



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

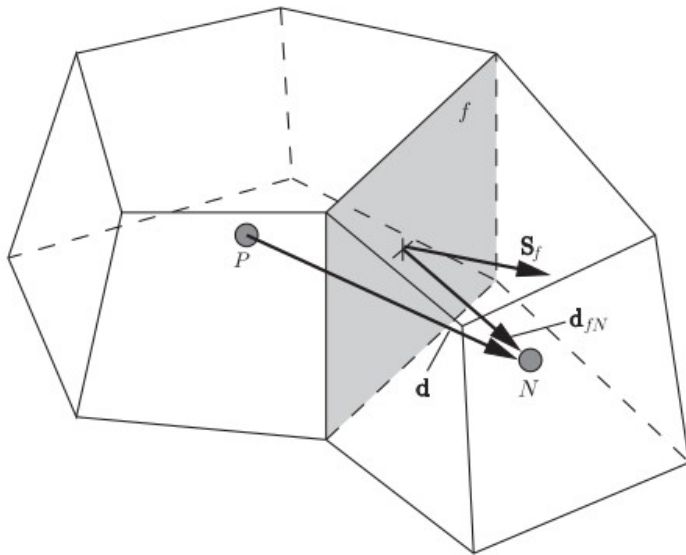
Numerical Heat Transfer
for Applications

**FVM for steady state
heat conduction**

Dr Valerio D'Alessandro

Finite volume discretization of Laplace equation

The method is based on discretising the integral form of governing equations over each control volume.



$$\nabla^2 T = 0 \implies \int_{V_P} \nabla^2 T dV = 0$$

$$\int_{V_P} \nabla^2 T dV = \int_{V_P} \nabla \cdot (\nabla T) dV$$



$$\int_{V_P} \nabla \cdot (\nabla T) dV = \int_{\partial V_P} \nabla T \cdot d\mathbf{S} \simeq \sum_f (\nabla T)_f \cdot \mathbf{S}_f$$

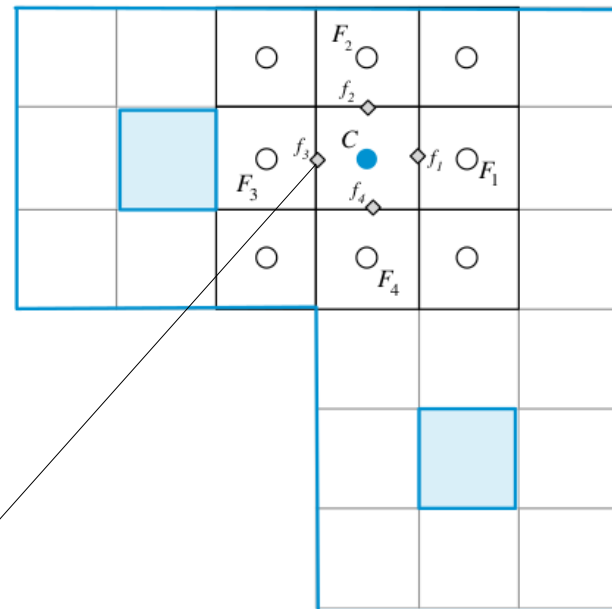
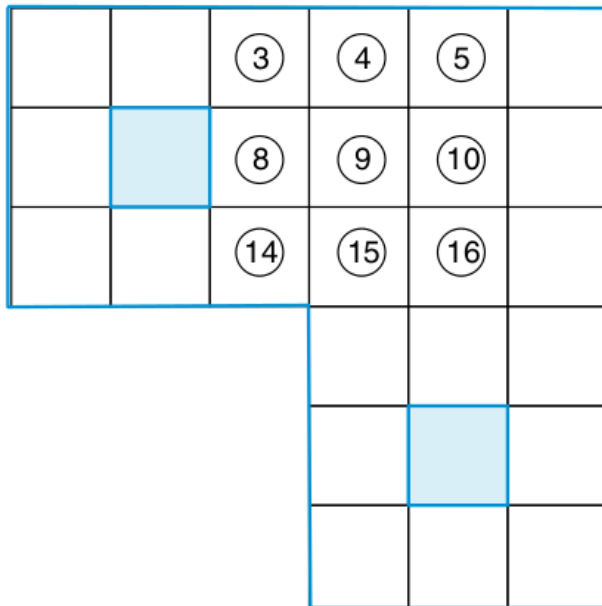


$$\sum_f (\nabla T)_f \cdot \mathbf{S}_f = 0$$

$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P) dV = \mathbf{0}$$

Finite volume discretization of Laplace equation

$$\sum_f (\nabla T)_f \cdot \mathbf{S}_f = 0$$



$$(\nabla T)_{f_1} \cdot \mathbf{S}_{f_1} + (\nabla T)_{f_2} \cdot \mathbf{S}_{f_2} + (\nabla T)_{f_3} \cdot \mathbf{S}_{f_3} + (\nabla T)_{f_4} \cdot \mathbf{S}_{f_4} = 0$$

Finite volume discretization of Laplace equation

$$\sum_f (\nabla T)_f \cdot \mathbf{S}_f = 0$$

$$(\nabla T)_{f_1} \cdot \mathbf{S}_{f_1} + (\nabla T)_{f_2} \cdot \mathbf{S}_{f_2} + (\nabla T)_{f_3} \cdot \mathbf{S}_{f_3} + (\nabla T)_{f_4} \cdot \mathbf{S}_{f_4} = 0$$

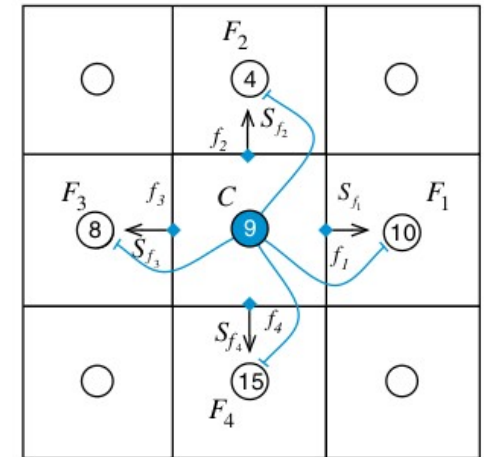


$$\frac{T_{F_1} - T_C}{|\mathbf{F}_1 - \mathbf{C}|} |\mathbf{S}_{f_1}| + \frac{T_{F_2} - T_C}{|\mathbf{F}_2 - \mathbf{C}|} |\mathbf{S}_{f_2}| + \frac{T_{F_3} - T_C}{|\mathbf{F}_3 - \mathbf{C}|} |\mathbf{S}_{f_3}| + \frac{T_{F_4} - T_C}{|\mathbf{F}_4 - \mathbf{C}|} |\mathbf{S}_{f_4}| = 0$$



$$-\left(\frac{|\mathbf{S}_{f_1}|}{|\mathbf{d}_1|} + \frac{|\mathbf{S}_{f_2}|}{|\mathbf{d}_2|} + \frac{|\mathbf{S}_{f_3}|}{|\mathbf{d}_3|} + \frac{|\mathbf{S}_{f_4}|}{|\mathbf{d}_4|} \right) T_C + \frac{|\mathbf{S}_{f_1}|}{|\mathbf{d}_1|} T_{F_1} + \frac{|\mathbf{S}_{f_2}|}{|\mathbf{d}_2|} T_{F_2} + \frac{|\mathbf{S}_{f_3}|}{|\mathbf{d}_3|} T_{F_3} + \frac{|\mathbf{S}_{f_4}|}{|\mathbf{d}_4|} T_{F_4} = 0$$

