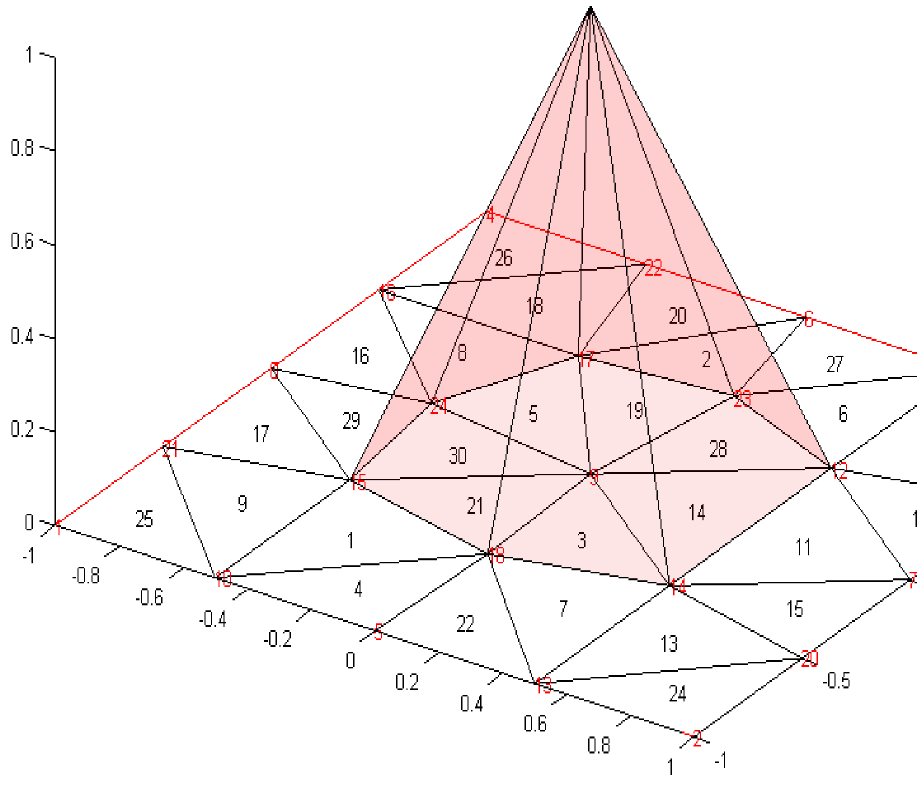
2D Finite Elements
Stationary solution

Version 2017.1

Finite elements

- The method is based on the projection of the weak form of a differential equation on functions with limited support
- We choose the base functions φ_j as pyramidal functions with a vertex in $\underline{x}_j$, unitary height and base made by all the triangles having a vertex in $\underline{x}_j$



Diffusion-transport-reaction equation

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

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

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

Finite elements

- Let’s start with the stationary Poisson equation: we project it on an arbitrary test function v

$$\int_\Omega [\nabla \cdot (\mu \nabla u)] v \,\mathrm{d}\underline{x} = \int_\Omega f \, v \,\mathrm{d}\underline{x}$$

- We apply Green theorem and consider homogeneous Dirichlet and Neumann B.C.

$$\int_\Omega [\nabla \cdot (\mu \nabla u)] v \,\mathrm{d}\underline{x} = \int_\Omega \mu \nabla u \cdot \nabla v \,\mathrm{d}\underline{x} = \int_\Omega f \, v \,\mathrm{d}\underline{x}$$

- This is the weak (variational) formulation of the problem. Any solution of the initial differential equation $\nabla \cdot (\mu \nabla u) = f$ is also a solution of the variational formulation of the problem, while the contrary is only true when some conditions apply on the domain and the problem coefficients. The solution of the variational problem is called *weak* solution of the differential equation

Finite elements

- Once we have chosen the base functions φ_j as N_p functions with finite support defined on the domain $V_{N_p} \subset V$, we express u as $u = \sum_{j=1}^{N_p} u_j \varphi_j$
- We have to consider N_p equations in u_i like $$\sum_{j=1}^{N_p} \int_\Omega u_i \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x} = \sum_{i=1}^{N_p} \int_\Omega f \varphi_i \mathrm{d}\underline{x}$$
- To solve with MatLab the problem in the form $Du=b$ we have to calculate the integrals

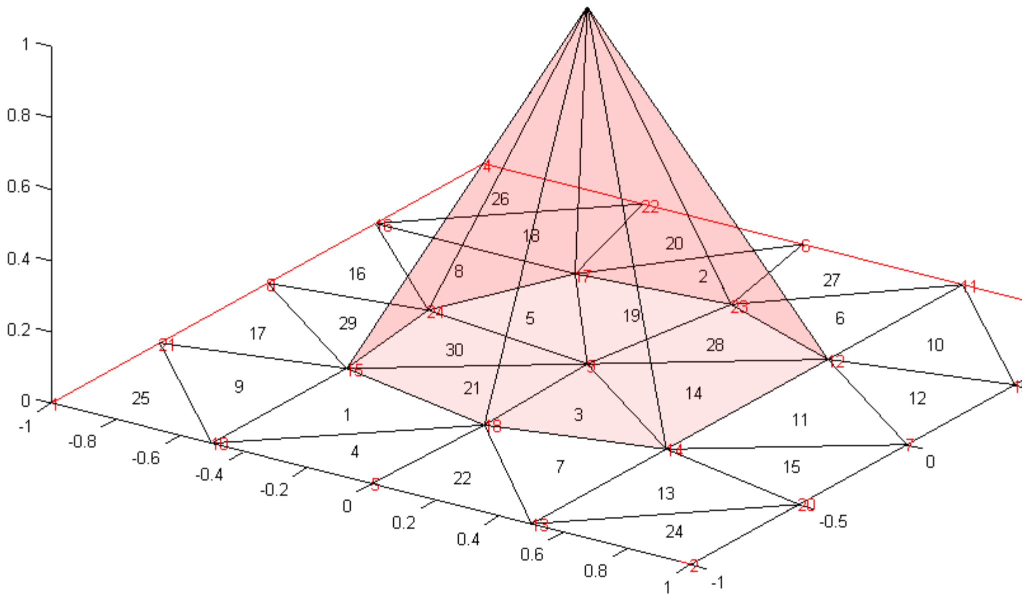
$$d_{i,j} = \int_\Omega \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x} \qquad b_i = \int_\Omega f \varphi_i \mathrm{d}\underline{x}$$

Constant terms vector $b_i = \int_\Omega f \varphi_i \mathrm{d}\underline{x}$

- For any triangle T_k with area A_k belonging to the support φ_j we can estimate the force in its center of mass $\underline{x}_{b,k}$ and consider it as a constant
- We obtain therefore

$$b_i = \sum_k f\left(\underline{x}\right)\Big|_{\underline{x}=\underline{x}_{b,k}} \int_{T_k} \varphi_i \mathrm{d}\underline{x},$$

but this integral is simply the volume of a pyramid with unitary height and base A_k



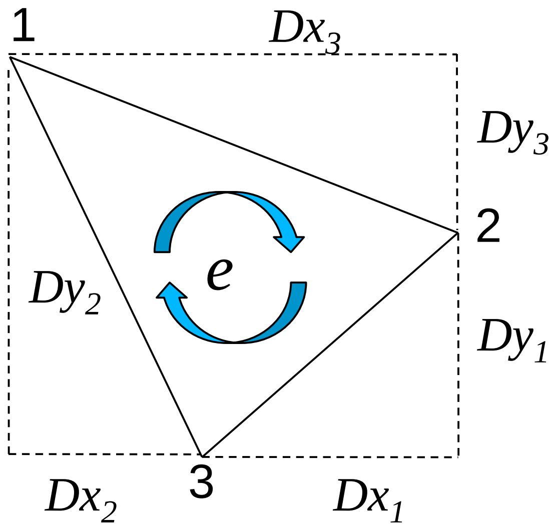
- Therefore,

$$b_i = \sum_k f\left(\underline{x}\right)\Big|_{\underline{x}=\underline{x}_{b,k}} \frac{A_k}{3}$$

Diffusion term calculation

$$d_{i,j} = \int_\Omega \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x}$$

- We need to calculate the gradient of the test functions, computing the x- and y- derivatives of a plane passing in (x_1,x_2,x_3), (y_1,y_2,y_3), $(0,0,1)$
- Let's define $Dx_1=x_3-x_2$, $Dx_2=x_1-x_3$, $Dx_3=x_2-x_1$



- From geometrical considerations we obtain that the equation of the plane passing in 3 points is

$$\det\begin{pmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{pmatrix} = \det\begin{pmatrix} x-x_1 & y-y_1 & z \\ Dx_3 & Dy_3 & 0 \\ -Dx_2 & -Dy_2 & 1 \end{pmatrix} = 0$$

Diffusion term calculation

$$d_{i,j} = \int_\Omega \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x}$$

- If we calculate the determinant we obtain

$$\varphi_3 = z = \frac{xDx_3 - yDx_3 + y_1Dx_3 - x_1Dy_3}{Dx_3Dy_2 - Dx_2Dy_3}$$

- The denominator is $2A_k$: since the vertices are numbered anti-clockwise the following relationships holds:

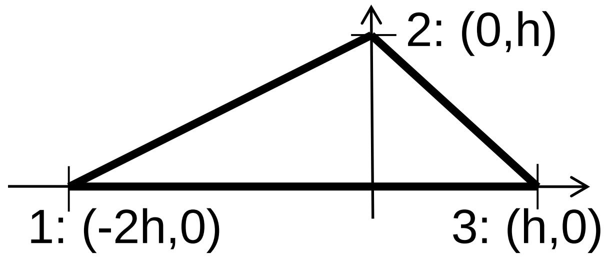
$$A = -\frac{1}{2}\det\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} = \ldots = \frac{1}{2}\left(Dx_3Dy_2 - Dx_2Dy_3\right)$$

- In general we have $\nabla \varphi_j = \begin{pmatrix} \dfrac{\partial \varphi_j}{\partial x} \\ \dfrac{\partial \varphi_j}{\partial y} \end{pmatrix} = \begin{pmatrix} \dfrac{1}{2A} Dy_j \\ -\dfrac{1}{2A} Dx_j \end{pmatrix} = \dfrac{1}{2A}\begin{pmatrix} Dy_j \\ -Dx_j \end{pmatrix}$

Diffusion term calculation

$$\nabla \varphi_3 = \frac{1}{2A}\begin{pmatrix} y_2 - y_1 \\ x_1 - x_2 \end{pmatrix}$$

- Example: consider the triangle



- $A = \frac{2h \cdot h}{2} + \frac{h \cdot h}{2} = \frac{3}{2}h^2$

$$\frac{1}{2A} = \frac{1}{2\frac{3}{2}h^2} = \frac{1}{3h^2}$$

$$\nabla \varphi_3 = \frac{1}{3h^2}\begin{pmatrix} h - 0 \\ -2h - 0 \end{pmatrix} = \frac{1}{3h^2}\begin{pmatrix} h \\ -2h \end{pmatrix} = \frac{1}{h}\begin{pmatrix} 1/3 \\ -2/3 \end{pmatrix}$$

Diffusion term calculation

$$d_{i,j} = \int_\Omega \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x}$$

- We evaluate then

$$d_{i,j} = \int_\Omega \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x} =$$

$$= \sum_k \mu \int_{T_k} \frac{1}{2A_k} \begin{pmatrix} Dy_{j,k} \\ -Dx_{j,k} \end{pmatrix} \cdot \frac{1}{2A_k} \begin{pmatrix} Dy_{i,k} \\ -Dx_{i,k} \end{pmatrix} \mathrm{d}x\mathrm{d}y =$$

$$= \sum_k \frac{\mu}{4{A_k}^2} \left(Dy_{j,k} Dy_{i,k} + Dx_{j,k} Dx_{i,k}\right) \int_{T_k} \mathrm{d}x\mathrm{d}y =$$

$$= \sum_k \frac{\mu}{4A_k} \left(Dy_{j,k} Dy_{i,k} + Dx_{j,k} Dx_{i,k}\right)$$

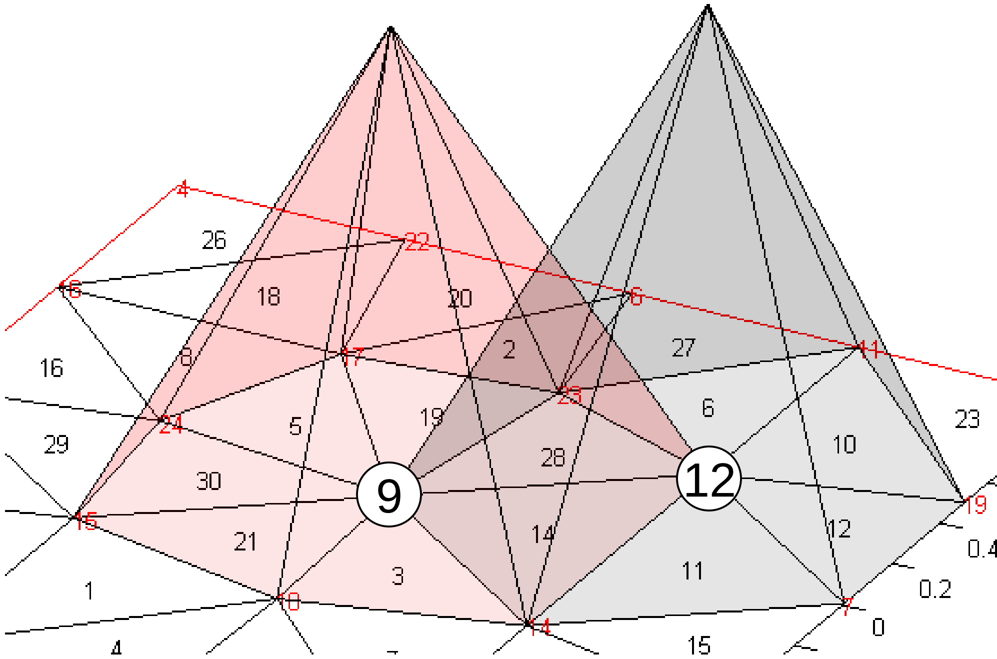
where index k runs on all the triangles with vertices i_k and j_k

Diffusion term calculation

$$d_{i,j} = \int_\Omega \left(\mu \nabla \varphi_j\right) \cdot \nabla \varphi_i \mathrm{d}\underline{x}$$

- If we define $\theta_k(i,j) = \frac{\mu}{4A_k}\left(Dy_{j,k}Dy_{i,k} + Dx_{j,k}Dx_{i,k}\right)$

$$d_{9,9} = \theta_{30}(9,9) + \theta_5(9,9) + \\ +\theta_{19}(9,9) + \theta_{28}(9,9) + \\ +\theta_{14}(9,9) + \theta_3(9,9) + \\ +\theta_{21}(9,9)$$



$$d_{9,12} = \theta_{28}(9,12) + \theta_{14}(9,12)$$

$$\theta_{28}(9,12) = \frac{\mu}{4A_{28}}\left(Dy_{12,28}Dy_{9,28} + Dx_{12,28}Dx_{9,28}\right) =$$

$$= \frac{\mu}{4A_{28}}\left[\left(y_{23} - y_9\right)\left(y_{12} - y_{23}\right) + \left(x_{23} - x_9\right)\left(x_{12} - x_{23}\right)\right]$$

Transport term calculation

$$t_{i,j} = \int_\Omega \left(\underline{\beta} \cdot \nabla \varphi_j\right) \varphi_i \mathrm{d}\underline{x}$$

- Let’s consider the term $\underline{\beta} \cdot \nabla u(x, y)$ with $\underline{\beta} = \left(\beta_x, \beta_y\right)$
- We project on the test functions φj and we expand on function

$$T = \sum_{j=1}^{N_p} \int_\Omega u_i \left(\underline{\beta} \cdot \nabla \varphi_j\right) \varphi_i \mathrm{d}\underline{x}$$

- Recalling the result obtained with the gradient,

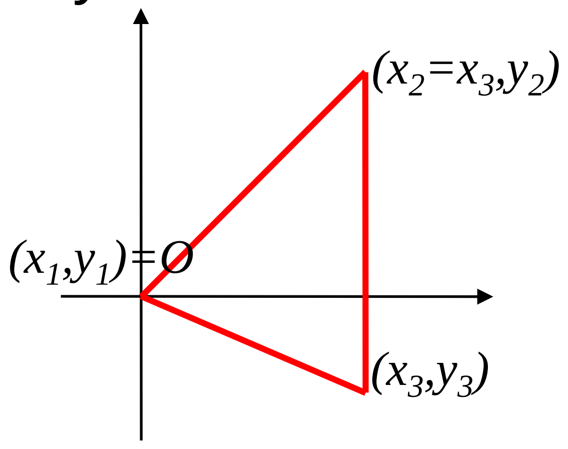
$$t_{i,j} = \int_\Omega \left(\underline{\beta} \cdot \nabla \varphi_j\right) \varphi_i \mathrm{d}\underline{x} = \sum_k \int_{T_k} \frac{1}{2A_k} \left(\beta_x, \beta_y\right) \cdot \begin{pmatrix} Dy_{j,k} \\ -Dx_{j,k} \end{pmatrix} \varphi_i \mathrm{d}x\mathrm{d}y =$$

$$= \sum_k \frac{1}{2A_k} \left(\beta_x Dy_{j,k} - \beta_y Dx_{j,k}\right) \int_{T_k} \varphi_i \mathrm{d}x\mathrm{d}y = \sum_k \frac{1}{6} \left(\beta_x Dy_{j,k} - \beta_y Dx_{j,k}\right)$$

Time-derivative term calculation

$$m_{i,i} = \int_\Omega \varphi_i \varphi_i \,\mathrm{d}\underline{x}$$

- We consider now the term $$\frac{\partial^n}{\partial t^n} u(x,y)$$
- We project on the test functions and we extract the time derivative operator from the space integral
$$\frac{\partial^n}{\partial t^n} \sum_{j=1}^{N_p} \int_\Omega u_i \varphi_j \varphi_i \,\mathrm{d}\underline{x}$$
- Lets orient the triangle with a vertex in the origin and a side parallel to the y axis



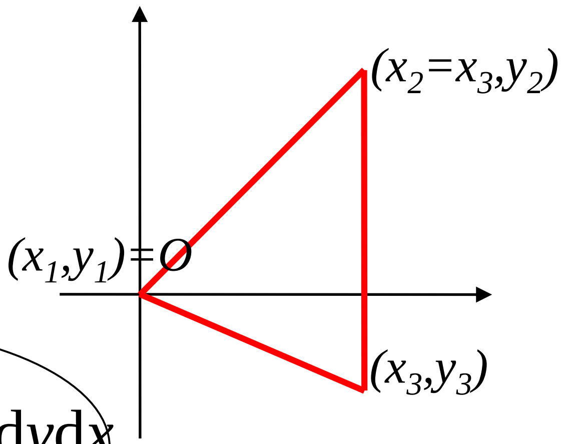
Time-derivative term calculation

$$m_{i,i} = \int_\Omega \varphi_i \varphi_i \,\mathrm{d}\underline{x}$$

- We consider the terms with same index i

$$m_{i,i} = \int_\Omega \varphi_i \varphi_i \,\mathrm{d}\underline{x} = \sum_k \int_0^{x_2} \int_{\frac{y_3}{x_3 \equiv x_2} x}^{\frac{y_2}{x_2} x} \left(1 - x/x_2\right)^2 \mathrm{d}y\mathrm{d}x$$

- For the $m_{i,j}$ term we have



$$m_{i,j\neq i} = \int_\Omega \varphi_i \varphi_j \,\mathrm{d}\underline{x} =$$

$$= \sum_k \int_0^{x_2} \int_{\frac{y_3}{x_3 \equiv x_2} x}^{\frac{y_2}{x_2} x} \left(1 - x/x_2\right) \frac{1}{2A_k}\left(xy_2 - yx_2\right) \mathrm{d}y\mathrm{d}x$$

Plane passing in $(x3,y3,1)$

$$m_{i,j} = \sum_k \frac{A_k}{12}\left(1 + \delta\left(i, j\right)\right)$$

Time-derivative term calculation

$$m_{i,i} = \int_\Omega \varphi_i \varphi_i \,\mathrm{d}\underline{x} = \sum_k \int_0^{x_2} \int_{\frac{y_3}{x_3 \equiv x_2} x}^{\frac{y_2}{x_2} x} \left(1 - x/x_2\right)^2 \mathrm{d}y\mathrm{d}x \qquad m_{i,j\neq i} = \sum_k \int_0^{x_2} \int_{\frac{y_3}{x_3 \equiv x_2} x}^{\frac{y_2}{x_2} x} \left(1 - x/x_2\right) \frac{1}{2A_k}\left(xy_2 - yx_2\right)\mathrm{d}y\mathrm{d}x$$

- Of course, the integrals can be hand calculated
- It is possible to evaluate them using symbolic calculation in `MATLAB`:

```
>> syms x a b real          %real symbolic variables
>> y=cos(x);                %analytic expression
>> diff(y)                  %derivative
      ans = -sin(x);
>> z=int(y+x,x,a,b)         %func. to int., variable, lim.
      z = -sin(a)-1/2*a^2+sin(b)+1/2*b^2
>> subs(z,{a,b},{1,2})      %numerical value: a=1, b=2
      ans = 1.5678
>> simple(cos(x)^2+sin(x)^2) %simplify expr.
      ans = 1
```

Time-derivative term calculation

$$m_{i,i} = \int_\Omega \varphi_i \varphi_i \, \mathrm{d}\underline{x} = \sum_k \int_0^{x_2} \int_{\frac{y_3}{x_3 \equiv x_2}x}^{\frac{y_2}{x_2}x} \left(1 - x/x_2\right)^2 \mathrm{d}y\mathrm{d}x$$

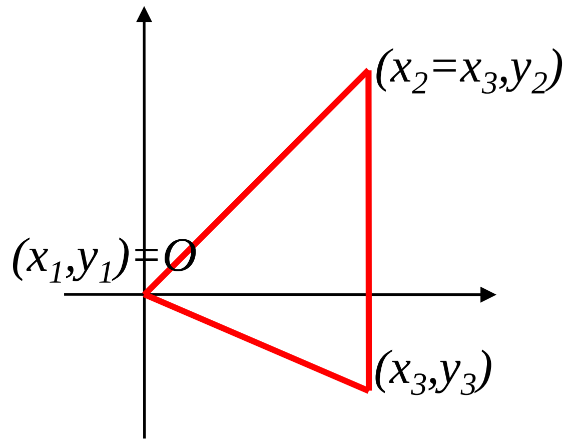
```
%in this case: first evaluate the inner integral, in dy
>>syms y3 x2 y2 x y A real
>>aux=int(
      (1-x/x2)^2,         %function to integrate
      y,                  %integration variable
      y3/x2*x,            %lower limit
      y2/x2*x );          %upper limit
%then evaluate the outer integral, in dx
>>m_ii=int(aux,x,0,x2);
>>simple(m_ii)
      ans = -1/12*x2*(y3-y2)
```

- Base

-2*Area

Triangle heigth

$$m_{i,j} = \sum_k \frac{A_k}{12}\left(1 + \delta(i,j)\right)$$



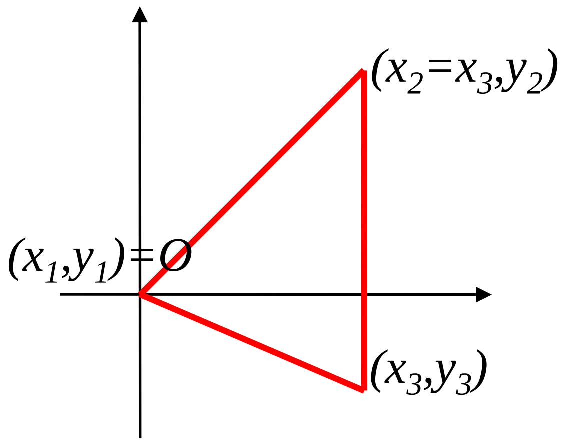
Time-derivative term calculation

$$m_{i,j\neq i}=\sum_k\int_0^{x_2}\int_{\underset{x_3\equiv x_2}{\frac{y_3}{x_2}x}}^{\frac{y_2}{x_2}x}\left(1-x/x_2\right)\frac{1}{2A_k}\left(xy_2-yx_2\right)\mathrm{d}y\mathrm{d}x$$

```
%First, evaluate the inner integral, in dy
>>syms y3 x2 y2 x y A real
>>aux=int(
    (1-x/x2)/(x2*(y2-y3))*(x*y2-y*x2),%func.to int.
      y,                    %integration variable
      y3/x2*x,              %lower limit
      y2/x2*x );            %upper limit
%then evaluate the outer integral, in dx
>>m_ij=int(aux,x,0,x2);
>>simple(m_ij)
      ans = 1/24*x2*(y2-y3)
```

