SimpleFEM

an introduction to the Q1P0 element and its implementation

C. Thieulot

May 2011

Warning

The following presentation is about the Finite Element Method but it isn't

- comprehensive

Warning

The following presentation is about the Finite Element Method but it isn't

- comprehensive
- mathematically sound

Warning

The following presentation is about the Finite Element Method but it isn't

- comprehensive
- mathematically sound
- about Garfield at all.

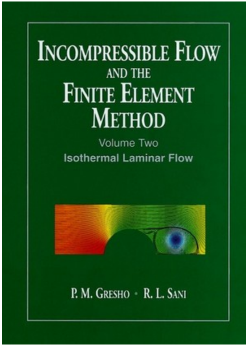
Warning

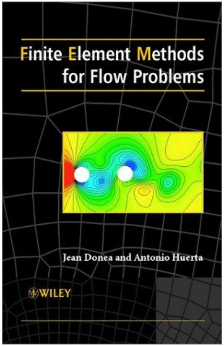
The following presentation is about the Finite Element Method but it isn't

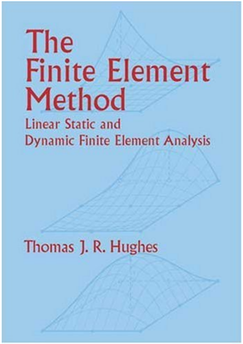
- comprehensive
- mathematically sound
- about Garfield at all.

$\rightarrow$ focus on incompressible Stokes flow, Q1P0 element

Literature


INCOMPRESSIBLE FLOW
AND THE
FINITE ELEMENT
METHOD
Volume Two
Isothermal Laminar Flow
P. M. GRESHO · R. L. SANI


Finite Element Methods
for Flow Problems
Jean Donea and Antonio Huerta
WILEY


The
Finite Element
Method
Linear Static and
Dynamic Finite Element Analysis
Thomas J. R. Hughes

The physics

The mechanical behavior of Earth materials which compose the crust and the mantle is described by means of the Stokes equation :

$$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0}$$

Solenoidal constraint due to the **incompressibility** condition :

$$\boldsymbol{\nabla} \cdot \mathbf{v} = 0$$

In order for the problem to be closed, the stress tensor must be related to velocity and pressure :

$$\boldsymbol{\sigma} = -p\mathbf{1} + \mathbf{s}$$

The deviatoric stress tensor is related to the velocity gradient through the dynamic viscosity μ as follows :

$$\mathbf{s} = 2\mu\dot{\boldsymbol{\epsilon}} \qquad \dot{\boldsymbol{\epsilon}} = \frac{1}{2}\left(\boldsymbol{\nabla}\mathbf{v} + (\boldsymbol{\nabla}\mathbf{v})^T\right)$$

Penalty formulation

Material is assumed to be weakly compressible :

$$p = -\lambda \, \boldsymbol{\nabla} \cdot \mathbf{v} = -\lambda(\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy})$$

λ is the penalty factor ($\sim$ bulk viscosity), with $\lambda >> \mu$.

Penalty formulation

Material is assumed to be weakly compressible :

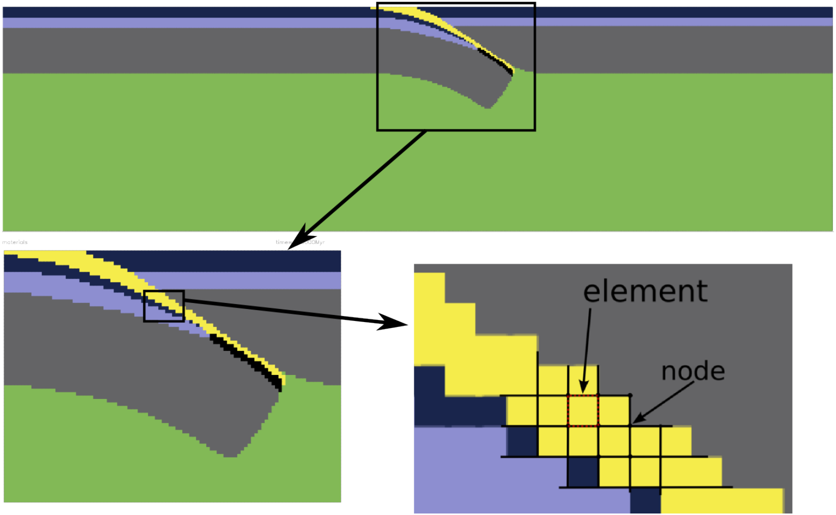
$$p = -\lambda \, \nabla \cdot \mathbf{v} = -\lambda(\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy})$$

