2D Finite Elements

Version 2017.1

Steps of Finite Elements analysis

- We can organize the analysis of a generic problem with n-dimension Finite Elements (F.E.) into the following steps:
  1. Definition of the (differential) equations describing the problem
  2. Proper choice of the computational domain and its physical properties
  3. Correct assignment of the B.C.s
  4. Mesh generation
  5. Stiffness matrix, mass matrix and constant terms vector construction
  6. Solution of the obtained linear system
  7. Solution post-processing (plot, derivation, max/min, interpolation,…)

1. Definition of the differential equations describing the problem

- During this part of the course we focus on the (temporal evolution of the) 2D diffusion-transport-reaction problem:

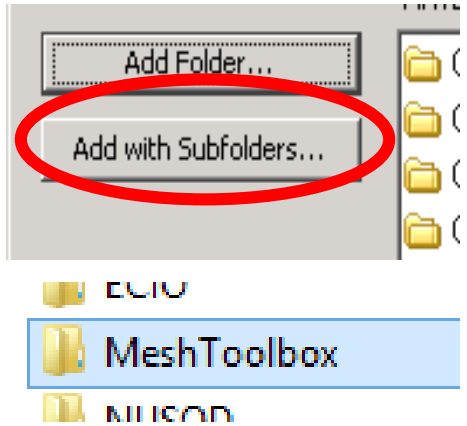
$$\rho \frac{\partial^n u(x,y;t)}{\partial t^n} + \nabla \cdot (\mu \nabla u(x,y;t)) - \underline{\beta} \cdot \nabla u(x,y;t) + \sigma u(x,y;t) = f(x,y;t)$$

- This equation is used to model many physical phenomena, and plays an important role in different fields:
  - Mechanics
  - Thermodynamics
  - Electronics
  - …

2-6. Mesh toolbox

- For the points 2 to 6 we use a collection of MATLAB functions (“toolbox”) available on the Portale della Didattica. This toolboxprovides a set of low level routine (respect to the ones available with Comsol MultiPhysics, Hypermesh, PDEtool, etc.) to access all the mesh data (vertices, triangles, edges, …) and build the linear system obtained with the F.E. approximation of the given problem.
- Since the toolbox contains many folders, to be able to use it regardless of the current MATLAB folder, we need to add it to the MATLAB path, a list of folders that MATLAB scans when searching for a requested function.
- For additional details, see “path” and “Function Precedence Order” in the MATLAB help

How to install the mesh toolbox



- To add the toolbox's folders to the Search Path:
  - **Extract** the files and folders from the archive MeshToolbox.zip to a folder (ex. MeshToolbox)
  - In MATLAB, choose **File→Set path** from the menu (version<2012b) or **Set Path** in the **Home** Tab
  - In the Set Path dialog, press **Add with subfolders…** and select the root extracted folder
    - NB: select the extacted folder, not its subfolders, since they are automatically added using the "Add with subfolders..." option
  - In the LAIB, simply close the dialog without saving using the **Close** button; on your computer choose **Save** to save the changes (you might need to be an administrator to perform this operation)
- Alternatively, once the files have been extracted, change the current folder to MeshToolbox and execute the script `setpath.m`

MATLAB structures

- In MATLAB it is possible to logically regroup heterogeneous data types using a single structure variable

```
>> Student.Name='Donald';
>> Student.ID=313313;
```

- `Student` is variable of type *structure*, `Name` and `ID` are *fields* of the structure
- We can of course create vectors and matrices of structures:

```
>> Student(100).Name = 'Mickey';
```

- You can also create a structure using the command `struct` and specifying couples of name-values:

```
>> a=struct('Name','Pap','ID',313);
>> isfield(a,'Name') %returns true
```

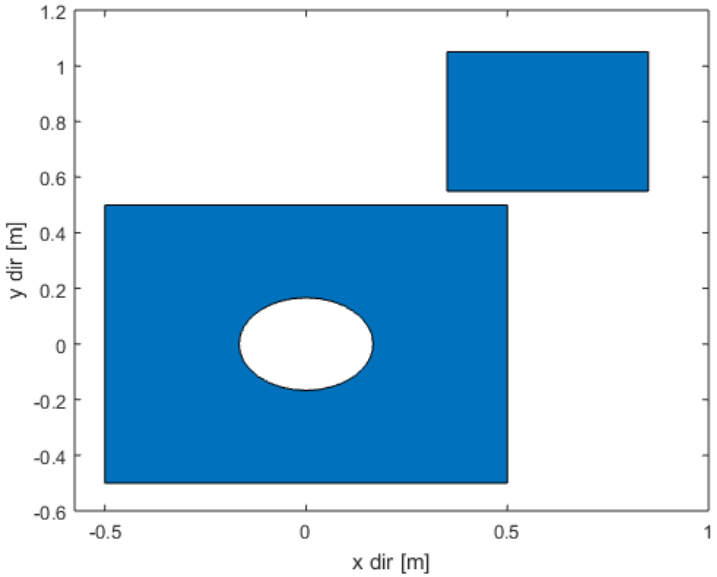
MATLAB Classes (Advanced topic)

- A MATLAB class is similar to a struct but also allows to
  - Define properties, and functions (“methods”) acting only on *instances* of that class (i.e., a variable of such a class)
  - Define a “constructor”, which is a function called automatically when you create a variable of that ; this is useful, e.g., to automatically initialize field, set counters to zero, etc)
  - Re-define fundamental operators (+, -, *, /, &,…) to allow a more intuitive execution of some commands
- All the properties and methods of a class are collected in a file, having the same name as the class, starting with the keyword `classdef`
- The function `isa(variable,'classname')` returns true when the variable belongs to the class “classname”
- In MATLAB all the variables are indeed instances of a class (class `int`, class `float`, class `double`…)

```
>>a=10.5; isa(a,'double') %returns 1
```

On regions, borders, and edges

- We call *region* a collection of closed regions, eventually with holes. Each region has one or more *borders*
- Each border is formed by a closed sequence of points in the 2D cartesian plane linked together by lines. These connecting lines are called *edges*.
- The code for the definition of regions is included in `region.m`.
- It allows to
  - Create a region from (x,y) coordinates
  - Translate, rotate and expand a region
  - Calculate the intersections between different regions



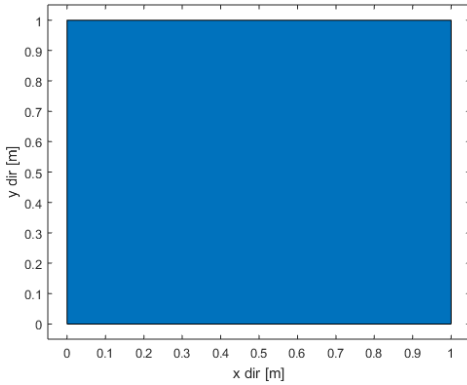
A *region* with 2 *regions* + 1 *hole* = *3 borders, having* 40 edges
(the circular hole is approximated with 32 edges)

`region` class: construction

- To define an object of class `region`, simply call `region(xx,yy)`, where `xx` and `yy` are vectors with the coordinates of the nodes:

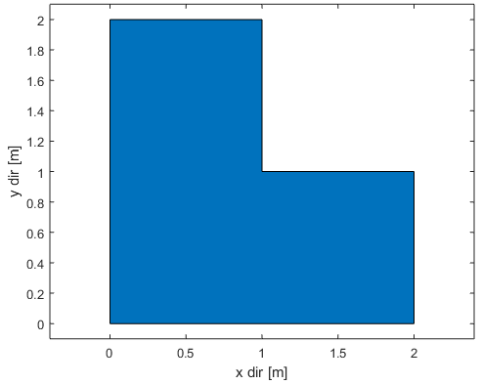
```
>> region([0,0,1,1],[0,1,1,0]);
```

returns a squared region




```
>> R=region([0,2,2,1,1,0],[0,0,1,1,2,2]);
```

returns a L regiond domain



- To build complex regions it is generally easier to start from the basic regions available in the class `regions`
- To plot a region, use the `draw` method of the class:

```
>> R.draw(); %or draw(R);
```

Methods of the `regions` class

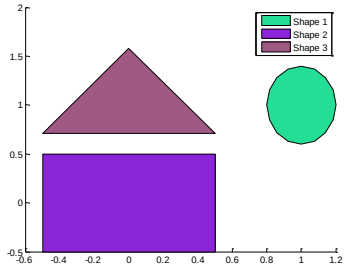
- `regions.rect` returns a rectangle
  - `rect()` returns a square with side 1 centered in `(0,0)`
  - `rect([x0,y0])` returns a square with side 1 centered in (`x0,y0)`
  - `rect([x0,y0],[w,h])` returns a rectangle with base `w` and heigth `h`, centered in `(x0,y0)`
  - `rectN([x0,y0],[x1,y1])` returns a rectangle given two opposite vertices
- Examples:

```
>>r=regions.rect();
>>r=regions.rect([2,3],[3,1]);
```

`region` class: construction

- `regions.circle()` behaves similarly:
  - `circle()` returns a circle with unitary radius centred in the origin, approximated as a polygon with 32 edges
  - `circle([x0,y0])` returns a circle with unitary radius centred in (x0,y0) , approximated as a polygon with 32 edges
  - `circle([x0,y0],[a,b])` returns an ellipsis with semi axis a and b and centred in (x0,y0).
  - `circle([x0,y0],[a,b],n)` returns an ellipsis with semi axis a and b and centred in (x0,y0), approximate a n-sided polygon
- `triangle([x0,y0], side)` returns an equilateral triangle given its center of mass and its side
- Examples:

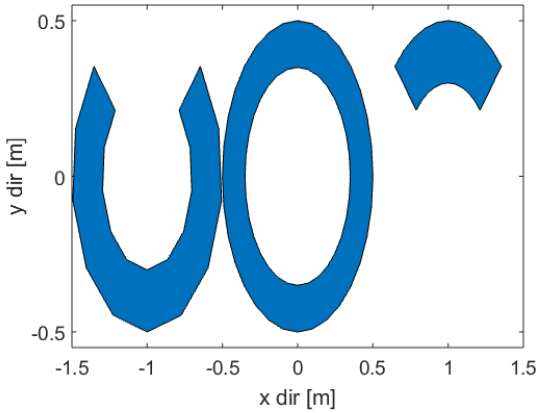
```
>>r=regions.triangle([0,1],1);
>>r=regions.rect();
>>r=regions.circle([1,1],[.2, .4]);
```



