Appendix A: A general-purpose program for designing classical IIR digital filters (some modifications have been performed!)

```
*****************************************************************************************
% A general purpose matlab m-file (iirgen.m)
% for designing classical IIR filter for both
%cascade-form implementation and for
% a parallel connection of two all-pass
% filters.
% Subroutines required in addition to the
% standard matlab commands:
%
% abutter.m
% acheby1.m
% acheby2.m
% aellip.m
% convlow.m
% aminord.m
% anorm2.m
% astoprip.m
% apassrip.m
% aminoms.m
% bilin.m
% translow.m
% transsub.m
%
% The main purpose of this program is to transfer the design
% of a lowpass, higpass, bandpass, and bandstop filters to
% that of a lowpass filter with passband edge angle at pi/2.
% The design of this filter is accomplished with the aid
% of the corresponding analog filter. For this purpose, the
% bilinear transformation is used. Finally, the prototype
% filter is converted back to the desired IIR filter design
% by using a proper transformantion.
% For details, see the lecture notes.
% Both conventional cascade-form realizations and  lattice wave
% digital filters (parallel connections of two allpass filters)
% can be designed.

% Tapio Saramäki  21.11.95
% This program can be found in SUN's:
% ~ts/matlab/dsp/iirgen.m
```

```
clear all;close all;
disp('I am a program for designing classical IIR filters')
disp('First, tell me the filter type:')
disp(' ')
disp('1 for lowpass')
disp('2 for highpass')
disp('3 for bandpass')
disp('4 for bandstop')
disp(' ')
type=input('Your selection: ');
inf=input('Give 1 for detailed information, 0 for not');
disp('Then, tell me the filter criteria:')
if inf==1
disp('The passband ripple is given in desibels such that');
disp('the maximum value of the amplitude response is 0 dB');
disp('and the minimum value is -Ap dB (Ap is posiive!!)');
end
disp(' ')
Ap=input('Ap:  ');
disp(' ')
If info==1
Dips ('the stop ripple is given in decibels such that')
Dips ('the minimum of the amplitude response is -As dB')
Dips ('as is positive, also called the minimum attenuation')
End
dips(')
As=input('As:  ');
disp(' ')
if type==1
  if inf==1
  disp('For the lowpass design give the passband and stopband')
  disp('edges a fraction of pi (half the sampling rate)')
  disp('The passband edge must be less than the stopband edge!')
  end
  disp(' ')
  omegap=input('Passband edge:  ');
  disp(' ')
  omegas=input('Stopband edge:  ');
  disp(' ')
end
if type==2
  if inf==1
  disp('For the highpass design give the passband and stopband')
  disp('edges a fraction of pi (half the sampling rate)')
  disp('The passband edge must be larger than the stopband edge!')
```

```
  end
  disp(' ')
  omegap=input('Passband edge:  ');
  disp(' ')
  omegas=input('Stopband edge:  ');
  disp(' ')
end
if type==3
  if inf==1
  disp('For the bandpass design give two passband and stopband')
  disp('edges a fraction of pi (half the sampling rate)')
  disp('The passband edges must be between the stopband edges!')
  end
  disp(' ')
  omegap(1)=input('Lower passband edge:  ');
  disp(' ')
  omegap(2)=input('Upper passband edge:  ');
  disp(' ')
  omegas(1)=input('Lower stopband edge:  ');
  disp(' ')
  omegas(2)=input('Upper stopband edge:  ');
  disp(' ')
end
if type==4
  if inf==1
  disp('For the bandstop design give two passband and stopband')
  disp('edges a fraction of pi (half the sampling rate)')
  disp('The stopband edges must be between the passband edges!')
  end
  disp(' ')
  omegap(1)=input('Lower passband edge:  ');
  disp(' ')
  omegap(2)=input('Upper passband edge:  ');
  disp(' ')
  omegas(1)=input('Lower stopband edge:  ');
  disp(' ')
  omegas(2)=input('Upper stopband edge:  ');
  disp(' ')
end
%-------------------------------------------------------------
% Realization form
%-------------------------------------------------------------
disp('Next I like to know the realization form:')
disp(' ')
disp('1 for the cascade-form realization')
```

```
disp(' ')
disp('2 for the parallel connection of two allpass filters')
disp(' ')
ireal=input('Your selection: ');
disp(' ')
%-------------------------------------------------------------
% First, I convert the filter design to the design of a lowpass
% filter with the passband cutoff frequency at omegap=pi/2.
% For this purpose, I use the well-known transformations; see
% the lecture notes. The stopband edge of the lowpass filter is
% determined such that after applying the transformation we end
% up with a filter meeting the given criteria. For all filters,
% the passband edge or edges are the desired ones. The same is
% true for the stopband edge for lowpass and highpass filters.
% In the bandpass and stopband cases, only one stopband edge is
% the desired one. The remaning one is located at the frequency
% point exceeding the criteria more or less. This is because the
% transformation is not able to fix all the four edges
% simultaneously.
%
% It should be pointed out when the transformations are applied
% when converting the desired lowpass filter to the bandpass or
% bandstop filters, the filter order is doubled.
% In the lowpass and highpass cases, the order remains
% the same.
