ff,p]=leastsq(list(fun,t,y/Ia),p0);

g=syslin('c',p(1)/(s*(s+p(2))));

xbasc()
bb_step(Ia*g,2)
plot(t,y)
```

Lanciando questo script si ottiene la figura 7.1, con i dati misurati e la risposta del sistema identificato.

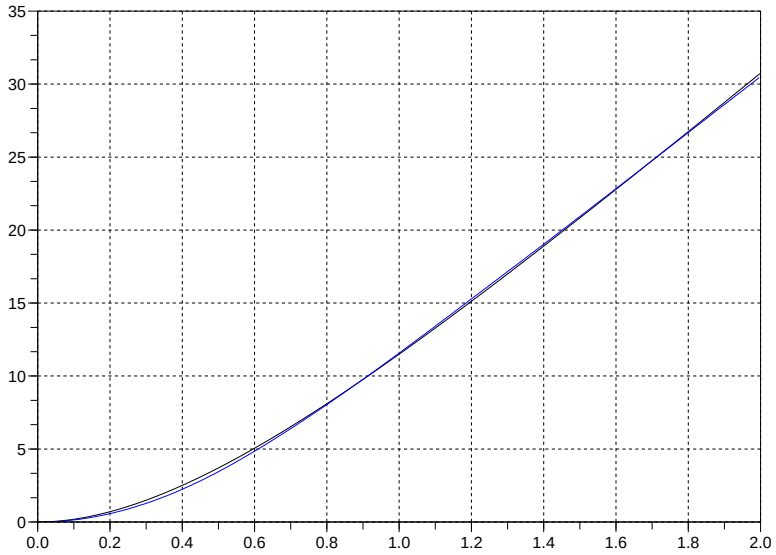

Figura 7.1: Risposta del sistema identificato con i dati registrati

La funzione di trasferimento risulta essere

```
-->g
 g  =
```

```
    0.0999468
-------------------
                  2
1.9453917s + s
```

7.2 Identificazione non parametrica

Nell'allegato B sono riportate 2 funzioni che possono essere utilizzate per effettuare una identificazione non parametrica partendo da misurazioni sul processo (*bb_freqid* e *xspectrum*). Applicando un segnale specifico al processo, per esempio un segnale prbs (pseudo rando binary signal), opuure un rumore bianco, oppure ancora un segnale di tipo sweep, è possibile effettuare una identificazione non parametrica di un processo.
La figura 7.2 mostra uno schema per una identificazione basata su un segnale prbs, calcolato tramite lo script seguente:

```
y=prbs_a(20000,15000);
write('prbs.dat',y');
```

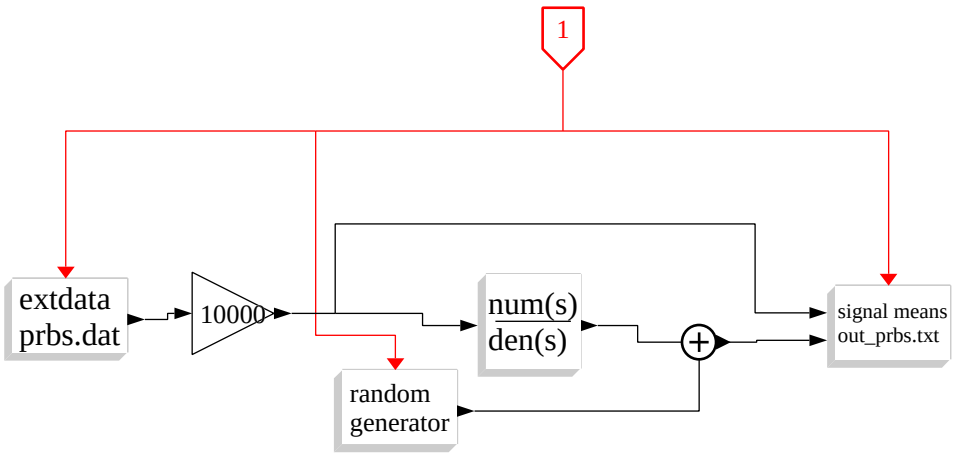

Figura 7.2: Schema per l'identificazione con entrata prbs

Il sistema viene identificato utilizzando la generazione di codice fornita da Linux RTAI. All'entrata troviamo un modulo RTAI che carica il segnale d'ingresso da un file, mentre all'uscita abbiamo il modulo *signal_means* che fa una media di tutti i dati raccolti dopo un certo tempo in cui il segnale di ingresso viene ripetutamente immesso nel sistema. In questa simulazione, il segnale in uscita viene addizionato con un segnale casuale per simulare il rumore della misurazione reale. Il fatto di fare una media su più misure in uscita permette di attenuare l'effetto del rumore di misura.
Con lo script seguente è possibile fare l'identificazione non parametrica del processo, rappresentando i due risultati su un diagramma di ampiezza (*gainplot*).

