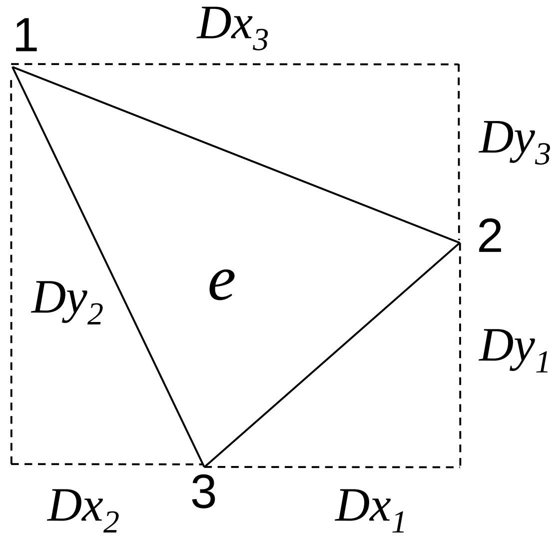
2D Finite Elements
Time-dependent solution

Version 2017.1

Mass matrix calculation

- The calculation of the mass matrix follows a logic similar to the one used for the stiffness matrix:



For each triangle e

For each local vertex i $(1 \rightarrow 3)$

ii=Dof Numbering(i)

If vertex ii is unknown

For each local vertex j $(1 \rightarrow 3)$

jj=Dof Numbering(j)

If vertex jj is unknown

$$M(ii,jj)=M(ii,jj)+rho(e)*Area(e)*(1+\delta(i,j))/12$$

Mass matrix calculation

```
function M = buildMass(Me)
V =Me.Triangles.Vertices; Dof =Me.Nodes.Dof; numDof =max(Dof);
Areas=Me.Triangles.Areas; Mcontr=Me.MatrixContributions;
pos = 1; row = zeros(Mcontr, 1); col = zeros(Mcontr, 1);
m = zeros(Mcontr, 1); rho = Me.evaluateProperty('rho');
for e = 1:size(V, 1)    %main loop over each triangle
    for ni = 1:3        %for each vertex of this triangle
        ii = Dof(V(e, ni));
        if ii > 0       %is it a degree of freedom? Yes
            for nj = 1:3 %second loop
                jj = Dof(V(e, nj));
                if jj > 0 %add to the mass matrix
                    m(pos) = 1/12 * Areas(e) * rho(e) * ...
                        ((ii == jj) + 1);
                    row(pos) = ii; col(pos) = jj; pos = pos + 1;
                end
            end
        end
    end
end
M = sparse(row, col, m, numDof, numDof);
```

$$M = \begin{cases} \dfrac{A}{12} & se\, i \neq j \\ \dfrac{A}{6} & se\, i = j \end{cases}$$

Mass matrix calculation

- The lumping technique allows to build a diagonal mass matrix: we use therefore a vector to store the data
- This approach is particularly convenient when we use Explicit Euler method

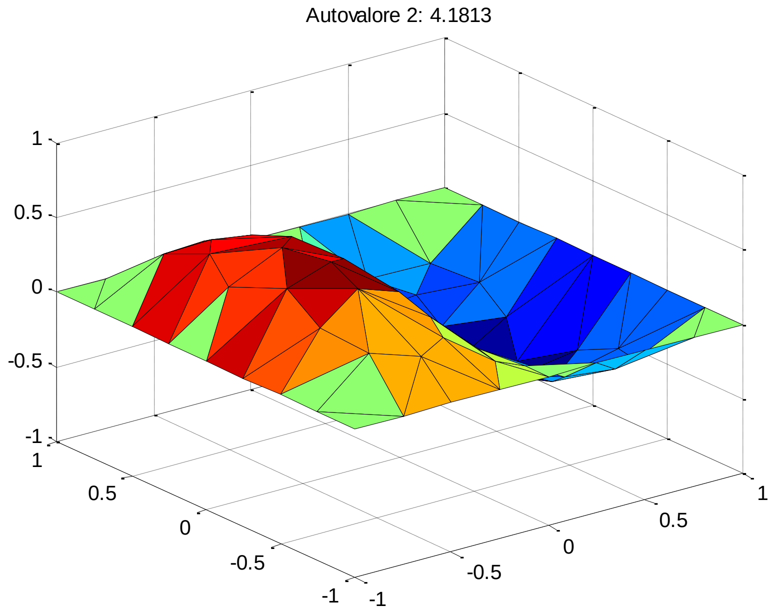
```
function M = buildMassLumping(Me)
rho = Me.evaluateProperty('rho');
V=Me.Triangles.Vertices;
Dof=Me.Nodes.Dof;
NumDof = max(Dof);
M = zeros(NumDof,1);
for e=1:size(Tr,1)
    for ni=1:3
        ii = Dof(V(e,ni));
        if ii > 0
            M(ii) = M(ii) + Areas(e) / 3 * rho(e);
        end
    end
```

$$\tilde{M} = diag\left(\sum_{i=1}^{N_h} b_{ij}\right) = \begin{cases} 0 & se\, i \neq j \\ \dfrac{A}{3} & se\, i = j \end{cases}$$

