LEBREF2D: a 2D mesh refinement package
A short documentation

Leonardo Rocchi
September, 2018

Contents

LEBREF2D is a MATLAB package for the mesh refinement of 2D domains based on an efficient usage of MATLAB built-in functions as well as vectorization. The package includes functions for uniform mesh refinements and local mesh refinements based on the longest edge bisection (LEB) algorithm [6] which is a variant of the newest vertex bisection (NVB) algorithm [2, 4, 7].

1 Data structures

Consider a 2D domain $D \subset \mathbb{R}^2$, with a regular polygonal boundary denoted by ∂D. Let $\mathcal{T}$ be a *conforming* mesh (or triangulation) of the domain D, i.e.,

(i) $\mathcal{T} = \{K_1, \dots, K_M\}$ is a finite set of $M > 0$ triangles (*elements*) $K_i = \text{conv}\{\mathbf{x}_j, \mathbf{x}_m, \mathbf{x}_n\}$ such that $|K_i| > 0$, $i = 1, \dots, M$. Here, $\mathbf{x}_j, \mathbf{x}_m, \mathbf{x}_n$ are the *nodes* (or vertices) of K_i and $|\cdot|$ denotes the area;

(ii) the triangulation $\mathcal{T}$ covers the closure $\overline{D}$ of D;

(iii) the intersection of two elements is either empty, a common node, or a common edge (an *edge* is naturally defined as the line connecting two nodes, i.e., $\text{conv}\{\mathbf{x}_m, \mathbf{x}_n\}$);

(iv) a boundary edge (i.e., those edges lying on ∂D) belongs to one element only.

Figure 1.1 shows three examples of *non-conforming* meshes.

Let N be the number of nodes (including the boundary ones) and let $\mathcal{N} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be the set of nodes. The data representation of the mesh is as done through a structure `MESH` which contains five fields, `coord`, `elem`, `int`, `bnd`, `elbnd`:

- `MESH.coord` is a $N \times 2$ matrix containing the physical coordinates of all nodes. The i-th row of `MESH.coord` stores the coordinates of the i-th node $\mathbf{x}_i = (x_i, y_i) \in \overline{D}$:

$$\texttt{MESH.coord(i,:)} = [x_i \;\; y_i].$$

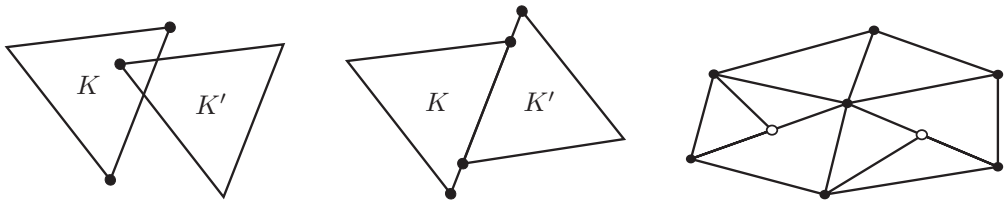


Figure 1.1. Example of *non conforming* triangulations. Intersection of two elements which is not a node (left). Intersection of two element which is not and edge (middle). Presence of *hanging nodes* (right).

- `MESH.elem` is a $M \times 3$ matrix containing the elements nodes' numbers. That is, the ℓ-th element $K_\ell = \{\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k\} \in \mathcal{T}$ with nodes $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k \in \mathcal{N}$ is stored in

  $$\texttt{MESH.elem}(\ell,:) = [i \;\; j \;\; k],$$

  with the nodes counted in **counterclockwise** order.

- `MESH.int` is a $N_D \times 1$ vector containing the interior nodes' numbers (N_D is the number of interior nodes). For example, if the interior nodes are $\mathbf{x}_3$, $\mathbf{x}_9$, $\mathbf{x}_{15}$, and $\mathbf{x}_{21}$, then

  $$\texttt{MESH.int} = [3 \;\; 9 \;\; 15 \;\; 21]^T.$$

- `MESH.bnd` is a $N_{\partial D} \times 1$ vector containing the boundary nodes' numbers ($N_{\partial D}$ is the number of boundary nodes). For example, if the boundary nodes are the $\mathbf{x}_2$, $\mathbf{x}_{10}$, $\mathbf{x}_{19}$, then

  $$\texttt{MESH.bnd} = [2 \;\; 10 \;\; 19]^T.$$

  Notice that $N = N_D + N_{\partial D}$ and that $\{\texttt{MESH.int, MESH.bnd}\} = \{1, \dots, N\}$.

