# ESERCITAZIONI MATLAB SVOLTE ESAME DI IDENTIFICAZIONE DEI MODELLI E CONTROLLO OTTIMO (PROF. F. GAROFALO)

(Rif. Blocco C.Bassi solo per le tracce. NON per lo svolgimento)

1) Calcolare i punti di stazionarietà della F stabilendo se sono di minimo o massimo utilizzando un algoritmo quasi-newton. Tolleranza di terminazione dell'algoritmo sulla variabile =1e-3. Tolleranza sulla funzione =1e-8. 80 iterazioni. 1) MFILE function f=provaesame(x)  $f=6*x(2)^2-4*x(1)+3*x(1)^2-x(2)$ %definisce le variabili x e y 2) >> syms x y  $>> gradx=diff('6*y^2-4*x+3*x^2-y',x,1)$  %calcola la derivata della 'f',rispetto a x,di ordine 1 gradx = -4+6\*x $>> grady=diff('6*y^2-4*x+3*x^2-y',y,1)$ grady = 12\*y-1 >> s=solve(gradx,grady,'x,y')%risolve il sistema di equazioni s = x: [1x1 sym]y: [1x1 sym] >> s.x %mostra il valore di x ans = 2/3 >> s.y ans = 1/12 >> options=optimset('tolx',1e-3,'tolfun',1e-8,'maxiter',80);

```
>> [x,fval,exitflag,output,grad,hessian]=fminunc(@esame,[2/3;1/12],options)
f =
  -1.3750
x =
   0.6667
   0.0833
fval =
  -1.3750
exitflag =
    -2
output =
       iterations: 1
       funcCount: 33
        stepsize: []
   firstorderopt: 8.9407e-008
       algorithm: 'medium-scale: Quasi-Newton line search'
         message: [1x81 char]
grad =
  1.0e-007 *
   0.4470
   0.8941
hessian =
   6.0000
             12.0000
>> eig(hessian)
ans =
   6.0000
  12.0000
se gli autovalori sono positivi il punto è di minimo
se exitflag è =0 bisogna aumentare il numero di iterazioni
se ci fossero più punti di stazionarietà si lancia un FMINUNC per ognuno
```

```
2) Minimizzare f=100*(x(2)-x(1)^2)^2+(1-x(1))^2
con x0=[-1.2;1]
questa volta usiamo FMINSEARCH, che solitamente conviene per funzioni con esponente
massimo maggiore o uguale a 3
Scriviamo MFILE
function f=pag238(x)
f = (100*(x(2)-x(1)^2)^2+(1-x(1))^2)
poi nella command window
>>options=optimset('Largescale','off','display','final');
>> x0 = [-1.2;1];
>>[x,fval,exitflag,output]=fminsearch(@pag238,x0,options)
Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-004
x =
   1.0000
   1.0000
fval =
  8.1777e-010
exitflag =
     1
output =
    iterations: 85
     funcCount: 159
    algorithm: 'Nelder-Mead simplex direct search'
      message: [1x196 char]
```

```
3) Risolvere il problema di ottimizzazione
```

```
\min f(x)=f=\exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1)
con i vincoli:
x1*x2-x1-x2 <= -1.5
x1*x2>= -10
%i vincoli vanno riconsiderati sempre come <=
1) MFILE per f(x)
function f=pag292(x)