λ is the penalty factor ($\sim$ bulk viscosity), with $\lambda >> \mu$.

$$\left.\begin{array}{c} \nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \\ \boldsymbol{\sigma} = -p\mathbf{1} + \mathbf{s} \\ \mathbf{s} = 2\mu\dot{\boldsymbol{\epsilon}} \\ \dot{\boldsymbol{\epsilon}} = \frac{1}{2}\left(\nabla v + (\nabla v)^T\right) \\ p = -\lambda \cdot \nabla \cdot \mathbf{v} \end{array}\right\} \Rightarrow \nabla \cdot (\mu \nabla \mathbf{v}) + \lambda \nabla (\nabla \cdot \mathbf{v}) + \mathbf{b} = 0$$

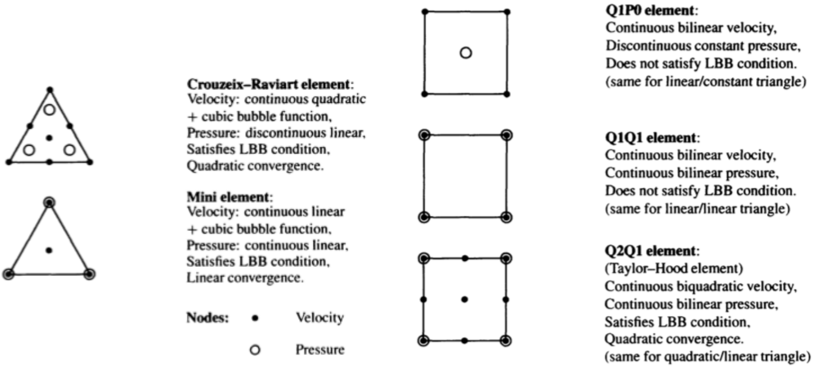
$\Rightarrow$ decoupling of $\mathbf{v}$ and p equations.

Concept



PDE's $\rightarrow$ discretisation $\rightarrow$ FEM formulation $\rightarrow$ $\mathbf{A} \cdot \mathbf{x} = \mathbf{B}$

Which element ?



(Donea & Huerta)

Not all elements are born equal.

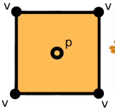
Which element ?

Crouzeix–Raviart element:
Velocity: continuous quadratic
+ cubic bubble function,
Pressure: discontinuous linear,
Satisfies LBB condition,
Quadratic convergence.

Mini element:
Velocity: continuous linear
+ cubic bubble function,
Pressure: continuous linear,
Satisfies LBB condition,
Linear convergence.

Nodes: • Velocity

○ Pressure



Q1P0 element

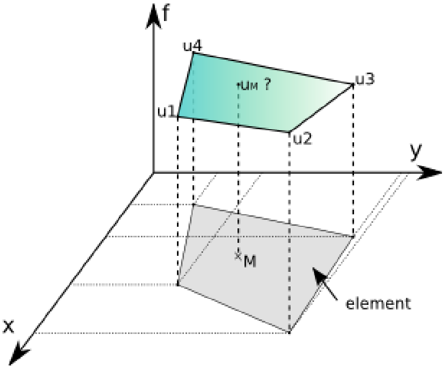
Continuous bilinear velocity
Discontinuous constant pressure

Q1Q1 element:
Continuous bilinear velocity,
Continuous bilinear pressure,
Does not satisfy LBB condition.
(same for linear/linear triangle)

Q2Q1 element:
(Taylor–Hood element)
Continuous biquadratic velocity,
Continuous bilinear pressure,
Satisfies LBB condition,
Quadratic convergence.
(same for quadratic/linear triangle)

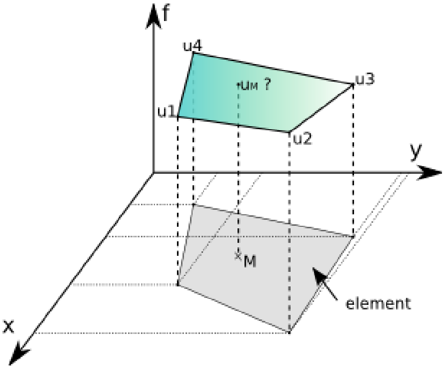
Not all elements are born equal.