2*Area

$$m_{i,j}=\sum_k\frac{A_k}{12}\left(1+\delta\left(i,j\right)\right)$$



Reaction term calculation

$$r_{i,i} = \int_\Omega \varphi_i \varphi_i \, \mathrm{d}\underline{x}$$

- We consider the term $\sigma u\left(x, y\right)$ with a constant σ
- We project on the test functions

$$R = \sum_{j=1}^{N_p} \int_\Omega \sigma u_i \varphi_j \varphi_i \, \mathrm{d}\underline{x}$$

- This is the same integral we already calculated for the mass matrix!

$$r_{i,j} = \sum_k \sigma_k \frac{A_k}{12} \left(1 + \delta(i, j)\right)$$

Conclusions

- Summarising,

$$\nabla\cdot\left(\mu\nabla u(x,y)\right)\rightarrow Du\rightarrow d_{i,j}=\sum_k\frac{\mu}{4A_k}\left(Dy_{j,k}Dy_{i,k}+Dx_{j,k}Dx_{i,k}\right)$$

$$\underline{\beta}\cdot\nabla u(x,y)\rightarrow Tu\rightarrow t_{i,j}=\sum_k\frac{1}{6}\left(\beta_x Dy_{j,k}-\beta_y Dx_{j,k}\right)$$

$$\frac{\partial^n}{\partial t^n}u(x,y)\rightarrow M\frac{\partial^n}{\partial t^n}u\rightarrow m_{i,j}=\sum_k\frac{A_k}{12}\left(1+\delta(i,j)\right)$$

$$\sigma u(x,y)\rightarrow Ru\rightarrow r_{i,j}=\sum_k\frac{\sigma A_k}{12}\left(1+\delta(i,j)\right)$$

$$f(x,y)\rightarrow b\rightarrow b_i=\sum_k f(\underline{x})\Big|_{\underline{x}=\underline{x}_{b,k}}\frac{A_k}{3}$$

- The diffusion-transport-reaction equation can be expressed as $Du-Tu+Ru=Au=f$

Poisson problem

- We want to write a function to solve the Poisson problem in stationary conditions using the finite element method, using the mesh we just generated, and considering homogeneous Dirichlet boundary conditions (BC)
- The problem to solve is $\nabla \cdot \left(\mu \nabla u(x, y)\right) = f(x, y)$
- We have now to build the stiffness matrix of the problem

Matrices data structure

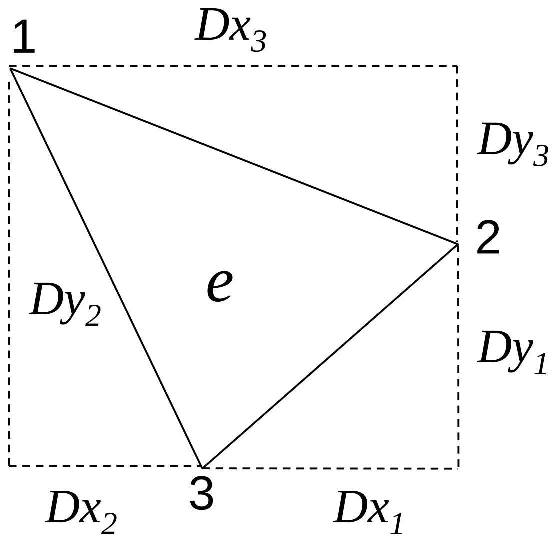
- Since we are dealing with a (possibly) “large” number of dof, it’s important to store the stiffness and mass matrices as sparse.
- Since we are going to build these matrices element by element, it’s faster to use MATLAB function sparse and temporarely store in 3 vectors the row index, column index and value of each entry

```
rows=[1; 1; 2; 3; 1]; cols=[1; 2; 3; 1; 1];
val=[1; 1; 3; 4; 2]; D=sparse(rows, cols, val, 3, 3)
ans = %values in the same position are summed
     3     1     0
     0     0     3
     4     0     0
```

- The number of entries (=length of rows, cols and val vectors) is approximately `Me.MatrixContributions`

Stiffness matrix construction

```
For each triangle $e$
    Evaluate $Dx$, $Dy$, $f(x_b,y_b)$, $mu(x_b,y_b)$
    For each local vertex $i$ $(1 \rightarrow 3)$
        $ii$=Internal Numbering($i$)
        If vertex $ii$ is unknown
            $b(ii) = b(ii) + (Area*f)/3;$
            For each local vertex $j$ $(1 \rightarrow 3)$
                $jj$=Internal Numbering(j)
                If vertex $jj$ is unknown
                    $A(ii,jj)=A(ii,jj)+(Dy(i)\,Dy(j)+Dx(i)Dx(j))/(4*Area)*mu$
```



```
function [D,b] =
dirichletHomo_BuildStiff(Me,f)

%Input:
%Me:  Mesh object
%f:   external source (MATLAB function)
%
%Output:
%D,b: Matrix and constant terms vector of the
%             system Du=b
%
%REMARK:
%The elastic coefficient is provided trough the
%property "mu" assigned to the regions
```

```
function [D,b] =
dirichletHomo_BuildStiff(Me,f)
```

```
%Indices of vertices of each triangle; 3 cols. matrix
V = Me.Triangles.Vertices;
%Area of each triangle; column vector
Areas = Me.Triangles.Areas;
%x and y coordinates of the centres of mass;col. vect.
CenterOfMass = Me.Triangles.CenterOfMass;
%x and y coordinates of the certices; column vectors
X=Me.Nodes.X; Y=Me.Nodes.Y;
%Numbering of Dirichlet(<0) and unknown (>0) nodes
Dof = Me.Nodes.Dof;
%number of internal nodes: we know that the numDof
%unknown nodes are numbered from 1 to numDof in
%Me.Nodes.Dof; its maximum is therefore the number %of
unknown (degrees of freedom)
numDof = max(Dof);
```

```
function [D,b] =
dirichletHomo_BuildStiff(Me,f)
```

```
%vectors preallocation: instead of allocating the
%sparse stiffness matrix with spalloc, we save the
%rows, columns and values corresponding to each
%contribution; at the end, we'll call sparse(...) to
%obtain the diffusion matrix. This solution is much
%faster than using spalloc
row = zeros(Me.MatrixContributions, 1);
col = zeros(Me.MatrixContributions, 1);
d = zeros(Me.MatrixContributions, 1);
pos = 1;  %we increase this index when we add an entry
b = zeros(numDof, 1);

%force, in the centre of mass of each triangle
force = f(CenterOfMass.X,CenterOfMass.Y);

%coefficient in front of the Laplace operator
mu = Me.evaluateProperty('mu');
```

function [D,b] = **dirichletHomo_BuildStiff**(Me,f)

```
for e= 1:size(V, 1)              %main loop over each triangle
    Dx(1) = X(V(e, 3)) - X(V(e, 2)); Dy(1) = Y(V(e, 3)) - Y(V(e, 2));
    Dx(2) = X(V(e, 1)) - X(V(e, 3)); Dy(2) = Y(V(e, 1)) - Y(V(e, 3));
    Dx(3) = X(V(e, 2)) - X(V(e, 1)); Dy(3) = Y(V(e, 2)) - Y(V(e, 1));
    for ni = 1:3 %for each vertex of this triangle
        ii = Dof(V(e, ni)); %ii can be <0 (Dirichlet) or >0 (dof)
        if ii > 0 %is it a dof? If so, 2nd loop
            for nj = 1:3
                jj = Dof(V(e, nj)); %is it a dof as well?
                if jj > 0 %yes, add the contribution to stiffness mat.
                    Dcomp = (Dy(ni)*Dy(nj)+Dx(ni)*Dx(nj))
                    d(pos) = mu(e)* Dcomp / (4*Areas(e));
                    row(pos) = ii; col(pos) = jj;
                    pos = pos + 1; %ready for the next element
                end
            end
            %add the external contribution to constant terms vector
            b(ii) = b(ii) + Areas(e) * force(e) / 3;
        end
    end
end
%build the sparse stiffness matrix D
D = sparse(row, col, d, numDof, numDof);
```

$$\sum_k \frac{\mu}{4A_k}\left(Dy_{j,k}Dy_{i,k} + Dx_{j,k}Dx_{i,k}\right)$$

$$\sum_k f(\underline{x})\big|_{\underline{x}=\underline{x}_{b,k}} \frac{A_k}{3}$$

Putting all together

```
Sh = regions.rect('mu',1);
Me = mesh2D(Sh,0.05);
f = @(x,y)-4*ones(size(x));
[D,b] = dirichletHomo_BuildStiff(Me, f);
u = D\b;
```

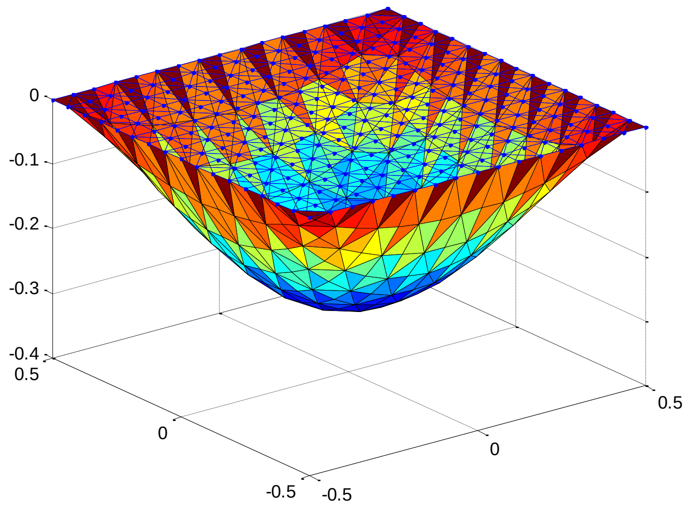
- Vector `u` contains the solution in the unknown nodes only; additional steps are required to generate the vector to be drawn:
  - We allocate a vector having the same size as the number of nodes
    ```
    uu=zeros(size(X));
    ```
  - In the unknown nodes we copy the solution
    ```
    uu(Me.Nodes.Dof>0)=u;               %unknown nodes
    ```
  - In the Dirichlet nodes we impose the known values (in this case, an homogeneous Dirichlet condition)
    ```
    uu(Me.Nodes.Dof<0)=0;     %known nodes
    ```
- The same result can be obtained using method `copyToAllNodes`:
  ```
  uu=Me.copyToAllNodes(u);
  ```

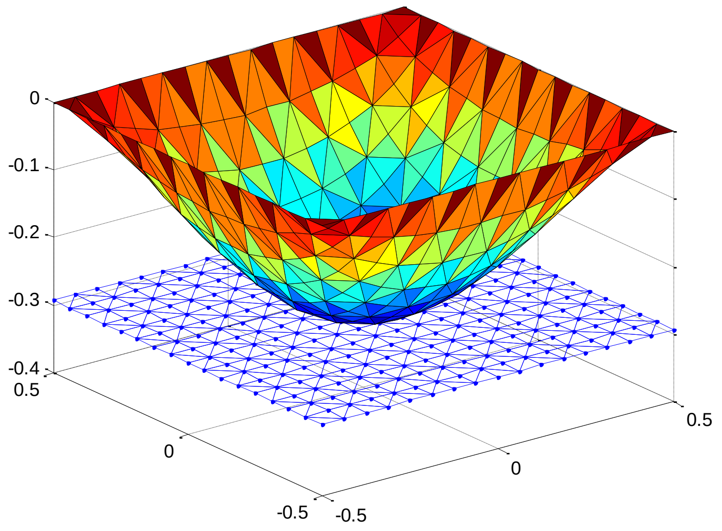
Solution representation

- We finally plot the solution using function `draw`

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

```
>>Me.draw(uu,'offset');
```

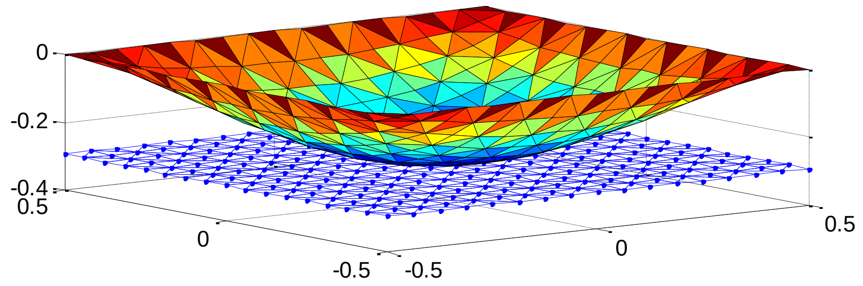


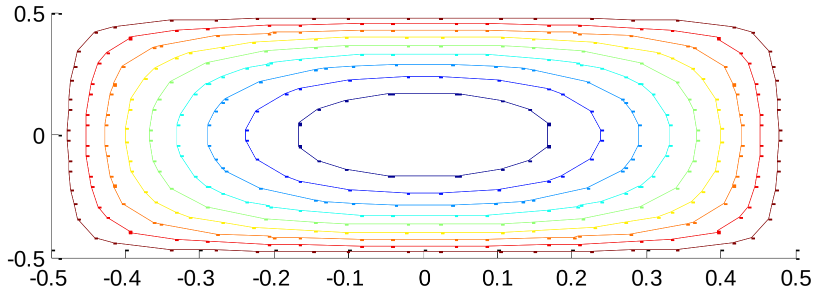


Solution representation

- Another possible visualization

```
figure;
subplot(2,1,1);
Me.draw(uu,'offset');
subplot(2,1,2);
Me.draw(uu,'contour');
```





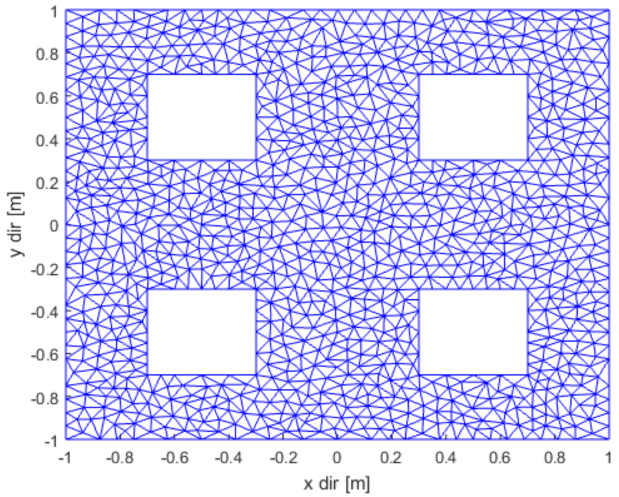
Function `draw`

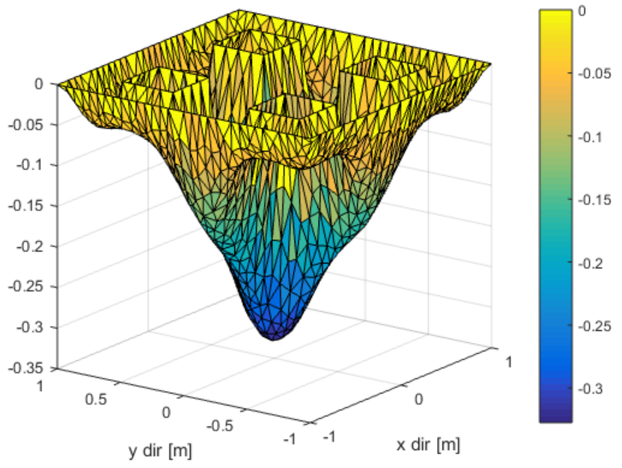
- The function `draw` accepts the following optional arguments

`e, edges`	`numbers the edges`
`n, nodes`	`numbers the nodes`
`t, triangles:`	`numbers the triangles`
`i, internalnodes`	`numbers the unknown nodes`
`d, dirichletnodes`	`numbers the known nodes`
`q, quality`	`draws the quality factor`
`c, contour`	`contour of the solution, hiding the mesh`
`o, offset`	`shows the mesh, aligned to the solution minimum`
`h, hidemesh`	`hides the mesh`
`s, showmesh`	`shows the mesh`

Example

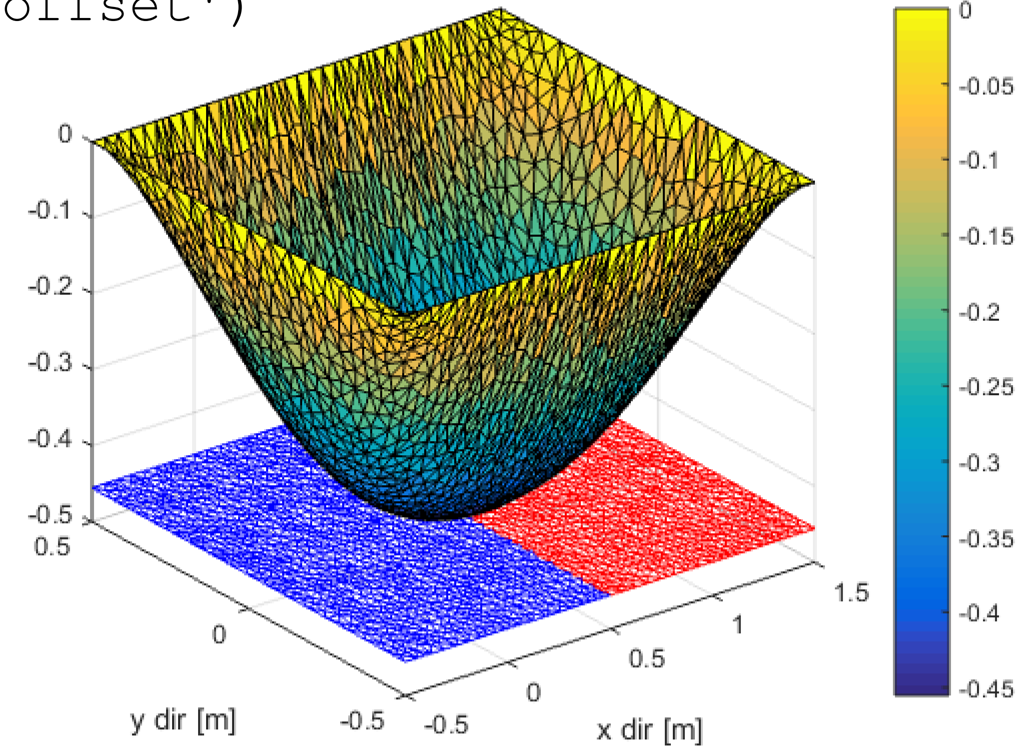
```
S = regions.rect('mu',1) * 2 ...
    -regions.rectN([-0.7, 0.3],[-0.3 0.7]) ...
    -regions.rectN([-0.7, -0.3],[-0.3, -0.7]) ...
    -regions.rectN([0.7, 0.3],[0.3, 0.7]) ...
    -regions.rectN([0.7, -0.3],[0.3, -0.7]);
Me = mesh2D(s);
[A,b] = dirichletHomo_BuildStiff(Me);
u = A\b; uu = me.copyToAllNodes(u);
Me.draw(uu);
```





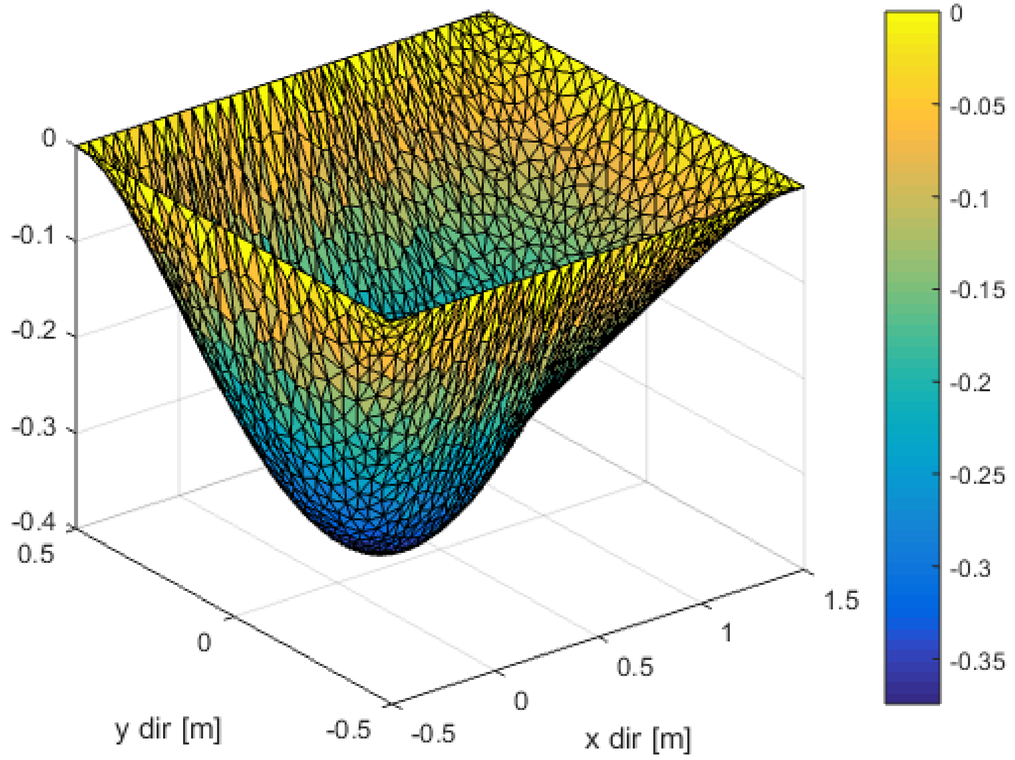
Example (2 regions)

```
Sh(1)=regions.rect('mu',1);
Sh(2)=regions.rect('mu',1)+[1,0];
Me=Sh.mesh2D();
[A,b]= dirichletHomo_BuildStiff(Me);
uu=Me.copyToAllNodes(A\b);
Me.draw(uu,'showmesh','offset')
```



Example
(1 region, non-constant material)

```
Sh = regions.rect([0.5,0],[2,1], 'mu',@(x,y)(1+(x>0.5)));
Me = mesh2D(Sh);
f = -4*ones(size(x));
[D,b] = dirichletHomo_BuildStiff(Me, f);
uu = Me.copyToAllNodes(D\b);
figure;
Me.draw(uu);
```



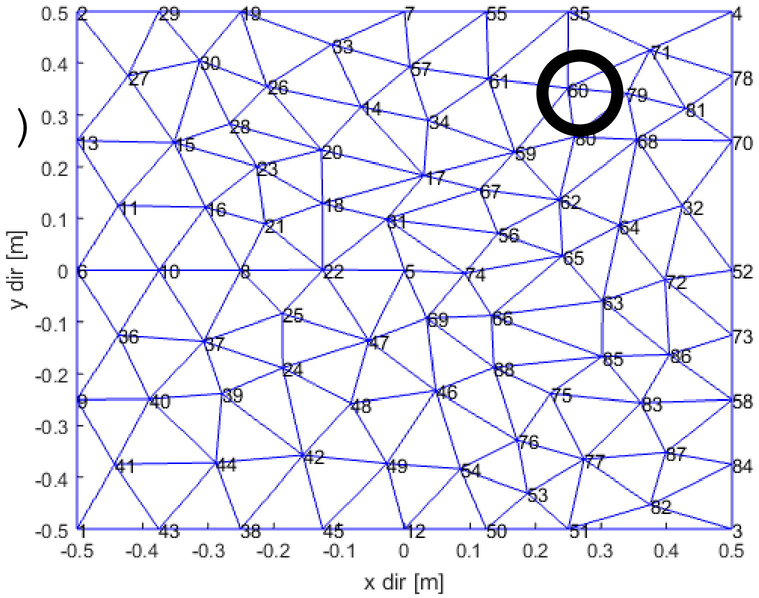
Solution correctness

- In order to correctly judge the results of any numerical elaboration, it is always important to evaluate with attention if they represent a realistic result or not, and to try to compare them, if possible, with solutions obtained with other models or from experimental measurements.
- When we work with Finite Elements, we can also check if the used mesh was refined enough or not, by estimating how much the solution (in a point, or along a line) changes by changing the maximum triangle side.