$$a_C T_C + \sum_{nb} a_{nb} T_{nb} = 0$$



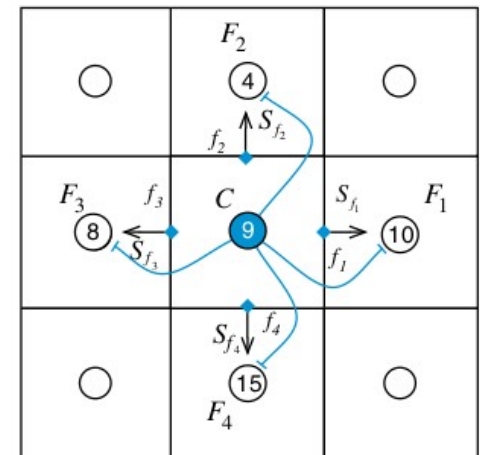
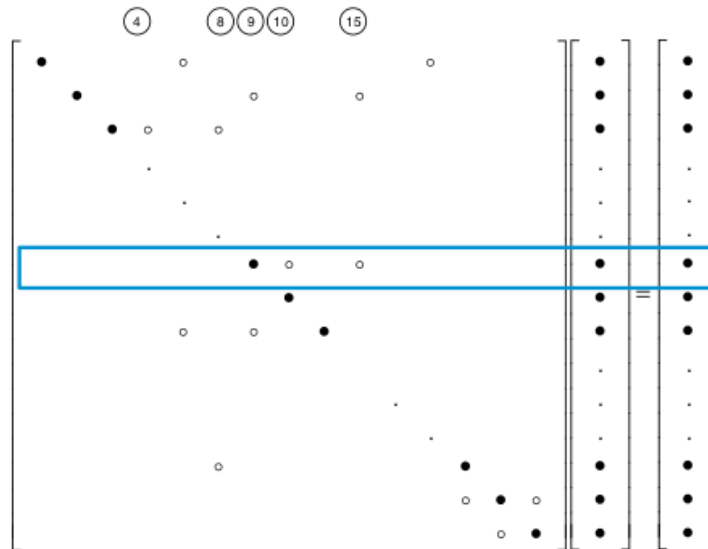
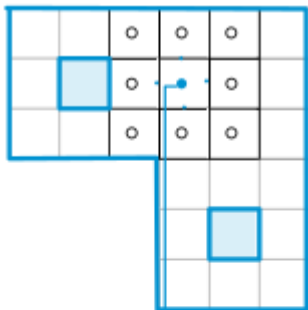
Finite volume discretization of Laplace equation

$$a_C T_C + \sum_{nb} a_{nb} T_{nb} = 0$$

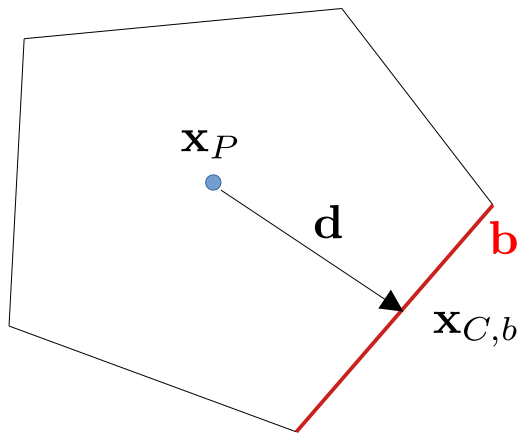
$$a_C T_C + a_{F_1} T_{F_1} + a_{F_2} T_{F_2} + a_{F_3} T_{F_3} + a_{F_4} T_{F_4} = 0$$

The above equation can be re-written using global element assembly.


$$a_9 T_9 + a_{10} T_{10} + a_4 T_4 + a_8 T_8 + a_{15} T_{15} = 0$$



Finite volume discretization of Laplace equation – boundary faces



$$\int_{\partial V_P} \nabla T \cdot d\mathbf{S} \simeq \sum_f (\nabla T)_f \cdot \mathbf{S}_f$$


 $(\nabla T)_b \cdot \mathbf{S}_b$

Boundary contribution can be included to faces' flux in two different ways: *fixed temperature value* or *fixed gradient*.

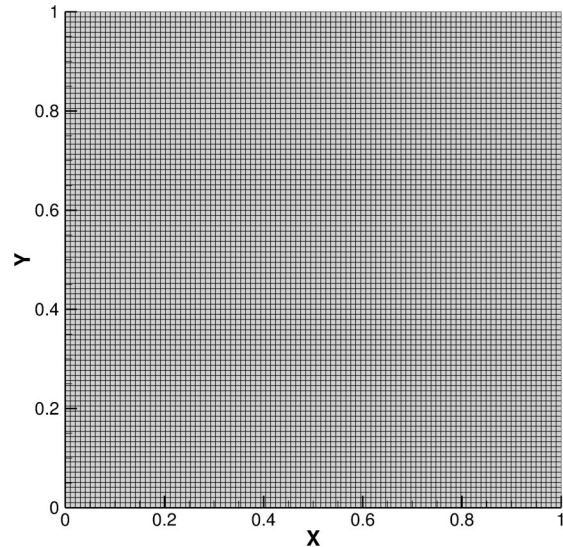
$$(\nabla T)_b \cdot \mathbf{S}_b \Rightarrow (\nabla T)_b \cdot \mathbf{S}_b = |\mathbf{S}_b| \frac{T_b - T_P}{|d|}$$

$$a_C T_C + \sum_{nb} a_{nb} T_{nb} = 0$$

$$a_C T_C + \underbrace{\sum_{nb} a_{nb} T_{nb}}_{\text{Boundary conditions provide RHS terms}} = b_C$$

