**Function VoronoiBound User’s Instructions**

**Description**

Function VoronoiBound, developed for Matlab 2014b, calculates a Voronoi diagram with inner and outer bounds. It supplements the Matlab existing functions, Voronoi and VoronoiDiagram, by defining *finite inner and outer bounds*. Additionally, VoronoiBound is robust and capable of handling collinear points, whereas the Matlab existing function DelaunayTriangulation returns a null set. Function VoronoiBound should work with past and future Matlab versions, as long as the functions that it calls are compatible with Matlab 2014b.

**Input-Output**

[VX, VY] = VoronoiBound (PX, PY, OX, OY) produces a 2-D Voronoi diagram for the points in vectors PX and PY. The Voronoi diagram is bounded by the polygon defined by the vertices in vectors OX and OY. For the i-th point, the corresponding Voronoi region is the polygon formed by the vectors in the i-th cell entry of VX and VY, i.e., VX{i} and VY{i} – note the use of the curly brackets { } –. Points outside of the bounds return an empty Voronoi region.

[...] = VoronoiBound (PX, PY, OX, OY, IX, IY) adds an inner boundary to the calculation of the Voronoi diagram. The inner boundary is a polygon described by vertices in vectors IX and IY.

[VX, VY, L] = VoronoiBound (...) adds the logical vector L to flag the points that fall within the bounding polygon(s).

VoronoiBound (…) returns figures as the only output.

This function uses built-in and well documented Matlab functions.

**Example**

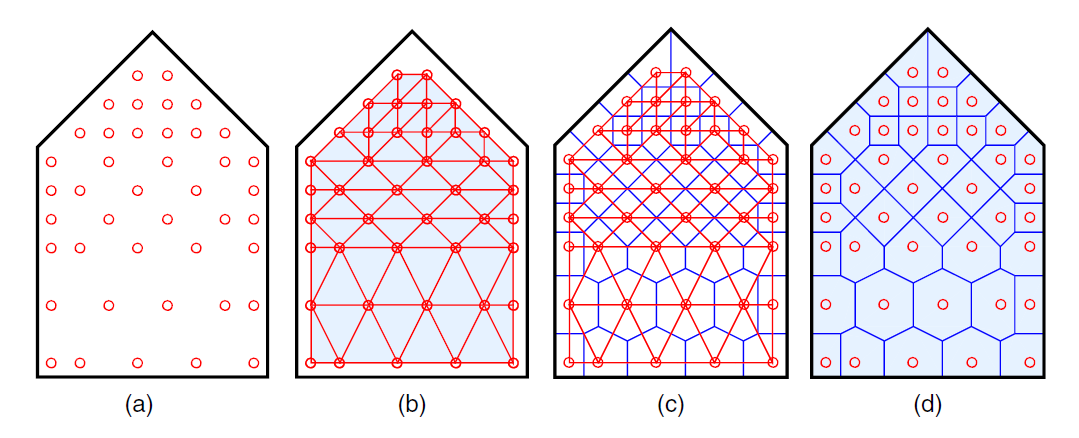
The command VoronoiBound () or VoronoiBound uses a set of 30 randomly generated taps, an inner boundary and an outer boundary and outputs the following figures. Markers “o” and “x” indicate taps that fall inside and outside of the boundaries, respectively.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

1. pressure taps and bounds; (b) Delaunay triangulation; (c) perpendicular bisectors; (d) Voronoi diagram

Fig. 1 Example of pressure tap tributary area assignment from 30 random points

**Application**



1. pressure taps; (b) Delaunay triangulation; (c) perpendicular bisectors; (d) Voronoi diagram

Fig. 2 Example of pressure tap tributary area assignment from TPU data

This function is useful in defining tributary areas of pressure taps regularly or irregularly placed on the outside surface of a wind tunnel model. For example, the Voronoi diagram is applied to a gable building end wall from Tokyo Polytechnic University’s database (Tamura 2012). In Figure 2a, circles indicate pressure tap locations. (The size of the circles is not representative of the size of the pressure taps.) In Figure 2b, Delaunay (1934) triangulation connects the point taps to form triangles that cover the entire space bounded by the taps without overlapping and do not have any tap within the triangle’s circumcircle. By drawing perpendicular bisectors to the sides of the Delaunay triangles (Figure 2c), one obtains a Voronoi diagram (Figure 2d). Regions formed from these bisectors contain one tap each and bound points that are closer to that tap than to any other tap. The area created around each tap is the tributary area of that tap.