`findClosestNode` mesh2D

- The method `findClosestNode` of the `mesh2D` class returns the index (in the global numbering) of the closest node respect to the supplied *(x,y)* point:

```
Me = mesh2D(regions.rect());
Me.draw('node');
Me.findClosestNode([0.25 0.31])
    % ans = 60
```



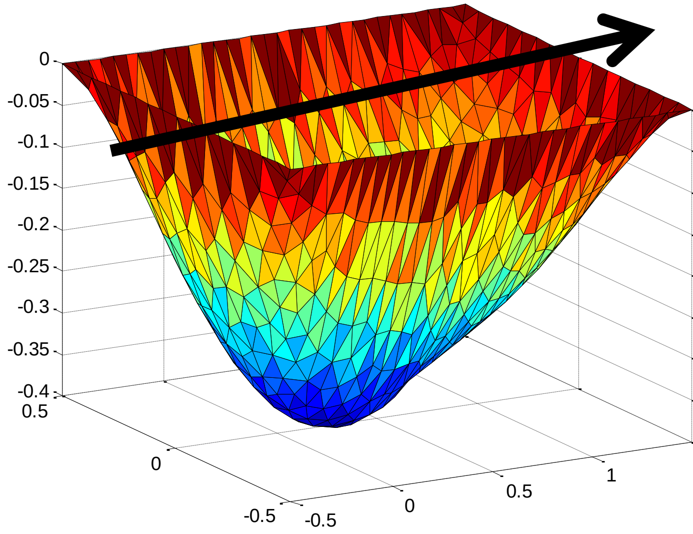
- Called with 2 output arguments it also returns the index of the triangle which contains *(x,y)*

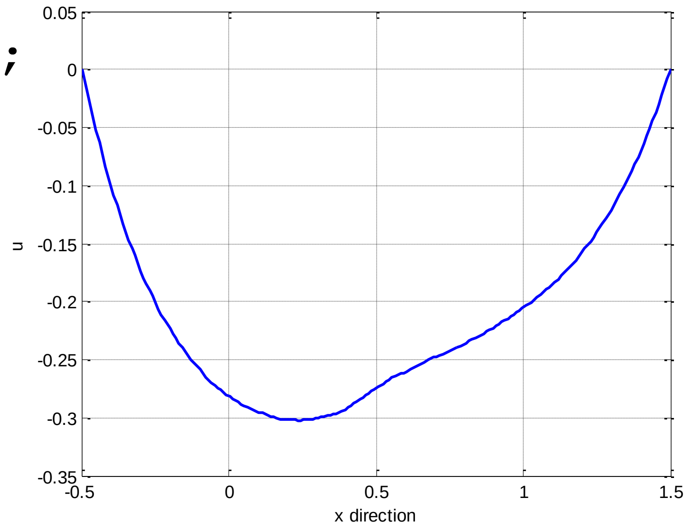
```
>>[node, triangle] = Find(Me,[0.25, 0.31]);
>>triangle
   136
```

Method `Interp`

- It evaluates the solution on the specified points, linearly interpolating the solutions obtained in the vertices of the triangles containing the points

```
N=200;
ptx=linspace(-0.5,1.5,N).';
pty=zeros(N,1);
%200 pts like (x,0)
z=Me.interpolate(uu,[ptx,pty]);
figure;
plot(ptx,z);
```

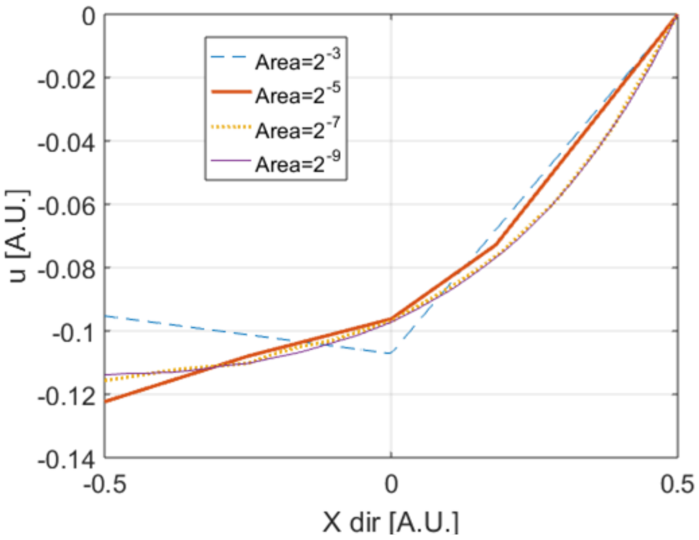


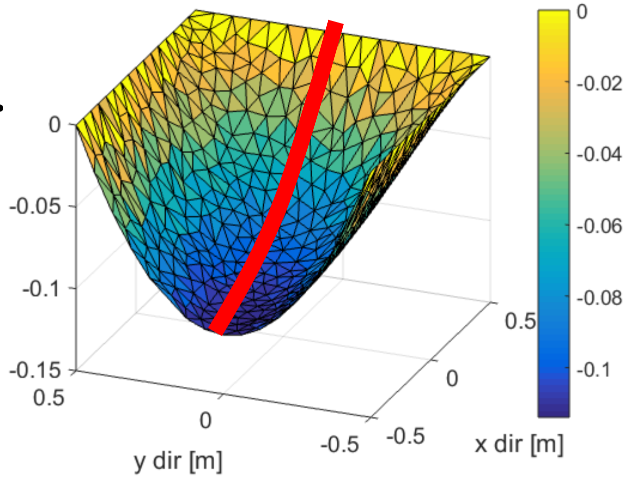


Method `interpolate`

- It is usefull, for instance, to compare solutions obtained with different mesh sizes

```
S=regions.rect('mu',1);
S.Borders(1).Bc(1)=
   boundaries.neumann(0);
Areas=2.^-(3:2:9);
y=zeros(200,1);
x=linspace(-0.5,0.5,200).';
figure; hold on;
for kl=1:length(Areas)
    Me=mesh2D(S,Areas(kl));
    [A,b]=dirichletHomo_BuildStiff (...
         Me,@(x,y)-ones(size(x)));
    uu=Me.copyToAllNodes(A\b);
    z=Me.interpolate(uu,[x,y]);
    plot(x,z);
end
```





Transport and reaction terms

- In order to include transport and reaction terms, the function becomes, for instance

  ```
  function [D,T,R,b] =
  dirichletHomo_DiffTransReact_BuildStiff (Me,f)
  ```

  and where we previously built matrix D as

  ```
  d(pos)=c(e)*(Dy(ni)*Dy(nj)+Dx(ni)*Dx(nj))/(4.0*Area);
  ```

  we have to calculate

  ```
  t(pos)=(-beta(e,1)*Dy(nj)+beta(e,2)*Dx(nj))*1/6;
  d(pos)=c(e)*(Dy(ni)*Dy(nj)+Dx(ni)*Dx(nj))/(4.0*Area);
  r(pos)=a(e)*Area*(1+(ni==nj))/12;
  ```

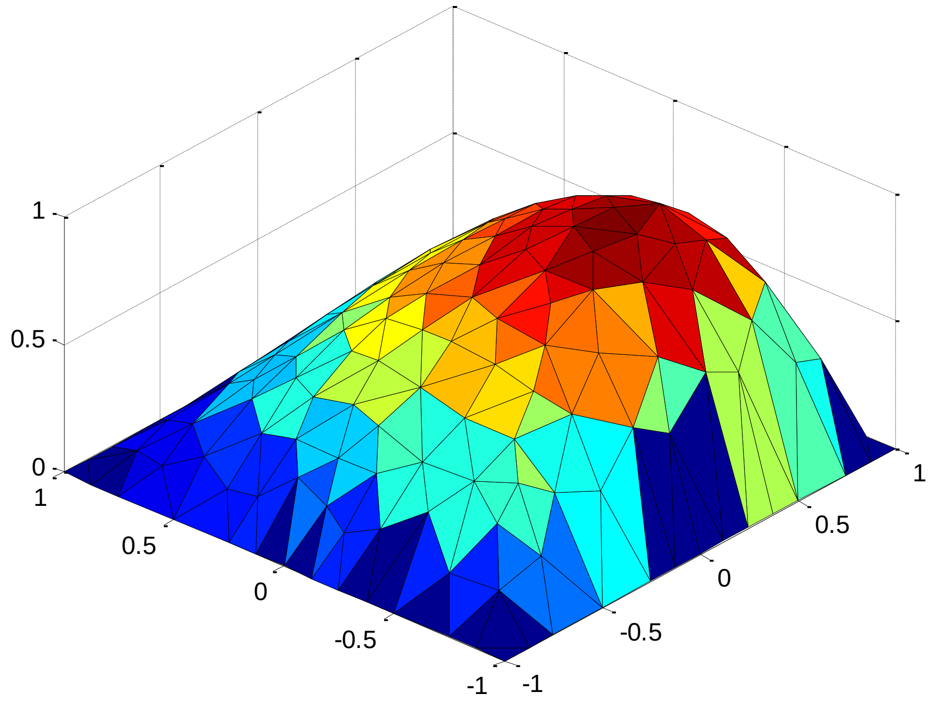
  and then sum the three contribution

Transport-diffusion term

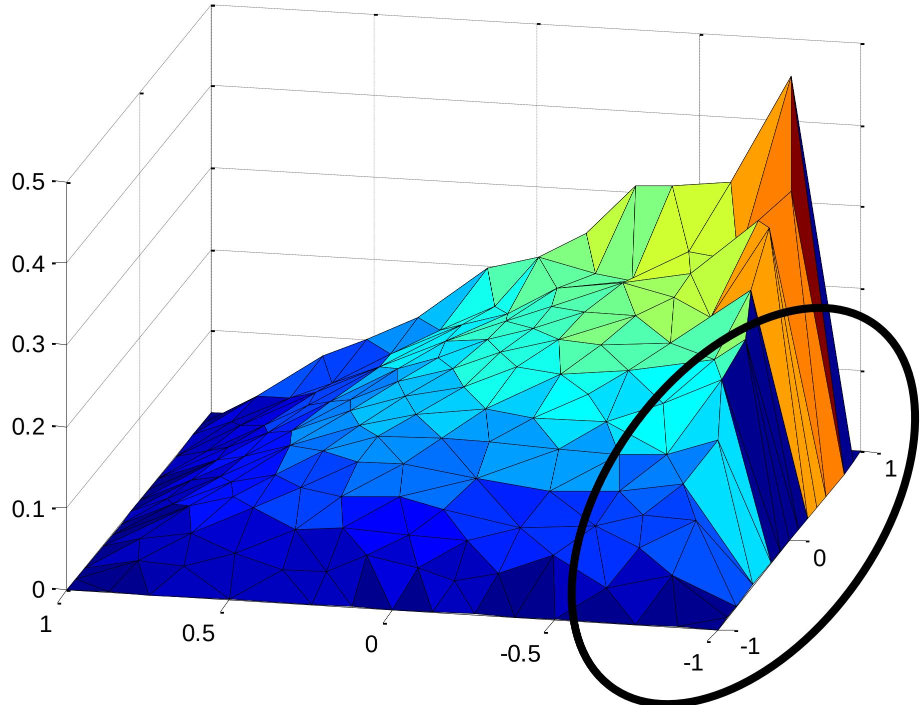
```
>>[D,T,R,b] =
dirichletHomo_DiffTransReact_BuildStiff (Me,f);
>>u = (D+T)\b;
```

What happens if we change Beta?

`Beta=[1, 1]`



`Beta=[30, 30]`



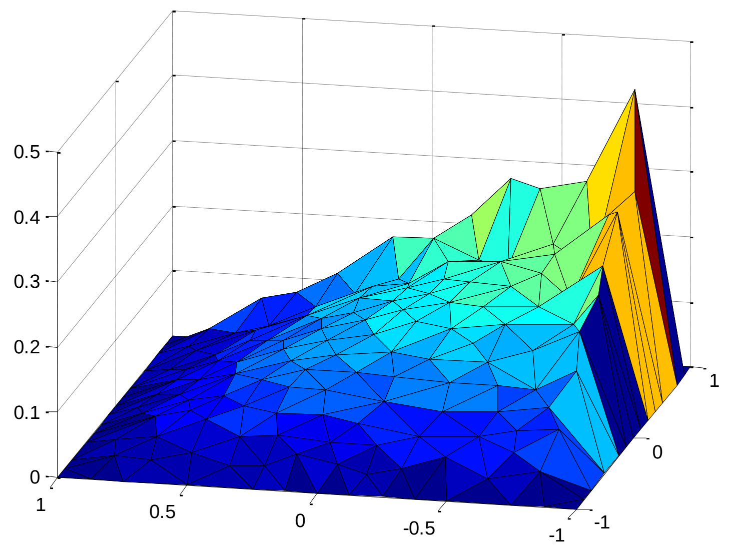
Transport-diffusion term

- Let $P_{e,k} = |\beta| h_k / 6\mu$ be the Péclet number local to each triangle k, with h_k length of the longest side
- It can be shown that if P_e >1 the solution becomes oscillating when we approach the boundary layer, whose dimension is approximately $\mu / |\beta|$
- To avoid this problem we can refine the mesh near the boundary layer inserting regions with lower values of the `MeshMaxArea` property

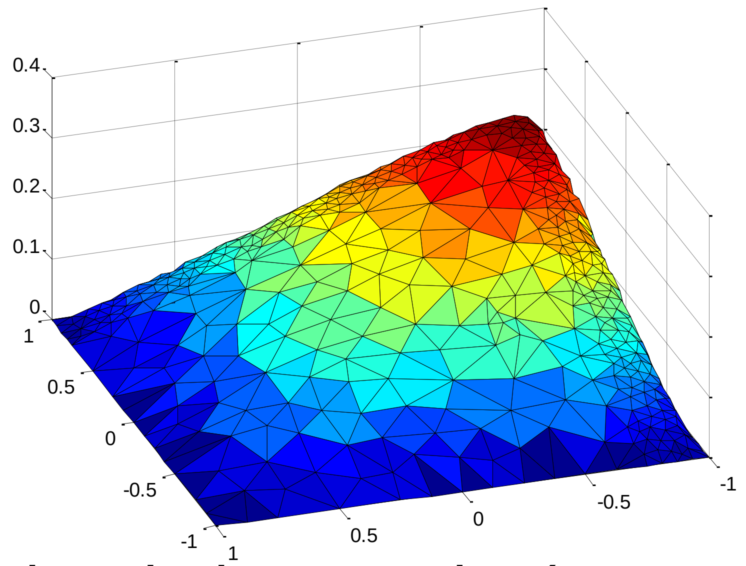
Transport-diffusion term

Beta=[30, 30]

Uniform mesh



Refined mesh



- Other much mode sophisticated techniques exist (GLS, SUPG, DW,…)

Non-homogeneous Dirichlet conditions

- Let’s consider the system $Au=b$, with A built on all the nodes (and not only the internal ones): `size(A)=length(Me.Nodes.X)`
- Let u_2 and u_4 be nodes with Dirichlet boundary conditions and u_1, u_3 internal nods

$$\begin{pmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{pmatrix} \begin{pmatrix} u_1 \\ \boxed{u_2} \\ u_3 \\ \boxed{u_4} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

Non-homogeneous Dirichlet conditions

- The system is equivalent to the following, where u_2 (known quantity) is moved to the right side of the equality

$$\begin{pmatrix} k_{11} & 0 & k_{13} & k_{14} \\ 0 & 1 & 0 & 0 \\ k_{31} & 0 & k_{33} & k_{34} \\ k_{41} & 0 & k_{43} & k_{44} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} b_1 - \boxed{k_{12}u_2} \\ u_2 \\ b_3 - \boxed{k_{32}u_2} \\ b_4 - \boxed{k_{42}u_2} \end{pmatrix}$$

Non-homogeneous Dirichlet conditions

- The same can be applied to u_4

$$\begin{pmatrix} k_{11} & 0 & k_{13} & 0 \\ 0 & 1 & 0 & 0 \\ k_{31} & 0 & k_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} b_1 - k_{12}u_2 - k_{14}u_4 \\ u_2 \\ b_3 - k_{32}u_2 - k_{34}u_4 \\ u_4 \end{pmatrix}$$

- and the system can be reduced to

$$\begin{pmatrix} k_{11} & k_{13} \\ k_{31} & k_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_3 \end{pmatrix} = \begin{pmatrix} b_1 - k_{12}u_2 - k_{14}u_4 \\ b_3 - k_{32}u_2 - k_{34}u_4 \end{pmatrix}$$

Non-homogeneous Dirichlet conditions

- The problem can be reduced to the following linear system, where the only unknown quantities refer to internal nodes

$$\begin{pmatrix} k_{11} & k_{13} \\ k_{31} & k_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_3 \end{pmatrix} = \begin{pmatrix} b_1 - k_{12}u_2 - k_{14}u_4 \\ b_3 - k_{32}u_2 - k_{34}u_4 \end{pmatrix}$$

- Summary: if node i is adjacent to a node j having non homogeneous Dirichlet BC, we have to evaluate the term k_{ij} and subtract the quantity $k_{ij}\, u_j$ to b_i

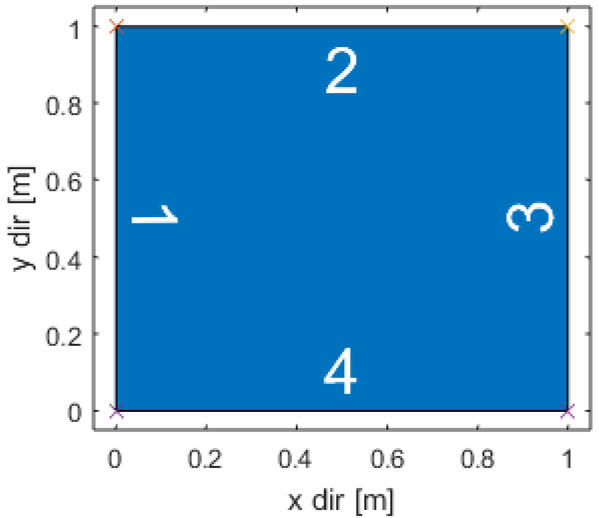
Non-homogeneous Dirichlet conditions: example

- Lets consider a squared geometry

  ```
  Sh=regions.rect('mu',1)+[0.5 0.5];
  Sh.draw('edge')
  ```

- We apply non homogeneous Dirichlet B.C.s on edges 2, 3, and 4:
  - The height of edge 3 is 1
  - On edges 2 and 4 the displacement is x
  - On edge 1 we have 0 displacement (homogeneous Dirichlet B.C.)
- We include diffusion effects only

```
Sh.Borders(1).Bc(3) =
     boundaries.dirichlet(1);
Sh.Borders(1).Bc([2,4]) =
     boundaries.dirichlet(@(x,y)x);
Me=mesh2D(Sh,1e-3);
```

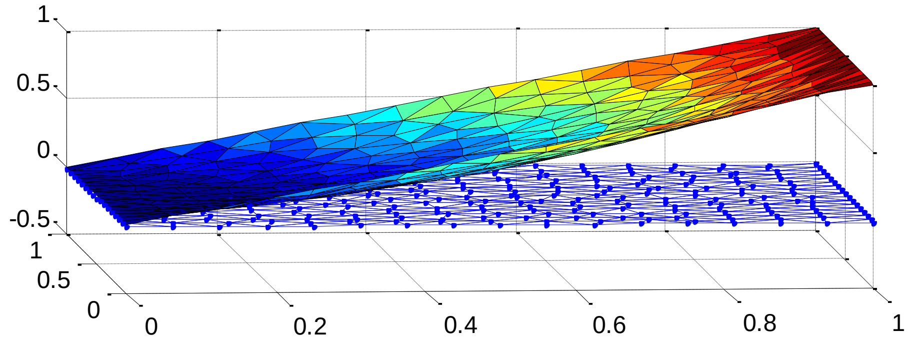


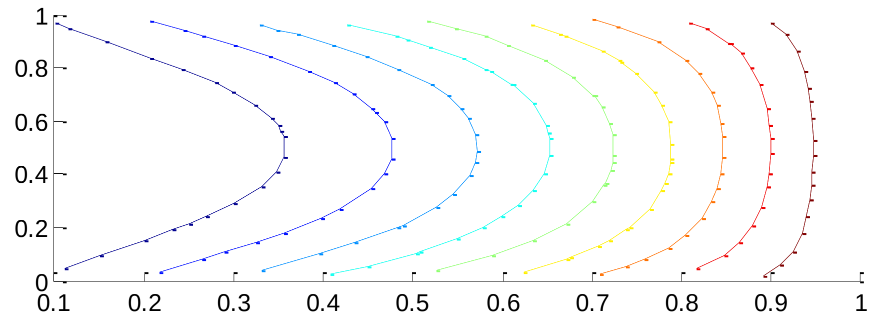
function [D,b] = dirichletNonHomo_BuildStiff(Me,f)

```
%in the main loop...
for ni = 1:3
   ii = Dof(V(e,ni));          %is it a dof?
   if ii > 0                   %yes! second loop on the vertices
      for nj = 1:3
         jj = Dof(V(e,nj));
         Dcom = (Dy(ni)*Dy(nj)+Dx(ni)*Dx(nj));
         dtmp = c(e)*DCom/(4.0*Areas(e));
         if jj > 0             %is it a dof as well?
            d(pos) = dtmp; row(pos) = ii;
            col(pos) = jj; pos = pos+1;
         else
            val = Me.BC.DirichletNodes(-jj,2);
            b(ii) = b(ii) - dtmp*val ;
          end
       end
      b(ii) = b(ii) + Areas(e)*force(e)/3.0;
   end
end
```

Program execution

```
[D,b]= dirichletNonHomo_BuildStiff(Me,f);
uu=Me.copyToAllNodes(D\b);
figure;
subplot(2,1,1);
Me.draw(uu,'offset');
subplot(2,1,2);
Me.draw(uu,'contour');
```

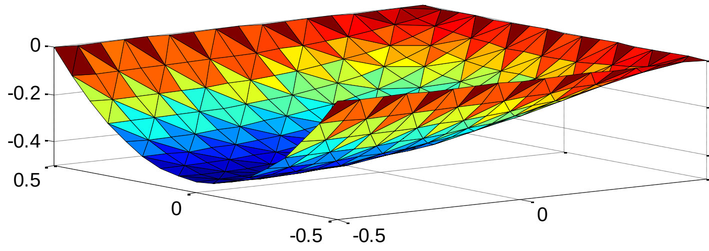


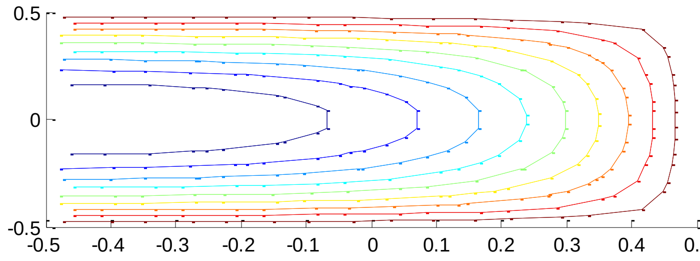


Homogeneous Neumann boundary conditions

- The code we already wrote is able to handle sides with homogeneous Neumann B.C.s

```
Sh = regions.rect('mu',1);
Sh.Borders(1).Bc(1) =
   boundaries.neumann(0);
Me = mesh2D(Sh,0.05);
f = @(x,y)-4*ones(size(x));
[D,b] =
   dirichletHomo_BuildStiff(Me,f);
u=D\b;
```





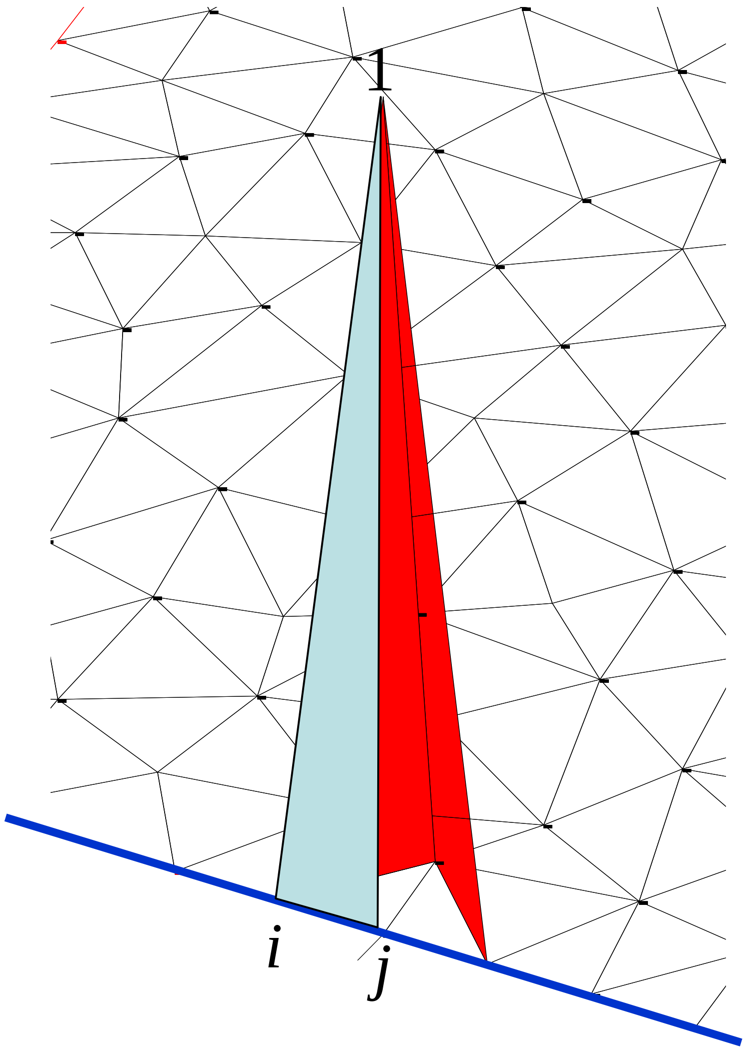
Non-homogeneous Neumann boundary conditions

