```
0001 function A= CDV_Bfilters(wname, request, FTYPE)
0002
0003 % CDV_BFILTERS Wavelet Boundary Filters.
0004 %   returns the internal low/high-pass filters, and corresponding
0005 %   edge low/high-pass filters and the Pre/Post filters. 'wname'
0006 %   is a string containing the wavelet name.(see WFILTERS for more
0007 %   information) and 'request' is a string, 'lowpass', 'highpass',
0008 %   'precondition' or 'postcondition', for desired computations.
0009 %   'FTYPE' is a string for edge filter type, 'TYPE-1', 'TYPE-2'
0010 %   or 'TYPE-3' (see reference [1] for these variation details).
0011 %   This routine is inspired by works of Chyzak et al [2].
0012 %
0013 %   A = CDV_BFILTERS('wname', 'LOWPASS', FTYPE) computes
0014 %   and returns the low-pass filter structure in
0015 %   A={ A.inner, A.left and A.right}.
0016 %
0017 %   A = CDV_BFILTERS('wname', 'HIGHPASS', FTYPE) computes
0018 %   and returns the high-pass filter structure in
0019 %   A={ A.inner, A.left and A.right}.
0020 %
0021 %   A = CDV_BFILTERS('wname', 'PRECONDITION', FTYPE) computes
0022 %   and returns the pre-conditioning matrix structure in
0023 %   A={ A.left and A.right}.
0024 %
0025 %   A = CDV_BFILTERS('wname', 'POSTCONDITION', FTYPE) computes
0026 %   and returns the post-conditioning matrix structure in
0027 %   A={ A.left and A.right}.
0028 %
0029 %   See also: WFILTERS
0030 %
0031 %   Reference:
0032 %   [1] W. S. Lee, A. A. Kassim, "Image Coding with Edge Preservation using
0033 %   Boundary Wavelets", To Be Submitted, 2004.
0034 %
0035 %   [2] F. Chyzak, P. Paule, O. Scherzer, A. Schoisswohl and B. Zimmermann,
0036 %   "The Construction of Orthonormal Wavelets using Symbolic Methods and a
0037 %   Matrix Analytical Approach for Wavelets on the Interval", Experimental
0038 %   Mathematics., vol. 10, no. 1, pp. 67-86, 2001.
0039 %---------------------------------------------------------------------------
0040 %   Copyright 2004 Lee Wei Siong
0041 %   $Revision: 0.3 $  $Date: 2004/07/01 $
0042 %
0043 %   This program is free software; you can redistribute it and/or modify
0044 %   it under the terms of the GNU General Public License as published by
0045 %   the Free Software Foundation; either version 2 of the License, or
0046 %   (at your option) any later version.
0047 %
0048 %   This program is distributed in the hope that it will be useful,
0049 %   but WITHOUT ANY WARRANTY; without even the implied warranty of
0050 %   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
0051 %   GNU General Public License for more details.
0052 %
0053 %   You should have received a copy of the GNU General Public License
0054 %   along with this program; if not, write to:
0055 %     Free Software  Foundation, Inc.,
0056 %     59 Temple Place, Suite 330,
```

```
0057 %     Boston, MA 02111-1307 USA
0058 %--------------------------------------------------------------------------
0059 %--------------------------------------------------------------------------
0060 % Version History:
0061 %   Added: Type-0, for Meyer's construction.
0062 %--------------------------------------------------------------------------
0063
0064 if (nargin~=3)
0065    error('usage A = CDV_Bfilters( wname, request, ftype)');
0066    A=[];
0067    return;
0068 end;