Boundary conditions provide RHS terms

Ex2a_FVM – Steady state heat transfer in 1-D configuration



$$\nabla^2 T = 0$$

$$T(x = 0) = 1$$

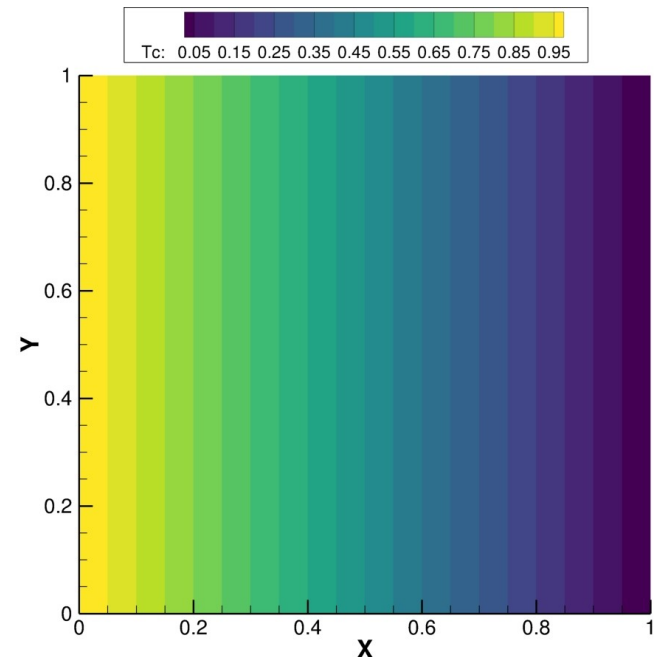
$$T(x = 1) = 1$$

$$\partial_n T(y = 0) = 0$$

$$\partial_n T(y = 1) = 0$$



$$T(x) = 1 - \frac{x}{L}$$



Ex2a_FVM – Steady state heat transfer in 1-D configuration

Matlab code

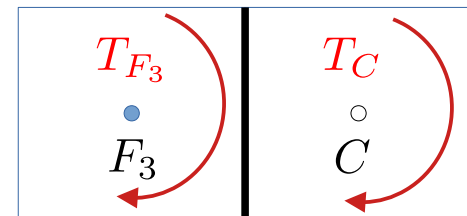
```
format long e

caseName='quad_100x100';
m=mesh_data(caseName);

Nc=m.numberOfElements;
Ni=m.numberOfInteriorFaces;
Nb=m.numberOfBElements;
A=zeros(Nc,Nc);

for i=1:Nc
    cp = m.elements(i).Cc;
    for j=1:length(m.elements(i).iNeighbours)
        iN = m.elements(i).iNeighbours(j);
        cn = m.elements(iN).Cc;
        dPN = cn - cp;
        mdPN = sqrt(sum(dPN.*dPN));
        for ii=1:length(m.elements(i).iFaces)
            fid = m.elements(i).iFaces(ii);
            for jj=1:length(m.elements(iN).iFaces)
                fjd = m.elements(iN).iFaces(jj);
                if ( fjd<=Ni && fid == fjd)
                    A(i,i) = A(i,i) - m.faces(fid).Af/mdPN ;
                    A(i,iN) = m.faces(fid).Af/mdPN;
                end
            end
        end
    end
end
end
```

$\text{fid} == \text{fjd}$



$$a_C T_C + \sum_{nb} a_{nb} T_{nb} = 0$$

\downarrow \downarrow
 $A(i,i)$ $A(i,iN)$

Ex2a_FVM – Steady state heat transfer in 1-D configuration

Matlab code

```
b=zeros(Nc,1);
ib = m.boundaries(4).startFace;
nb = m.boundaries(4).numberOfBFaces;
Tb = 1.0;
for i=ib:ib+nb-1
    iN      = m.faces(i).iOwner;
    cf      = m.faces(i).Cf;
    cP      = m.elements(iN).Cc;
    d       = cf - cP;
    magd    = sqrt(sum(d.*d));
    A(iN,iN) = A(iN,iN) - m.faces(i).Af/magd;
    b(iN)    = -Tb*m.faces(i).Af/magd ;
end
```

```
ib = m.boundaries(2).startFace;
nb = m.boundaries(2).numberOfBFaces;
Tb = 0.0;
for i=ib:ib+nb-1
    iN      = m.faces(i).iOwner;
    cf      = m.faces(i).Cf;
    cP      = m.elements(iN).Cc;
    d       = cf - cP;
    magd    = sqrt(sum(d.*d));
    A(iN,iN) = A(iN,iN) - m.faces(i).Af/magd;
    b(iN)    = -Tb*m.faces(i).Af/magd ;
end
```

$$a_C T_C + \sum_{nb} a_{nb} T_{nb} = b_C$$



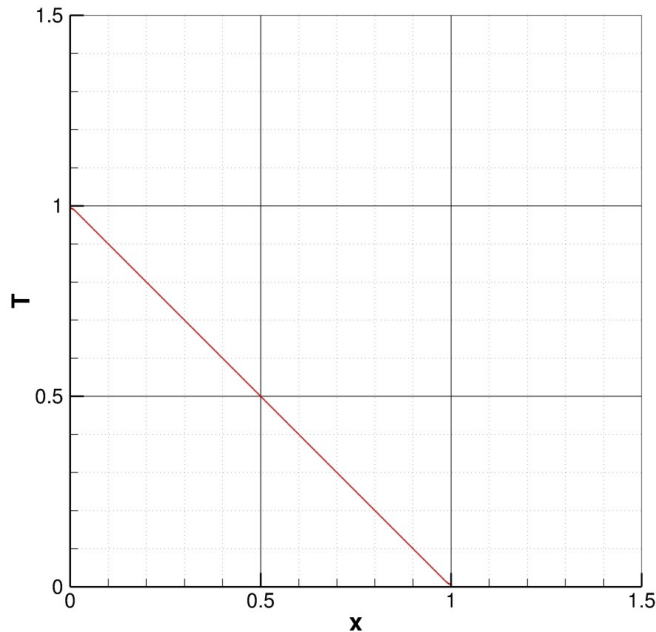
$$(\nabla T)_b \cdot \mathbf{S}_b = |\mathbf{S}_b| \frac{T_b - T_P}{|\mathbf{d}|}$$

$$\underbrace{a_C T_C - \frac{|\mathbf{S}_b|}{|\mathbf{d}|} T_C}_{A(iN, iN)} + \sum_{nb} a_{nb} T_{nb} = \underbrace{-\frac{|\mathbf{S}_b|}{|\mathbf{d}|} T_b}_{b(iN)}$$

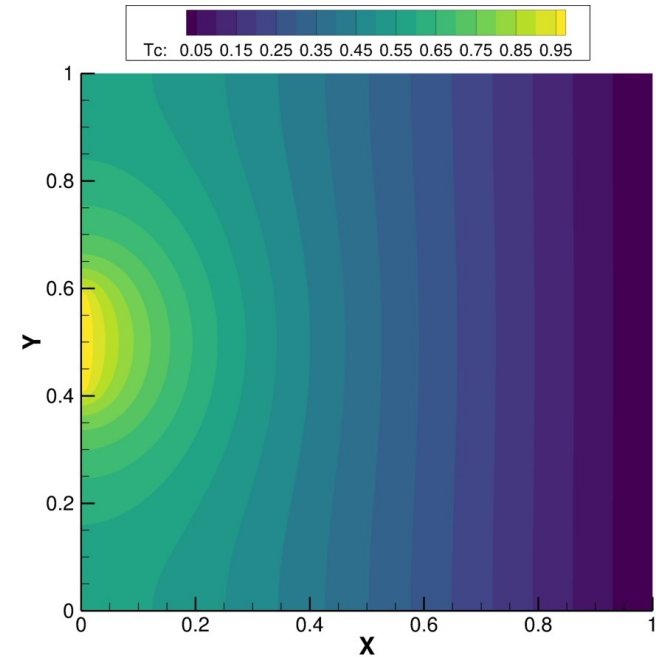
Ex2a_FVM – Steady state heat transfer in 1-D configuration

Matlab code

```
Tc = pcg(-A,-b,1e-12,1000);
% Tc = bicg(A,b,1e-12,1000);
wrtfld(1, m , Tc , 'Tc', caseName)
```



The solution is exactly the same of the analytic one.



1D solution is obtained fixing $T_b = 1.0$ to boundaries 4 and 5. Differently on boundary 2 we set $T_b = 0.0$.

