Technical Details on a Family of Omnidirectionally Balanced Symmetric-Antisymmetric Multiwavelets

Li Hua and James E. Fowler

Engineering Research Center
Mississippi State University

Technical Report MSSU-COE-ERC-02-08
May 2002

1. Introduction

In [1], a family of biorthogonal omnidirectionally balanced symmetric-antisymmetric (OBSA) multiwavelets is designed for use in vector wavelet transforms (VWTs). In this report, we present in-depth details on our implementation of the construction procedure of the OBSA filters. For more details on the theoretical derivation of the OBSA technique, refer to [1].

2. Construction Procedure

To find OBSA filter coefficients, we use a combination of symbolic solution via Matlab's symbolic toolbox and numerical solution via a gradient descent.

Step 1: For given filter lengths, we define coefficient matrices with matrix elements as symbolic variables; the symmetric-antisymmetric condition,

$$H_n = SH_{K_u+K_l-n}S, \qquad \widetilde{H}_n = S\widetilde{H}_{\widetilde{K}_u+\widetilde{K}_l-n}S, \tag{1}$$

where $[K_l, K_u]$ and $[\widetilde{K}_l, \widetilde{K}_u]$ are the intervals of support of the FIR multiscaling filters H_n and $\widetilde{H}_n$, respectively, and $S = \mathrm{diag}(1, -1, \ldots, (-1)^N)$, greatly reduces the number of variables.

Step 2: We impose the perfect-reconstruction condition,

$$\sum_n H_n \widetilde{H}^T_{n+2k} = \delta(k)I, \tag{2}$$

the omnidirectionally balancing condition,

$$H(1) = \widetilde{H}(1) = I, \tag{3}$$

and the vanishing-moments conditions,

$$\frac{d^k}{dz^k}H(z)\,\big|_{z=-1} = 0, \qquad \frac{d^{\tilde{k}}}{dz^{\tilde{k}}}\widetilde{H}(z)\,\big|_{z=-1} = 0, \tag{4}$$

for $k = 0, \ldots, p$ and $\tilde{k} = 0, \ldots, \tilde{p}$ (to impose $(p, \tilde{p})$ vanishing moments). We use Matlab's symbolic `solve` function to obtain symbolic solutions. Multiple solutions involving undetermined variables and complex numbers are possible. We eliminate from further consideration all complex-valued solutions as well as solutions that lead to a purely diagonal structure. We arbitrarily choose among multiple solutions that differ only in a change of sign.

Step 3: We use a gradient descent to numerically find values of undetermined variables so as to locally minimize the ideal-lowpass objective function,

$$E_{lp} = \alpha \int_0^{\omega_1} (1 - |H(\omega)|)^2 d\omega + (1-\alpha) \int_{\omega_1}^{\pi} |H(\omega)|^2 d\omega + \alpha \int_0^{\omega_1} (1 - |\widetilde{H}(\omega)|)^2 d\omega + (1-\alpha) \int_{\omega_1}^{\pi} |\widetilde{H}(\omega)|^2 d\omega, \tag{5}$$

where α is a weighting parameter, and ω_1 denotes the passband of the ideal lowpass filter. We fix $\alpha = \frac{1}{2}$, and $\omega_1 = \frac{\pi}{2}$.

Step 4: To solve for G_n and $\widetilde{G}_n$ filters, we again use Matlab's symbolic tools. SA conditions similar to (1) reduce the number of variables,

$$G_n = S G_{L_u + L_l - n} S, \qquad \widetilde{G}_n = S \widetilde{G}_{\widetilde{L}_u + \widetilde{L}_l - n} S, \tag{6}$$

where $[L_l, L_u]$ and $[\widetilde{L}_l, \widetilde{L}_u]$ are the intervals of support of the FIR multiwavelet filters G_n and $\widetilde{G}_n$, respectively. These support intervals are related to those of H_n and $\widetilde{H}_n$ via

$$L_l = -(K_u - 1), \tag{7}$$

$$\widetilde{L}_l = -(K_u - 1), \tag{8}$$

$$L_u = L_l + (\widetilde{K}_u - \widetilde{K}_l), \tag{9}$$

$$\widetilde{L}_u = \widetilde{L}_l + (K_u - K_l). \tag{10}$$

We solve the biorthogonality equations

$$\sum_n H_n \widetilde{G}^T_{n+2k} = 0, \tag{11}$$

$$\sum_n G_n \widetilde{G}^T_{n+2k} = \delta(k) I, \tag{12}$$

for symbolic solutions.

Step 5: We use a gradient descent to numerically find values of undetermined variables so as to locally minimize the ideal-highpass objective function,

$$E_{hp} = \alpha \int_0^{\omega_2} |G(\omega)|^2 d\omega + (1-\alpha) \int_{\omega_2}^{\pi} (1 - |G(\omega)|)^2 d\omega + \alpha \int_0^{\omega_2} |\widetilde{G}(\omega)|^2 d\omega + (1-\alpha) \int_{\omega_2}^{\pi} (1 - |\widetilde{G}(\omega)|)^2 d\omega, \tag{13}$$

with ω_2 denoting the stopband of the ideal highpass filter. Again, we fix $\alpha = \frac{1}{2}$, and $\omega_2 = \frac{\pi}{2}$.

3. Construction of the OBSA5-3 Filters

Using the above procedure, we have obtained OBSA filters with H_n length 5 and $\widetilde{H}_n$ length 3. In solving for these biorthogonal OBSA5-3 filters, we use $(p, \tilde{p}) = (2, 0)$ in Step 2 of the above procedure, and we eliminate from further consideration all complex-valued solutions as well as solutions that lead to a purely diagonal structure. We are left with several nondiagonal solutions that differ only in a change of sign. Arbitrarily choosing one, we have

$$H_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{8} & \frac{3}{64\gamma} \\ \gamma & \frac{1}{8} \end{bmatrix}, \tag{14}$$

$$H_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & \frac{3}{32\gamma} \\ 2\gamma & \frac{1}{2} \end{bmatrix}, \tag{15}$$

$$H_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{3}{4} & 0 \\ 0 & \frac{3}{4} \end{bmatrix}, \tag{16}$$

$$H_1 = SH_{-1}S, \tag{17}$$

$$H_2 = SH_{-2}S, \tag{18}$$

$$\widetilde{H}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & 4\gamma \\ \frac{3}{16\gamma} & \frac{1}{2} \end{bmatrix}, \tag{19}$$

$$\widetilde{H}_0 = \frac{1}{\sqrt{2}} I, \tag{20}$$

$$\widetilde{H}_1 = S\widetilde{H}_{-1}S. \tag{21}$$

In Step 3, we obtain $\gamma = \frac{\sqrt{3}}{8}$. In Step 4, we arrive at

$$G_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{0.10826939409537}{\beta} & \frac{-3}{16\beta} \\ \frac{-3}{16\zeta} & \frac{0.10823695928027}{\zeta} \end{bmatrix}, \tag{22}$$

