APPENDICE A: Codice Matlab

Lo scopo di questa appendice è di fornire al lettore un utile riferimento del codice Matlab di maggior rilievo utilizzato. In particolare, si riportano i codici degli algoritmi di stima frequenziale non presenti nei toolbox Matlab. Il codice che implementa l'algoritmo FFT è built-in Matlab (funzione *fft*), MUSIC (e quindi Pisarenko) si trova nel Signal Processing Toolbox (funzioni *rootmusic* e *pmusic*) mentre il denoising mediante wavelet è implementato all'interno del Wavelet Toolbox (funzione *wden*).
I listati di codice sotto riportati vengono utilizzati all'interno di S-function di cui, per semplicità, non si riporta l'intero contenuto, ma solo la parte essenziale. Seguendo lo standard delle S-function, con la variabile *u* si intende il segnale in ingresso da elaborare. Generalmente, inoltre, si indicherà con la variabile *numFrequenze* il numero di componenti frequenziali del segnale.

A.1 Algoritmi

A.1.1 FFT ad N frequenze

```
% uso tanti punti FFT quanti sono i campioni del segnale per avere
% il segnale ricostruito di lunghezza pari al segnale iniziale
campioniFFT = length(u);

% calcolo la DFT del segnale d'ingresso utilizzando la FFT
y = fft(u,campioniFFT);

% trovo lo spettro in potenza
Pyy = y.* conj(y) / campioniFFT;

% calcolo gli indici all'interno di y delle numFrequenze frequenze che presentano maggior potenza
[frequenze, potenze] = calcolaFrequenze (Pyy(1:(campioniFFT/2+1)),numFrequenze);

% frequenze reali stimate
frequenzeFinali = (frequenze - 1) / (campioniFFT) * (1/periodo);

% preparo il vettore di cui calcolare la IDFT, lasciando elementi non nulli solo in
% corrispondenza delle frequenze stimate
vettore = zeros(1,length(y));
for ciclo = 1:numFrequenze
   vettore(frequenze(ciclo)) = y(frequenze(ciclo));
end

% ricostruzione del segnale
segnaleRicostruito = real(ifft(vettore,campioniFFT))*2;
```

A.1.2 FFT con soglia

```
% uso tanti punti FFT quanti sono i campioni del segnale per avere
% il segnale ricostruito di lunghezza pari al segnale iniziale
campioniFFT = length(u);

% calcolo la DFT del segnale d'ingresso utilizzando la FFT
y = fft(u,campioniFFT);

% calcolo le ampiezze ed il loro valore massimo
ampiezze = abs(y);
[maxAmpiezza indice] = max(ampiezze);
% metto a zero tutte le ampiezze delle sinusoidi che non interessano
y(find(ampiezze(:) < maxAmpiezza*soglia)) = 0;

% ricostruzione del segnale
segnaleRicostruito = real(ifft(y));
```

A.1.3 Smyth

```
% calcolo del valor medio del segnale
valorMedio = sum(u)/length(u);
u = u - valorMedio;

[n cy]=size(u);
 tempo=(1:n)';

% il numero di esponenziali complessi è doppio del numero di componenti sinusoidali inserito
numFrequenze=2*numFrequenze;

X=sparse([ ],[ ],[ ],n,n-numFrequenze,(n-numFrequenze)*(numFrequenze+1) );

% creo la matrice Q1 presentata in (4.4.4)
Q=eye(numFrequenze+1);
Q=Q+Q(numFrequenze+1:-1:1,:);
Q(:,floor(numFrequenze/2+2):numFrequenze+1)=[];
Q=sqrt(Q/2);

% form u
Y=zeros(n-numFrequenze,numFrequenze+1);
for j=1:numFrequenze+1
   Y(:,j)=u(j:n-numFrequenze+j-1);
end;
YQ=Y*Q;

% Pisarenko vincolato
B=( YQ'*YQ )./(n-numFrequenze);
[x d]=eig(B); [l jmin]=min(diag(d)); g=x(:,jmin);
c=Q*g;

% Osborne/Bresler/Macovsky vincolato
gold=zeros(numFrequenze/2+1,1);
iter=0;
while norm(g-gold) > 1e-6 & iter < 50;
  iter=iter+1;
  gold=g;
  for j=1:n-numFrequenze,
    X(j:j+numFrequenze,j)=conj(c);
  end;