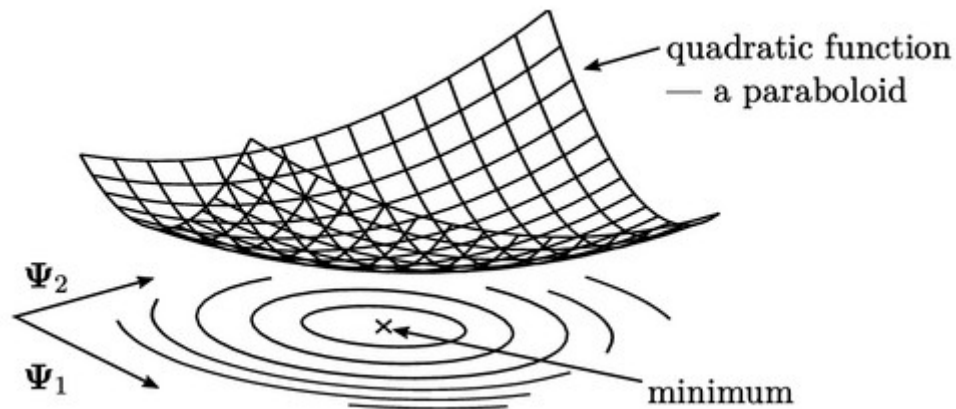
Commenting in the code the T_b implementation for bnd 5 we obtain a classical 2D steady state heat conduction problem.

Linear systems

Descent methods provide alternative **efficient matrix solvers** that are often used in CFD. Basically, descent methods represent the equations which are being solved as a minimisation problem:

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{x}^T \mathbf{b} \quad \leftarrow \quad \nabla f(\mathbf{x}) = \mathbf{A} \mathbf{x} - \mathbf{b}$$

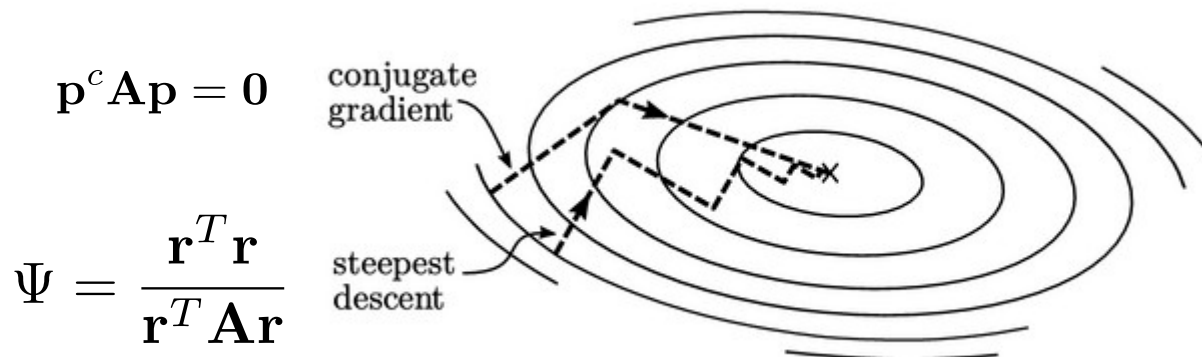
Equating the gradient of the function f to zero corresponds to a minimum in the quadratic function. At the same time, it is the solution to $\mathbf{A}\mathbf{x}=\mathbf{b}$. The method is therefore concerned with finding the minimum of the quadratic form efficiently.



Linear systems

The search for the minimum involves a series of updates of the form: $\mathbf{x} = \mathbf{x}^c + \Psi \mathbf{p}$
the column vector \mathbf{p} provides the direction of the line search towards the minimum; the scalar Ψ provides the magnitude of the line in that direction.

- **Steepest descent:** the intuitive way to reach the minimum is to follow the direction of steepest descent. The method is fairly simple because the direction is defined by the negative of gradient of the quadratic form f . The method reaches the minimum point along a zigzag path where consecutive directions are orthogonal. The directions do not change and result in a larger number of very short steps, the more the paraboloid is stretched in one direction.
- The **conjugate gradient (CG)** method chooses search directions that are conjugate. This means each new direction corresponds to the previous one, following the equation reported in the figure



Linear systems

The conjugate gradient (CG) method was designed for symmetric matrices. The **bi-conjugate gradient (BiCG)** is the CG variant usefull for non-symmetric matrices.

```
Tc = bicg(A,b,1e-12,1000);
```

```
x = bicg(A,b,tol,maxit,M) specifies a preconditioner matrix M
```

Preconditioning is another important element in the solution of linear system. Indeed, preconditioning matrix represent a specific matrix which is used to generate an equivalent problem from the starting one which easier to be solved:

$$\mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad (\mathbf{MA}) \mathbf{x} = \mathbf{Mb}$$

Possible preconditioning strategies also avaible within Matlab/Octave are the following:

LU (Lower Upper factorization) $\mathbf{M} = \mathbf{LU}$
Cholesky $\mathbf{M} = \mathbf{LL}^T$

lu
ichol

Linear systems

The conjugate gradient (CG) method was designed for symmetric matrices. The **bi-conjugate gradient (BiCG)** is the CG variant usefull for non-symmetric matrices.

```
Tc = bicg(A,b,1e-12,1000);
```

```
x = bicg(A,b,tol,maxit,M) specifies a preconditioner matrix M
```

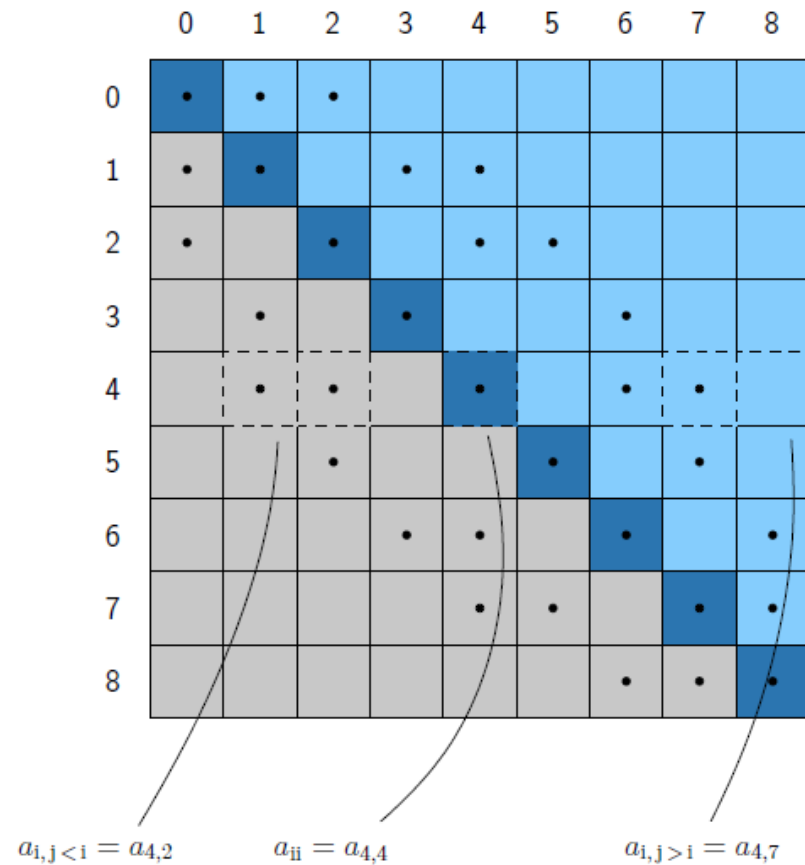
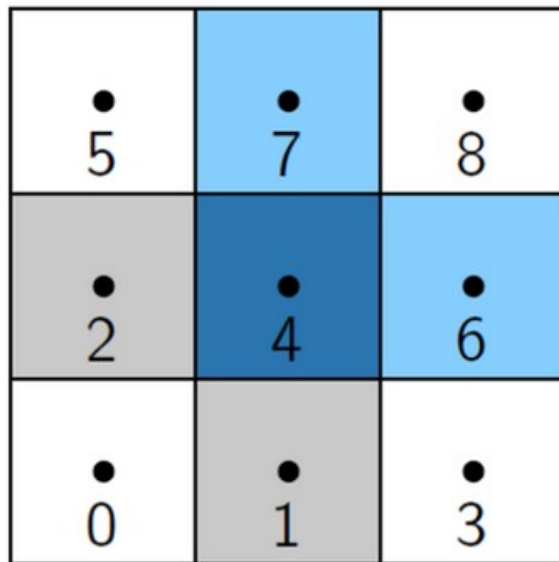
Preconditioning is another important element in the solution of linear system. Indeed, preconditioning matrix represent a specific matrix which is used to generate an equivalent problem from the starting one which easier to be solved:

$$\mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad (\mathbf{MA}) \mathbf{x} = \mathbf{Mb}$$