%
% The synthesis of the prototype filter is performed with the
% aid of analog filters using the bilinear transformation
%-------------------------------------------------------------
%
%-------------------------------------------------------------
% Passband edge for the prototype lowpass filter as a fraction of pi
%-------------------------------------------------------------
normomp=.5;
%-------------------------------------------------------------
% Stopband edge for the prototype lowpass filter as a fraction of pi
%-------------------------------------------------------------
normoms=convlow(omegap,omegas,type);
%-------------------------------------------------------------
% Edges for the analog lowpass prototype filter
%-------------------------------------------------------------
Omegap=1;
Omegas=tan(pi*normoms/2);
%-------------------------------------------------------------
% Time to find the minimum orders for the four different analog
```

```
% filter types (Butterworh, Chebyshev, inverse Chebyshev, and
% elliptic)
%-------------------------------------------------------------
NN(1)=ceil(aminord(Ap,As,Omegas,1));
if ireal==2 & rem(NN(1),2)==0
  NN(1)=NN(1)+1; % odd order for the parallel connection
end
NN(2)=ceil(aminord(Ap,As,Omegas,2));
if ireal==2 & rem(NN(2),2)==0
  NN(2)=NN(2)+1; % odd order is required
end
NN(3)=ceil(aminord(Ap,As,Omegas,3));
if ireal==2 & rem(NN(3),2)==0
  NN(3)=NN(3)+1; % odd order is required
end
NN(4)=ceil(aminord(Ap,As,Omegas,4));
if ireal==2 & rem(NN(4),2)==0
  NN(4)=NN(4)+1; % odd order is required
end
%-------------------------------------------------------------
% For bandpass and bandstop cases, the order is doubled
%-------------------------------------------------------------
NNN=NN;
if type==3 | type==4
  NNN=2*NN;
end
disp('The minimum orders are the following: ');
fprintf('Butterworth filter:       order = %g\',NNN(1));
disp(' ');
fprintf('Chebyshev Type I filter:  order = %g\',NNN(2));
disp(' ');
fprintf('Chebyshev Type II filter: order = %g\',NNN(3));
disp(' ');
fprintf('Elliptic (Cauer) filter:  order = %g\',NNN(4));
disp(' ');
disp('First, tell me the filter type:')
disp(' ')
disp('1 for Butterworth')
disp('2 for Chebyshev Type I')
disp('3 for Chebyshev Type II')
disp('4 for Elliptic (Cauer)')
disp(' ')
iirtyp=input('Your selection: ');
disp(' ')
%-------------------------------------------------------
```

```
% Some properties of the filter under consideration
%-------------------------------------------------------
N=NN(iirtyp);
NOVE=NNN(iirtyp);
%-------------------------------------------------------
% A^2 corresponding to epsilon^2=1
%-------------------------------------------------------
A2 = anorma2(N,Omegas,iirtyp);
%-------------------------------------------------------
% Stopband ripple corresponding to the specified value
% of Ap
%-------------------------------------------------------
Ass=astoprip(Ap,A2);
%-------------------------------------------------------
% Passband ripple corresponding to the specified value
% of As
%-------------------------------------------------------
App=apassrip(As,A2);
%-------------------------------------------------------
disp(' ')
disp('Before actual filter design, it is worth telling')
disp('the facts:')
fprintf('For the given order = %g\',NOVE);
disp(' ')
disp(' ');
disp('the following is valid: ')
fprintf('For the given passband ripple Ap = %g\',Ap);
disp(' ');
fprintf('the stopband attenuation is As = %g\',Ass);
disp(' ');
fprintf('For the given stopband attenuation As = %g\',As);
disp(' ');
fprintf('the passband ripple is Ap = %g\',App);
disp(' ');
disp('You are able to select Ap and As between the above')
disp('limits. You are also allowed to increase the filter')
disp('order, if desired')
disp(' ')
if inf==1
disp('If it seems that there is not enough tolerance for')
disp('the coefficient quantization, you can increase')
disp('the filter order:')
disp('For the cascaded form realization, the order for')
disp('the bandpass and bandstop filters must be even,')
disp('whereas for the parallel connection of two allpass')
```

```
disp('filters, the order must two times an odd integer.')
disp('Also, for the parallel connection of two allpass')
disp('filters, the order must be odd in the lowpass and')
disp('highpass cases.')
end
disp(' ')
disp('1 for changing the order')
disp('2 for not changing')
disp(' ')
ineword=input('Your selection: ');
disp(' ')
if ineword==1
  fprintf('Previous order = %g\',NOVE);
  disp(' ');
  NOVE=input('New order  :');
  N=NOVE;
  if type==3 | type==4
    N=NOVE/2;
  end
%-----------------------------------------------------
% A^2 corresponding to epsilon^2=1
%-----------------------------------------------------
  A2 = anorma2(N,Omegas,iirtyp);
%-----------------------------------------------------
% Stopband ripple corresponding to the specified value
% of Ap
%-----------------------------------------------------
  Ass=astoprip(Ap,A2);
%-----------------------------------------------------
% Passband ripple corresponding to the specified value
% of As
%-----------------------------------------------------
  App=apassrip(As,A2);
%-----------------------------------------------------
end
%-----------------------------------------------------
% Find out the minimum transition band to just meet the
% the given passband and stopband criteria
%-----------------------------------------------------
Omegass=aminoms(N,Ap,As,iirtyp);
omnewlows=2*atan(Omegass);
zz=transsub(omegap,exp(j*omnewlows),type);
omms=sort(abs(angle(zz)))/pi;
disp(' ')
disp('There are the following alternatives to design');
```

```
disp('the filter:');
disp(' ');
disp('1: the passband and stopband criteria are just met');
disp('and transition bandwidth(s) is (are) minimized');
if type < 3
  fprintf('The stopband edge is at %g\',omms);disp(' ');
end
if type > 2
  fprintf('The stopband edges are at %g\',omms(1));
  fprintf(' and %g\',omms(2));disp(' ')
end
disp(' ');
disp('2: the passband criterion is just met and the');
disp('stopband attenuation is maximized');
fprintf('In this case: Ap=%g\',Ap);
fprintf(' and As=%g\',Ass);disp(' ')
disp(' ');
disp('3: the stopband criterion is just met and the');
disp('passband variation is minimized');
fprintf('In this case: Ap=%g\',App);
fprintf(' and As=%g\',As);disp(' ')
disp(' ');
disp('4: both the passband and stopband criteria are')
disp('are exceeded in the desired manner');
disp(' ');
disp(' ');
ioptyp=input('Your selection: ');
disp(' ')
if ioptyp==1
  App=Ap;Ass=As;
end
if ioptyp==2
  Omegass=Omegas;App=Ap;
end
if ioptyp==3
  Omegass=Omegas;Ass=As;
end
if ioptyp==4
  Omegass=Omegas;
  ll=0;
  while ll==0
    disp(' ')
    disp('The given criteria are met by selecting Ap')
    fprintf('to vary between %g\',App);
    fprintf(' and As=%g\',Ap);disp(' ');
```

```
    disp(' ')
    disp('or by selecting As to vary');
    fprintf('between %g\',As);
    fprintf(' and As=%g\',Ass);disp(' ')
    disp(' ')
    ia=input('1 for selecting Ap and 2 for selecting As:  ');
    if ia==1
      disp(' ')
      Appp=input('Value for Ap: ');
      disp(' ')
      Asss=astoprip(Appp,A2);
    end
    if ia==2
      disp(' ')
      Asss=input('Value for As: ');
      disp(' ')
      Appp=apassrip(Asss,A2);
    end
    disp(' ')
    fprintf('In this case: Ap=%g\',Appp);
    fprintf(' and As=%g\',Asss);disp(' ')
    disp(' ')
    ll=input('1 for being satisfied, 0 for not:  ');
    disp(' ')
  end
  App=Appp;Ass=Asss;
end
%-------------------------------------------------
% Butterworth
%-------------------------------------------------
if iirtyp==1
  [azer,apol,ascale]=abutter(N,App);
end
%-------------------------------------------------
% Chebyshev or Chebyshev Type I
%-------------------------------------------------
if iirtyp==2
  [azer,apol,ascale]=acheby1(N,App);
end
%-------------------------------------------------
% inverse Chebyshev or Chebyshev Type II
%-------------------------------------------------
if iirtyp==3
  [azer,apol,ascale]=acheby2(N,App,Omegass);
end
```

```
%-------------------------------------------------
% Elliptic or Cauer
%--------------------------------------------------
if iirtyp==4
  [azer,apol,ascale]=aellip(N,App,Omegass);
end
%-------------------------------------------------
% Bilinear transformation
%--------------------------------------------------
zerr=azer;
poll=apol;
[zer,pol,scalel]=bilin(zerr,poll,iirtyp,App);
%----------------------------------------------------
% Transform the prototype filter into the desired form
%-----------------------------------------------------
[pole,zero,scale]=translow(omegap,pol,zer,type,scalel);
%-------------------------------------------------
%
%*****************************************************
% CASCADE-FORM REALIZATION
%*****************************************************
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Sections for the cascaded realization %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% See this to know how the file cascade is formed %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For odd-order lowpass and highpass cases, there is one
% first order section, N1=1. Otherwise, there exist no
% first order sections, N1=0. The number of second-order
% sections is N2=floor(N/2) for lowpass and highpass
% cases and N2=N for bandpass and bandstop cases.