$$G_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{-0.21653878819074}{\beta} & 0 \\ 0 & \frac{-0.21647391856054}{\zeta} \end{bmatrix}, \tag{23}$$

$$\widetilde{G}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.57726378282811\beta & \beta \\ \zeta & -0.57743676850863\zeta \end{bmatrix}, \tag{24}$$

$$\widetilde{G}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 2.30905513131246\beta & -2\beta \\ -2\zeta & 2.30974707403452\zeta \end{bmatrix}, \tag{25}$$

$$\widetilde{G}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} -3.46358269696868\beta & 0 \\ 0 & -3.46462061105178\zeta \end{bmatrix}, \tag{26}$$

$$G_2 = SG_0S, \tag{27}$$

$$\widetilde{G}_2 = S\widetilde{G}_0S, \tag{28}$$

$$\widetilde{G}_3 = S\widetilde{G}_{-1}S. \tag{29}$$

In Step 5, we find $\beta = 0.26713242893612$ and $\zeta = 0.26712267687476$.

Thus, the primary multiscaling filter coefficients are

$$H_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{8} & \frac{\sqrt{3}}{8} \\ \frac{\sqrt{3}}{8} & \frac{1}{8} \end{bmatrix}, \tag{30}$$

$$H_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{4} \\ \frac{\sqrt{3}}{4} & \frac{1}{2} \end{bmatrix}, \tag{31}$$

$$H_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{3}{4} & 0 \\ 0 & \frac{3}{4} \end{bmatrix}, \tag{32}$$

$$H_1 = SH_{-1}S, \tag{33}$$

$$H_2 = SH_{-2}S. \tag{34}$$

The dual multiscaling filter matrices are

$$\widetilde{H}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}, \tag{35}$$

$$\widetilde{H}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{36}$$

$$\widetilde{H}_1 = S\widetilde{H}_{-1}S. \tag{37}$$

The primary multiwavelet filter matrices are

$$G_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.40530232337032 & -0.70189905713332 \\ -0.70192468192399 & 0.40519569714787 \end{bmatrix}, \tag{38}$$

$$G_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.81060464674064 & 0 \\ 0 & -0.81039139429573 \end{bmatrix}, \tag{39}$$

$$G_2 = SG_0S. \tag{40}$$

The dual multiwavelet filter matrices are

$$\widetilde{G}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.15420587644373 & 0.26713242893612 \\ 0.26712267687476 & -0.15424645532993 \end{bmatrix}, \tag{41}$$

$$\widetilde{G}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.61682350577492 & -0.53426485787225 \\ -0.53424535374952 & 0.61698582131974 \end{bmatrix}, \tag{42}$$

$$\widetilde{G}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.92523525866238 & 0 \\ 0 & -0.92547873197961 \end{bmatrix}, \tag{43}$$

$$\widetilde{G}_2 = S\widetilde{G}_0S, \tag{44}$$

$$\widetilde{G}_3 = S\widetilde{G}_{-1}S. \tag{45}$$

The multiwavelet and multiscaling functions for this OBSA5-3 VWT are shown in Fig. 1.

4. Construction of the OBSA7-5 Filters

Using the above procedure, we also obtained OBSA filters of length $7/5$, with vanishing moments of order $(p, \tilde{p}) = (3, 1)$. In the symbolic solution of Step 2, we arrive at

$$H_{-3} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{9+32\lambda}{32(32\lambda-1)} & \frac{48\lambda+512\lambda^2+3}{512(32\lambda-1)\xi} \\ \xi & \lambda \end{bmatrix}, \tag{46}$$

$$H_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{7+96\lambda}{16(32\lambda-1)} & \frac{48\lambda+512\lambda^2+3}{128(32\lambda-1)\xi} \\ 4\xi & 2\lambda+\frac{1}{8} \end{bmatrix}, \tag{47}$$

$$H_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{5(-5+96\lambda)}{32(32\lambda-1)} & \frac{5(48\lambda+512\lambda^2+3)}{512(32\lambda-1)\xi} \\ 5\xi & -\lambda+\frac{1}{2} \end{bmatrix}, \tag{48}$$

$$H_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{5(-3+32\lambda)}{8(32\lambda-1)} & 0 \\ 0 & -4\lambda+\frac{3}{4} \end{bmatrix}, \tag{49}$$

$$\widetilde{H}_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} 4\lambda & 4\xi \\ \frac{48\lambda+512\lambda^2+3}{128(32\lambda-1)\xi} & \frac{9+32\lambda}{8(32\lambda-1)} \end{bmatrix}, \tag{50}$$

$$\widetilde{H}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & 8\xi \\ \frac{48\lambda+512\lambda^2+3}{64(32\lambda-1)\xi} & \frac{1}{2} \end{bmatrix}, \tag{51}$$

$$\widetilde{H}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1-8\lambda & 0 \\ 0 & \frac{96\lambda-13}{4(32\lambda-1)} \end{bmatrix}, \tag{52}$$

$$H_1 = SH_{-1}S, \tag{53}$$

$$H_2 = SH_{-2}S, \tag{54}$$

$$H_3 = SH_{-3}S, \tag{55}$$

$$\widetilde{H}_1 = S\widetilde{H}_{-1}S, \tag{56}$$

$$\widetilde{H}_2 = S\widetilde{H}_{-2}S, \tag{57}$$

while in the numerical optimization of Step 3, we obtain $\xi = 0.10519112376824$ and $\lambda = 0.13007432508989$.

For the multiwavelet filters, the symbolic solution of Step 4 yields

$$G_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{0.05472555812358}{\rho} & \frac{-0.04423654423260}{\rho} \\ \frac{-0.04423654423260}{\tau} & \frac{0.05470324933616}{\tau} \end{bmatrix}, \tag{58}$$

$$G_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{-0.05259066122941}{\rho} & \frac{0.08847308846520}{\rho} \\ \frac{0.08847308846520}{\tau} & \frac{-0.05257176749455}{\tau} \end{bmatrix}, \tag{59}$$

$$G_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{-0.00426979378833}{\rho} & 0 \\ 0 & \frac{-0.00426296368321}{\tau} \end{bmatrix}, \tag{60}$$

$$\widetilde{G}_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1.23660765742734\rho & \rho \\ \tau & -1.23711196416754\tau \end{bmatrix}, \tag{61}$$

$$\widetilde{G}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 3.66163923915863\rho & -4\rho \\ -4\tau & 3.66307495957489\tau \end{bmatrix}, \tag{62}$$

$$\widetilde{G}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} -3.51708803978847\rho & 5\rho \\ 5\tau & -3.51829216079169\tau \end{bmatrix}, \tag{63}$$

$$\widetilde{G}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 2.18411291611436\rho & 0 \\ 0 & 2.18465833076869\tau \end{bmatrix}, \tag{64}$$

$$G_2 = SG_0S, \tag{65}$$

$$G_3 = SG_{-1}S, \tag{66}$$

$$\widetilde{G}_2 = S\widetilde{G}_0S, \tag{67}$$

$$\widetilde{G}_3 = S\widetilde{G}_{-1}S, \tag{68}$$

$$\widetilde{G}_4 = S\widetilde{G}_{-2}S, \tag{69}$$

and the numeric optimization of Step 5 gives us $\rho = 0.12552236275346$ and $\tau = 0.12549181109414$.