- `MESH.elbnd` is a $M_{\partial D} \times 2$ matrix containing the boundary elements and the (local) number of the corresponding edge lying on the boundary ($M_{\partial D}$ is the number of boundary elements).

  We call *boundary element* an element having one edge lying on the boundary (i.e., two nodes $\mathbf{x}_i, \mathbf{x}_j \in \partial D$). For example, if there are 4 boundary elements K_6, K_{10}, K_{22}, K_{34} with the 1-st, 3-th, 1-st, and 3-th edges on the boundary, respectively, then

  $$\texttt{MESH.elbnd} = \begin{bmatrix} 6 & 1 \\ 10 & 3 \\ 22 & 1 \\ 34 & 3 \end{bmatrix}.$$

Figure 1.2 shows a visual representation of the data structure `MESH` for the default mesh of the square domain $D = (-1, 1)^2$.

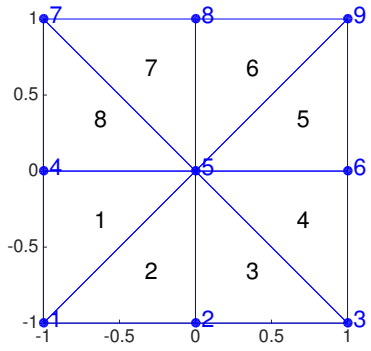
Two important features/assumptions of LEBREF2D are the following:

(A1) (**enumeration of edges**). Given the element $K = \{\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k\} \in \mathcal{T}$, with 1-st node $\mathbf{x}_i$, 2-nd node $\mathbf{x}_j$, and 3-rd node $\mathbf{x}_k$ counted counterclockwise, we assume that

- the 1-st edge of K is the one in front of $\mathbf{x}_i$, i.e., conv$\{\mathbf{x}_j, \mathbf{x}_k\}$,
- the 2-nd edge of K is the one in front of $\mathbf{x}_j$, i.e., conv$\{\mathbf{x}_k, \mathbf{x}_i\}$,
- the 3-rd edge of K is the one in front of $\mathbf{x}_k$, i.e., conv$\{\mathbf{x}_i, \mathbf{x}_j\}$.

(A2) (**longest edge**). The enumeration is such that the longest edge of the generic element K_ℓ is the **second one** (conv$\{\mathbf{x}_k, \mathbf{x}_i\}$).

See Figure 1.3 for a mesh example showing the local enumeration of the edges of the mesh.



.coord

1	-1.0	-1.0
2	0.0	-1.0
3	1.0	-1.0
4	-1.0	0.0
5	0.0	0.0
6	1.0	0.0
7	-1.0	1.0
8	0.0	1.0
9	1.0	1.0

.elem

1	5	4	1
2	1	2	5
3	5	2	3
4	3	6	5
5	5	6	9
6	9	8	5
7	5	8	7
8	7	4	5

.int

5

.bnd

1
2
3
4
6
7
8
9

.elbnd

1	1	3
2	2	1
3	3	3
4	4	1
5	5	3
6	6	1
7	7	3
8	8	1

Figure 1.2. Mesh for the square domain $D = (-1, 1)^2$, with nodes in blue and elements in black, and corresponding representation of the `MESH` structure's fields.

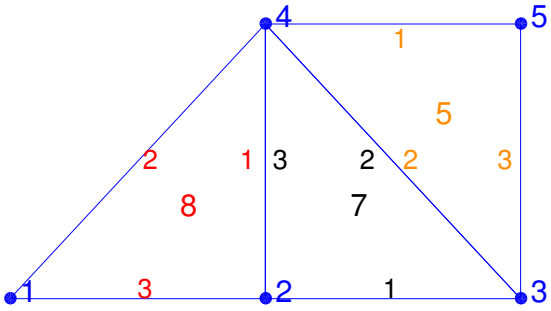


Figure 1.3. Local enumeration of edges for a given mesh (mesh nodes in blue). The example shows three elements K_8, K_7, and K_5, with the element numbers in the middle, and the local enumeration of the edges (with the same corresponding colours). Note that the longest edges of the elements are the second edges. Also note that the i-th edge of an element may not be the the i-th edge of a neighbour element ($i = 1, 2, 3$). For instance, the edge $\text{conv}\{\mathbf{x}_4, \mathbf{x}_2\}$ is the 1-st edge of element K_8 and the 3-rd edge of element K_7.

