POLITECNICO DI TORINO

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Aerospaziale

Barbera’s TRiangler 3.0

BTR User’s Manual

Andrea Barbera,
Dr. in Aerospace Engineering

ab_borgorevel@yahoo.it

ABSTRACT

BTR 3.0 is a library created for Matlab 6.5 that can triangulate a plain surface that may contain every kind of bond on the domain as holes, concavities or inner segments.

To insert the domain it is necessary inserting some vertexes that bound it and specifying in which relation are them. Then BTR 3.0 calculates which is the best mesh that fits the domain.

Parameters of the triangulation are the maximum area of every triangle, the smallest angle among the ones added by BTR 3.0 and the regions in which these parameters are active.

Output of this function is the set of data necessary to apply a finite elements method to the triangulation. It includes boundary conditions and elements lists.

INPUTS

BTR has 3 input variables: `Domain`, `BC` and `RefiningOptions`. Among these `Domain` is the main input, but also other two inputs must be considered. We remember that Matlab is case-sensitive, so problems may arise if users don’t pay attention to caps characters.

ELIGIBLE DOMAINS

Every domain or bond must be delimitated by segments, ad descrived below.

“DOMAIN” VARIABLE

Domain is a structure that is composed by different fields. In particular there are:

```
Domain.InputVertex
Domain.Boundary
Domain.Holes
Domain.Segments
```

Every field written above must be included in `Domain` for the right running of BTR.

DOMAIN.INPUTVERTEX

`Domain.InputVertex` is a matrix containing the coordinates of every vertex useful to describe the domain. The i-esim row represents the i-esim input vertex. The first component of i-esim row represents the x-coordinate of i-esim vertex, the second component represent the y-coordinate.

An example is:

```
Domain.InputVertex = [0 0
   5 0
   5 5
   0 5];
```

In this example the first vertex is (0,0), the second is (5,0) the third is (5,5) and the fourth is (0,5)

The order used to memorise vertexes is irrelevant.

DOMAIN.BOUNDARY

`Domain.Boundary` is a structure containing “`Values`” as a field. `Domain.Boundary.Values` is an array containing the references of the boundary of the domain. Every number of this array is a reference to a row of `Domain.InputVertex`.
`Domain.Boundary.Values` is a sorted list that says that there is a domain border between two adjacent elements of `Domain.Boundary.Values` and between its first and last element.

Referring to `Domain.InputVertex` written before a squared domain may be described in different ways. For example, considering domain in figure 1

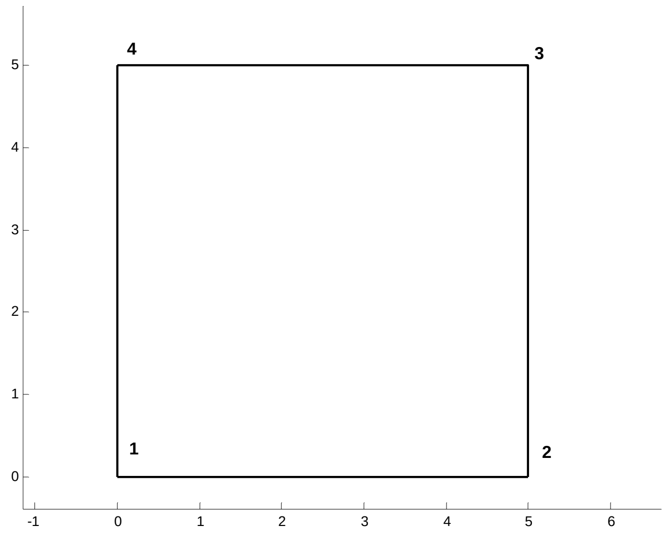


figure 1

We may impose:

```
Domain.Boundary.Values = [1 2 3 4];
```

or

```
Domain.Boundary.Values = [3 4 1 2];
```

but not

```
Domain.Boundary.Values = [1 3 4 2];
```

because there are no borders between Vertexes 1 and 3 or between Vertexes 2 and 4.

This sorted list must be "open" and not "closed": BTR understands that a border must be inserted between first and last element of `Domain.Boundary.Values`, and forcing last domain border would give an error.

DOMAIN.HOLES

`Domain.Holes` is a structure containing "`Hole`" as a field. `Domain.Holes` contains the references of every hole in the domain. Field `Hole` is an array whose dimensions are equal to the number of holes presents in the domain. Every field Hole has values as sub-field. This is an array that contains references to some vertexes inserted in `Domain.InputVertex`.
`Domain.Holes.Hole(i).Values` is a sorted list that says that there is a border of the i-esim hole between two adjacent elements of `Domain.Holes.Hole(i).Values` and between its first and last element.