```

```
  B=( YQ'*((X'*X)\YQ) )./(n-numFrequenze);
  [x d]=eig(B); [l jmin]=min(diag(d)); g=x(:,jmin);
  c=Q*g;
end;

% minimi quadrati vincolato
gold=zeros(numFrequenze/2+1,1);
iter=0;
while norm(g-gold) > 1e-6 & iter < 50;
  iter=iter+1;
  gold=g;
  for j=1:n-numFrequenze,
    X(j:j+numFrequenze,j)=conj(c);
  end;
  MY=(X'*X)\Y;
  v=MY*c;
  V=zeros(n,numFrequenze+1);
  for j=1:numFrequenze+1,
    V(j:n-numFrequenze+j-1,j)=v;
  end;
  B=Q'*( Y'*MY-V'*V )*Q./(n-numFrequenze);
  [x d]=eig(B); [l jmin]=min(diag(d)); g=x(:,jmin);
  c=Q*g;
end;

% estraggo le pulsazioni delle componenti sinusoidali
om=sort(imag(log(roots(c(numFrequenze+1:-1:1) ))));
omega=om(numFrequenze/2+1:numFrequenze);

% ricostruisco le ampiezze e le fasi delle varie sinusoidi
A=[sin(tempo*omega.') cos(tempo*omega.')];
b=A\u;
b1=b(1:numFrequenze/2);
b2=b(numFrequenze/2+1:numFrequenze);
fase=atan(b2./b1);
ampiezza=sqrt(b1.*b1+b2.*b2);

% ricostruzione del segnale
mu=A*b;

% ripristino il valor medio del segnale
mu = mu + valorMedio;

% pulsazioni reali
omega = omega / periodo;

% segnale ricostruito
segnaleRicostruito = mu;
```

A.2 Altre funzioni rilevanti utilizzate

A.2.1 S-function masterRelè

```
function [sys, x0, str, ts] = masterRelè(t, x, u, flag, mu1, d1)
%   Calcola l'uscita del Master Relè

switch flag
case 0
  [sys, x0, str, ts] = mdlInitializeSizes(mu1, d1);
case 2
  sys = mdlUpdate(t, x, u, mu1, d1);
case 3
  sys = mdlOutputs(t, x, u, mu1, d1);
case {1, 4, 9}
  sys = [ ];
end
%   Fine di masterRelè

% ---------------------------------------------------------------------------------------------
%   Inizializzazione
% ---------------------------------------------------------------------------------------------
function [sys, x0, str, ts] = mdlInitializeSizes(mu1, d1)
%   Definisce le caratteristiche di base del blocco S-Function:
%   tempo di campionamento ts, condizioni iniziali x0 e
%   chiama la funzione simsize per definire una struttura sizes,inizializzarla
%   e convertirla a un vettore delle dimensioni di sizes

%   La funzione simsizes crea la struttura sizes
sizes = simsizes;
%   Si caricano nella struttura sizes le informazioni di inizializzazione
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 3;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

%   Si caricano nel vettore sys le informazioni di sizes
sys = simsizes(sizes);

prevInput = 0;
prevOutput = mu1-2*d1;
output = mu1-2*d1;
x0 = [prevInput;prevOutput;output];  % stati iniziali
str = [ ];
ts = [0, 0];    % tempo di campionamento
%Fine di mdlInitializeSize

% ---------------------------------------------------------------------------------------------
%   Aggiorna gli stati discreti e calcola il segnale in uscita
% ---------------------------------------------------------------------------------------------
function sys = mdlUpdate(t, x, u, mu1, d1)
prevInput = x(1);
prevOutput = x(2);
output = x(3);
if (prevInput < 0) & (u > 0)
  output = -prevOutput + 2*mu1;
elseif (prevInput > 0) & (u < 0)
  if prevOutput ~= (mu1 - 2*d1)
```

```
    output = prevOutput - 2*d1;
  else
    output = prevOutput;
  end
else
  output = prevOutput;
end
prevInput = u;
prevOutput = output;
sys = [prevInput;prevOutput;output];
% Fine di mdlUpdate

% ---------------------------------------------------------------------------------------------
%   Si assegna l'uscita alla S-function.