- We consider the Poisson problem $\nabla \cdot (\mu \nabla u) = f$ with boundary conditions $\vec{n} \cdot (\mu \nabla u) = g$ on $\partial\Omega$
- This means that we impose that the solution has a derivative equal to g on the domain boundary $\partial\Omega$
- We project on an arbitrary test function v
$$\int_\Omega [\nabla \cdot (\mu \nabla u)] v \mathrm{d}\underline{x} = \int_\Omega f\, v \mathrm{d}\underline{x}$$
- and we apply the Green Theorem
$$\int_\Omega [\nabla \cdot (\mu \nabla u)] v \mathrm{d}\underline{x} = \int_\Omega \mu \nabla u \cdot \nabla v \,\mathrm{d}\underline{x} - \int_{\partial\Omega} g\, v \mathrm{ds} = \int_\Omega f\, v \mathrm{d}\underline{x}$$
- We need to evaluate the term $g \int_{\partial\Omega} \varphi_j \,\mathrm{d}s$

Non-homogeneous Neumann boundary conditions

- The term represents the area of a triangle with height 1 and a base equal to the distance between nodes *i* and *j*.
- Its it therefore

$$\frac{g}{2}\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$



```
function [A,b] =
neumannNonHomo_BuildStiff(Me,f)
```

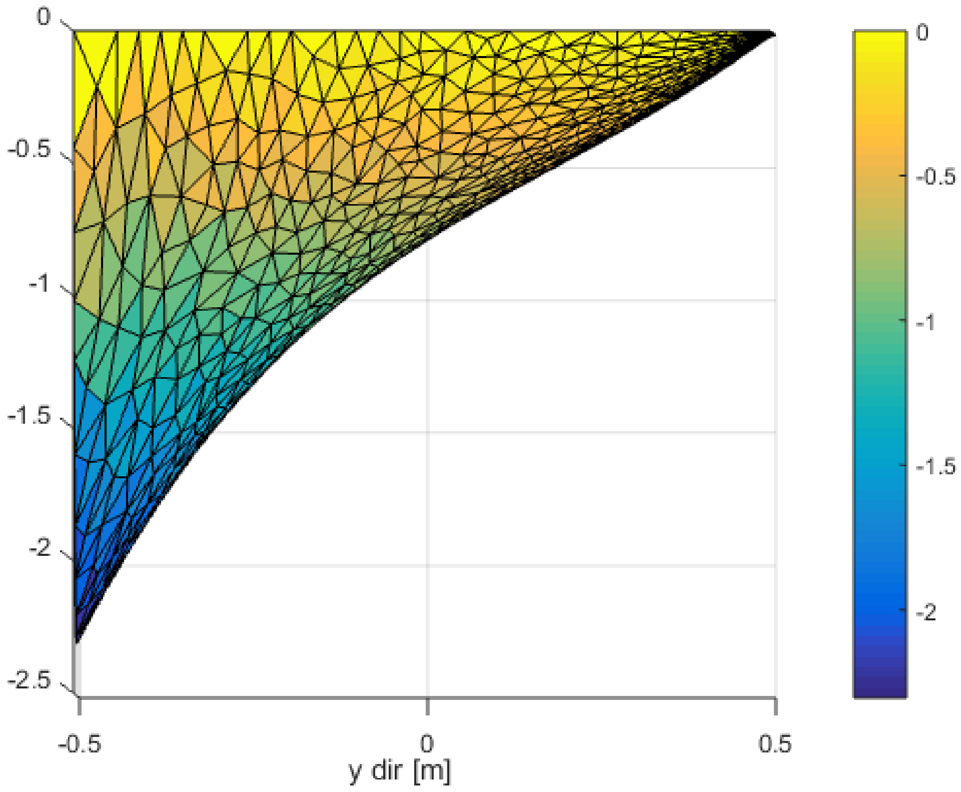
- The code is the same as the one we wrote for homogeneous Dirichlet boundary conditions
- After the loop over the triangles, we add to the constant terms vector the contribution of the non-homogeneous Neumann BC:
  - For each side with a Neumann condition
    1. Evaluate its length `L`
    2. Get the value `g`, imposed using `boundary.Neumann(g)` and stored in the second column of `Me.BC.NeumannEdges`
    3. For each vertex which is a dof, add `g/2*L` to the corresponding element in the constant terms vector.

    NB: the nodes of a Neumann edge can be: a) both dofs or b) 1 dof and 1 Dirichlet node, if the edge is connected to a Dirichlet B.C.

```
function [A,b] =
neumannNonHomo_BuildStiff(Me,f)
```

```
%after the loop over the triangles
Edges=Me.Edges;
Neumann=Me.BC.NeumannEdges;
for k=1:size(Neumann,1)  %loop over each Neumann edge
    Node1=Edges(Neumann(k),1);      %index of node 1
    Node2=Edges(Neumann(k),2);      %index of node 2
    dx=Nodes.X(Node1)-Nodes.X(Node2);
    dy=Nodes.Y(Node1)-Nodes.Y(Node2);
    L=sqrt(dx*dx+dy*dy);            %edge length
    g=Me.BC.NeumannEdges(k,2);      %Neumann condition
    if Dof(Node1)>0                 %if node 1 a dof?
        b(Dof(Node1))=b(Dof(Node1))+g/2*L;
    end
    if Dof(Node2)>0
        b(Dof(Node2))=b(Dof(Node2))+g/2*L;
    end
end
```

Non-homogeneous Neumann boundary conditions



```
Sh = regions.rect('mu',1);
Sh.Borders(1).Bc(4) = ...
   boundaries.neumann(-5);
Me = mesh2D(Sh,1e-3);
f = @(x,y)-4*ones(size(x));
[D,b] = neumannNonHomo_BuildStiff(Me,f);
uu = Me.copyToAllNodes(D\b);
figure; Me.draw(uu);
```

Numerical gradient evaluation

- The equation of a plane passing for in (x_j,y_j,z_j) can be obtained solving

$$\det\begin{pmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{pmatrix} = \det\begin{pmatrix} x-x_1 & y-y_1 & z-z_1 \\ Dx_3 & Dy_3 & Dz_3 \\ Dx_2 & Dy_2 & Dz_2 \end{pmatrix} 0$$

and therefore it is

$$\left(x-x_1\right)\left(Dy_3Dz_2-Dy_2Dz_3\right)-\left(y-y_1\right)\left(Dx_3Dz_2-Dx_2Dz_3\right)$$
$$+\left(z-z_1\right)\left(Dx_3Dy_2-Dx_2Dy_3\right)=0$$

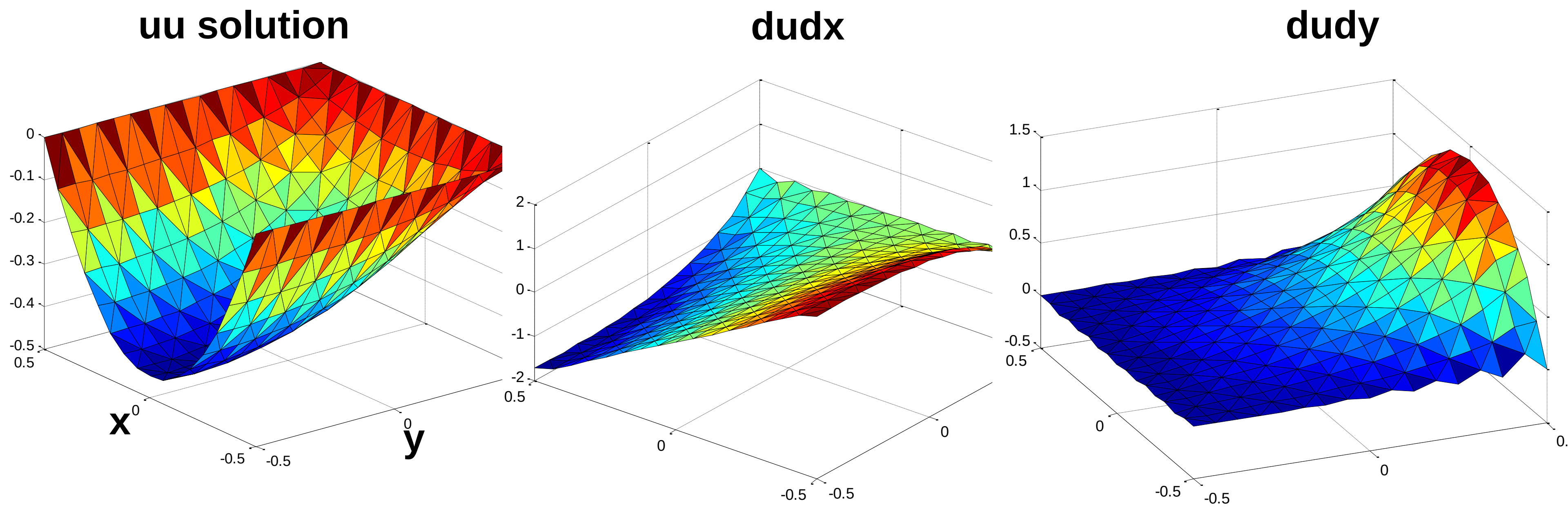
- We have then

$$\frac{\partial z}{\partial x}=\frac{Dy_3Dz_2-Dy_2Dz_3}{2Area};\ \frac{\partial z}{\partial y}=\frac{-Dx_3Dz_2+Dx_2Dz_3}{2Area}$$

Numerical gradient evaluation

- Method `gradient` returns the gradient of the solution

```
 [dudx,dudy]=Me.gradient(uu);
Me.draw(dudx);
Me.draw(dudy);
```



Example #1: Darcy law

- Darcy law describes the flow of a fluid in porous medium with permeability k_p and viscosity μ: $v(x,y) = -k_p \nabla p(x,y) / \mu$
- Starting from Navier-Stokes equation, under the hypothesis of uncompressible fluid, we obtain that the velocity field is solenoidal $\nabla \cdot v(x,y) = 0$
- By taking the divergence of Darcy law we obtain

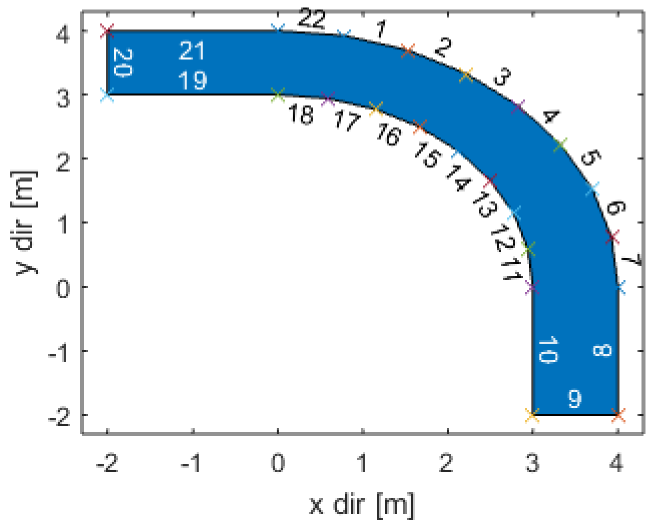
  $$\nabla \cdot \nabla p = \nabla^2 p = 0$$

  and therefore the pressure field is described by the Poisson equation
- We now calculate the pressure and velocity profiles on a generic domain

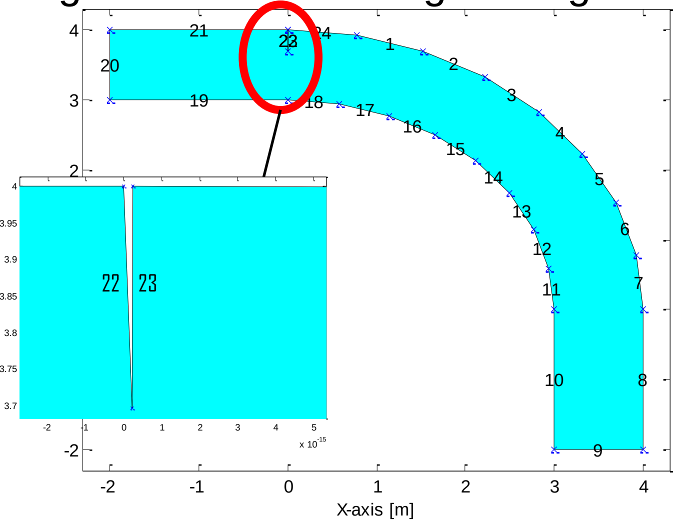
Example #1: 1 – domain definition

- We consider a curved pipe connected to two linear pipes
- Since we are working with curved regions, it’s a good practice to define a grid using the `grid2D` class

```
grid2D.setSteps(1e-6);
S=regions.rect([-1 3.5],[2 1])+...
regions.sector([0,0],4,3,[0 90],8)+...
regions.rect([3.5 -1],[ 1 2])
figure;S.draw('e');
```



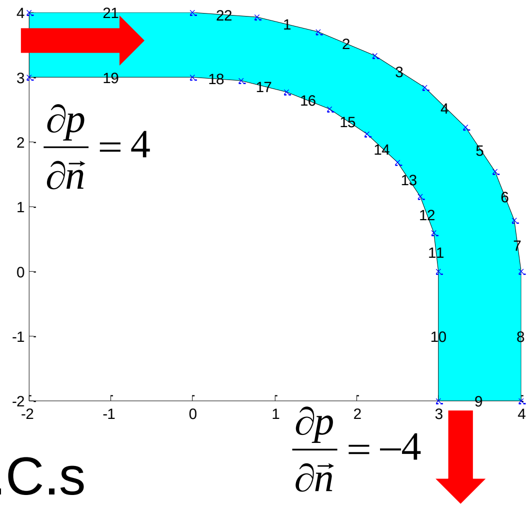
Without the grid, numerical errors introduced by the finite precision of the computer, could introduce extra, wrong, edges when combining the regions



Example #1: 2 – boundary conditions definition

- A simple approach is to assign the velocities in two terminal edges (= non homogeneous Neumann B.C.) and homogeneous Neumann conditions of the other edges.

```
S.Borders.Bc(:)  = boundaries.neumann(0);
S.Borders.Bc(9)  = boundaries.neumann(-4);
S.Borders.Bc(20)= boundaries.neumann(4);
```


$\frac{\partial p}{\partial \vec{n}} = 4$
$\frac{\partial p}{\partial \vec{n}} = -4$
21
22
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

- In this way the problem presents only Neumman B.C.s and presents therefore an infinite number of solutions, all with the same derivative (= velocity profile): if we want to obtain a correct solution for the pressure field we need to assign a Dirichlet B.C. in at least one point (or one edge):

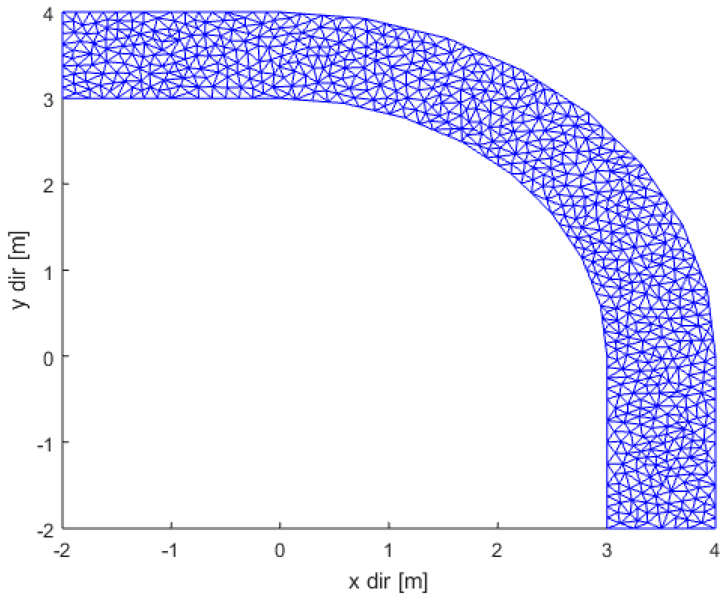
```
S.Borders.Bc(20) = boundaries.dirichlet(PressureIn);
```

Of course, we can also apply Dirichlet B.C.s on the two terminal edges, specifying the in and out pressures:

```
S.Borders.Bc(9) = boundaries.dirichlet(PressureOut);
S.Borders.Bc(20) = boundaries.dirichlet(PressureIn);
```

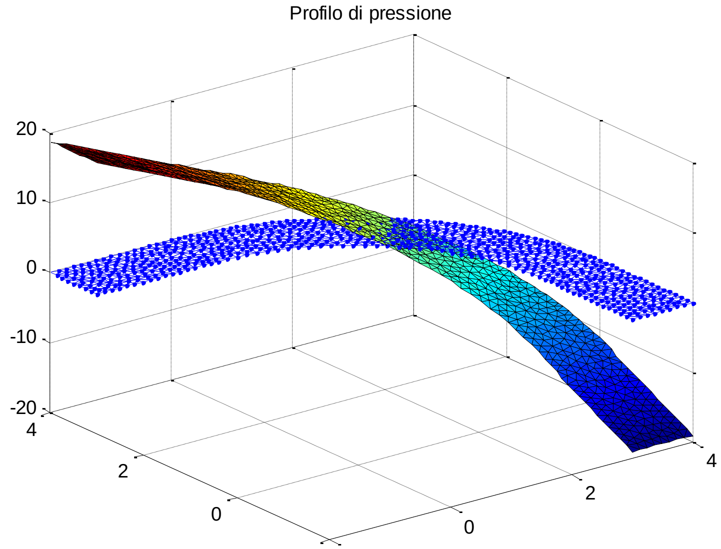
Example #1:
3 – Mesh, 4 – Stiffness matrix

```
%Mesh
Me = mesh2D(S,1e-2);
figure;
Me.draw();


%Stiffness matrix:
%in this case there is no
%external source, but we need
%to include the non Homogeneous
%Neumann B.C.s, and/or the
%non Homogeneous Dirichlet B.C.s.
[D,b] = BuildStiffnessDarcy(Me);
```



Example #1:
5 – Solution, 6 – plot

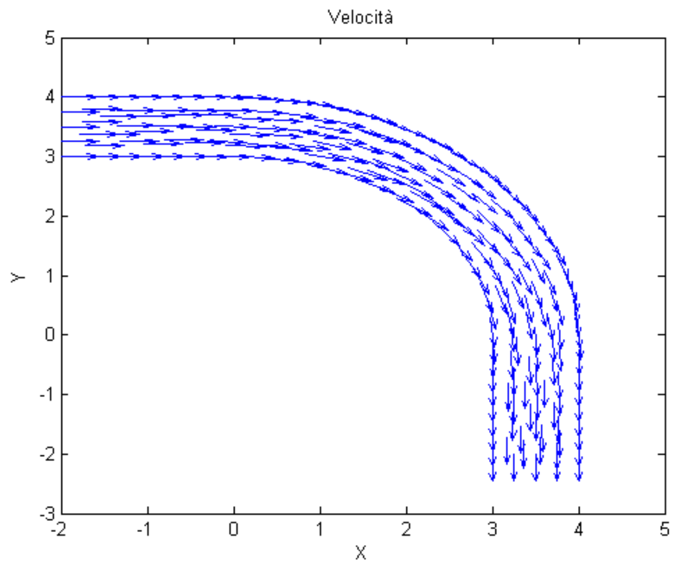
```
%Calculate and draw the pressure
uu = Me.copyToAllNodes(D\b);
figure;
Me.draw(uu);
```



```
% Calculate and draw the velocity
```

% $v(x,y) \propto -\nabla p(x,y) = \nabla[-p(x,y)]$

```
[ux,uy] = Me.gradient(-uu);
figure;
quiver(Me.Nodes.X, Me.Nodes.Y,...
    ux, uy);
```



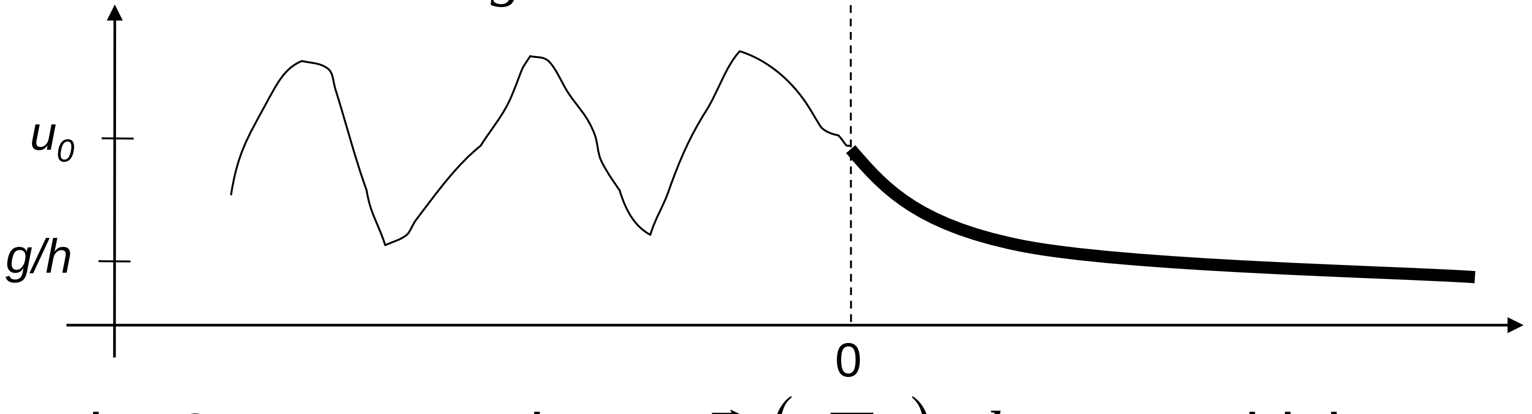
Numerical integral evaluation

- The integral can be easily evaluated using a formula which is the generalization of the trapezium method we used in the 2D case:
  - For each triangle *e* with vertices (i,j,k) we calculate the volume of the prism $A_e(u_i+u_j+u_k)/3$; we then add these partial volumes together
- The method `Me.integrate(uu)` calculates the numerical integral of the solution over the whole domain
- If more than one region was used, it is possible to evaluate the integral only on certain regions passing their indices as 3rd parameter

```
>>I=Me.integrate(uu, [1 4]);
  %it evaluates the integral over the regions 1&4
```

Robin boundary conditions

- In the 1D case, Robin Boundary conditions correspond to impose that the solution, at the end of the interval, respects the condition $\mathrm{d}u/\mathrm{d}x + hu = g$
- The solution is $u(x) = [u(0) - g/h]\exp(-hx) + g/h$
- In other words, we require the solution to exponentially tend to the value g/h



- In the 2D case we have $\vec{n}\cdot(\mu\nabla u) + hu = g$ which can also be written as $\vec{n}\cdot(\mu\nabla u) = h(u_\infty - u)$ with $u_\infty = g/h$

Robin boundary conditions

- Lets consider the Poisson equation $\nabla \cdot (\mu \nabla u) = f$ with BC $\vec{n} \cdot (\mu \nabla u) + hu = g$ on $\partial\Omega$
- This means that we impose that the solution trends to the value g with an exponential behaviour
- We project on an arbitrary test function v
$$\int_\Omega [\nabla \cdot (\mu \nabla u)] v \mathrm{d}\underline{x} = \int_\Omega f\, v \mathrm{d}\underline{x}$$
- We apply the Green theorem
$$\int_\Omega [\nabla \cdot (\mu \nabla u)] v \mathrm{d}\underline{x} = \int_\Omega \mu \nabla u \cdot \nabla v \,\mathrm{d}\underline{x} - \int_{\partial\Omega} g\, v \mathrm{ds} + \int_{\partial\Omega} hu\, v \mathrm{ds} = \int_\Omega f\, v \mathrm{d}\underline{x}$$
- We already evaluated (non-homogeneous Neumann BC) the term related to $g \int_{\partial\Omega} \varphi_j \,\mathrm{d}s$
- We have to evaluate numerically $\int_{\partial\Omega} huv \,\mathrm{d}s$

Robin boundary conditions

- Lets then evaluate $\int_{\partial\Omega} h\varphi_i\varphi_j \mathrm{d}s$
- We remember that the integration line is the base of a triangle lying on a plane which is othogonal to the xy plane; we call d the length of such a base and we evaluate the integral by placing $\mathrm{d}s$ parallel to the base itself

$$\int_{\partial\Omega} h\varphi_i\varphi_j \mathrm{d}s = \begin{cases} \int_0^d \frac{x'}{d}\left(1-\frac{x'}{d}\right)\mathrm{d}x' = \frac{d}{6} \ se\, i \neq j \\ \int_0^d \left(\frac{x'}{d}\right)^2 \mathrm{d}x' = \frac{d}{3} \ se\, i = j \end{cases}$$