% For further use, we generate a file called cascade
% containing first N1, N2, and the scaling constant
% scale. Then first order blocks are given in the from
% [1+A1(k)z^(-1)+A2(k)z^(-2)]/[1+B1(k)z^(-1)+B2(k)z^(-2)]
% for k=1,2,...,N2. cascade contains first the vectors
% A1 and A2, then B1 and finally B2. If there is a
% first-order section of the form
% [1+az^(-1)]/[1+bz^(-1)], a is first, then b.
% The sections are formed in such a way that, after
% filter scaling, the output noise variance due to the
% multiplication roundoff errors is rather low.
% Note that for the bandstop filter, there are two
```

```
% real poles when the order is two times an odd integer.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N1=0;
NODD=rem(N,2);
if type > 2
  N2=N;
  for kk=1:1:N2;
    B1(kk)=-2*real(pole(kk));
    B2(kk)=abs(pole(kk))*abs(pole(kk));
    A2(kk)=1;
    A1(kk)=-2*real(zero(kk));
    if rem(N2,2)==1 & kk==N2 & type==4
      B1(kk)=-(pole(kk)+pole(kk+1));
      B2(kk)=pole(kk)*pole(kk+1);
    end
    if NODD==1 & kk==N2
      A2(kk)=-1;
      A1(kk)=0;
    end
  end
end
if type < 3
  N2=floor(N/2);
  N1=rem(N,2);
  for kk=1:1:N2;
    B1(kk)=-2*real(pole(kk));
    B2(kk)=abs(pole(kk))*abs(pole(kk));
    A2(kk)=1;
    A1(kk)=-2*real(zero(kk));
  end
  if N1==1
    b=-pole(N2+1);
    a=-zero(N2+1);
  end
end
%
if ireal==1
%-----------------------------------------------------------
% Form the vector 'cascade'
%-----------------------------------------------------------
  cascade=[N1 N2 real(scale) A1 A2 B1 B2];
  if N1==1
    cascade=[cascade a b ];
  end
```

```
  disp('I print the data in the file cascade for the')
  disp('cascade-form realization')
  disp('For details, see inside the file iirgen.m')
  save cascade cascade -ascii -double
end
if ireal==2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Sections for the parallel connection of %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% two allpass sections
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% See this to know how the file allpass is formed %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% When forming the parallel connection of two allpass
% sections, only the poles are needed. We separate the
% the poles between the allpass sections C(z) and D(z).
% In the lowpass and highpass cases, there is one real
% pole, NC1=1, indicating that this pole is given to
% C(z). C(z) contains the second innermost, the fourth
% innermost pole pair and so on. D(z) contains the
% innermost pole pair, the third innermost pole pair
% and so on. The number of pole pairs included in C(z)
% and D(z) are indicated by NC2 and ND2.
% For bandpass and bandstop cases, the above second
% order blocks are formed in such a way that C1(z)
% contains the above second-order sections with
% denominator 1+B1(k)z^{-1}+B2(k)z^{-2} for k=N,
% N-3,N-4, N-6,N-7 ...N is half the filter order.
% D(z) contains the remaining sections. For bandstop
% case 1+B1(N)z^{-1}+B2(N)z^{-2} is factorizable into
% two second order sections 1+Cb(k)z^{-1} k=1,2 so that
% NC1=2.
% The vector parallel contains first NC1,NC2, and NC2.