`region` class: construction

- `regions.sector` returns a (sector of) annulus
  `>>R=regions.sector(center, extRadius, inRadius, angles, sides)`
  - `center` is the center of the circles
  - `extRadius` and `inRadius` are the inner and outer radius, respectively
  - `angles` is a 2-components vector (in degrees!)
  - `sides` is the number of sides to use to approximate each circumference
- Example:

```
>>draw(
  regions.sector([-1,0],.5,.3,...
      [135,405],10)+
  regions.sector([0,0],.5)+
  regions.sector([1,0],.5,.3,[45,135],10)
  )
```



`region` class: operations

- You can change the position of a `region` object adding a 2 components vector with the x- and y- offsets

```
>>r=regions.rect([3,4]);
>>t=s+[3;4];
```

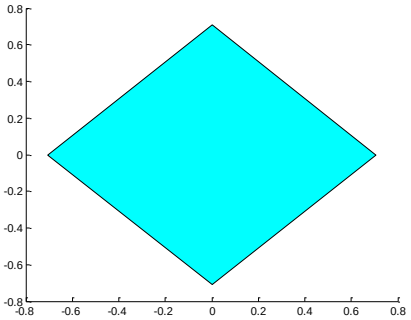
- It is possible to multiply a `region` object by a 2-components vector, shrinking or dilating in the x and y directions respectively

```
>>t=s*[2;2];
>>c=regions.circle()/2;
```

`region` class: operations

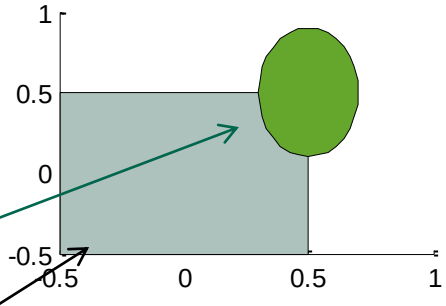
- It is possible to rotate a region using the method `rotate(angle)`

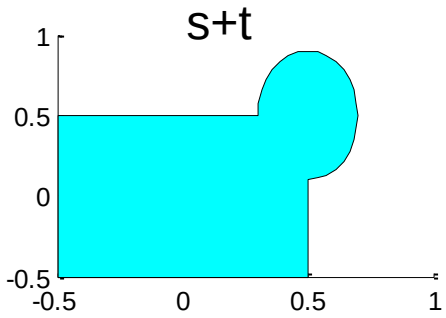
```
>>s=regions.rect().rotate(45);
>>s.draw();
```

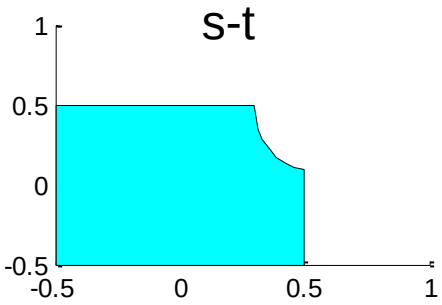


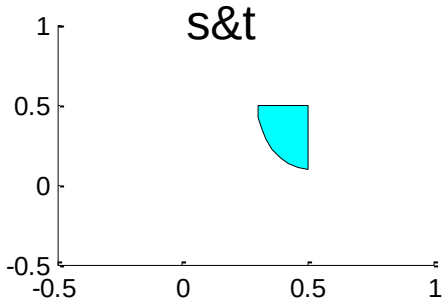
- Finally, it is possible to combine regions using operators +, -, &

```
>> t=regions.circle
   ([0.5,0.5],
    [0.2,0.4],32)
>> s=regions.rect();
```





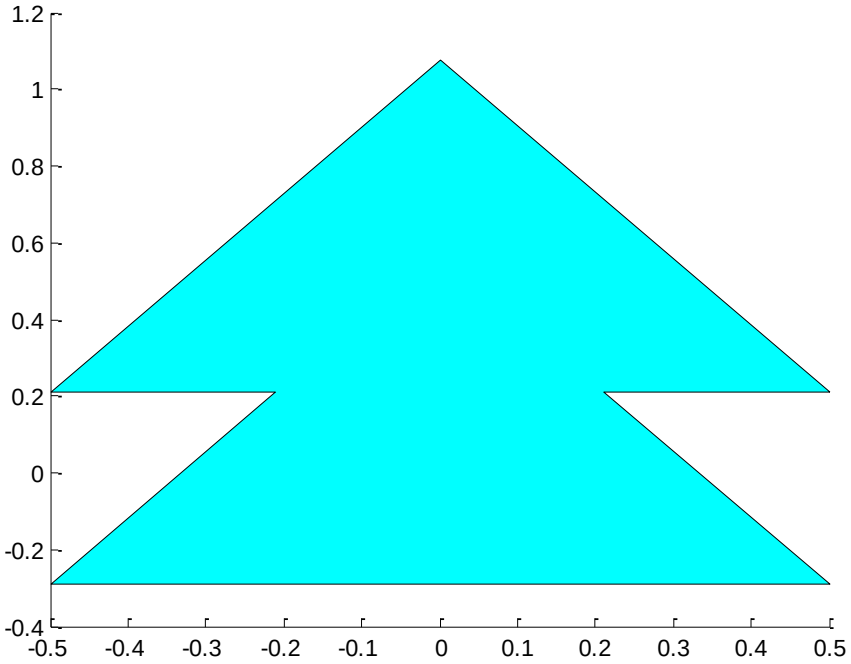




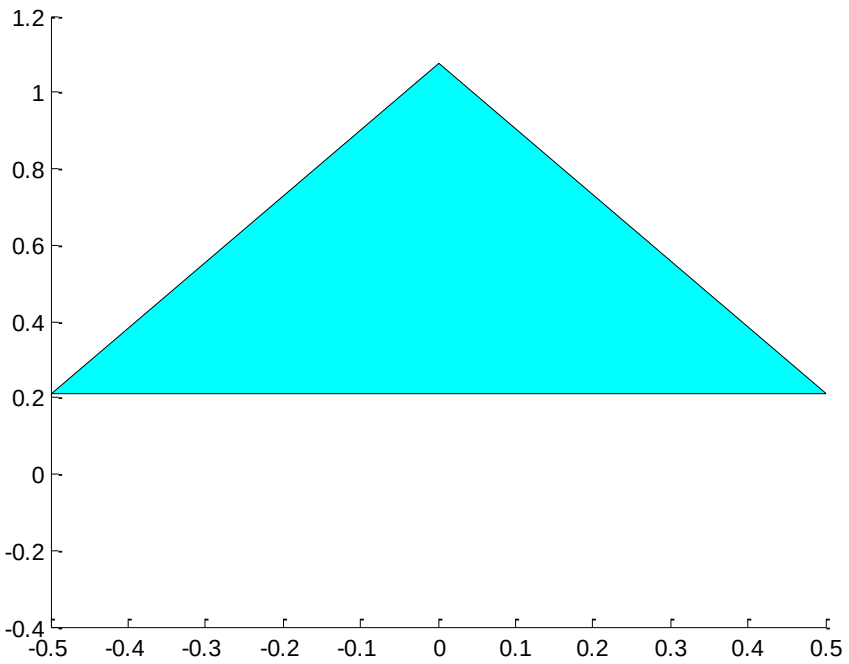
`region` class: operations

- Be carefull with operators precedence!!!

```
regions.triangle+...
(regions.triangle+[0,1])
1]
```




```
regions.triangle+
   regions.triangle+[0
```

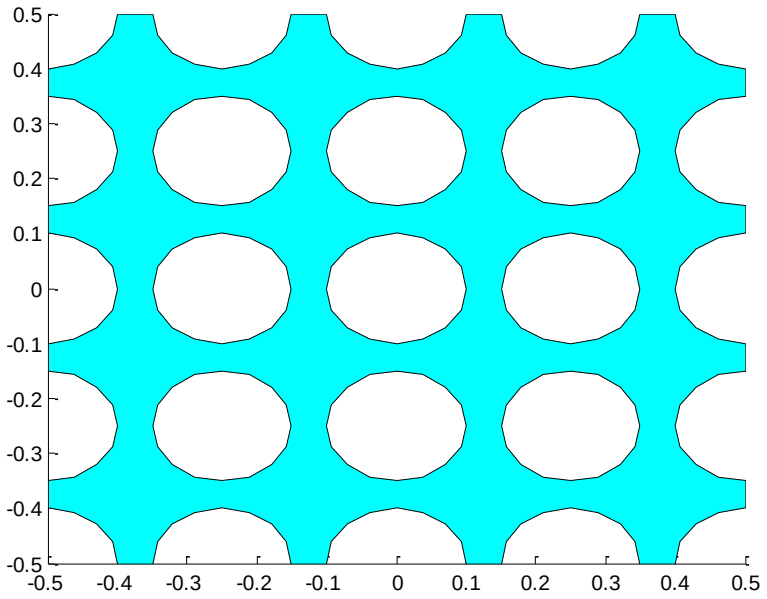


`region` class: operations

- It is then easy to generate regions with holes:

```
R=regions.rect();
for r=0:4,
    for c=0:4,
       R=R-regions.circle([r/4-.5,c/4-
.5],0.1)
    end,
end
R.draw();
```

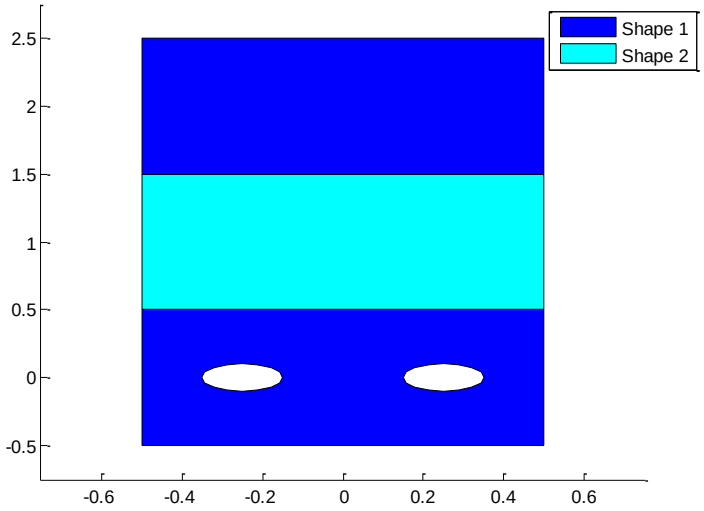
- The region is made by
  10 borders (1 for the outside
  and 9 for the internal holes)