Shape functions



$$u_M = f(u_1, u_2, u_3, u_4, x, y)$$

Shape functions



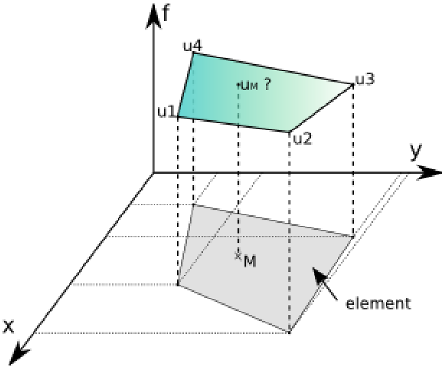
$u_M = f(u_1, u_2, u_3, u_4, x, y)$

For M(x,y) inside the element :

$$u(x,y) = \sum_{i=1}^{4} N_i(x,y)\ u_i$$

$$v(x,y) = \sum_{i=1}^{4} N_i(x,y)\ v_i$$

Shape functions



$$u_M = f(u_1, u_2, u_3, u_4, x, y)$$

For M(x,y) inside the element :

$$u(x,y) = \sum_{i=1}^{4} N_i(x,y)\ u_i$$

$$v(x,y) = \sum_{i=1}^{4} N_i(x,y)\ v_i$$

$$\dot{\epsilon}_{xx}(x,y) = \frac{\partial u}{\partial x} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial x}\ u_i$$

$$\dot{\epsilon}_{yy}(x,y) = \frac{\partial v}{\partial y} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial y}\ v_i$$

under the bonnet of the FEM engine (1)

The tensor $\boldsymbol{\sigma}$ is symmetric (*i.e.* $\sigma_{xy} = \sigma_{yx}$).
It can be cast in vector format :

$$\left(\begin{array}{c} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{array}\right) = \left(\begin{array}{c} -p \\ -p \\ 0 \end{array}\right) + 2\mu \left(\begin{array}{c} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{array}\right)$$

under the bonnet of the FEM engine (1)

The tensor $\boldsymbol{\sigma}$ is symmetric (*i.e.* $\sigma_{xy} = \sigma_{yx}$).
It can be cast in vector format :

$$
\begin{aligned}
\left(\begin{array}{c} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{array}\right) &= \left(\begin{array}{c} -p \\ -p \\ 0 \end{array}\right) + 2\mu \left(\begin{array}{c} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{array}\right) \\
&= \lambda \left(\begin{array}{c} \dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ 0 \end{array}\right) + 2\mu \left(\begin{array}{c} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{array}\right)
\end{aligned}
$$

under the bonnet of the FEM engine (1)

The tensor $\boldsymbol{\sigma}$ is symmetric (*i.e.* $\sigma_{xy} = \sigma_{yx}$).
It can be cast in vector format :

$$
\begin{aligned}
\left(\begin{array}{c} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{array}\right) &= \left(\begin{array}{c} -p \\ -p \\ 0 \end{array}\right) + 2\mu \left(\begin{array}{c} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{array}\right) \\
&= \lambda \left(\begin{array}{c} \dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ 0 \end{array}\right) + 2\mu \left(\begin{array}{c} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{array}\right) \\
&= \left[\lambda \underbrace{\left(\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right)}_{\mathbf{K}} + \mu \underbrace{\left(\begin{array}{ccc} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{array}\right)}_{\mathbf{C}} \right] \cdot \left(\begin{array}{c} \frac{\partial u}{\partial x} \\ \\ \frac{\partial v}{\partial y} \\ \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{array}\right)
\end{aligned}
$$

under the bonnet of the FEM engine (2)