Eigenfunctions evaluation

- We need to calculate the eigenvalues of $M^{-1}A$
- Specific algorithms exist, which allow to calculate eigenvalues and eigenvectors without calculating the inverse of M
  They are implemented, e.g., in the function `eigs`

```
[V,E]=eigs(A,M,2,'SM');
```



Temporal analysis

- When we consider a first order time derivative, we can use the well known integration methods, with M mass matrix

$$\begin{cases} \dfrac{\partial u}{\partial t} + \nabla \cdot (\mu \nabla u) = f(t,x,y) \rightarrow M\dfrac{\partial u}{\partial t} = -Du + b \\ u(t=0) = u_0 \end{cases}$$

- Explicit Euler: $Mu^{k+1} = Mu^k - D\Delta t u^k + b^k \Delta t$

  → convergence problems if dt too large
- Implicit Euler: $(M + D\Delta t)u^{k+1} = Mu^k + b^{k+1}\Delta t$
- CN: $$\left(M + \frac{D\Delta t}{2}\right)u^{k+1} = Mu^k - \left(\frac{D}{2}u^k - \frac{b^{k+1}}{2} - \frac{b^k}{2}\right)\Delta t$$

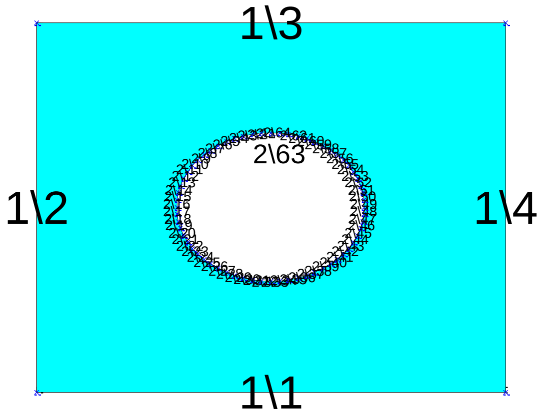
Example #1: heating of a squared domain

$$\rho c_v \frac{\partial T}{\partial t} - \nabla \cdot \left(k \nabla T \right) = f$$

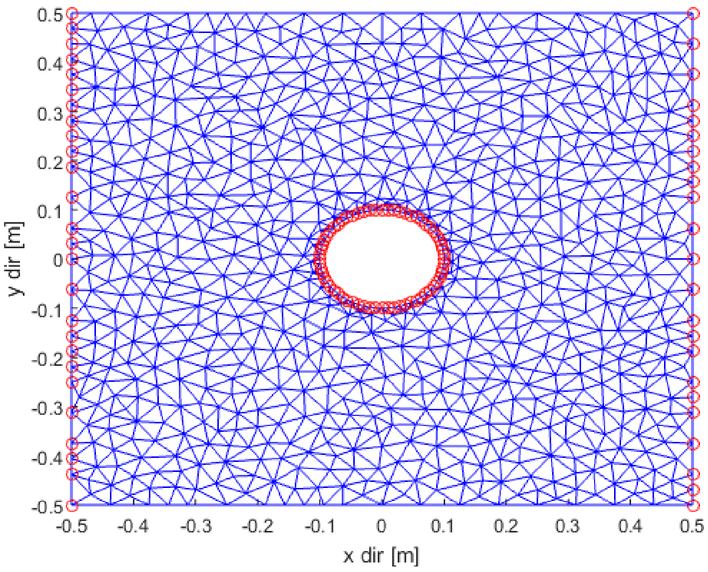
- We study the temporal evolution of the temperature on an rectangular aluminium plate with a hole in the middle; the following B.C.s are applied
  - Homogeneous Neumann on two opposite external sides (perfect insulation)
  - Homogenerous Dirichlet on the two other external sides
  - Dirichlet, with a temperature of 20°C on the internal edges
- For the considered material we have
  - density ρ: 2700 kg/m^3
  - specific heat capacity c_v: 896.9 J/(kg K)
  - thermal conductivity k: 237 W/(m K)
- An external source is applied:

$$f(t) = \begin{cases} 0 & \text{if } t \le 0 \\ 30t & \text{if } 0 < t < 1000 \\ 30000 & \text{if } t \ge 1000 \end{cases}$$

Example #1: definition of geometry, B.C. and mesh




```
S=regions.rect('mu', 237/2700/897)-
   regions.circle([0,0],[0.2,0.2],64);
figure;Sh.draw('e');

Sh.Borders(1).Bc([1,3]) =
   boundaries.neumann(0);
Sh.Borders(2).Bc(:) =
   boundaries.dirichlet(20);
figure;Sh.draw('bc');

Me=mesh2D(S,0.001);
figure;Me.draw('d');
```