```
data=read('out_prbs.txt',-1,2);
```

```
x=data(:,1);
y=data(:,2);
ts=1/1000;

P1=bb_freqid(x,y);
frq1=0:1/ts/2/length(P1):(1/ts/2-1/ts/2/length(P1));
xbasc()

subplot(211)
gainplot(frq1(2:$),P1(2:$))
xtitle('bb_freqid');

P2=xspectrum(x,y);
frq2=0:1/ts/2/length(P2):(1/ts/2-1/ts/2/length(P2));

subplot(212)
gainplot(frq2(2:$),P2(2:$))
xtitle('xspectrum');
```

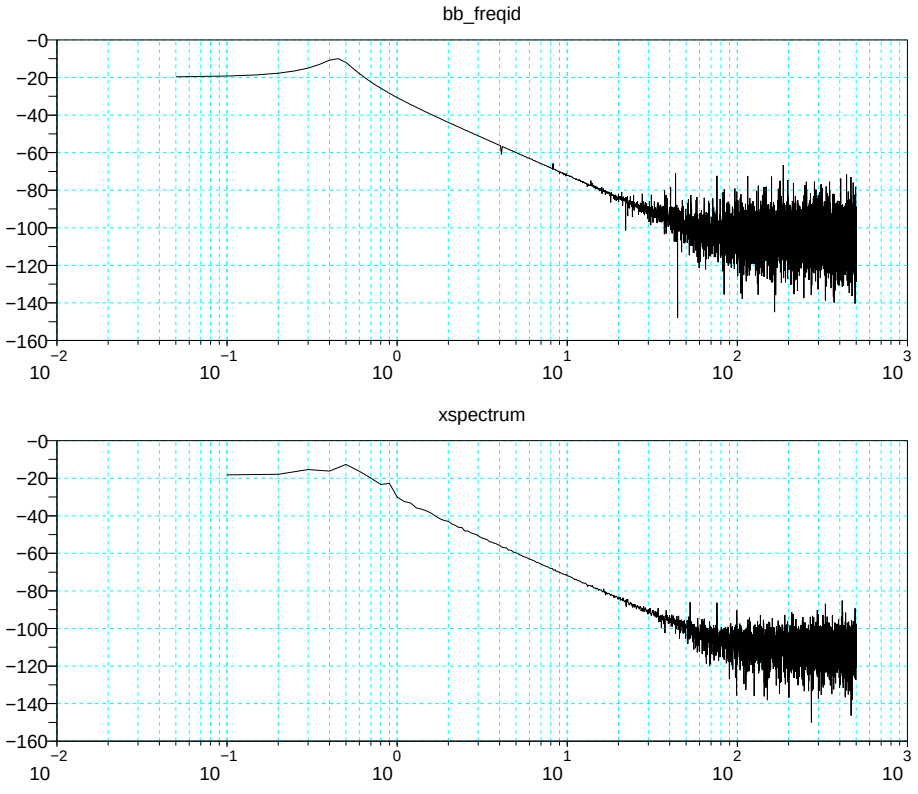

Figura 7.3: Identificazione non parametrica

Il risultato è rappresentato nella figura 7.3.

7.3 Identificazione parametrica

Scilab mette a disposizione diverse funzioni per effettuare una identificazione parametrica, partendo da diversi tipi di dati del processo (vedi tabella 7.1).

time_id	identificazione SISO con minimi quadrati (discreto)
imrep2ss	realizzazione nello spazio degli stati dalla risposta all'impulso
frep2tf(frq,reps,dg,dom)	funzione di trasferimento da risposta in frequenza
frfit(w,fresp,order)	identificazione dalla frequenza
mrfit(w,fresp,order)	identificazione dalla frequenza (solo ampiezza)

Tabella 7.1: Funzioni per l'identificazione parametrica

Appendice A

Istallazione di funzioni esterne

Per semplificare la soluzione di determinati problemi, alcuni calcoli sono stati riuniti in funzioni. Queste funzioni sono implementate utilizzando il linguaggio di programmazione messo a disposizione dall'ambiente Scilab. Il loro utilizzo non è immediato, come ad esempio in altri linguaggi come Matlab, ma queste funzioni devono essere prima caricate nell'ambiente di Scilab. La soluzione più semplice è quella di raccogliere tutte queste funzioni all'interno di una stessa cartella, e di poi caricarle all'inizio della sessione Scilab tramite il comando