Remember that

$$\frac{\partial u}{\partial x} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial x} \, u_i \qquad \frac{\partial v}{\partial y} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial y} \, v_i$$

$$\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial y} \, u_i + \sum_{i=1}^{4} \frac{\partial N_i}{\partial x} \, v_i$$

so that

$$\left(\begin{array}{c} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{array} \right) = \underbrace{\left(\begin{array}{cccccccc} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial x} \end{array} \right)}_{\mathbf{B}} \cdot \underbrace{\left(\begin{array}{c} u1 \\ v1 \\ u2 \\ v2 \\ u3 \\ v3 \\ u4 \\ v4 \end{array} \right)}_{\mathbf{V}}$$

under the bonnet of the FEM engine (3)

Finally,

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = (\lambda \mathbf{K} + \mu \mathbf{C}) \cdot \mathbf{B} \cdot \mathbf{V}$$

Weak form (1)

We start again from

$$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0}$$

For the N_i's 'regular enough', we can write :

$$\int_{\Omega_e} N_i \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} d\Omega + \int_{\Omega_e} N_i \mathbf{b} d\Omega = 0$$

We can integrate by parts and drop the surface term :

$$\int_{\Omega_e} \boldsymbol{\nabla} N_i \cdot \boldsymbol{\sigma} d\Omega = \int_{\Omega_e} N_i \mathbf{b} d\Omega$$

or,

$$\int_{\Omega_e} \begin{pmatrix} \frac{\partial N_i}{\partial x} & 0 & \frac{\partial N_i}{\partial y} \\ 0 & \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} d\Omega = \int_{\Omega_e} N_i \mathbf{b} d\Omega$$

Weak form (2)

Let $i = 1, 2, 3, 4$ and let us define

$$\mathbf{N}_b^T = (N_1 b_x, N_1 b_y, ... N_4 b_x, N_4 b_y)$$

then we can write

$$\int_{\Omega_e} \mathbf{B}^T \cdot [\lambda \mathbf{K} + \mu \mathbf{C}] \cdot \mathbf{B} \cdot \mathbf{V} d\Omega = \int_{\Omega_e} \mathbf{N}_b d\Omega$$

and finally :

$$\underbrace{\left(\int_{\Omega_e} \mathbf{B}^T \cdot [\lambda \mathbf{K} + \mu \mathbf{C}] \cdot \mathbf{B} d\Omega \right)}_{A_{el}(8 \times 8)} \cdot \underbrace{\mathbf{V}}_{(8x1)} = \underbrace{\int_{\Omega_e} \mathbf{N}_b d\Omega}_{B_{el}(8 \times 1)}$$

or,

$$\left[\underbrace{\left(\int_{\Omega_e} \lambda \mathbf{B}^T \cdot \mathbf{K} \cdot \mathbf{B} d\Omega \right)}_{A_{el}^{\lambda}(8 \times 8)} + \underbrace{\left(\int_{\Omega_e} \mu \mathbf{B}^T \cdot \mathbf{C} \cdot \mathbf{B} d\Omega \right)}_{A_{el}^{\mu}(8 \times 8)} \right] \cdot \underbrace{\mathbf{V}}_{(8x1)} = \underbrace{\int_{\Omega_e} \mathbf{N}_b d\Omega}_{B_{el}(8 \times 1)}$$

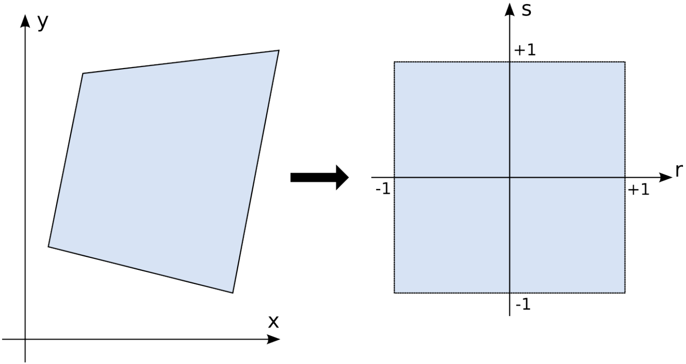
Numerical integration (1)

We need to compute quantities involving integrals over the element.
It is carried out in two steps :

1. change of variables
2. quadrature

Numerical integration (2)

To exploit the full flexibility of FEM, we employ isoparametric elements. Each element in physical space is mapped onto the reference element with fixed shape, size, and orientation.