f = \exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1)
2) MFILE per i vincoli non lineari
function [C,Ceq]=vincpag292(x)
C=[x(1)*x(2)-x(1)-x(2)+1.5;-x(1)*x(2)-10] %matrice per vincoli > o <
Ceq=[] %matrice per vincoli =
%(le matrici si devono scrivere anche se non ci sono vincoli del loro tipo)
3)>> options =optimset('largescale','off','display','iter','maxiter',60);
%il punto e virgola mi permette di non sprecare spazio per visualizzare le
opzioni ora
4)>> x0=[-1;1];
5)>>[x,fval,exitflag,output,lambda,grad,hessian]=fmincon(@pag292,x0,[],[],[],
[],[],[],@vincpag292,options)
Optimization terminated: first-order optimality measure less
 than options. Tolfun and maximum constraint violation is less
 than options. TolCon.
Active inequalities (to within options.TolCon = 1e-006):
                       ineqlin
                                  ineqnonlin
  lower
             upper
                                      1
                                      2
x =
   -9.5474
    1.0474
```

```
fval =
    0.0236
exitflag =
     1
output =
       iterations: 8
       funcCount: 36
         stepsize: 1
        algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
    firstorderopt: 8.5133e-007
     cgiterations: []
         message: [1x144 char]
lambda =
         lower: [2x1 double]
         upper: [2x1 double]
         eqlin: [0x1 double]
      eqnonlin: [0x1 double]
       ineqlin: [0x1 double]
    ineqnonlin: [2x1 double]
grad =
   0.0184
   -0.0023
hessian =
    0.0167 -0.0039
   -0.0039
             0.0025
>> eig (hessian)
ans =
    0.0177
    0.0015
Le matrici da usare per i vincoli lineari nella sintassi di fmincon si
ricavano da questa notazione:
A*x<b
Aeq*x=beq
Lb<x<Ub
FINE
```

```
4) Progettare un regolatore LQR , con Q=1 e R=0,01 per il seguente sistema dinamico
lineare TC
xpunto = [-1 \ 1; \ 1 \ -2]x + [1;0]u
y = [1 \ 0]x
x0 = [0;10]
______
>> A = [-1 \ 1; \ 1 \ -2];
>>b=[1;0];
>>C=[1 0];
>>x0=[0;10];
>>Q=[1 0;0 1]; %deve avere le stessa dimensioni di A
>>N=0 %è un'altra matrice della funzione obiettivo, è nulla in questo caso
>>O=[C;C*A] %matrice di osservabilità
>> rank(0)
ans =
    2
>> %la matrice O ha rango pieno e dunque il sistema è osservabile
>> T=[b A*b] %matrice di controllabilità
T =
    1
        -1
    0
         1
>> rank(T)
ans =
>> % T ha rango pieno, dunque il sistema è controllabile
>> [K,S,e]=lqr(A,b,Q,R,N)
K =
   9.3193 2.7444
S =
   0.0932
            0.0274
   0.0274
            0.2449
e =
 -10.1041
```