- These contributions create a matrix having the same size as the stiffness matrix

Robin boundary conditions

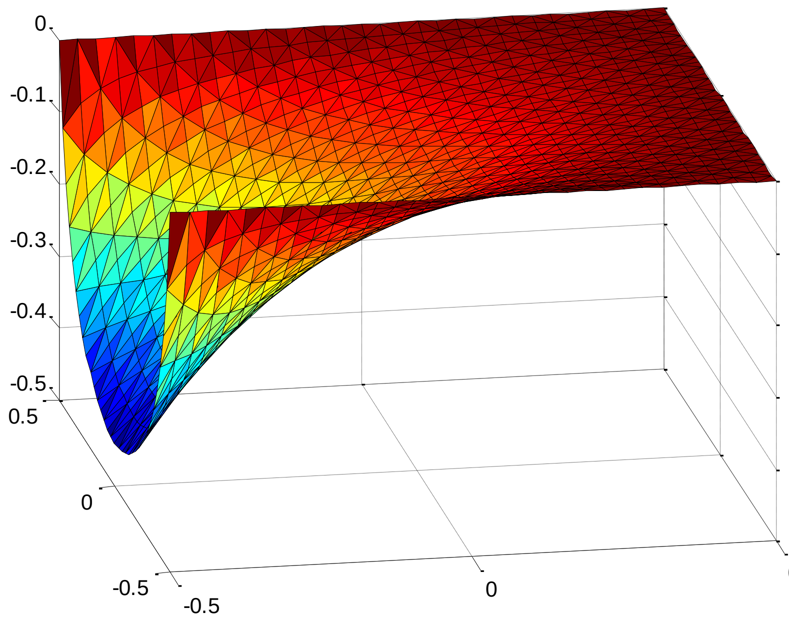
- When we encounter Robin BC, we need therefore to evaluate some contributions on the dofs lying on the domain border
- If the side *l* presents Robin BC, and if the node `ii1` on such a side is a dof, we need to evaluate
  - A contribution to the constant terms vector
    - `b(ii1)=b(ii1)+g/2*dist;`
  - A contribution on the main diagonal of the stiffness matrix
    - `D(ii1,ii1)=D(ii1,ii1)+h*dist/3;`
  - (If both the side's nodes are dof) the mutual contributions outside the main diagonal of the stiffness matrix
    - `D(ii1,ii2)=D(ii1,ii2)+h*dist/6;`

```
function [D,b] = robin_BuildStiff(Me,f)
```

```
%after the loop over the triangles
Robin=Me.BC.RobinEdges;
for k=1:size(Robin,1)
    Node1 = Edges(Robin(k,1),1); Node2 = Edges(Robin(k,1),2);
    dx = Nodes.X(Node1) - Nodes.X(Node2);
    dy = Nodes.Y(Node1) - Nodes.Y(Node2);
    L = sqrt(dx*dx+dy*dy);   %edge length
    ii1 = Dof(Node1); ii2 = Dof(Node2);
    g = Robin(k,3);   h = Robin(k,2);
    if ii1>0 && ii2<0 %ii1 is dof, ii2 is known
        b(ii1) = b(ii1)+g/2*L; row(pos) = ii1; col(pos) = ii1;
        d(pos) = h*L/3; pos = pos+1;
    elseif ii1<0 && ii2>0 %ii1 is known, ii2 is dof
        b(ii2) = b(ii2)+g/2*L; row(pos) = ii2; col(pos) = ii2;
        d(pos) = h*L/3; pos = pos+1;
    else %both are dof: 4 contributions to D
        b(ii1) = b(ii1)+g/2*L;     b(ii2) = b(ii2)+g/2*L;
        row(pos:pos+3)=[ii1;ii2;ii1;ii2];
        col(pos:pos+3)=[ii1;ii2;ii2;ii1];
        d(pos:pos+3)=[2;2;1;1]*h*L/6;
        pos=pos+4;
    end, end
```

```
function [A,b] =
BuildStiffnessDiffRobin (Me,f)
```

```
S = regions.rect('mu',1);
%the solution exponentially goes to the value g/h
S.Borders(1).Bc(1) = boundaries.robin(2,-2); %h=2,g=-2
Me=mesh2D(Sh,0.03);
[D,b]= robin_BuildStiff(Me,@(x,y)zeros(size(x)));
uu = Me.copyToAllNodes(D\b);
figure;
Me.draw(uu);
```



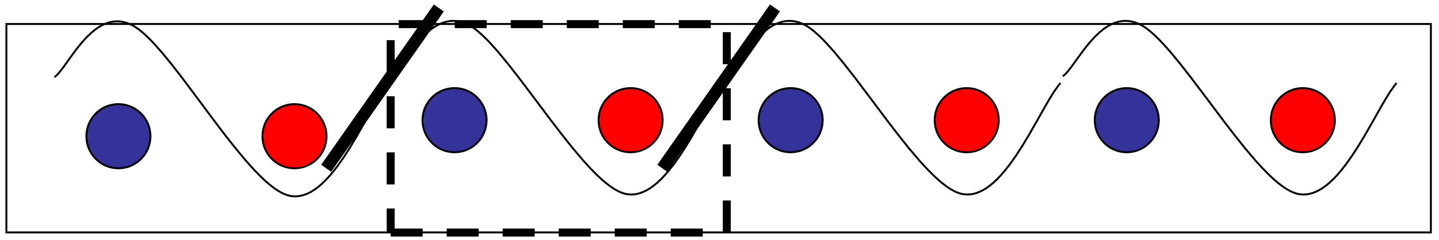
Robin and non-homogen. Dirichlet conditions

- The previous formulation is correct for all the sides with a Robin condition, and even for the sides that “touch” edges with homogeneous Dirichlet conditions.
- If, on the contrary, an edge with Robin BC has a vertex with a NON-homogeneous Dirichlet condition, we need to add a contribution also to the constant terms vector (for simplicity D is considered here as full matrix)

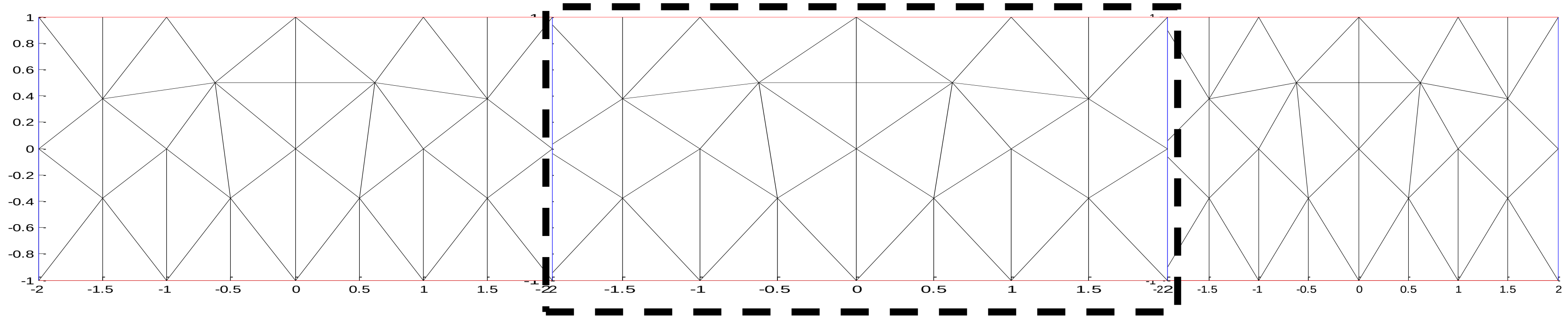
```
...
if ii1>0 && ii2<0 %ii1 dof, ii2 known
    Dirichlet=Me.BC.DirichletNodes(-ii2,2);
    b(ii1) = b(ii1)+g/2*dist-h*dist/6*Dirichlet;
    D(ii1,ii1) = D(ii1,ii1)+h*dist/3;
elseif ii1<0 && ii2>0 %ii1 known, ii2 dof
    Dirichlet=Me.BC.DirichletNodes(-ii1,2);
    b(ii2) = b(ii2)+g/2*dist-h*dist/6*Dirichlet;
    D(ii2,ii2)=D(ii2,ii2)+h*dist/3;
else  %Then ii1 and ii2 are both dof
...
```

Periodic boundary conditions

- Lets consider the elementary cell of a periodic structure



- We cannot apply the techniques we have encountered up to now: we only know that the solution is the same at the opposite edges



Periodic boundary conditions

- In the previous slide the periodicity is only in the *x* – direction and, if the domain is centred around the origin, from a generic *(x,y)* point placed on the left (right) border we can easily obtain corresponding “twin” node on the right (left) border, since its coordinates are *(-x,y)*
- When we define the BC we write then

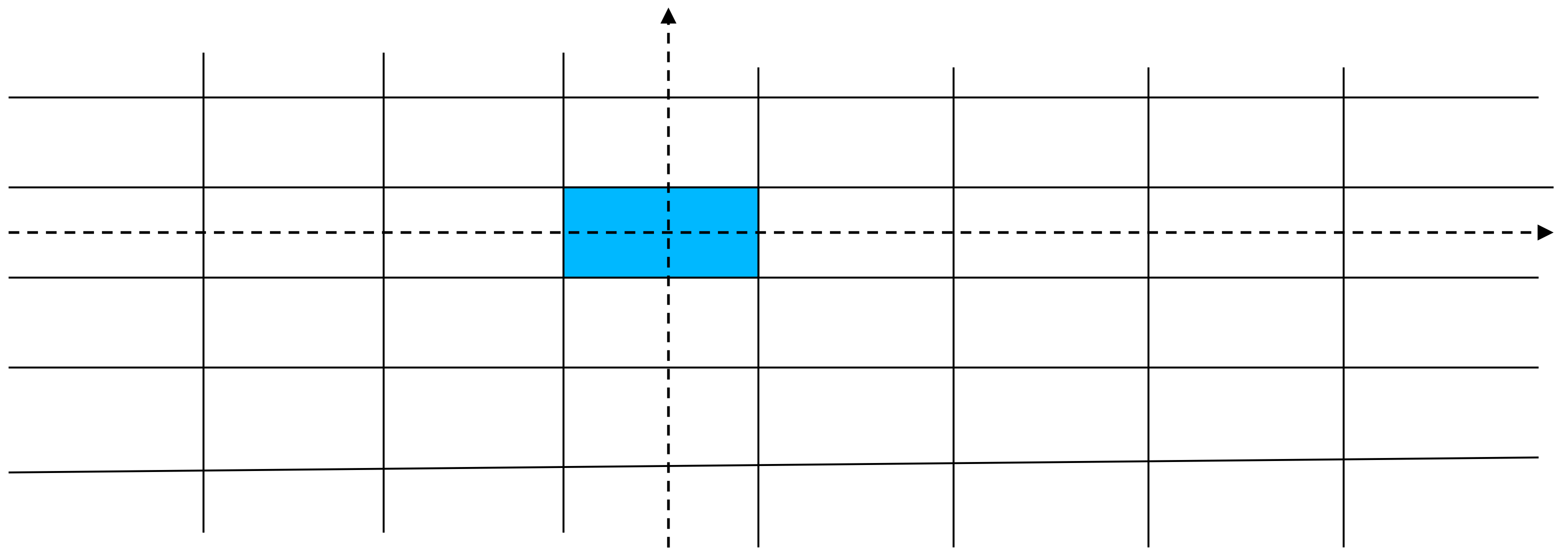
```
S.Borders(1).Bc(1)=boundaries.periodic(@(x,y)[-x,y]);
S.Borders(1).Bc(3)=boundaries.periodic(@(x,y)[-x,y]);
```

to indicate that the twin nodes can be obtained by getting the *(–x, y)* coordinates of the original node

Periodic boundary conditions

- If we have a grid periodic in both x and y directions, we use

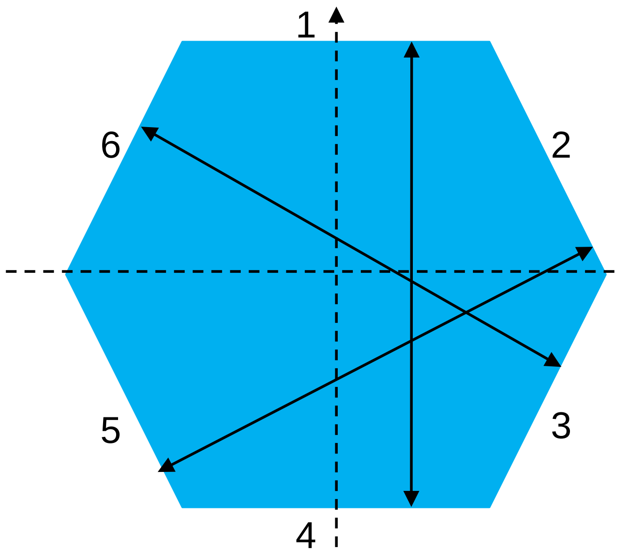
```
S.Borders.Bc(1)=boundaries.periodic(@(x,y) [-x,y]);
S.Borders.Bc(3)=boundaries.periodic(@(x,y) [-x,y]);
S.Borders.Bc(2)=boundaries.periodic(@(x,y) [x,-y]);
S.Borders.Bc(4)=boundaries.periodic(@(x,y) [x,-y]);
```

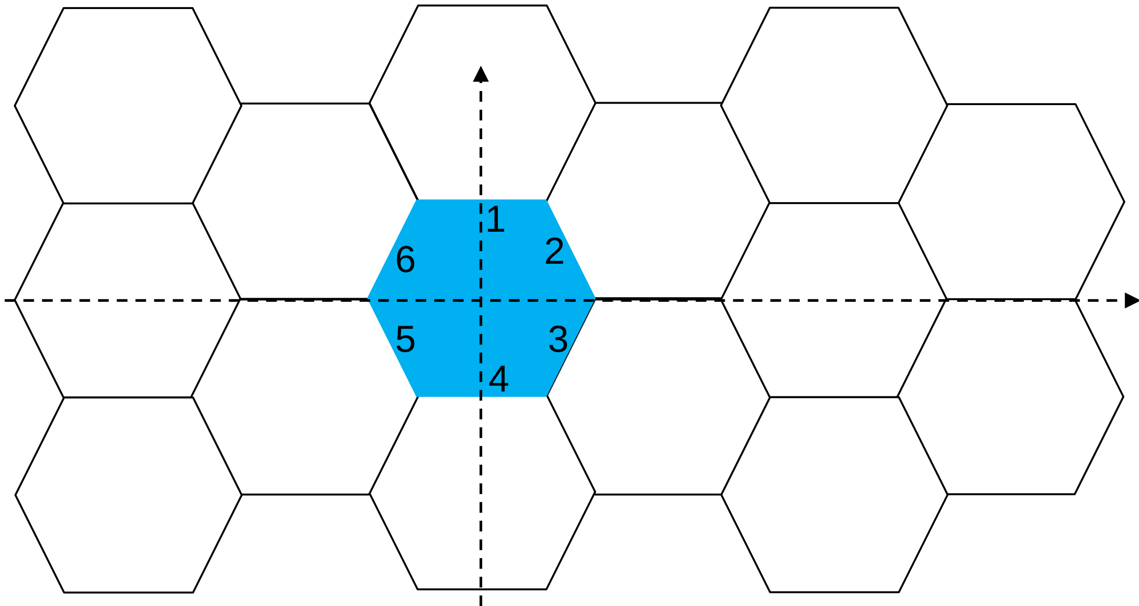


Periodic boundary conditions

- If we have an hexagonal fundamental cell, we use

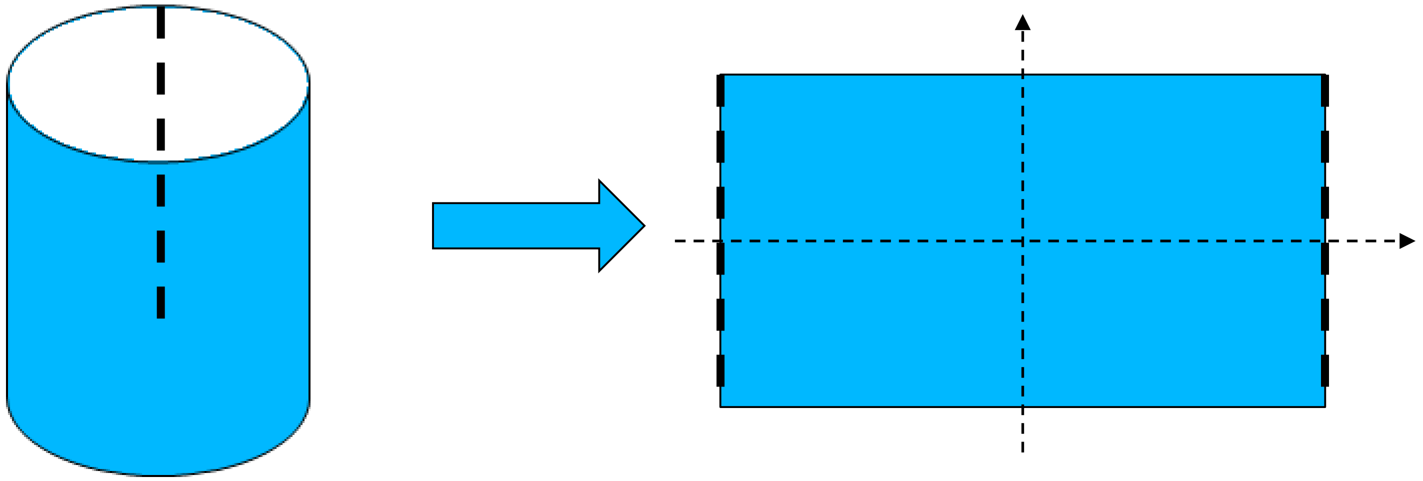
```
f25=@(x,y)0.5*[-x-sqrt(3)*y, -sqrt(3)*x+y];
f36=@(x,y)0.5*[-x+sqrt(3)*y,  sqrt(3)*x+y];
S.Borders(1).Bc(1) = boundary.periodic(@(x,y)[x,-y]);
S.Borders(1).Bc(4) = boundary.periodic(@(x,y)[x,-y]);
S.Borders(1).Bc(2) = boundary.periodic(f25);
S.Borders(1).Bc(5) = boundary.periodic(f25);
S.Borders(1).Bc(2) = boundary.periodic(f36);
S.Borders(1).Bc(4) = boundary.periodic(f36);
```





Periodic boundary conditions

- We can also use periodic boundary conditions to study problems whose domain lies on the surface of a cylinder



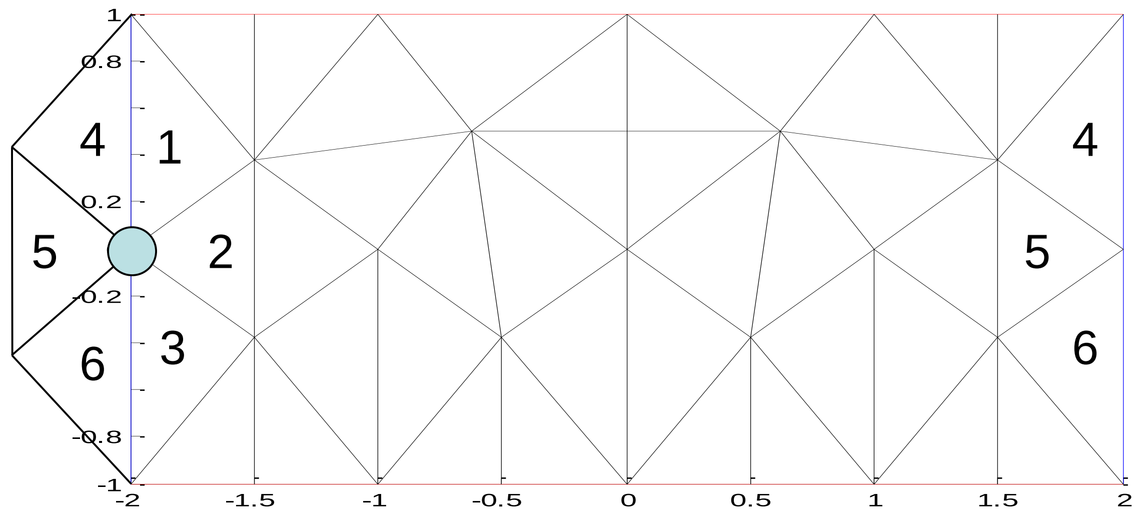
```
S.Borders(1).Bc([1,4]) = ...
    boundaries.periodic(@(x,y)[x,-y]);
```

Periodic boundary conditions

- The object `Me` contains the fields `Me.Nodes.TwinNode`, vector whose length is equal to the total number of nodes
- For each node k,
  - if `Me.Nodes.TwinNode(k)=0`, then the node k does not belong to an edge with periodic conditions (so there is nothing to do)
  - otherwise, `Me.Nodes.TwinNode(k)` contains the index of the twin node of the node k in the dof numbering. We'll have to modify the stiffness matrix.