For example we may insert the same square described before with two triangular holes in the domain. We compose `Domain.InputVertex` with every vertex useful for the domain in a random order, but the shape of the domain and the holes is accurately described in other fields of `Domain`:

```
Domain.InputVertex = [ 0 0
  5 0
  5 5
  0 5
  1 1
  1 2
  2 1
  3 4
```

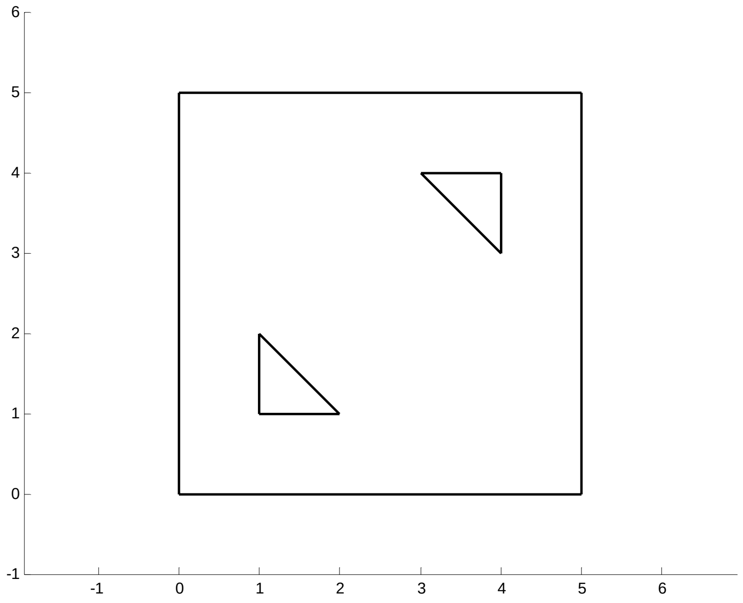
```
        4  3
        4  4];

Domain.Boundary.Values = [1  2  3  4];

Domain.Holes.Hole(1).Values = [5  6  7];

Domain.Holes.Hole(1).Values = [8  9  10];
```

This brings to a domain like the one described in figure 2:



As for the ones described in `Domain.Boundary`, **these sorted lists are "open" and not "closed".**

If in the domain there is no hole, `Domain.Holes` must however assume following value: `Domain.Holes.Hole = [ ];`

DOMAIN.SEGMENTS

`Domain.Segments` is a structure containing "`Segment`" as a field, and its usage is very similar to the one of `Domain.Holes`. `Domain.Segments` contains the references of segments that may be forced in the domain. Field `Segment` is an array whose dimensions are equal to the number of holes presents in the domain. Every field `Hole` has "`values`" as sub-field. This is an array that contains references to some vertexes inserted in `Domain.InputVertex`.
`Domain.Segments.Segment(i).Values` is a sorted list that says that there is a border of i-esim segment between two adjacent elements of `Domain.Segments.Segment(i).Values`.

`Domain.Segments` may contain references to vertexes that are inserted in the domain only for this purpose or may use vertexes that have been inserted to shape domain or holes.

As for the ones described in `Domain.Holes`, **if in the domain there is no segment, `Domain.Segments` must however assume following value: `Domain.Segments.Segment = [ ];`**

Sample of usage of segments are reported in appendixes.

TRICK: FORCE VERTEXES IN GRID

Every vertex of `InputVertex` will however be inserted in the mesh, even if it will not be used to shape the domain, a hole or a segment. In the same way a single vertex may be used to shape different elements, as for example the domain and a segment, or two different segments.

TROUBLESHOOTING

→ Inserting twice the same vertex will give an error.

→ Elements of triangulations must be coherent and may not intersect each other, but may however touch themselves in a single point (so if a "cross" composed by segments must be inserted in the domain also the intersection point must be inserted in `InputVertex`).

"BC" VARIABLE

BC is a structure that is composed by different fields. In particular there are:

```
BC.Values
BC.InputVertexValues
BC.Boundary
BC.Holes
BC.Segments
```

Every field written above must be included in `BC` for the right running of BTR

BC.VALUES

`BC.Values` is list of BC values that will be set in domain borders. Conventionally:

Values in even position (i.e. 1st, 3rd...) are Dirichlet conditions
Values in odd position (i.e. 2nd, 4th...) are Neumann conditions

If it is used a different number of Dirichlet and Neumann condition some -1s may be for example put to fill BC.Values.