% After that, the first order denominator sections are
% included. Then, the second order sections of C(z) and
% finally the second-order sections of D(z)
%-----------------------------------------------------
NC1=0;
if type < 3
  NC1=1;
  Cb(1)=b(1);
  ll=0;
  for k=2:2:N2
    ll=ll+1;
```

```
    CB1(ll)=B1(N2+1-k);
    CB2(ll)=B2(N2+1-k);
  end
  ll=0
  for k=1:2:N2
    ll=ll+1;
    DB1(ll)=B1(N2+1-k);
    DB2(ll)=B2(N2+1-k);
  end
  NC2=length(CB1);
  ND2=length(DB1);
end
if type > 2
  BB1=rot90(rot90(B1));
  BB2=rot90(rot90(B2));
  CB1(1)=BB1(1);
  CB2(1)=BB2(1);
  ll=1;
  for k=1+4:4:N2
    ll=ll+1;
    CB1(ll)=BB1(k-1);
    CB2(ll)=BB2(k-1);
    ll=ll+1;
    CB1(ll)=BB1(k);
    CB2(ll)=BB2(k);
  end
  ll=0;
  for k=1+2:4:N2
    ll=ll+1;
    DB1(ll)=BB1(k-1);
    DB2(ll)=BB2(k-1);
    ll=ll+1;
    DB1(ll)=BB1(k);
    DB2(ll)=BB2(k);
  end
  NC2=length(CB1);
  ND2=length(DB1);
end
if type==4
  CCB1=CB1;
  CCB2=CB2;
  CB1=CCB1(2:length(CCB1));
  CB2=CCB2(2:length(CCB2));
  NC2=NC2-1;
  NC1=2;
```

```
  Cb(1)=-pole(N2);
  Cb(2)=-pole(N2+1);
end
%-------------------------------------------------------------
% Form the vector 'parallel'
%-------------------------------------------------------------
parallel=[NC1 NC2 ND2];
if NC1>0
  parallel=[parallel Cb];
end
parallel=[parallel CB1 CB2 DB1 DB2];
disp('I print the data in the file parallel for the')
disp('parallel-form realization')
disp('For details, see inside the file iirgen.m')
  save parallel parallel -ascii -double
end
%----------------------------
% Time to plot the responses
%----------------------------
AAA=rot90(zero);
BBB=rot90(pole);
HH=rot90(ones(size(1:1:2^13)));
GDD=rot90(zeros(size(1:1:2^13)));
for k=1:N2
  A=[1 A1(k) A2(k)];
  B=[1 B1(k) B2(k)];
  [H,ww]=freqz(A,B,2^13);
  HH=HH.*H;
  [GD,w]=grpdelay(A,B,2^13);
  GDD=GDD+GD;
end
if N1 > 0
  for k=1:1
    A=[1 a];
    B=[1 b];
    [H,ww]=freqz(A,B,2^13);
    HH=HH.*H;
    [GD,w]=grpdelay(A,B,2^13);
    GDD=GDD+GD;
  end
end
HH=scale*HH;
phase=unwrap(angle(HH));
for i=1:length(HH)
  phad(i)=-phase(i)/ww(i);
```

```
end
if type==1
  omp1=0;omp2=omegap(1);
end
if type==2
  omp1=omegap(1);omp2=1;
end
if type==3
  omp1=omegap(1);omp2=omegap(2);
end
if type==4
  omp1=0;omp2=1;
end
figure(1)
subplot(2,2,3)
%----------------------------------
% Amplitude response in the passband
%----------------------------------
plot(w/pi,20*log10(abs(HH)));grid;axis([omp1 omp2 -Ap 0]);
ylabel('Amplitude in dB');
xlabel('Angular frequency omega/pi');
subplot(2,2,1)
%----------------------------------
% Amplitude response in dB
%----------------------------------
plot(w/pi,20*log10(abs(HH)));grid;
axis([0 1 -2*As 10]);
ylabel('Amplitude in dB');
xlabel('Angular frequency omega/pi');
subplot(2,2,2)
%----------------------------------
% Phase response
%----------------------------------
plot(w/pi,phase/pi);grid;
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
subplot(2,2,4)
%----------------------------------
% Phase delay
%----------------------------------
plot(w/pi,phad);grid;
ylabel('Phase delay in samples');
xlabel('Angular frequency omega/pi');
figure(2)
subplot(2,2,1)
```

```
%----------------------------------
% Pole-zero plot
%----------------------------------
zplane(AAA,BBB);title('Pole-zero plot');
subplot(2,2,2)
%----------------------------------
% Group delay
%----------------------------------
plot(w/pi,GDD);grid;
ylabel('Group delay in samples');
xlabel('Angular frequency omega/pi');
subplot(2,1,2)
%----------------------------------
% Impulse response
%----------------------------------
B=real(scale*poly(zero));
A=real(poly(pole));
impz(B,A);grid;
ylabel('Impulse response');
xlabel('n in samples')
%-----------------------------------------
% More responses for the parallel connection
%-----------------------------------------
% Allpass sections
%-----------------------------------------
BC=1;
if NC1 > 0
  for k=1:NC1
    BC=conv(BC,[1 Cb(k)]);
  end
end
if NC2 > 0
  for k=1:NC2
    BC=conv(BC, [1 CB1(k) CB2(k)]);
  end
end
BD=1;
if ND2 > 0
  for k=1:ND2
    BD=conv(BD, [1 DB1(k) DB2(k)]);
  end
end
AC=rot90(rot90(BC));
AD=rot90(rot90(BD));
[HC,ww]=freqz(AC,BC,2^13);
```

```
[HD,ww]=freqz(AD,BD,2^13);
phaseC=unwrap(angle(HC));
phaseD=unwrap(angle(HD));
figure(3)
subplot(2,2,1)
plot(w/pi,20*log10(abs(HC)));grid;
title('First allpass filter')
ylabel('Amplitude in dB');
xlabel('Angular frequency omega/pi');
subplot(2,2,3)
zplane(AC,BC);title('Pole-zero plot');
subplot(2,2,2)
plot(w/pi,phaseC/pi);grid;
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
subplot(2,2,4)
impz(AC,BC);grid;
ylabel('Impulse response');
xlabel('n in samples')
figure(4)
subplot(2,2,1)
plot(w/pi,20*log10(abs(HD)));grid;
title('Second allpass filter')
ylabel('Amplitude in dB');
xlabel('Angular frequency omega/pi');
subplot(2,2,3)
zplane(AD,BD);title('Pole-zero plot');
subplot(2,2,2)
plot(w/pi,phaseD/pi);grid;
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
subplot(2,2,4)
impz(AD,BD);grid;
ylabel('Impulse response');
xlabel('n in samples')
if NC1+2*NC2 > 2*ND2
  PHA1=phaseC;
  PHA2=phaseD;
else
  PHA1=phaseD;
  PHA2=phaseC;
end
figure(5)
subplot(211)
plot(w/pi,PHA1/pi,w/pi,PHA2/pi);grid;
```

```
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
title('Phases of the allpass sections')
subplot(212)
plot(w/pi,1+PHA1/pi,w/pi,PHA2/pi);grid;
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
title('Phases of the allpass sections: pi is added to one of the phases')
figure(6)
subplot(211)
plot(w/pi,(PHA1-PHA2)/pi);grid;
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
title('Phase difference')
subplot(212)
plot(w/pi,1+(PHA1-PHA2)/pi);grid;
ylabel('Phase as a fraction of pi');
xlabel('Angular frequency omega/pi')
title('Phase difference: pi is added to one of the phases')
figure(7)
subplot(211)
plot(w/pi,20*log10(abs(cos((PHA1-PHA2)/2))));grid;
ylabel('Amplitude in dB');
xlabel('Angular frequency omega/pi')
title('Amplitude response when adding the allpass sections')
subplot(212)
plot(w/pi,20*log10(abs(cos((PHA1+pi-PHA2)/2))));grid;
ylabel('Amplitude in dB');
xlabel('Angular frequency omega/pi')
title('Amplitude response when subtracting the allpass sections')
******************************************************************************
function [zer,pol,scale] = abutter(N,Ap)
% [zer,pol,scale] = abutter(N,Ap) determines the poles and
% zeros as well as the scaling constant of a stable analog
% Butterworth filter of order N such that the value of the
% amplitude response in decibels is 0 at Omega=0 and -Ap
% at Omega=Omegap=1 (passband cutoff point).