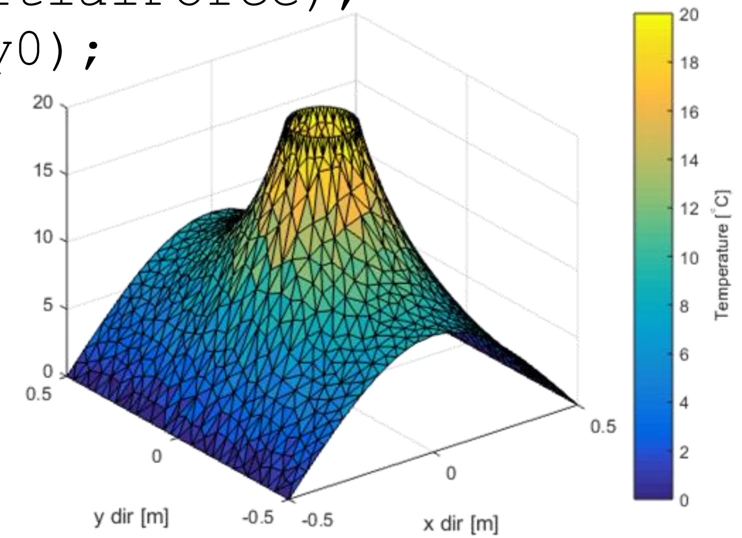
```
function [D, bBoundary, bExt] =
heatEquationExtForce_BuildStiff(Me,f)
```

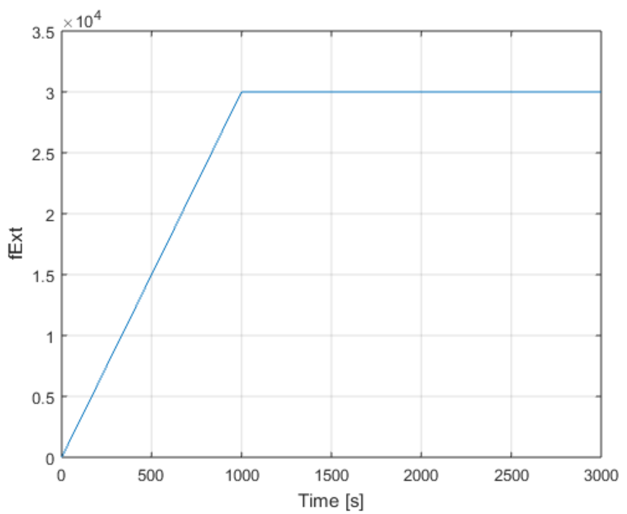
- We need to "split" the constant terms vector in two parts: b=bBoundary+bExt(t)
  - bBoundary, related to the BC.s, is constant, since the applied BC.s do not change with respect to time
  - bExt is time dependant and follows the assigned law

```
if ii > 0          %dof, second loop on the vertices
   for nj=1:3
      jj = Dof(V(e,nj));   %is it unknown as well?
      if jj > 0 ...
      else
         val=Me.BC.DirichletNodes(-jj,2);
         bBoundary(ii) = bBoundary(ii) - dtmp*val;
      end
   end
   bExt(ii) = bExt(ii) + Areas(e)*force(e)/3.0;
end
```

Example #1: Initial solution, f=0

```
f0 = @(x,y)ones(size(x)); %assume f0=1
[D,bBorder,bExt]=heatEquationExtForce_BuildStiff(Me, f0);
InitialForce = 1000;     %force is 1000*f0
uStationary0 = D\(bBorder+bExt*InitialForce);
uu = Me.copyToAllNodes(uStationary0);
figure;
Me.draw(uu);
ylabel('Temperature [^\circC]');
colorbar();
```






```
M=buildMass(Me);
Tend=4000; %end time, in s
f=@(t)InitialForce+(t<1000)*(30*t)
  +(t>=1000)*30000;
figure;fplot(f,[0,100]);
```

Example #1: Temporal evolution

- All the discussed methods follow the following steps:
  - Choice of the time step, e.g. `dt = 0.1;`
  - Definition of the initial temperatures `u = uStationary0;`
  - Temporal loop: `for k = 1:Tend/dt,`
  - Save indices of the nodes which are dof: `Dofs=Me.Nodes.dof>0;`
  - In each time considered step:
    - calculate the new force `T4=f(t);`
    - Solve the linear system to calculate the new solution

```
u=…………………
uu(Dof) = u;        %update only dofs, it’s faster!
```

    - (if required) redraw the solution, for instance with

```
Me.draw(uu,'hidemesh');%I hide the mesh
title(['t= ' num2str(t) 's']); %elapsed seconds
zlim([0 20]);  %fixed vertical range
caxis([0 20]); %fixed color scale
view ([0 90]);    %top view
drawnow();        %force MATLAB to immediately redraw
```