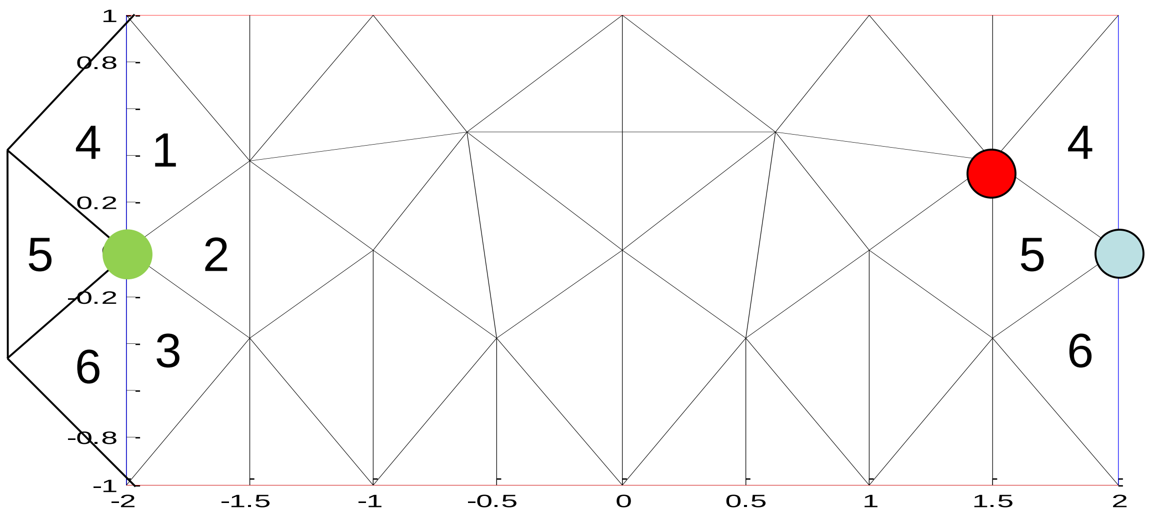
Periodic boundary conditions

- Lets study a structure periodic in only one direction
- When we build the stiffness matrix and we consider the node ○ , on the diagonal we find the contribution of triangles 1, 2, 3, but if we consider the periodic structure, also triangles 4,5, 6 bring a contribution



Periodic boundary conditions

- In a similar way, the node ● gives a contribution both on the node ● and on its twin ● on the other side



- The field `Me.PeriodicNodes` allows to easily obtain the index of node ● given the node ● (and vice versa)

Periodic boundary conditions

- In the loop used to build the stiffness matrix we have

```
if jj > 0 %add contribution to the stiffness matrix
   Dtmp = Dy(ni)*Dy(nj)+Dx(ni)*Dx(nj)
   d(pos) = mu(e)*Dtmp/(4.0*Areas(e));
   row(pos) = ii; col(pos) = jj; pos = pos+1;
   gem=Me.Nodes.TwinNode(V(e,ni));%any twin node?
   if gem > 0  %yes! add to the twin node too
      row(pos) = gem;
      if nj~=ni, col(pos)=jj; else, col(pos)=gem; end
      d(pos) = mu(e)*Dtmp/(4.0*Areas(e)); pos = pos+1;
   end
end
```

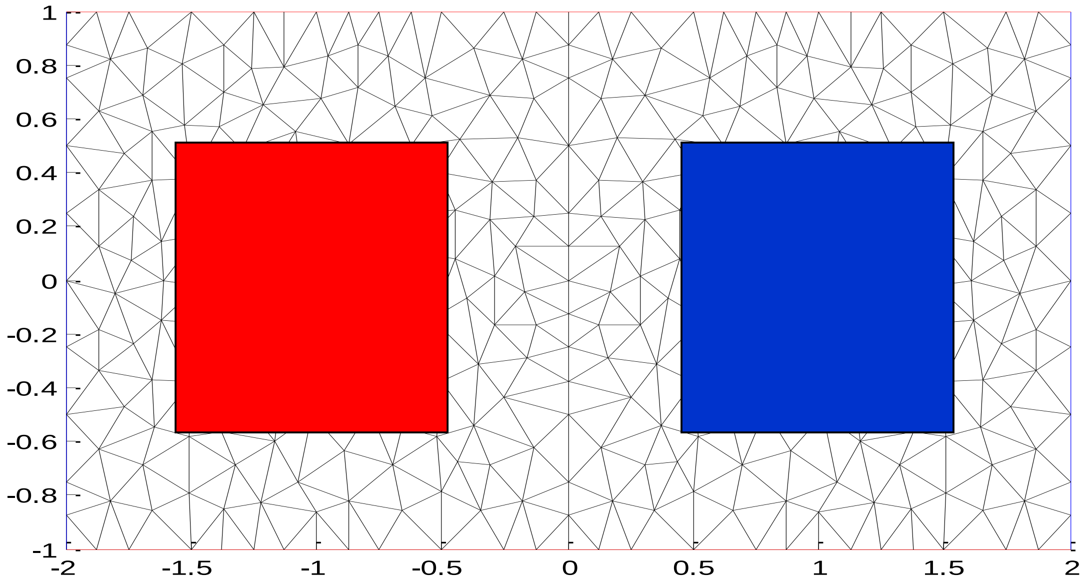
- NB!!! The same logic applies when an external force is present, and for non homogeneous Dirichlet, non homogeneous Neumann, Robin B.C.s

Periodic boundary conditions

- We consider a diffusion problem with

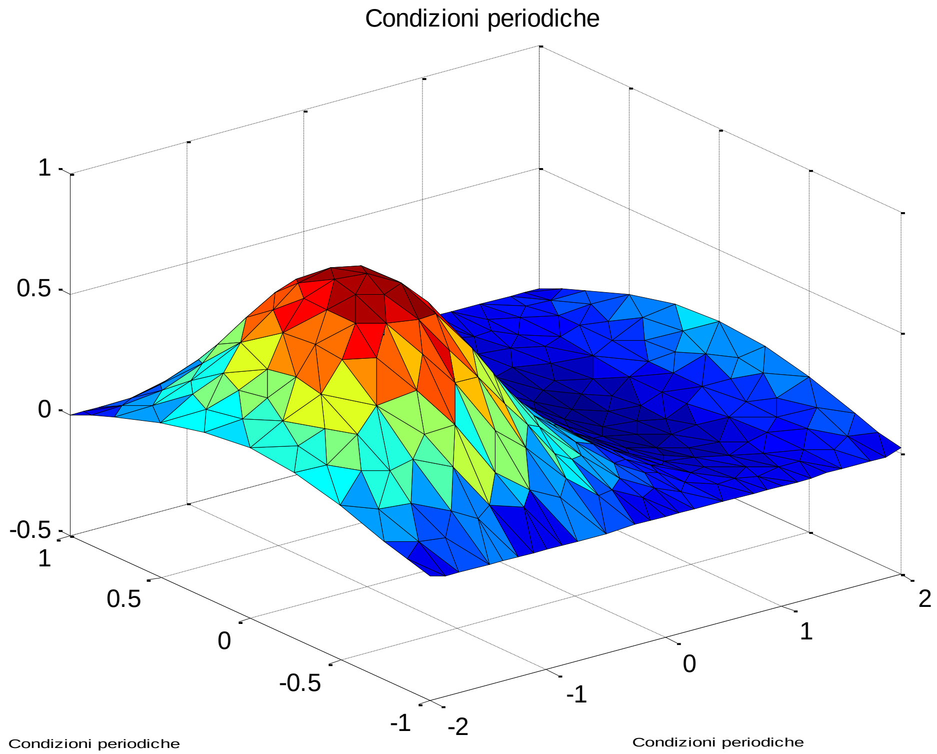
```
f = @(x,y)(4.*(x<-.5).*(x>-1.5).*(abs(y)<.5)...
    -1.*(x>.5).*(x<1.5).*(abs(y)<.5));
```

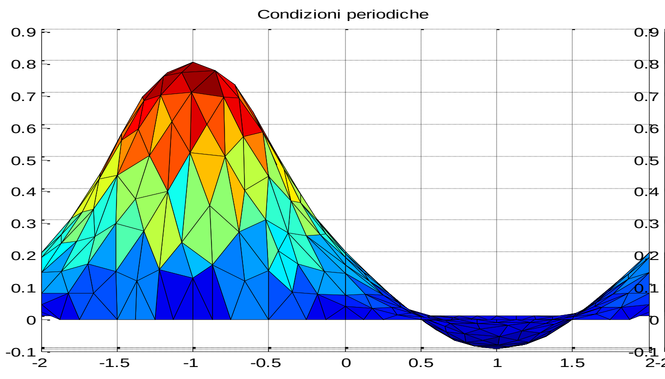
which returns 4 in the red regions and -1 in the blue one

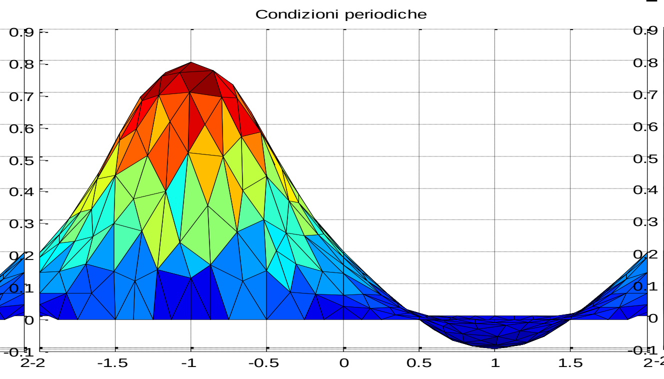


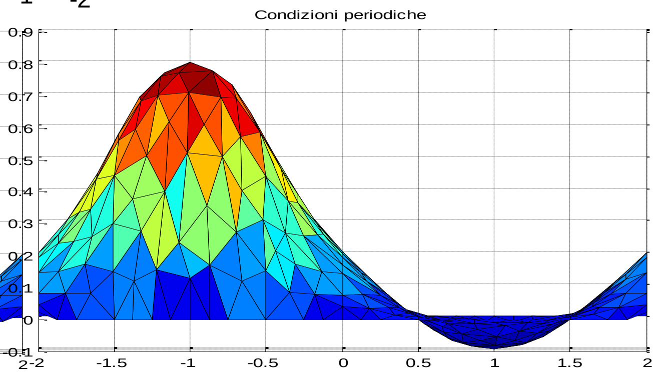
Periodic boundary conditions

- The solution is really periodic!



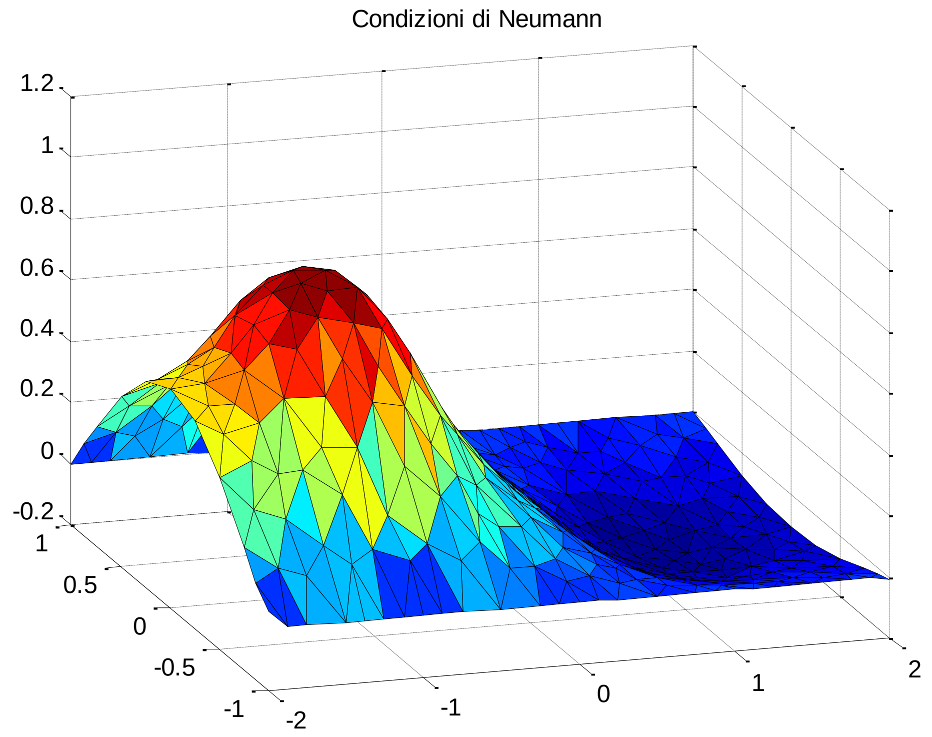


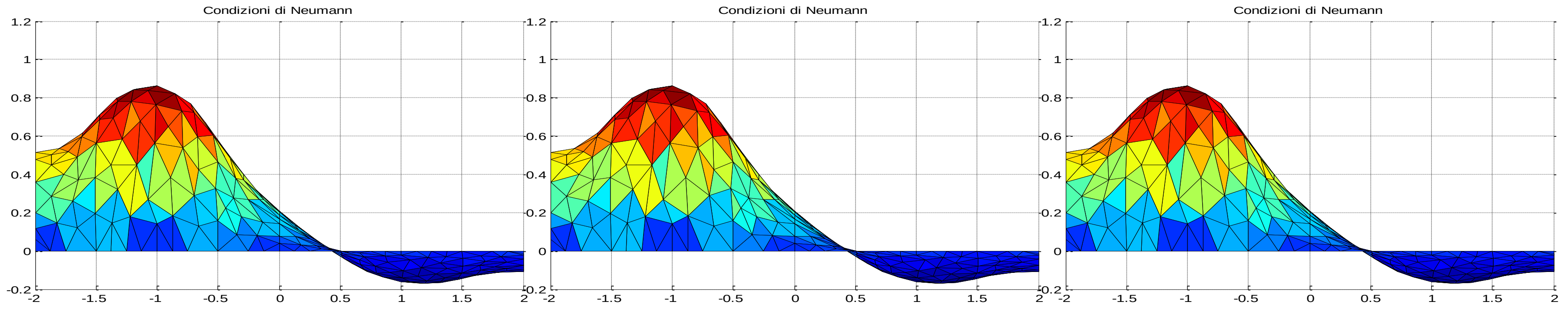




Periodic boundary conditions

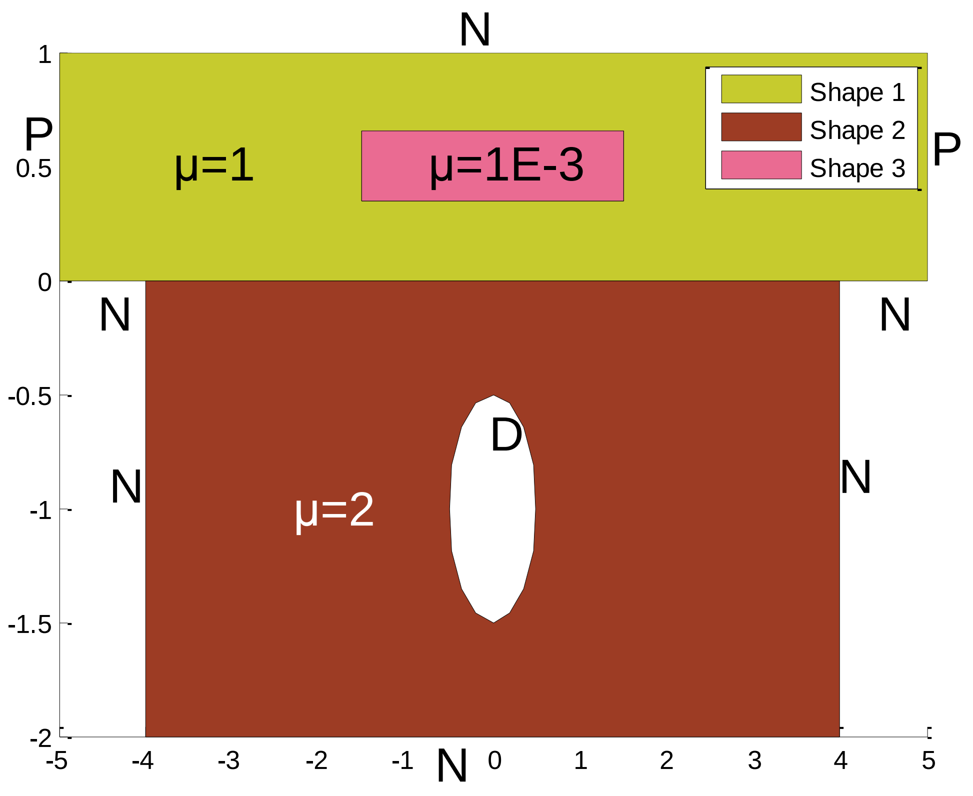
- By imposing Neumann conditions we have no continuity at the borders anymore
- It is possible to apply Neumann boundary conditions only if the phenomenon extinguishes at the borders





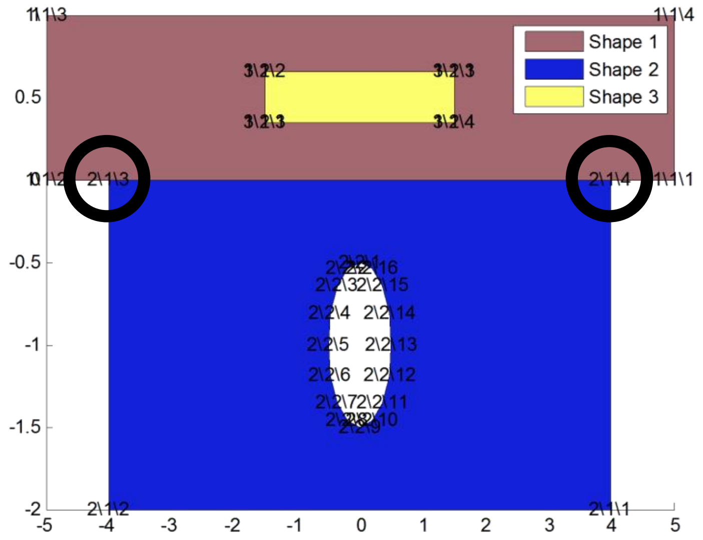
Example #2

- We consider a T-like structure, made by 3 regions, each one having a different elastic coefficient
- Boundary conditions are indicated in the figure



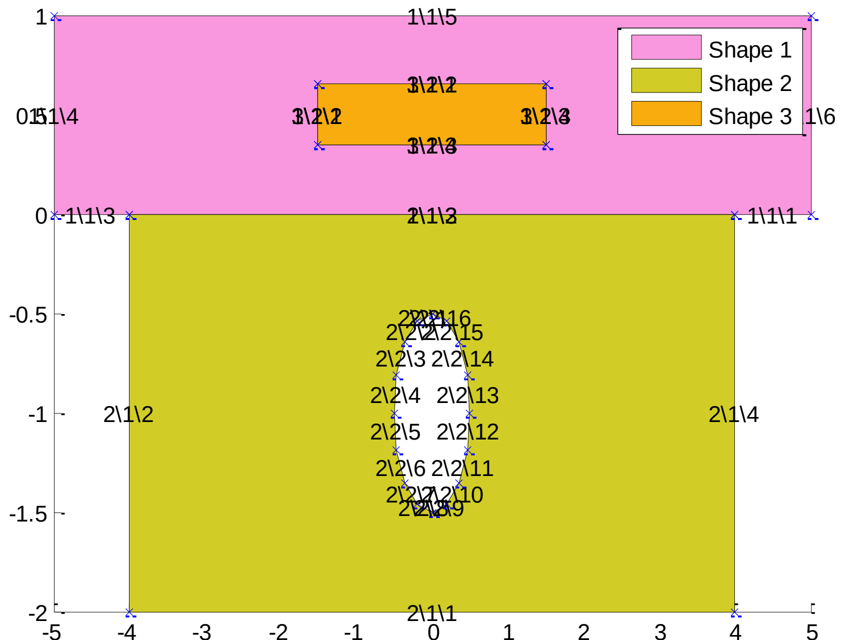
Example #2
1 – domain definition

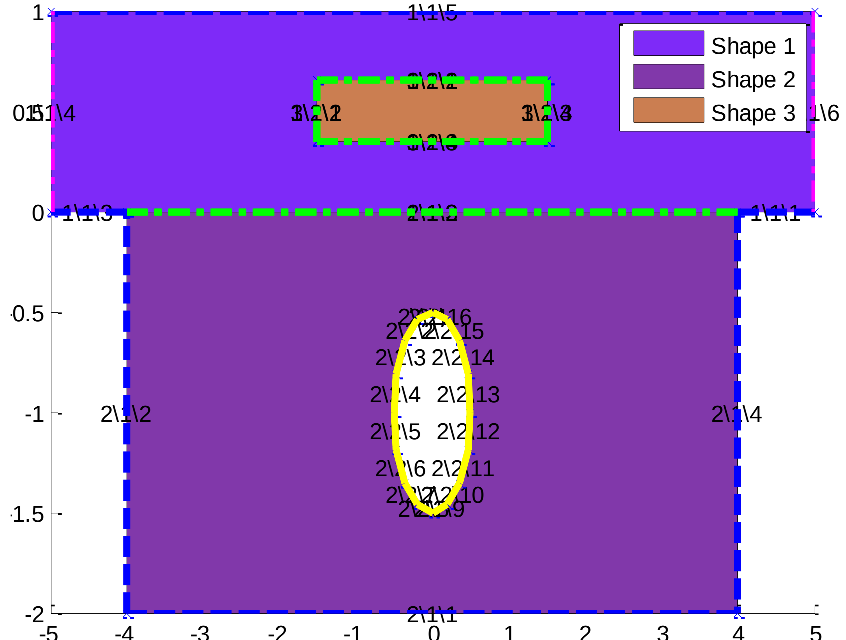
```
S(1) = regions.rect([0,0.5],[10,1],'mu',1)-
   regions.rect([0,0.5],[3,0.3]);
S(2) = regions.rect([0,-1],[8,2],'mu',2)-
   regions.circle([0,-1],0.5);
S(3) = regions.rect([0,0.5],[3,0.3],'mu',1e-3);
figure;S.draw('n');

%Add (4,0)&(-4,0) to region 1, border 1, after node 1
S(1).Borders(1) =
   S(1).Borders(1).insertNode(1,[4 0;-4 0]);
figure;
S.draw('e');
```



Example #2
2 – boundary conditions definition

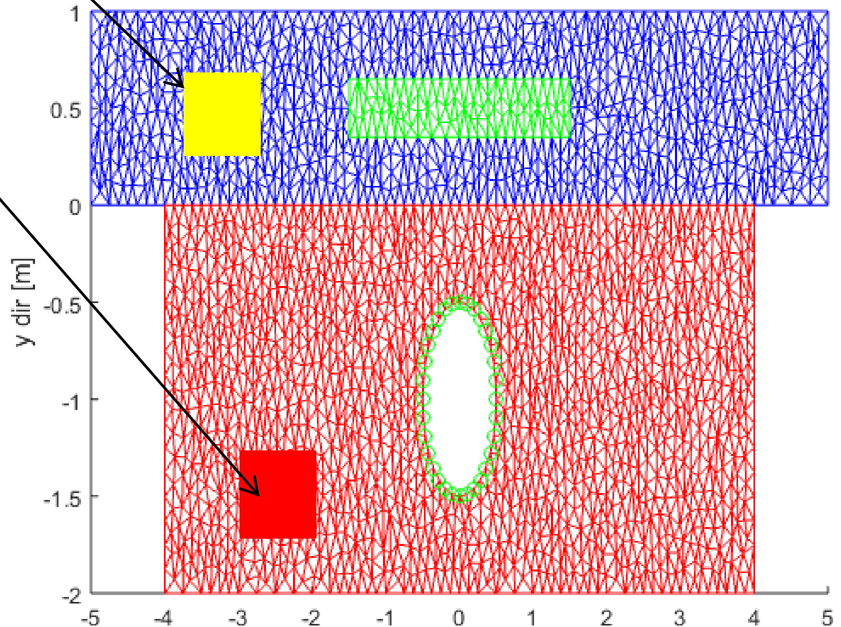
```
S(1).Borders(1).Bc([1,3,5]) = boundary.neumann(0);
S(1).Borders(1).Bc(2) = boundary.continuity();
S(1).Borders(2).Bc(:) = boundary.continuity();
S(2).Borders(1).Bc([1,2,4]) = boundary.neumann(0);
S(2).Borders(1).Bc(3) = boundary.continuity();
S(3).Borders(1).Bc(:) = boundary.continuity();
S(1).Borders(1).Bc([6,4]) = ...
   boundary.periodic(@(x,y)[-x,y]);  S.draw('bc');
```





Example #2
3 – mesh

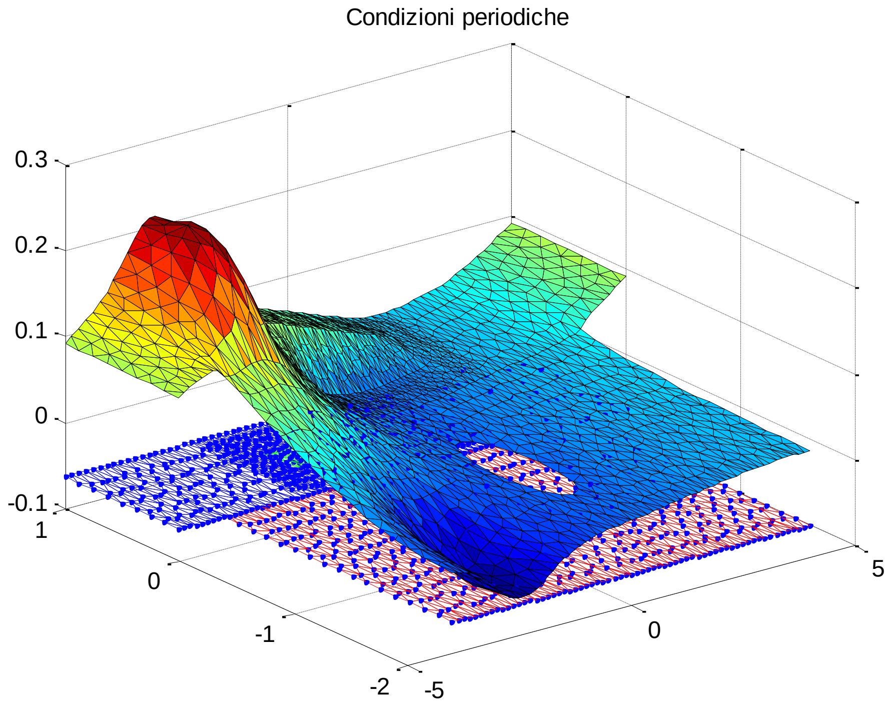
-
  ```
  Me = mesh2D(S, 0.001);
  figure;
  Me.draw('d');
  ```
- Apply an the external source in the highlighted regions
  ```
  xyc1 = [-3,0.5];      %center of yellow box, f=+1
  xyc2 = [-2.5,-1.5]; %center of red box, f=-1
  wh = [1,0.5];         %width & height of both regions
  f = @(x,y)InRect([x,y],xyc1,wh)
     -InRect([x,y],xyc2,wh);
  ```

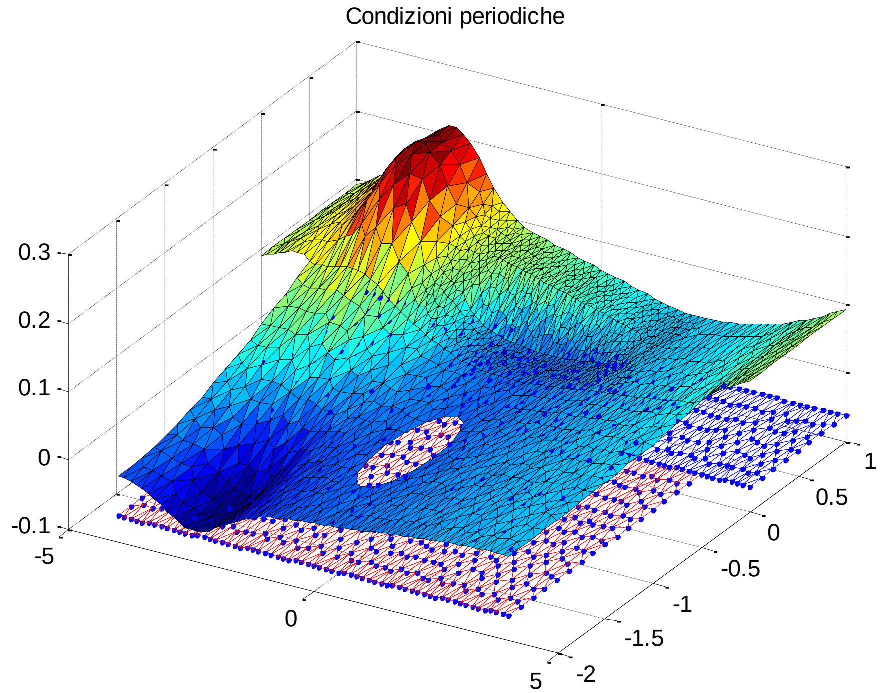


Example #2: 4 – stiffness matrix construction, 5 – solution, 6 – plot

- We build the stiffness matrix and we solve the problem

```
[D,b] = periodic_BuildStiff (Me, f);
uu = Me.copyToAllNodes(pcg(D, b, 1e-3, 1000));
figure;
Me.draw(uu, 'offset'); title('Periodic conditions');
```