Vectors of `region` objects

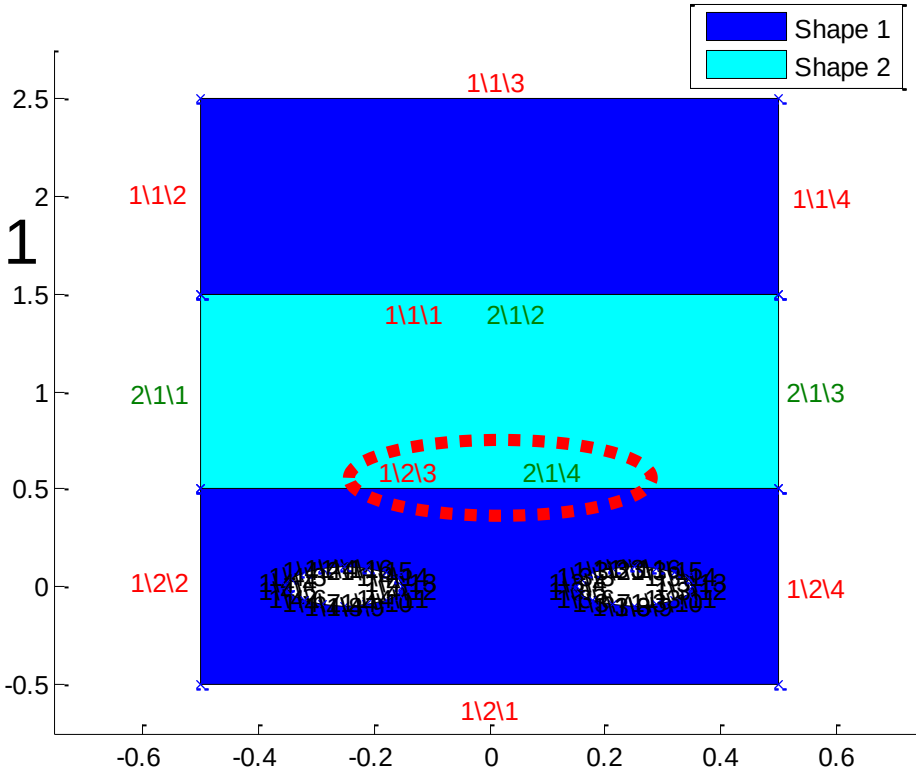
- It is possible to define vectors of `region` objects, used to mesh stratified structures or geometries with **different materials**
- The method `draw` draws every `region` with a different color

```
>>s(1)=regions.rect()-...
 regions.circle([-0.25 0],[0.1 0.1])-...
 regions.circle([0.25 0],[0.1 0.1])+...
 regions.rect([0,2]);
>>s(2)=regions.rect([0,1]);
>>s.draw();
% S(1) has 2 regions, 4 borders
% S(2) has 1 region and 1 border
```



`region` class: edges numbers

- Each edge is uniquely identified; the “address” of each edge can be displayed using
  `>>Draw(S,'edge')`
  which indicates on each edge the number of the Region/Border/Edge
- For instance, 1/2/3 indicates that the selected edge is the 3rd of the 2nd Border of Region 1
- This edge is coincident with 2/1/4, i.e. the 4th edge of the 1st Border of region 2



`region` class: info

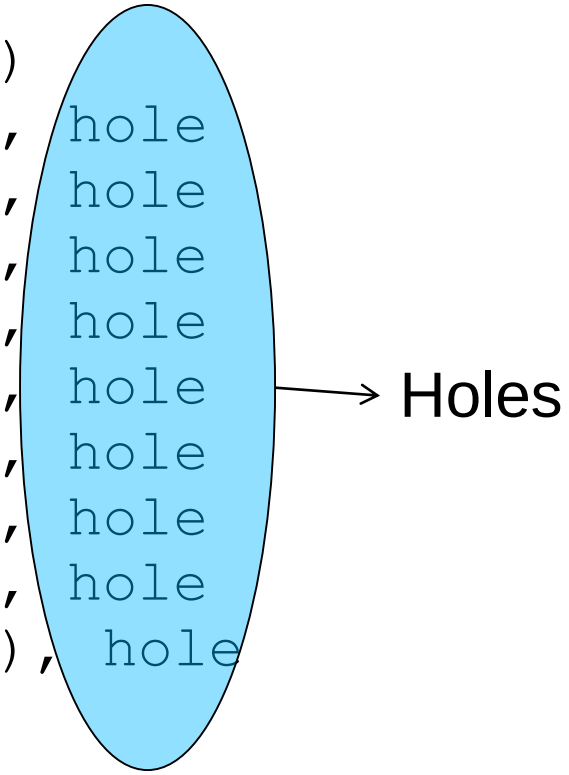
- Additional info are printed when you type the name of the instance on the prompt

```
>>R
 1x2 region objects; 8 borders and 144 nodes in
   total
```

- The method `display(region, level)` allows to obtain info more or less datailed according to the required “level”: the higher this parameter is (from 1 to 3), the more complete is the description which is printed

`region` class: info

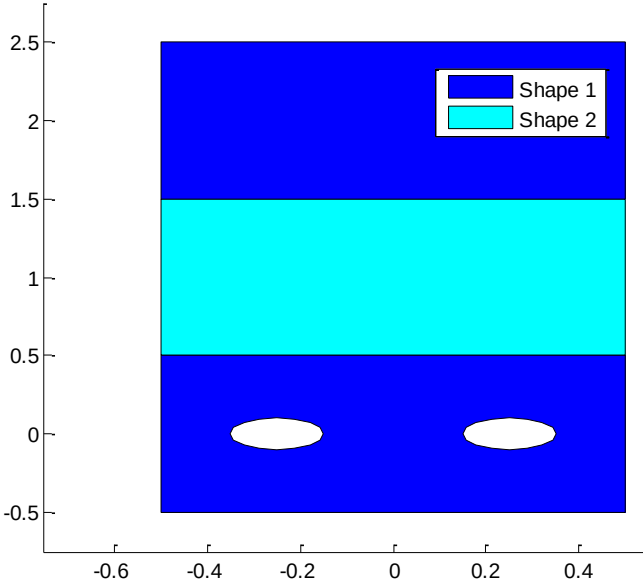
```
>>display(R,3)
  1x1 Region object(s); 10 polygon(s) and 272
  node(s) in total
  Region 1: 10 polygon(s)
      Polygon 1: 128 node(s)
      Polygon 2: 16 node(s), hole
      Polygon 3: 16 node(s), hole
      Polygon 4: 16 node(s), hole
      Polygon 5: 16 node(s), hole
      Polygon 6: 16 node(s), hole
      Polygon 7: 16 node(s), hole
      Polygon 8: 16 node(s), hole
      Polygon 9: 16 node(s), hole
      Polygon 10: 16 node(s), hole
```



`region` class: info

- If the first parameter is a vector, this function prints the details of each region:

```
 >> display(s,2)
1x2 Region object(s); 5 polygon(s) and 44
node(s) in total
Region 1: 4 polygon(s)
    mu: 2
    Polygon 1: 4 node(s)
    Polygon 2: 4 node(s)
    Polygon 3: 16 node(s), hole
    Polygon 4: 16 node(s), hole
Region 2: 1 polygon(s)
    mu: 1
    Polygon 1: 4 node(s)
```



User defined properties

- It is possible to associate user-defined properties to a region, directly when creating it

```
>>s = regions.rect('elasticity',1,'density',2);
```

  or later:

```
>>s = regions.rect();
>>s = s.addProperty('elasticity',1);
>>s = s.addProperty('elasticity',@(x,y)x+5*y);
```

- A property must have a MATLAB valid name (no spaces, accents, ets.); it can be expressed as a constant or as a two variables function @(x,y)
- It is possible to define an arbitrary number of properties:

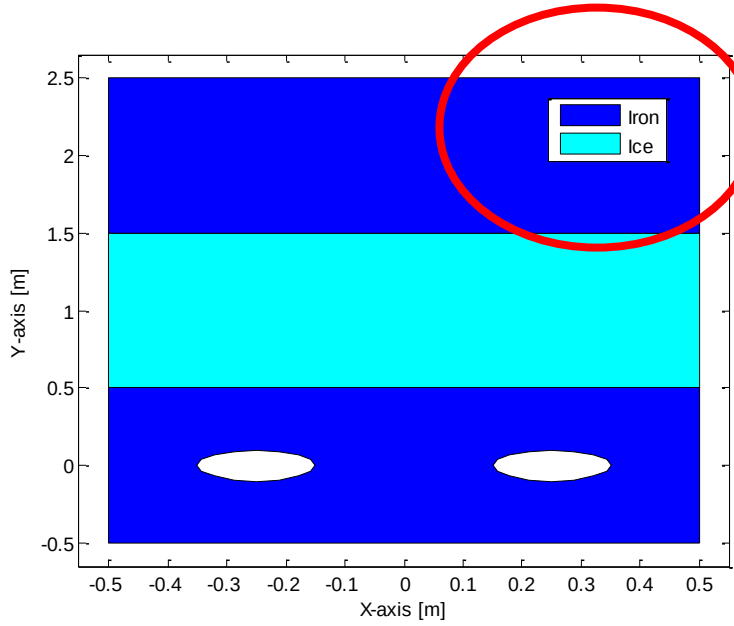
```
>> s.addProperty(s, 'elasticity',5);
>> s.addProperty(s, 'density',@(x,y) sin(x)+y);
>> s.addProperty (s, 'coeff_A',1,'coeff_B',2);
```

User defined properties

- If a property called "`name`" has been defined, it is used as legend entry

```
>>s(1)=...
>>s(2)=...
>>s(1)=s(1).addProperty(s(1), 'name', 'Iron');
>>s(2)=s(2).addProperty(s(2), 'name', 'Ice');
>>figure; s.draw;
```

- Another important property is "`MeshMaxArea`", which is the maximum area of the triangles generated in that region

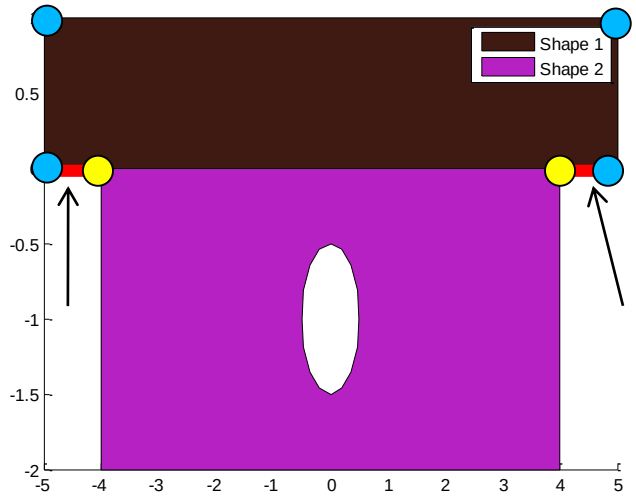


`region` class: adding a node

- In some cases it is important to add one or more points to a pre-existing region
- Let's consider a regions' vector such as