Thus, the primary multiscaling filter coefficients are

$$H_{-3} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.13006802877092 & 0.10513373811384 \\ 0.10519112376824 & 0.13007432508989 \end{bmatrix}, \tag{70}$$

$$H_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.38513605754184 & 0.42053495245536 \\ 0.42076449507295 & 0.38514865017979 \end{bmatrix}, \tag{71}$$

$$H_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.36993197122908 & 0.52566869056920 \\ 0.52595561884119 & 0.36992567491011 \end{bmatrix}, \tag{72}$$

$$H_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.22972788491632 & 0 \\ 0 & 0.22970269964042 \end{bmatrix}, \tag{73}$$

$$H_1 = SH_{-1}S, \tag{74}$$

$$H_2 = SH_{-2}S, \tag{75}$$

$$H_3 = SH_{-3}S. \tag{76}$$

The dual multiscaling filter matrices are

$$\widetilde{H}_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.52029730035958 & 0.42076449507295 \\ 0.42053495245536 & 0.52027211508368 \end{bmatrix}, \tag{77}$$

$$\widetilde{H}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.50000000000000 & 0.84152899014590 \\ 0.84106990491072 & 0.50000000000000 \end{bmatrix}, \tag{78}$$

$$\widetilde{H}_{0} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.04059460071915 & 0 \\ 0 & -0.04054423016737 \end{bmatrix}, \tag{79}$$

$$\widetilde{H}_{1} = S\widetilde{H}_{-1}S, \tag{80}$$

$$\widetilde{H}_{2} = S\widetilde{H}_{-2}S. \tag{81}$$

The primary multiwavelt filter matrices are

$$G_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.43598253668206 & -0.35241962676793 \\ -0.35250542522982 & 0.43591090812388 \end{bmatrix}, \tag{82}$$

$$G_{0} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.41897443671220 & 0.70483925353586 \\ 0.70501085045965 & -0.41892588079006 \end{bmatrix}, \tag{83}$$

$$G_{1} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.03401619993973 & 0 \\ 0 & -0.03397005466764 \end{bmatrix}, \tag{84}$$

$$G_{2} = SG_{0}S, \tag{85}$$

$$G_{3} = SG_{-1}S. \tag{86}$$

The dual multiwavelet filter matrices are

$$\widetilde{G}_{-2} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.15522191495930 & 0.12552236275346 \\ 0.12549181109414 & -0.15524742090962 \end{bmatrix}, \tag{87}$$

$$\widetilde{G}_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.45961760884996 & -0.50208945101383 \\ -0.50196724437657 & 0.45968591085066 \end{bmatrix}, \tag{88}$$

$$\widetilde{G}_{0} = \frac{1}{\sqrt{2}} \begin{bmatrix} -0.44147320076617 & 0.62761181376729 \\ 0.62745905547072 & -0.44151685521608 \end{bmatrix}, \tag{89}$$

$$\widetilde{G}_{1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.27415501375102 & 0 \\ 0 & 0.27415673055007 \end{bmatrix}, \tag{90}$$

$$\widetilde{G}_{2} = S\widetilde{G}_{0}S, \tag{91}$$

$$\widetilde{G}_{3} = S\widetilde{G}_{-1}S, \tag{92}$$

$$\widetilde{G}_{4} = S\widetilde{G}_{-2}S. \tag{93}$$

The multiwavelet and multiscaling functions for this OBSA7-5 VWT are shown in Fig. 2.

