```matlab
function [tauPPxx,  tauPPxy,  tauPPxz, ...
          tauPPyx,  tauPPyy,  tauPPyz, ...
          tauPPzx,  tauPPzy,  tauPPzz, ...
          tauDPx,   tauDPy,   tauDPz,  ...
          tauDPx_t, tauDPy_t, tauDPz_t,...
          nxDP,     nyDP,     nzDP,    ...
          nxPP,     nyPP,     nzPP,  ...
          tauDD] = convFace(T, tau, k, formula)

%[tauDP, nxDP, nyDP, nzDP,tauDD] = convFace(T, tau, k, formula)
%
%Input:
%              T: expanded terahedrization
%            tau: penalization parameter for HDG (Nelts x 4)
%              k: polynomial degree
%        formula: quadrature formula in 2d (N x 4 matrix)
%Output:
%        tauPP  :   d3 x d3   x Nelts, with <tau P_i,P_j>_{\partial K}
%        tauDPx : 4*d2 x d3   x Nelts, with <tau D_i,P_j>_e, e\in E(K)
%        tauDPy : 4*d2 x d3   x Nelts, with <tau D_i,P_j>_e, e\in E(K)
%        tauDPz : 4*d2 x d3   x Nelts, with <tau D_i,P_j>_e, e\in E(K)
%         nxDP  : 4*d2 x d3   x Nelts, with  <nx D_i,P_j>_e, e\in E(K)
%         nyDP  : 4*d2 x d3   x Nelts, with  <ny D_i,P_j>_e, e\in E(K)
%         nzDP  : 4*d2 x d3   x Nelts, with  <nz D_i,P_j>_e, e\in E(K)
%
%Last modified: March 15, 2024

% Redefinition of elements
Nelts = size(T.elements,1);
Nnodes= size(formula,1);

d2=nchoosek(k+2,2);
d3=nchoosek(k+3,3);

% Definition of the element area * Evaluation of Tau coeficients
TauArea= T.area(T.facebyele').*tau;     % 4 x Nelts
Area   = T.area(T.facebyele');          % 4 x Nelts
T.perm = T.perm';                       % 4 x Nelts

% Definition on the reference element
s=formula(:,2);         % Reference "X-Axis"
t=formula(:,3);         % Reference "Y-Axis"
weights=formula(:,4);   % Weights of formulas

% --------------------- Computation <tau*Pi,Pj> ------------------- %

O=zeros(size(s));   % Big O definition

% Groups of quadrature points on the face K_hat
% This will need the use of the permutation matrix to conserve
% the matching orientations for these points
points3d=[s,t,O;...
          s,O,t;...
          O,s,t;...
          s,t,1-s-t];

% Evaluation of the quadrature points on face K_hat
pb=dubiner3d(2*points3d(:,1)-1, ...  % Evaluation on quadrature points
```

```
            2*points3d(:,2)-1, ...  % of the Dubiner basis.
            2*points3d(:,3)-1, k);  % (4 * Nnodes) x d3

pbweights=bsxfun(@times,[weights;...   % Weights definition
                         weights;...   % on the basis defined.
                         weights;...