-2.2152

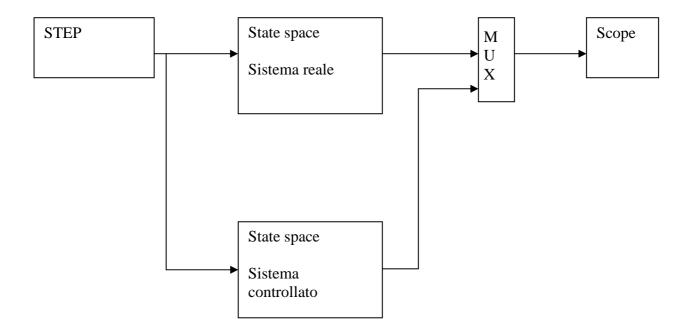
>> %questi risultati servono per simulare la dinamica del sistema a ciclo chiuso in simulink

>>%definiamo le matrici del sistema controllato

Ac=[A-b\*K] Bc=[0;0] Dc=0

% x0,C,Q,R,K,N sono le stesse

>>simulink %creare il modello



```
dell'algoritmo):
-verificare che sia osservabile e controllabile.
-progettare un regolatore di stato come insieme di osservatore e controllore di stato
dove l'errore di stima deve andare a zero in meno di 10 secondi.
-confrontare l'uscita del sistema reale (sottoposto ad un gradino uniforme) con
l'uscita del sistema controllato verificando che quest'ultima sia migliore in termini
di cifra di merito rispetto a quella del sistema originale.
>> A = [-1 \ 0; 2 \ -3];
B = [1;0];
C=[0 1];
O=[A;C*A]; %matrice di osservabilità
T=[B A*B]; %matrice di controllabilità
x0=[1;1];
rank(0)
rank(T)
ans =
     2
ans =
>>%entrambe le matrici O e T hanno rango pieno, dunque il sistema è
osservabile e controllabile
>>%L'osservatore è descritto dal seguente sistema:
xpuntostimato=Ao*xstimato+Bo*u
ystimato=Co*xstimato+Do*u
>> % ta<10 implica 5tau<10 implica tau<2 implica (-1/lambda)<2 implica
lambda<-0.5 implica lambda1=-1(autovalore dominante) e lambda2=-10 (il
dominante maggiorato di almeno un ordine di grandezza)
>> L=acker(A',C',[-1;-10])'
L =
     0
%sono i valori da moltiplicare al vettore -k al fine di
ottenere u=-k*xstimato
>> Ao=[A-L*C];
>> Bo=[1 0;0 7];
>> Co=[1 0;0 1];
>> Do=[0 0:0 0]; %L,Ao,Bo,Co,Do sono le matrici dell'osservatore
```

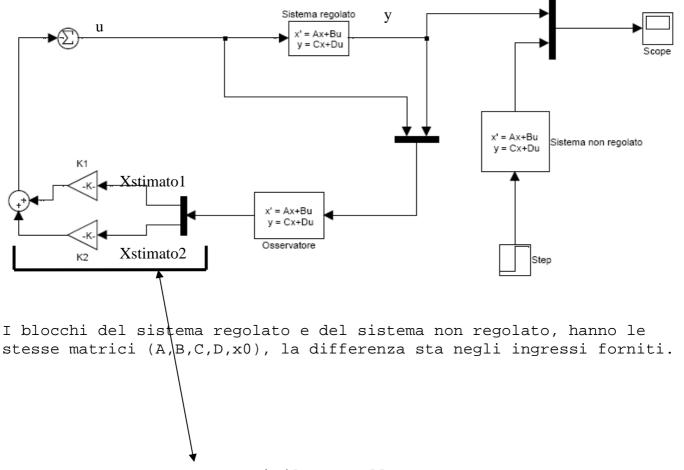
5) Per il sistema dinamico TC LTI (le matrici sono indicate nelle prime 3 righe

```
>> %scriviamo le matrici del controllore
>> Q=[1 0;0 1];
>> R=0.01;
>> K=lqr(A,B,Q,R)

K =
    9.5041    2.3339
```

>>%passo a simulink per disegnare il sistema

>>simulink



Questo è il controllore

#### FILTRO DI KALMAN

```
Dato il sistema:
xpunto(t)=A*x(t)+B*u(t)+G*w(t)
y(t) = C*x(t) + D*u(t) + H*w(t) + v(t)
dove w(t) è il rumore di processo [GWN] con media nulla e matrice
dicovarianza O
dove v(t) è il rumore di misura [GWN] con media nulla e matrice di
covarianza R
Il predittore di kalman assume la forma:
xpuntostimato(t) = A*xstimato(t) + B*u(t) + L*(y(t) - ystimato(t)
ystimato=C*xstimato(t)
dato il sistema
x(1)punto=x(2)(t)+w(t)
x(2)punto=-x(1)(t)-x(2)(t)+u(t)
y(t)=x(2)(t)+v(t)
w(t) #GWN(0;0.01)
v(t) #GWN(0;0.01)
u(t)=2
1)costruire il filtro di kalman per la stima dello stato
2) confrontare le uscite stimate con quelle reali
definiamo le matrici in un MFILE
A=[0 1;-1 -1]; %sono i coeff delle x nelle prime due eq.
B=[0 1;1 0]; %sono i coeff. di u e w nelle prime due eq.
C=[0 1]; %coeff. delle x nella 3a eq.
D=[0 0]; %coeff. delle u nella 3a eq.
x0 = [0;0];
G=[1;0]; %coeff. di w(t)
Q = 0.01;
R=0.01;
N=0; %matrice di correlaz. tra w e v, è nulla per l'ipotesi di rumore bianco
L=lqe(A,G,C,Q,R,N); %N si può anche omettere perchè è nulla
Af = [A-L*C];
Bf = [ \ 0 \ L(1); 1 \ L(2) \ ]; \ \&L(1) \ e \ L(2) \ sono \ le \ componenti \ di \ L
Df = [0 \ 0];
```