2 Longest Edge Bisection

The Longest Edge Bisection (LEB) algorithm [6] is a special case of the Newest Vertex Bisection algorithm [2, 4, 7].

Let $K \in \mathcal{T}_0$ be an element of the mesh $\mathcal{T}_0$ at the 0-th iteration (in other words, after 0 refinements). The NVB algorithm chooses a *reference* edge of K. The LEB algorithm chooses the longest edge as the reference one. Then, successive refinements of $\mathcal{T}_0$ are as follows:

(a) the midpoint $\mathbf{x}_L$ of the longest edge of K becomes a new node, and K is bisected along $\mathbf{x}_L$ and the node opposite to the longest edge; in LEBREF2D the longest edge is the second (local) edge and the node opposite to the longest edge is the second (local) node (ref. to assumptions (A1)-(A2) and see Figure 1.3);

(b) the bisection produces two children of K and the two edges in front of $\mathbf{x}_L$ becomes the new reference edges of the children (they will be naturally the longest edges).

Usually, one iteration of the following steps above will results in *hanging nodes*, which are "not-a-node" of the mesh, that is, they lie on the edges of some element (see Figure 1.1(right)). Therefore, additional bisections have to be done in order to obtain a refined conforming mesh. This is shown in Figure 2.1, where longest edges are denoted by double lines and the edges that have to be bisected are denoted by black dots.

Figure 2.1(left) shows one iteration of steps (a)-(b). If there is an hanging node of K that has to be "resolved", we then assume that the edge where the hanging node lies on has to be bisected **together with** the longest edge of K which need to be bisected first (see Figure 2.1(middle-left

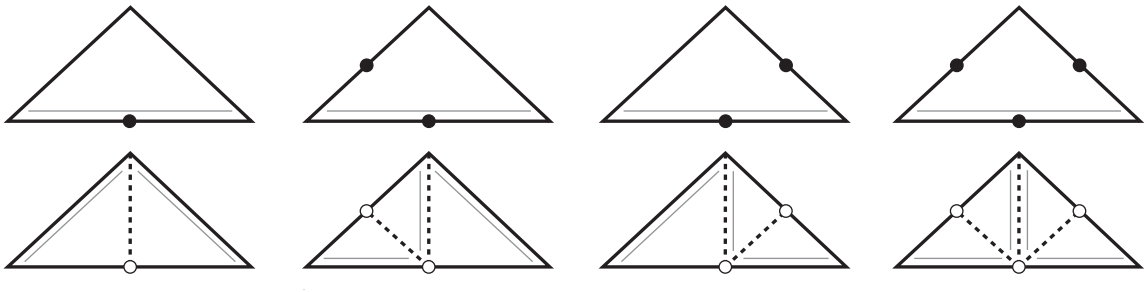


Figure 2.1. Longest edge bisection closures for 1 (left), 2 (middle-left and middle-right), and 3 edges marked for bisection. Double lines denotes the longest edges and black dots denotes the edges to be bisected.

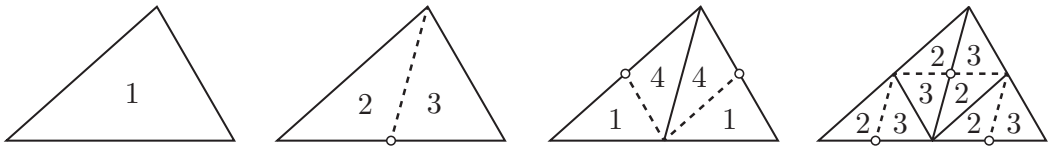


Figure 2.2. Similarity classes which occur using iterated longest edge bisections.

and middle-right)). If there are two hanging nodes, then all edges have to be bisected, doing three iteration of the steps (a)-(b) (Figure 2.1(right)). This way, an element can generate 2, 3, or 4 children according to the number of edges bisected. Obviously, the procedure terminates after a finite number of bisections due to the finite number of edges of the mesh. Notice that longest (newest) edge bisection iterations lead to only 4 similarity classes of elements (see Figure 2.2).

Figure 2.3 shows how the longest edge bisection iterations are performed by LEBREF2D. (a) It is given an example mesh $\mathcal{T}$ of 10 elements. (b) Initially 3 elements are marked for refinement. These elements are $\mathcal{M} = \{K_3, K_5, K_9\} \subseteq \mathcal{T}$. (c) Their longest edges are bisected. This introduces two hanging nodes, the red midpoints of the edges shared by elements K_9 and K_8, and K_5 and K_6, respectively. Then, in order to resolve the hanging nodes, elements K_8, K_6 are marked for refinement. (d) Refinement of K_6 does not lead to further hanging nodes. Instead, the bisection of the longest edge of K_8 introduces one more hanging node on the edge shared by K_7. (e) As before, also K_7 is marked for refinement. After the refinement of K_7 no further hanging nodes are introduced. The refinement is terminated. Notice that although $\#\mathcal{M} = 3$, the final number of elements that have been refined is 6.