5. Matlab Code

- The MATLAB program for the design of biorthogonal OBSA filter coefficients with a general structure is as following.

```
function OBSA(M,N,StartH,StartHT,DerM,DerN,GDerM,GDerN,symflag,s1,s2,inivar)

%
%  OBSA(M,N,StartH,StartHT,DerM,DerN,GDerM,GDerN,symflag,s1,s2,inivar)
%
%    For general omnidirectionally balanced multiwavelets design.
%
%    M: length of basis lowpass filter H
%    N: length of dual lowpass filter H~,
%       M and N must be either both even or both odd
%    StartH: the start index of H's nonzero coefficient
%    StartHT: the start index of H~'s nonzero coefficient
%    DerM: the vanishing moments required for H
%    DerN: the vanishing moments required for H~
%    GDerM: the vanishing moments required for G
%    GDerN: the vanishing moments required for G~
%    symflag: flag indicates the desired symmetry
%             -1 - symmetric-antisymmetric
%              1 - symmetric
%    s1: the index of which result to select in solutions of H, H~
%    s2: the index of which to select in solutions of G, G~
%    inivar: the initial value of gradient descending search method
%

%
% NOTE:
% RELATION BETWEEN FILTER SUPPORT INTERVALS AND VARIABLES USED HERE:
%   StartH = K_l;
%   StartHT = \tilde{K}_l;
%   M = K_u - K_l + 1;
%   N = \tilde{K}_u - \tilde{K}_l + 1;
%   StartG = L_l;
%   StartGT = \tilde{L}_l;
%   GM = L_u - L_l + 1;
%   GN = \tilde{L}_u - \tilde{L}_l + 1;
%

if mod(M-N,2) ~= 0
   disp('The length of basis and dual filters must be both even or both odd.');
end

Sr = [1 0; 0 symflag]
I = eye(2);
O = zeros(2);
J = [0 1; 1 0];

%
% SYMOBLIC SET UP OF MATRIX COEFFICIENTS, SA CONDITION
%

for l = 1:ceil((M-1)/2)
   index = 4*(l-1);
```

```
   eval(['syms x' num2str(index+1) ...
         ' x' num2str(index+2) ...
         ' x' num2str(index+3) ...
         ' x' num2str(index+4)]);
   eval(['P' num2str(l-1) ...
         ' = [x' num2str(index+1) ...
         ' x' num2str(index+2) ...
         '; x' num2str(index+3) ...
         ' x' num2str(index+4) ...
             ']']);
   eval(['P' num2str(M-l) ...
         ' = Sr*P' num2str(l-1) ...
         '*Sr']);
end
indexP = index+4;
if mod(M,2) ~= 0
   index = 2*(M-1);
   eval(['syms x' num2str(index+1) ...
         ' x' num2str(index+2)]);
   eval(['P' num2str((M-1)/2) ...
         ' = [x' num2str(index+1) ' 0;' ...
         ' 0 x' num2str(index+2) ']']);
   indexP = index+2;
end

for l = 1:ceil((N-1)/2)
   index = 4*(l-1);
   eval(['syms y' num2str(index+1) ...
         ' y' num2str(index+2) ...
         ' y' num2str(index+3) ...
         ' y' num2str(index+4)]);
   eval(['Q' num2str(l-1) ...
         ' = [y' num2str(index+1) ...
         ' y' num2str(index+2) ...
         '; y' num2str(index+3) ...
         ' y' num2str(index+4) ...
             ']']);
   eval(['Q' num2str(N-l) ...
         ' = Sr*Q' num2str(l-1) ...
         '*Sr']);
end
indexQ = index+4;
if mod(N,2) ~= 0
   index = 2*(N-1);
   eval(['syms y' num2str(index+1) ...
         ' y' num2str(index+2)]);
   eval(['Q' num2str((N-1)/2) ...
         ' = [y' num2str(index+1) ' 0;' ...
         ' 0 y' num2str(index+2) ']']);
   indexQ = index+2;
end


%
% DESIGN CONSTRAINTS, PR, OB, VANISHING MOMENTS
%

syms z;
```

```
P = 0;
for l = 0:M-1
   eval(['P = P+P' num2str(l) '*(z^' num2str(l+StartH) ');']);
end
P = simple(P/2)

Q = 0;
for l = 0:N-1
  eval(['Q = Q + Q' num2str(l) '*(z^' num2str(l+StartHT) ');']);
end
Q = simple(Q/2)

Dis = max(StartH+M-1, StartHT+N-1) - min(StartH, StartHT) + 1;
Num = ceil(Dis/2);

for l = 1-Num:Num-1

  if l == 0
     eval(['E1' num2str(l+Num) '= -2*I;']);
  else
     eval(['E1' num2str(l+Num) ' = 0;']);
  end

  for k = 0:M-1
      note = StartH+k+2*l-StartHT;
      if note < N & note >= 0
          str = ['Q' num2str(note)];
      else
          str = 'O';
      end
      eval(['E1' num2str(l+Num) ' = E1' num2str(l+Num) ...
            '+ P' num2str(k) '*' str '.'';']);
  end

      eval(['E1' num2str(l+Num) ' = simple(E1' num2str(l+Num) ')']);

      if l == 1-Num
         eval(['eqns = matrixtoeqn(E11);']);
      else
         eval(['eqns = [eqns '','' matrixtoeqn(E1' num2str(l+Num) ')];']);
      end
end

E2 =  subs(P,'z',1) - I;
E2 = simple(E2)
eqns = [eqns ',' matrixtoeqn(E2)];

for l = 0:DerM
   eval(['E3' num2str(l) '= subs(diff(P,''z'',' num2str(l) ...
         '),''z'',-1);']);
   eval(['E3' num2str(l) '= simple(E3' num2str(l) ')']);
   eval(['eqns = [eqns '','' matrixtoeqn(E3' num2str(l) ')];']);
end

E4 = subs(Q,'z',1) - I;
E4 = simple(E4)
eqns = [eqns ',' matrixtoeqn(E4)];

for l = 0:DerN
```

```
   eval(['E5' num2str(l) '= subs(diff(Q,''z'',' num2str(l) ...
         '),''z'',-1);']);
   eval(['E5' num2str(l) '= simple(E5' num2str(l) ')']);
   eval(['eqns = [eqns '','' matrixtoeqn(E5' num2str(l) ')];']);
end


%
% SOLVING EQUATIONS
%

strg = 'cel = {';

for l = 1:indexP
   strg = [strg ' ''x' num2str(l) ''''];
end
for l = 1:indexQ
   strg = [strg ' ''y' num2str(l) ''''];
end
strg = [strg '};'];
eval(strg);
cel = sort(cel);

str = [];
for l = 1:length(cel)
   if l ~= 1
      str = [str ','];
   end
   str = [str cel{l}];
end

eval(['[' str '] = solve(eqns,' str ')']);


%
% SELECT SOLUTION
%

for l = 1:indexP
    eval(['X' num2str(l) '=x' num2str(l) '(s1);']);
end
for l = 1:indexQ
    eval(['Y' num2str(l) '=y' num2str(l) '(s1);']);
end

for l = 1:ceil((M-1)/2)
   index = 4*(l-1);
   eval(['P' num2str(l-1) ...
         ' = [X' num2str(index+1) ...
         ' X' num2str(index+2) ...
         '; X' num2str(index+3) ...
         ' X' num2str(index+4) ...
              ']']);
   eval(['P' num2str(M-l) ...
         ' = Sr*P' num2str(l-1) ...
         '*Sr']);
end
if mod(M,2) ~= 0
   index = 2*(M-1);
```

```
   eval(['P' num2str((M-1)/2) ...
         ' = [X' num2str(index+1) ' 0;' ...
         ' 0 X' num2str(index+2) ']']);
end

for l = 1:ceil((N-1)/2)
   index = 4*(l-1);
   eval(['Q' num2str(l-1) ...
         ' = [Y' num2str(index+1) ...
         ' Y' num2str(index+2) ...
         '; Y' num2str(index+3) ...
         ' Y' num2str(index+4) ...
             ']']);
   eval(['Q' num2str(N-l) ...
         ' = Sr*Q' num2str(l-1) ...
         '*Sr']);
end
if mod(N,2) ~= 0
   index = 2*(N-1);
   eval(['Q' num2str((N-1)/2) ...
         ' = [Y' num2str(index+1) ' 0;' ...
         ' 0 Y' num2str(index+2) ']']);
end

P = 0;
for l = 0:M-1
   eval(['P = P+P' num2str(l) '*(z^' num2str(l+StartH) ');']);
end
P = simple(P/2)
Q = 0;
for l = 0:N-1
  eval(['Q = Q + Q' num2str(l) '*(z^' num2str(l+StartHT) ');']);
end
Q = simple(Q/2)

%
% FILTER OPTIMIZATION
%

[cel, res] = FilterOptimize(P, Q, 0, inivar)

for l = 1:length(cel)
     eval([cel{l} '= res(l)'])
end

H = [];
for l = 0:M-1
   eval(['P' num2str(l) ' = simple(subs(P' num2str(l) '));'], ...
        ['P' num2str(l) ' = double(subs(P' num2str(l) '));']);
   eval(['H = [H; P' num2str(l) '];']);
end
HT = [];
for l = 0:N-1
   eval(['Q' num2str(l) ' = simple(subs(Q' num2str(l) '));'], ...
        ['Q' num2str(l) ' = double(subs(Q' num2str(l) '));']);
   eval(['HT = [HT; Q' num2str(l) '];']);
end

if (~isreal(H) | ~isreal(HT))
```

```
     H
     HT
     error('Complex result obtained');
     return;
end

double(H)
double(HT)

clear eqns;

%
% DESIGNING HIGHPASS FILTERS G & G~
%