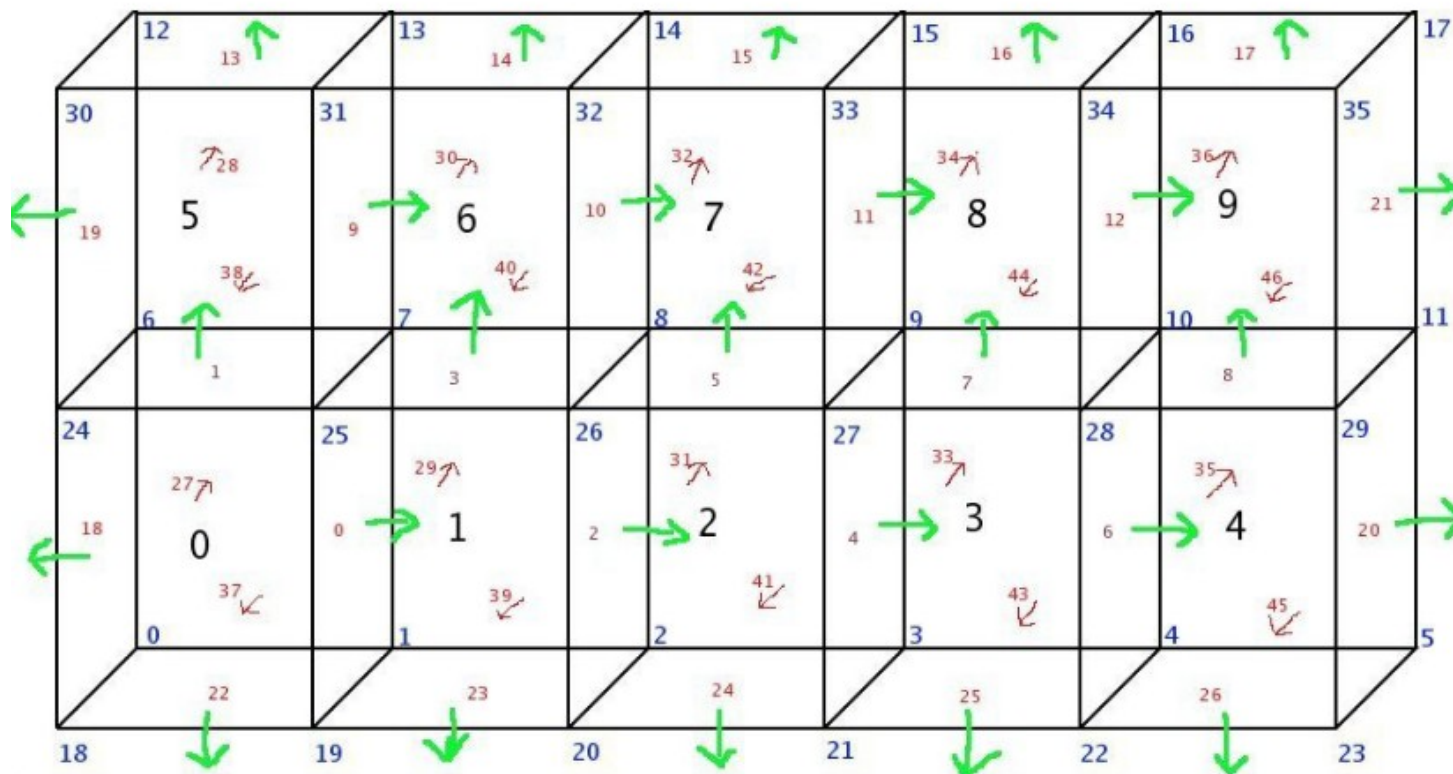
```
[L,U]=lu(A);
```

```
Tc = pcg(A,b,1e-12,1000,L, U);
```

Ex2a.1_FVM - Steady state heat transfer - matrix sparsity handling



Ex2a.1_FVM - Steady state heat transfer - matrix sparsity handling



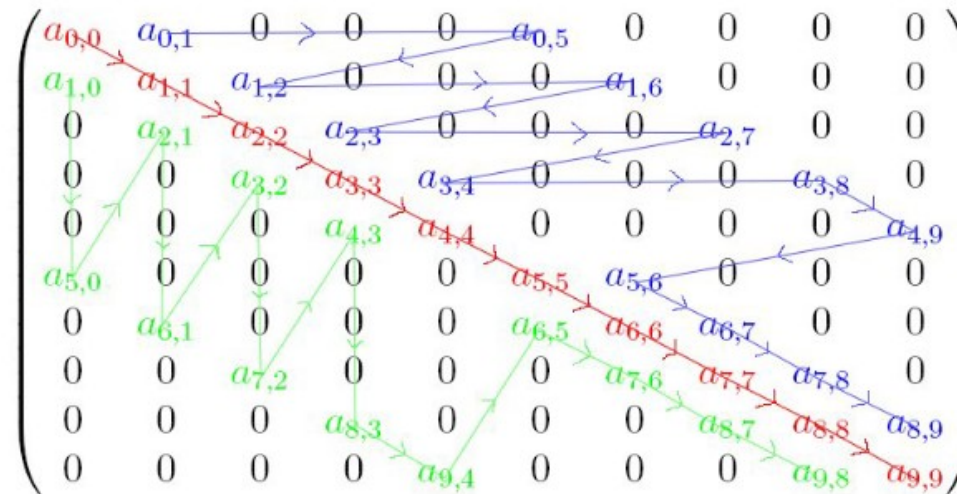
Ex2a.1_FVM – Steady state heat transfer – matrix sparsity handling

- 10 volumes/cells
- 36 nodes/corners
- 47 faces, 13 internal faces

$$\begin{pmatrix} a_{0,0} & a_{0,1} & 0 & 0 & 0 & a_{0,5} & 0 & 0 & 0 & 0 \\ a_{1,0} & a_{1,1} & a_{1,2} & 0 & 0 & 0 & a_{1,6} & 0 & 0 & 0 \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & a_{2,7} & 0 & 0 \\ 0 & 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & a_{3,8} & 0 \\ 0 & 0 & 0 & a_{4,3} & a_{4,4} & 0 & 0 & 0 & 0 & a_{4,9} \\ a_{5,0} & 0 & 0 & 0 & 0 & a_{5,5} & a_{5,6} & 0 & 0 & 0 \\ 0 & a_{6,1} & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} & 0 & 0 \\ 0 & 0 & a_{7,2} & 0 & 0 & 0 & a_{7,6} & a_{7,7} & a_{7,8} & 0 \\ 0 & 0 & 0 & a_{8,3} & 0 & 0 & 0 & a_{8,7} & a_{8,8} & a_{8,9} \\ 0 & 0 & 0 & 0 & a_{9,4} & 0 & 0 & 0 & a_{9,8} & a_{9,9} \end{pmatrix}$$