An example of `BC.Values` usage is following one:

```
BC.Value = [ 3e-1 2e4 0 3e4 -1 0 -1 4e4];
```

In this case are used two Dirichlet condition (3e-1 with index 1 and 0 with index 3) and four Neumann conditions (2e4 with index 2, 3e4 with index 4, 0 with index 6 and 4e4 with index 8).

BC.INPUTVERTEXVALUES

`BC.InputVertexValues` is list of BC values that are set in `InputVertex`. These conditions must be Dirichlet ones (because it has no sense imposing a Neumann condition in a vertex). Conventionally instead of the values references to `BC.Values` are used. If a vertex is "free" and there is no BC applied on it a conventional 0 is put in correspondence of it.

An example of `BC.InputVertexValues` usage is following one, with reference to previous `BC.Values`:

```
Domain.InputVertex = [0 0
  5 0
  5 5
  0 5];

BC. InputVertexValues = [0 0 1 3];
```

This means that on vertexes (0,0) and (5,0) there is no BC setted, (5,5) has a Dirichled BC = 3e-1 and (0,5) has a Dirichled BC = 0.

BC.Boundary

`BC.Boundary` is a structure containing "`Values`" as a field. `BC.Boundary.Values` is an array containing the references of the BCs presents on domain boundary borders. Every number of this array is a reference to an element of `BC.InputVertexValues` .
`BC.Boundary.Values` is a sorted list that identifies which BC applies to a particular domain border. In particular if we know that there is a border between `Domain.Boundary.Values(i)` and `Domain.Boundary.Values(i+1)` the value to set on that border is given as a reference by `BC.Boundary.Values(i)`. Of course border identified by `Domain.Boundary.Values(n)` and `Domain.Boundary.Values(1)` (where n is number or vertexes that shape the domain) has a BC that is given by reference by `BC.Boundary.Values(n)`.

Referring to domain described in figure 1, with

```
Domain.InputVertex = [ 0 0
  5 0
  5 5
  0 5];
```

and

```
Domain.Boundary.Values = [1 2 3 4];

BC.Boundary.Values = [3 3 2 4];

BC.Values = [ 3e-1 2e4 0 3e4 -1 0 -1 4e4];
```

the lower and the right borders have a Dirichlet BC equal to 0, the upper border has a Neumann BC equal to 2e4 and the left border has a Neumann BC equal to 0.

BC.Holes and BG.Segments

Usage of `BC.Holes` and `BC.Segments` is similar to usage of `BC.Boundary`, with the difference that, like for `Domain` structure, these structures admit fields called "`Hole`" or "`Segment`" that are array in a number coherent with ones in `Domain` structure. Sample of usage of these structures are reported in appendixes.

If in the domain there are no holes or segments, `Domain.Holes` or `Domain.Segments` must however assume following values: `Domain.Holes.Hole = [ ]; or Domain.Segments.Segment = [ ];`

BC must be created even in not interested in its effect of triangulation.

Trick: not intrested in BC

If not interested in BC, it will be sufficient impose `BC.Values = 0;` and create arrays of ones as sub-fields of `BC` structure of lengths coherent with ones of `Domain` structure. This will not slow the generation of the triangulation

"REFININGOPTIONS" VARIABLE

BTR 3.0 creates a Delaunay triangulation that fits the domain and then refines the mesh according to parameter chosen by users, inserting new vertexes in best positions. To set these parameters users must consider `RefiningOptions` structure, whose fields are:

```
RefiningOptions.CheckArea
RefiningOptions.AreaValue
RefiningOptions.CheckAngle
RefiningOptions.AngleValue
RefiningOptions.Subregions
```

Every field written above must be included in `RefiningOptions` for the right running of BTR

REFININGOPTIONS.CHECKAREA AND REFININGOPTIONS.AREAVALUE

One of the possible grid refining continues splitting triangles until the larger triangle has an area value lesser than a threshold value.
If `RefiningOptions.CheckArea` (a character variable) assumes values `'y'` or `'Y'` BTR controls the size of every triangle, imposing a threshold value equal to `RefiningOptions.AreaValue`. If not interested in size of triangles it will be sufficient to impose `RefiningOptions.CheckArea` equal to `'n'` or `'N'`.

REFININGOPTIONS.CHECKANGLE AND REFININGOPTIONS.ANGLEVALUE

The other possible grid refining continues splitting triangles until every newly inserted angle in the domain is greater than a threshold value. The value of smallest newly inserted angle is also called quality of the grid.