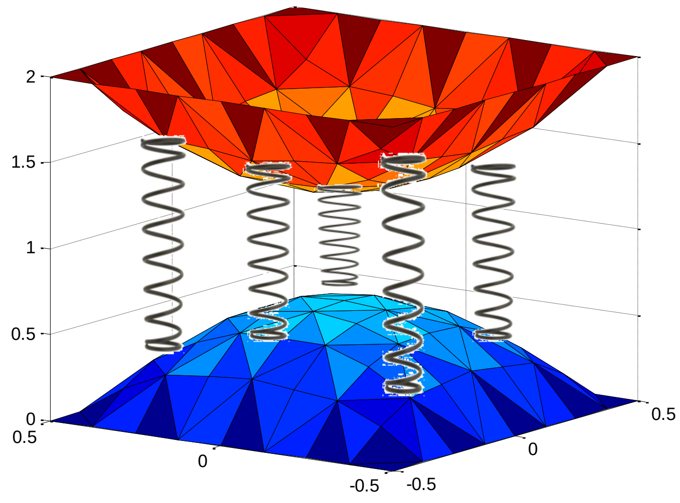




Example #3 (`RunCoupledSystems.m`)

- Let's consider two elastic surfaces linked togheter
- The borders of the first one are placed in *z=0*, while the borders of the second one are placed in *z=2*

$$\begin{cases} \mu_1 \nabla^2 u_1 + k(u_1 - u_2) = 0 \\ \mu_2 \nabla^2 u_2 + k(u_2 - u_1) = 0 \end{cases}$$



- NB: we consider Dirichlet B.C.s only. The case with different B.C.s on the top and bottom surfaces requires more attention to consistently re-map the unknown nodes of the two domains; see Example #3/b

$$\begin{cases}\mu_1\nabla^2 u_1 + k(u_1 - u_2) = 0\\ \mu_2\nabla^2 u_2 + k(u_2 - u_1) = 0\end{cases} \Rightarrow \begin{cases}D_1 u_1 + R_1(u_1 - u_2) = 0\\ D_2 u_2 + R_2(u_2 - u_1) = b_2\end{cases}$$

```
%domain with %Dirichlet hom. on the 4 sides
S1 = regions.rect('mu',1, 'sigma',1);
Me1 = mesh2D(S1);      %first mesh
Me1.draw();
%Stiffness matrix
[D1,R1,b1] = coupledDirichlet_BuildStiff(Me1);
%Domain with non hom. Dirichlet on the 4 sides (z=2)
S2 = regions.rect('mu',5, 'sigma',1);
S2.Borders(1).Bc(:) = boundaries.dirichlet(2);
Me2 = mesh2D(S2);      %second mesh
%stiffness matrix
[D2,R2,b2] = coupledDirichlet_BuildStiff(Me2);
```

Example #3
Direct method

$$\begin{cases} D_1 u_1 + R_1 (u_1 - u_2) = 0 \\ D_2 u_2 + R_2 (u_2 - u_1) = b_2 \end{cases}$$

- We solve the system directly, using a new unknown vector u obtained concatenating the two systems u_1 and u_2:

$$\begin{cases} (D_1 + R_1) u_1 - R_1 u_2 = 0 \\ -R_2 u_2 + (D_2 + R_2) u_2 = b_2 \end{cases} \Rightarrow \begin{bmatrix} D_1 + R_1 & -R_1 \\ -R_2 & D_2 + R_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ b_2 \end{bmatrix}$$

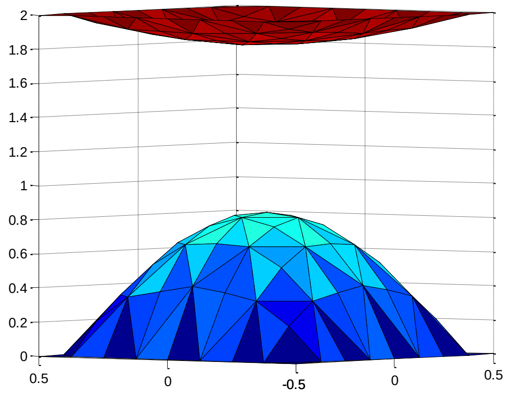
```
Atotal=[D1+R1 –R1; -R2 D2+R2];
btotal=[b1;b2];
u=Atotal\btotal;
u1=u(1:end/2);
u2=u(end/2+1:end);
```

Example #2
Iterative method

$$\begin{cases} D_1 u_1 + R_1(u_1 - u_2) = 0 \\ D_2 u_2 + R_2(u_2 - u_1) = b_2 \end{cases}$$

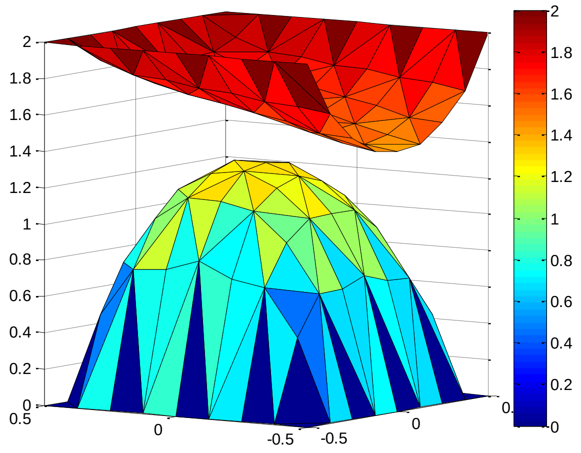
- Lets suppose we know u_1 : we can then calculate u_2. Once we know u_2, we can re-calculate u_1.
- We repeat this loop until the relative change of u_1 in (0,0) between two iterations is less than the required tolerance

```
poscenter=Me2.findClosestNode(Me2,[0,0]);
poscenterlocal=Me2.Nodes.Dof(poscenter);
u1old=0;
while err>toll && iter<itermax
   u1=pcg(D1+R1,b1+R1*u2,1e-5,2000);
   u2=pcg(D2+R2,b2+R2*u1,1e-5,2000);
   iter=iter+1;
   err= abs(u1(poscenterlocal)-u1old);
   u1old=u1(poscenterlocal);
end
```



Example #2 /b (`RunCoupledSystems2.m`)

- In the previous case, the applied B.C.s where the same and we take advantage of the correspondence between the unknown nodes of the two membranes (i.e. unknown node 34 in the top membrane was connected to node 34 in the bottom membrane.
- This is not anymore true if the B.C.s applied on corresponding edges are different; in this case also the number of unknown nodes is not the same



The top membrane has a Neumann B.C.
While the total number of nodes is 81 for both the geometries, the number of unknown nodes is 81-25 in the top one, and 81-32 in the bottom one.

Example #2 /b

- The only solution is therefore to build the full stiffness matrix, considering all the nodes, also the Dirichlet ones.
- We therefore need to generate matrices D according to this schema (with u_2 and u_4 Dirichlet nodes)

$$\begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ 0 & 1 & 0 & 0 \\ d_{31} & d_{32} & d_{33} & d_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ u_2 \\ b_3 \\ u_4 \end{pmatrix}$$

while matrices R are

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ 0 & 0 & 0 & 0 \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

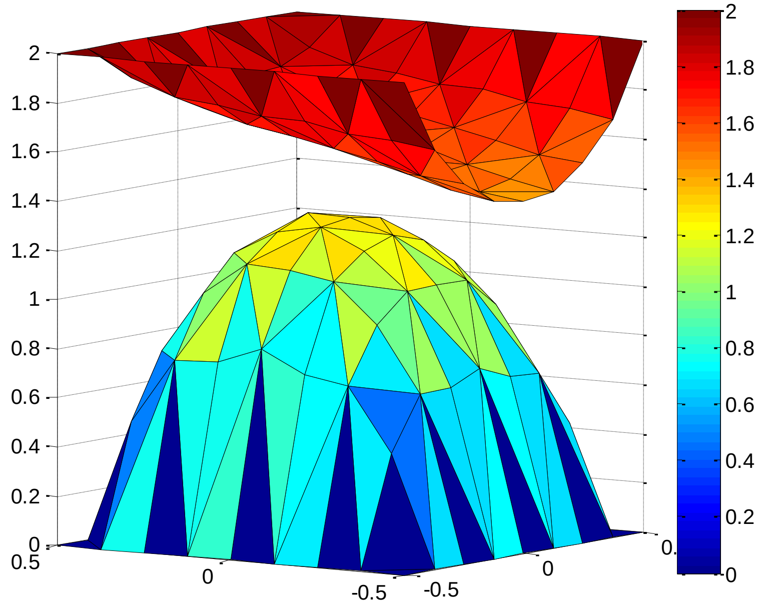
- The mass matrices must be generated in a similar way

Example #3 /b
BuildStiffnessCoupledSystems2

```
for i=1:3
   ii = Tr(e,i); %ii in the global numbering!
   if NI(ii) > 0 %dof?
      for j=1:3
         jj = Tr(e,j); %add the value
         d=mu(e)*(Dy(i)*Dy(j)+Dx(i)*Dx(j))/(4.0*Area(e));
         r=sigma(e)*Area(e)*(1+(i==j))/12;
         %suppose D and R full for simplicity
         D(ii,jj)=D(ii,jj) + d;
         R(ii,jj)=R(ii,jj) + r;
      end
   end
end
%diagonal 1s and values on the const. terms vector
ii=Me.BC.DirichletNodes(k,1);
    D(ii,ii)=1;
    b(ii)=Me.BC.DirichletNodes(k,2);
end
```

Example #3 /b

```
Atot=[D1+R1 -R1; -R2 D2+R2];
btot=[b1;b2];
uu=Atot\btot;
uu1=uu(1:end/2);
%uu1 is the displacement in all the nodes of region1
uu2=uu(end/2+1:end);
%uu2 is the displacement in all the nodes of region2
figure;
Me1.draw(uu1,'h');
hold on;
Me2.draw(uu2,'h');
```



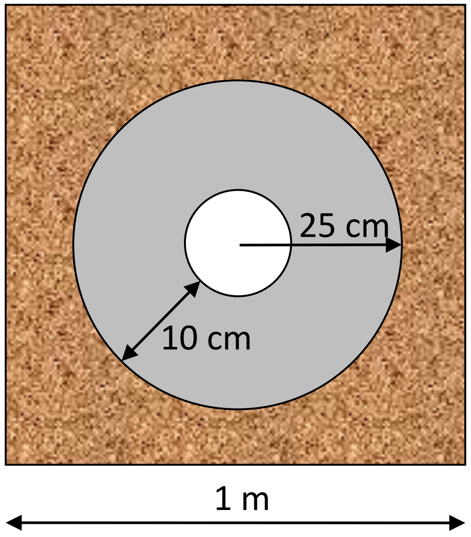
Heat equation

- When we consider the heat equation we have
  $$\frac{\partial T}{\partial t} - \nabla \cdot \left(\frac{k}{\rho c_v} \nabla T \right) + \underline{\beta} \cdot \nabla T = \frac{q}{c_v}$$
  where:
  - Density ρ → kg / m^3
  - Thermal conductivity k → W / (m K)
  - Specific heat capacity c_v → J / (kg K)
- With to the proposed formulation of the heat equation, the Robin B.C.
  $$\frac{k}{\rho c_v} \partial T / \partial n + \frac{\alpha}{\rho c_v} T = \frac{\alpha}{\rho c_v} T_\infty$$
  must be imposed as `boundaries.robin(alfa/rho/cv, alfa/rho/cv*Tinf)`,
  with `alfa` convective heat transfer coefficient [W / (m^2 K)]
- In the same way the Neumann B.C. $k / \rho c_v \, \partial T / \partial n = \Phi / \rho c_v$

  must be imposed as `boundaries.neumann(flux/rho/cv)` with the flux measured in W/m^2

Example #4

- We consider a square domain of side 1m. This domain is representative of a horizontal section of a chimney. Within the square domain is placed a circular pipe carrying fumes. The temperature of these fumes is assumed constant and equal to 40°C. The characteristics of the pipe are as follows: outer radius 25cm, 10cm thick.
- The chimney is thermally insulated on two opposite sides. On the other two sides, the chimney is in contact with the ambient air. On the windward side, assume a Robin boundary condition with T_∞=10°C. On the leeward side, assume a constant temperature on the wall, equal to 10°C. Inside the chimney assume continuity of the contact temperature between the tube and the chimney.
- For the materials, consider a brick chimney in thermal conductivity of 0.8W/(m K) and an aluminum tube of thermal conductivity 260W/(m K); it uses a coefficient of convective heat transfer equal to 10W/(m^2 K).

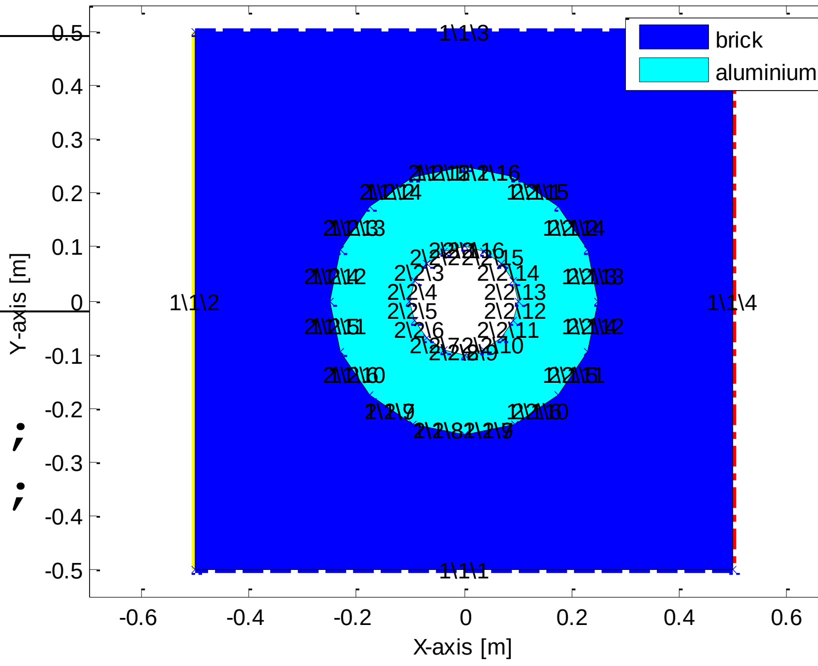


Example #4

In a MATLAB script, implement the following steps.

- Using the library region generate a geometry representative of the problem
- Associate the correct boundary conditions according to the data available in the problem description
- Generate the mesh
- Open the MatLab function `dirichletHomo_BuildStiff` (available in the folder "Examples\BasicBCs_Diffusion") and adapt it by following the steps listed below
- Using the attributes of the mesh just created in part A, find the information describing the boundary conditions
- If necessary, modify the calculation of the stiffness matrix and the constant terms vector to take into account the Dirichlet / Neumann / Robin boundary conditions
- Consider whether you need to add any external force in the calculation
- Solve the problem numerically and plot the temperature trend on a section along one of the domain diagonals
- Evaluate the flux exchanged with the air on the side with Robin B.C.

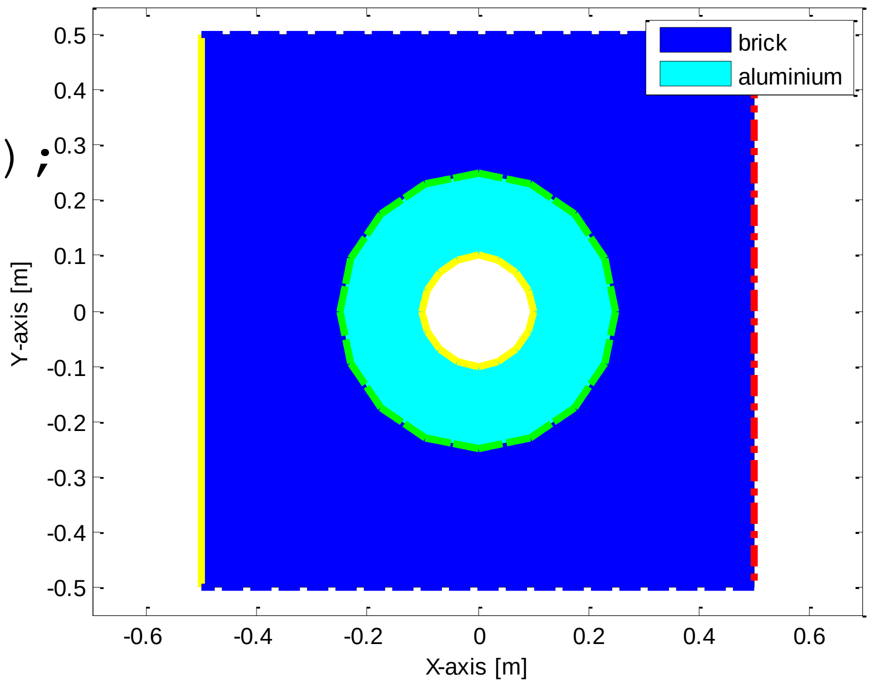
Example #4
1 – domain definition



```
DensityBricks = 1500;
ThermalConductivityBricks = 0.8;
SpecificHeatCapacityBricks= 790;
DensityAl = 2700;
ThermalConductivityAl = 260;
SpecificHeatCapacityAl = 900;
DiffusivityBricks=ThermalConductivityBricks/
   (DensityBricks*SpecificHeatCapacityBricks);
DiffusivityAl=ThermalConductivityAl/
   (DensityAl*SpecificHeatCapacityAl);
ConvectiveHeatTransfer = 10;
Sh(1)=regions.rect('mu', DiffusivityBricks)-
   regions.circle([0,0],0.25);
Sh(2)=regions.circle([0,0],0.25,'mu', DiffusivityAl)-
   regions.circle([0,0],0.10);
Sh.draw('edge');
axis equal;
```

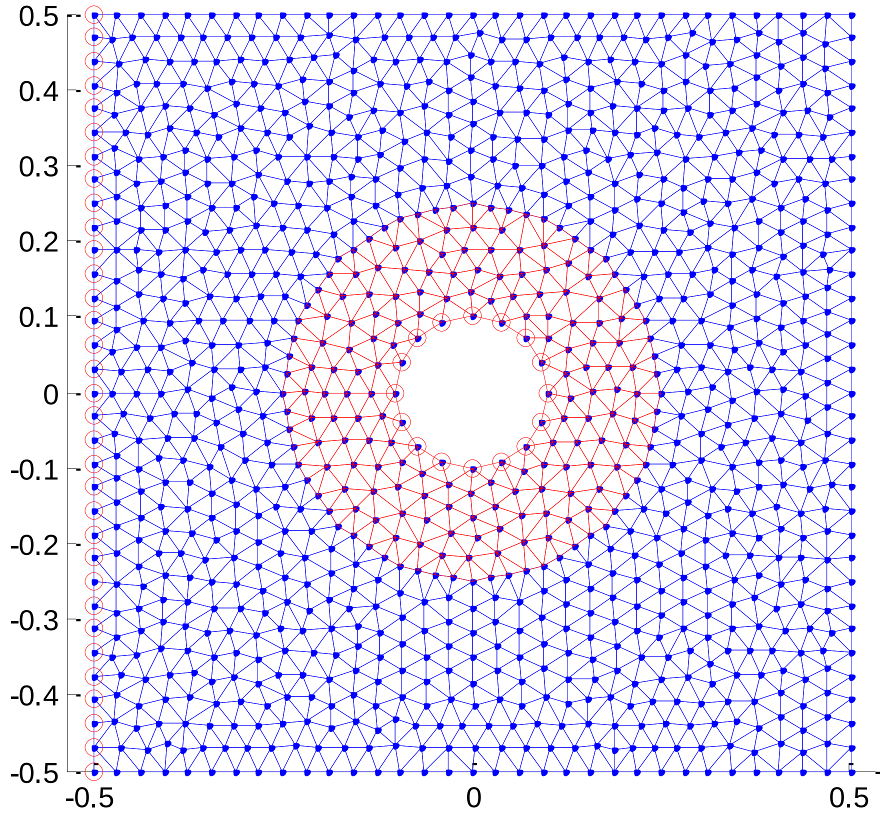
Example #4
2 – boundary conditions definition

```
Tair = 10; Tpipe = 40;
Sh(1).Borders(1).Bc([1,3]) = boundaries.neumann(0);
Sh(1).Borders(1).Bc(2) = boundaries.dirichlet(Tair);
Sh(1).Borders(1).Bc(4) = boundaries.robin(
   ConvectiveHeatTransfer,
   ConvectiveHeatTransfer *Tair);
Sh(1).Borders(2).Bc(:) =
   boundaries.continuity();
Sh(2).Borders(2).Bc(:) =
   boundaries.dirichlet(Tpipe);
Sh(2).Borders(1).Bc(:) =
   boundaries.countinuity();
figure;
Sh.draw('bc');
axis equal
```



Example #4
3 – mesh

```
Me = mesh2D(Sh,0.03);
figure;
Me.draw('dirichlet');
axis equal
```

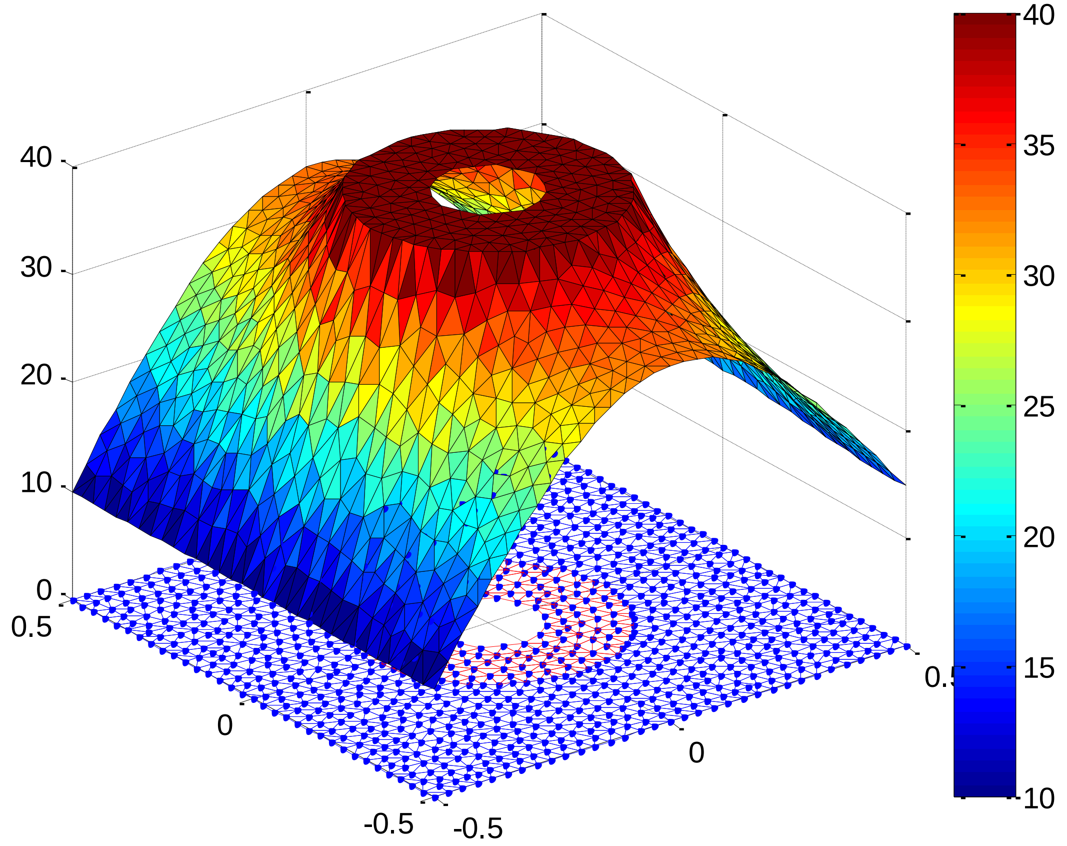


Example #4
4 – stiffness matrix construction

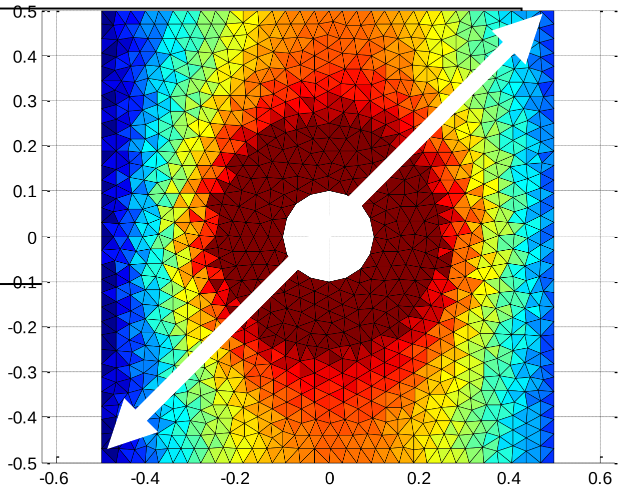
- We need to build a function to assemble the stiffness matrix accounting for the following B.C.s:
  - Non homogeneous Dirichlet
  - Homogeneous Neumann
  - Robin
- <u>We do NOT need to include an external force</u>
- We can therefore start from an already available function, such as `BuildStiffnessDiffDirichletNonHomo` (handling non homogeneous Dirichlet B.C.s), adding the code for the Robin B.C.s
- See function `chimney_BuildStiff` for the complete construction of the stiffness matrix