Esplicit Euler method (EE)

$$Mu^{k+1} = Mu^{k} - D\Delta t u^{k} + b^{k}\Delta t$$

```
dt=0.1;
u=uStationary0;
Ddt=D*dt;
for k=1:Tend/dt,
    t=k*dt;
    F=f(t-dt); %previous step!
    u=u-M\(Ddt*u-dt*(bBoundary+bExt*F));
    uu(Dofs)=u;
    Me.draw(uu,'hidemesh');    hold off;
    zlim([0,20]);
    caxis([0,20]);
    view([0,90]);
    title(['t=',num2str(t) 's']); drawnow();
end
```

NB! Please pay attention to the correct choice of dt: $dt \leq 2/\lambda_{\max}$ with $\lambda_{\max}$ largest eigenvalue, in module, of $M^{-1}A$, to be calculated as `eigs(A,M,1,'LM')` and **NEVER** taking the inverse of M!

Implicit Euler method (IE)

$$(M + D\Delta t)u^{k+1} = Mu^k + b^{k+1}\Delta t$$

```
dt = 0.1;
u = uStationary0;
S = (M+D*dt);
for k=1:Tend/dt,
    t = k*dt;
    F = f(t); %current step!
    u = S\(M*u+dt*(bBoundary+bExt*F));
    uu(Dofs) = u;
    Me.draw(uu,'hidemesh');
    hold off;
    zlim([0,20]);
    caxis([0,20]);
    view([0,90]);
    title(['t=',num2str(t) 's']);
    drawnow();
end
```

Crank-Nicholson method (CN)

$$\left(M+\frac{D\Delta t}{2}\right)u^{k+1}=Mu^{k}-\left(\frac{D}{2}u^{k}-\frac{b^{k+1}+b^{k}}{2}\right)\Delta t$$

```
dt=10;
u=uStationary0;
B=D*dt/2;  CN1=M+B;  CN2=M-B;
Fold=f(0);
for k=1:Tend/dt,
   t=k*dt;
   F=f(t);
   u=CN1\(CN2*u+dt*(bBoundary+bExt*(F+Fold)/2));
   uu(Dofs)=u;
   Me.draw(uu,'hidemesh');
   hold off;
   zlim([0,20]);
   caxis([0,20]);view([0,90]);
   title(['t=',num2str(t) 's']);drawnow();
   Fold=F;
end
```

Temporal evolution

- At every time step, we need to solve a linear system that allows to calculate the new approximation of the solution
- If the mass matrix (EE) or both the stifness and mass matrices (IE, CN) do not change in time, it is generally very efficient to evaluate their preconditioners (see functions `ilu` and `ichol`).
- This approach, used in conjunction with an iterative method starting from the solution at the previous step, can significantly reduce the required computation time

Implicit Euler method (IE) with preconditioning

```
dt = 0.1;
u = uStationary0;
S = (M+D*dt);
%u=S\(M*u+bdt*T4); since at each step I must solve the
%system Su=..., then I pre-factor S
opt = struct('type','ict','droptol',1e-4);
R = ichol(S,opt);
for k=1:Tend/dt,
   t = k*dt;
   F = f(t);
   [u, flag] = pcg(S,M*u+dt(bBoundary+bExt*F),
      1e-6,1000,R',R,u);
   uu(Dofs) = u;
   Me.draw(uu,'hidemesh');zlim([0,20]);
   caxis([0,20]);view([0,90]);
   title(['t=',num2str(t) 's']);drawnow();
end
```

NB: to estimate the time reduction, do not draw the solution!

`ODExx` functions

The syntax is the same for all the functions of the family

```
[t,y]=ode45(dydt,[T0, Tend], y0, odepar)
```

- `t` column vector of N components containg the time values the solution was calculated
- `y` matrix of N rows, each one containing the solution calculated at the corresponding time step indicated in the t vector
- `dydt` function to integrate. It MUST receive two input parameters, (time and y), like `dydt=@(t,y)siny)+t`
- [`T0`, `Tend`] 2 components vector, indicating the initial and final integration time
- `y0` problem initial values
- `odepar` optional variable created by the `odeset` function to set additional integration parameters such as relative or absolute tolerances, mass matrix, maximum time step,...

```
odepar=odeset('Absrel',1e-6,'AbsTol',1e-3', 'NonNegative',1,
'MaxStep',1e-1, 'Mass',M);
```

`ODExx` functions

- In MATLAB, to numerically integrate (systems of) differential equations the `odexx` functions are available

Solver	Problem Type	Order of Accuracy	When to Use
`ode45`	Nonstiff	Medium	Most of the time. This should be the first solver you try.
`ode23`	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
`ode113`	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
`ode15s`	Stiff	Low to medium	If `ode45` is slow because the problem is stiff.
`ode23s`	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
`ode23t`	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
`ode23tb`	Stiff	Low	If using crude error tolerances to solve stiff systems.