If `RefiningOptions.CheckAngle` (a character variable) assumes values `'y'` or `'Y'` BTR controls the angles of every triangle, imposing a threshold value equal to `RefiningOptions.AngleValue`. If not interested in quality of triangles it will be sufficient to impose `RefiningOptions.CheckAngle` equal to `'n'` or `'N'`. `RefiningOptions.AngleValue` is inserted in degrees.

The algorithm used in the triangulation may converge only if `RefiningOptions.AngleValue` is lesser or equal than 30°.

REFININGOPTIONS.SUBREGIONS

It is possible refining not the whole domain but only a subset of it. Using `RefiningOptions.Subregions` it is possible to control any number of these subregions.

`RefiningOptions.Subregions` is a structure that has "`Subregion`" as only field. This is an array whose length is equal to the number of refining subregions.
`RefiningOptions.Subregions.Subregion(i)` has some fields similar to ones that `RefiningOptions` has. In particular i-esim subregion has these fields:

```
RefiningOptions.Subregions.Subregion(i).CheckArea
RefiningOptions.Subregions.Subregion(i).AreaValue
RefiningOptions.Subregions.Subregion(i).CheckAngle
RefiningOptions.Subregions.Subregion(i).AngleValue
RefiningOptions.Subregions.Subregion(i).Type
RefiningOptions.Subregions.Subregion(i).Descriptors
```

`CheckAngle`, `CheckArea`, `AngleValue`, `AreaValue` have the same usage described before, with the difference that the refining is applied only to the i-esim subregion.

The region is identified by fields "`Type`" and "`Descriptors`". Type is a generic function in subdirectory "/methods" that gives back a boolean value that becomes true if the point belongs to the subregion. Usually `Type` gives only the shape of the subregion, and values are contained in `Descriptors`, but this is not a rule. Users may create their own methods to refine particular subregions.

BTR 3.0 contains two methods, `‘elliptic’` and `‘rectangular’`. Description of these methods may be found in the test of the function.

This is `‘elliptic’`, a sample of method.

```
function [Flag] = elliptic (Descriptors, P)

%
% Descriptors: it specify the limits of the ractangle and it's in the form:
%    Descriptors = [xC, yC, a, b]
%       c = [xC yC] is the center of the ellipse
%       a is the semiaxis in x-direction
%       b is the semiaxis in y-direction
%
%    ( The ellipse equation used is (xP-xC)^2/a^2 + (yP-yC)^2/b^2 = 1 )
%
% P: the generic point (P = [xP,yP]) that must control if belongs or not no
% the region
%
% Flag: the boolean that is true if P belongs to the region

if ( (P(1) - Descriptors(1)) / Descriptors(3) )^2 + ( (P(2) - Descriptors(2)) ...
         / Descriptors(4) )^2 <= 1
    Flag = true;
else
    Flag = false;
end

return
```

Usages of `RefiningOptions.Subregions` are reported in appendixes.

If refining of subregions isn’t required, `RefiningOptions.Subregions` must however assume following value: `RefiningOptions.Subregions = [ ];`

TROUBLESHOOTING

→ Refining a subregion may ruin result of a previous refining: this may be for example the case of an area refining that follow a quality refining.

→ The point that determines if a triangle belongs or not to a subregion is the centre of gravity of the triangle. If this point lies outside the region also the triangle does. To improve this research may be useful consider refining regions a little larger than they would otherwise be.

OUTPUT

BTR has a single output variable, geom. This variable is a structure that keeps information of every feature of the grid.

GEOM.ELEMENTS.COORDINATES

`geom.elements.coordinates` is a (nV x 2) matrix that contains the list of coordinates of every vertex, where nV is the number of vertexes present in the mesh.
So the i-esim row of the matrix contains [xV yV], the x-coordinate and the y-coordinate of i-esim vertex.

Input vertexes may not coincide with first vertexes of `geom.elements.coordinates`, but they will surely be memorized in random positions.

GEOM.ELEMENTS.TRIANGLES

`geom.elements.triangles` is a (nT x 3) matrix that contains the link between every triangle and vertexes that bound it. nT is the number of triangles present in the mesh.
So the i-esim row of the matrix contains [V1 V2 V3], the three vertexes that belong to i-esim triangle.

Vertexes are memorized in anticlockwise order.