3 Detail-grid

We first define what a uniform refinement is. We say that $\widehat{\mathcal{T}}$ is a *uniform refinement* of a given mesh $\mathcal{T}$ if all edges of $\mathcal{T}$ are bisected, i.e., the refinement of each element produces 4 children; in particular $\#\widehat{\mathcal{T}} = 4 \cdot \#\mathcal{T}$.

By *detail-grid* we refer to as the "mesh" of edge-midpoints that should be introduced, or added, to to current mesh $\mathcal{T}$ in order to obtain the uniformly refined mesh $\widehat{\mathcal{T}}$. That is, given a mesh stored in `MESHX`, its associated detail-grid is the data structure `MESHY` containing information about all edge-midpoints coordinates, position, and numbers.

The terminology comes from the Finite Element Method. Suppose that $X = X(\mathcal{T})$ is the space of (P1) piecewise-linear basis hat functions associated with the nodes of the mesh $\mathcal{T}$ and $Y = Y(\mathcal{T})$ is the space of (P1) piecewise-linear basis functions associated with edge midpoints of $\mathcal{T}$ (i.e., these hat functions equal to 1 at edge-midpoints and equal to 0 at mesh nodes). The space Y is called the *detail space* of X. In particular, the enriched space $\widehat{X} = \widehat{X}(\widehat{\mathcal{T}}) = X \oplus Y$ is associated with the uniform refinement $\widehat{\mathcal{T}}$ of $\mathcal{T}$.

Figure 3.1 shows an example of detail-grid for a mesh $\mathcal{T}$ (in blue) with 4 elements. In green it

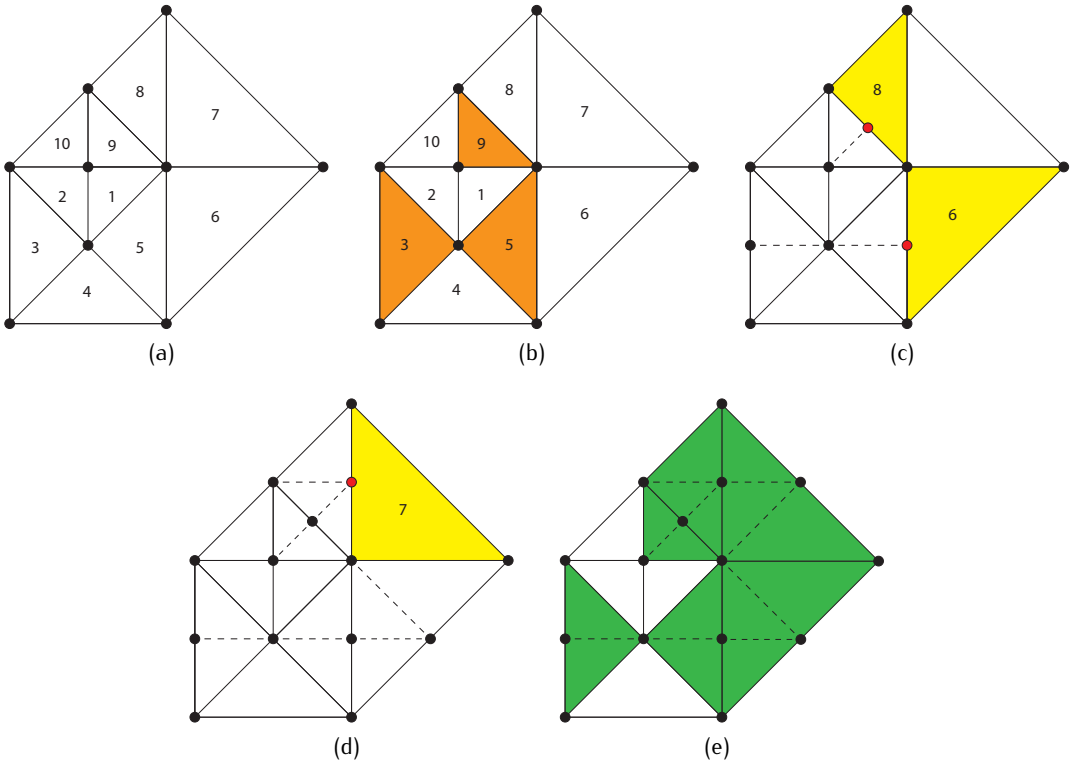


Figure 2.3. (a) Initial mesh $\mathcal{T}$ with 10 elements. (b) Marked elements $\mathcal{M}$ (in orange). (c) Refinement of the marked elements may introduce hanging nodes. Elements sharing the introduced hanging nodes are also marked for refinement. (d) Refinement of the new marked elements. (e) Final refined mesh (total refined elements in green).

