Generating 2D-Domain Mesh in Using MATLAB Program

MATLAB program: domainMesh2d produces the following outputs:

- **p** : the node positions which is N-by-2 array containing the x, y coordinates for each of the N nodes.
- **t** : the triangle indices. The row associated with each triangle has 3 integer entries to specify node numbers in that triangle.
- **fd** : distance function for the domain geometry. This function returns the signed distance from each node location p to the closest boundary.
- **h(x,y)** : the (relative) desired edge length function as a function fh which returns h for all input points.
- **h0** : the distance between points in the initial distribution p0. For uniform meshes (h(x, y) = constant), the element size in the final mesh will usually be a little larger than this input.
- **bbox = [xmin, ymin; xmax, ymax]** : an array of the bounding box for the region.
- **pfix** : an array with two columns of the fixed node positions.
- **varargin** : additional parameters to the functions fd and fh.

```
function [p,t]=domainMesh2d(fd,fh,h0,bbox,pfix,varargin)
%%  setting parameters
dptol=.001; ttol=.1; Fscale=1.2; deltat=.2; geps=.001*h0; deps=sqrt(eps)*h0;
% 1. Create initial distribution in bounding box (equilateral triangles)
[x,y]=meshgrid(bbox(1,1):h0:bbox(2,1),bbox(1,2):h0*sqrt(3)/2:bbox(2,2));
x(2:2:end,:)=x(2:2:end,:)+h0/2;                          % Shift even rows
p=[x(:),y(:)];                                           % List of node coordinates
%
% 2. Remove points outside the region, apply the rejection method
p=p(feval(fd,p,varargin{:})<geps,:);                     % Keep only d<0 points
r0=1./feval(fh,p,varargin{:}).^2;                        % Probability to keep point
p=[pfix; p(rand(size(p,1),1)<r0./max(r0),:)];            % Rejection method
N=size(p,1);                                             % Number of points N
pold=inf;                                                 % For first iteration
%
while 1
%
% 3. Retriangulation by the Delaunay algorithm
if max(sqrt(sum((p-pold).^2,2))/h0)>ttol          % Any large movement?
pold=p;                                           % Save current positions
t=delaunayn(p);                                   % List of triangles
pmid=(p(t(:,1),:)+p(t(:,2),:)+p(t(:,3),:))/3;     % Compute centroids
t=t(feval(fd,pmid,varargin{:})<-geps,:);           % Keep interior triangles
%
% 4. Describe each bar by a unique pair of nodes
bars=[t(:,[1,2]);t(:,[1,3]);t(:,[2,3])];          % Interior bars duplicated
bars=unique(sort(bars,2),'rows');                  % Bars as node pairs
%
% 5. Graphical output of the current mesh
trimesh(t,p(:,1),p(:,2),zeros(N,1))
view(2),axis equal,axis off,drawnow
end
%
% 6. Move mesh points based on bar lengths L and forces F
barvec=p(bars(:,1),:)-p(bars(:,2),:);                    % List of bar vectors
L=sqrt(sum(barvec.^2,2)); % L = Bar lengths
hbars=feval(fh,(p(bars(:,1),:)+p(bars(:,2),:))/2,varargin{:});
L0=hbars*Fscale*sqrt(sum(L.^2)/sum(hbars.^2));           % L0 = Desired lengths
F=max(L0-L,0);                                            % Bar forces (scalars)
```

```
Fvec=F./L*[1,1].*barvec; % Bar forces (x,y components)
Ftot=full(sparse(bars(:,[1,1,2,2]),ones(size(F))*[1,2,1,2],[Fvec,-Fvec],N,2));
Ftot(1:size(pfix,1),:)=0;                             % Force = 0 at fixed points
p=p+deltat*Ftot;                                      % Update node positions
% 7. Bring outside points back to the boundary
d=feval(fd,p,varargin{:}); ix=d>0;                    % Find points outside (d>0)
dgradx=(feval(fd,[p(ix,1)+deps,p(ix,2)],varargin{:})-d(ix))/deps; % Numerical
dgrady=(feval(fd,[p(ix,1),p(ix,2)+deps],varargin{:})-d(ix))/deps; % gradient
p(ix,:)=p(ix,:)-[d(ix).*dgradx,d(ix).*dgrady];        % Project back to boundary
%
% 8. Termination criterion: All interior nodes move less than dptol (scaled)
if max(sqrt(sum(deltat*Ftot(d<-geps,:).^2,2))/h0) < dptol,
      break;
end
return
end
```