0069
0070 [h,g] = wfilters(wname,'d');           % get the standard wavelet filters
0071 N = length(h)/2;                       % # of moments, assumption made.
0072 T = T_matrix(N, FTYPE)                 % transform matrix
0073
0074 H_LR = cell(2,1);
0075 H_LR{1} = H_matrix(fliplr(h), N, FTYPE);   % half-space matrix, left
0076 H_LR{2} = H_matrix(h, N, FTYPE);           % half-space matrix, right
0077
0078 A0_L = A0_matrix(H_LR{1}, N, FTYPE);
0079 A0_R = A0_matrix(H_LR{2}, N, FTYPE);
0080
0081
0082 A_LR = cell(2,1);
0083
0084 A_LR{1} = inv(chol(T*A0_L*T')')           % orthonormalizing matrix, left
0085 A_LR{2} = inv(chol(T*A0_R*T')');           % orthonormalizing matrix, right
0086
0087 %fprintf('Condition Number Left: %f, base loss:%f\n',cond(T*A0_L*T'),log10(cond(T*A0_L*T')));
0088 %min(min((T*A0_L*T')))
0089 %max(max((T*A0_L*T')))
0090 %fprintf('Condition Number Right: %f, base loss:%f\n',cond(T*A0_R*T'),log10(cond(T*A0_R*T')));
0091 %min(min((T*A0_R*T')))
0092 %max(max((T*A0_R*T')))
0093 switch lower(request)
0094    case 'lowpass'
0095       %first the left border
0096       [K1, K2] = size(T);
0097       AK = [size(A_LR{1}); size(A_LR{2})];
0098       mb = cell(2,1);
0099
0100       switch lower(FTYPE)
0101          case {'type-i', 'type-1'}
0102             T_ext = [T zeros(K1,2*N-1);
0103                      zeros(2*N-1, K2) eye(2*N-1)];
0104                %size(T_ext)
0105             for m = 1 :1 :2
0106                A_ext = [A_LR{m} zeros(AK(m,1),2*N-1);
0107                         zeros(2*N-1, AK(m,2)) eye(2*N-1)];
0108                mb{m} = A_LR{m}*T*H_LR{m}*(pinv(T_ext))*(inv(A_ext));
0109             end;
0110
0111          case {'type-ii', 'type-2'}
0112                T_ext = [T zeros(K1, 2*N-2);
```

```
0113                    zeros(2*N-2, K2) eye(2*N-2)];
0114                %size(T_ext)
0115            for m = 1 :1 :2
0116                   A_ext = [A_LR{m} zeros(AK(m,1), 2*N-2);
0117                        zeros(2*N-2, AK(m,2)) eye(2*N-2)];
0118               mb{m} = A_LR{m}*T*H_LR{m}*(pinv(T_ext))*(inv(A_ext));
0119            end;
0120
0121         case {'type-iii', 'type-3'}
0122                T_ext =[ T zeros(K1,4*N-K2);
0123                     zeros(4*N-K2, K2), eye(4*N-K2);
0124                     zeros(K2-K1,4*N)];
0125                %size(T_ext)
0126            for m = 1 :1 :2
0127                   A_ext = [A_LR{m} zeros(AK(m,1), 4*N-AK(m,2));
0128                        zeros(4*N-AK(m,1), AK(m,2)), eye(4*N-AK(m,1))];
0129               mb{m} = A_LR{m}*T*H_LR{m}*(pinv(T_ext))*(inv(A_ext));
0130               [b1,b2] = size(mb{m});
0131
0132               mb{m} = mb{m}(1:b1-1, 1:b2-2-N);      % misc matrix truncation.
0133
0134            end;
0135      end;
0136      a = fliplr(h);
0137      A = struct('inner',a,'left', mb{1},'right', mb{2});
0138
0139   case 'highpass'
0140        %first the left border
0141        [K1, K2] = size(T);
0142      AK = [size(A_LR{1}); size(A_LR{2})];
0143      mb = cell(2,1);
0144      switch lower(FTYPE)
0145         case {'type-i', 'type-1'}
0146                T_ext = [T zeros(size(T));
0147                     zeros(2*N-1, 2*N-1) eye(2*N-1)];
0148                % ATHH_{t}A^{-1} is two scale relation between
0149                % H_j^{edge} and H_{j-1}^edge & H_{j-1}.
0150
0151                for m = 1 :1 :2
0152                   A_ext = [A_LR{m} zeros(AK(m,1), 2*N-1);
0153                            zeros(2*N-1, AK(m,2)) eye(2*N-1)];