is showed the uniform refinement which would introduces the edge-midpoints. Here, the refinement is done by 3 longest edge bisection iterations on each element. Notice, that edge-midpoints have their own new enumeration which is independent by the enumeration of the nodes. In particular, this means that the edge-midpoints' numbers are also the (global) numbers of the edges of the mesh. Moreover, the edge-midpoints of an element K are stored so that the j-th local midpoint of K is the one lying on the j-th edge of K, $j = 1, 2, 3$; see `MESHY.elem` in Figure 3.1 and cf. Figure 1.3.

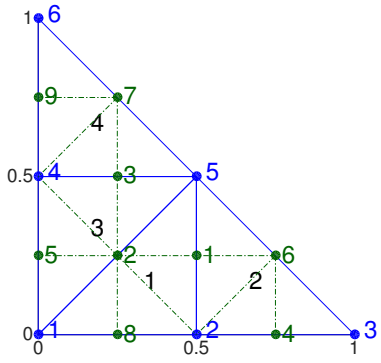
The function `detailgrid` (see Section 5) takes in input a mesh `MESHX` and returns the associated `MESHY`, which is a data structure composed of three field, `coord`, `elem`, and `bnd`, which store the coordinates of all midpoints, the element-mapping matrix with respect to the edge, and the midpoints lying on the boundary, respectively.

The function `detailgrid` is used by the adaptive refinement routines in order to store information about those midpoints that will become the new nodes of the refined mesh.

4 Solving PDEs

The local refinement routines can be particularly useful if LEBREF2D is applied, e.g., to an adaptive Finite Element code for solving Partial Differential Equations (PDEs) [5]. In such a case, at current iteration ℓ, one computes a discrete solution u_ℓ for a given underlying mesh $\mathcal{T}_\ell$ of the domain D of the PDE, and then estimates the local error contributions from each element of the mesh using some criteria (*a posteriori* error estimation [1]). This way, a set of marked elements $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ are determined and a new mesh $\mathcal{T}_{\ell+1}$ is obtained from $\mathcal{T}_\ell$ by refining its marked elements $\mathcal{M}_\ell$.

Although this is not the primal goal of the package, routines of LEBREF2D can be easily adapted and used in the setting of adaptive Finite Elements codes [3].



.coord

1	0.50	0.25
2	0.25	0.25
3	0.25	0.75
4	0.75	0.00
5	0.00	0.25
6	0.75	0.25
7	0.25	0.75
8	0.25	0.00
9	0.00	0.75

.elem

1	1	2	8
2	4	6	1
3	5	2	3
4	3	7	9

.bnd

4
5
6
7
8
9

Figure 3.1. Example of detail grid for a given mesh $\mathcal{T}$ (in blue) with 4 elements. Representation of the associated MESHY structure's fields storing information about all edge-midpoints of $\mathcal{T}$. The detail grid is returned by the function detailgrid.

5 List of functions

In the following, we list the functions of LEBREF2D:

- [MESHX] = adjustunstructmesh(MESHX)
  If the input MESHX contains an unstructured mesh, the function changes the local order of the elements in MESHX.elem so that the longest edges of all elements are the second ones (a generic unstructured mesh may not satisfy this condition which is required by the refinement routine; see assumption (A2)). More info in help adjustunstructmesh.

- [MESHX.elem] = bisection(MMele,MMedge,markedge,MESHX.elem,MESHY.elem);
  Main routine performing the bisection of the marked elements. In particular, MMele is the overall set of marked elements, including the extra elements which have to be refined in order to avoid hanging nodes (see Figure 2.3), and MMedge is the associated set of overall marked edges which will be bisected.

  The output MESHX.elem is the element mapping matrix of the new refined mesh.