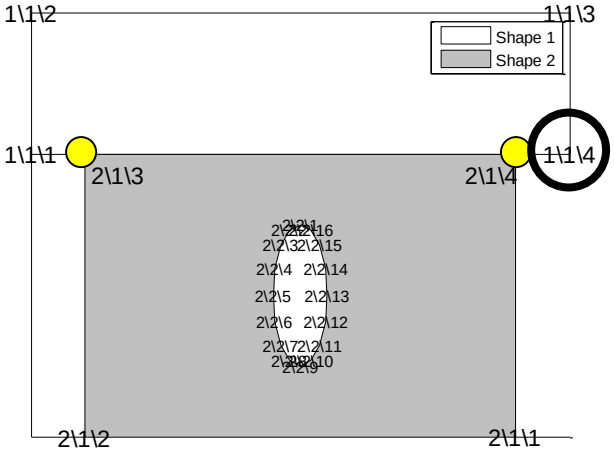
```
>>Sh(1)=regions.rect([0 0.5],[10,1]);
>>Sh(2)=regions.rect([0 -1],[8,2])-...
  regions.circle([0 -1],0.5);
```

- In order to apply the desired boundary conditions on the two segments indicated with the arrows, we need to add two nodes ○ to divide the bottom side of the largest brown rectangle



`region` class: adding a node

- To do so, we start by calling the function `draw(Sh,'n')`, which reports the number of each vertex
- In this case we need to add the nodes [4,0] and [-4,0] after 1\1\4, that is, the fourth node of the first border of the first region
- We then use the method `insertNode` to add the new points




```
Sh(1).Borders(1)=
Sh(1).Borders(1).insertNode(4,[4 0;-4 0]);
```

- The first input parameter indicates AFTER which vertex we want to add the points, the second one contains a matrix with 2 columns (x,y) and a row for each node to insert

Snap to grid

- Numerical rounding errors due to the finite machine precision could generate problems when merging regions
- In order to solve this problem, we can define a grid and snap the regions nodes to it.
- The grid spacing is defined using `grid2D.setSteps(dx,dy)`:

```
>> grid2D.setSteps (0.01, 0.01);
>> display(regions.rect([0,0],[sqrt(2), sqrt(2)]),3)
1x1 Region object(s); 1 polygon(s) and 4 node(s)
Region 1: 1 polygon(s)
    Polygon 1: 4 node(s)
   -0.7100   -0.7100    0.7100    0.7100
   -0.7100    0.7100    0.7100   -0.7100
```

- `grid2D.remove()` removes the grid
- Initially, and after `clear all`, the grid step is 1e-10

Boundary conditions

- Each edge of each border is associated to a B.C., trough an object of class `boundary`.
- By default, all the edges receive a homogeneous Dirichlet B.C.
- We can apply any boundary condition (Dirichlet, Neumann , Robin, Periodic, continuity) using the methods available in the `boundaries` class:

```
S(1).Borders(1).Bc(1)=boundaries.dirichlet(3)
S(1).Borders(1).Bc(3)=boundaries.neumann(4)
S(1).Borders(1).Bc(2)=boundaries.robin(h,g)
```

$$u = u_0$$

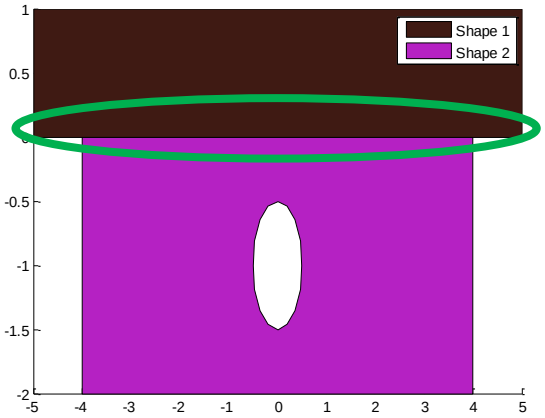
$$\vec{n} \cdot \left(\mu \nabla u \right) = \psi$$

$$\vec{n} \cdot \left(\mu \nabla u \right) + hu = g$$

Boundary conditions

- On the edges belonging to two different regions, we don't need to impose any condition; the nodes there are simply unknown, and we ask for the continuity of the solution

```
S(2).Borders(1).Bc(2)=...
    boundaries.continuity();
```



- Finally, if the geometry we consider is the elementary cell of a periodic structure, we may apply periodic solutions to the borders

```
S(1).Borders(1).Bc(2)=...
    boundaries.periodic(@(x,y)periodic_law);
```

Boundary conditions

- You can pass not only constant values

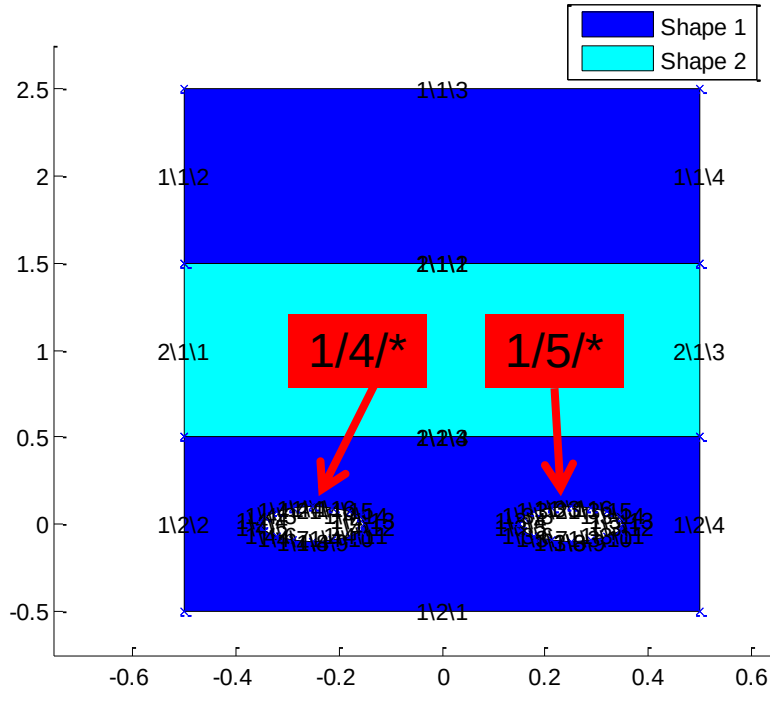
```
>>S.Borders(1).Bc(1)=boundaries.dirichlet(3)
```

  but also functions of (x,y) to obtain position-dependent boundary conditions

```
>>S.Borders(1).Bc(3)=boundaries.dirichlet(@(x,y)x+y)
```

- To find the indices of the edges, use `s.draw(s,'e')`
- To assign Omogeneous Neumann conditions on the two cavities, we use

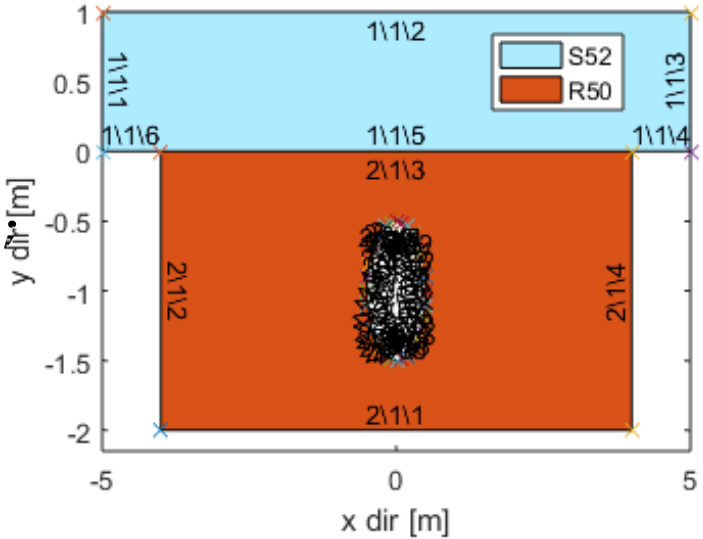
```
>>b(1).Borders(4).Bc(:)=
        boundaries.neumann(0);
>>b(1).Borders(5).Bc(:)=
        boundaries.eumann(0);
```



Boundary conditions

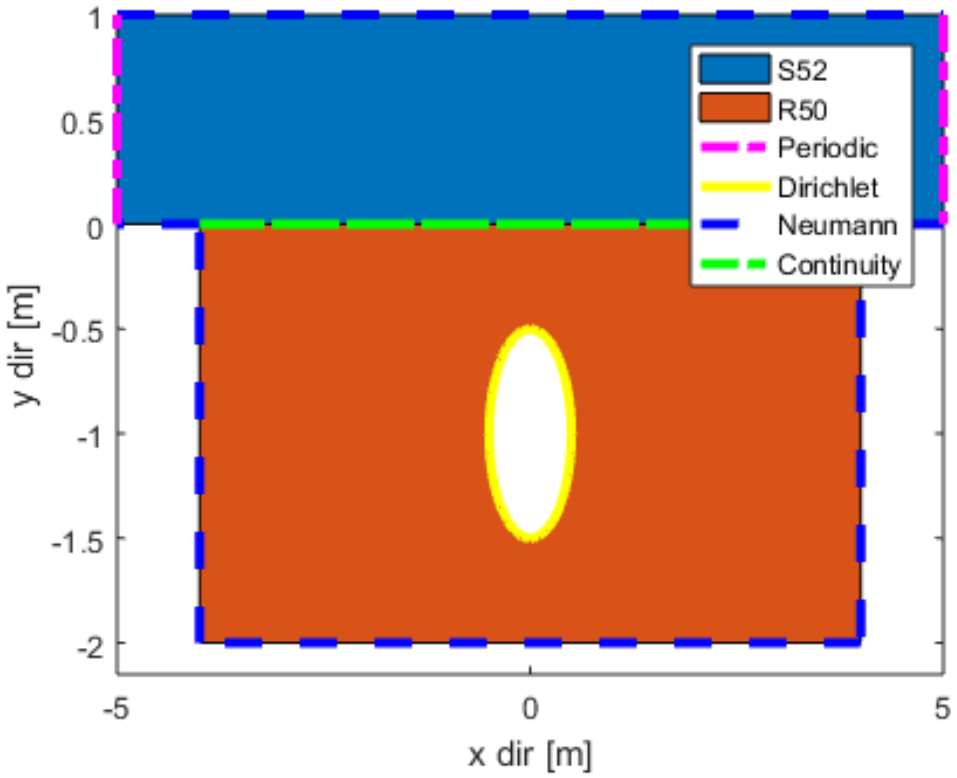
- To verify if all the boundary conditions were correctly assigned, we call `S.draw(Region,'bc')`

```
Sh(1)=regions.rect([0,0.5],[10,1]);
Sh(2)=regions.rect([0,-1],[8,2])-regions.circle([0,-
1],0.5);
Sh(1).Borders(1)=...
   Sh(1).Borders(1).insertNode(4,[4,0;-4,0]);
figure; Sh.draw('edge');
Sh(1).Borders(1).Bc([2, 4, 6])=boundaries.neumann(0);
Sh(1).Borders(1).Bc(5)=...
   boundaries.continuity();
Sh(1).Borders(1).Bc([1, 3])=...
   boundaries.periodic(@(x,y)[-x,y]);
Sh(2).Borders(1).Bc([1, 2, 4])=...
   boundaries.neumann(0);
Sh(2).Borders(1).Bc(3)=...
   boundaries.continuity();
figure; Sh.draw('bc');
```



