Implementing Delaunay and Advancing Front Meshing

Saad Qadeer

March 21, 2016

1 Introduction

In this study, we implement the Delaunay and Advancing Front algorithms in two dimensions using mesh-sizing functions.

2 Delaunay Meshing

Let $\Omega \subset \mathbb{R}^2$ be a polygon and let h be a mesh-sizing function on Ω; we then require that $h > 0$. We implement a Delaunay meshing algorithm that sizes the elements as specified by h.

We first consider placing nodes on the boundary. The corners of Ω must necessarily be included in the set of nodes. Consider the edge e_i between two such adjacent corner nodes $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ and let $\mathbf{p}$ be the unit vector that points from one to another. Suppose we have admitted a point $\mathbf{x}_0$ on e_i. To find the next valid node on this boundary, we compare the jump Δx to the $h(\mathbf{x}_0 + \mathbf{p}\Delta x)$. Assuming Δx is small, we have

$$\begin{aligned} h(\mathbf{x}_0 + \mathbf{p}\Delta x) &= \Delta x \\ h(\mathbf{x}_0) + (\Delta x)\mathbf{p} \cdot \nabla h(\mathbf{x}_0) &= \Delta x \\ \Delta x &= \frac{h(\mathbf{x}_0)}{1 - \mathbf{p} \cdot \nabla h(\mathbf{x}_0)}. \end{aligned}$$

Starting from $\mathbf{x}_i$ therefore, we can compute the sizes of the jumps along e_i that take us to $\mathbf{x}_{i+1}$. Care must be taken, however, to prevent points from falling too closely to $\mathbf{x}_{i+1}$. This is achieved by requiring that $\|\mathbf{x}_{i+1} - \mathbf{x}_0\| > 2\Delta x$ at each step.

Once the boundary nodes have been determined, a Delaunay triangulation is carried out. To refine this mesh according to h, we first detect the "worst" triangle. In general, this is the triangle whose edge lengths are much larger compared to the values of h at the triangle nodes. To be more precise, we compute

$$\frac{\text{sum of edge lengths}}{\text{sum of } h\text{-values at the nodes}}$$

for each triangle and identify the one for which this quantity is the largest. This is continued until this quantity is less than one for all the triangles.

Next, we find the circum-center $\mathbf{c}$ and circum-radius r of this triangle. To accept $\mathbf{c}$ as a valid node, it must satisfy two conditions:

(a) $\mathbf{c} \in \Omega$;

(b) the orthogonal distance from $\mathbf{c}$ to any of the edges of Ω must be at least αr, where $0 < \alpha < 1$ is a constant to be set. From observations, $\alpha = 0.5$ appears to work reasonably well.

The first condition above is obvious. The justification for (b) is that otherwise, circumcenters for triangles on the boundaries of Ω can lie very close to the boundary, yielding triangles with large obtuse angles. If any of these conditions fails, we replace $\mathbf{c}$ by the midpoint of the triangle edge that differs most from the h-values at its nodes.

With the new point decided, we re-triangulate using the Bowyer-Watson algorithm. To avoid re-triangulating the entire domain, we find the triangles whose circum-circles contain the latest point $\mathbf{c}$. These triangles are deleted to obtain a void, which is then triangulated by calling the Delaunay command. Care must be taken to ensure that the resulting triangles lie inside Ω.

The MATLAB function `DelMesh` implements this formulation. As inputs, it requires

- `pv`: a $2 \times n$ matrix recording the nodes of the polygon Ω when going around it counter-clockwise. The first and last nodes must be the same.
- `h`: the mesh-sizing function; this is required to be positive on Ω.

The outputs are the mesh as well as

- `p`: list of nodes.
- `t`: list of triangles in terms of node positions as in `p`.
- `e`: boundary nodes.

3 Advancing Front

Let Ω and h be as in Section 2. We initiate the meshing process by choosing boundary nodes exactly as we did above. The initial front is chosen to be the entire boundary.

The advancing front method proceeds by choosing a section of the front that is moved inwards. This choice is fairly arbitrary; we take it to be the largest distance between two adjacent nodes on the front. Let the two nodes be $\mathbf{x}_A$ and $\mathbf{x}_B$. A third node $\mathbf{x}_C$ is chosen inside the region enveloped by the front so that $(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C)$ is an equilateral triangle.

The point $\mathbf{x}_C$ is now moved by a distance Δs along the perpendicular bisector of the segment between $\mathbf{x}_A$ and $\mathbf{x}_B$ such that

$$h(\mathbf{x}_C + \mathbf{q}\Delta s) \;=\; L(\Delta s) \tag{1}$$

where $\mathbf{q}$ is the unit vector orthogonal to the segment between $\mathbf{x}_A$ and $\mathbf{x}_B$ and pointing towards the untriangulated territory; and $L(\Delta s) = \|\mathbf{x}_B - (\mathbf{x}_C + \mathbf{q}\Delta s)\|$ is the distance from the new point to $\mathbf{x}_B$. The condition ensures that the value of the mesh-sizing function at

the new point agrees with its distance from $\mathbf{x}_B$ (and equivalently $\mathbf{x}_A$). Assuming that Δs is small, we can consider Taylor expansions of both sides of (1) and get

$$\begin{aligned} h(\mathbf{x}_C) + (\Delta s)\nabla h(\mathbf{x}_C) \cdot \mathbf{q} &= L(0) + (\Delta s)L'(0) \\ \Delta s &= \frac{L(0) - h(\mathbf{x}_C)}{\nabla h(\mathbf{x}_C).\mathbf{q} - L'(0)}. \end{aligned} \tag{2}$$

Note that $d = L(0)$ is merely the edge length of the equilateral triangle $(\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C)$. It can also be easily shown that

$$L'(0) = \frac{\mathbf{q}.(\mathbf{x}_C - \mathbf{x}_B)}{d}$$

so we obtain a more refined estimate $\mathbf{x}'_C = \mathbf{x}_C + (\Delta s)\mathbf{q}$ for the new candidate.

Next, we find all the nodes (excluding $\mathbf{x}_A$ and $\mathbf{x}_B$) in the current front that are less than $\frac{1}{2}L(\Delta s)$ away from $\mathbf{x}'_C$. If a point is found that yields a valid triangle, it is accepted and the process moves forward. If no such point is found, we tentatively consider $\mathbf{x}'_C$; if the exterior angles of the resulting triangle are not too small, we accept this as our new point. In the eventuality that this also fails, we increase the searching distance by 20% and keep iterating till we determine a valid triangle.

Once a new triangle is decided upon, we need to update the front as well as keep track of voids that may have been created by building a triangle using non-adjacent nodes in a front. These voids are filled in later on using the same method recursively.

With all of Ω triangulated, we may wish to avoid certain "bad" triangles, in particular those that have large obtuse angles (i.e., greater that 120°). Such triangles are weeded out by applying edge flipping on offensive edges.

The MATLAB function `AFMesh` implements this formulation. The inputs and outputs are precisely the same as for `DelMesh`.