% It returns the real zeros and poles in vectors zer and pol.
% pol is of length N containing the poles of the filter in
% such a way that they are put in order according to the
% decreasing imaginary part.
% zer= [ ] since all the zeros are lying at the infinity.
% The scaling constant is returned in scale.

% For details, see the lecture notes by Tapio Saramäki.
```

```
% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/abutter.m
%---------------------------------------------------------------
zer=[];
epsilon=sqrt(10^(Ap/10)-1);
k=1:1:N;
pol=exp(j*pi*(1/2+(2*k-1)/(2*N)))/(epsilon^(1/N));
scale=real(prod(-pol));

*************************************************************************

function [zer,pol,scale] = acheby1(N,Ap)
% [zer,pol,scale] = acheby1(N,Ap) determines the poles and
% zeros as well as the scaling constant of a stable analog
% Chebyshev (Chebyshev Type I) filter of order N such that
% the amplitude response in decibels is varies between 0
% and -Ap in the passband [0, Omegap] with Omegap=1.
% It returns the real zeros and poles in vectors zer and pol.
% pol is of length N containing the poles of the filter in
% such a way that they are put in order according to the
% decreasing imaginary part.
% zer= [ ]  since all the zeros are lying at the infinity.
% The scaling constant is returned in scale.

% For details, see the lecture notes by Tapio Saramäki.
% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/acheby1.m
%---------------------------------------------------------------
zer=[];
epsilon=sqrt(10^(Ap/10)-1);
gamma=(1+sqrt(1+epsilon^2))/epsilon;
gamma=gamma^(1/N);
gamma1=-(gamma-1/gamma)/2;
gamma2=(gamma+1/gamma)/2;
k=1:1:N;
a=(2*k-1)*pi/(2*N);
pol=gamma1*sin(a)+j*gamma2*cos(a);
scale=real(prod(-pol));
if rem(N,2)==0
scale=scale/sqrt(1+epsilon^2); % N is even
end

*************************************************************************

function [zer,pol,scale] = acheby2(N,Ap,Omegas)
```

```
% [zer,pol,scale] = acheby2(N,Ap,Omegas) determines the
% poles and zeros as well as the scaling constant of a
% stable analog inverse Chebyshev  (Chebyshev Type II)
% filter of order N such that the value of the amplitude
% response in decibels is 0 at Omega=0 and -Ap at Omega=
% Omegap=1 (passband cutoff point) and the the stopband
% edge is located at Omega=Omegas.
% It returns the real zeros and poles in vectors zer and pol.
% pol is of length N containing the poles of the filter in
% such a way that they are put in order according to the
% decreasing imaginary part.
% zer is of length N for N even and of length N-1 for N odd
% since one zero is lying in this case at the infinity.
% zer contains the real zeros according to the decreasing
% imaginary part.
% The scaling constant is returned in scale.

% For details, see the lecture notes by Tapio Saramäki.
% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/acheby2.m
%-------------------------------------------------------------
% The poles and zeros
zer=[];
epsilon2=sqrt(10^(Ap/10)-1);
%-------------------------------------------------------------
% The poles and zeros of the inverse Chebyshev filter are
% determined according to A (see the lecture notes). There-
% fore, we first determine the value of A based on the fact
% that the stopband attenuation of the Chebyshev and inverse
% Chebyshev filters are the same for the same values of Ap
% and Omegas
%-------------------------------------------------------------
% The value of the squared-magnitude function at Omega=
% Omegas is 1/(1+[epsilon2*cosh{N*acosh(Omega_s)}]^2):
%-------------------------------------------------------------
stop=N*acosh(Omegas);
stop=epsilon2*cosh(stop);
stop=1/(1+stop^2);
%-------------------------------------------------------------
% 1/A^2=stop ----->
%-------------------------------------------------------------
A=1/stop;
A=sqrt(A);
%-------------------------------------------------------------
% Now we are ready to find out the poles and zeros
```

```
%-------------------------------------------------------------
% First poles
%-------------------------------------------------------------
gamma=A+sqrt(A^2-1);
gamma=gamma^(1/N);
gamma1=-(gamma-1/gamma)/2;
gamma2=(gamma+1/gamma)/2;
k=1:1:N;
a=(2*k-1)*pi/(2*N);
alpha=gamma1*sin(a);
beta=gamma2*cos(a);
den=alpha.*alpha+beta.*beta;
pol=Omegas*alpha-j*Omegas*beta;
pol=pol./den;
%-------------------------------------------------------------
% Then zeros; first zeros on the upper half plane
%-------------------------------------------------------------
if rem(N,2)==0 M=N/2; end
if rem(N,2)==1 M=(N-1)/2;end
k=M:-1:1;
a=pi*(2*k-1)/(2*N);
a=cos(a);
zer=j*Omegas*ones(size(a))./a;
%-------------------------------------------------------------
% Then, all the zeros on the imaginary axis
%-------------------------------------------------------------
zer=[zer -fliplr(zer)];
%-------------------------------------------------------------
% The scaling constant
%-------------------------------------------------------------
scale=1;
if rem(N,2)==0 scale=sqrt(1/(1+epsilon2));end
scale=scale*real(prod(-pol))/real(prod(-zer));

*******************************************************************************

function [zer,pol,scale] = aellip(N,Ap,Omegas)
% [zer,pol,scale] = aellip(N,Ap,Omegas) determines the
% poles and zeros as well as the scaling constant of a
% stable analog elliptic (Cauer) filter of order N such
% that the amplitude response in decibels is varies
% between 0 and -Ap in the passband [0, Omegap] with
% Omegap=1 and the the stopband edge is located at Omega=
% Omegas.
% It returns the real zeros and poles in vectors zer and pol.
```

```
% pol is of length N containing the poles of the filter in
% such a way that they are put in order according to the
% decreasing imaginary part.
% zer is of length N for N even and of length N-1 for N odd
% since one zero is lying in this case at the infinity.
% zer contains the real zeros according to the decreasing
% imaginary part.
% The scaling constant is returned in scale.

% This file has been constructed based on the article
% S. Darlington, "Simple alforithms for elliptic filters
% and generalizations thereof", IEEE Trans. Circuits and
% Systems, CAS-25, pp. 975 - 980, Dec. 1978.
% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/aellip.m
%-------------------------------------------------------------
%------------
% A(i+1)==a_i
%------------
A(1)=sqrt(Omegas);
for i=1:4
 A(i+1)=A(i)^2+sqrt(A(i)^4-1);
end
%------------------
% J(i+1)==J_i
J(5)=2^(N-1)*A(5)^N;
for i=1:4
 J(5-i)=(J(5-i+1)+1/J(5-i+1))/2;
 J(5-i)=sqrt(J(5-i));
end
K=sqrt(10^(Ap/10)-1);
%---------------
% SS(I+1)==S_I0
%---------------
SS(2)=1/K+sqrt(1/(K*K)+1);
for i=3:4
  SS(i)=J(i-1)*SS(i-1);
  if SS(i) <= 10^150
    SS(i)=SS(i)+sqrt(SS(i)*SS(i)+1);
  else
    SS(i)=2*SS(i);
  end
end
%---------------------------------------
% To prevent overflows calculate S_40/J3,
```

```
% instead of S_40
%--------------------------------------
SS(5)=SS(4)+sqrt(SS(4)*SS(4)+1/(J(4)*J(4)));
%--------------------
% Calculate S50==s_50
%--------------------
S50=J(5)/(SS(5)*J(4));
if S50 <= 10^150
  S50=S50+sqrt(S50*S50+1);
else
  S50=2*S50;
end
S50=S50^(1/N);
%----------------------------------------
% Poles of the filter
%----------------------------------------
k=1:1:N;
pol=S50*exp(j*pi*(.5+(2*k-1)/(2*N)));
for i=1:5
  pol=(pol-ones(size(pol))./pol)/(2*A(6-i));
end
[Y,I]=sort(-imag(pol));
pol=pol(I);
im=imag(pol);
re=real(pol);
pol=A(1)*(-re+j*im);
%----------------------------------------
% Zeros of the filter
%----------------------------------------
NCZ=floor(N/2);
kk=1:1:NCZ;
zer=A(5)*ones(size(kk))./(cos(pi*(2*kk-1)/(2*N)));
for i=1:4
  zer=(zer+ones(size(zer))./zer)/(2*A(5-i));
end
zer=A(1)*zer;
zer=[j*zer -j*zer];
%----------------------------------------
% The scaling constant
%----------------------------------------
epsilon2=10^(Ap/10)-1;
scale=1;
if rem(N,2)==0 scale=sqrt(1/(1+epsilon2));end
scale=scale*real(prod(-pol))/real(prod(-zer));
```

```
*************************************************************************

function normoms = convlow(omegap,omegas,type)
% Given the digital filter type:
% type=1  for lowpass
% type=2  for highpass
% type=3  for bandpass
% type=4  for bandstop
% as well as the passband and stopband edges included
% in omegap and omegas,
% normoms = convlow(omegap,omegas,type) converts the
% design problem to that of a lowpass filter with
% passband edge at omega=pi/2. It should be pointed out
% when the transformations are applied when converting
% the desired lowpass filter to the bandpass or bandstop
% filters, the filter order is doubled.
% In the lowpass and highpass cases, the order remains
% the same.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/anorma2
%---------------------------------------------------------
tp=0.5*pi;
Op=pi*omegap;
Os=pi*omegas;
if type==1
  alpha=sin((tp-Op(1))/2)/sin((tp+Op(1))/2);
  real=-2*alpha+(1+alpha^2)*cos(Os(1));
  imag=(1-alpha^2)*sin(Os(1));
  normoms=atan2(imag,real)/pi;
end
if type==2
  alpha=-cos((tp+Op(1))/2)/cos((tp-Op(1))/2);
  real=2*alpha+(1+alpha^2)*cos(Os(1));
  imag=(1-alpha^2)*sin(Os(1));
  normoms=1-atan2(imag,real)/pi;
end
if type==3
  alpha=cos((Op(2)+Op(1))/2)/cos((Op(2)-Op(1))/2);
  k=cot((Op(2)-Op(1))/2)*tan(tp/2);
  a1=-2*alpha*k/(k+1);a2=(k-1)/(k+1);
  imag=-a1*sin(Os)-a2*sin(2*Os);
  real=1+a1*cos(Os)+a2*cos(2*Os);
  oms=2*Os+2*atan2(imag,real)-pi;
  normoms=min(abs(oms(1)),oms(2))/pi
```

```
end
if type==4
  alpha=cos((Op(2)+Op(1))/2)/cos((Op(2)-Op(1))/2);
  k=tan((Op(2)-Op(1))/2)*tan(tp/2);
  a1=-2*alpha/(k+1);a2=(1-k)/(k+1);
  imag=-a1*sin(Os)-a2*sin(2*Os);
  real=1+a1*cos(Os)+a2*cos(2*Os);
  oms=2*Os+2*atan2(imag,real);
  oms(1)/pi
  2-oms(2)/pi
  normoms=min(oms(1),2*pi-oms(2))/pi;
end

*************************************************************************

function N = aminord(Ap,As,Omegas,iirtyp)
% Given the analog filter type:
% iirtyp=1  for Butterworth
% iirtyp=2  for Chebyshev or Chebyshev Type I
% iirtyp=3  for inverse Chebyshev or Chebyshev Type II
% iirtyp=4  for elliptic or Cauer
% as well as the maximum passband variation Ap in dB,
% the minimum stopband attenuation As and the stopband
% edge Omegas,
% N = aminord(Ap,As,Omegas,iirtyp) finds the minimum
% order to meet the given criteria.
% N is given as a rational number.
% In practice, N=ceil(N)!!

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/aminord.m
% For details, see the lecture notes as well as
% S. Darlington, "Simple alforithms for elliptic filters
% and generalizations thereof", IEEE Trans. Circuits and
% Systems, CAS-25, pp. 975 - 980, Dec. 1978.
%-------------------------------------------------------
epsilon2=10^(Ap/10)-1;
A2=10^(As/10);
if iirtyp==1
N=log10((A2-1)/epsilon2)/(2*log10(Omegas));
end
if iirtyp==2 | iirtyp==3
  N=acosh(sqrt((A2-1)/epsilon2))/acosh(Omegas);
end
if iirtyp==4
```

```
  A=sqrt(Omegas);
  for k=1:4
    A=A*A;
    A=A+sqrt(A*A-1);
  end
  JJ=sqrt(sqrt((10^(As/10)-1)/epsilon2));
  for k=1:4
    JJ=JJ*JJ;
    if JJ <= 10^150
      JJ=JJ+sqrt(JJ*JJ-1);
     else
      JJ=2*JJ;
     end
  end
  N=log10(2*JJ)/log10(2*A);
end

*************************************************************************

function A2 = anorma2(N,Omegas,iirtyp)
% Given the analog filter type:
% iirtyp=1  for Butterworth
% iirtyp=2  for Chebyshev or Chebyshev Type I
% iirtyp=3  for inverse Chebyshev or Chebyshev Type II
% iirtyp=4  for elliptic or Cauer
% as well as the filter order N and the stopband edge
% Omegas,
% A2 = anorma2(N,Omegas,iirtyp) finds the value of A^2
% in the case where epsilon^2=1 the passband edge is
% Omegap=1.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/anorma2.m
% For details, see the lecture notes as well as
% S. Darlington, "Simple alforithms for elliptic filters
% and generalizations thereof", IEEE Trans. Circuits and
% Systems, CAS-25, pp. 975 - 980, Dec. 1978.