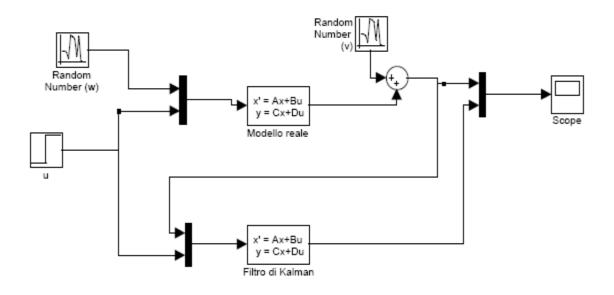
### il filtro sarà così rappresentato da questo sistema:

```
xpuntostimato(t)=Af*xstimato(t)+Bf*[u(t);y(t)]
ystimato=C*xstimato(t)

>>Simulink
modello reale:
   State-space(A,B,C,D,x0)
   step(0,0,2,0)%sarebbe la u(t)=2
   random-number w (0,0.01,0,0)
   random-number v (0,0.01,0,0)
```

#### filtro:

state-space(Af,Bf,C,Df,x0)



Nella command window:
>> pag311mat %assegna le matrici

In Simulink lanciare la simulazione

#### Da qui in poi: IDENTIFICAZIONE

#### 7) Dato il seguente sistema

$$y(t)=1*y(t-1)-0.7*y(t-2)+e(t)$$
  
 $e(t)\sim WGN(0,0.5)$ 

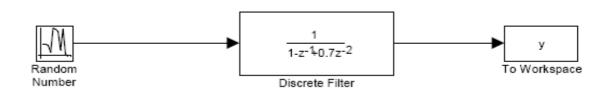
- 1- se ne simuli il comportamento in matlab/simulink generando una sequenza di 500 campioni di y
- 2- scrivere una procedura matlab per la stima ai minimi quadrati dei parametri con e(t)~WGN
- 3- Effettuare la stima dei parametri ipotizzando e(t)=sin(t)+sin(2t)
- 4- Grafica l'errore di predizione nei casi 2 e 3

\_\_\_\_\_

%Sappiamo che z^-1\*y(t)=y(t-1) quindi il nostro sistema diventa:  $y(t)=z^-1*y(t)-0.7*z^-2*y(t)+e(t)$  isoliamo le y(t), raccogliamo a fattor comune e ricaviamo y(t):  $y(t)=e(t)/A(z^-1) \quad \text{QUINDI è UN MODELLO } \textbf{AR con } A=[1 \ -1 \ 0.7]$ 

%1)costruiamo il modello AR in simulink:

- discrete-discrete filter: numeratore=[1];denominatore=[1 -1 0.7].
- sources-random number: (mettiamo i parametri di e(t)) = (0,0.5,0,0)
- sinks-to workspace:(y,inf,1,1,array)
- simulation-configuration param.: variable-step; discrete
- simulation stop time=499
- play simulation (nella workspace compare il vettore y)



%2) procedura matlab per la stima ai minimi quadrati:
la stima di teta sarà chiamata "tetastar" e vale la formula

