

1. EXPLANATION OF FINITE ELEMENT CODE

A finite element is a triple (T, Π, Σ) consisting of a closed polygonal domain T (triangle or quadrilateral in our case), a finite dimensional space of shape functions $\Pi \subset C^0(T)$, and a set Σ of independent linear functionals on Π (each functional is called a degree of freedom). A space $V_h \subset \mathbb{L}^1(\Omega)$ on a bounded domain Ω is called finite element space based on $M \in \mathbb{N}$ finite elements (T_i, Π_i, Σ_i) , if

$$\text{int}(T_i) \cap \text{int}(T_j) = \emptyset, \quad i \neq j, \quad \bar{\Omega} = \bigcup_{i=1}^M T_i,$$

and if for each $v \in V_h$, the restriction to $\text{int}(T_j)$ equals $p|_{\text{int}(T_j)}$ for exactly one $p \in \Pi_j$.

Finally, the space V_h is generated with affine transformations, if V_h is based on finite elements (T_j, Π_j, Σ_j) , $j = 1, \dots, M$ which are all obtained through invertible affine transformations F_j from a reference element $(T_{\text{ref}}, \Pi_{\text{ref}}, \Sigma_{\text{ref}})$, where

$$T_j = F_j(T_{\text{ref}}), \quad \Pi_j = \{p_{\text{ref}} \circ F_j^{-1} \mid p_{\text{ref}} \in \Pi_{\text{ref}}\}$$

and

$$\Sigma_j = \{\Lambda \mid \Lambda(p) = \Lambda_{\text{ref}}(p \circ F_j), \Lambda_{\text{ref}} \in \Sigma_{\text{ref}}\}.$$

In the following, we assume that the triangulation is admissible in the sense that any two polygons $T_i \neq T_j$ with non-empty intersection either have a full edge or a single vertex in common (no hanging vertices). Moreover, the members of Σ_{ref} should be delta-functionals at points $x_1^{\text{ref}}, \dots, x_s^{\text{ref}} \in T_{\text{ref}}$ where possible points on ∂T_{ref} are distributed in such a way that they always coincide on common edges of transformed elements. Moreover, the restriction of a shape function on an edge of ∂T_{ref} should be uniquely determined by the degrees of freedom on that edge.

With these assumptions, the degrees of freedom which guarantee $V_h \subset H^1(\Omega)$ are easily obtained by removing duplicate functionals corresponding to nodes $F_i(x_k^{\text{ref}}) = F_j(x_l^{\text{ref}})$ on the boundary of adjacent elements.

With these considerations as background, we now explain the basic steps of the Matlab FE code:

```
% set problem parameter
FEM = parameters;
% generate the mesh
```

```

FEM = trimesh(FEM);
% generate reference element
FEM = P1(FEM);
% generate finite element space
FEM = FEspace(FEM);
% assemble FE matrix
FEM = assembleLS(FEM);
% solve problem
v = FEM.matrices.A\FEM.matrices.b;

```

In line 2, problem parameters are stored in the struct `FEM.parameters` including geometry and mesh size information needed in the triangulation step (line 4). The resulting struct `FEM.Th` in combination with the necessary information `FEM.Ref` on the reference element obtained in line 6 is then used in line 8 to generate the finite element space `FEM.Vh`. In the present case, the space is constructed from a triangular mesh and P_1 -shape functions. The information contained in the struct `FEM.Vh` is required in the assembly step together with parameters in `FEM.parameters` which describe, for example, the right hand side or the boundary data. The resulting finite element matrices and vectors are returned in the struct `FEM.matrices` which can then be used for solving the problem with appropriate methods.

In the following subsections, each subroutine and the corresponding output data structure is described in more detail.

1.1. The triangulation. Starting point for the construction of the finite element data is a triangulation $\mathcal{T}_h = \{T_1, \dots, T_M\}$ of the bounded polygonal domain $\Omega \subset \mathbb{R}^{\dim}$ with $\dim = 2$. The polygonal sets T_i which are all of the same type like triangles ($q = 3$), quadrilaterals ($q = 4$), etc., are described by specifying the q vertices of their boundaries. This is implemented with two separate matrices. The matrix `elements` of dimension $M \times q$ contains in row i the vertex numbers of ∂T_i . The sequence of the numbers has to reflect the following orientation of ∂T_i : when moving from vertex j to vertex $j + 1$, the set T_i has to be on the left hand side. The required vertex coordinates are contained in the matrix `corners` of dimension $nc \times \dim$ which contains in each row a vertex coordinate vector.

In addition to this inevitable grid data, there may be other relevant information associated to the triangulation. For example, the problem may contain the side condition that the solution has specified values on parts of the boundary $\partial\Omega$. This so called Dirichlet condition may

be incorporated directly into the definition of the finite element space. To this end, the matrix `Dirichlet` of dimension $n_s \times 2$ contains in its rows the vertex numbers of the segments which make up the Dirichlet boundary. The finite element space will later consist of functions which are identically zero on these segments (including the end points). In the same way, other parts of the boundary or boundaries of subdomains can be specified.