                         weights],pb);

% Indexing based evaluation: Definition of pbweights as a column vector
% Take in count the evaluation it's respectibly:
% It's operated k * Nnodes : (k + 1)(Nnodes + 1) with k = 0 : 3
pbpb = zeros(d3, d3, 4);

pbpb(:, :, 1) = pbweights(1:Nnodes,:)'            *pb(1:Nnodes,:);
pbpb(:, :, 2) = pbweights(Nnodes+1:2*Nnodes,:)'   *pb(Nnodes+1:2*Nnodes,:);
pbpb(:, :, 3) = pbweights(2*Nnodes+1:3*Nnodes,:)' *pb(2*Nnodes+1:3*Nnodes,:);
pbpb(:, :, 4) = pbweights(3*Nnodes+1:4*Nnodes,:)' *pb(3*Nnodes+1:4*Nnodes,:);

% Storage reservation for components matrices
tauPPxx = zeros([d3, d3*Nelts]);
tauPPxy = zeros([d3, d3*Nelts]);
tauPPxz = zeros([d3, d3*Nelts]);

tauPPyx = zeros([d3, d3*Nelts]);
tauPPyy = zeros([d3, d3*Nelts]);
tauPPyz = zeros([d3, d3*Nelts]);

tauPPzx = zeros([d3, d3*Nelts]);
tauPPzy = zeros([d3, d3*Nelts]);
tauPPzz = zeros([d3, d3*Nelts]);

nxPP = zeros([d3, d3*Nelts]);
nyPP = zeros([d3, d3*Nelts]);
nzPP = zeros([d3, d3*Nelts]);

% Expansion of the matrix per components per face
for l=1:4

    % Current integral computation: Face l

    % Normal components separation
    Nx=T.normals(:,3*(l-1)+1)';
    Ny=T.normals(:,3*(l-1)+2)';
    Nz=T.normals(:,3*(l-1)+3)';

    % Computation of different coordinates the values of the accumulated
    % integrals:
    % - X-Component * X-Component
    tauPPxx = tauPPxx + kron(TauArea(l,:) .* (Ny.^2 + Nz.^2), pbpb(:, :, l));

    % - X-Component * Y-Component
    tauPPxy = tauPPxy + kron(TauArea(l,:) .* Nx .* Ny, pbpb(:, :, l));

    % - X-Component * Z-Component
    tauPPxz = tauPPxz + kron(TauArea(l,:) .* Nx .* Nz, pbpb(:, :, l));

    % - Y-Component * X-Component
    tauPPyx = tauPPyx + kron(TauArea(l,:) .* Ny .* Nx, pbpb(:, :, l));
```

```matlab
    % - Y-Component * Y-Component
    tauPPyy = tauPPyy + kron(TauArea(l,:) .* (Nx.^2 + Nz.^2), pbpb(:, :, l));

    % - Y-Component * Z-Component
    tauPPyz = tauPPyz + kron(TauArea(l,:) .* Ny .* Nz, pbpb(:, :, l));

    % - Z-Component * X-Component
    tauPPzx = tauPPzx + kron(TauArea(l,:) .* Nz .* Nx, pbpb(:, :, l));

    % - Z-Component * Y-Component
    tauPPzy = tauPPzy + kron(TauArea(l,:) .* Nz .* Ny, pbpb(:, :, l));

    % - Z-Component * Z-Component
    tauPPzz = tauPPzz + kron(TauArea(l,:) .* (Nx.^2 + Ny.^2), pbpb(:, :, l));

    % Computation of common coordinates of accumulated integrals:
    % - Nx-Component
    nxPP = nxPP + kron(Area(l,:) .* Nx, pbpb(:, :, l));

    % - Ny-Component
    nyPP = nyPP + kron(Area(l,:) .* Ny, pbpb(:, :, l));

    % - Nz-Component
    nzPP = nzPP + kron(Area(l,:) .* Nz, pbpb(:, :, l));

end

% Reshape of the matrices on the desired size
tauPPxx = reshape(0.5 *tauPPxx, [d3, d3, Nelts]);
tauPPxy = reshape(0.5 *tauPPxy, [d3, d3, Nelts]);
tauPPxz = reshape(0.5 *tauPPxz, [d3, d3, Nelts]);

tauPPyx = reshape(0.5 *tauPPyx, [d3, d3, Nelts]);
tauPPyy = reshape(0.5 *tauPPyy, [d3, d3, Nelts]);
tauPPyz = reshape(0.5 *tauPPyz, [d3, d3, Nelts]);

tauPPzx = reshape(0.5 *tauPPzx, [d3, d3, Nelts]);
tauPPzy = reshape(0.5 *tauPPzy, [d3, d3, Nelts]);
tauPPzz = reshape(0.5 *tauPPzz, [d3, d3, Nelts]);

nxPP = reshape(0.5 *nxPP, [d3, d3, Nelts]);
nyPP = reshape(0.5 *nyPP, [d3, d3, Nelts]);
nzPP = reshape(0.5 *nzPP, [d3, d3, Nelts]);

% -------------- Computation <alpha*D,P>, alpha=tau,nx,ny,nz ------------- %

% Definition of the matrix pb, where the columns represent the
% Respective evaluation per coordinates of the nodes
pb=[pb(1:Nnodes,:),pb(Nnodes+1:2*Nnodes,:),...
    pb(2*Nnodes+1:3*Nnodes,:),pb(3*Nnodes+1:4*Nnodes,:)]; % Nnodes x(4 * d3)

% Possible permutations of the face evaluated: 6 possible combinations
points2d=[s,t;...
          t,s;...