StartG = -(StartHT+N-1-1);
GM = N;
StartGT = -(StartH+M-1-1);
GN = M;

for l = 1:ceil((GM-1)/2)
   index = 4*(l-1);
   eval(['syms r' num2str(index+1) ...
         ' r' num2str(index+2) ...
         ' r' num2str(index+3) ...
         ' r' num2str(index+4)]);
   eval(['GP' num2str(l-1) ...
         ' = [r' num2str(index+1) ...
         ' r' num2str(index+2) ...
         '; r' num2str(index+3) ...
         ' r' num2str(index+4) ...
             ']']);
   eval(['GP' num2str(GM-l) ...
         ' = Sr*GP' num2str(l-1) ...
         '*Sr']);
end
indexP = index+4;
if mod(GM,2) ~= 0
   index = 2*(GM-1);
   eval(['syms r' num2str(index+1) ...
         ' r' num2str(index+2)]);
   eval(['GP' num2str((GM-1)/2) ...
         ' = [r' num2str(index+1) ' 0;' ...
         ' 0 r' num2str(index+2) ']']);
   indexP = index+2;
end

for l = 1:ceil((GN-1)/2)
   index = 4*(l-1);
   eval(['syms t' num2str(index+1) ...
         ' t' num2str(index+2) ...
         ' t' num2str(index+3) ...
         ' t' num2str(index+4)]);
   eval(['GQ' num2str(l-1) ...
         ' = [t' num2str(index+1) ...
         ' t' num2str(index+2) ...
         '; t' num2str(index+3) ...
         ' t' num2str(index+4) ...
             ']']);
```

```
   eval(['GQ' num2str(GN-l) ...
         ' = Sr*GQ' num2str(l-1) ...
         '*Sr']);
end
indexQ = index+4;
if mod(GN,2) ~= 0
   index = 2*(GN-1);
   eval(['syms t' num2str(index+1) ...
         ' t' num2str(index+2)]);
   eval(['GQ' num2str((GN-1)/2) ...
         ' = [t' num2str(index+1) ' 0;' ...
         ' 0 t' num2str(index+2) ']']);
   indexQ = index+2;
end


%
% DESIGN CONSTRAINTS, ORTHOGONALITY, VANISHING MOMENTS
%

GP = 0;
for l = 0:GM-1
   eval(['GP = GP+GP' num2str(l) '*(z^' num2str(l+StartG) ');']);
end
GP = simple(GP/2)

GQ = 0;
for l = 0:GN-1
  eval(['GQ = GQ + GQ' num2str(l) '*(z^' num2str(l+StartGT) ');']);
end
GQ = simple(GQ/2)

Dis = max(StartH+M-1, StartGT+GN-1) - min(StartH, StartGT) + 1;
Num = ceil(Dis/2);
eqns = [];

for l = 1-Num:Num-1

  eval(['E1' num2str(l+Num) ' = 0;']);

  for k = 0:M-1
      note = StartH+k+2*l-StartGT;
      if note < GN & note >= 0
          str = ['GQ' num2str(note)];
      else
          str = 'O';
      end
      eval(['E1' num2str(l+Num) ' = E1' num2str(l+Num) ...
                '+ P' num2str(k) '*' str '.'';']);
  end

      flag = eval(['isequal(E1' num2str(l+Num) ', zeros(2,2))']);

      if ~flag
        eval(['E1' num2str(l+Num) ' = simple(E1' num2str(l+Num) ')'], ...
             ['E1' num2str(l+Num) ]);

        if isempty(eqns)
            eval(['eqns = matrixtoeqn(E1' num2str(l+Num) ');']);
```

```
         else
            eval(['eqns = [eqns '','' matrixtoeqn(E1' num2str(l+Num) ')];']);
        end
      end
end


Dis = max(StartG+GM-1, StartGT+GN-1) - min(StartG, StartGT) + 1;
Num = ceil(Dis/2);

for l = 1-Num:Num-1

  if l == 0
     eval(['E2' num2str(l+Num) ' = -2*I;']);
  else
     eval(['E2' num2str(l+Num) ' = 0;']);
  end

  for k = 0:GM-1
      note = StartG+k+2*l-StartGT;
      if note < GN & note >= 0
          str = ['GQ' num2str(note)];
      else
          str = 'O';
      end
      eval(['E2' num2str(l+Num) ' = E2' num2str(l+Num) ...
                 '+ GP' num2str(k) '*' str '.'';']);
  end

      eval(['E2' num2str(l+Num) ' = simple(E2' num2str(l+Num) ')'], ...
           ['E2' num2str(l+Num) ]);

      eval(['eqns = [eqns '','' matrixtoeqn(E2' num2str(l+Num) ')];']);
end

for l = 0:GDerM
   eval(['E4' num2str(l) '= subs(diff(GP,''z'',' num2str(l) ...
         '),''z'',-1);']);
   eval(['E4' num2str(l) '= simple(E4' num2str(l) ')']);
   eval(['eqns = [eqns '','' matrixtoeqn(E4' num2str(l) ')];']);
end

for l = 0:GDerN
   eval(['E5' num2str(l) '= subs(diff(GQ,''z'',' num2str(l) ...
         '),''z'',-1);']);
   eval(['E5' num2str(l) '= simple(E5' num2str(l) ')']);
   eval(['eqns = [eqns '','' matrixtoeqn(E5' num2str(l) ')];']);
end

%
% SOLVING EQUATIONS
%

strg = 'cel = {';

for l = 1:indexP
   strg = [strg ' ''r' num2str(l) ''''];
end
for l = 1:indexQ
```

```
   strg = [strg ' ''t' num2str(l) ''''];
end
strg = [strg '};'];
eval(strg);
cel = sort(cel);

str = [];
for l = 1:length(cel)
   if l ~= 1
      str = [str ','];
   end
   str = [str cel{l}];
end

eval(['[' str '] = solve(eqns,' str ')']);


%
% SELECT SOLUTION
%

for l = 1:indexP
    eval(['R' num2str(l) '= r' num2str(l) '(s2);']);
end
for l = 1:indexQ
    eval(['T' num2str(l) '= t' num2str(l) '(s2);']);
end

for l = 1:ceil((GM-1)/2)
   index = 4*(l-1);
   eval(['GP' num2str(l-1) ...
         ' = [R' num2str(index+1) ...
         ' R' num2str(index+2) ...
         '; R' num2str(index+3) ...
         ' R' num2str(index+4) ...
             ']']);
   eval(['GP' num2str(GM-l) ...
         ' = Sr*GP' num2str(l-1) ...
         '*Sr']);
end
if mod(GM,2) ~= 0
   index = 2*(GM-1);
   eval(['GP' num2str((GM-1)/2) ...
         ' = [R' num2str(index+1) ' 0;' ...
         ' 0 R' num2str(index+2) ']']);
end

for l = 1:ceil((GN-1)/2)
   index = 4*(l-1);
   eval(['GQ' num2str(l-1) ...
         ' = [T' num2str(index+1) ...
         ' T' num2str(index+2) ...
         '; T' num2str(index+3) ...
         ' T' num2str(index+4) ...
             ']']);
   eval(['GQ' num2str(GN-l) ...
         ' = Sr*GQ' num2str(l-1) ...
         '*Sr']);
end
```

```
if mod(GN,2) ~= 0
   index = 2*(GN-1);
   eval(['GQ' num2str((GN-1)/2) ...
         ' = [T' num2str(index+1) ' 0;' ...
         ' 0 T' num2str(index+2) ']']);
end

GP = 0;
for l = 0:GM-1
   eval(['GP = GP+GP' num2str(l) '*(z^' num2str(l+StartG) ');']);
end
GP = simple(GP/2)

GQ = 0;
for l = 0:GN-1
  eval(['GQ = GQ + GQ' num2str(l) '*(z^' num2str(l+StartGT) ');']);
end
GQ = simple(GQ/2)

%
% FILTER OPTIMIZATION
%

[cel, res] = FilterOptimize(GP, GQ, 1, inivar)

for l = 1:length(cel)
     eval([cel{l} '= res(l)'])
end

G = [];
for l = 0:GM-1
   eval(['GP' num2str(l) ' = simple(subs(GP' num2str(l) '));'], ...
        ['GP' num2str(l) ' = double(subs(GP' num2str(l) '));']);
   eval(['G = [G; GP' num2str(l) '];']);
end
GT = [];
for l = 0:GN-1
   eval(['GQ' num2str(l) ' = simple(subs(GQ' num2str(l) '));'], ...
        ['GQ' num2str(l) ' = double(subs(GQ' num2str(l) '));']);
   eval(['GT = [GT; GQ' num2str(l) '];']);
end

if (~isreal(G) | ~isreal(GT))
     G
     GT
     error('Complex result obtained');
     return;
end


%
%  SAVE IN FILES - QccPack vfb format
%
format long

st = min([StartG, StartGT, StartHT, StartH]);

ID='VFB0.17';
Head = 1;
```

```
H = [zeros((StartH-st)*2,2); H]
H = double(H./sqrt(2));
H1 = [2;size(H,1)/2;0];

G = [zeros((StartG-st)*2,2);  G]
G = double(G./sqrt(2));
G1 = [2;size(G,1)/2;0];

HT = [zeros((StartHT-st)*2,2); HT]
HT = double(HT./sqrt(2));
H2 = [2;size(HT,1)/2;0];

GT = [zeros((StartGT-st)*2,2); GT]
GT = double(GT./sqrt(2));
G2 = [2;size(GT,1)/2;0];

format
fid=fopen('log.vfb','w');
fprintf(fid,'%s\n\n',ID);
fprintf(fid,'%d\n\n',Head);
fprintf(fid,'%d\n',H1);
fprintf(fid,' %.16e %.16e\n %.16e %.16e\n',H');
fprintf(fid,'\n\n');
fprintf(fid,'%d\n',G1);
fprintf(fid,' %.16e %.16e\n %.16e %.16e\n',G');
fprintf(fid,'\n\n');
fprintf(fid,'%d\n',H2);
fprintf(fid,' %.16e %.16e\n %.16e %.16e\n',HT');
fprintf(fid,'\n\n');
fprintf(fid,'%d\n',G2);
fprintf(fid,' %.16e %.16e\n %.16e %.16e\n',GT');
fclose(fid);

save 'log.mat';
return;
```