- All these functions have the same calling syntax, but implement different algorithms

`ODExx` functions: examples

- Let's consider some examples of Cauchy problems integrated over the interval [0,1]
- $$\begin{cases} \mathrm{d}y/\mathrm{d}t + a\sin(t) - y^2 = 0 \\ y(0) = 0 \end{cases} \rightarrow \begin{cases} \mathrm{d}y/\mathrm{d}t = -a\sin(t) + y^2 \\ y(0) = 0 \end{cases}$$

```
>>[t,y]=ode45(@(t,y)-a*sin(t)+y.^2,[0,1],0);
```

- $$\begin{cases} \mathrm{d}^2 y/\mathrm{dt}^2 - \exp(t+y) = 0 \\ y(0) = 0 \\ dy/dt\big|_{t=0} = 2 \end{cases} \rightarrow \begin{cases} \mathrm{d}y_1/\mathrm{dt} = y_2 \\ \mathrm{d}y_2/\mathrm{dt} = \exp(t + y_1) \\ y_1(0) = 0 \\ y_2(0) = 2 \end{cases}$$

```
>>[t,y]=ode45(@(t,y)[y(2);exp(t+y(1)),[0,1],[0;2]);
```

Example #1: `ODExx` functions

$$M\frac{\partial u}{\partial t} = -Du + b$$

```
u0 = uStationary0;
figure;
fode = @(t,u)-D*u+(bBoundary+bExt*f(t));
odepar = odeset('Mass',M);     %Mass matrix
[t,U] = ode45(fode, [0,Tend],u0,odepar);
size(U)                  % 24809 x 769 -> 769 time steps
for k = 1:10:length(t) %now simply plot the solution
   uu(Dofs) = U(k,:);
   Me.draw(uu,'hidemesh');
   zlim([0 TMax]);
   caxis([0 TMax]);
   %  view([0 90]);
   title(['t= ' num2str(t(k)) 's']);
   drawnow();
end
```

Time varying Dirichlet B.C.s

- Let's consider the system $M\,\partial u/\partial t + Du = b$ with M and D built on all the nodes (and not only the internal ones)
- Let nodes 1 and 3 be dof, node 2 a node on an edge with time invariant Dirichlet B.C (u_2) and node 4 on an edge with a time-varying Dirichlet B.C. ($u_4(t)$)

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} \partial u_1/\partial t \\ \partial u_2/\partial t \\ \partial u_3/\partial t \\ \partial u_4/\partial t \end{pmatrix} = \begin{pmatrix} m_{11} & 0 & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & 0 & m_{33} & 0 \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} \partial u_1/\partial t \\ \partial u_2/\partial t \\ \partial u_3/\partial t \\ \partial u_4/\partial t \end{pmatrix} + \begin{pmatrix} m_{12}\,\partial u_2/\partial t + m_{14}\,\partial u_4/\partial t \\ 0 \\ m_{42}\,\partial u_2/\partial t + m_{44}\,\partial u_4/\partial t \\ 0 \end{pmatrix}$$

- Moving to the system involving only the dof we finally have

$$\begin{pmatrix} m_{11} & m_{13} \\ m_{31} & m_{33} \end{pmatrix} \begin{pmatrix} \partial u_1/\partial t \\ \partial u_3/\partial t \end{pmatrix} + \underbrace{\partial u_2/\partial t}_{=0} \begin{pmatrix} m_{12} \\ m_{42} \end{pmatrix} + \partial u_4/\partial t \begin{pmatrix} m_{14} \\ m_{44} \end{pmatrix} = \begin{pmatrix} m_{11} & m_{13} \\ m_{31} & m_{33} \end{pmatrix} \begin{pmatrix} \partial u_1/\partial t \\ \partial u_3/\partial t \end{pmatrix} + \partial u_4/\partial t\, m_{\mathrm{var},4}$$

- We need to take into account this additional contribution

Example #2: heating of a squared domain

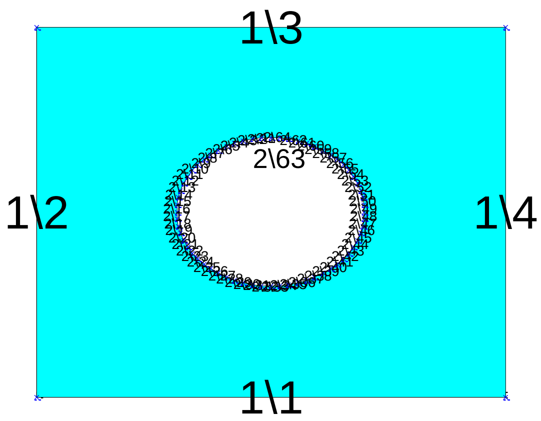
- We study the same problema as Example#1 but the following B.C.s are now applied:
  - Homogeneous Neumann on two opposite external sides (perfect insulation)
  - Non homogeneous Dirichlet on the two other external sides, where T=20°
  - Dirichlet on the internal edges, with the law