**References**

Delaunay, B. (1934). “On the Empty Sphere.” In Memory of Georges Voronoi, *Bulletin of the USSR Academy of Sciences, Section: Mathematics and Natural Sciences*. 6, 793-800.

MATLAB (2014) MATLAB documentation 2014b, *The Mathworks, Inc*.

Tamura, Y. (2012). “Aerodynamic database of low-rise buildings.”

〈http://www.wind.arch.t-kougei.ac.jp/info\_center/windpressure/lowrise/mainpage.html〉 (Sep. 29, 2015).

Voronoi, G. (1908). “New Applications of Continuous Parameters to the Theory of Quadratic Forms.” *J. of Pure and Applied Mathematics*, 133(133), 97-178. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1515/crll.1908.133.97](https://dx.doi.org/10.1515%2Fcrll.1908.133.97).

**Contacts** Dat Duthinh, NIST (301) 975-4357 [dduthinh@nist.gov](mailto:dduthinh@nist.gov)

Brian M. Phillips, University of Maryland, College Park [(301) 405-1961](tel:%28301%29%20405-1961) [bphilli@umd.edu](mailto:bphilli@umd.edu)

**Disclaimer** Certain trade names or company products are specified in this document to specify adequately the procedure used. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the product is the best available for the purpose. The authors’ limited rights to the deployment of this program are limited by a license agreement between NIST and The MathWorks. The license agreement can be found at www.mathworks.com/license/. The authors, NIST, the University of Maryland, and The MathWorks and its licensors are excluded from all liability for damages or any obligation to provide remedial actions.

**Source code for Function VoronoiBound**

%% VoronoiBound Calculates a Voronoi diagram with inner and outer bounds.

% [VX,VY] = VoronoiBound(PX,PY,OX,OY) produces a 2D Voronoi diagram for

% the points in vectors PX and PY. The Voronoi diagram will be bounded by

% the polygon described by the vertices in vectors OX and OY.

% For the i-th point, the cooresponding Voronoi region is the polygon

% formed by the vectors in the i-th cell entry of VX and VY. Points

% outside of the bounds return an empty Voronoi region.

%

% [...] = VoronoiBound(PX,PY,OX,OY,IX,IY) adds an inner boundary to

% the calculation of the Voronoi diagram. The inner boundary is a polygon

% described by vertices in vectors IX and IY.

%

% [VX,VY,L] = VoronoiBound(...) adds the logical vector L to

% flag the points that fall within the bounding polygon(s).

%

% VoronoiBound(...) returns a figure as the only output.

%

% [...] = VoronoiBound() uses a set of example data.

% Last Updated: May 25, 2018; 12:10 PM EDT

% Brian Phillips; University of Maryland; bphilli@umd.edu

%% Inputs

if nargin==0

P = gallery('uniformdata',[30 2],0);

% P=[0.25 0.25; 0.25 0.75; 0.75 0.25; 0.75 0.75;0.5,1];

% P=[0.25 0.25; 0.5 0.25; 0.75 0.25; .9 0.25; .1 0.25];

x\_bound=[0;0;0.5;1;1;nan;0.65;0.65;0.35;0.35];

y\_bound=[0;1;1.5;1;0;nan;0.35;0.85;0.85;0.35];

end

if nargin>=2

% Matrix of (x,y) tap locations

Px=varargin{1}(:);

Py=varargin{2}(:);

P=[Px,Py];

x\_bound=[];

y\_bound=[];

end

if nargin>=4

x\_outer=varargin{3}(:);

y\_outer=varargin{4}(:);

% Clockwise outer boundary (vertices)

[x\_outer\_cw,y\_outer\_cw]=poly2cw(x\_outer,y\_outer);

x\_bound=x\_outer\_cw;

y\_bound=y\_outer\_cw;

end

if nargin>=6

x\_inner=varargin{5}(:);

y\_inner=varargin{6}(:);

% Counter-clockwise interior boundary (vertices)

[x\_inner\_ccw,y\_inner\_ccw]=poly2ccw(x\_inner,y\_inner);

x\_bound=[x\_outer\_cw;nan;x\_inner\_ccw];

y\_bound=[y\_outer\_cw;nan;y\_inner\_ccw];

end

%% Main

% Extract taps that fall within boundaries

[in,on]=inpolygon(P(:,1),P(:,2),x\_bound,y\_bound);