0154                   H_edge = A_LR{m}*T*H_LR{m}*(pinv(T_ext))*(inv(A_ext));
0155                   C = H_edge'*H_edge;
0156                   C = eye(size(C))-C;
0157                   G_half = C(1 :N, 1 :3*N-1);              % matrix for $\phi^{half}$.
0158                   C_bar = G_half(:, N+1 :2 :3*N-1);
0159                   [L,U,P] = lu(sparse(inv(C_bar)),0.0);    % Unpivoted LU factor on C_bar^{-1}
0160                   mb{m} = Gram_Schmidt(U*G_half);          % Orthonomalization
0161                end;
0162
0163         case {'type-ii', 'type-2'}
0164                T_ext = [T zeros(K1,2*N-2);
0165                     zeros(2*N-2, K2) eye(2*N-2)];
0166                for m = 1 :1: 2
0167                    A_ext = [A_LR{m} zeros(AK(m,1), 2*N-2);
0168                        zeros(2*N-2, AK(m,2)) eye(2*N-2)];
```

```
0169                    H_edge = A_LR{m}*T*H_LR{m}*(pinv(T_ext))*(inv(A_ext));
0170                    C = H_edge'*H_edge;
0171                    C = eye(size(C))-C;
0172                    G_half = C(1 :N-1, :);
0173                    C_bar = G_half(:, N+1 :2: 3*N-2);
0174                    [L,U,P] = lu(sparse(inv(C_bar)),0.0);
0175                    mb{m} = Gram_Schmidt(U*G_half);
0176                end;
0177
0178          case {'type-iii', 'type-3'}
0179                T_ext =[ T zeros(K1,4*N-K2);
0180                     zeros(4*N-K2, K2), eye(4*N-K2);
0181                     zeros(K2-K1, 4*N)];
0182             for m = 1 :1: 2
0183                A_ext = [A_LR{m} zeros(AK(m,1), 4*N-AK(m,2));
0184                             zeros(4*N-AK(m,1), AK(m,2)), eye(4*N-AK(m,1))];
0185                    H_edge = A_LR{m}*T*H_LR{m}*(pinv(T_ext))*(inv(A_ext));
0186                    C = H_edge'*H_edge;
0187                    C = eye(size(C))-C;
0188                    G_half = C(1 :N+1, 1:3*N);
0189                    C_bar = G_half(:, N:2:3*N);
0190                    [L,U,P] = lu(sparse(inv(C_bar)),0.0);
0191                % We can perform Gram-Schmidt orthogonalization
0192                    % on rows of staggered U*G_half. Or alternatively:
0193                A_w = U*G_half*G_half'*U';
0194                    B = inv(chol(A_w)');
0195                mb{m} = B*U*G_half;
0196                [b1,b2] = size(mb{m});
0197                    mb{m} = mb{m}(1:b1-1, 1:b2-2);
0198             end;
0199       end;
0200       a = fliplr(g);
0201       A = struct('inner',a,'left',mb{1},'right',mb{2});
0202
0203    case 'precondition'
0204       % Calculating the Pre/Post Filter
0205       % remember that A is the basis transformation matrix
0206       % to orthognalize $\phi^{half}$ to $\phi^{edge}$
0207
0208       switch lower(FTYPE)
0209          case {'type-i', 'type-1', 'type-ii', 'type-2'}
0210                V = T(1:N, N :1 :2*N-1)';     % binomial coefficients
0211                b = inv(V*A_LR{2}');
0212                a = inv(V*A_LR{1}');
0213
0214          case {'type-iii', 'type-3'}
0215                V = T(1:N, N+1 :1 :2*N)';     % binomial coefficients
0216                %b = V*A_LR{2}';
0217                %a = V*A_LR{1}';
0218                %[b1,b2] = size(b);
0219                %b = b(1:b1-1, 1:b2-1);
0220                %a = b(1:b1-1, 1:b2-1);
0221                %b=inv(b);
0222                %a=inv(a);
0223
0224                b = inv(V*A_LR{2}')
```

```
                  a = inv(V*A_LR{1}')
                  %[b1,b2] = size(b);
                  %b = b(1:b1-1, 1:b2-1);
                  %a = b(1:b1-1, 1:b2-1);
      end;
      A = struct('left', a, 'right',b);

   case 'postcondition'
      switch lower(FTYPE)
         case {'type-i', 'type-1', 'type-ii', 'type-2'}
               V = T(1 :N, N :1 :2*N-1)';    % binomial coefficients
               b = V*A_LR{2}';
               a = V*A_LR{1}';

         case {'type-iii', 'type-3'}
               V = T(1:N, N+1 :1 :2*N)';     % binomial coefficients
               %b = V*A_LR{2}';
               %a = V*A_LR{1}';
               %[b1,b2] = size(b);
               %b = b(1:b1-1, 1:b2-1);
               %a = b(1:b1-1, 1:b2-1);
               b = V*A_LR{2}'
               a = V*A_LR{1}'
               %[b1,b2] = size(b);