GEOM.ELEMENTS.BORDERS

`geom.elements.borders` is a (nB x 4) matrix that contains the link between every border and vertexes that bound it and triangles that surround it. nB is the number of borders present in the mesh.
So the i-esim row of the matrix contains [V1 V2 T1 T2]: V1 and V2 are the two vertexes that belong to i-esim border, T1 and T2 are the two triangles that surround the i-esim border

If a border is adjacent to a region that mustn’t be meshed the reference in `geom.elements.borders` is conventionally -1.

GEOM.ELEMENTS.NEIGHBOURHOOD

`geom.elements.neighbourhood` is a (nT x 9) matrix that contains the link between every triangle and three adjacent triangles and borders. nT is the number of triangles present in the mesh.
So the i-esim row of the matrix contains [T1 T2 T3 B1 B2 B3 Ref1 Ref2 Ref3]: T1, T2 and T3 are the three triangles that are adjacent to the i-esim triangle, B1 is the border between i-esim triangle and T1, B2 is the border between i-esim triangle and T2, B3 is the border between i-esim triangle and T3. Ref may generically assume values from 1 to 3: if j is a triangle adjacent to i the j-esim row of TT will contain the reference to i-esim triangle in one of the first three columns. The number of column (first, second or third) is Ref for the i-esim row of TT, six columns after the one in which there is there reference to j-esim triangle.

If a border is adjacent to a region that mustn’t be meshed the reference in `geom.elements.triangles` is conventionally -1. In this case also related reference assumes value -1.

GEOM.ELEMENTS.VERTEXESNEIGHBOURHOOD

`geom.elements.vertexesneighbourhood` is an array whose length is nV, the number of vertexes in the triangulation. This array contains the link of a vertex and adjacent borders and vertexes, and these informations are stored in fields called ‘`B`’ (borders) and ‘`V`’ verteces.
So the i-esim element of the array is composed by: `geom.elements.vertecesneighbourhood.B` = [B1,B2,...,BN] if the i-esim vertex is adjacent to N borders and `geom.elements.vertecesneighbourhood.V` = [V1,V2,...,VN] if the i-esim vertex is adjacent to N vertexes.

The j-esim vertex of `geom.elements.vertecesneighbourhood.V` belongs always to the j-esim border of `geom.elements.vertecesneighbourhood.B`.

GEOM.NELEMENTS.NTRIANGLES

`geom.nelements.nTriangles` contains the number of triangles in the grid.

GEOM.NELEMENTS.NBORDERS

`geom.nelements.nBorders` contains the number of borders in the grid.

GEOM.NELEMENTS.NVERTECES

`geom.nelements.nVerteces` contains the number of verteces in the grid.

GEOM.PIVOT.NODELIST

`geom.pivot.nodelist` is an array whose length is nV, the number of vertexes in the triangulation. This array contains informations about BC on every vertex: if the i-esim component of `geom.pivot.nodelist` is equal to zero no BC is assigned for the i-esim vertex, otherwise the i-esim component assumes a value that is a reference explained in `BC.Values`.

GEOM.PIVOT.DI

`geom.pivot.Di` is a (nDi x 2) matrix that contains the list of coordinates of every vertex in which a Dirichlet boundary condition has been assigned, and nDi is the number of these vertexes present in the mesh.
So the i-esim row of the matrix contains [iDi BCvalue], where iDi is a vertex and BCvalue is a reference to a Dirichlet BC, as explained in `BC.Values`.

GEOM.PIVOT.NE

`geom.pivot.Ne` is a (nNe x 2) matrix that contains the list of coordinates of every border in which a Neumann boundary condition has been assigned, and nNe is the number of these borders present in the mesh.
So the i-esim row of the matrix contains [iNe BCvalue], where iNe is a vertex and BCvalue is a reference to a Neumann BC, as explained in `BC.Values`.

GEOM.SUPPORT.TINFO

`geom.support.TInfo` is an array whose length is nT, the number of triangles in the triangulation. This array contains informations about every triangle. The i-esim element of `geom.support.TInfo` is composed by:

`geom.support.TInfo(i).Area` = the area of i-esim triangle.

`geom.support.TInfo(i).B` = a measure of quality of i-esim triangle. Calling α (in rad) the smallest angle of the i-esim triangle the relation to find B is $B = \frac{1}{2 \cdot \sin(\alpha)}$.

`geom.support.TInfo(i).Circumcenter` = an array composed by [xC yC] that contains the coordinates of the circumcenter of i-esim triangle.

`geom.support.TInfo(i).Circumradius` = the squared of the circumradius of the i-esim triangle.