          1-s-t,s;...
          s,1-s-t;...
          t,1-s-t;...
```

```
        1-s-t,t];

% Definition of 2D dubiner (face) evaluation.
db=dubiner2d(2*points2d(:,1)-1,2*points2d(:,2)-1,k);      % 6 * Nnodes x d2

% Same process made with the 3D dubiner, and the matrix it's defined for
% all possible permutations
db=[db(1:Nnodes,:),db(Nnodes+1:2*Nnodes,:),...
    db(2*Nnodes+1:3*Nnodes,:),db(3*Nnodes+1:4*Nnodes,:),...
    db(4*Nnodes+1:5*Nnodes,:),db(5*Nnodes+1:6*Nnodes,:)]; % Nnodes x (6 * d2)

db=bsxfun(@times,weights,db);   % weights formulas times dubiner eval

allproducts=db'*pb;                                       % 6 * d2 x 4 * d3

% Auxiliar function for indexing of the elements
block2   = @(x) (1+(x-1)*d2):(x*d2);         % Face based indexing: 2D
blockdof = @(x) (1+(x-1)*2*d2):((2*x-1)*d2); % Dof based indexing: 2D
block3   = @(x) (1+(x-1)*d3):(x*d3);         % Element based indexing: 3D

% Storage reservation for elements
tauDPx = zeros(4*2*d2,d3*Nelts); % Polynomial products : Reference
tauDPy = zeros(4*2*d2,d3*Nelts); % Polynomial products : Reference
tauDPz = zeros(4*2*d2,d3*Nelts); % Polynomial products : Reference

tauDPx_t = zeros(4*2*d2,d3*Nelts); % Polynomial products : Tangential
tauDPy_t = zeros(4*2*d2,d3*Nelts); % Polynomial products : Tangential
tauDPz_t = zeros(4*2*d2,d3*Nelts); % Polynomial products : Tangential

nxDP = zeros(4*2*d2,d3*Nelts); % X-normal components
nyDP = zeros(4*2*d2,d3*Nelts); % Y-normal components
nzDP = zeros(4*2*d2,d3*Nelts); % Z-normal components

for l=1:4

    % Normal components separation
    Nx=T.normals(:,3*(l-1)+1)';
    Ny=T.normals(:,3*(l-1)+2)';
    Nz=T.normals(:,3*(l-1)+3)';

    % For each face we must define a different transformation matrix, this
    % process is done using:
    oneface=T.faces(T.facebyele(:,l),1:3);

    % Coordinates related to the face
    x=T.coordinates(oneface(:),1);
    y=T.coordinates(oneface(:),2);
    z=T.coordinates(oneface(:),3);

    x12=x(Nelts+1:2*Nelts)-x(1:Nelts); %x_2-x_1
    x13=x(2*Nelts+1:end)-x(1:Nelts);   %x_3-x_1

    y12=y(Nelts+1:2*Nelts)-y(1:Nelts); %y_2-y_1
    y13=y(2*Nelts+1:end)-y(1:Nelts);   %y_3-y_1

    z12=z(Nelts+1:2*Nelts)-z(1:Nelts); %z_2-z_1
    z13=z(2*Nelts+1:end)-z(1:Nelts);   %z_3-z_1
```

```
    % The transformation asociated it's defined as:
    A_0 = [x12, y12, z12]; A_1 = [x13, y13, z13];

    %A_0 = A_0 ./ sqrt(sum(A_0.^2, 2)); A_1 = A_1 ./ sqrt(sum(A_1.^2, 2));

    % With the basis d2 defined per face, we can now define the elements

    % The definition of corresponding permutations it's made in logical
    % expressions, it's verified that the current permutation it's
    % corresponding for each element before the kronigger product it's made
    for mu=1:6

        % ----------------------- Tau x-component ----------------------- %
        % Normals product definition:
        nmu = [Ny.^2 + Nz.^2; -Nx .* Ny; -Nx .* Nz];

        % - Reference element Integral
        % Permutation verification: A_1 definition
        taumu = TauArea(l,:) .* A_1(:, 1)' .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present :
        tauDPx(blockdof(l),:) = tauDPx(blockdof(l),:) + ...