tetastar = PHIspada\*y

in matlab la PHIspada si ottiene con il comando pinv(PHI).

```
Per realizzare la PHI (ha dimensioni [N x(na+nb)]) dobbiamo scrivere un
MFILE:
N = 500
na=2i
nb=0;
N=length(y);
y=[zeros(na,1);y];
PHI=[];
for k=1:N-1
    phi = [y(k+na:-1:k+1)'];
    PHI=[PHI;phi];
end
tetastar=pinv(PHI)*y(na+2:N+na)
al=tetastar(1:na)%vettore composto dai primi na termini di tetastar
bl=tetastar(na+1:na+nb)%vettore composto dai restanti termini di tetastar
ystima=PHI*tetastar
e=y(na+2:N+na)-ystima
m=mean(e)
sqm=(e'*e)/N-1
                %salva come idenpag315
```

```
Nella command window:
>> idenpag315

N =
    500

tetastar =
    0.9855
    -0.7121

a1 =
    0.9855
    -0.7121
```

```
b1 =
Empty matrix: 0-by-1

ystima = %qui ci sono i 500 valori y
in colonna

e = % qui compaiono i valori di e

m =

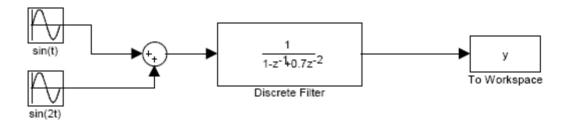
0.0354

sqm =

-0.4932
```

#### \$3) Effettuare la stima dei parametri ipotizzando e(t)=sin(t)+sin(2t)

```
Lo schema è lo stesso di prima, ma qui gli ingressi sono due funzioni seno di frequenza 1 e 2: sources-sin:(1,0,1,0,0) sources-sin:(1,0,2,0,0)
```

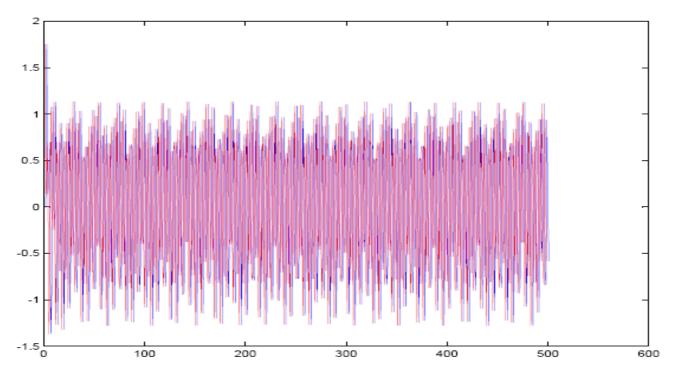


Cambiare 'e' con 'el' nel MFILE idenpag315 (all'esame dovete crearne uno nuovo)

```
Nella command window
>>idenpag315
idenpag315
N =
   500
tetastar =
   1.0082
   -0.9443
a1 =
    1.0082
   -0.9443
b1 =
   Empty matrix: 0-by-1
ystima = %valori
e1 = %valori
m =
    0.0023
sqm =
  -0.4142
```

## %4) Grafica l'errore di predizione nei casi 2 e 3

- >> plot(e1,'blue')
- >> hold on
- >> plot(e,'red')

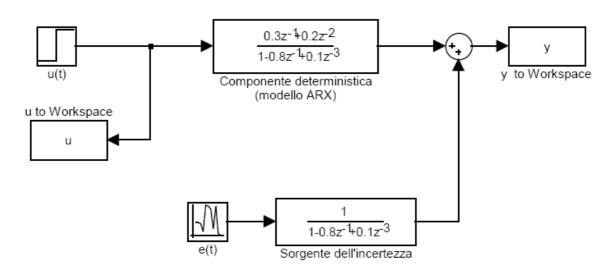


%L'errore ampiezza minore ci indicherà in quale dei due casi con l'identificazione è stata migliore, poiché vorrebbe dire che valori i ystima si avvicinano di più a quelli di y.

#### 8) Dato:

```
 y(t)*(1-0.8*z^{-1}+0*z^{-2}+0.1*z^{-3})=u(t)*(0+0.3*z^{-1}+0.2*z^{-2})+e(t) \\  ovvero \\  A(z^{-1})*y(t)=B(z^{-1})*u(t)+e(t) \\  ovvero \\  y(t)=\left\{[B(z^{-1})*u(t)]/\ A(z^{-1})\right\}+e(t) \ \ \dot{e} \ \ un \ \ \textbf{modello ARX con incertezza} \\
```