%-----------------------------------------------------------
if iirtyp==1
  A2=1+Omegas^(2*N);
end
if iirtyp==2 | iirtyp==3
  A2=cosh(N*acosh(Omegas));
  A2=1+A2*A2;
end
```

```
if iirtyp==4
  A=sqrt(Omegas);
  for k=1:4
    A=A*A;
    A=A+sqrt(A*A-1);
  end
  JJ=2^(N-1)*A^N;
  for k=1:4
    JJ=sqrt((JJ+1/JJ)/2);
  end
  A2=1+JJ^4;
end

**************************************************************************

function Ap = apassrip(As,A2)
% After knowing the value of A^2 for epsilon^2=1,
% Ap = apassrip(As,A2) finds the minimum passband
% variation Ap expressed in desibels for the given
% stopband attenuation As.
% Note that both Ap and As are positive.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/apassrip.m
%--------------------------------------------------------
FF=A2-1;
BB=10^(As/10)-1;
epsilon2=BB/FF;
Ap=10*log10(1+epsilon2);

**************************************************************************

function As = astoprip(Ap,A2)
% After knowing the value of A^2 for epsilon^2=1,
% As = astoprip(Ap,A2) finds the minimum stopband
% attenuation As for the given passband ripple Ap
% expressed decibels.
% Note that both Ap and As are positive.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/astoprip.m
%--------------------------------------------------------
epsilon2=10^(Ap/10)-1;
FF=A2-1;
As=10*log10(1+epsilon2*FF);
```

```
*************************************************************************

function Omegas = aminoms(N,Ap,As,iirtyp)
% Given the analog filter type:
% iirtyp=1  for Butterworth
% iirtyp=2  for Chebyshev or Chebyshev Type I
% iirtyp=3  for inverse Chebyshev or Chebyshev Type II
% iirtyp=4  for elliptic or Cauer
% as well as the filter order N, the the maximum
% passband variation Ap and the minimum stopband
% attenuation As (both positive),
% Omegas = aminoms(N,Ap,As,iirtyp) finds the minimum
% stopband edge Omegas when the passband edge is
% Omegap=1.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/aminoms.m
%-------------------------------------------------------
epsilon2=10^(Ap/10)-1;
A2=10^(As/10);
if iirtyp==1
  Omegas=((A2-1)/epsilon2)^(1/(2*N));
end
if iirtyp==2 | iirtyp==3
  Omegas=sqrt((A2-1)/epsilon2);
  Omegas=acosh(Omegas)/N;
  Omegas=cosh(Omegas);
end
if iirtyp==4
  JJ=sqrt(sqrt((10^(As/10)-1)/epsilon2));
  for k=1:4
    JJ=JJ*JJ;
    if JJ <= 10^150
      JJ=JJ+sqrt(JJ*JJ-1);
     else
      JJ=2*JJ;
     end
  end
  A=JJ/(2^(N-1));
  A=A^(1/N);
  for k=1:4
    A=sqrt((A+1/A)/2);
  end
  Omegas=A^2;
```

```
end

*************************************************************************

function [zer,pol,scale]=bilin(zerr,poll,iirtyp,Ap)
% Given the analog filter type:
% iirtyp=1  for Butterworth
% iirtyp=2  for Chebyshev or Chebyshev Type I
% iirtyp=3  for inverse Chebyshev or Chebyshev Type II
% iirtyp=4  for elliptic or Cauer
% as well as the passband variation Ap, its zeros
% included in zerr and poles icluded in poll,
% [zer,pol,scale]=bilin(zerr,poll,iirtyp) finds out
% the zeros of the corresponding digital filter
% included in zer, the poles included in in pol, and
% the scaling constant scale included in scale in the
% case where the bilinear transformation maps Omegap=
% 1 to omegap=pi/2. The transformation is simply z=
% (1+s)/(1-s). If some of the zeros of the analog
% filter lie at infinity, they are mapped to z=-1.
% In this case the length of zerr is lower.
% The scaling constant is determined such that at
% omega=1, the desired value is achieved.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/translow.m
%-------------------------------------------------
% First poles
%-------------------------------------------------
poll1=poll+1;
poll2=-poll+1;
pol=poll1./poll2;
%-------------------------------------------------
% Then zeros
%-------------------------------------------------
if length(zerr) > 1
  zerr1=zerr+1;
  zerr2=-zerr+1;
  zer=zerr1./zerr2;
end
if length(zerr) < length(poll)
  zer=[zer -ones(size(length(zerr):1:length(poll)-1))];
end
%-------------------------------------------------------
% Sort the poles according to the decreasing
```

```
% imaginary part
%-----------------------------------------------------
[Y,I]=sort(-imag(pol));
pol=pol(I);
%-----------------------------------------------------
% Sort the zeros according to the decreasing
% imaginary part
%-----------------------------------------------------
[Y,I]=sort(-imag(zer));
zer=zer(I);
%-----------------------------------------------------
% Desired value of the amplitude response at omega=1;
% For Butterworth and inverse Chebyshev filters, the
% desired value is equal to unity. The same is true for
% odd order Chebyshev filters and elliptic filters.
% For even order Chebyshev filters and elliptic filters,
% the desired value is sqrt(1/(1+epsilon^2)), where
% epsilon^2=10^(Ap/10)-1.
%-----------------------------------------------------
scale=1;
N=length(pol);
if (iirtyp==2 & rem(N,2)==0) | (iirtyp==4 & rem(N,2)==0)
  epsilon2=10^(Ap/10)-1;
  scale=sqrt(1/(1+epsilon2));
end
scale=scale*(prod(-pol+1))/(prod(-zer+1));

**************************************************************************

function [pole,zero,scale]=...
translow(omegap,pollow,zerlow,type,scalow)
% Given the digital filter type:
% type=1  for lowpass
% type=2  for highpass
% type=3  for bandpass
% type=4  for bandstop
% as well as the passband edges included in omegap,
% [pole, zero] = translow(omegap,pollow,zerlow,type)
% converts the poles pollow and zeros zerlow
% of a lowpass filter with passband edge
% at omega=pi/2 to the poles and zeros of the
% corresponding desired filter.