- [MESH] = crackdomain(reflev,slith)
  Structured crack domain $D = (-1,1)^2 \setminus (-1,0)\times\{0\}$ grid generation (square domain $(-1,1)^2$ with a crack along the line $(-1,0)\times\{0\}$).

  reflev is an optional input which denotes the initial *refinement level*. This is a non negative integer (i.e., 0, 1, 2,...) which indicates how many uniform refinements the default mesh (for reflev=0) consists of. The default mesh for reflev=0 has 9 vertices and 8 elements.

  slith denotes instead half of the maximum "height" of the crack. That is, the crack is created by connecting two points, $(-1,-\texttt{slith})$ and $(-1,+\texttt{slith})$ with the origin $(0,0)$. Default value slith $= 0.01$.

- [MESHY,edgelep] = detailgrid(MESHX,ixy)
  Linear detail-grid generator. For an input MESHX which contains information about a mesh $\mathcal{T}$, the function returns the data structure MESHY associated with all edge-midpoints that would be introduced by a uniform refinement of $\mathcal{T}$; cf. description in the Section 3 and Figure 3.1 to see how the data structure MESHY looks like.

  The extra output edgelep is the edge-midpoint element-position $N_E \times 4$ matrix (here, N_E is the number of edges of the mesh). Suppose that its i-th row is as follows:

  (i-th row) $\quad 3 \quad 4 \quad 3 \quad 1$

This mean that the i-th (global) edge-midpoint lies on the (i-th) edge of the mesh in common with elements K_3 and K_4, and that it lies on the 3-rd local edge of K_3 and on the 1-st local edge of K_4. For example, this is the case of the row number 3 for the `edgelep` matrix (i.e., the 3-rd edge of the mesh) in the example showed in Figure 3.1.

Basically, the first two entries (1-st, 2-nd columns) indicate which elements the edge-midpoints are shared by, while the second two entries (3-rd, 4-th columns) indicate which local edges of these elements the edge-midpoint lie on.

The full `edgelep` matrix for the example in Figure 3.1 is the following:

$$\texttt{edgelep} = \begin{bmatrix} 1 & 2 & 1 & 3 \\ 1 & 3 & 2 & 2 \\ 3 & 4 & 3 & 1 \\ 2 & 2 & 1 & 1 \\ 3 & 3 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 4 & 4 & 2 & 2 \\ 1 & 1 & 3 & 3 \\ 4 & 4 & 3 & 3 \end{bmatrix}$$

Notice that for boundary edge-midpoints, the 1-st and 2-nd columns as well as the 3-rd and 4-th columns are repeated (in the example there are only 3 internal edge-midpoints and 6 boundary edge-midpoints).

Finally, `ixy` is an optional input to decide whether or not the coordinates of the edge-midpoints, i.e. the field `MESHY.coord`, have to be computed or not.

- `[edgenodes] = edge2nodes(MESHX,edgenum,edgelep)`
  Given in input the edge number `edgenum`, it returns their two extremal nodes $\mathbf{x}_k, \mathbf{x}_m$, i.e., such that the `edgenum`-th edge of the mesh `MESHX` is $\text{conv}\{\mathbf{x}_k, \mathbf{x}_m\}$.

  `edgelep` is an optional input to be used when the detail grid is not available; see `detailgrid`.

- `[MMele,MMedge] = getallmarkelem(Mele,MESHY.elem,edgelep)`
  Given in input the set `Mele` of marked elements for refinement, it returns the set of overall marked elements to be refined so that no hanging nodes will be introduced during the mesh refinement step. Then, the output `MMele` is the union of the set `Mele` and all the additional elements containing hanging nodes that would be introduced during the refinement of `Mele` (see example in Figure 2.3).

  The second output `MMedge` is the set of overall marked edges associated with `MMele` that have to be bisected.

  Note that no real mesh refinement is performed in this function.

- `lebdemo(domain)`
  Demo performing automatic adaptive mesh refinements of 4 available domains according to the optional input `domain`:

  - `domain` $= 1$ for structured square domain $D = (-1, 1)^2$; see `squaredomain`;
  - `domain` $= 2$ for structured the L-shaped domain $D = (-1, 1)^2 \setminus (-1, 0]^2$; see `lshapedomain`;
  - `domain` $= 3$ for unstructured L-shaped domain $D = (-1, 1)^2 \setminus (-1, 0]^2$; see `lshapedomainunstruct`;
  - `domain` $= 4$ for structured crack domain $D = (-1, 1)^2 \setminus (-1, 0) \times \{0\}$; see `crackdomain`.