Numerical integration (3)

Let f be a function. From the chain rule of calculus, we write

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r}$$

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s}$$

or in matrix form :

$$\begin{pmatrix} \frac{\partial f}{\partial r} \\ \\ \frac{\partial f}{\partial s} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{pmatrix}}_{\mathbf{J}} \cdot \begin{pmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

where $\mathbf{J}$ is called the Jacobian of the transformation

Numerical integration (4)

We have :

$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{pmatrix} = \mathbf{J}^{-1} \cdot \begin{pmatrix} \frac{\partial f}{\partial r} \\ \\ \frac{\partial f}{\partial s} \end{pmatrix}$$

One can prove that

$$\int \int_{\Omega_e} dxdy = \int_{-1}^{+1} \int_{-1}^{+1} |J| drds$$

so that for instance

$$\int \int_{\Omega_e} \frac{\partial f}{\partial x} dxdy = \int_{-1}^{+1} \int_{-1}^{+1} \left(\tilde{J}_{11} \frac{\partial f}{\partial r} + \tilde{J}_{12} \frac{\partial f}{\partial s} \right) |J| drds$$

where $\tilde{J}_{ij}$ are the components of $\mathbf{J}^{-1}$.

Numerical integration (5) - the Gauss quadrature

- a quadrature rule is an approximation of the definite integral of a function, usually stated as a weighted sum of function values at specified points within the domain of integration.
- an $n-$point Gaussian quadrature rule is constructed to yield an exact result for polynomials of degree 2n-1

$$\int_{-1}^{+1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

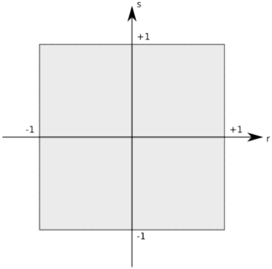
$$\int\int_{-1}^{+1} f(x,y)dxdy \approx \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(x_i, y_j)$$

$$\Rightarrow \quad \int_{\Omega_e} \lambda \mathbf{B}^T \cdot \mathbf{K} \cdot \mathbf{B} d\Omega = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j |J| \left(\lambda \mathbf{B}^T \cdot \mathbf{K} \cdot \mathbf{B}\right)_{(x_i, y_j)}$$

Numerical integration (6) - the Gauss quadrature

number of points	position r_i	weight w_i
1	0	2
2	$-1/\sqrt{3}$ $+1/\sqrt{3}$	1 1
3	$-\sqrt{15}/5$ 0 $+\sqrt{15}/5$	$5/9$ $8/9$ $5/9$

Shape functions (2)



$$
\begin{aligned}
N_1(r,s) &= 0.25(1-r)(1-s) \\
N_2(r,s) &= 0.25(1+r)(1-s) \\
N_3(r,s) &= 0.25(1+r)(1+s) \\
N_4(r,s) &= 0.25(1-r)(1+s)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial N_1}{\partial r}(r,s) &= -0.25(1-s) \\
\frac{\partial N_2}{\partial r}(r,s) &= +0.25(1-s) \\
\frac{\partial N_3}{\partial r}(r,s) &= +0.25(1+s) \\
\frac{\partial N_4}{\partial r}(r,s) &= -0.25(1+s)
\end{aligned}
$$

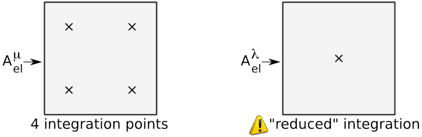
$$
\begin{aligned}
\frac{\partial N_1}{\partial s}(r,s) &= -0.25(1-r) \\
\frac{\partial N_2}{\partial s}(r,s) &= -0.25(1+r) \\
\frac{\partial N_3}{\partial s}(r,s) &= +0.25(1+r) \\
\frac{\partial N_4}{\partial s}(r,s) &= +0.25(1-r)
\end{aligned}
$$

FEM algorithm

1. partition domain Ω into elements Ω_e, $e = 1, \ldots n_{el}$.