Boundary conditions

- In the resulting figure, every side is drawn with a color depending on the applyed boundary condition:
  - Yellow: Dirichlet
  - Blue: Neumann
  - Green: Continuity
  - Red: Robin
  - Magenta: Periodic

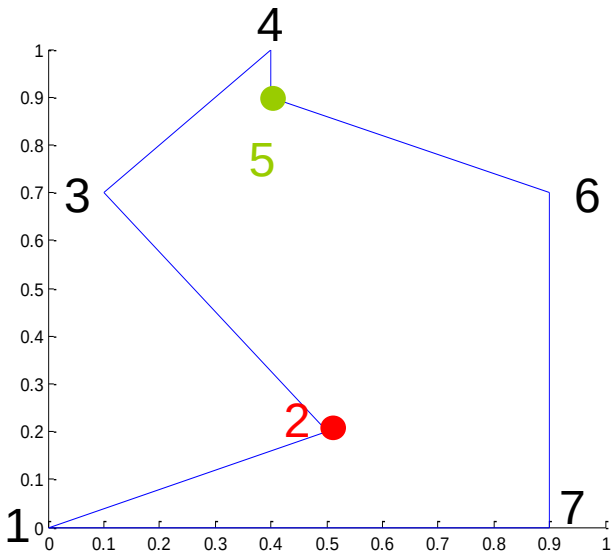



```
1\1\[2 4 6]   Neumann(0)
1\1\5         Continuity
1\1\[1 3]     Periodic
2\1\[1 2 4]   Neumann(0)
2\1\3         Continuity
Altrove       Dirichlet
```

Mesh creation

- The software we will use to generate the mesh is based on Triangle, A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator, by J.R. Shewchuk https://www.cs.cmu.edu/~quake/triangle.html
- Let's consider a generic region defined by its coordinates:

```
coord=[ 0, 0;
     .5,.2; .1,.7;
     .4, 1; .4,.9;
     .9,.7; .9, 0];
```



- The region object is created as

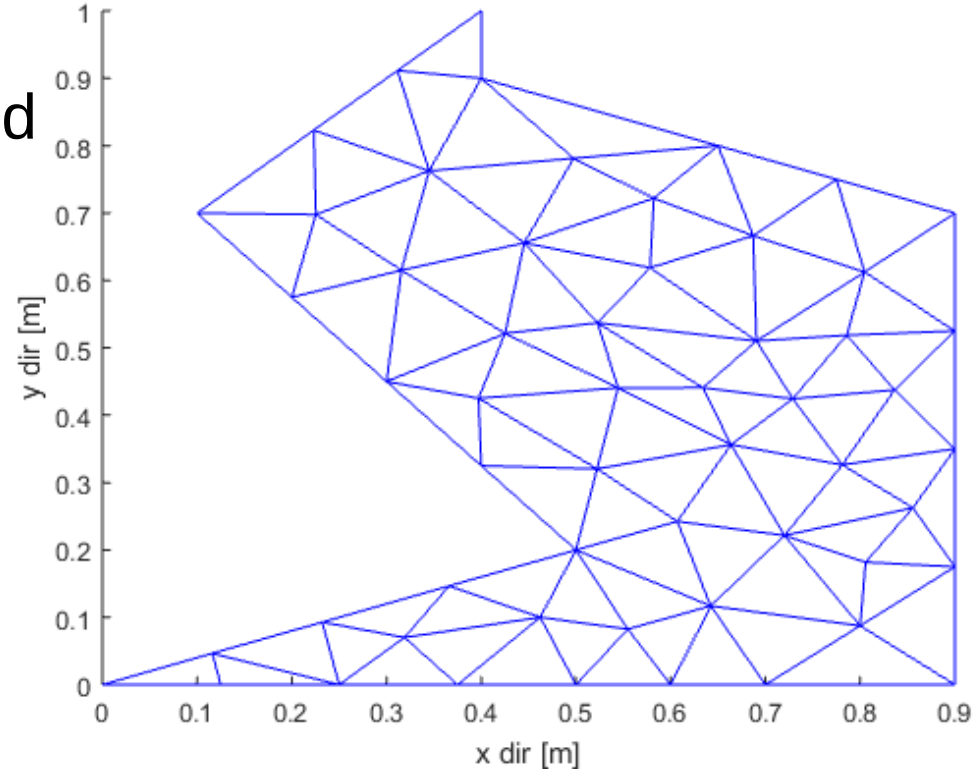
```
>>s=region(coord(:,1),coord(:,2));
```

Mesh creation

- To build a triangular mesh with homogeneous Dirichlet boundary conditions and with a maximum triangle area equal to 0.01 it is sufficient to call the function `Mesh` as:

```
>>Me=mesh2D(s,0.01);
```

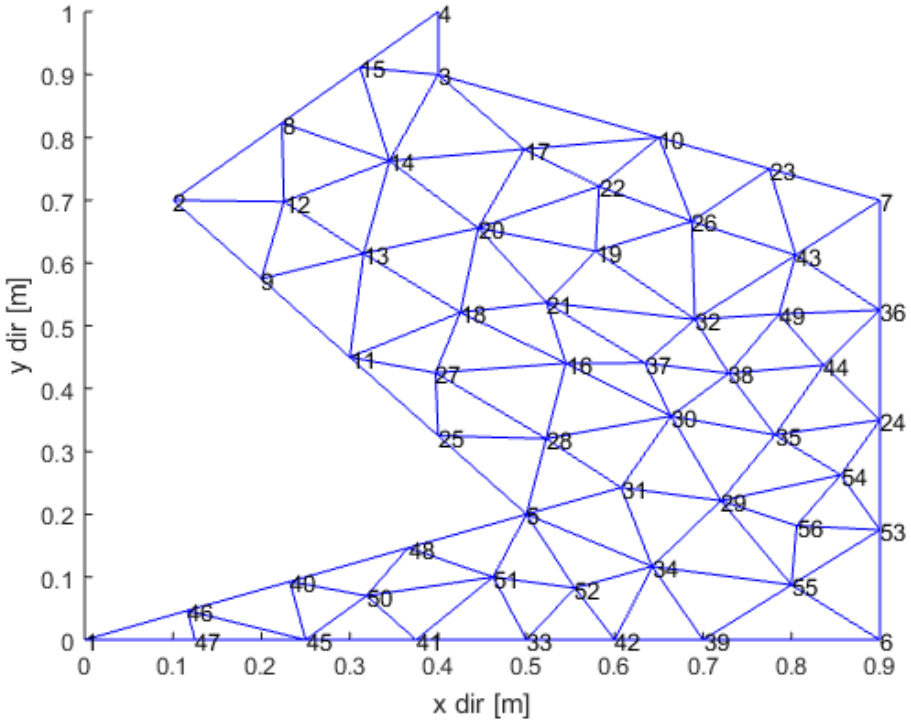
- `Me` is an instance of the `mesh2D` class, implemented in `mesh2D.m`
- The method `Me.draw` allows to draw the result



Nodes and edges numbering

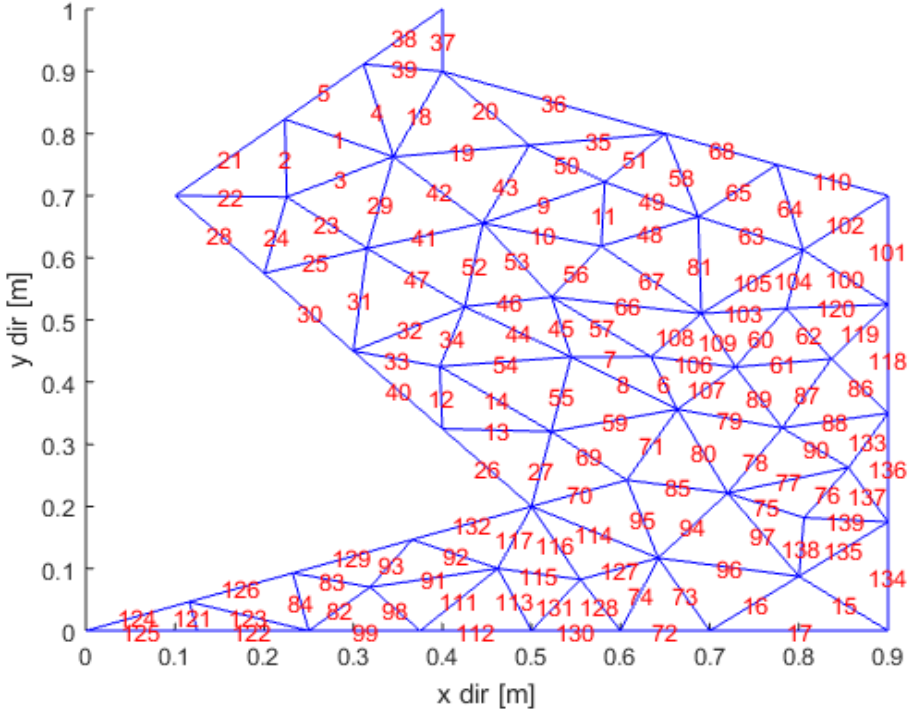
Nodes

`Me.draw('n')`



Edges

`Me.draw('e')`



`Mesh2D` object properties

```
>>Me
```

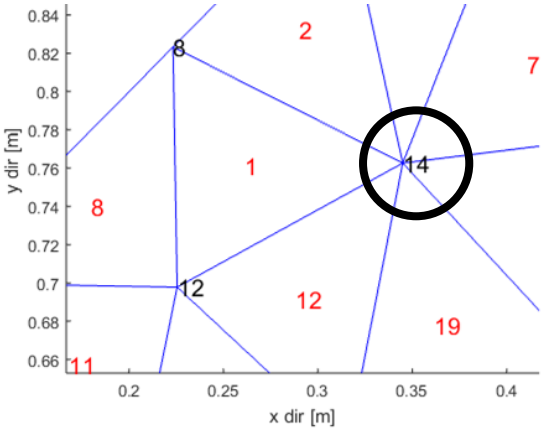
```
          Nodes:   [1x1 struct]
                     X:              [56x1 double]
                     Y:              [56x1 double]
                     Dof:            [56x1 double]
                     TwinNode:       [56x1 double]

      Triangles:   [1x1 struct]
                     Vertices:       [84x3 double]
                     CenterOfMass:   [1x1 struct]
                     Areas:          [84x1 double]
                     Region          [84x1 double]

          Edges:   [139x2 double]

             BC:   [1x1 struct]
                     DirichletNodes:[26x2 double]
                     NeumannEdges:  [0x2 double]
                     RobinEdges:    [0x3 double]

MatrixContributions:   424
              Time:   [0 0]
           Regions:   [1x1 region]
```

```
Me.Nodes.X [56,1],
Me.Nodes.Y [56,1]
```