Default `domain` $= 1$. Throughout iterations, the number of marked elements are chosen randomly using a certain threshold parameter `markpar` $\in (0, 1]$ and the MATLAB built-in function `randperm`:

```
Mele = randperm(nel,ceil(markpar*nel))'.
```

Here, `nel` is the number of current elements of the mesh. Notice that if `markpar` $= 1$, all elements are marked.

- `[MESHX,MMele]` $=$ `lebmeshref(MESHX,Mele)`
  Main routine for the mesh refinement step. It takes in input the current mesh $\mathcal{T}_\ell$ stored in `MESHX` and the set `Mele` of marked elements $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ and returns the new refined mesh $\mathcal{T}_{\ell+1}$ as well as the set `MMele` of overall elements that have been refined from $\mathcal{T}_\ell \to \mathcal{T}_{\ell+1}$.

- `[MESH]` $=$ `lshapedomainunstruct`
  Unstructured L-shaped domain $D = (-1, 1)^2 \setminus (-1, 0]^2$ grid generation;

- `[MESH]` $=$ `lshapedomain(reflev)`
  Structured L-shaped domain $D = (-1, 1)^2 \setminus (-1, 0]^2$ grid generation;

  `reflev` is an optional input which denotes the initial refinement level; cf. description in `crackdomain`.

- `plotmarkedelem(MESH,Mele,plotelem,plotvtx)`
  Plots an input mesh and a set of marked elements `Mele` with colour. Default colour is orange.

  `plotelem` and `plotvtx` are optional inputs to plot also the elements' numbers and the nodes' numbers, respectively.

- `plotmesh(MESH,titleplot,plotelem,plotvtx)`
  Plots the input mesh. `titleplot` is an optional input for the title of the plot.

  `plotelem` and `plotvtx` are also optional inputs; see description in `plotmarkedelem`.

- `plotmeshxandy(MESHX,MESHY,reftype,plotelem,plotvtx)`
  Plots the mesh `MESHX` and its associated detail-grid `MESHY`.

  `reftype` decides the type of uniform refinements the detail-grid is associated with. They are the *red* (`reftype` $= 1$) and *bisec3* (`reftype` $= 2$) refinement.

  Figure 5.1 shows an example. The *red* uniform refinement refines an element by connecting the 3 edge-midpoints. This introduces a "central" child in the middle of the element (Figure 5.1(a)). The *bisec3* uniform refinement is the result of 3 longest edge bisection iterations on each element (Figure 5.1(c)). The plots on the right show the resulted uniformly refined meshes if either red or bisec3 refinement is done (Figure 5.1(b) and (d), respectively).

  Remind that a detail-grid `MESHY` does not provide information about the type of bisections to be done. It only store information about midpoints. However, a mesh resulted by a bisec3 uniform refinement, as in Figure 5.1(d), is what naturally follows by longest edge bisection iterations.

  `plotelem` and `plotvtx` are optional inputs; see description in `plotmarkedelem`.

- `[MESH]` $=$ `squaredomain(reflev)`
  Structured square domain $D = (-1, 1)^2$ grid generation.

  `reflev` is an optional input which denotes the initial refinement level; cf. description in `crackdomain`.