Ex2a.1_FVM - Steady state heat transfer - matrix sparsity handling

- The order of accessing matrix elements in OpenFOAM is as follows:



lower(), diag(), upper()

Owners define matrix rows, while neighbours are adopted to define the matrix columns

Ex2a.1_FVM – Steady state heat transfer – matrix sparsity handling

Matlab code

```
Nc=m.numberOfElements;
Ni=m.numberOfInteriorFaces;
Nb=m.numberOfBElements;
b=zeros(Nc,1);

i0r = zeros(Ni,1);
i0c = zeros(Ni,1);
v = zeros(Ni,1);
%
for i=1:Ni
    i0r(i) = m.faces(i).i0owner;
    i0c(i) = m.faces(i).iNeighbour;
%
    cp = m.elements(i0r(i)).Cc;
    cn = m.elements(i0c(i)).Cc;
    dPN = cn - cp;
    mdPN = sqrt(sum(dPN.*dPN));
    v(i) = m.faces(i).Af/mdPN;
end
%
S = sparse(i0r',i0c',v,Nc,Nc);
S = S + S';
%
```

```
diag = zeros(Nc,1);
%
for i=1:Nc
%
    cp = m.elements(i).Cc;
    for j=1:length(m.elements(i).iNeighbours)
        iN = m.elements(i).iNeighbours(j);
        cn = m.elements(iN).Cc;
        dPN = cn - cp;
        mdPN = sqrt(sum(dPN.*dPN));
        for ii=1:length(m.elements(i).iFaces)
            fid = m.elements(i).iFaces(ii);
            for jj=1:length(m.elements(iN).iFaces)
                fjd = m.elements(iN).iFaces(jj);
                if ( fjd<=Ni && fid == fjd)
                    diag(i) = diag(i) -
                        m.faces(fid).Af/mdPN;
                end
            end
        end
    end
end
end
```

Ex2a.1_FVM - Steady state heat transfer - matrix sparsity handling

Matlab code

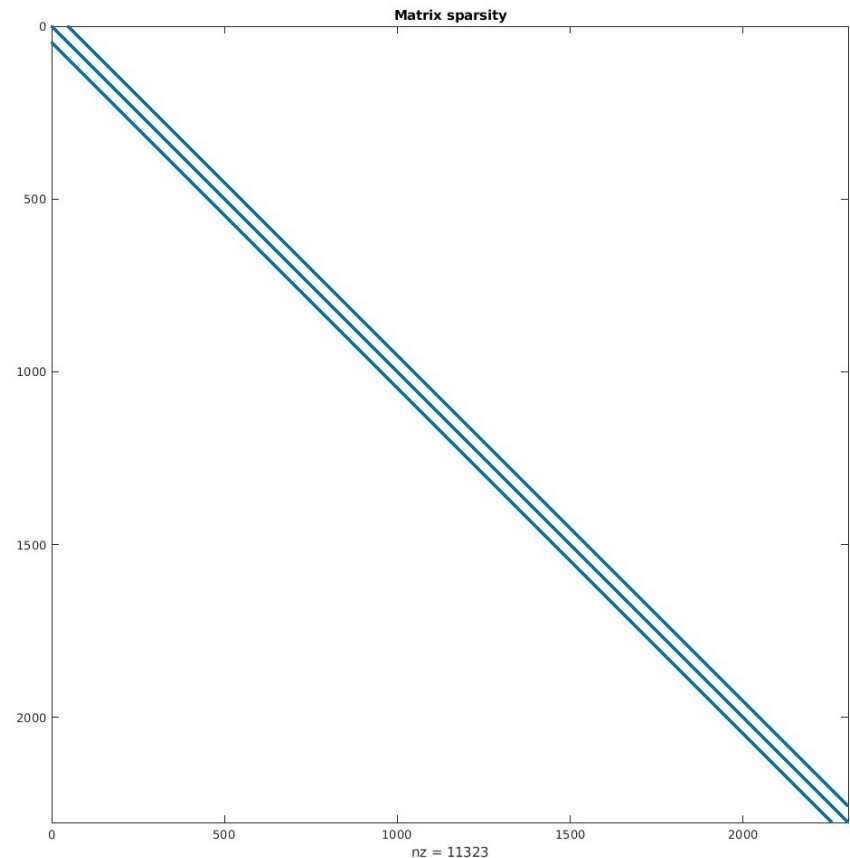
```

ib = m.boundaries(2).startFace;
nb = m.boundaries(2).numberOfBFaces;
Tb = 0.0;
for i=ib:ib+nb-1
    iN      = m.faces(i).iOwner;
    cf      = m.faces(i).Cf;
    cP      = m.elements(iN).Cc;
    d       = cf - cP;
    magd    = sqrt(sum(d.*d));
    diag(iN) = diag(iN) - m.faces(i).Af/magd;
    b(iN)   = -Tb*m.faces(i).Af/magd ;
end

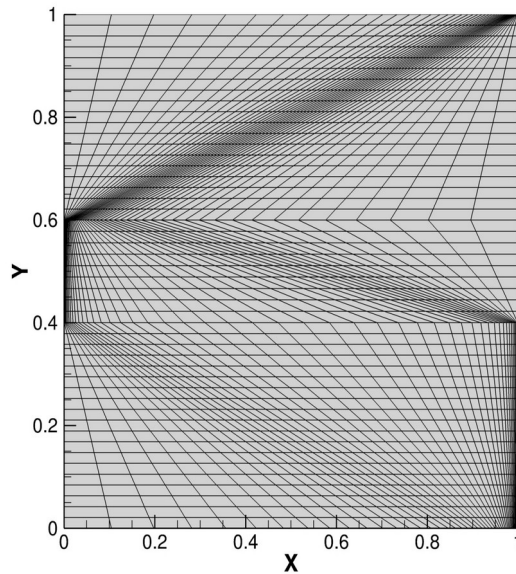
S = S + spdiags(diag,0,Nc,Nc) ;
bsp = sparse(b);

Tc = bicg(S,bsp,1e-12,1000);
%
wrtfld(1, m , Tc , 'Tc', caseName)
%
%-----
%
figure
spy(S)
title('Matrix sparsity')

```



Ex2b_FVM - Steady state heat transfer on non orthogonal grid



$$\nabla^2 T = 0$$

$$T(x = 0) = 1$$

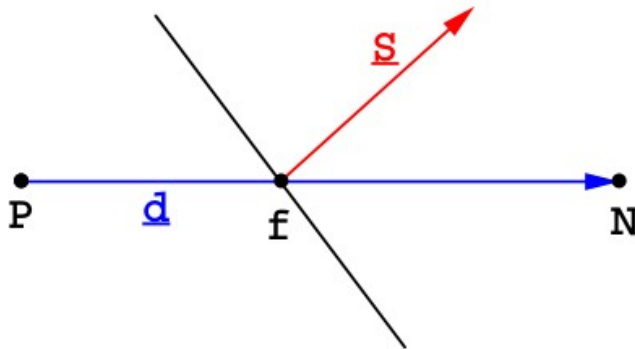
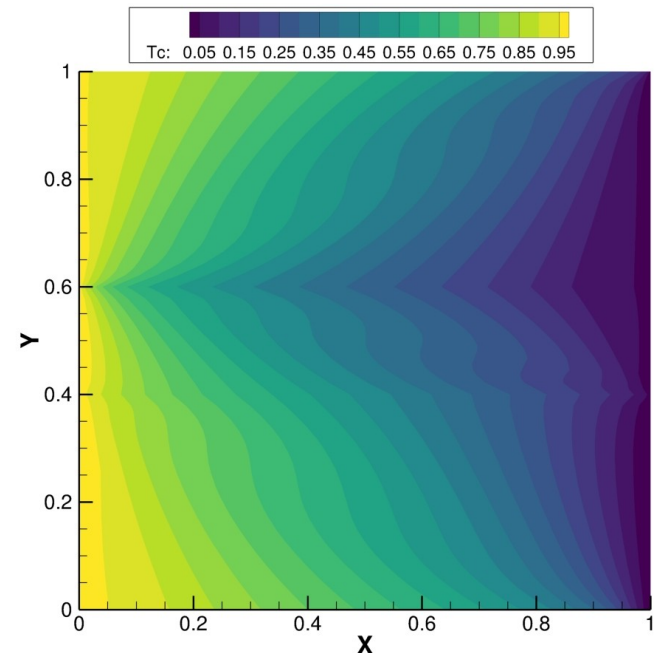
$$T(x = 1) = 1$$