                             kron(taumu,allproducts(block2(mu),block3(l)));

        % - Tangential component integral
        taumu_t = TauArea(l,:) .* sum(nmu .* A_1') .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPx_t(blockdof(l),:) = tauDPx_t(blockdof(l),:) + ...
                           kron(taumu_t,allproducts(block2(mu),block3(l)));

        % - Reference element Integral
        % Permutation verification: A_0 definition
        taumu = TauArea(l,:) .* A_0(:, 1)' .*(T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPx(blockdof(l) + d2,:) = tauDPx(blockdof(l) + d2,:) + ...
                             kron(taumu,allproducts(block2(mu),block3(l)));

        % - Tangential component integral
        taumu_t = TauArea(l,:) .* sum(nmu .* A_0') .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPx_t(blockdof(l) + d2,:) = tauDPx_t(blockdof(l) + d2,:) + ...
                           kron(taumu_t,allproducts(block2(mu),block3(l)));

        % ----------------------- Tau y-component ----------------------- %
        % Normals product definition:
        nmu = [-Ny .* Nx; Nx.^2 + Nz.^2; -Ny .* Nz];

        % - Reference element Integral
        % Permutation verification: A_1 definition
        taumu = TauArea(l,:) .* A_1(:, 2)' .* (T.perm(l,:)==mu);
```

```
        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPy(blockdof(l),:) = tauDPy(blockdof(l),:) + ...
                                kron(taumu,allproducts(block2(mu),block3(l)));

        % - Tangential component integral
        taumu_t = TauArea(l,:) .* sum(nmu .* A_1') .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPy_t(blockdof(l),:) = tauDPy_t(blockdof(l),:) + ...
                              kron(taumu_t,allproducts(block2(mu),block3(l)));

        % - Reference element Integral
        % Permutation verification: A_0 definition
        taumu = TauArea(l,:) .* A_0(:, 2)' .*(T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPy(blockdof(l) + d2,:) = tauDPy(blockdof(l) + d2,:) + ...
                                kron(taumu,allproducts(block2(mu),block3(l)));

        % - Tangential component integral
        taumu_t = TauArea(l,:) .* sum(nmu .* A_0') .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPy_t(blockdof(l) + d2,:) = tauDPy_t(blockdof(l) + d2,:) + ...
                              kron(taumu_t,allproducts(block2(mu),block3(l)));

        % ----------------------- Tau z-component ---------------------- %
        % Normals product definition:
        nmu = [-Nz .* Nx; -Nz .* Ny; Nx.^2 + Ny.^2];

        % - Reference element Integral
        % Permutation verification: A_1 definition
        taumu = TauArea(l,:) .* A_1(:, 3)' .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPz(blockdof(l),:) = tauDPz(blockdof(l),:) + ...
                                kron(taumu,allproducts(block2(mu),block3(l)));

        % - Tangential component integral
        taumu_t = TauArea(l,:) .* sum(nmu .* A_1') .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPz_t(blockdof(l),:) = tauDPz_t(blockdof(l),:) + ...
                              kron(taumu_t,allproducts(block2(mu),block3(l)));

        % - Reference element Integral
        % Permutation verification:
        % aparecer, aparece nmu y la componente que necesitamos!
        taumu = TauArea(l,:) .* A_0(:, 3)' .*(T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
```

```
        tauDPz(blockdof(l) + d2,:) = tauDPz(blockdof(l) + d2,:) + ...