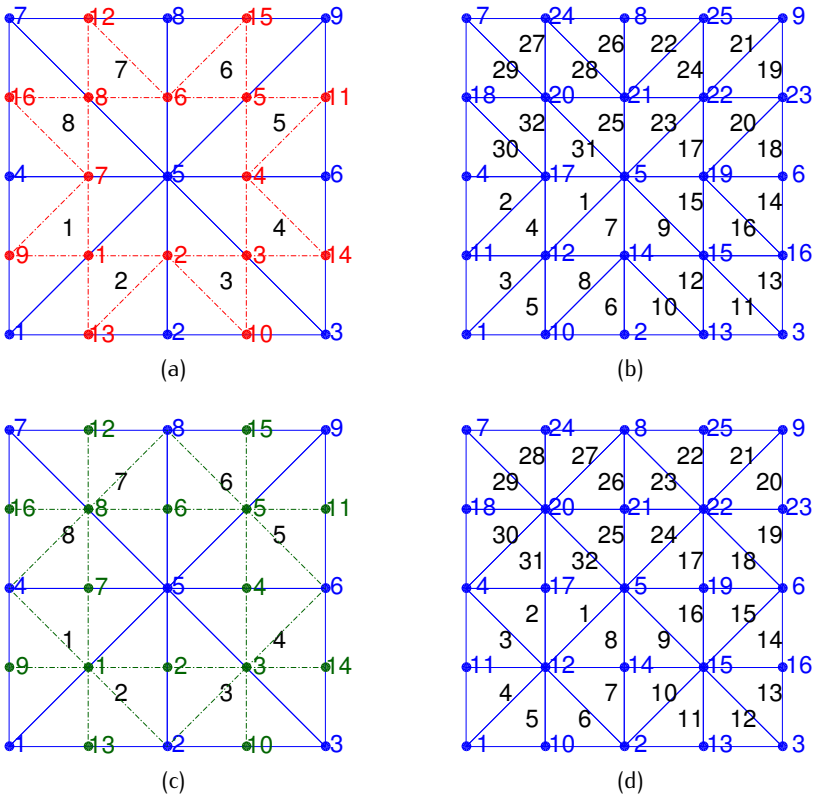


Figure 5.1. Two detail-grids for a given mesh $\mathcal{T}$ and associated uniformly refined meshes $\widehat{\mathcal{T}}$. Blue numbers denotes the nodes of $\mathcal{T}$ and black numbers the elements. (a)-(b) Detail-grid (edge-midpoints in red) corresponding to a *red* uniform refinement; (c)-(d) Detail-grid (edge-midpoints in green) corresponding to a *bisec3* uniform refinement.

- `[MESHREF] = unimeshref(MESH,reftype,iplot)`
  Given an input mesh `MESH`, it returns the uniformly refined mesh `MESHREF`. The input `reftype` decides the refinement-type, red (`reftype` $= 1$) or bisec3 (`reftype` $= 2$); see Figure 5.1.
  `iplot` is an optional input to plot the uniformly refined mesh.

6 Useful commands

- number of elements and number of nodes of the mesh, respectively:

```
nel = size(MESH.elem,1);
nvtx = size(MESH.coord,1);
```

- recover coordinates of internal and boundary nodes, respectively:

```
xyint = MESH.coord(MESH.int,:);
xybd = MESH.coord(MESH.bnd,:);
```

- get the nodes of the k-th element:

```
elnodes = MESH.elem(k,:);
```

- get all elements sharing the node $\mathbf{x}_k$:

```
find( sum( ismember(MESH.elem, k ), 2) ) ;
```

- number of edge-midpoints (and then edges) of the mesh (using `MESHY`):

$$\texttt{nedg} = \texttt{size(MESHY.coord,1);}$$

- get the midpoints of the k-th element (using `MESHY`):

$$\texttt{midpel} = \texttt{MESHY.elem(k,:);}$$

- recover the two elements sharing the k-th edge of the mesh (using `MESHY`):

$$\texttt{find(sum(ismember(MESHY.elem, k), 2));}$$

- recover the 3 edge-neighbors of the i-th element K_i, i.e., the elements which share at most one edge with K_i (using `MESHY` and `edgelep`):

$$\begin{aligned}\texttt{neigh} &= \texttt{edgelep(MESHY.elem(i,:), 1:2);}\\ \texttt{neigh} &= \texttt{neigh(neigh} \sim\texttt{= i);}\end{aligned}$$

  If K_i is a boundary element, then $\#\texttt{neigh} = 2$ instead of 3.

References

[1] Mark Ainsworth and J. Tinsley Oden, *A posteriori error estimation in finite element analysis*, Pure and Applied Mathematics (New York), Wiley, 2000.

[2] E. Bänsch, *Local mesh refinement in* 2 *and* 3 *dimensions*, Impact Comput. Sci. Engrg., 3 (1991), pp. 181–191.

[3] A. Bespalov and L. Rocchi, *Stochastic T-IFISS*, January 2018. Available online at http://web.mat.bham.ac.uk/A.Bespalov/software/index.html#stoch_tifiss.

[4] I. Kossaczký, *A recursive approach to local mesh refinement in two and three dimensions*, J. Comput. Appl. Math., 55 (1994), pp. 275–288.

[5] R. H. Nochetto and A. Veeser, *Primer of Adaptive Finite Element Methods*, in Multiscale and Adaptivity: Modeling, Numerics and Applications, vol. 2040, Springer-Verlag Berlin Heidelberg, 2012, pp. 125–225.

[6] M. C. Rivara, *Mesh refinement processes based on the generalized bisection of simplices*, SIAM J. Numer. Anal., 21 (1984), pp. 604–613.

[7] R. Verfürth, *A Review of A Posteriori Error Estimation and Adaptive Mesh-Rrefinement Techniques*, Adv. Numer. Math., Wiley-Teubner, 1996.