1.2. The reference element. All elements are assumed to be affine transformations of a single reference element $(T_{\text{ref}}, \Pi_{\text{ref}}, \Sigma_{\text{ref}})$ where the degrees of freedom are delta-functionals localizing at points of T_{ref} which are stored in the matrix `nodes` of size $s \times 2$. The list `nodes` also contains the vertices of the polygon T_{ref} . The corresponding row indices are collected in the vector `shape` where the ordering is compatible with the one of the elements in the triangulation: when moving from node `shape(i)` to node `shape(i+1)`, the reference element T_{ref} is located on the left hand side. When there are degrees of freedom associated to nodes located on the interior of edge i (starting at vertex i and ending at vertex $i + 1$), the corresponding node numbers (row indices of `nodes`) are listed in the vector `onedges{i}`, again with the ordering described above. Finally, the numbers of interior nodes are listed in arbitrary order in the vector `interior`.

The number of vertices, interior nodes and nodes on edge 1, edge 2, etc. are stored in this sequence in the vector `nodedis` and `nnodes` contains the total number of nodes.

To set up the affine map which transforms the reference element T_{ref} to a specific element T_j , the matrix `makeF` is needed. If j is the element number, then `vnum=Th.elements(j, :)` is the list of vertex numbers so that `p=Th.corners(vnum, :)` is a matrix with the vertex coordinates in its rows. Using `makeF`, the affine map is constructed with the product `F=makeF*p` so that `[1,x,y]*F` are the coordinates of the point in T_j belonging to $(x, y) \in T_{\text{ref}}$.

Apart from geometrical specifications of T_{ref} and Σ_{ref} , the data structure concerning the reference element also contains full information on the shape functions Π_{ref} . For example, the function `basis` yields the values of the i -th basis function at the points specified by the $n \times 2$ -matrix `x` when calling it with `basis(i,x)`. Similarly, the partial derivative with respect to the first and the second coordinate are provided with the functions `dbasis{1}` and `dbasis{2}`.

The assembly of the finite element matrices and vectors is carried out numerically using quadrature rules. In general, the degree of exactness should be high enough to guarantee exact integration in the constant coefficient case (with polynomials as shape functions). The corresponding integration weights and nodes are stored in `intw` and `intx` where the latter is a $m \times 2$ matrix. The values of all base functions at the integration nodes are precomputed and stored in the variable `baseval` - more precisely, `baseval(:,i)` are the function values of basis function `i`. Similarly, the values of the partial derivatives are available in `dbaseval(1, :, i)` and `dbaseval(2, :, i)`.

Similarly, for integration along edge `k` of the reference element, the matrices `bdrx{k}` and `bdrw{k}` contain integration nodes and weights while the vectors `basevalbdr{k}(:, i)` and `dbasevalbdr{k}(1, :, i)`, `dbasevalbdr{k}(2, :, i)` are the corresponding values of basis function `i` and its partial derivatives at the integration nodes on edge `k`. It should be noticed that the measure represented by `bdrw{k}` is the standard Lebesgue measure of the unit interval which serves as domain of the edge parametrization. The missing Jacobian factor has to be included when integrating over edges of the transformed element.

1.3. The finite element space. Detailed information about the finite element space is set up using the triangulation `Th` and the reference element `Ref`.

In each of the `M` elements T_j , the i th reference node has a unique global node number which is obtained with `k=elementnodes(j,i)`. The coordinates of node `k` are obtained with `nodes(k,:)`. The number of the nodal basis function belonging to node `k` can be found with `nodetodof(k)`. If the node does not correspond to a basis function because it is located on a Dirichlet boundary, the result is zero. The inverse mapping is realized with the list `dofonode` which yields the node number corresponding to the number of a nodal basis function. The dimension of the finite element space is available as `ndof`.

Whenever two element vertices `i<j` are connected by an edge, the matrix entry `k=edgelist(i,j)` yields the number of the corresponding edge. More information on edge `k` is stored in the cell array `edge`.

For example, `edge{k}.n` is the number of elements adjacent to edge `k`. Their element numbers are available from the vector `edge{k}.element`. Moreover, the number of the reference edge number which is mapped to the edge under consideration is stored for each of the relevant elements

in `edge{k}.edge`. Finally, `edge{k}.nds` is the number of interior nodes on edge `k`.

1.4. The assembly. In the assembly step, the information on the finite element space and the equation parameters is used to generate the stiffness matrix, the mass matrix and the load vector. Since the construction depends strongly on the equation under consideration, it should be custom edited.

The assembly is carried out by loops over elements (and maybe edges, if the stiffness matrix contains boundary integrals). The required element matrices are constructed first.