                             kron(taumu,allproducts(block2(mu),block3(l)));

        % - Tangential component integral
        taumu_t = TauArea(l,:) .* sum(nmu .* A_0') .* (T.perm(l,:)==mu);

        % The Tau DP on the block it's redifined if
        % the correct permutation it's present
        tauDPz_t(blockdof(l) + d2,:) = tauDPz_t(blockdof(l) + d2,:) + ...
                           kron(taumu_t,allproducts(block2(mu),block3(l)));

        % ------------------------- n x-normal ------------------------- %
        % Permutation verification: A_1 definition
        nxmu = Area(l,:) .* (Nz .* A_1(:, 2)' - Ny .* A_1(:, 3)') .* (T.perm(l,:)==mu);

        % The nx on the block it's redifined if the
        % correct permutation it's present
        % - Reference element Integral
        nxDP(blockdof(l),:) = nxDP(blockdof(l),:) + ...
                              kron(nxmu,allproducts(block2(mu),block3(l)));

        % Permutation verification: A_1 definition
        nxmu = Area(l,:) .* (Nz .* A_0(:, 2)' - Ny .* A_0(:, 3)') .* (T.perm(l,:)==mu);

        % The nx on the block it's redifined if the
        % correct permutation it's present
        % - Reference element Integral
        nxDP(blockdof(l) + d2,:) = nxDP(blockdof(l) + d2,:) + ...
                              kron(nxmu,allproducts(block2(mu),block3(l)));

        % ------------------------- n y-normal ------------------------- %
        % Permutation verification: A_1 definition
        nymu = Area(l,:) .* (Nx .* A_1(:, 3)' - Nz .* A_1(:, 1)') .* (T.perm(l,:)==mu);

        % The ny on the block it's redifined if the
        % correct permutation it's present
        % - Reference element Integral
        nyDP(blockdof(l),:) = nyDP(blockdof(l),:) + ...
                              kron(nymu,allproducts(block2(mu),block3(l)));

        % Permutation verification: A_1 definition
        nymu = Area(l,:) .* (Nx .* A_0(:, 3)' - Nz .* A_0(:, 1)') .* (T.perm(l,:)==mu);

        % The ny on the block it's redifined if the
        % correct permutation it's present
        % - Reference element Integral
        nyDP(blockdof(l) + d2,:) = nyDP(blockdof(l) + d2,:) + ...
                              kron(nymu,allproducts(block2(mu),block3(l)));

        % ------------------------- n z-normal ------------------------- %
        % Permutation verification: A_1 definition
        nzmu = Area(l,:) .* (Ny .* A_1(:, 1)' - Nx .* A_1(:, 2)') .* (T.perm(l,:)==mu);

        % The nz on the block it's redifined if the
        % correct permutation it's present
        % - Reference element Integral
        nzDP(blockdof(l),:) = nzDP(blockdof(l),:) + ...
                              kron(nzmu,allproducts(block2(mu),block3(l)));
```

```
        % Permutation verification: A_1 definition
        nzmu = Area(l,:) .* (Ny .* A_0(:, 1)' - Nx .* A_0(:, 2)') .* (T.perm(l,:)==mu);

        % The nz on the block it's redifined if the
        % correct permutation it's present
        % - Reference element Integral
        nzDP(blockdof(l) + d2,:) = nzDP(blockdof(l) + d2,:) + ...