- `Me.Nodes` contains information about the triangles vertices; it's a struct with fields `X, Y, Dof, TwinNode`
- `Me.Nodes.X` and `Me.Nodes.Y` are column vectors, storing in each row the x and y coordinate of a mesh node. In this case, they have 56 rows→we are dealing with 56 nodes

```
>>[Me.Nodes.X(14), Me.Nodes.Y(14)]
ans= 0.3450    0.7626
```

- The field `Me.Nodes.TwinNode` is used to identify the nodes (in the global numbering) which belongs to edges with periodic boundary conditions



`Me.Nodes.Dof [56x1]`

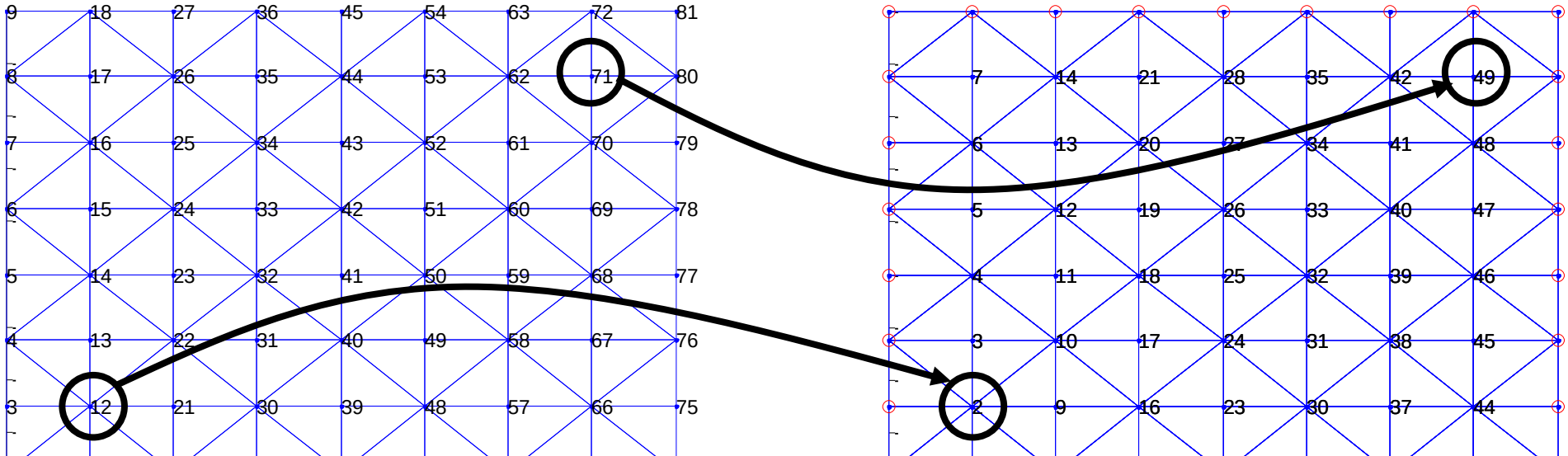
- The nodes can be divided into two different groups:
  - nodes belonging to edges with Dirichlet boundary conditions, where the solution of the problem is already known
  - nodes belonging to the inner part of the domain, or lying on an edge with Neumann/Robin/Periodic B.C.s: they are degrees of freedom to our problem, also called “unknown nodes”
- To build the stiffness and mass matrices we need a numbering covering only the unknown nodes: this second numbering is saved in the column vector `Me.Nodes.Dof`
  - For each node k lying on an edge with a Dirichlet boundary condition, in `Me.Nodes.Dof(k)` is a negative integer referring to a row in `Me.Bc.DirichletNodes`
  - Otherwise, the solution in the node is unknown and we find a progressive integer greater than zero.

`Me.Nodes.Dof [56x1]`

- The vector length is the number of nodes in the mesh
  - For each node k lying on an edge with a Dirichlet boundary condition, in `Me.Nodes.Dof(k)` is a negative integer referring to a row in `Me.Bc.DirichletNodes`
  - Otherwise, the solution in the node is unknown and we find a progressive integer greater than zero

Global numbering

Dof numbering



`Me.Nodes.Dof [56x1]`

- Therefore:
  - If `Me.Nodes.Dof(12)=1`, then I know that node 12 is a dof and in the new numbering it became node 1
  - If `Me.Nodes.Dof(18)=-11`, then I know that node 18 is on a Dirichlet edge
- How many total unknown nodes are there?

```
>>N_i=max(Me.Nodes.Dof)
```

- How many known (Dirichlet) nodes are there?

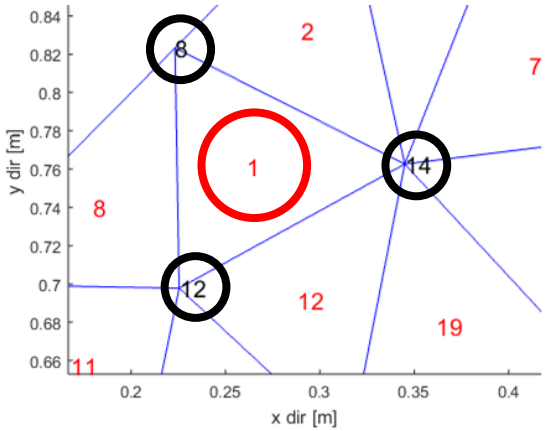
```
>>min(Me.Nodes.Dof)
>>length(Me.Nodes.Dof)-N_i
```

Me.Triangles.Vertices [84x3]

- It contains information about the triangles; it's a struct with fields `Vertices, CenterOfMass, Area, Region`
- `Me.Triangles.Vertices` returns the indices of the 3 vertices of each triangle, in the global numbering; in this case it has 84 rows→we are dealing with 84 triangles

```
>>Me.Triangles.Vertices(1,:)
    ans= 14     8     12
```

means that the triangle 1 vertices are the nodes 14, 8 and 12



What are the x-coordinates of the 3rd triangle's vertices ?

```
>>Me.Nodes.X(Me.Triangles.Vertices(3,:))
```

Vertices numbers

```
Me.Triangles.CenterOfMass,
Me.Triangles.Areas [84,1],
Me.Triangles.Regions [84,1]
```

- `Me.Triangles.CenterOfMass` is a struct with 2 fields, `x` and `y`, containing two vectors with the x- and y-coordinates of the triangles center of mass

```
>> Me.Triangles.CenterOfMass.X(1) % 0.2645
>> Me.Triangles.CenterOfMass.Y(1) % 0.7612
```

- `Me.Triangles.Areas` is a column vector which contains the areas of the triangles

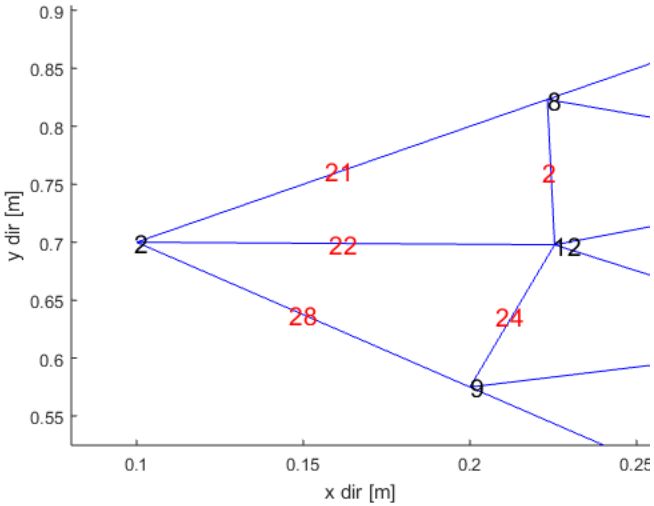
```
>> Me.Triangles.Areas(1)  % 0.0076
```

- `Me.Triangles.Regions` is a column vector: each element indicates the index of the region the triangle belongs to. This is particularly useful to obtain the physical properties associated to each triangle trough its region.

```
Me.Edges: [139x2 double]
```

- The matrix in `Me.Edges` indicates, in each row, the indices of the vertices connected by that edge

```
>>Me.Edges(22,:)
ans= 2 12
```



- What is the length of edge 28?

```
V=Me.Edges(28,:);                    %13 14
X1=Me.Nodes.X(V(1));                 %0.5252
X2=Me.Nodes.X(V(2));                 %0.6991
Y1=Me.Nodes.Y(V(1));                 %0.3879
Y2=Me.Nodes.Y(V(2));                 %0.5602
l=sqrt((X2-X1)^2+(Y2-Y1)^2);         %0.2448
```

or, in a more compact way:

```
l=norm([diff(Me.Nodes.X(V)), diff(Me.Nodes.Y(V))])
```

`Me.Time, Me.MatrixContributions, Me.Regions`

- The property `Me.Time` indicates the time required to generate the mesh (1st element) and to assign the BC.s (2nd elements)
- `Me.MatrixContributions` is the number of non zero elements in the stiffness matrix, for the considered BC.s
- `Me.Regions` is just a backup copy of the `Region` objects passed as input to `mesh2D`

Me.BC.DiricheletNodes [26x2 double]