- 1) simulazione con N=1000 campioni di y e u
- sources-step: u(t),(0,0,1,0)
- sources-random number: e(t), (0,0.5,0,0)%non ci sono i dati, li ho scelti io
- discrete-discrete filter: 'componente deterministica', ([0 0.3 0.2], [1 -0.8 0 0.1],1)
- discrete-discrete filter:'sorgente incertezza',([1],[1 -0.8 0 0.1],1)
- sinks-to workspace: 'u to workspace', (u,inf,1,-1,array)
- sinks-to workspace:'y to workspace',(y,inf,1,-1,array)
- simulation stop time: 999, normal
- simulation parameter: fixed-step, discrete
- play simulation (nella workspace compaiono i vettori)

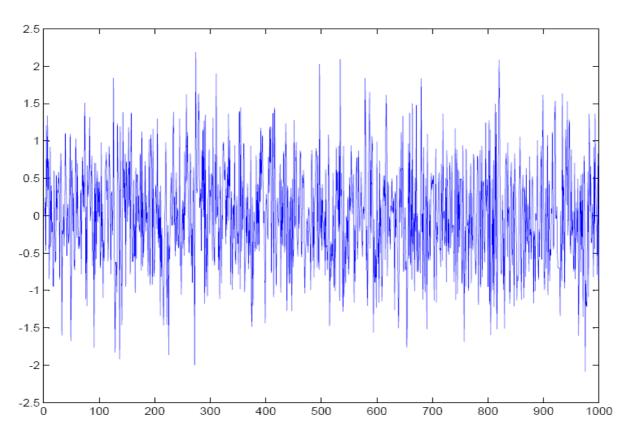


2)Procedura per la stima ai minimi quadrati dei parametri Nel MFILE idenpag322:

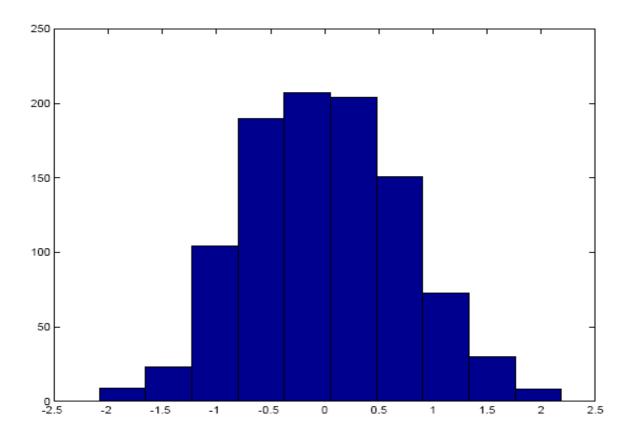
```
ystima=PHI*teta
e=y(na+2:N+na)-ystima
m=mean(e)
sqm=(e'*e)/N-1 %salva mfile
>>idenpag322
....(cut)....
teta =
   0.7445
   -0.0096
   -0.1230
    0.7890
   -0.1563
a1 =
   0.7445
   -0.0096
   -0.1230
b1 =
   0.7890
   -0.1563 %notare che teta=[a1;b1]
ystima = ...
e =...
m =
-8.3417e-016
sqm =
   -0.4757
```

3)grafico dell'errore di predizione

>> plot(e,'blue')



>> hist(e) %restituisce un istogramma (nel nostro caso ci fa vedere che l'errore ha distribuzione gaussiana)