Some Help Functions

```
%% Circle
function d=dcircle(p,xc,yc,r)
d=sqrt((p(:,1)-xc).^2+(p(:,2)-yc).^2)-r;
%% Rectangle
function d=drectangle(p,x1,x2,y1,y2)
d=-min(min(min(-y1+p(:,2),y2-p(:,2)), ...
-x1+p(:,1)),x2-p(:,1));
%% Union
function d=dunion(d1,d2)
d=min(d1,d2);
%% Difference
function d=ddiff(d1,d2)
d=max(d1,-d2);
%% Intersection
function d=dintersect(d1,d2)
d=max(d1,d2);
%% Shift points
function p=pshift(p,x0,y0)
p(:,1)=p(:,1)-x0;
p(:,2)=p(:,2)-y0;
%% Rotate points around origin
function p=protate(p,phi)
A=[cos(phi),-sin(phi);sin(phi),cos(phi)];
p=p*A;
%% Interpolate d(x,y) in meshgrid matrix
function d=dmatrix(p,xx,yy,dd,varargin)
d=interp2(xx,yy,dd,p(:,1),p(:,2),'*linear');
% Interpolate h(x,y) in meshgrid matrix
function h=hmatrix(p,xx,yy,dd,hh,varargin)
h=interp2(xx,yy,hh,p(:,1),p(:,2),'*linear');
%% Uniform h(x,y) distribution
function h=huniform(p,varargin)
h=ones(size(p,1),1);
```

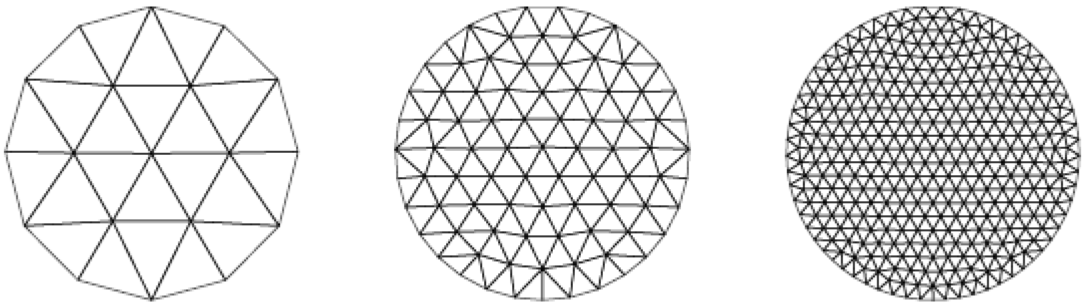
Example 1. Unit Circle.

Let $d = \sqrt{x^2 + y^2}$-1 be specified as an inline function.

- For a uniform mesh, h(x, y) returns a vector of 1's. The circle has bounding box $-1 \le x \le 1, -1 \le y \le 1$, with no fixed points. A mesh with element size approximately h0 = 0.2 is generated with two lines of code:

```
>> fd=inline('sqrt(sum(p.^2,2))-1','p');
>> [p,t]=distmesh2d(fd,@huniform,0.2,[-1,-1;1,1],[]);
```

The plots (1a), (1b), and (1c) show the resulting meshes for h0 = 0.4, h0 = 0.2, and h0 = 0.1.

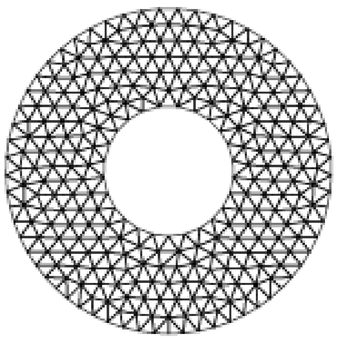


Inline functions are defined without creating a separate file. The first argument is the function itself, and the remaining arguments name the parameters to the function (help inline brings more information). Please note the comment near the end of the paper about the relatively slow performance of inline functions.