P\_inl=in==1|on==1;

P\_ind=find(P\_inl);

P\_in=P(P\_ind,:);

% Built-in MATLAB function

warning('off','MATLAB:triangulation:EmptyTri2DWarnId');

DT=delaunayTriangulation(P\_in);

% Ensure that Delaunay triangulation was successful

if isempty(DT.ConnectivityList)~=1

% Built-in MATLAB functions

[V,R]=voronoiDiagram(DT);

[vx,vy]=voronoi(DT);

vx=vx.';

vy=vy.';

% Eliminate zero-length vectors

vd=sqrt((vx(:,2)-vx(:,1)).^2+(vy(:,2)-vy(:,1)).^2);

vx(vd==0,:)=[];

vy(vd==0,:)=[];

% Find unique values in vx (and vy) to replace "infinite vertices"

[u,~,indu]=unique([vx(:),vy(:)],'rows');

uu=u(accumarray(indu,1)==1,:);

ux=uu(:,1);

uy=uu(:,2);

% Create infinite vectors to help bound infinite regions

kx=zeros(length(ux),2);

ky=zeros(length(uy),2);

for ii=1:length(ux)

[induu]=find(vx==ux(ii)&vy==uy(ii));

[indi,~]=ind2sub(size(vx),induu);

kx(ii,:)=[vx(indi,1);vx(indi,2)].';

ky(ii,:)=[vy(indi,1);vy(indi,2)].';

end

% Extend the infinite line segments

kd=sqrt((kx(:,2)-kx(:,1)).^2+(ky(:,2)-ky(:,1)).^2);

kextend=max(abs([x\_bound;y\_bound]))\*3;

kx(:,2)=kx(:,1)+(kx(:,2)-kx(:,1))./kd\*kextend;

ky(:,2)=ky(:,1)+(ky(:,2)-ky(:,1))./kd\*kextend;

% Sort the infinite line segments in counter-clockwise order

xmid=(max(P\_in(:,1))-min(P\_in(:,1)))/2;

ymid=(max(P\_in(:,2))-min(P\_in(:,2)))/2;

[~,kbound]=sort(atan2(ky(:,2)-ymid,kx(:,2)-xmid));

kbound(end+1)=kbound(1);

% Initialize variables

x\_trib\_in=cell(length(R),1);

y\_trib\_in=cell(length(R),1);

R\_new=cell(length(R),1);

V\_new=V;

% Loop over all tap tributary areas

for ii=1:size(R,1)

% Eliminate repeated vertices

[~,IA,~]=unique([V(R{ii},1),V(R{ii},2)],'rows','stable');

R{ii}=R{ii}(IA);

% Fix regions with infinite vertices

if any(R{ii}==1)

% Look for infinite vectors that begin closest to each end of unbounded region

v1\_ind=knnsearch([kx(:,1),ky(:,1)],[V(R{ii}(2),1),V(R{ii}(2),2)],'Distance','euclidean','k',2);

v2\_ind=knnsearch([kx(:,1),ky(:,1)],[V(R{ii}(end),1),V(R{ii}(end),2)],'Distance','euclidean','k',2);

flag1=kx(v1\_ind(1),1)==kx(v1\_ind(2),1)&&ky(v1\_ind(1),1)==ky(v1\_ind(2),1);

flag2=kx(v2\_ind(1),1)==kx(v2\_ind(2),1)&&ky(v2\_ind(1),1)==ky(v2\_ind(2),1);

% Method 1: use if infinite vectors begin in same location

if flag1==1||flag2==1

for jj=1:length(kbound)-1

% Trial vertices

v1=kbound(jj);

v2=kbound(jj+1);

Vx=[kx(v2,2);kx(v2,1);V(R{ii}(3:end-1),1);kx(v1,1);kx(v1,2)];

Vy=[ky(v2,2);ky(v2,1);V(R{ii}(3:end-1),2);ky(v1,1);ky(v1,2)];

% Update V if trial vertices enclose tap

if inpolygon(P\_in(ii,1),P\_in(ii,2),Vx,Vy)==1

lv=size(V,1);

V\_new(lv+1,:)=[Vx(1),Vy(1)];