$$\partial_n T(y = 0) = 0$$

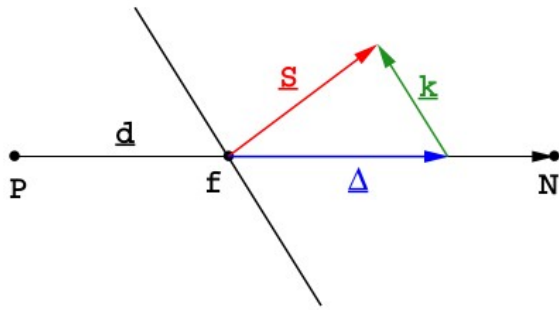
$$\partial_n T(y = 1) = 0$$



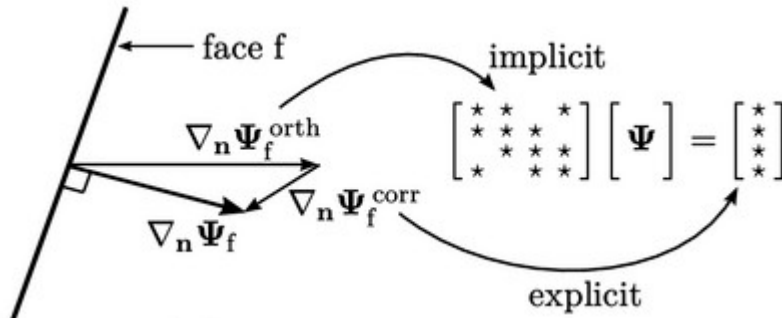
$$T(x) = 1 - \frac{x}{L}$$



Ex2b_FVM - Steady state heat transfer on non orthogonal grid

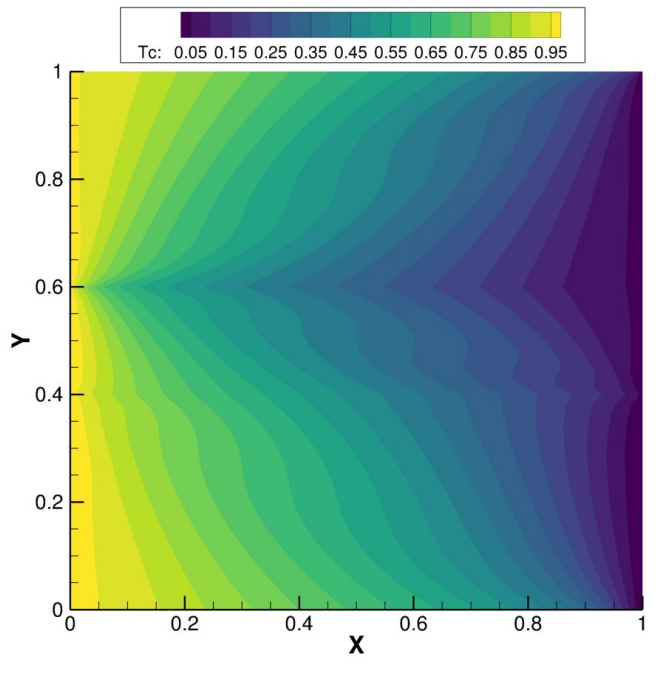


$$(\nabla T)_f \cdot \mathbf{S}_f = |\Delta| \frac{T_N - T_P}{|d|} + \mathbf{k} \cdot (\nabla T)_f$$

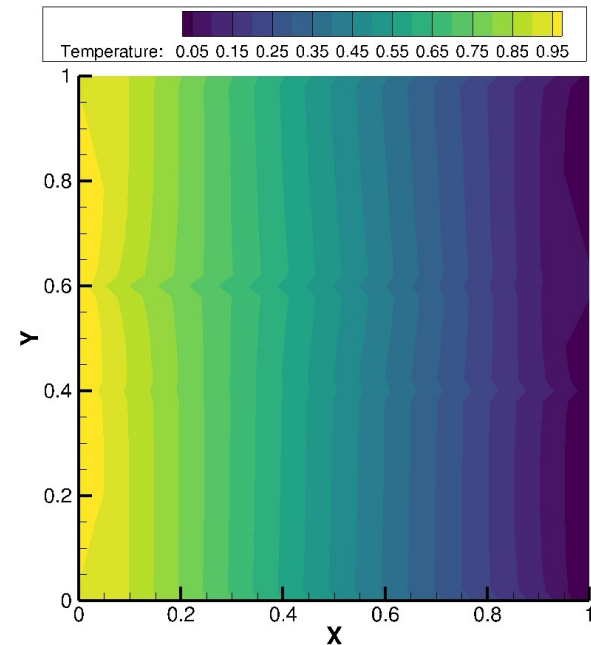


The non-Orthogonal correction is **explicit**, i.e. calculated using known values of T . Thus, the gradient flux need updating within an iterative sequence to maintain accuracy.