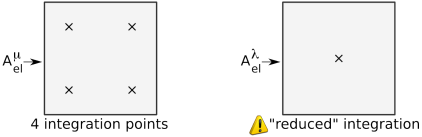
FEM algorithm

1. partition domain Ω into elements Ω_e, $e = 1, \ldots n_{el}$.
2. loop over elements and for each element compute $\mathbf{A}_{el}$, $\mathbf{B}_{el}$



FEM algorithm

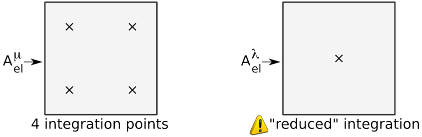
1. partition domain Ω into elements Ω_e, $e = 1, \ldots n_{el}$.
2. loop over elements and for each element compute $\mathbf{A}_{el}$, $\mathbf{B}_{el}$



3. a node belongs to several elements
   $\rightarrow$ need to assemble $\mathbf{A}_{el}$ and $\mathbf{B}_{el}$ in $\mathbf{A}$, $\mathbf{B}$

FEM algorithm

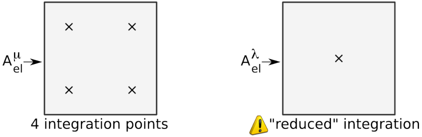
1. partition domain Ω into elements Ω_e, $e = 1, \ldots n_{el}$.
2. loop over elements and for each element compute $\mathbf{A}_{el}$, $\mathbf{B}_{el}$



3. a node belongs to several elements
   $\rightarrow$ need to assemble $\mathbf{A}_{el}$ and $\mathbf{B}_{el}$ in $\mathbf{A}$, $\mathbf{B}$
4. apply boundary conditions

FEM algorithm

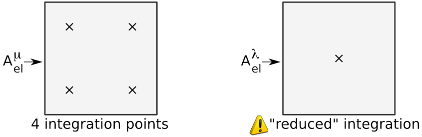
1. partition domain Ω into elements Ω_e, $e = 1, \ldots n_{el}$.
2. loop over elements and for each element compute $\mathbf{A}_{el}$, $\mathbf{B}_{el}$



3. a node belongs to several elements
   $\rightarrow$ need to assemble $\mathbf{A}_{el}$ and $\mathbf{B}_{el}$ in $\mathbf{A}$, $\mathbf{B}$
4. apply boundary conditions
5. solve system : $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{B}$

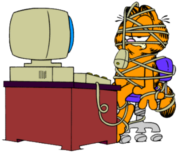
FEM algorithm

1. partition domain Ω into elements Ω_e, $e = 1, \ldots n_{el}$.
2. loop over elements and for each element compute $\mathbf{A}_{el}$, $\mathbf{B}_{el}$



3. a node belongs to several elements
   $\rightarrow$ need to assemble $\mathbf{A}_{el}$ and $\mathbf{B}_{el}$ in $\mathbf{A}$, $\mathbf{B}$
4. apply boundary conditions
5. solve system : $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{B}$
6. visualise/analyse $\mathbf{x}$

SimpleFEM



- 2D
- fortran90
- simple (naive ?) approach
  - no attention to performance
  - no attention to memory
  - no subroutines, modules, ...
- available at : `http://cedricthieulot.net`

Files

- `blas_routines.f90`
  contains a few subroutines from the BLAS (Basic Linear Algebra Subprograms) library
- `linpack_d.f90`
  software library for performing numerical linear algebra. makes use of the BLAS libraries for performing basic vector and matrix operations.
- `simplefem.f90`
  the FEM code

Declarations (1)

terminology : "dof" = degree of freedom = unknown

```
integer,parameter :: m=4                          ! number of nodes making an element
integer,parameter :: ndof=2                       ! number of dofs per node

integer nnx                                       ! number of grid points in the x direction
integer nny                                       ! number of grid points in the y direction
integer np                                        ! number of grid points

integer nelx                                      ! number of elements in the x direction
integer nely                                      ! number of elements in the y direction
integer nel                                       ! number of elements

integer Nfem                                      ! size of the FEM matrix

integer,dimension(:,:),allocatable :: icon        ! connectivity array

integer,dimension(:),allocatable :: ipvt          ! work array needed by the solver
```