% ---------------------------------------------------------------------------------------------
function sys = mdlOutputs(t, x, u, mu1, d1)
output = x(3);
sys = output;
% Fine di mdlOutputs
```

A.2.2 S-function calcolaD1D2D3

```
function [sys,x0,str,ts] = calcolaD1D2D3(t,x,u,flag, numPeriodi)
%   Calcola:
%   D1 dutycicle del primo fronte d'onda del segnale,
%   D2 dutycicle del secondo fronte d'onda del segnale,
%   D3 è il ritardo tra i due fronti d'onda, normalizzato.
%   2TC è il periodo di oscillazione dell'uscita
%   syncro è una variabile di sincronizzazione con la S-Function CalcolaYK
%   numPeriodi è il numero di periodi dopo i quali si inizia l'analisi del segnale

switch(flag)
case 0
  [sys,x0,str,ts] = mdlInitializeSizes;
case 1
  sys = [ ];
case 2
  sys = mdlUpdate(t,x,u);
case 3
  sys = mdlOutputs(t,x,u,numPeriodi);
case {4, 9}
  sys=[ ];
otherwise
  error(['unhandled flag =',num2str(flag)]);
end;
%   Fine di calcolaD1D2D3

% ---------------------------------------------------------------------------------------------
%   Inizializzazione
% ---------------------------------------------------------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 9;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);

x0 = [-1, 0, 4, 0, 0, 0, 0,-1, 0]; %   Stati iniziali
```

```
str = [ ];
ts = [0, 0];
%Fine di mdlInitializeSize

% --------------------------------------------------------------------------------------------
%   Aggiorna gli stati discreti nel vettore che servirà per calcolare le uscite
% --------------------------------------------------------------------------------------------
function sys = mdlUpdate(t,x,u)
% memorizza l'ingresso all'istante precedente
prevInput = u;

% istante a cui l'ingresso cambia segno
timeHit = x(2);

% serve per discriminare se si sta calcolando D1 D2 D3 o il periodo
state = x(3);
d(1) = x(4);
d(2) = x(5);
d(3) = x(6);
Tc2 = x(7);
syncro = x(8);

% periodi già processati
periodi = x(9);

% u è l'ingresso attuale, x(1) è l'ingresso al passo precedente
% Quando u*x(1) < -eps allora è avvenuto un passaggio per lo zero
% si deve iniziare a calcolare una delle quantità D1 D2 D3 2TC
if (u*x(1) < -eps)
  switch (state)
  case 4
    if (u > eps)    % x(1), cioè prevInput, è negativo e u è positivo
      state = 0;
    end;
  case 0
    d(1) = t - timeHit;
    state = mod(state + 1, 4);
  case 1
    d(3) = t - timeHit + d(1);
    state = mod(state + 1, 4);
  case 2
    d(2) = t - timeHit;
    state = mod(state + 1, 4);
  case 3
    Tc2 = t - timeHit + d(2) + d(3);
    syncro = -syncro;
    periodi = periodi + 1;
    state = mod(state + 1, 4);
  otherwise
    error(['state unhandled num: ',num2str(state)]);
  end;
  timeHit = t;      % perchè avviene un cambio di segno nell'ingresso
end;
sys = [prevInput, timeHit, state, d(1), d(2), d(3), Tc2, syncro, periodi];
% Fine di mdlUpdate

% --------------------------------------------------------------------------------------------
%   Si calcolano le uscite della S-function
%   D1,D2,D3 si calcolano normalizzandoli rispetto al periodo 2Tc che è in x(7)
% --------------------------------------------------------------------------------------------
function sys = mdlOutputs(t,x,u,numPeriodi)
state = x(3);
```

```
%inizio i calcoli solo dopo il numero di periodi stabilito
if ((state == 3) & (u*x(1) < -eps) & (x(7) ~= 0)) & (numPeriodi < x(9))
  sys(1) = x(4) / x(7);    %   D1
  sys(2) = x(5) / x(7);    %   D2
  sys(3) = x(6) / x(7);    %   D3
  sys(4) = x(7);           %   2Tc
  sys(5) = x(8);           %   syncro
  sys(6) = x(9);           %   periodi processati
else
  sys = [ ];
end;
%   Fine di mdlOutputs
```

A.2.1 S-function calcolaCk

```
function [sys,x0,str,ts] = calcolaCK(t,x,u,flag,numArmoniche)
%   calcola i coefficienti Ck dell'espressione delle armoniche di x(t) come in (5.2.4)
%   riceve in ingresso le quantità D1 D2 D3 calcolate dalla funzione CalcolaD1D2D3

switch flag,
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(numArmoniche); % Inizializzazione