- The property `Me.BC` contains info about the applied B.C.s
- `Me.BC.DiricheletNodes` is a 2-columns matrix. The 1st column contains the indices, in the global numbering, of the nodes on edges with Dirichlet BC.s; the 2nd column contains the values of the Dirichlet BC.:
  If we have:

```
[ 1     4     %non homog. Dirichlet
  2     4     %non homog. Dirichlet
 10     0     %homog. Dirichlet
 18     0...] %row 11: the 11th Dirichlet node is
               node 18 in the global numbering.
```

then we have `Me.Nodes.Dof(1)=-1`, `Me.Nodes.Dof(18)=-4`, since they aren't dofs

`Me.BC.NeumannEdges`, `Me.BC.RobinEdges`

- The property `Me.BC.NeumannEdges` contain on the first column the indices of the triangle edges which present a Neumann boundary conditions. The second column contains the values of the boundary conditions evaluated in the centre of each edge
- The property `Me.BC.RobinEdges` contain on the first column the indices of the triangle edges which present a Robin boundary condition. The second and third columns contain the values of the g and h coefficients in the center of each edge

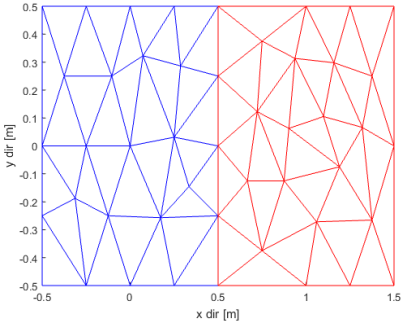
Triangles size and refinement

- The maximum triangle area is calculated as follows:
  - IF the property `MeshMaxArea` has been defined, then the value is used for that region
  - ELSE IF a second parameter (positive scalar) is passed to `mesh2D`, then it is used as maximum triangle area
  - ELSE the program uses as maximum triangle's area 1/25 of the area of the rectangle enclosing all the passed regions
- (Optional) After the first mesh has been evaluated, the mesh is refined using the refining function provided as second parameter to `mesh2D`. The function must accept two input vectors (x and y coordinates) and return the maximum desired area in the provided points.

Triangles size and refinement

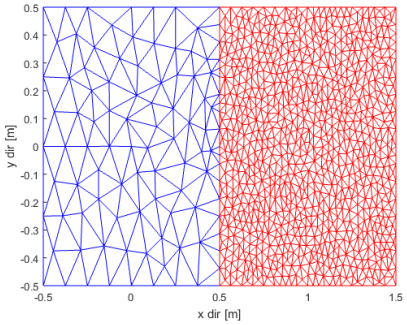
- Example 1

```
s(1)=regions.rect();
s(2)=regions.rect()+[1,0];
Me=mesh2D(s);
max(Me.Triangles.Areas) %0.0400
```



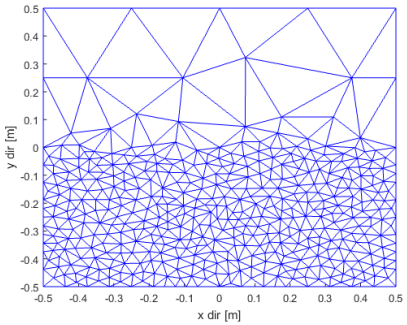
- Example 2

```
s(1)=regions.rect('MeshMaxArea',1e-2);
s(2)=regions.rect()+[1,0];
Me=mesh2D(s,1e-3);    %used in s(2)
T=Me.Triangles;
max(T.Areas(T.Region==1)) %0.0098
max(T.Areas(T.Region==2)) %9.9986e-04
```



- Example 3

```
s=regions.rect();
Me=mesh2D(s,@(x,y)1-0.999*(y<0));
%1 if y>=0, 0.0001 if y<0
```



`InRect` and `InCircle` functions

- The function `InRect` can simplify the functions used for the refinement

```
f=@(x,y)(4.*(x<-.5).*(x>-1.5).*(abs(y)<.5)
-1.*(x>.5).*(x<1.5).*(abs(y)<.5))
```

- It returns 1 if the point *(x,y)* falls in the region centered in (*xc*, *yc*) and with amplitudes *w*, *h*:

```
xyc1=[-1 0];                %1st rectangle center
xyc2=[1 0];                 %2nd rectangle center
wh=[1,1];                   %width and heigth
f=@(x,y)4*InRect([x,y],xyc1,wh)-InRect([x,y],xy2,wh);
%insted of f=@(x,y)(4.*(x<-.5).*(x>-1.5).*(abs(y)<.5)
%              -1.*(x>.5).*(x<1.5).*(abs(y)<.5))
```

- The function `InCircle(point, center, radius)` behaves similarly:

```
InCircle([1,1],[0,0],1)  → ans = 0
InCircle([1,1],[0,0],5)  → ans = 1
```

Triangles size and refinement

- When a complex refining is required, it can be useful to write a function stored in a m-file, e.g.:

```
function s = MyRaf(x,y)
   s=0.1*ones(size(x)); %default value
   for k=1:length(x) %loop over the nodes
   if x(k)<0.5 & y(k)>1
         s(k)=0.05
   elseif x(k)>1
         s(k)=0.01
   end
end
```

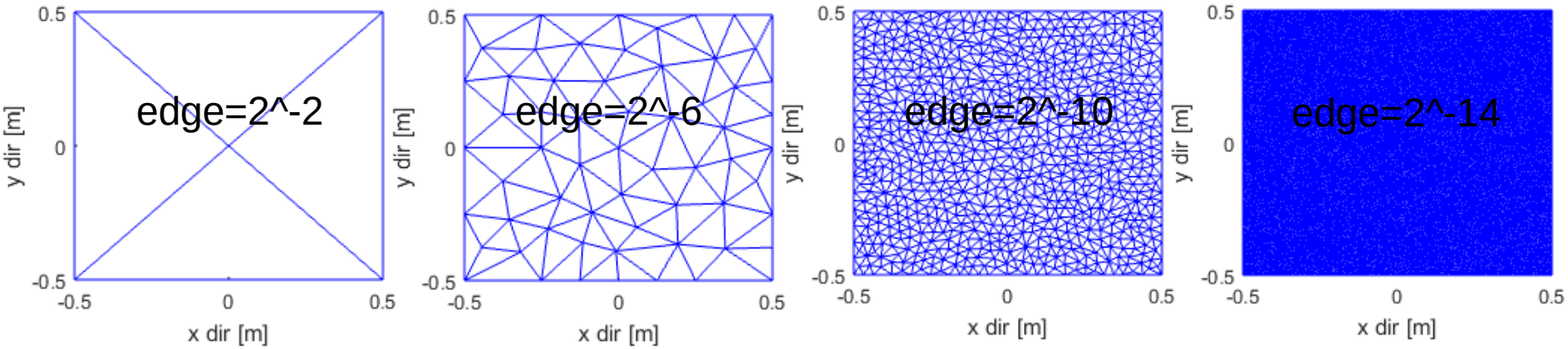
- And the pass it to mesh2D:

```
Me = mesh2D(S,@MyRaf);
```

Triangles size and refinement

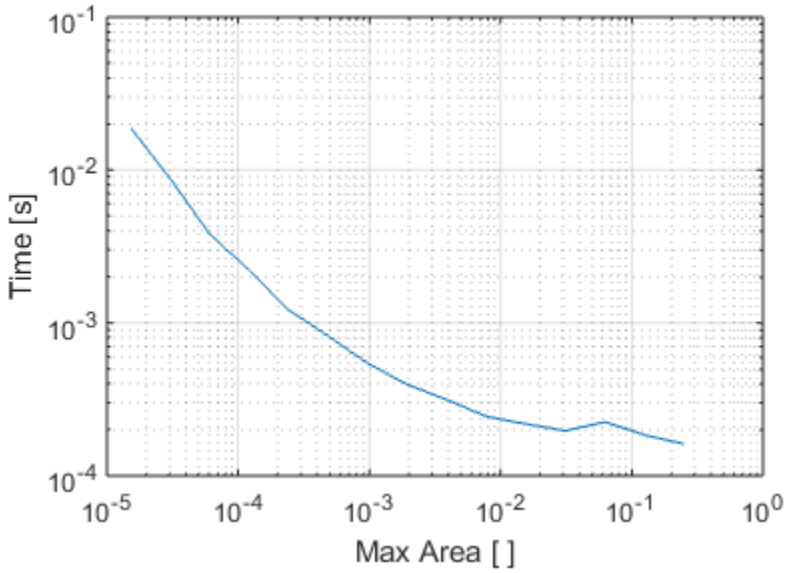
- The time required to calculate the mesh depends on the geometry and on the maximum area

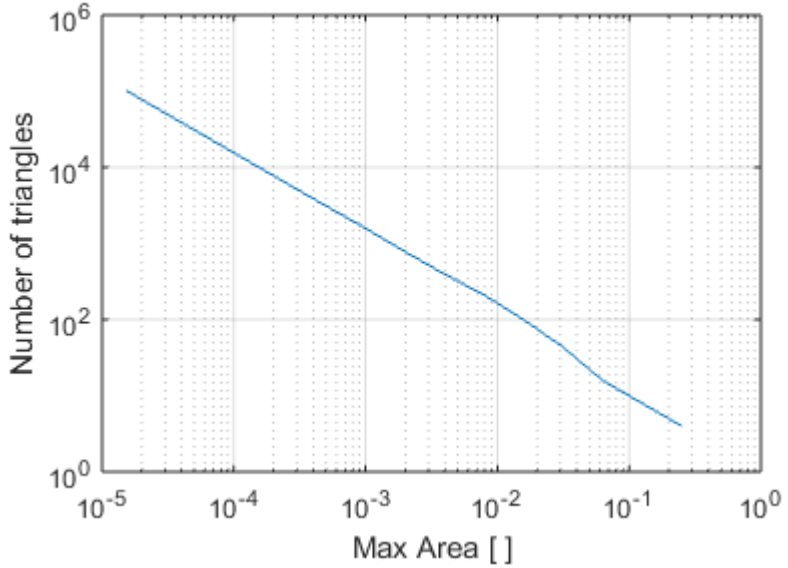
```
sizes=2.^-(2:16); Sh=regions.rect();
Nrep=10;%repeat for better estimate of time
for k=1:length(sizes),
    for rep=1:Nrep
          tic;  Me=mesh2D(Sh, sizes(k)); time(k)=toc;
    end
    tr(k)=size(Me.Triangles.Vertices,1);
end
```



Triangles size and refinement

```
figure; loglog(sizes,time/Nrep)
xlabel('Max Area [ ]');ylabel('Time [s]');

figure; loglog(sizes,tr); grid on;
xlabel('Max Area [ ]');ylabel('# of triangles [s]');
```