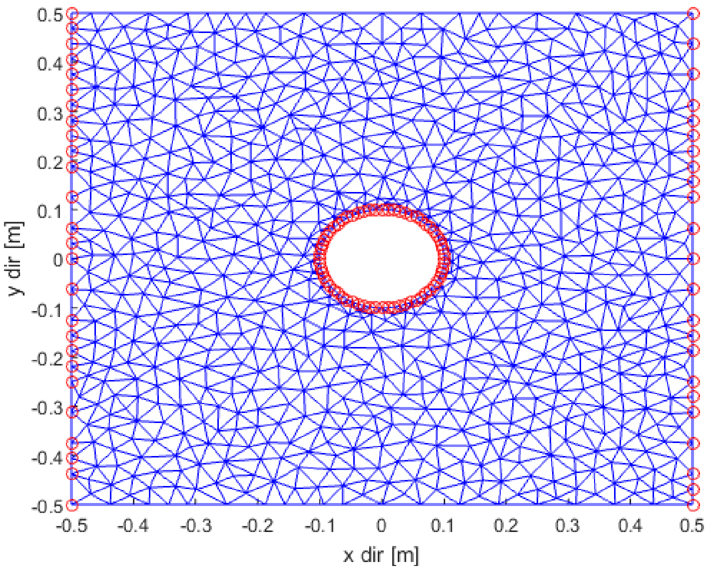
$$f_D(t) = 20 + \begin{cases} 0 & \text{if } t \leq 0 \\ 8t & \text{if } 0 < t < 10 \\ 80 & \text{if } t \geq 10 \end{cases}$$

- No external source is applied

Example #2: definition of geometry, B.C. and mesh

```
S=regions.rect('mu', 237/2700/897)-
   regions.circle([0,0],[0.2,0.2],64);
figure;Sh.draw('e');

Sh.Borders(1).Bc([1,3]) =
   boundaries.neumann(0);
Sh.Borders(2).Bc(:) =
   boundaries.dirichlet(1);
figure;Sh.draw('bc');

Me=mesh2D(S,0.001);
figure;Me.draw('d');
```





Example #2

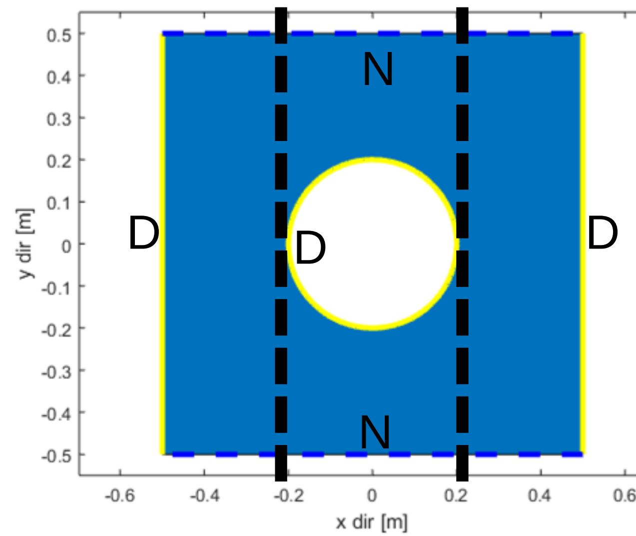
- We need to split vector b into a constant term and a time varying term (for the nodes on the internal border)

$$b(t) = b_{const} + b_{\text{var}}(t) = b_{const} + b_{\text{var}}\big|_{1^\circ} f_D(t)$$

- Moreover, we need to generate a proper vector m_{var} to take into account the contribution from the Mass matrix
- Which nodes are proving a time-varying Dirichlet B.C.? Depending on the geometry and on the B.C.s, many tests are generally possible; in this case it's sufficient to check if |x|<=0.2

```
DirichletNodesCircle=
    Me.find(@(x,y)abs(x)<=0.2,'d');
DirichletNodesExtBorder =
    Me.find(@(x,y)abs(x)>0.2,'d');
```

- NB: additional b contributions are required if different laws are applied to the domain edges and if external forces are applied



Example #2: stiffness matrix

```
function [D,bconst,bvar]=heatEquationVariableDirichlet_BuildStiff(Me)
V=Me.Triangles.Vertices;Areas=Me.Triangles.Areas;Nodes=Me.Nodes;
Dof=Me.Nodes.Dof; numDof = max(Dof);
bconst = zeros(numDof,1); bvar = zeros(numDof,1);
for e=1:N_Tr
    ...
    for ni=1:3
        ii = Dof(V(e,ni));
        if ii > 0
            for nj=1:3
                jj = NI(Tr(e,nj));
                d=c(e)*(Dy(ni)*Dy(nj)+Dx(ni)*Dx(nj))/(4.0*Areas(e)) ;
                if jj > 0, ...
                else %Non homogeneous Dirichlet B.C.
                  val=Me.DirichletNodes(-jj,2);
                  if abs(Me.Nodes.Y(V(e,nj)))<=0.2 %inner circle?
                      bvar(ii) = bvar(ii) - dtmp*val ;
                  else
                      bconst(ii) = bconst(ii) - dtmp*val ;
                  end
                end
            end
        end,    end,    end
```