  case 3
    sys = mdlOutputs(t,x,u,numArmoniche); % Calcola le uscite
  case { 1, 2, 4, 9 }
    sys = []; % flags non utilizzati
  otherwise
    error(['Unhandled flag = ',num2str(flag)]); % Error handling
end;
%   Fine di calcolaCk


% --------------------------------------------------------------------------------------------
%   Inizializzazione
% --------------------------------------------------------------------------------------------

function [sys,x0,str,ts] = mdlInitializeSizes(numArmoniche)
%   Definisce le caratteristiche di base del blocco S-Function:
%   il tempo di campionamento ts,
%   chiama la funzione simsize per definire una struttura sizes,
%   inizializzarla e convertirla a un vettore delle dimensioni di sizes

%   La funzione simsizes crea la struttura sizes
sizes = simsizes;
%   Si caricano nella struttura sizes le informazioni di inizializzazione
sizes.NumContStates= 0;
sizes.NumDiscStates= 0;
sizes.NumOutputs=    2*numArmoniche;
sizes.NumInputs=     3;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
%   Si caricano nel vettore sys le informazioni di sizes
sys = simsizes(sizes);
x0 = [ ];    % non si hanno condizioni iniziali
str = [ ];
ts = [5 0]; % tempo di campionamento
%   Fine di mdlInitializeSizes.
```

```
% ------------------------------------------------------------------------------------------
%   Si svolgono i calcoli per determinare le uscite:
%       Ck=D1*sinc(k*d1)*exp(-j*pi*k*d1) + D2*sinc(k*d2)*exp(-j*2*pi*k*(D3+0.5*d2))
%   per k=1...numArmoniche
% ------------------------------------------------------------------------------------------
function sys = mdlOutputs(t,x,u,numArmoniche)
d1=u(1);
d2=u(2);
d3=u(3);

for k = 1:numArmoniche
            res(k) = d1*sinc(k*d1)*exp(-j*pi*k*d1) + d2*sinc(k*d2)*exp(-j*2*pi*k*(d3+0.5*d2));
            if isnan(res(k))    %controllo su un eventuale valore not a number
               sys(2*k-1) = 0;
               sys(2*k) = 0;
            else
               sys(2*k-1) = real(res(k));
               sys(2*k) = imag(res(k));
             end
end
% Fine di mdlOutputs
```

A.2.1 S-function calcolaYk

```
function [sys, x0, str, ts] = calcolaYK(t, x, u, flag,numArmoniche)
%   calcola i coefficienti Yk dell'espressione delle armoniche di y(t) come in (5.2.8)
%   il vettore u contiene il valore del segnale y(t), il suo periodo e la variabile syncro

switch (flag)
case 0
  [sys, x0, str, ts] = mdlInitializeSizes(numArmoniche); % Inizializzazione
case 1
  sys = mdlDerivatives(t, x, u, numArmoniche);   %   derivazione degli stati
case 2
  sys = mdlUpdate(t, x, u, numArmoniche);    %   aggiornamento
case 3
  sys = mdlOutputs(t, x, u, numArmoniche);   % Calcola le uscite
case {4, 9}
  sys = [ ];    % flags non utilizzati
end;
%   Fine di calcolaYK

% ------------------------------------------------------------------------------------------
%   Inizializzazione
% ------------------------------------------------------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes(numArmoniche)
sizes = simsizes;
sizes.NumContStates  = numArmoniche*2 + 1;       % parte reale e immaginaria e valor medio
sizes.NumDiscStates  = 4 + (numArmoniche*2 + 1) * 3;
sizes.NumOutputs     = numArmoniche*2 + 1 + 1;   % tutti gli Yk e la variabile di sincronizzazione
sizes.NumInputs = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);

%variabili che mappano gli stati discreti per leggibilità,
prevSyncro = -1;      % valore precedente di syncro
prevT = 0;            % periodo precedente
timeHit = 0;          % istante in cui y(t) cambia segno
intTimeHit = zeros(numArmoniche*2 + 1,1);  % integrale al tempo timehit precedente
xPrev= zeros(numArmoniche*2 + 1,1);    % vettore degli stati precedenti
```

```
outputs= zeros(numArmoniche*2 + 1,1);  % vettore delle uscite

x0 = [zeros(numArmoniche*2 + 1,1);prevSyncro;prevT;timeHit;intTimeHit;xPrev;outputs;0]; % stati iniziali
str = [ ];
ts = [0, 0];    % tempo di campionamento
%   Fine di mdlInitializeSizes.