Ex2b_FVM – Steady state heat transfer on non orthogonal grid

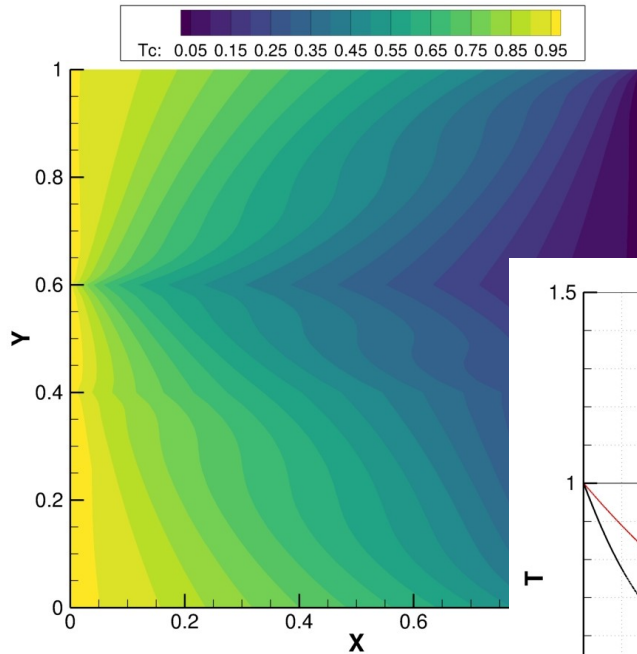


Solution **without**
non-Orthogonal corrections

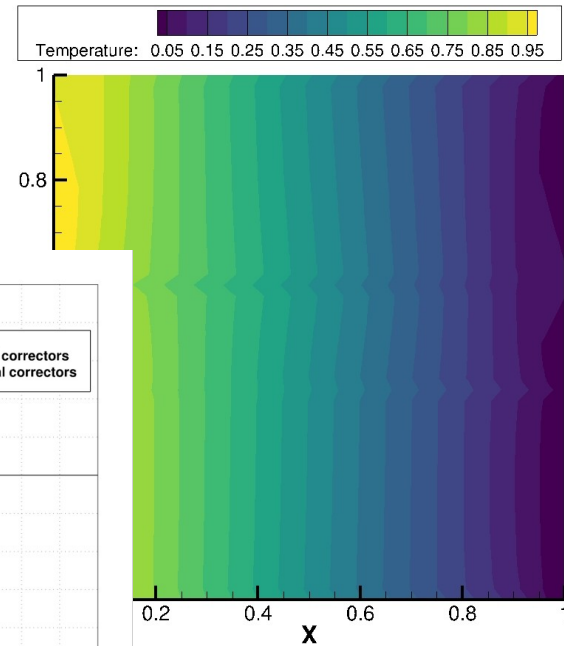
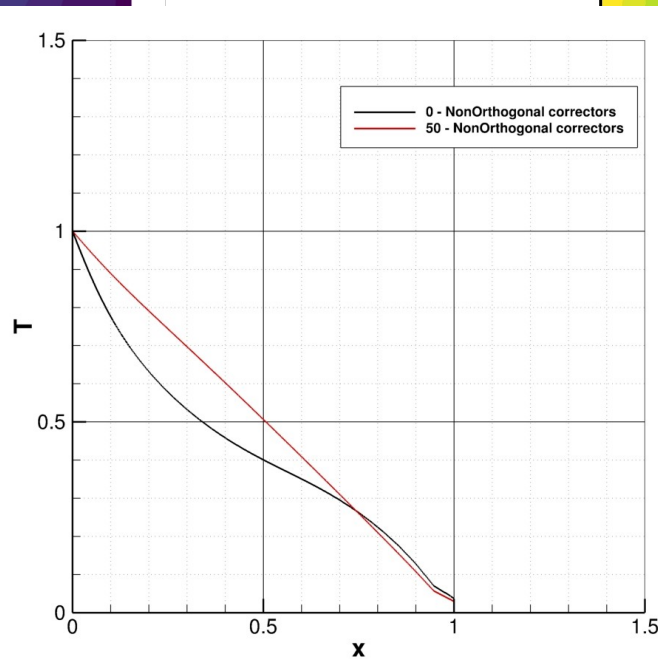


Solution **with 50**
non-Orthogonal correctors
(OpenFOAM solver)

Ex2b_FVM – Steady state heat transfer on non orthogonal grid



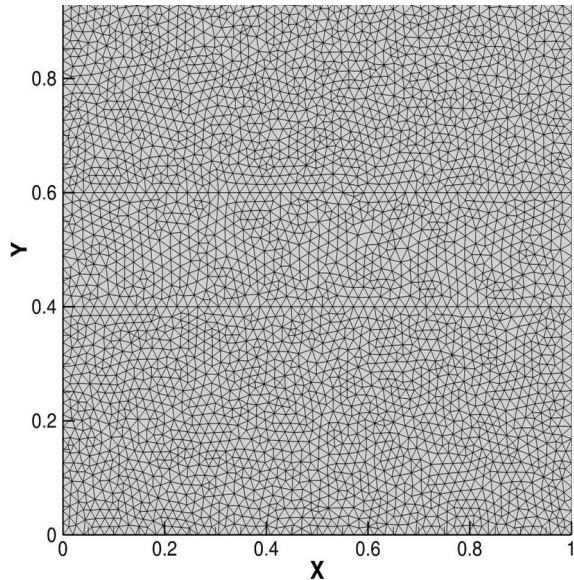
Solution **without**
non-Orthogonal correc



tion **with 50**
Orthogonal correctors
(nFOAM solver)

Without non-Orthogonal correctors the solution loses even its physical meaning

Ex2c_FVM – Steady state heat transfer on triangular cells grid



$$\nabla^2 T = 0$$

$$T(x = 0) = 1$$

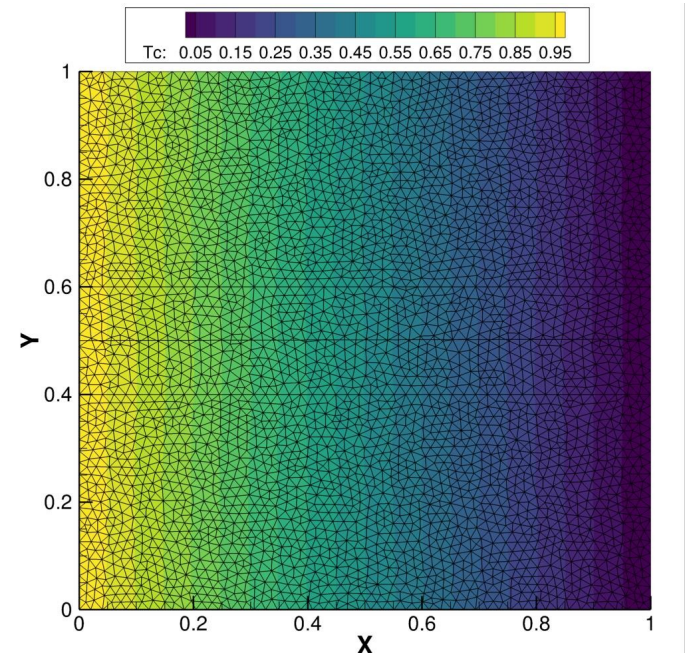
$$T(x = 1) = 1$$

$$\partial_n T(y = 0) = 0$$

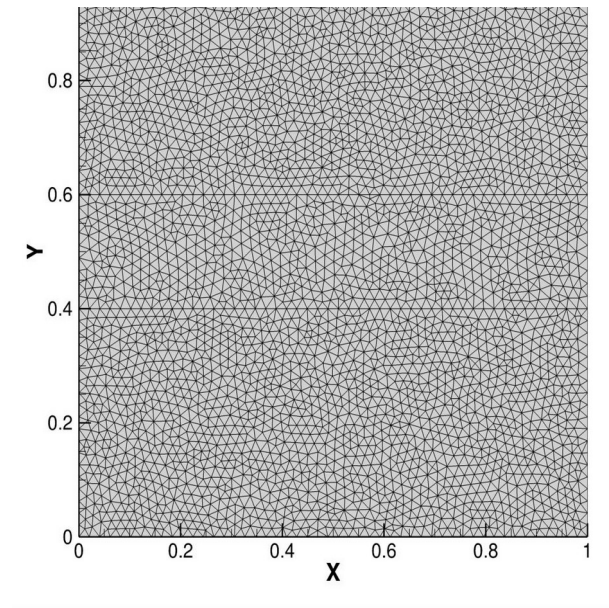
$$\partial_n T(y = 1) = 0$$



$$T(x) = 1 - \frac{x}{L}$$



Ex2c_FVM – Steady state heat transfer on triangular cells grid



$$\nabla^2 T = 0$$

$$T(x = 0) = 1$$

$$T(x = 1) = 1$$

$$\partial_n T(y = 0) = 0$$

$$\partial_n T(y = 1) = 0$$



$$T(x) = 1 - \frac{x}{L}$$