               %b = b(1:b1-1, 1:b2-1);
               %a = a(1:b1-1, 1:b2-1);
      end;
      A = struct('left',a, 'right',b);

   otherwise
        A = [];
        disp('usage: A= CDV_Bfilters(wname, request)');
        disp('  REQUEST must be ''lowpass'', ''highpass'', ''precondition'' or ''postcondition'
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% S U B   R O U T I N E S
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T = T_matrix(N, TYPE)

switch lower(TYPE)
    case {'type-0'}
        T = eyes(2*N-1, 2*N-1);

    case {'type-i', 'type-1','type-ii', 'type-2'}
        T = zeros(N, 2*N-1);
        for k = 0 :1 :N-1
            for L = 1-N :1 :2*N-2+(1-N)
                if (N-k-1 >= 0) && (abs(N-L-1) >= abs(N-k-1))
                    indx = nchoosek(N-L-1, N-k-1);
                    if (size(indx) == [1 1])
                        T(k+1, L+1-(1-N)) = indx;
                    end;
                end;
            end;
```

```
0281         end;
0282
0283     case {'type-iii', 'type-3'}
0284         T = zeros(N, 2*N);
0285         for k = 1 :1 :N
0286             for L = 1-N: 1: 2*N-2+(1-N)+1
0287                 if (N-k-1 >= 0) && (abs(N-L-1) >= abs(N-k-1))
0288                     indx = nchoosek(N-L-1, N-k-1);
0289                     if (size(indx) == [1 1])
0290                         T(k,L+1-(1-N)) = indx;
0291                     end;
0292                 end;
0293             end;
0294         end;
0295         T(:,2*N)=0;
0296         T(N,2*N)=1;
0297
0298     otherwise
0299         disp('T_matrix: Unknown Type');
0300         T = [];
0301         return;
0302 end;
0303 %---------------------------------------------------------------------
0304
0305 function H = H_matrix(h,N, TYPE)
0306
0307 switch lower(TYPE)
0308     case {'type-i', 'type-1','type-0'}
0309         H = zeros(2*N-1, 4*N-2  );
0310         for k = 1-N :1 :N-1
0311             for L = 1-N :1 :3*N-2
0312                 indx=L-2*k;
0313                 indx=indx+N;
0314                 if (indx>0) && (indx<=2*N)
0315                     H(k-(1-N)+1, L-(1-N)+1)= h(indx);
0316                 end;
0317             end;
0318         end;
0319
0320     case {'type-ii', 'type-2'}
0321         H = zeros(2*N-1, 4*N-3);
0322         for k = 1-N :1 :N-1
0323             for L = 1-N+1 :1 :3*N-1
0324                 indx = L-2*k;
0325                 indx = indx+N;
0326                 if (indx > 0) && (indx <= 2*N)
0327                     H(k-(1-N)+1, L-(1-N+1)+1) = h(indx);
0328                 end;
0329             end;
0330         end;
0331
0332     case {'type-iii', 'type-3'}
0333         H = zeros(2*N, 4*N );
0334         for k = 1-N :1 :N
0335             for L = 1-N :1 :3*N
0336                 indx = L-2*k;
```

```
0337                 indx = indx+N;
0338                 if (indx > 0) && (indx <= 2*N)
0339                     H(k-(1-N)+1, L-(1-N)+1) = h(indx);
0340                 end;
0341             end;
0342         end;
0343
0344     otherwise
0345         disp('H_matrix: Unknown Type');
0346         H = [];
0347         return;
0348 end;
0349
0350 %---------------------------------------------------------------------
0351 function A0= A0_matrix(H, N, TYPE)
0352
0353 switch lower(TYPE)
0354    case {'type-i', 'type-1','type-0'}
0355       H1= H(:, 1 :2*N-1);
0356       H2= H(:, 2*N :4*N-2);
0357       A0= zeros(2*N-1, 2*N-1);
0358
0359    case {'type-ii', 'type-2'}
0360       H1= H(:, 1 :2*N-1);
0361       H2= H(:, 2*N-1 :4*N-3);
0362       H2(:,1)= 0;
0363       A0= zeros(2*N-1, 2*N-1);
0364
0365    case {'type-iii', 'type-3'}
0366       H1= H(:, 1 :2*N);
0367       H2= H(:, 2*N+1 :4*N);
0368       A0= zeros(2*N, 2*N);
0369
0370    otherwise
0371       disp('A0_matrix: Unknown Type');
0372       A0 = [];
0373       return;
0374 end;
0375
0376 k= 0; e= 1;
0377
0378 %repeat until nearing convergence
0379 while e > 1E-032
0380     A1= A0+(H1^k)*H2*H2'*((H1')^k);
0381     e= max(max(abs(A1-A0)));
0382     k= k+1;
0383     A0= A1;
0384 end;
0385 %tmp=cond(A0);
0386 %fprintf('Condition Number: %f\n',tmp);
0387
0388 %---------------------------------------------------------------------
```