% ---------------------------------------------------------------------------------------------
%   Preparazione degli Yk
% ---------------------------------------------------------------------------------------------
function sys = mdlDerivatives(t, x, u,numArmoniche)
%   calcola gli integrali necessari per calcolare Y1 Y2 Y3
%   in parte reale e parte immaginaria
%   e Y0 (non ha parte immaginaria per definizione)
timeHit = x((numArmoniche*2 + 1) + 3);
syncro = u(3);

if syncro == 0
   sys = zeros(1,2*numArmoniche + 1);
else
   for i = 1:numArmoniche
      sys(i*2 - 1) = u(1) * cos(2* pi * i * (1/u(2)) * (t-timeHit))/u(2);
      sys(i*2) = -u(1) * sin(2 * pi * i * (1/u(2)) * (t-timeHit) )/u(2);
   end
   sys(2*numArmoniche + 1) = u(1)/u(2);    %valore medio
end
%   Fine di mdlDerivatives


% ---------------------------------------------------------------------------------------------
% Aggiornamento degli stati
% ---------------------------------------------------------------------------------------------
function sys = mdlUpdate(t, x, u, numArmoniche)
numYk = 2*numArmoniche + 1;
syncro = u(3);
prevSyncro = x(numYk + 1);% è una copia di syncro
prevT = x(numYk + 2);
timeHit = x(numYk + 3);
intTimeHit = x(numYk + 4:numYk + 4 + numYk - 1);
xPrev = x(2*numYk + 4:2*numYk + 4 + numYk - 1);
outputs = x(3*numYk + 4:3*numYk + 4 + numYk - 1);
sincronizzazione = -1;   % quando viene messa a 1 dà il via al calcolo del sistema non lineare
if syncro ~= prevSyncro
     timeHit = t;
     newIntTimeHit = x(1:numYk);   % calcolato in mdlDerivates a questo istante
     outputs = newIntTimeHit -intTimeHit;
     intTimeHit = x(1:numYk);
     sincronizzazione = 1;
end

prevSyncro = syncro;
sys = [prevSyncro;prevT;timeHit;intTimeHit;xPrev;outputs;sincronizzazione];
%   Fine di mdlUpdate

% ---------------------------------------------------------------------------------------------
%   Si determinano le uscite:
%   si crea un vettore outputs dal vettore degli stati che concide poi con sys
% ---------------------------------------------------------------------------------------------
function sys = mdlOutputs(t, x, u, numArmoniche)
numYk = 2*numArmoniche + 1;
outputs = [x(3*numYk + 4:3*numYk + 4 + numYk - 1); x(length(x))];
```

```
sys = outputs;
% Fine di mdlOutputs
```

A.2.5 S-function risolviSistema

```
function [sys, x0, str, ts] = risolviSistema(t,x,u,flag,stimaa,stimab,stimadeltax,stimaritardo, ritardoMinimo,
ritardoMassimo, deltaRitardo, numArmoniche, metodo, mediazione)
%   Risolve il sistema finale in (5.2.13)

switch (flag)
case 0
  [sys, x0, str, ts] = mdlInitializeSizes(numArmoniche);
case 2
  sys = mdlUpdate(t,x,u, numArmoniche, mediazione);
case 3
  sys = mdlOutputs(t, x, u, stimaa, stimab, stimadeltax, stimaritardo, ritardoMinimo, ritardoMassimo, deltaRitardo,
numArmoniche, metodo);
case {1,4,9}
  sys = [ ];
end
%   Fine di risolviSistema


% ------------------------------------------------------------------------------------------
%   Inizializzazione
% ------------------------------------------------------------------------------------------
function [sys, x0, str, ts] = mdlInitializeSizes(numArmoniche)
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = numArmoniche*4 + 3;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 2*(2*numArmoniche) + 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);

x0 = [1 zeros(1, sizes.NumDiscStates - 1) ];

str = [];
ts = [-1, 0];

global deltaX;
global a;
global b;
global ritardo;

deltaX = 0;
a = 0;
b = 0;
ritardo = 0;
%   Fine di mdlInitializeSizes.