V\_new(lv+2,:)=[Vx(end),Vy(end)];

R\_new{ii}=[lv+1,R{ii}(2:end),lv+2];

end

end

% Method 2: (quicker) use if infinite vectors begin in unique locations

else

% Update V using infinite vectors

lv=size(V,1);

V\_new(lv+1,:)=[kx(v1\_ind(1),2),ky(v1\_ind(1),2)];

V\_new(lv+2,:)=[kx(v2\_ind(1),2),ky(v2\_ind(1),2)];

R\_new{ii}=[lv+1,R{ii}(2:end),lv+2];

end

else

R\_new{ii}=R{ii};

end

% Make voronoi regions clockwise

[xcw,ycw]=poly2cw(V\_new(R\_new{ii},1),V\_new(R\_new{ii},2));

% Calculate the portion of tributary area bound within limits

[x\_trib\_temp,y\_trib\_temp]=polybool('intersection',xcw,ycw,x\_bound,y\_bound);

x\_trib\_in{ii}=x\_trib\_temp;

y\_trib\_in{ii}=y\_trib\_temp;

end

% If Delaunay triangulation was unsuccessful (e.g., due to collinear points), add four bounding points

else

% Append list of points with four new bounding points

Pextend=max(abs([x\_bound;y\_bound]))\*10;

P\_add=[P\_in;[1 1; -1 1; -1 -1; 1 -1]\*Pextend];

% Built-in MATLAB functions

DT=delaunayTriangulation(P\_add);

[V,R]=voronoiDiagram(DT);

% Remove Voronoi regions created by additional points

R\_new=R(1:size(P\_in,1));

V\_new=V;

% Initialize variables

x\_trib\_in=cell(length(R\_new),1);

y\_trib\_in=cell(length(R\_new),1);

% Loop over all tap tributary areas

for ii=1:size(R\_new,1)

% Make Voronoi regions clockwise

[xcw,ycw]=poly2cw(V\_new(R\_new{ii},1),V\_new(R\_new{ii},2));

% Calculate the portion of tributary area bound within limits

[x\_trib\_temp,y\_trib\_temp]=polybool('intersection',xcw,ycw,x\_bound,y\_bound);

x\_trib\_in{ii}=x\_trib\_temp;

y\_trib\_in{ii}=y\_trib\_temp;

end

end

% Initialize variables

x\_trib=cell(size(P,1),1);

y\_trib=cell(size(P,1),1);

% Expand to all taps (including those outside bounds)

for ii=1:length(P\_ind)

x\_trib{P\_ind(ii)}=x\_trib\_in{ii};

y\_trib{P\_ind(ii)}=y\_trib\_in{ii};

end

%% Outputs

% Define outputs of function

if nargout==3

varargout={x\_trib,y\_trib,P\_inl};

elseif nargout==2

varargout={x\_trib,y\_trib};

else

varargout={};

end

%% Plot

if nargout==0

% Outer limits for plot based on outer boundaries

xmax=max(x\_bound);

xmin=min(x\_bound);

ymax=max(y\_bound);

ymin=min(y\_bound);

% Figure 1: Taps and bounds

figure

hold on

% Plot taps

plot(P(P\_inl,1),P(P\_inl,2),'ok')

plot(P(~P\_inl,1),P(~P\_inl,2),'xk')

% Plot surface bounds

if any(isnan(x\_bound))

ind=find(isnan(x\_bound));

plot([x\_bound(1:ind-1);x\_bound(1)],[y\_bound(1:ind-1);y\_bound(1)],'-k','linewidth',2)

plot([x\_bound(ind+1:end);x\_bound(ind+1)],[y\_bound(ind+1:end);y\_bound(ind+1)],'-k','linewidth',2)

else

plot([x\_bound;x\_bound(1)],[y\_bound;y\_bound(1)],'-k','linewidth',2)

end

hold off

axis([xmin,xmax,ymin,ymax])

title('Taps and Bounds')

% If Delaunay Triangulation is not empty

if isempty(DT.ConnectivityList)~=1

% Figure 2: Delaunay Triangulation

figure

hold on

for ii=1:size(DT.ConnectivityList,1)

patch(DT.Points(DT.ConnectivityList(ii,:),1),DT.Points(DT.ConnectivityList(ii,:),2),[0 0.1 1],'EdgeColor',[1 0 0],'linewidth',2,'FaceAlpha',0.05)

end

% Plot taps

plot(P(P\_inl,1),P(P\_inl,2),'ok')

plot(P(~P\_inl,1),P(~P\_inl,2),'xk')

