# Documentation and implementation

January 15, 2008

The main files of the package are listed below

- `z=evalArgy(x,y,p)`: `x`, `y` are scalars (or row vectors of the same length) with the coordinates of a point (or a set of points) where we will evaluate the basis functions; `p` is a $2 \times 3$ matrix that determines the triangle. Its $j$th column are the coordinates of the $j$th vertex. The only output argument (`z`) is a $21 \times k$ matrix $k$ being the length of `x` or `y`, where the $i$th row is the values of the $i$th element of the basis at the points given by `x`, `y`.

- `[dx,dy]=evalGradArgy(x,y,p)` returns in `dx`,`dy` the values of the $x-$ and $y-$ derivatives of the elements of the local basis of the Argyris element at $(\mathtt{x},\mathtt{y})$. All the input and output arguments follow the convention above.

- `[dxx,dxy,dyy]=evalHessArgy(x,y,p)` evaluates the second derivatives of the elements of the Argyris basis at $(\mathtt{x},\mathtt{y})$.

We have some auxiliary routines used in the computations:

- `[C,B,b,Th]=changeOfBasis(p)` computes $C$, $B$, $\mathbf{b}$ and $\Theta$, the matrices (and vector) involved in the changes of coordinates between the reference triangle and the triangle given by `p`.

- `[zx,zy]=khat2k(x,y,B,b)` returns in (`zx`,`zy`) the image of points by the linear map $(x,y)^\top \mapsto B(x,y)^\top + b$. In our computations, $B$ and $\mathbf{b}$ are the coefficients of the transformation mapping $\widehat{K}$ to $K$. `[zx,zy]=k2khat(x,y,B,b)` computes images with the inverse mapping. As before, `x` and `y` can be row vectors.

There is a data file

- `coefRef.dat`: in this file it is saved a $21 \times 21$ matrix. The $i$th row contains the coefficients in the monomial basis of the $i$th element of the Argyris basis in the reference triangle.

This matrix is used by the main routines `evalArgy, evalGradArgy, evalHessArgy`. These functions proceed to check at the beginning of the code the existence of the variable `coefRef`. In case that it has not been declared, the file `coefRef.dat` is loaded and stored in a global variable of the same name. Then `coefRef` is now defined and becomes accessible for the all the routines from now on. Therefore, the file `coefRef.dat` is loaded only once per session.

Finally, we have the routine

- `coefRef=reference:` returns the array `coefRef` with the coefficients of the Argyris basis on the reference triangle. This function can be used also to compute the Argyris basis on a user-chosen triangle. It requires the symbolic toolbox.

In practise, there is no need to execute this file since the matrix `coefRef` is available as a data file. We have included it here for the sake of completeness.

In the remainder of the section we give some details about the implementation of this package, starting with the function `changeOfBasis`. The computation of $B$, $\mathbf{b}$ and $\Theta$ is straightforward and is made in an internal function called `afftrans`. The bulk of the code is devoted to the computation of $C$. We distinguish three parts. The first part defines some geometric quantities

```
v=[p(:,2)- p(:,1), p(:,3)-p(:,1), p(:,3)-p(:,2)];
[B,b,Th]=afftrans(p);
sides=diag([norm(v(:,1)),norm(v(:,2)),norm(v(:,3))]);
aux=sides^(-2)*[0  1; -1   0; -1/sqrt(2) -1/sqrt(2)]*B';  % see (3)
R=[0 -1; 1 0];
```

Next we construct the matrix $D$:

```
f=dot(aux',v);
g=dot(aux',R*v);
D=blkdiag(eye(3),B',B',B',Th,Th,Th,[diag(g) diag(f)]);
```

Note the use of the Matlab command `dot` to compute $f_\alpha$ and $g_\alpha$ by doing the dot product between the columns of appropriate matrices. The command `blkdiag` is finally employed to assembly the matrix $D$.

The construction of the matrix $E$ is done similarly

```
E=zeros(24,21);
E(1:21,:)=blkdiag(eye(18),sides);
E(22:24,1:3)= 15/8*[-1  1 0;  -1  0 1; 0 -1 1];
E(22:24,4:9)=-7/16*[v(:,1)' v(:,1)'  0  0;
                    v(:,2)' 0  0     v(:,2)';...
                    0  0    v(:,3)'  v(:,3)'];
w=[v(1,:).^2; 2*v(1,:).*v(2,:); v(2,:).^2]';
E(22:24,10:18)=1/32*[-w(1,:)    w(1,:)    0  0  0;.
                     -w(2,:)   0  0  0     w(2,:);...
                     0  0  0   -w(3,:)     w(3,:)];
```

The program finishes by computing $C$

```
C=D*E;
```

Function `evalArgy` performs the evaluation of the Argyris basis in the triangle specified by `p`. This is done in the following lines

```
[C,B,b]=changeOfBasis(p);
[x,y]=k2khat(x,y,B,b);
z=monomials(x,y);
z=C'*coefRef*z;
```

The point $(x, y)$ is mapped first into the reference triangle and next the elements of the monomial basis are evaluated at this point (note that the $i$th row of `coefRef*z` corresponds to $\widehat{N}_i(x, y)$). The change of basis, and therefore the evaluation of the local basis $N_i(x, y)$ in the user-specified triangle, is carried out by the left multiplication by `C'`.

This code can be vectorized just by allowing both `x,y` to be row vectors of the same length. If $k$ is the length of `x`, `y`, `z`, becomes a $21 \times k$ matrix in all the occurrences.

To compute the first derivatives we use the chain rule (recall that the gradient is seen columnwise)

$$(\nabla N_j(\mathbf{x}))^\top = \sum_{i=1}^{n} c_{ij} (\nabla \widehat{N}_i)^\top \circ F^{-1}(\mathbf{x}) B^{-1}$$

The following lines, which belong to the function `evalGradArgy`, evaluate the gradient

```
[C,B,b]=changeOfBasis(p);
[x,y]=k2khat(x,y,B,b);
k=length(x);
mx=derx(x,y);
my=dery(x,y);
grads=zeros(21,2*k);
grads(:)=[mx(:) my(:)]/B;
grads=C'*coefRef*grads;
dx=grads(:,1:k);
dy=grads(:,k+1:2*k);
```

Functions `derx`, `dery` return a column vector with the derivatives of the monomial basis evaluated at (`x`, `y`). The columnwise access to the elements of a matrix in Matlab is used here to set `grads` in such a way that after running the six first lines, `grads` has in the first $k$ columns $\partial_x (m_i \circ F^{-1})(x_j, y_j)$ (here $m_i$ denotes the $i$th element of the basis of monomials and $F$ the affine mapping from $\widehat{K}$ onto $K$) whereas $\partial_y (m_i \circ F^{-1})(x_j, y_j)$ are stored in the last $k$ columns. Finally, left multiplication by `coefRef` and `C'` makes the change of basis.

The evaluation of the second derivatives, which is done in `evalHessArgy`, is implemented in the same manner:

```
[C,B,b,Th]=changeOfBasis(p);
[x,y]=k2khat(x,y,B,b);
k=length(x);
mxx=derxx(x,y);
mxy=derxy(x,y);
myy=deryy(x,y);
hessian=zeros(21,3*k);
hessian(:)=[mxx(:) mxy(:) myy(:)]/Th';
hessian=C'*coefRef*hessian;
```

```
dxx=hessian(:,1:k);
dxy=hessian(:,k+1:2*k);
dyy=hessian(:,2*k+1:3*k);
```