```
--> getd '<cartella>'
```

Ogni utente ha a disposizione un suo file di inizializzazione *.scilab*, che in Linux si trova sotto la cartella nascosta

```
.Scilab/scilab-4.1.1
```

o una cartella analoga, dipendente dalla versione utilizzata di Scilab. Il caricamento della cartella delle funzioni può essere inserito in questo file nascosto *.scilab*, ed automaticamente eseguito ad ogni inizio di sessione.
Ci sono altri metodi per crearsi dei propri ambienti di funzioni. Uno è quello di compilare tutte le funzioni personali, creando una libreria e caricando poi questa libreria tramite lo script di inizializzazione. Occorre quindi aggiungere alla cartella delle funzioni il seguente *Makefile*

```
SHELL = /bin/sh

SCIDIR=/usr/local/scilab-4.1.1

include $(SCIDIR)/Makefile.incl

.SUFFIXES: .sci .bin $(SUFFIXES)

NAME = mylib
NAM = $(PWD)

MACROS = $(wildcard *.sci)

include $(SCIDIR)/macros/Make.lib

cleanpwd:
        rm -f *.bin names lib tmp_Bin tmp_Macros genlib *~
```

sostituendo eventualmente il percorso della cartella di Scilab nella variabile *SCIDIR* e il nome della libreria in *NAME*. A questo punto è sufficiente lanciare il comando *make* da una shell di Linux e aggiungere la linea

```
load('cartella/lib')
```

al file *.scilab*, dove *cartella* è sostituito dal percorso completo dove sono i files compilati. L'ultima possibiltà è quella di crearsi un proprio ambiente di funzioni sotto la cartella */usr/local/scilab-4.1.1/contrib*. In questo caso si può inizialmente creare una cartella (p.es chiamata *rego*), con una sottocartella (chiamata ad esempio *macros*). Nella cartella *macros* si mettono tutte le funzioni, mentre la cartella *rego* contiene unicamente il file *builder.sce* e il file *loader.sce* con i codici seguenti

```
// File builder.sce
//-----------------

mode(-1);
// specific part
libname='rego' // name of scilab function library [CUSTOM]

// generic part
// get the absolute path
[units,typs,nams]=file();
clear units typs
for k=size(nams,'*'):-1:1
  l=strindex(nams(k),'builder.sce');
  if l<>[] then
    DIR=part(nams(k),1:l($)-1);
    break
  end
end

if ~MSDOS then // Unix Linux
  if part(DIR,1)<>'/' then DIR=getcwd()+'/'+DIR,end
  MACROS=DIR+'macros/' // Path of the macros directory
else  // windows- Visual C++
  if part(DIR,2)<>':' then DIR=getcwd()+'\'+DIR,end
  MACROS=DIR+'macros\' // Path of the macros directory
end

//compile sci files if necessary and build lib file
genlib(libname,MACROS)
//clear  DIR libname MACROS genlib

cd('..')
```

e

```
// loader.sce
// ----------

mode(-1);
// specific part
libname='rego' // name of scilab function library [CUSTOM]

// generic part
// get the absolute path
```

```
[units,typs,nams]=file();
clear units typs
for k=size(nams,'*'):-1:1
  l=strindex(nams(k),'loader.sce');
  if l<>[] then
    DIR=part(nams(k),1:l($)-1);
    break
  end
end

if ~MSDOS then // Unix Linux
  if part(DIR,1)<>'/' then DIR=getcwd()+'/'+DIR,end
  MACROS=DIR+'macros/' // Path of the macros directory
else  // windows- Visual C++
  if part(DIR,2)<>':' then DIR=getcwd()+'\'+DIR,end
  MACROS=DIR+'macros\' // Path of the macros directory
end

lib(MACROS)
```

Una volta aperto Scilab, ci si sposta nella cartella *rego* e si lancia il comando