Example #4
5 – linear system solution & 6 – plot

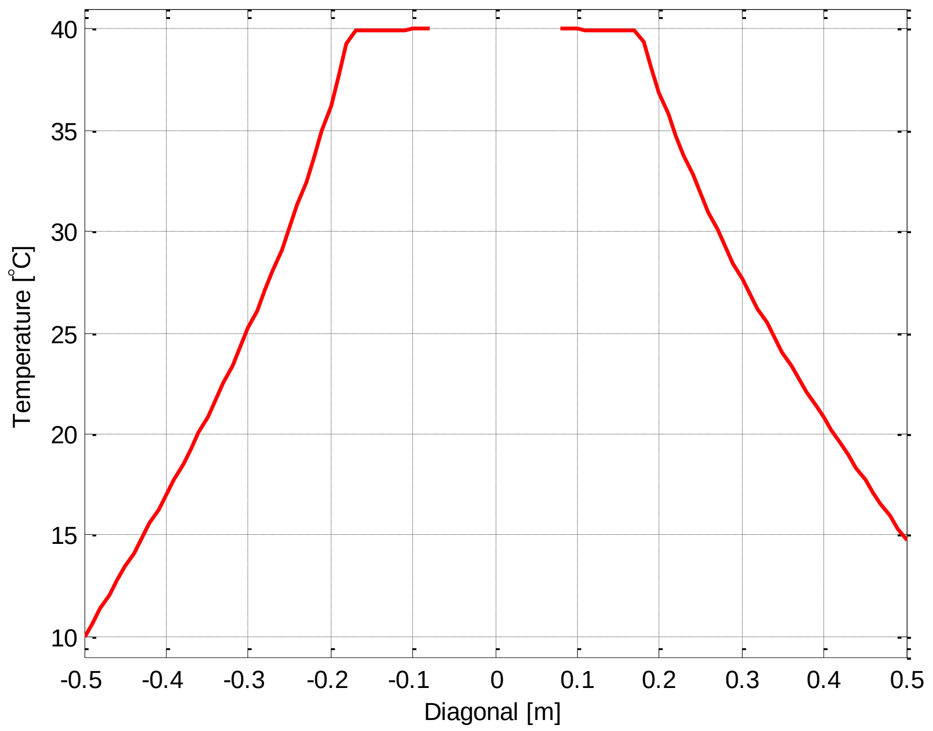
```
[A,B]= chimney_BuildStiff (Me);
T = A \ b; %or, e.g., T = pcg(D,b,1e-5,1000);
TT = Me.copyToAllNodes(T);
figure;
Me.draw(TT);
```



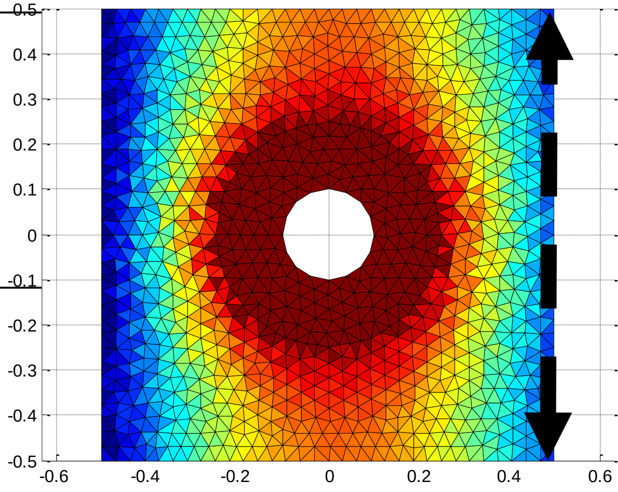
Example #4
6 – interpolation



```
x=-0.5:0.01:0.5;
y=x;
TDiag=Me.interpolate(TT, [x;y]');
figure;
plot(x,TDiag,'r');
ylabel('Temperature [^\circC]');
xlabel('Diagonal [m]');
grid on;
```



Example #4
Flux calculation



- To evaluate the flux per meter of height exchanged between the right vertical side and the air, we calculate

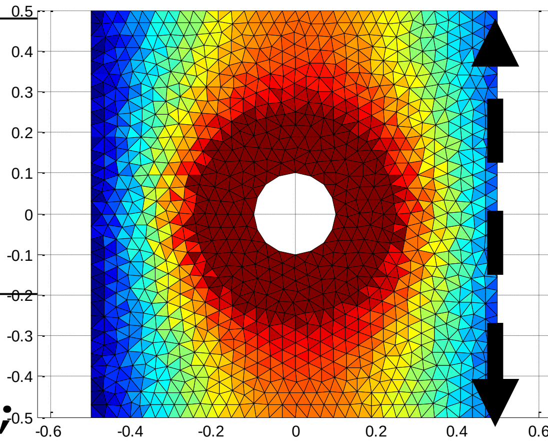
$$\frac{\Phi}{l} = h\int_{edge} \left(T - T_{air}\right) ds$$

- This expression can be approximated as

$$\frac{\Phi}{l} \approx h\sum_{k} \left[\left(T_{k,1} - T_{air}\right) + \left(T_{k,2} - T_{air}\right)\right]\frac{l_k}{2}$$

with k index of the edges on the Robin side having length l_k, and $T_{k,1}$ and $T_{k,2}$ temperatures at the two vertices of edge k

Example #4
Flux calculation

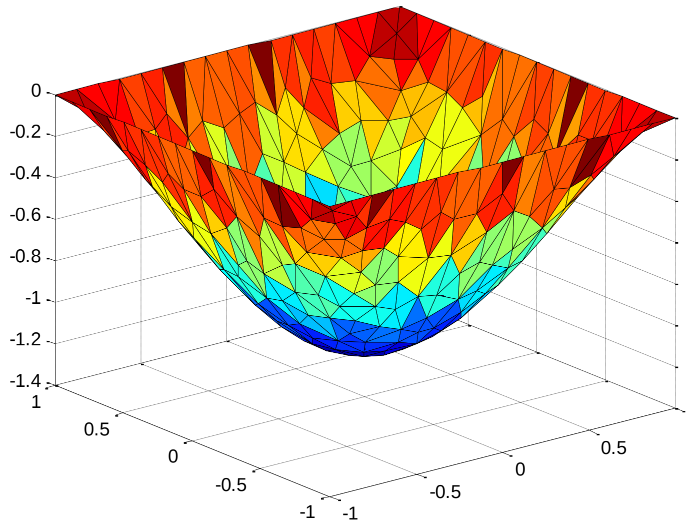
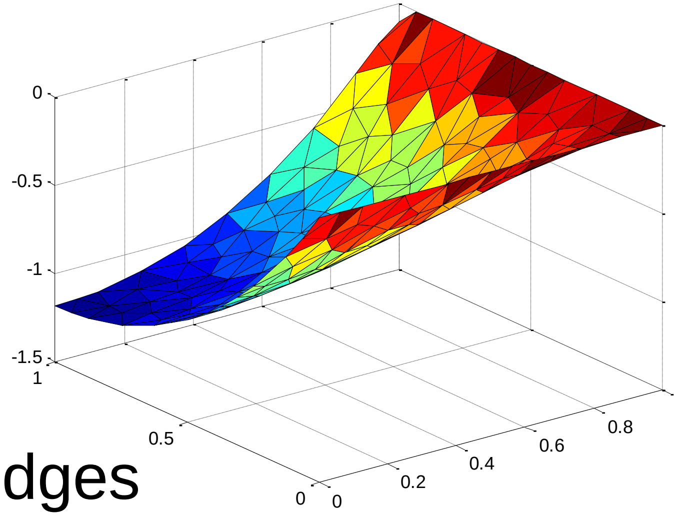


```
%couples of indices of the nodes
Nodes=Me.Edges(Me.BC.RobinEdges(:,1),:);
Flux=0;
%for exch Robin edge
for k=1:size(Nodes,1)
   Node1=Nodes(k,1);
   Node2=Nodes(k,2);
   %the node are on vertical segments, I calculate its
   %length simply as difference of the y-coordinates
   dy=Me.Nodes.Y(Node1)-Me.Nodes.Y(Node2);
   %temperature on the two nodes
   T1=TT(Node1);
   T2=TT(Node2);
   Flux=Flux+((T1-10)+(T2-10))*abs(dy);
end

Flux=Flux*ConvectiveHeatTransfer/2;
```

$$\frac{\Phi}{l} \approx h\sum_k \left[\left(T_{k,1}-T_{air}\right)+\left(T_{k,2}-T_{air}\right)\right]\frac{l_k}{2}$$

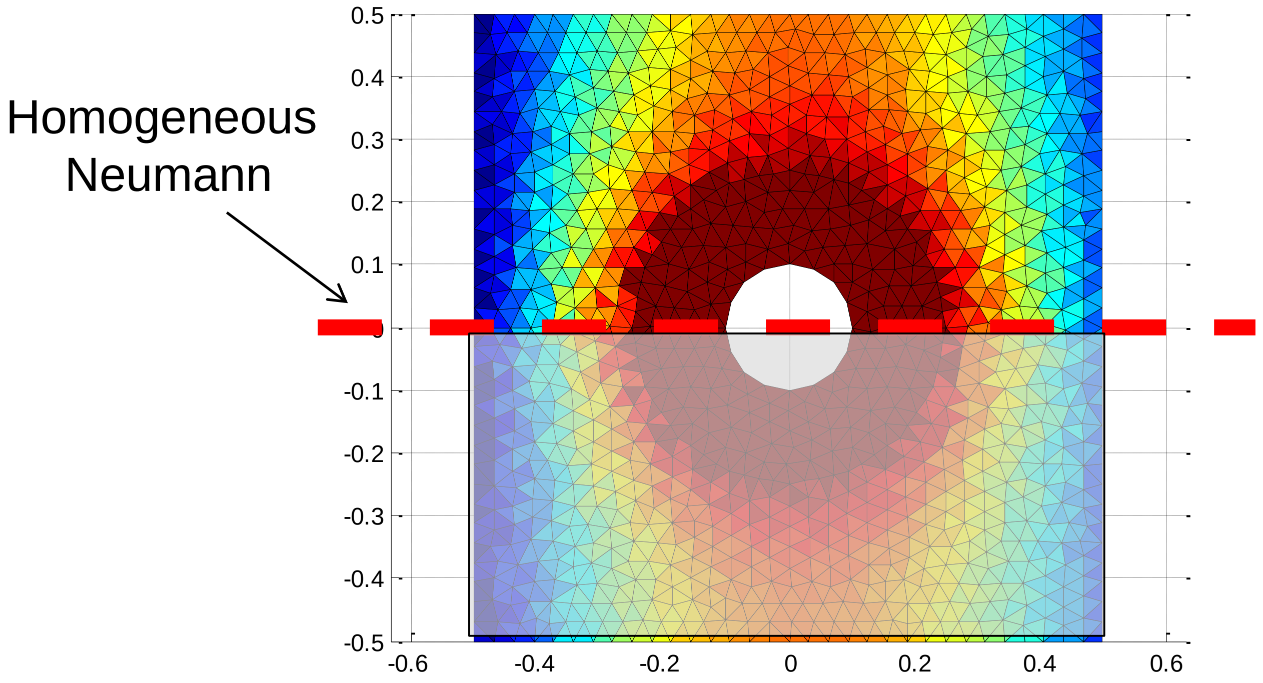
Geometry simplification

- It is often possible to reduce the dimension of the domain under exam, due to particular symmetries of the domain, the BCs and the external forces. In this way reduce the number of unknowns and the memory/time required to solve the problem
- Example: squared domain, constant *μ* and Dirichlet BC on the 4 sides, *f* even function: the solution as well is even. I consider ¼ of the domain placing homogeneous *Neumann* conditions on the internal edges

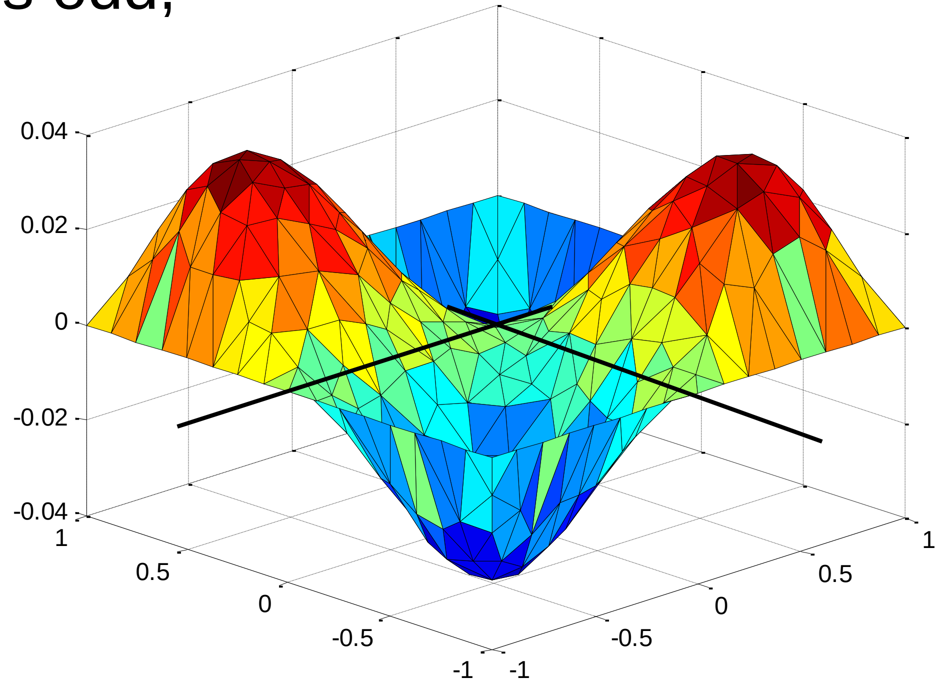
Geometry simplification

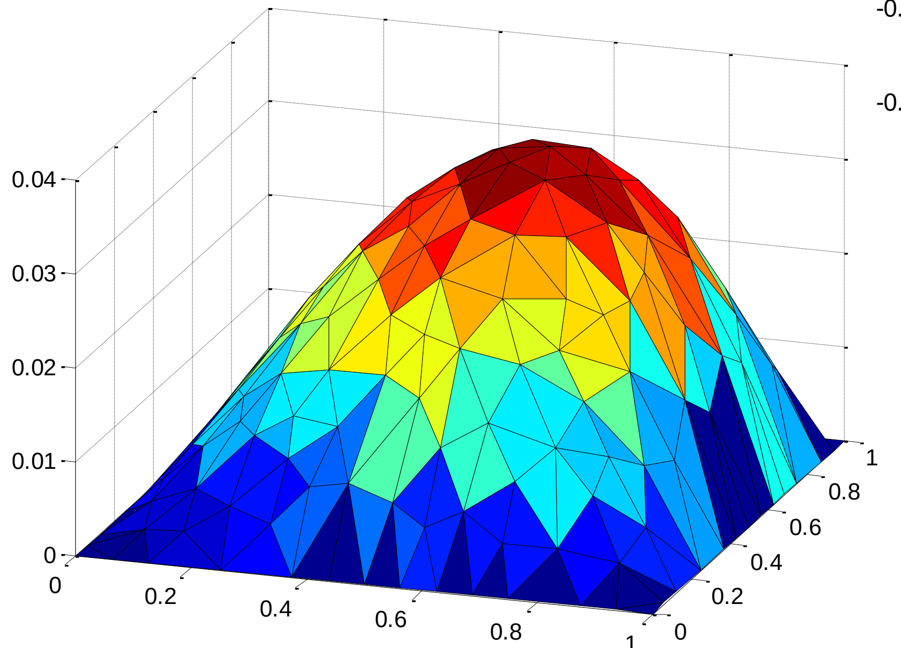
- Also, in the chimney problem, it was possible to analyse only half domain to the vertical symmetry, applying an homogeneous Neumann B.C. on the symmetry axes.



Geometry simplification

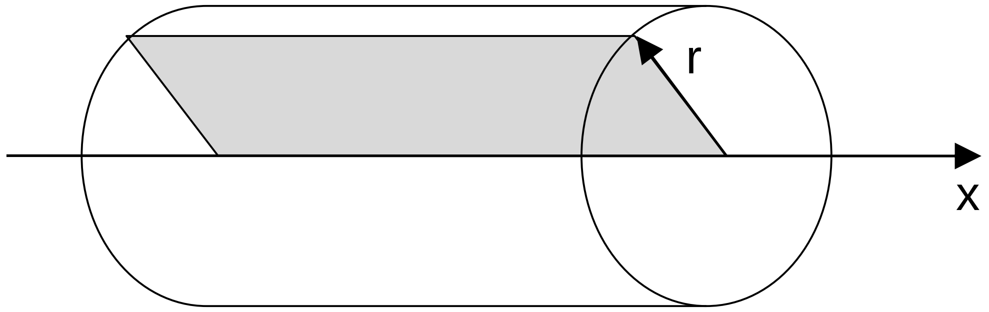
- If on the contrary the force is odd, it is possible to study ¼ of the domain, imposing homogeneous *Dirichlet* conditions on the internal edges





Cylindrical Coordinates

- If the 3D geometry we are interested in shows a cylindrical symmetry, then we can reduce it to a 2D geometry



$$\alpha \nabla_c^2 u(x,r) - \underline{\beta} \cdot \nabla_c u(x,r) = 0$$

$$\underline{\beta} = \left(\beta_x, \beta_y\right)$$

$$\nabla_c^2 u(x,r) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r}$$

$$\nabla_c u(x,r) = \left[\frac{\partial u}{\partial x}, \frac{\partial u}{\partial r}\right]$$

- Therefore

$$\alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial r^2}\right) - \left(\beta_x, \beta_y - \frac{\alpha}{r}\right) \cdot \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial r}\right) = 0 \rightarrow \alpha \nabla^2 u - \underline{\tilde{\beta}} \cdot \nabla u = 0$$

and it is simply required to modify the transport term!

Cylindrical Coordinates

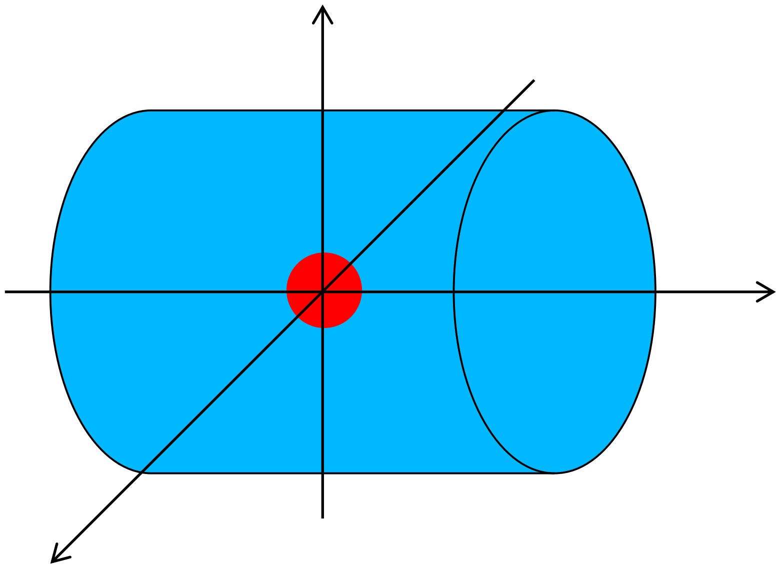
$$\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial r^2}\right)-\left(\beta_x,\beta_y-\frac{\alpha}{r}\right)\cdot\left(\frac{\partial u}{\partial x},\frac{\partial u}{\partial r}\right)=0\rightarrow\mu\nabla^2 u-\underline{\tilde{\beta}}\cdot\nabla u=0$$

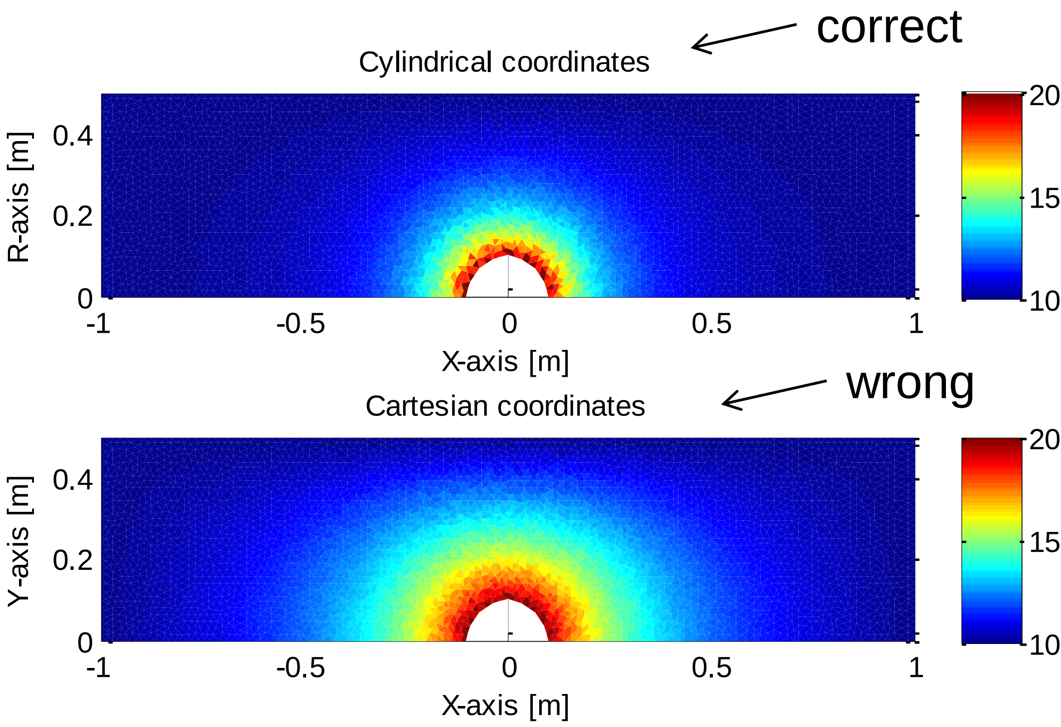
- The term $\left(\beta_x,\beta_y-\mu/r\right)$, when a sufficiently dense mesh is used, can be approximated as $\left(\beta_x,\beta_y-\mu/r\right)$ with r_B distance of the center of mass to the symmetry axes.
- The inner loop used for the stiffness matrix construction becomes therefore (see `cylindrical_BuildStiff`)

```
betax = beta(e,1);
betar = beta(e,2)-mu(e)/rb;%correction for cylind. coords
diffusion=mu(e)*(Dr(i)*Dr(j)+Dx(i)*Dx(j))/(4.0*Areas(e));
transport=-(betax*Dr(j)-betar*Dx(j))*1/6;
if jj > 0
    row(pos)=ii; col(pos)=jj;
    d(pos)= diffusion + transport; pos=pos+1;
else
    value=Me.BC.DirichletNodes(-jj,2);
    b(ii)=b(ii)-(diffusion+transport)*value;
end
```

Cylindrical Coordinates

- Let’s consider a 2m high cylinder with a base diameter of 1m, full of water whose external temperature is fixed at 10°C. In the center of the cylinder is placed a sphere having a temperature of 20°C




correct
Cylindrical coordinates
R-axis [m]
X-axis [m]
wrong
Cartesian coordinates
Y-axis [m]
X-axis [m]