- The gradient-descent search for numeric filter optimization is implemented in the following MATLAB function.

```
function [cel, result] = FilterOptimize(H, HT, LorH, inivar)

%
%  FilterOptimize(H, HT, LorH, option, inivar)
%
%    The optimization on the vector filters, through the gradient
%    decending search algorithm.
%
%    H: refinement mask of H or G, H(z)=sum{P_k*z^k}/2;
%    HT: refinement mask of H~ or G~, H(z)=sum{Q_k*z^k}/2;
%    LorH: flag indicates if the filter is lowpass or highpass
%          0 - lowpass filter
%          1 - highpass filter
%    inivar: the initial value in search algorithm
%

syms w real;

HW = subs(H,'z',exp(-i*w))
HTW = subs(HT,'z',exp(-i*w))

cel = FindUnknowVar(HW, HTW)
if isempty(cel)
     return;
end

initial = inivar*ones(length(cel),1);
min = initial;
grad = initial;
step = 0.001;
count = 0;
dfactor = 1;
dcount = 0;
accuracy = 0.001;
stopcount = 40;
lastgrad = grad;

while (1)
   initial = min;

   for k = 1:length(cel)

     % NUMERICAL INTEGRAL
     grad(k) = 0;
     point = initial;
     point(k) = point(k) - 2*step;
     f1 = NumerIntegral(HW,HTW,cel,point,LorH);
     point = initial;
     point(k) = point(k) - 1*step;
     f2 = NumerIntegral(HW,HTW,cel,point,LorH);
     point = initial;
     point(k) = point(k) + 1*step;
     f3 = NumerIntegral(HW,HTW,cel,point,LorH);
     point = initial;
     point(k) = point(k) + 2*step;
     f4 = NumerIntegral(HW,HTW,cel,point,LorH);
```

```
      grad(k) = (-f4+8*f3-8*f2+f1)/(12*step);
    end

    grad
    if (norm(grad) < accuracy)
      initial = initial + grad;
      break;
    elseif (norm(grad) > 30)
      grad = grad/norm(grad)/5/dfactor;
    elseif (norm(grad) > 10)
      grad = grad/norm(grad)/10/dfactor;
    elseif (norm(grad) > 1)
      grad = grad/norm(grad)/20/dfactor;
    elseif (norm(grad) > 0.5)
      grad = grad/40/dfactor;
    elseif (norm(grad) > 0.1)
      grad = grad/50/dfactor;
    elseif (norm(grad) > 0.01)
      grad = grad/60/dfactor;
    elseif (norm(grad)>0.001)
      grad = grad/100/dfactor;
    else
      grad = grad/10;
    end

    min = initial - grad
    for k = 1:length(grad)
      if (grad(k)*lastgrad(k) > 0)
         break;
      end
      if (k == length(grad))
        dcount = dcount + 1;
      end
    end

    if (dcount > 2)
      dfactor = dfactor*2;
      dcount = 0;
    end
      lastgrad = grad;

    count = count + 1;

    if count > stopcount
       min = (initial + min)/2;
       break;
    end
  end

  result = min;
```