`draw` method

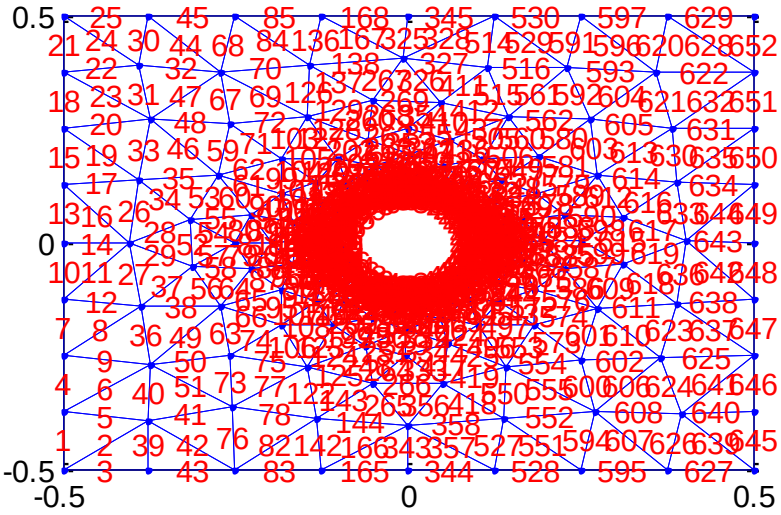
- It allows to draw the result of a mesh

```
>>Me.draw();
```

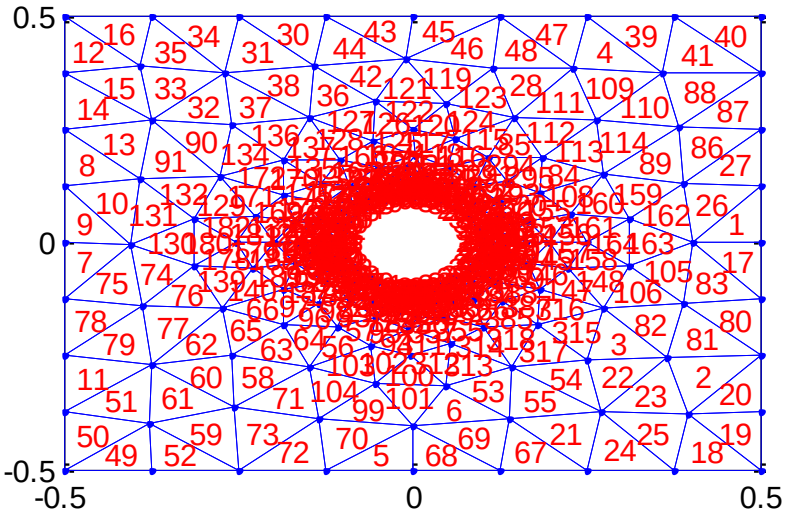
- In order to draw other useful quantities, you can pass a string as second argument:
  - *'edge'*, *'e'* to visualize the sides' numbering
  - *'node'*, *'n'* to visualize the edges' numbering
  - *'triangle', 't'*, to visualize the triangles' numbering
  - *'internalnode'*,*'i'* to visualize the unknown nodes' numbering
  - *'dirichlet'*,*'d'* , to visualize the known nodes' numbering
- Additional parameters will be introduces after the stiffness matrix construction

`draw` method

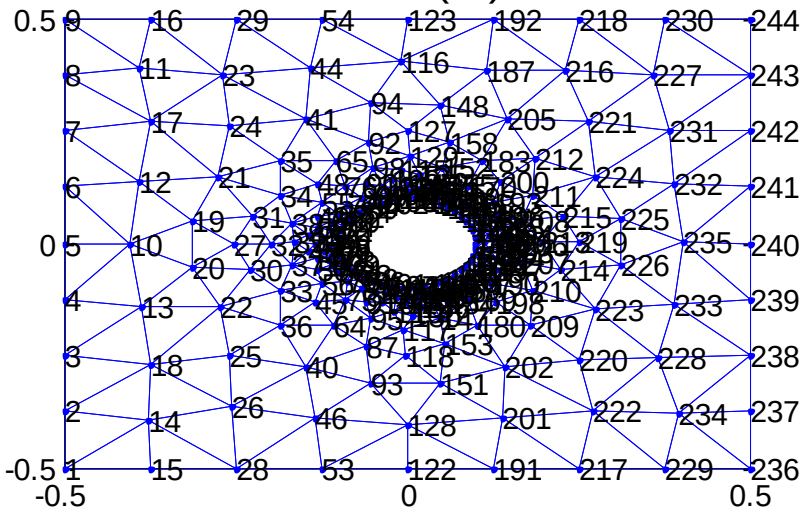
Me.draw('e')



Me.draw('t')



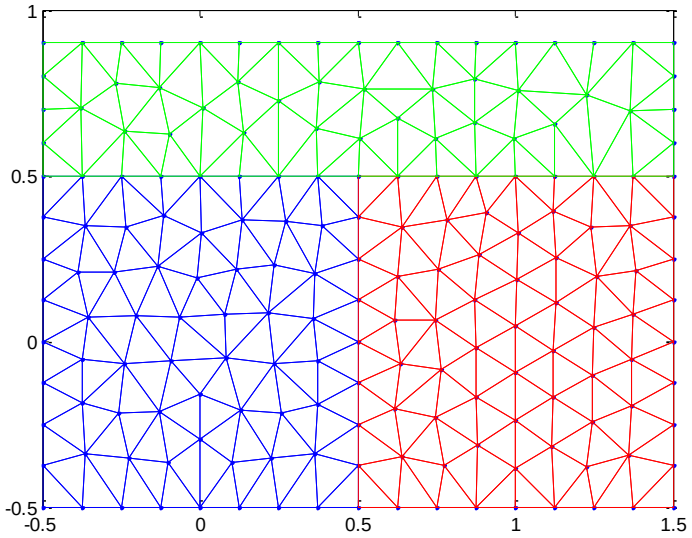
Me.draw('n')



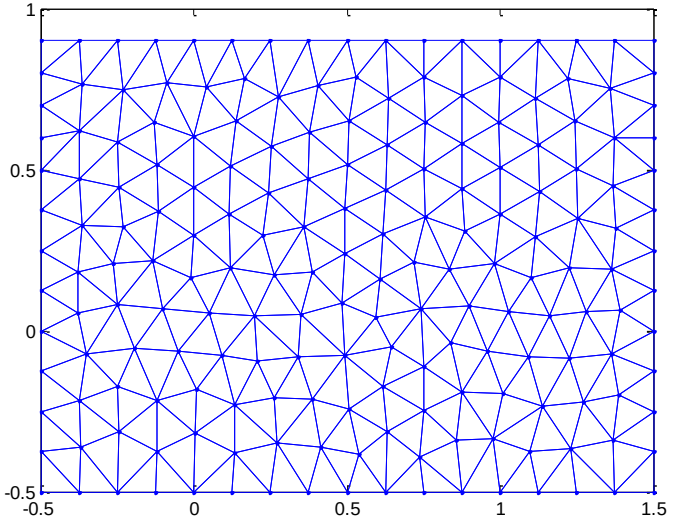
Mesh of a regions' vector

- If you pass to the function `mesh2D` a vector of regions (=different materials), you obtain the mesh of all the regions

```
s(1)=regions.rect();
s(2)=regions.rect([1,0]);
s(3)=regions.rect([.5,.7],[2 .4])
Me=Mesh(s); Draw(Me);
```




```
s=regions.rect()+
regions.rect([1,0])+
regions.rect([.5,.7],[2,.4]);
Me=Mesh(s); Draw(Me);
```

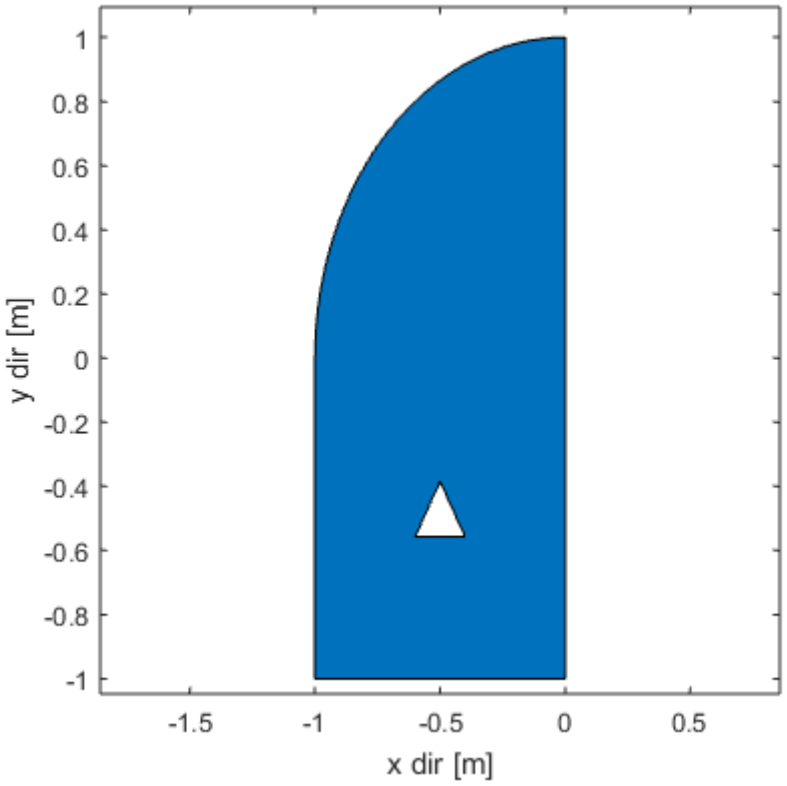


Exercise #1

- Given the region `S=regions.rect()`, assuming a maximum triangle area=1e-3
  - Calculate the average perimeter of the triangles
  - Calculate the average length of the edges
  - Calculate the number of nodes lying in a circle with center in the origin and radius 0.5
  - Calculate the average distance of the centers of mass with respect to the point (2,3)
  - Generate the histogram of the triangles areas (see function `hist` in MATLAB)

Exercise #2

- Generate the region reported on the right, generate a mesh with a suitable max triangle area and calculate the total area of the region
- Hint: generate a circle, then remove its right and bottom parts, then add a square below, finally subtract the triangle



Exercise #3

Let's consider a squared domain 1m x 1m, representing the horizontal section of a chimney. In the centre of the domain we have a circular pipe (external radius: 25cm, internal radius: 15cm) whose internal border has a temperature of 40°C (smoke temperature).

The chimney is isolated on two opposite edges (Homogeneous Neumann B.C.), while on the two other edges we a Robin B.C. with T_∞=10°C and a Dirichlet B.C. with T0=10°C.

Describe the domain in terms of region objects and assign the correct B.C.s; then mesh the domain using a maximum triangle area Tmax=1e-4