```
function [Mi, CMy_xz, CMz_xy, CMz_yx, CMx_yz, CMx_zy, CMy_zx] = curlMatrix(T,k,formula)

% [Mi, CMy_xz, CMz_xy, CMz_yx, CMx_yz, CMx_zy, CMy_zx] = curlMatrix(T,k,formula)
% Input:
%           T: expanded tetrahedrization
%           k: polynomial degree
%     formula: quadrature formula in 3d (N x 5 matrix)
%
% Output:
% {curl_x, ..., curl_z}: Each cell contains the curl component of per
%                        variable
%
% Last modified: Febrary 24, 2024

% Recovery of defined elements
Nelts=size(T.elements,1);
Nnodes=size(formula,1);
d3=nchoosek(k+3,3);

% Elements definition of the Piola transform : *^T
x12=T.coordinates(T.elements(:,2),1)-T.coordinates(T.elements(:,1),1); %x2-x1
x13=T.coordinates(T.elements(:,3),1)-T.coordinates(T.elements(:,1),1); %x3-x1
x14=T.coordinates(T.elements(:,4),1)-T.coordinates(T.elements(:,1),1); %x4-x1
y12=T.coordinates(T.elements(:,2),2)-T.coordinates(T.elements(:,1),2); %y2-y1
y13=T.coordinates(T.elements(:,3),2)-T.coordinates(T.elements(:,1),2); %y3-y1
y14=T.coordinates(T.elements(:,4),2)-T.coordinates(T.elements(:,1),2); %y4-y1
z12=T.coordinates(T.elements(:,2),3)-T.coordinates(T.elements(:,1),3); %z2-z1
z13=T.coordinates(T.elements(:,3),3)-T.coordinates(T.elements(:,1),3); %z3-z1
z14=T.coordinates(T.elements(:,4),3)-T.coordinates(T.elements(:,1),3); %z4-z1

axy=y14.*z12-y12.*z14;
axz=y12.*z13-y13.*z12;

ayx=x14.*z13-x13.*z14;
ayz=x13.*z12-x12.*z13;

azx=x13.*y14-x14.*y13;
azy=x14.*y12-x12.*y14;

% Definition of normalized coordinates
xhat=formula(:,2);
yhat=formula(:,3);
zhat=formula(:,4);

% Evaluation on the reference element
[P, Px, Py, Pz]=dubiner3d(2*xhat-1,2*yhat-1,2*zhat-1,k);  % Nnd x d3

% Rescaling: Inner derivative
Px = 2*Px; Py = 2*Py; Pz = 2*Pz;

wP=bsxfun(@times,formula(:,5),P);

% Definition of submatrices, product of P by P: Mass matrix.
Mi = zeros(d3,Nelts*d3);

% ---------------------- Computation <PiPj> -------------------------- %
% Rotational arrays definition: Parallel calculation of matrices
for q = 1:Nnodes
```

```
    % Nodal computation of each matrix, recursion in Mi
    Mi = Mi + kron(T.volume', wP(q, :)' * P(q, :));

end

% Reshaping of the output Rotational Matrices
Mi = reshape(Mi, [d3, d3, Nelts]);

% ---------------------- Computation <PidPj> ------------------------- %

% Definition of hat matrices: Rotational computation
CMx_hat = 1/6 * wP' * Px;
CMy_hat = 1/6 * wP' * Py;
CMz_hat = 1/6 * wP' * Pz;

% Definition of inner derivative of transform
CMy_xz = kron(axz',CMy_hat); CMz_xy = kron(axy',CMz_hat);
CMz_yx = kron(ayx',CMz_hat); CMx_yz = kron(ayz',CMx_hat);
CMx_zy = kron(azy',CMx_hat); CMy_zx = kron(azx',CMy_hat);

% Reshape into a 3 dimentional matrix with Nelts
CMy_xz = reshape(CMy_xz,[d3,d3,Nelts]);
CMz_xy = reshape(CMz_xy,[d3,d3,Nelts]);

CMz_yx = reshape(CMz_yx,[d3,d3,Nelts]);
CMx_yz = reshape(CMx_yz,[d3,d3,Nelts]);

CMx_zy = reshape(CMx_zy,[d3,d3,Nelts]);
CMy_zx = reshape(CMy_zx,[d3,d3,Nelts]);

end
```