- Numeric integration as used in the numeric filter optimization is implemented in the following MATLAB function.

```
function res = NumerIntegral(HW,HTW,cel,point,LorH)

%  NumerIntegral(HW,HTW,cel,point,LorH)
%    Conduct numerical approximation of the integral
%    of HW and HTW.
%

alpha = 1/2;
doubleu = pi/2;

eval(['HW = subs(HW, cel, mat2cell(point));'], ...
     ['HW;']);
eval(['HTW = subs(HTW, cel, mat2cell(point));'], ...
     ['HTW;']);

if LorH == 0

   % LOWPASS VECTOR FILTER
   res = 0;
   for index = 1:4
     save temp.mat HW HTW;
     if mod(index,2) == 1
       eval(['res = res + alpha*quad8(''DiffNorm' ...
                  num2str(index) ''', 0, doubleu);']);
     else
       eval(['res = res + (1-alpha)*quad8(''DiffNorm' ...
                  num2str(index) ''', doubleu, pi);']);
     end
   end
else
   % HIGHPASS VECTOR FILTER
   res = 0;
   for index = 1:4
     save temp.mat HW HTW;
     if mod(index,2) == 1
       eval(['res = res + alpha*quad8(''DiffNorm' ...
                  num2str(index) ''', doubleu, pi);']);
     else
       eval(['res = res + (1-alpha)*quad8(''DiffNorm' ...
                  num2str(index) ''', 0, doubleu);']);
     end
   end
end
```

- Various utility routines
  - FindUnknowVar

```
function cel = FindUnknowVar(varargin)

%
%  function cel = FindUnknowVar(...)
%
%    Find out all the unknown variables in the input functions
%    and ouput the result into a cell.
%

num = nargin;
unknownvar = 'w, inf';

strg = 'cel = {';

for l = 1:num

  fun = varargin{l};
  unknown = findsym(fun);

  while ~isempty(unknown)
     [var, unknown] = strtok(unknown, ', ');
     if isempty(findstr(unknownvar, var))
        strg = [strg ' ''' var '''];
        unknownvar = [unknownvar var];
     end
  end

end

strg = [strg '};'];
eval(strg);
```

  - DiffNorm1

```
function res = DiffNorm1(var)

load temp.mat;

for l = 1:length(var)
     w = var(l);
     for m = 1:size(HW,1)
       for n = 1:size(HW,2)
         temp(m,n) = eval(char(HW(m,n)));
         temp(m,n) = abs(temp(m,n));
       end
     end

     res(l) = (1-norm(temp,'fro')/sqrt(2))^2;

end
```

– DiffNorm2

```
function res = DiffNorm2(var)

load temp.mat;

for l = 1:length(var)
     w = var(l);
     for m = 1:size(HW,1)
       for n = 1:size(HW,2)
         temp(m,n) = eval(char(HW(m,n)));
         temp(m,n) = abs(temp(m,n));
       end
     end

     res(l) = (norm(temp,'fro')/sqrt(2))^2;

end
```

– DiffNorm3

```
function res = DiffNorm3(var)

load temp.mat;

for l = 1:length(var)
     w = var(l);
     for m = 1:size(HTW,1)
       for n = 1:size(HTW,2)
         temp(m,n) = eval(char(HTW(m,n)));
         temp(m,n) = abs(temp(m,n));
       end
     end

     res(l) = (1-norm(temp,'fro')/sqrt(2))^2;

end
```

– DiffNorm4

```
function res = DiffNorm4(var)

load temp.mat;

for l = 1:length(var)
     w = var(l);
     for m = 1:size(HTW,1)
       for n = 1:size(HTW,2)
         temp(m,n) = eval(char(HTW(m,n)));
         temp(m,n) = abs(temp(m,n));
       end
     end

     res(l) = (norm(temp,'fro')/sqrt(2))^2;

end
```

– matrixtoeqn

```
function eqn = matrixtoeqn(A)

[M, N] = size(A);

eqn = '';

for m = 1:M
  for n = 1:N
    if ((m == 1) & (n == 1))
     if (char(A(m,n)) ~= 0)
      eqn = char(A(m, n));
     end
    else
     if (char(A(m,n)) ~= 0)
      eqn = [eqn ', ' char(A(m, n))];
     end
    end
  end
end
```

– str2cell

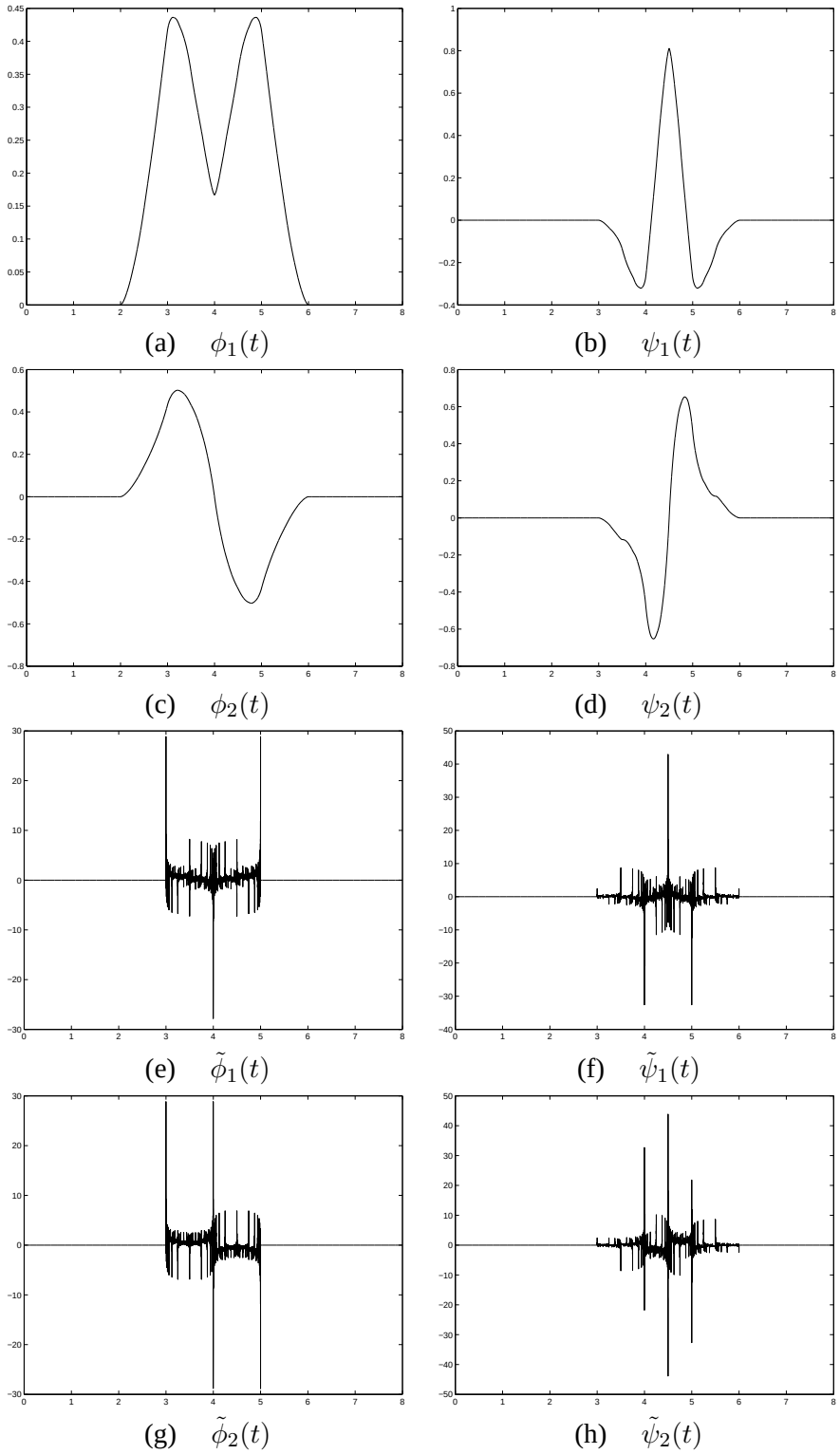
```
function cel = str2cell(str)

strg = 'cel = {';

while ~isempty(str)
     [var, str] = strtok(str, ', ');
     strg = [strg ' ''' var '''];
end
strg = [strg '};'];
eval(strg)
```