GEOM.SUPPORT.BINFO

`geom.support.BInfo` is a (nB x 3) matrix that contains information about every border. nB is the number of borders present in the mesh.
So the i-esim row of the matrix contains [Info1 Info2 Info2]: Info1 says if the i-esim border is a domain border (value = 1), an hole border (value = 2), a segment border (value = 3) or an inner border (value = 0). Info2 says if the i-esim border is a limit of the triangulation (value = 1) or not (value = 0). Info3 says if the i-esim border has some BC assigned (value = reference in `BC.Values`) or not (value = 0).

GEOM.SUPPORT.BCIRCLE

`geom.support.BCircle` is an array whose length is nB, the number of borders in the triangulation. This array contains informations about the bordercircle of every border. The bordercircle is the minimum circle that may contain the border (that thereafter is a diameter of the bordercircle). The i-esim element of `geom.support.BCircle` is composed by:

`geom.support.BCircle(i).Center` = an array composed by [xC yC] that contains the coordinates of the bordercircle of i-esim border.
`geom.support.TInfo(i).r2` = the squared of the circumradius of the i-esim triangle.

GEOM.SUPPORT.EB

`geom.support.EB` contains a structure necessary to refine the grid.

GEOM.INPUT.DOMAIN

`geom.input.Domain` contains the `Domain` input structure that has been used to create the mesh.

GEOM.INPUT.BC

`geom.input.BC` contains the `BC` input structure that has been used to create the mesh.

PARAMETERS

There are many parameters that are used by BTR to generate the mesh. Every parameter is stored in function `global_parameter` and may be modified by users. This may cause or solve some problems, according to the parameter.

PARAMETERS THAT WILL MOST PROBABLY BE MODIFIED

TOLERANCE

`tolerance` (default value `1e-12`) is the value used like a tolerance (two numbers are equal if their difference is lesser than tolerance).

SAFEMESHING

If `SafeMeshing` (default value `true`) is activated the program becomes slower but can triangulate also domains that otherwise would crash BTR (`SafeMeshing` must be activated if there are boundary borders very close to other ones).

RECURSIVEENCROACHINGLIMIT

`RecursiveEncroachingLimit` (default value `4`) is the limit after which BTR tries to stop recursive borders disecroaching. The more this value is high, the safer and slower BTR get in presence of small input angles

REFININGFOLDERNAME

`RefiningFolderName` (default value `'methods'`) is the name of the folder that contains refining methods.

PARAMETERS THAT WILL LESS PROBABLY BE MODIFIED

SPLITTINGBORDERSLIMITS

SplittingBordersLimits (default value `10`) sets the maximum number of borders that may be contained in single Encroachable subregion.

XBOXENLARGE, YBOXENLARGE

`XBoxEnlarge` (default value `0.5`) and `YBoxEnlarge` (default value `0.5`) are parameters used to create the containing box, the rectangle that must contain every input vertex. They determines the stretching of the containing box with reference to minimum rectangle containing every input vertex. These values must be positive.

GENERATIONUNTILRESEARCH

`GenerationUntilResearch` (default value `10`) says how the research of a particular triangle is made from a starting triangle.

FIRSTEXTIMATETMULTIPLIER, FIRSTEXTIMATEBMULTIPLIER, FIRSTEXTIMATEVMULTIPLIER

`FirstExtimateTmultiplier` (default value `8`), `FirstExtimateBmultiplier` (default value `8`) and `FirstExtimateVmultiplier` (default value `6`) are multipliers used to make a first estimate of the number of elements of the grid.

FIRSTMAXREFININGROOF

`FirstMaxRefiningRoof` (default value `1/5`) is a multiplier used to give a first estimate of the "refining roof", the maximum number of triangles that BRT tries to refine at every step.

INCREASINGREFININGROOF

`IncreasingRefiningRoof` (default value `3`) is a multiplier used to increase the refining roof after that it has been reached in a previous step.

RECURSIONSTACKLIMIT

`RecursionStackLimit` (default value `1000`) is the maximum recursion stack limit that is set in Matlab