% ------------------------------------------------------------------------------------------
%   Aggiornamento degli stati
%   Vettore di stato x = [periodo sotto processing, real(C1), real(C2),...,imag(C1),imag(C2),...,
%                         real(Y1),...,imag(Y1),...,
% ------------------------------------------------------------------------------------------

function sys = mdlUpdate(t,x,u, numArmoniche,mediazione)
```

```
   sincro = u(4*numArmoniche+3); %sincronizzazione all'istante attuale
   prevSincro = x(4*numArmoniche+3); %sincronizzazione all'istante precedente
   if (prevSincro == 1) & (sincro == 1)
      sincro = 0;
   end
   periodo = x(1);
   Ck = x(2:numArmoniche+1) + j * x(numArmoniche+2:numArmoniche*2 + 1);
   Yk = x(numArmoniche*2 + 2:numArmoniche*3 + 1) + j * x(numArmoniche*3 + 2:numArmoniche*4 + 1);

   omegak = x(numArmoniche*4 + 2);

   if (sincro == 1) & isempty(find(u(:) == 0))    % aggiorno solo quando ho terminato un altro periodo
      Cattuali = u(1:2:2*numArmoniche) + j*u(2:2:2*numArmoniche);            % i Ck calcolati per questo periodo
      Yattuali = u(2*numArmoniche + 1:2:4*numArmoniche) + j*u(2*numArmoniche + 2:2:4*numArmoniche);
      omegaAttuale = u(4*numArmoniche + 1);

      if mediazione == 1      % se si è impostata la mediazione parametri...

         Ck = (Ck*(periodo-1) + Cattuali) / periodo;
         Yk = (Yk*(periodo-1) + Yattuali) / periodo;
         omegak = (omegak*(periodo-1) + omegaAttuale) / periodo;
      else
         Ck = Cattuali;
         Yk = Yattuali;
         omegak = omegaAttuale;
      end
      periodo = periodo + 1;
   end

   sys = [periodo real(Ck)' imag(Ck)' real(Yk)' imag(Yk)' omegak sincro];

% Fine di mdlUpdate


% ------------------------------------------------------------------------------------------
%  risoluzione del sistema
% ------------------------------------------------------------------------------------------
function sys = mdlOutputs(t,x,u,stimaa,stimab,stimadeltax,stimaritardo, ritardoMinimo, ritardoMassimo, deltaRitardo,
numArmoniche, metodo)

global deltaX;
global a;
global b;
global ritardo;

sincro = x(4*numArmoniche+3);

indice = find(x(:) == 0);

if isempty(indice) & (sincro == 1)
   C = x(2:numArmoniche+1) + j * x(numArmoniche+2:numArmoniche*2 + 1);
   Y = x(numArmoniche*2 + 2:numArmoniche*3 + 1) + j * x(numArmoniche*3 + 2:numArmoniche*4 + 1);
   om = [1:numArmoniche]'*(2*pi/(u(4*numArmoniche + 1)));

   if metodo == 1  % calcolo della soluzione mediante la fsolve
      %   In fun viene passato il sistema non lineare
      xx = fsolve(@fun, [stimaa stimab stimadeltax stimaritardo],optimset('fsolve'), C, Y, om, numArmoniche);

      sys = real(xx);% E' la soluzione del sistema: [a; b; Deltax; Delay]'

   else    % calcolo della soluzione linearizzando il sistema
```

```
      indiceQualita = [ ];
      soluzioni = [ ];

      intervallo = ritardoMinimo:deltaRitardo:ritardoMassimo;

      % introduco un vettore di pesi...