Declarations (2)

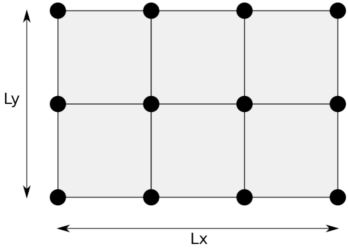
```fortran
real(8) Lx,Ly                                        ! size of the numerical domain
real(8) viscosity                                    ! dynamic viscosity of the material
real(8) density                                      ! mass density of the material
real(8) gx,gy                                        ! gravity acceleration
real(8) penalty                                      ! penalty parameter lambda

real(8), dimension(:),   allocatable :: x,y         ! node coordinates arrays
real(8), dimension(:),   allocatable :: u,v         ! node velocity arrays
real(8), dimension(:),   allocatable :: press       ! pressure

real(8), dimension(:),   allocatable :: B           ! right hand side
real(8), dimension(:,:), allocatable :: A           ! FEM matrix

real(8), dimension(:),   allocatable :: work        ! work array needed by the solver
real(8), dimension(:),   allocatable :: bc_val      ! array containing bc values
```

Declarations (3)

```
real(8), external :: b1,b2,uth,vth,pth              ! body force and analytical solution
real(8) rq,sq,wq                                    ! local coordinate and weight of qpoint
real(8) xq,yq                                       ! global coordinate of qpoint
real(8) uq,vq                                       ! velocity at qpoint
real(8) exxq,eyyq,exyq                              ! strain-rate components at qpoint
real(8) Ael(m*ndof,m*ndof)                          ! elemental FEM matrix
real(8) Bel(m*ndof)                                 ! elemental right hand side
real(8) N(m),dNdx(m),dNdy(m),dNdr(m),dNds(m)        ! shape fcts and derivatives
real(8) jcob                                        ! determinant of jacobian matrix
real(8) jcb(2,2)                                    ! jacobian matrix
real(8) jcbi(2,2)                                   ! inverse of jacobian matrix
real(8) Bmat(3,ndof*m)                              ! B matrix
real(8), dimension(3,3) :: Kmat                     ! K matrix
real(8), dimension(3,3) :: Cmat                     ! C matrix
real(8) Aref                                        !
real(8) eps                                         !
real(8) rcond                                       !
                                                    !
logical, dimension(:), allocatable :: bc_fix        ! prescribed b.c. array
logical, dimension(:,:), allocatable :: C           ! non-zero terms in FEM matrix
```

Declarations

```
Lx=3.d0
Ly=2.d0

nnx=4
nny=3


np=nnx*nny

nelx=nnx-1
nely=nny-1

nel=nelx*nely

viscosity=1.d0
density=1.d0

penalty=1.d7

Nfem=np*ndof

eps=1.d-10

Kmat(1,1)=1.d0;Kmat(1,2)=1.d0;Kmat(1,3)=0.d0
Kmat(2,1)=1.d0;Kmat(2,2)=1.d0;Kmat(2,3)=0.d0
Kmat(3,1)=0.d0;Kmat(3,2)=0.d0;Kmat(3,3)=0.d0

Cmat(1,1)=2.d0;Cmat(1,2)=0.d0;Cmat(1,3)=0.d0
Cmat(2,1)=0.d0;Cmat(2,2)=2.d0;Cmat(2,3)=0.d0
Cmat(3,1)=0.d0;Cmat(3,2)=0.d0;Cmat(3,3)=1.d0
```



$$\mathbf{K} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{C} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Allocating memory

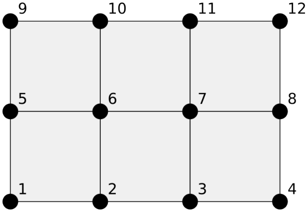
```
allocate(x(np))
allocate(y(np))
allocate(u(np))
allocate(v(np))

allocate(press(nel))

allocate(icon(m,nel))

allocate(A(Nfem,Nfem))
allocate(B(Nfem))
allocate(C(Nfem,Nfem))

allocate(bc_fix(Nfem))
allocate(bc_val(Nfem))
```