APPENDIX: SAMPLES

SAMPLE 1

This is a sample of triangulation of a complex domain, with two holes, different segments and different BC assigned to different elements. This is the script necessary to generate the mesh:

```
Domain.InputVertex = [ 0 0
    5 0
    5 5
    0 5
    1 1
    4 1.25
    1 1.5
    4 4
    4 3.5
    1 3.75];
Domain.Boundary.Values = 1:4 ;
Domain.Holes.Hole(1).Values = 5:7;
Domain.Holes.Hole(2).Values = 8:10;
Domain.Segments.Segment(1).Values = [4 10];
Domain.Segments.Segment(2).Values = [10 6];
Domain.Segments.Segment(3).Values = [6 2];

BC.Values = [2 4 6 8 10 12];
BC.Boundary.Values = [1 2 1 2];
BC.Holes.Hole(1).Values = [3 3 3];
BC.Holes.Hole(2).Values = [4 4 4];
BC.Segments.Segment(1).Values = [5];
BC.Segments.Segment(2).Values = [6];
BC.Segments.Segment(3).Values = [7];
BC.InputVertexValues = [1 1 0 0 3 3 3 0 0 0];


RefiningOptions.CheckArea = 'Y';
RefiningOptions.CheckAngle = 'N';
RefiningOptions.AreaValue = 0.1;
RefiningOptions.AngleValue = [ ];
RefiningOptions.Subregions = [ ];


[geom] = btr30(Domain,BC,RefiningOptions);
draw_grid (geom,1);
```

The result is reported in figure 3. Red borders are borders to which are assigned Dirichled conditions, blue borders are borders to which are assigned Neumann conditions. Vertexes that are circled in red are Dirichlet vertexes.