Another possibility is to discretize d(x, y) on a Cartesian grid, and interpolate at other points using the dmatrix function:

```
>> [xx,yy]=meshgrid(-1.1:0.1:1.1,-1.1:0.1:1.1); % Generate grid
>> dd=sqrt(xx.^2+yy.^2)-1; % d(x,y) at grid points
>> [p,t]=distmesh2d(@dmatrix,@huniform,0.2,[-1,-1;1,1],[],xx,yy,dd);
```

Example 2. Unit Circle with Hole. Removing a circle of radius 0.4 from the unit circle gives the distance function $d(x,y) = |0.7 - \sqrt{x^2 + y^2}| - 0.3$:

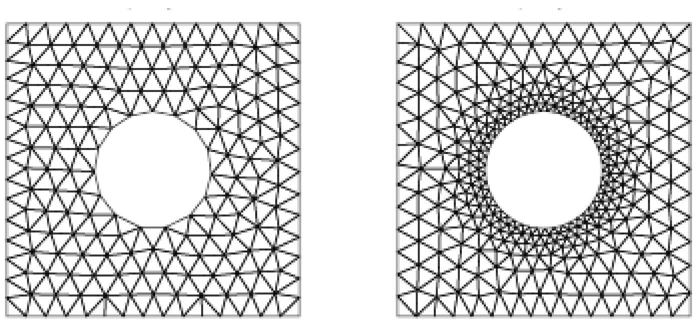


```
>> fd=inline('-0.3+abs(0.7-sqrt(sum(p.^2,2)))');
>> [p,t]=distmesh2d(fd,@huniform,0.1,[-1,-1;1,1],[]);
```

Equivalently, d(x, y) is the distance to the difference of two circles:

```
>> fd=inline('ddiff(dcircle(p,0,0,1),dcircle(p,0,0,0.4))','p');
```

Example 3. Square with Hole. We can replace the outer circle with a square, keeping the circular hole. Since our distance function drectangle is incorrect at the corners, we fix those four nodes (or write a distance function involving square roots):



```
>> fd=inline('ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.4))','p');
>> pfix=[-1,-1;-1,1;1,-1;1,1];
>> [p,t]=distmesh2d(fd,@huniform,0.15,[-1,-1;1,1],pfix);
```

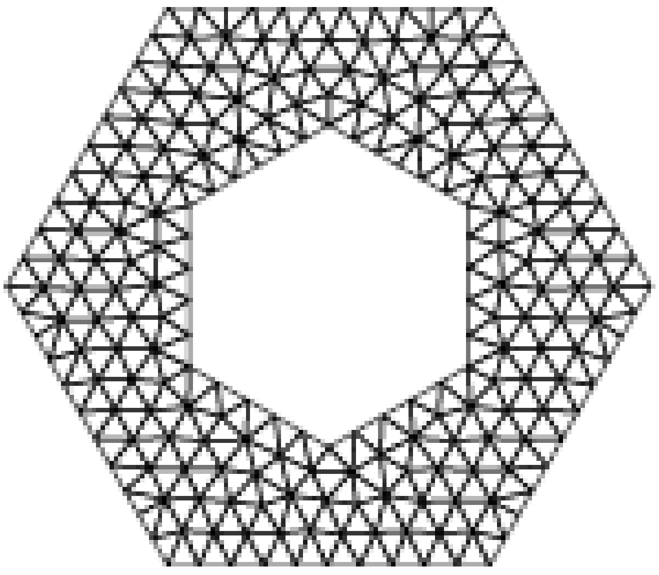
A non-uniform h(x, y) gives a finer resolution close to the circle (mesh (3b)):

```
>> fh=inline('min(4*sqrt(sum(p.^2,2))-1,2)','p');
>> [p,t]=distmesh2d(fd,fh,0.05,[-1,-1;1,1],pfix);
```

Example 4. Polygons. It is easy to create dpoly (not shown here) for the distance to a given polygon, using MATLAB's inpolygon to determine the sign. We mesh a regular hexagon and fix its six corners:

```
>> phi=(0:6)'/6*2*pi;
>> pfix=[cos(phi),sin(phi)];
>> [p,t]=distmesh2d(@dpoly,@huniform,0.1,[-1,-1;1,1],pfix,pfix);
```

Note that pfix is passed twice, first to specify the fixed points, and next as a parameter to dpoly to specify the polygon. In the following figure, we also removed a smaller rotated hexagon by using ddiff.