```
exec builder.sce
```

per compilare la prima volta tutti i files. Nelle prossime sessioni di Scilab sarà poi sufficiente andare a caricare questa libreria tramite il menu *contrib* o mettendo l'esecuzione di *loader.sce* nel file di inizializzazione *.scilab*.

Appendice B

Funzioni supplementari

B.1 Funzione “bb_lqr”

```
function k=bb_lqr(a,b,Q,R)
// Calculates the LQR gains for continous systems
[n1,d1]=size(a);
big=sysdiag(Q,R);
[w,wp]=fullrf(big,1e-20);
C1=wp(:,1:n1);
D12=wp(:,n1+1:$);
P=syslin('c',a,b,C1,D12);
[k,X]=lqr(P);
k=-k;
```

B.2 Funzione “bb_dlqr”

```
function k=bb_dlqr(fi,H,Q,R)
// Calculates the LQR gains for discrete systems
[n1,d1]=size(fi);
big=sysdiag(Q,R);
[w,wp]=fullrf(big,1e-20);
C1=wp(:,1:n1);
D12=wp(:,n1+1:$);
P=syslin('d',fi,H,C1,D12);
[k,X]=lqr(P);
k=-k;
```

B.3 Funzione “bb_step”

```
function []=bb_step(sl,tf,index,str)
// Step response for the system "sl" from 0 to final time "tf"
[lhs,rhs]=argn(0);
t=0:0.001:tf;
y=csim('step',t,sl);
plot2d(t,y);xgrid(1);
if rhs==4 then
   xstring(t(index),y(index),str)
end
```

B.4 Funzione "bode2freq"

```
function f=bode2freq(sys,val,fmin,fmax,typ)
// Interpolation for bode values

f=sqrt(fmin*fmax);
repf=repfreq(sys,[fmin,f,fmax]);
[db,phi]=dbphi(repf);

if typ=='db' then
  valf=db;
else
  valf=phi;
end

while(abs(val-valf(2))>1000*%eps)
  delta=val-valf;
  if delta(1)*delta(2) >=0 then
    fmin=f;
  else
    fmax=f;
  end
  f=sqrt(fmin*fmax);
  repf=repfreq(sys,[fmin,f,fmax]);
  [db,phi]=dbphi(repf);
  if typ=='db' then
    valf=db;
  else
    valf=phi;
  end
end
```

B.5 Funzione "comp_form"

```
function [Contr]=comp_form(A,B,C,D,Ts,K)
// Create the compact form of the Observer ABCD and the
// gain K,
//
// A,B,C,D: Observer matrices
// Ts: sampling time
// K: state feedback gains

Bu=B(:,1);
By=B(:,2:$);
Du=D(:,1);
Dy=D(:,2:$);

X=inv(1+K*Du);

Ac=A-Bu*X*K*C;
Bc=[Bu*X,By-Bu*X*K*Dy]
Cc=-X*K*C;
Dc=[X,-X*K*Dy]
Contr=syslin('d',Ac,Bc,Cc,Dc)
```

B.6 Funzione "comp_form_i"

```
function [Contr]=comp_form_i(A,B,C,D,Ts,K,Cy)
// Create the compact form of the Observer ABCD and the
// gain K, using an integrator at the input to eliminate the
// steady state error
//
// A,B,C,D: Observer matrices
// Ts: sampling time
// K: state feedback gains
// Cy: matrix to extract the output for the steady state feedback

[larg,rarg]=argn(0);

if rarg ~= 7 then
        Cy = [1];
end

ss_sys=syslin('d',A,B,C,D);
ss_sys(7)=Ts;
g_sys=ss2tf(ss_sys);

g_int=syslin('d',Ts/(%z-1));
g_int(7)=Ts;

gu=g_sys('num')(:,1)./g_sys('den')(:,1);
gy=g_sys('num')(:,2:$)./g_sys('den')(:,2:$);

nn=size(K,2);
Ke = K(1,nn);
K  = K(1,1:nn-1);

Greg=[-Ke*g_int/(1+K*gu),(Ke*Cy*g_int-K*gy)/(1+K*gu)];
Contr=tf2ss(Greg);
```

B.7 Funzione "init_par"