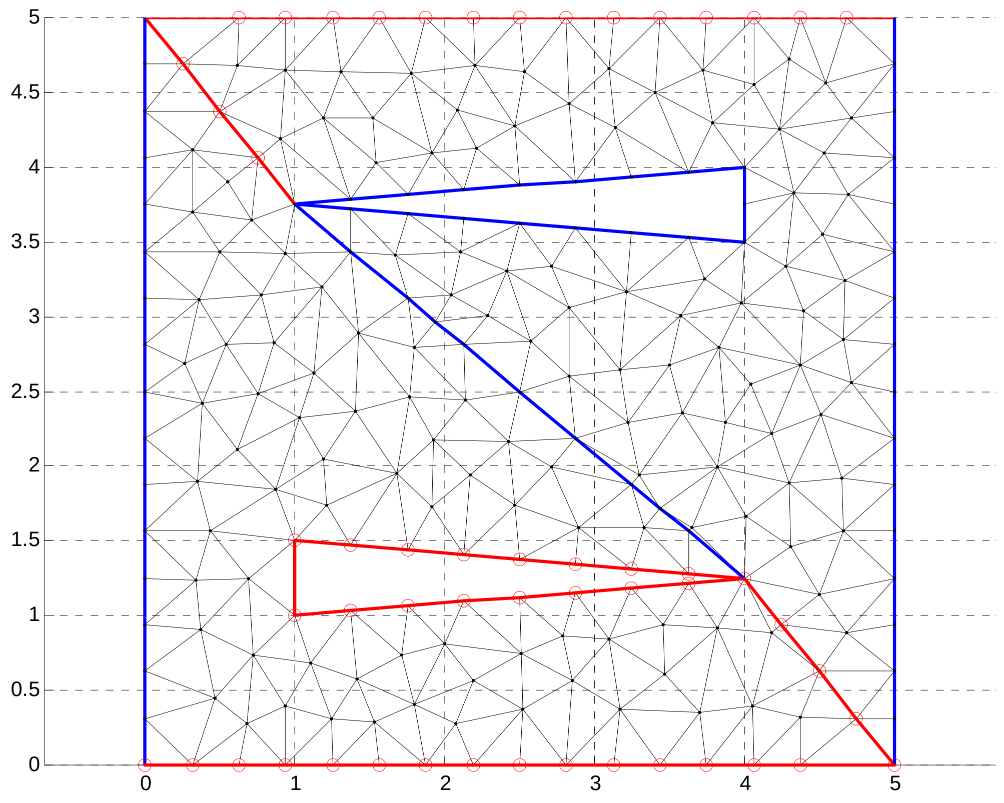


figure 3

SAMPLE 2

This is a sample of triangulation of a simplier domain, with many input vertexes on a single border and fewer in other borders. Two refining subregions are chosen, an elliptic one (in the center) and a rectangular one (on the left).

```
for i = 0: 99
    Domain.InputVertex(i+1,:) = [i/10 0];
end
Domain.InputVertex(101:106,:) = [ 10 0
    10 10
    6.2 10
    6 10
    5.8 10
    0 10];
Domain.Boundary.Values = 1:106;
Domain.Holes.Hole = [ ];
Domain.Segments.Segment = [ ];

BC.Values = 1;
BC.Boundary.Values = ones(1,106);
BC.Holes.Hole = [ ];
BC.Segments.Segment = [ ];
BC.InputVertexValues = ones(1,106);

RefiningOptions.CheckArea = 'N';
RefiningOptions.CheckAngle = 'Y';
RefiningOptions.AreaValue = 0.1;
RefiningOptions.AngleValue = 5;

RefiningOptions.Subregions.Subregion(1).Type = 'rectangular';
RefiningOptions.Subregions.Subregion(1).Descriptors = [2 0 10 0];
RefiningOptions.Subregions.Subregion(1).CheckArea = 'N';
RefiningOptions.Subregions.Subregion(1).CheckAngle = 'Y';
RefiningOptions.Subregions.Subregion(1).AreaValue = [ ];
RefiningOptions.Subregions.Subregion(1).AngleValue = 30;


RefiningOptions.Subregions.Subregion(2).Type = 'elliptic';
RefiningOptions.Subregions.Subregion(2).Descriptors = [6 6 3 2];
RefiningOptions.Subregions.Subregion(2).CheckArea = 'Y';
RefiningOptions.Subregions.Subregion(2).CheckAngle = 'N';
RefiningOptions.Subregions.Subregion(2).AreaValue = 0.01;
RefiningOptions.Subregions.Subregion(2).AngleValue = 30;


[geom] = btr30(Domain,BC,RefiningOptions);
draw_grid (geom,1);
```

and the result is reported in figure 4.

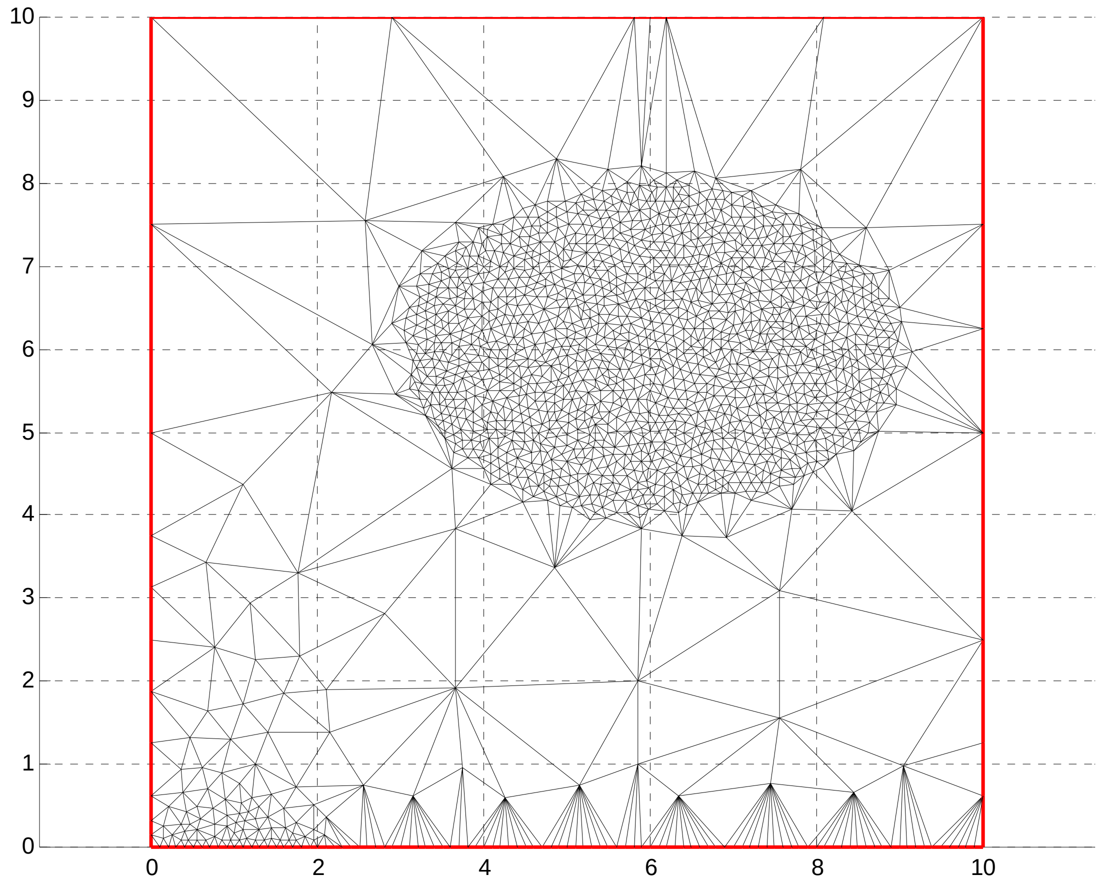


figure 4