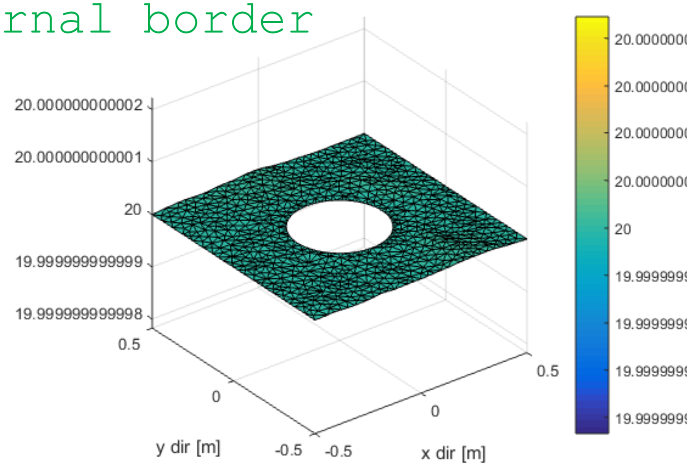
Example #2: mass matrix

```
function [M, mvar] = buildMassVariableDirichlet(Me)
V =Me.Triangles.Vertices; Dof =Me.Nodes.Dof; numDof =max(Dof);
Areas = Me.Triangles.Areas;
mvar = zeros(numDof, 1);
for e = 1:size(V, 1)    %main loop over each triangle
    for ni = 1:3        %for each vertex of this triangle
        ii = Dof(V(e, ni));
        if ii > 0       %is it a degree of freedom? Yes
            for nj = 1:3 %second loop
                jj = Dof(V(e, nj));
                if jj > 0 %add to the mass matrix
                    ...
                else
                    if abs(Me.Nodes.Y(V(e,nj)))<=0.2 %circle?
                        mvar(ii)=mvar(ii)+mtmp;
                    end
                end
            end
        end
    end
end
```

Example #2: stationary solution

- We solve the linear system in order to obtain the initial distribution of the temperature:

```
[D,bconst,bvar]=heatEquationVariableDirichlet_BuildStiff(Me);
T0=20; %initial temperature on the internal border
uStationary0=D\(bconst+bvar*T0);
uu=zeros(size(Me.Nodes.X));
uu(Dof>0)=uStationary0;
uu(DirichletNodesCircle)=T0;
uu(DirichletNodesExtBorder)=20;
```



- Since all the Dirichlet edges have a T=20°C and there is no external contribution, the temperature in each node of the domain is exactly 20°C.

Example #2: temporal evolution

- Using the Implicit Euler method:

$$M\frac{\partial u}{\partial t}+m_{\mathrm{var}}\frac{\mathrm{d}f_D}{\mathrm{d}t}+Du=b(t)\rightarrow$$

$$(M+D\Delta t)u^{k+1}=Mu^k+\Delta t\left(b_{\mathrm{const}}+b_{\mathrm{var}}f_D{}^{k+1}\right)-m_{\mathrm{var}}\left(f_D{}^{k+1}-f_D{}^{k}\right)$$

- Therefore:

```
[M, mvar] = buildMassVariableDirichlet(Me);
u=uStationary0; dt=1; A=(M+D*dt);
TCircleOld=fDirichlet(0);
for k=1:Tend/dt,
   TCircle=fDirichlet(k*dt); DeltaT=Tcircle-TCircleOld;
   u=A\(M*u+dt*(bconst+bvar*TCircle)-mvar*DeltaT);
   uu(Dof)=u;
   uu(DirichletNodesCircle)=TCircle;
   TCircleOld=TCircle;
end
```

Temporal analysis of elliptic problems

- Let's consider the temporal evolution of the elastic membrane (elliptic problem)

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} + \mu \Delta u + \sigma u = f \\ u(0) = u_0 \\ u'(0) = u_1 \end{cases}$$

- We obtain the following system $M\ddot{u}(t_n) + (D + R)u(t_n) = f(t_n)$ with M mass matrix, D diffusion term and R reaction term

- For simplicity, we assume constant B.C.s

Example: script RunDt

- We study the evolution of the position of a squared membrane placed in a viscous fluid, starting from a zero displacement, when a constant force is applied from t=0