References

[1] J. E. Fowler and L. Hua, "Wavelet transforms for vector fields using omnidirectionally balanced multiwavelets," *IEEE Transactions on Signal Processing*, May 2002, submitted.



(a) $\phi_1(t)$ (b) $\psi_1(t)$

(c) $\phi_2(t)$ (d) $\psi_2(t)$

(e) $\tilde{\phi}_1(t)$ (f) $\tilde{\psi}_1(t)$

(g) $\tilde{\phi}_2(t)$ (h) $\tilde{\psi}_2(t)$

Figure 1: The multiscaling and multiwavelet functions of our OBSA5-3 VWT with vanishing moments $(2, 0)$.

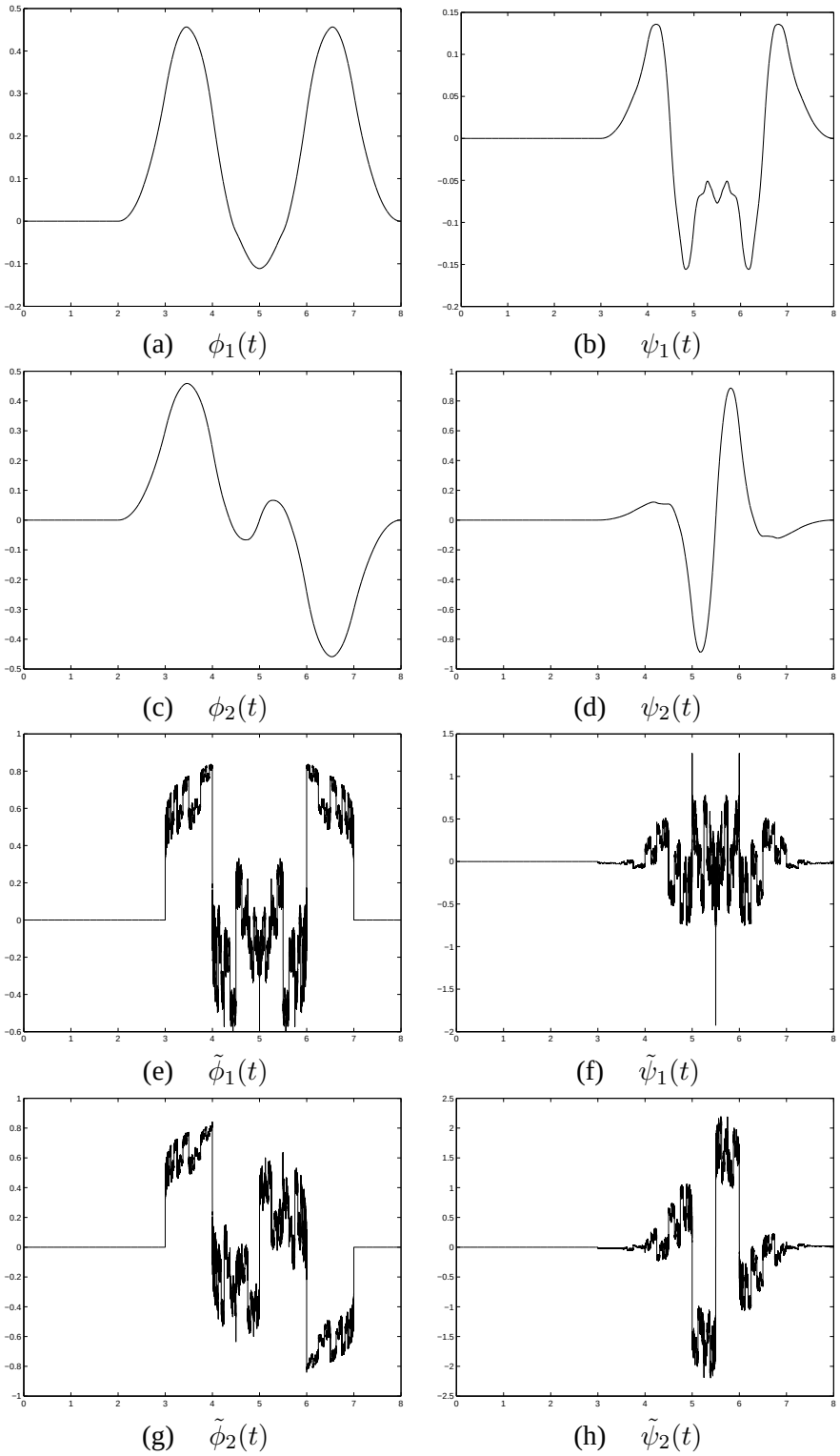


Figure 2: The multiscaling and multiwavelet functions of our OBSA7-5 VWT with vanishing moments $(3, 1)$.