                              kron(nzmu,allproducts(block2(mu),block3(l)));

    end
end

% Reshape of elements
tauDPx=reshape(0.5*tauDPx,[4*2*d2,d3,Nelts]); % Tau elements 4 * d2 x d3 x Nelts
tauDPy=reshape(0.5*tauDPy,[4*2*d2,d3,Nelts]); % Tau elements 4 * d2 x d3 x Nelts
tauDPz=reshape(0.5*tauDPz,[4*2*d2,d3,Nelts]); % Tau elements 4 * d2 x d3 x Nelts

tauDPx_t=reshape(0.5*tauDPx_t,[4*2*d2,d3,Nelts]); % Tau tangential 4 * d2 x d3 x Nelts
tauDPy_t=reshape(0.5*tauDPy_t,[4*2*d2,d3,Nelts]); % Tau tangential 4 * d2 x d3 x Nelts
tauDPz_t=reshape(0.5*tauDPz_t,[4*2*d2,d3,Nelts]); % Tau tangential 4 * d2 x d3 x Nelts

nxDP=reshape(0.5*nxDP,[4*2*d2,d3,Nelts]);     % Normal X 4 * d2 x d3 x Nelts
nyDP=reshape(0.5*nyDP,[4*2*d2,d3,Nelts]);     % Normal Y 4 * d2 x d3 x Nelts
nzDP=reshape(0.5*nzDP,[4*2*d2,d3,Nelts]);     % Normal Z 4 * d2 x d3 x Nelts

% Take in count this is the type (c) matrix shape.

% ------------------------- Computation tauDD -------------------------- %

% Dubiner 2D evaluation on nodes (No permutation needed)
d=dubiner2d(2*s-1,2*t-1,k);

dweights=bsxfun(@times,d,weights);  % Dubiner times weights of formulas

dwd=dweights'*d; % This is the DD computation then it's made the d

tauDD=zeros(4*2*d2,4*2*d2,Nelts); % Storage reservation

for l=1:4

    % For each face we must define a different transformation matrix, this
    % process is done using:
    oneface=T.faces(T.facebyele(:,l),1:3);

    % Coordinates related to the face
    x=T.coordinates(oneface(:),1);
    y=T.coordinates(oneface(:),2);
    z=T.coordinates(oneface(:),3);

    x12=x(Nelts+1:2*Nelts)-x(1:Nelts); %x_2-x_1
    x13=x(2*Nelts+1:end)-x(1:Nelts);   %x_3-x_1

    y12=y(Nelts+1:2*Nelts)-y(1:Nelts); %y_2-y_1
    y13=y(2*Nelts+1:end)-y(1:Nelts);   %y_3-y_1

    z12=z(Nelts+1:2*Nelts)-z(1:Nelts); %z_2-z_1
    z13=z(2*Nelts+1:end)-z(1:Nelts);   %z_3-z_1
```

```
    % The transformation asociated it's defined as:
    A_0 = [x12, y12, z12]; A_1 = [x13, y13, z13];

    %A_0 = A_0 ./ sqrt(sum(A_0.^2, 2)); A_1 = A_1 ./ sqrt(sum(A_1.^2, 2));

    % Dimentional definition of product per face
    tauDD_dimentional = reshape(kron(0.5*TauArea(l,:),dwd), [d2,d2,Nelts]);

    % Tau DD takes place on the 2 basis, we now define the block matrix
    % - First block: Basis A_1[d2] * A_1[d2]
    tauDD(blockdof(l), blockdof(l),:)      = tauDD(blockdof(l), blockdof(l),:) + ...
     bsxfun(@times, reshape(sum(A_1.*A_1, 2), [1, 1, Nelts]), tauDD_dimentional);

    % - Second block: Basis A_1[d2] * A_0[d2]
    tauDD(blockdof(l), d2+blockdof(l),:)   = tauDD(blockdof(l), d2+block2(l),:) + ...
     bsxfun(@times, reshape(sum(A_1.*A_0, 2), [1, 1, Nelts]), tauDD_dimentional);

    % - Third block: Basis A_0[d2] * A_1[d2]
    tauDD(d2+blockdof(l), blockdof(l),:)   = tauDD(d2+blockdof(l), blockdof(l),:) + ...
     bsxfun(@times, reshape(sum(A_0.*A_1, 2), [1, 1, Nelts]), tauDD_dimentional);

    % - Fourth block: Basis A_0[d2] * A_0[d2]
    tauDD(d2+blockdof(l), d2+blockdof(l),:) = tauDD(d2+blockdof(l), d2+blockdof(l),:) +↙
...
     bsxfun(@times, reshape(sum(A_0.*A_0, 2), [1, 1, Nelts]), tauDD_dimentional);

end

% This is the type (b) matrix, corresponding to the shape 4*d2 x d2 x Nelts
end
```