% Tapio Saramäki 20.1.1998; this program can be found in
% SUN's: ~ts/matlab/dsp/translow.m
```

```
%-------------------------------------------------------
odd=rem(length(pollow),2);
ll=(length(pollow)-1)/2+1;
for k=1:length(pollow)
  if type < 3
    pol=transsub(omegap,pollow(k),type);
    r=abs(pol);
    ang=abs(angle(pol));
    ang1=angle(pollow(k));
    sign=ang1/abs(ang1);
    pole(k)=r*exp(j*sign*abs(ang));
  end
  if type > 2
    pol=transsub(omegap,pollow(k),type);
    [Y,I]=sort(abs(angle(pol)));
    pol=pol(I);
    r=abs(pol);
    ang=abs(angle(pol));
    ang1=angle(pollow(k));
    sign=ang1/abs(ang1);
    pole(2*k-1)=r(1)*exp(j*sign*abs(ang(1)));
    pole(2*k)=r(2)*exp(j*sign*abs(ang(2)));
    if type==3 & k==ll & odd==1
      pole(2*k)=r(1)*exp(-j*sign*abs(ang(1)));
    end
  end
end
for k=1:length(zerlow)
  if type < 3
    zer=transsub(omegap,zerlow(k),type);
    r=abs(zer);
    ang=abs(angle(zer));
    ang1=angle(zerlow(k));
    sign=ang1/abs(ang1);
    zero(k)=r*exp(j*sign*abs(ang));
  end
  if type > 2
    zer=transsub(omegap,zerlow(k),type);
    angg=angle(zerlow(k));
    [Y,I]=sort(abs(angle(zer)));
    zer=zer(I);
    r=abs(zer);
    ang=abs(angle(zer));
    ang1=angle(zerlow(k));
    sign=ang1/abs(ang1);
```

```
    zero(2*k-1)=r(1)*exp(j*sign*abs(ang(1)));
    zero(2*k)=r(2)*exp(j*sign*abs(ang(2)));
    if type==4 & abs(angg-pi)< 10^(-12)
      zero(2*k)=r(1)*exp(-j*sign*abs(ang(1)));
    end
  end
end
%-----------------------------------------------
% Scaling constant
%-----------------------------------------------
scale=scalow;
Op=pi*omegap;
tp=pi/2;
if type==1;
  a=-sin((tp-Op(1))/2)/sin((tp+Op(1))/2);
end
if type==2;
  a=cos((tp+Op(1))/2)/cos((tp-Op(1))/2);
end
if type==3;
  k=cot((Op(2)-Op(1))/2)*tan(tp/2);
  a=-(k-1)/(k+1);
end
if type==4;
  k=tan((Op(2)-Op(1))/2)*tan(tp/2);
  a=(1-k)/(k+1);
end
scale=scale*prod(ones(size(zerlow))-a*zerlow);
scale=scale/prod(ones(size(pollow))-a*pollow);

**************************************************************************

function pole = transsub(omegap,pollow,type)
% Given the digital filter type:
% type=1  for lowpass
% type=2  for highpass
% type=3  for bandpass
% type=4  for bandstop
% as well as the passband edges included in omegap,
% pole = transsub(omegap,pollow,type) converts the
% pole pollow of a lowpass filter with passband edge
% at omega=pi/2 to the poles(s) of the corresponding
% desired filter. In the lowpass and highpass case,
% for the pole in the lowpass case, there exists one
% pole, whereas in the the bandpass and bandstop
```

```
% cases, there exist two poles.
% The zeros are treated in the same manner.

% Tapio Saramäki 20.11.1997; this program can be found in
% SUN's: ~ts/matlab/dsp/transsub.m
%-------------------------------------------------------
tp=0.5*pi;
Op=pi*omegap;
%-------------------------------------------------
% LOWPASS CASE
% pole=(pollow+alpha)/(1+alpha*pollow)
% alpha is the constant in the lowpass-to-lowpass
% transformation
%-------------------------------------------------
if type==1
  alpha=sin((tp-Op(1))/2)/sin((tp+Op(1))/2);
  pole=(pollow+alpha)/(1+alpha*pollow);
end
%-------------------------------------------------
% HIGHPASS CASE
% pole=-(pollow+alpha)/(1+alpha*pollow)
% alpha is the constant in the lowpass-to-highpass
% transformation
%-------------------------------------------------
if type==2
  alpha=-cos((tp+Op(1))/2)/cos((tp-Op(1))/2);
  pole=-(pollow+alpha)/(1+alpha*pollow);
end
%-------------------------------------------------
% BANDPASS CASE
% The two poles are the roots of the equation
% (1+a0*pollow)z^2+a1*(1+pollow)z+(a0+pollow)=0,
% where a0=(k-1)/(k+1) and a1=-2*alpha*k/(k+1).
% alpha and k are the constants in the lowpass-to-
% bandpass transformation.
%-------------------------------------------------
if type==3
  alpha=cos((Op(2)+Op(1))/2)/cos((Op(2)-Op(1))/2);
  k=cot((Op(2)-Op(1))/2)*tan(tp/2);
  a1=-2*alpha*k/(k+1);a0=(k-1)/(k+1);
  c(1)=1+a0*pollow;
  c(2)=a1*(1+pollow);
  c(3)=a0+pollow;
  pole=roots(c);
end
```

```
%-------------------------------------------------
% BANDSTOP CASE
% The two poles are the roots of the equation
% (1-a0*pollow)z^2+a1*(1-pollow)z+(a0-pollow)=0,
% where a0=(1-k)/(k+1) and a1=-2*alpha/(k+1).
% alpha and k are the constants in the lowpass-to-
% bandstop transformation.
%-------------------------------------------------
if type==4
  alpha=cos((Op(2)+Op(1))/2)/cos((Op(2)-Op(1))/2);
  k=tan((Op(2)-Op(1))/2)*tan(tp/2);
  a1=-2*alpha/(k+1);a0=(1-k)/(k+1);
  c(1)=1-a0*pollow;
  c(2)=a1*(1-pollow);
  c(3)=a0-pollow;
  pole=roots(c);
end

*************************************************************************
```