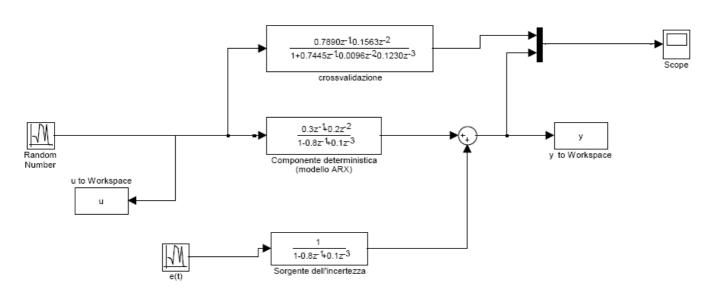


#### 4) Procedura di cross-validazione

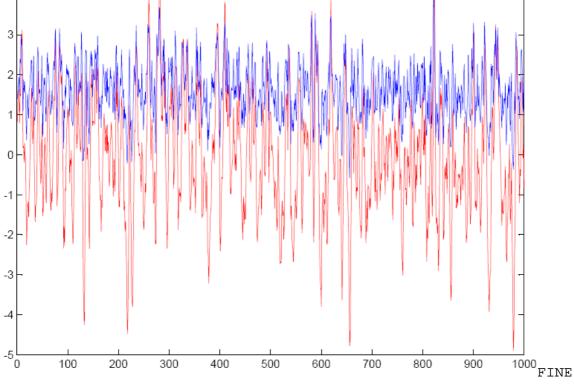
Consiste nel confrontare le uscite generate dal modello con le uscite generate dal predittore e cercare di capire in che misura il predittore riesce ad interpretare la variabilità dei dati prodotti dal modello.

Al precedente modello simulink bisogna aggiungere un blocco discrete filter che abbia:

numeratore: Bstima=[0 0.7890 -0.1563] %sono i valori di bl denominatore: Astima=[1 0.7445 -0.0096 -0.1230] %valori al sources-random number:(0,1,0,0)



>>plot(y,'red')
>>hold on
>>plot(ystima,'blue')
5
4-

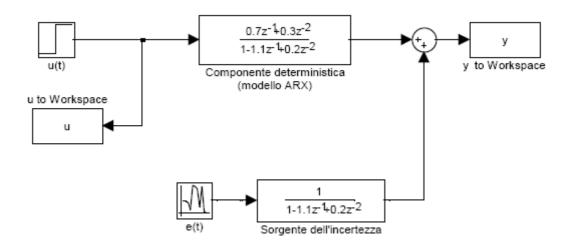


#### 9) PAG327 bassi

Svolgimento sintetico

```
A=[1 -1.1 0.2]

B=[0 0.7 0.3]
```



```
N = 500
na=3;
nb=3;
N=length(y);
y=[zeros(na,1);y];
u=[zeros(nb,1);u];
PHI=[];
for k=1:N-1
    phi=[y(k+na:-1:k+1)' u(k+nb:-1:k+1)']
    PHI=[PHI;phi]
end
teta=pinv(PHI)*y(na+2:N+na)
a=teta(1:na)
b=teta(na+1:na+nb)
ystima=PHI*teta
e=y(na+2:N+na)-ystima
m=mean(e)
sqm=(e'*e)/N-1
teta =
    1.0485
   -0.1580
    0.0021
    0.7109
    0.3272
    0.0371
a =
    1.0485
   -0.1580
    0.0021
```

-0.9990

# Sintassi per la stima ai minimi quadrati dei parametri

```
{f N}=numero di campioni (=simulation stop time)
na=massimo esponente del polinomio A;
nb=massimo esponente del polinomio B;
N=length(y); %stabilisce la dimensione N
y=[zeros(na,1);y]; %restituisce un vettore colonna in cui
i primi na termini sono zeri e poi ci sono i termini di y.
u=[zeros(nb,1);u]; %restituisce un vettore colonna in cui
i primi nb termini sono zeri e poi ci sono i termini di u.
PHI=[]; %definisce la matrice PHI.
for k=1:N-1
     phi=[y(k+na:-1:k+1)' u(k+nb:-1:k+1)'];
     PHI=[PHI;phi];
end
teta=pinv(PHI)*y(na+2:N+na)%ci dà il valore teta
stimato.
al=teta(1:na)%è un vettore composto dai primi na termini di
teta
b1=teta(na+1:na+nb) % è un vettore composto dai restanti
(nb) termini di teta
ystima=PHI*teta %è la y stimata
e=y(na+2:N+na)-ystima % errore ottenuto
m=mean(e) %valore medio dell'errore
sqm=(e'*e)/N-1%scarto quadratico medio dell'errore.
```