      vettorePesi = [1 1 1]';
      % ... e costruisco la matrice
      matricePesi = diag(vettorePesi);

     for cicloRitardo = intervallo
        expDelay = exp(-j*om*cicloRitardo);
        A(:,1) = (Y).*(om.^2);
        A(:,2) = -j*(Y).*om;
        A(:,3) = C.*expDelay;
        B = (Y(:));


        A = matricePesi * A;
        B = vettorePesi .* B;

        risultato = real(A\B);
        riProiezione = A*risultato;
        differenza = real(riProiezione - Y);
        soluzioni = [soluzioni; risultato'];

      % Calcolo l'indice di qualità per la soluzione trovata
        indiceQualita = [indiceQualita sum(differenza.^2)];
      end

      [minimo,indice] = min(indiceQualita);
      sys = [soluzioni(indice,:) intervallo(indice)]';
   end
   a = sys(1); b = sys(2); deltaX = sys(3); ritardo = sys(4);

else
  sys = [a b deltaX ritardo]';
end
%   fine di mdlOutputs

% ---------------------------------------------------------------------------------------------
% Fun contiene l'espressione del sistema d'equazioni da risolvere
% ---------------------------------------------------------------------------------------------
function F = fun(k, C, Y, om, numArmoniche)
   F = [ ];
   ff = [ ];
   for count = 1:numArmoniche
      ff(count) = (Y(count)/C(count))*(om(count)^2)*k(1) - j*(Y(count)/C(count))*om(count)*k(2) + ...
                  k(3)*exp((-j*om(count))*k(4)) - Y(count)/C(count);
      F = [F;real(ff(count));imag(ff(count))];
   end
```

A.2.6 S-function calcolaNonLinearità

```
function [sys, x0, str, ts] = calcolaNonLinearità(t, x, u, flag)
%   Ricostruisce per punti la non linearità
%   Per ogni simulazione si individuano una coppia di punti

switch flag,
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes;
```

```
  case 9
    sys = mdlTerminate(t, x, u);
  case { 1, 2, 4}
    sys = [ ];
end;
%   Fine di calcolaNonLinearità

% ---------------------------------------------------------------------------------------------
%   Inizializzazione
% ---------------------------------------------------------------------------------------------
function [sys, x0, str, ts] = mdlInitializeSizes

sizes = simsizes;
sizes.NumContStates= 0;
sizes.NumDiscStates= 0;
sizes.NumOutputs=    0;
sizes.NumInputs=     4;
sizes.DirFeedthrough=0;
sizes.NumSampleTimes=1;
sys = simsizes(sizes);

x0 = [ ];
str = [ ];
ts = [5, 0];
%   Fine di mdlInitializeSizes.

% ---------------------------------------------------------------------------------------------
%   Al termine della simulazione costruisce a coppie di punti la non linearità
%   Ogni volta vengono mantenuti nella figura i punti determinati nelle
%   simulazioni precedenti
% ---------------------------------------------------------------------------------------------
function  sys = mdlTerminate(t, x, u)

ulow = u(3);
uhigh = u(4);
xlow = u(1);
xhigh = u(2);
sys = [ ];

% nonlin  è una matrice che memorizza i punti della non linearità calcolati di volta in volta
% è globale x eventuali utilizzi nella finestra Matlab
global nonlin;
if (length(nonlin) ~= 0)
   heights = nonlin(:,1);
   compareLow = u(3)*ones(size(heights));
   indEqLow = find(compareLow == heights);
   nonlin(indEqLow,:)=[ ];
   heights = nonlin(:,1);
   compareHigh = u(4)*ones(size(heights));
   indEqHigh = find(compareHigh == heights);
   nonlin(indEqHigh,:)=[ ];
end
nonlin = [nonlin;[ulow, xlow; uhigh, xhigh]];
%   Fine di mdlTerminate
```