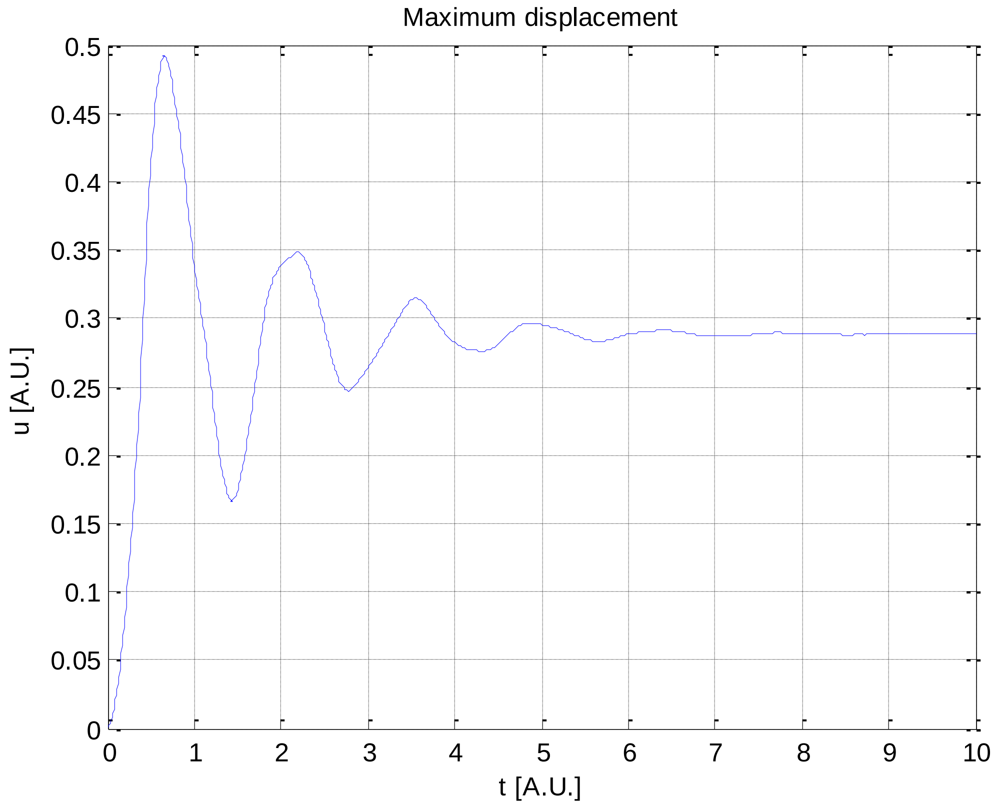
```
Sh = regions.rect('mu',1);
Me = mesh2D(Sh);
f = @(x,y)-4*ones(size(x));
[D,b] = dirichletHomo_BuildStiff(Me,f);
M = buildMass(Me);
uu = Me.copyToAllNodes(u, pcg(A,b,1e-3,250));
%well known stationary solution
figure;
Me.draw(uu,'hidemesh');

Dofs = Me.Nodes.Dof>0;
```

Example: script RunDt

```
%Numerical evolution, dumped oscillation
dt = 0.01;             %time steps
Tend = 10;             %simulation end time
NumIter = ceil(Tend/dt); %number of steps
displ=zeros(NumIter,1); %to store the max displacement
u0 = zeros(size(b)); %null initial displacement
u1 = u0;
a = 1.5;                %fluid friction coefficient
for i = 1:NumIter
   [u2,flag]=pcg(M*(1+a*dt),M*((2+a*dt)*u1-u0)-
      dt^2*(D*u1-b),[],30,[],[],u1);
   u0=u1; u1=u2;
   uu(Dofs)=u2;
   Me.draw(uu,'hidemesh'); zlim([-1 1]);
   drawnow; %to force a refresh of the figure,
   %otherwise it is updated at the end of the loop
   displ(i)=max(abs(u2)); %save the maximum displac.
end
```

Example: script RunDt

```
figure;
plot((1:NumIter)*dt, displ);
title('Maximum displacement');
xlabel('t [A.U.]');
ylabel('u [A.U.]');
grid on;
```



Summary of BuildStiff/Mass functions

File	**Boundary conditions**
dirichletHomo_BuildStiff	Homogeneous Dirichlet, homogeneous Neumann
dirichletNonHomo_BuildStiff	Non homogeneous Dirichlet, homogeneous Neumann
neumannNonHomo_BuildStiff	Homogeneous Dirichlet, non homogeneous Neumann
robin_BuildStiff	Homogeneous Dirichlet, homogeneous Neumann, Robin
dirichletHomo_DiffTransReact_BuildStiff	Homogeneous Dirichlet, homogeneous Neumann
periodic_BuildStiff	Homogeneous Dirichlet, homogeneous Neumann, periodic
coupledDirichlet_BuildStiff / coupledNeumann_BuildStiff	Non homogeneous Dirichlet, homogeneous Neumann
buildMass / buildMassLumping	-