```
function [xi,wn,s]=init_par(os,ts)
// Calculates the damping factor xi, the natural frequency wn
// and the pol paar s for a 2. order system with %OS os and
// setting time ts

xi=os2xi(os);
wn=ts2wn(ts,xi);
th=acos(xi);
s=-xi*wn+%i*wn*sqrt(1-xi*xi);
```

B.8 Funzione "os2xi"

```
function [xi]=os2xi(os)
// Calculates the damping factor xi for a 2. order system
// with %OS os

os=os/100;
xi=-log(os)/sqrt(%pi*%pi+log(os)*log(os));
```

B.9 Funzione "redobs"

```
function [A_redobs,B_redobs,C_redobs,D_redobs]=redobs(A,B,C,D,T,poles)
// Find the reduced order observer for the system A,B,C,D, with observer poles "poles"
// T is the matrix needed to get the pair [C;T] invertible

P=[C;T]
invP=inv([C;T])

AA=P*A*invP

ny=size(C,1)
nx=size(A,1)
nu=size(B,2)

A11=AA(1:ny,1:ny)
A12=AA(1:ny,ny+1:nx)
A21=AA(ny+1:nx,1:ny)
A22=AA(ny+1:nx,ny+1:nx)

L1=ppol(A22',A12',poles)';

nn=nx-ny;

A_redobs=[-L1 eye(nn,nn)]*P*A*invP*[zeros(ny,nn); eye(nn,nn)];

B_redobs=[-L1 eye(nn,nn)]*[P*B P*A*invP*[eye(ny,ny);
          L1]]*[eye(nu,nu) zeros(nu,ny);
          -D, eye(ny,ny)];

C_redobs=invP*[zeros(ny,nx-ny);eye(nn,nn)];

D_redobs=invP*[zeros(ny,nu) eye(ny,ny);
               zeros(nx-ny,nu) L1]*[eye(nu,nu) zeros(nu,ny);
               -D, eye(ny,ny)];
```

B.10 Funzione "ts2wn"

```
function [wn]=ts2wn(ts,xi)
// Calculates the natural frequency of a 2. order system
// with damp "xi" and setting time "ts"

tetaxi=acos(xi);
wn=(-log(0.02*sqrt(1-xi*xi)))/(ts*xi);
```

B.11 Funzione "xi2os"

```
function [os]=xi2os(xi)
// Calculate the %OS of a 2. order system with damping "xi"

os=100*exp(-xi*%pi/sqrt(1-xi*xi));
```

B.12 Funzione “xi2pm“

```
function [pm]=xi2pm(xi)
// approximate the phase marin of a 2.order
// system with damping "xi"

pm=atan(2*xi/sqrt(-2*xi^2+sqrt(1+4*xi^4)));
pm=pm/%pi*180;
```

B.13 Funzione “xw2s“

```
function [s]=xw2s(xi,wn)
// Calculates the complex pol paar "s" of a second order system
// with damping "xi" and natural frequency "wn"
s=-xi*wn+i*wn*sqrt(1-xi*xi);
```

B.14 Funzione “bb_freqid“

```
function P=bb_freqid(x,y);
// System identification using energy spectra
// u: input signal
// y: output signal

nz=size(x,'*');
xf=fft(x);
yf=fft(y);
xf=xf(2:nz/2);
yf=yf(2:nz/2);

N=size(xf,'*');
Fxx=1/N*xf'.'.*xf;
Fxy=1/N*xf'.'.*yf;

P=Fxy./Fxx;
```

B.15 Funzione “xspectrum“

```
function Txy=xspectrum(x,y)
// System identification with cross spectral analysis

nfft=int(size(x,'*')/2);
wind=window('hn',nfft)';
n=size(x,'*');
nwind=size(wind,'*');
index=1:nwind;
k=fix(n/nwind);

Pxx=zeros(nfft,1);
Pxy2=Pxx;Pxy=Pxx;

for i=1:k
  xw=wind.*detrend(x(index));
  yw=wind.*detrend(y(index));
```



```
  index=index+nwind;
  Xx=fft(xw(1:nfft));
  Yy=fft(yw(1:nfft));
  Xx2=abs(Xx).^2;
  Xy=Yy .* conj(Xx);
  Pxx=Pxx+Xx2;
  Pxy=Pxy+Xy;
end

if modulo(nfft,2)==1 then
  selct=(nfft+1)/2;
else
  selct=nfft/2+1;
end

Pxx=Pxx(1:selct);
Pxy=Pxy(1:selct);

Txy = Pxy./Pxx;
```