% Plot surface bounds

if any(isnan(x\_bound))

ind=find(isnan(x\_bound));

plot([x\_bound(1:ind-1);x\_bound(1)],[y\_bound(1:ind-1);y\_bound(1)],'-k','linewidth',2)

plot([x\_bound(ind+1:end);x\_bound(ind+1)],[y\_bound(ind+1:end);y\_bound(ind+1)],'-k','linewidth',2)

else

plot([x\_bound;x\_bound(1)],[y\_bound;y\_bound(1)],'-k','linewidth',2)

end

hold off

axis([xmin,xmax,ymin,ymax])

title('Delaunay Triangulation')

% Figure 3: Perpendicular Bisectors

figure

hold on

for ii=1:size(DT.ConnectivityList,1)

patch(DT.Points(DT.ConnectivityList(ii,:),1),DT.Points(DT.ConnectivityList(ii,:),2),[0 0.1 1],'EdgeColor',[1 0 0],'linewidth',2,'FaceAlpha',0.05)

end

% Plot taps

plot(P(P\_inl,1),P(P\_inl,2),'ok')

plot(P(~P\_inl,1),P(~P\_inl,2),'xk')

% Plot surface bounds

if any(isnan(x\_bound))

ind=find(isnan(x\_bound));

plot([x\_bound(1:ind-1);x\_bound(1)],[y\_bound(1:ind-1);y\_bound(1)],'-k','linewidth',2)

plot([x\_bound(ind+1:end);x\_bound(ind+1)],[y\_bound(ind+1:end);y\_bound(ind+1)],'-k','linewidth',2)

else

plot([x\_bound;x\_bound(1)],[y\_bound;y\_bound(1)],'-k','linewidth',2)

end

for ii=1:size(vx,1)

plot(vx(ii,:),vy(ii,:),'-b','linewidth',2)

end

for ii=1:size(kx,1)

plot(kx(ii,:),ky(ii,:),'-b','linewidth',2)

end

hold off

axis([xmin,xmax,ymin,ymax])

title('Perpendicular Bisectors')

end

% Figure 4: Voronoi Regions

N\_taps=size(P,1);

c\_map=jet(N\_taps);

figure

for ii=1:N\_taps

% Plot tributary areas with holes

if any(isnan(x\_trib{ii}))&&P\_inl(ii)

[patch\_f,patch\_v]=poly2fv(x\_trib{ii},y\_trib{ii});

patch('Faces',patch\_f,'Vertices',patch\_v,'FaceColor',c\_map(ii,:),'LineStyle','none','FaceAlpha',0.5);

ind=find(isnan(x\_trib{ii}));

patch(x\_trib{ii}(1:ind-1),y\_trib{ii}(1:ind-1),c\_map(ii,:),'FaceColor','none','linewidth',2);

patch(x\_trib{ii}(ind+1:end),y\_trib{ii}(ind+1:end),c\_map(ii,:),'FaceColor','none','linewidth',2);

% Plot tributary areas without holes

elseif P\_inl(ii)

patch(x\_trib{ii},y\_trib{ii},c\_map(ii,:),'linewidth',2,'FaceAlpha',0.5);

end

end

hold on

% Plot taps

plot(P(P\_inl,1),P(P\_inl,2),'ok')

plot(P(~P\_inl,1),P(~P\_inl,2),'xk')

% Plot surface bounds

if any(isnan(x\_bound))

ind=find(isnan(x\_bound));

plot([x\_bound(1:ind-1);x\_bound(1)],[y\_bound(1:ind-1);y\_bound(1)],'-k','linewidth',2)

plot([x\_bound(ind+1:end);x\_bound(ind+1)],[y\_bound(ind+1:end);y\_bound(ind+1)],'-k','linewidth',2)

else

plot([x\_bound;x\_bound(1)],[y\_bound;y\_bound(1)],'-k','linewidth',2)

end

hold off

axis([xmin,xmax,ymin,ymax])

title('Voronoi Regions')

end

warning('on','MATLAB:triangulation:EmptyTri2DWarnId');

end