Example 5. Geometric Adaptivity. Here we show how the distance function can be used in the definition of h(x, y), to use the local feature size for geometric adaptivity. The half-plane y > 0 has d(x, y) = −y, and our d(x, y) is created by an intersection and a difference:

$$d_1 = \sqrt{x^2 + y^2} - 1 \qquad (1)$$
$$d_2 = \sqrt{(x + 0.4)^2 + y^2} - 0.55 \qquad (2)$$
$$d = \max(d_1, -d_2, -y). \qquad (3)$$

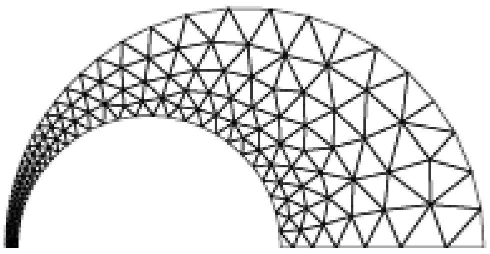
Next, we create two element size functions to represent the finer resolutions near the circles. The element sizes $h_1$and h_2 increase with the distances from the boundaries (the factor 0.3 gives a ratio 1.3 between neighboring elements):

$$h_1(x, y) = 0.15 - 0.2d_1 \qquad (4)$$
$$h_2(x, y) = 0.06 - 0.2d_2 \qquad (5)$$

These are made proportional to the two radii to get equal angular resolutions. Note the minus sign for d_1 since it is negative inside the region. The local feature size is the distance between boundaries, and we resolve this with at least three elements:

$$h_3(x, y) = (d_2 - d_1)/3. \qquad (6)$$

Finally, the three size functions are combined to yield the mesh as follows:



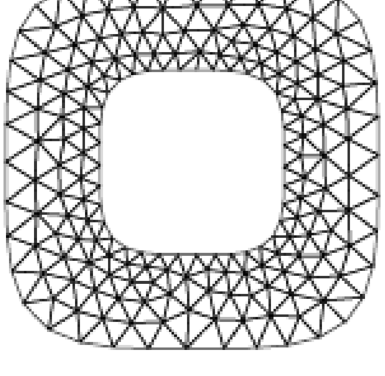
$$h = \min(h_1, h_2, h_3). \qquad (7)$$

The initial distribution had size h_0 = 0.05/3 and four fixed corner points.

Example 6. Implicit Expression. We now show how distance to level sets can be used to mesh non-standard geometries. We mesh the region between the level sets 0.5 and 1.0 of the superellipse

$$f(x, y) = (x^4 + y^4)^{1/4} \qquad (8)$$

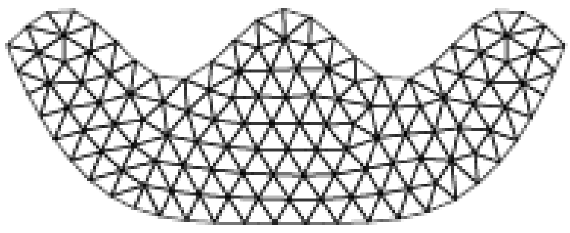
The boundaries of these geometries are not approximated by simpler curves, they are represented exactly by the given expressions. As the element size h_0 gets smaller, the mesh automatically fits to the exact boundary, without any need to refine the representation.



Example 7. Implicit Expression. We now show how distance to level sets can be used to mesh non-standard geometries. We mesh the region between the intersection of the following two regions:

$$y \leq \cos(x) \quad and \;\; y \geq 5\left(\frac{2\pi}{5\pi}\right)^4 - 5, \qquad (9)$$

With $-5\pi/2 \leq x \leq 5\pi/2$ and $-5 \leq y \leq 1$. The boundaries of these geometries are not approximated by simpler curves, they are represented exactly by the given expressions. As the element size h_0 gets smaller, the mesh automatically fits to the exact boundary, without any need to refine the representation.



Example 8. More Complex Geometry. This example shows a somewhat more complicated construction, involving set operations on circles and rectangles, and element sizes increasing away from two vertices and the circular hole.