Allocating memory

```
allocate(x(np))
allocate(y(np))
allocate(u(np))
allocate(v(np))

allocate(press(nel))

allocate(icon(m,nel))

allocate(A(Nfem,Nfem))
allocate(B(Nfem))
allocate(C(Nfem,Nfem))

allocate(bc_fix(Nfem))
allocate(bc_val(Nfem))
```

- size(A)=$\text{Nfem}^2 = \text{ndof}^2 \times \text{np}^2$
- with ndof=2, in double precision :
  $\text{np}=10^2$ size(A) = 320kb
  $\text{np}=50^2$ size(A) = 200Mb
  $\text{np}=100^2$ size(A) = 3.2Gb
  $\Rightarrow$ not a viable option in the end

grid points setup



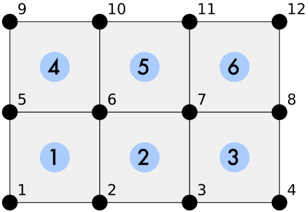

```
counter=0
do j=0,nely
   do i=0,nelx
      counter=counter+1
      x(counter)=dble(i)*Lx/dble(nelx)
      y(counter)=dble(j)*Ly/dble(nely)
   end do
end do
```

OUT/gridnodes.dat

xpos	ypos	node
0.00000	0.00000	1
1.00000	0.00000	2
2.00000	0.00000	3
3.00000	0.00000	4
0.00000	1.00000	5
1.00000	1.00000	6
2.00000	1.00000	7
3.00000	1.00000	8
0.00000	2.00000	9
1.00000	2.00000	10
2.00000	2.00000	11
3.00000	2.00000	12

connectivity setup

```
counter=0
do j=1,nely
   do i=1,nelx
      counter=counter+1
      icon(1,counter)=i+(j-1)*(nelx+1)
      icon(2,counter)=i+1+(j-1)*(nelx+1)
      icon(3,counter)=i+1+j*(nelx+1)
      icon(4,counter)=i+j*(nelx+1)
   end do
end do
```



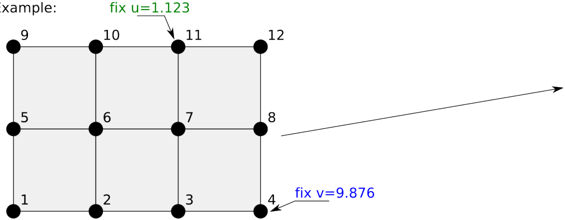
OUT/icon.dat

```
----------------------------------
---element #  1 -------------------
----------------------------------
node 1:  1 at pos.  0.000  0.000
node 2:  2 at pos.  1.000  0.000
node 3:  6 at pos.  1.000  1.000
node 4:  5 at pos.  0.000  1.000
----------------------------------
---element #  2 -------------------
----------------------------------
node 1:  2 at pos.  1.000  0.000
node 2:  3 at pos.  2.000  0.000
node 3:  7 at pos.  2.000  1.000
node 4:  6 at pos.  1.000  1.000
----------------------------------
---element #  3 -------------------
----------------------------------
node 1:  3 at pos.  2.000  0.000
node 2:  4 at pos.  3.000  0.000
node 3:  8 at pos.  3.000  1.000
node 4:  7 at pos.  2.000  1.000
----------------------------------
---element #  4 -------------------
----------------------------------
node 1:  5 at pos.  0.000  1.000
node 2:  6 at pos.  1.000  1.000
node 3: 10 at pos.  1.000  2.000
node 4:  9 at pos.  0.000  2.000

etc ...
```

boundary conditions setup

```
bc_fix=.false.

do i=1,np
   if (x(i).lt.eps) then
      bc_fix((i-1)*ndof+1)=.true. ; bc_val((i-1)*ndof+1)=0.d0
      bc_fix((i-1)*ndof+2)=.true. ; bc_val((i-1)*ndof+2)=0.d0
   endif
   if (x(i).gt.(Lx-eps)) then
      bc_fix((i-1)*ndof+1)=.true. ; bc_val((i-1)*ndof+1)=0.d0
      bc_fix((i-1)*ndof+2)=.true. ; bc_val((i-1)*ndof+2)=0.d0
   endif
   if (y(i).lt.eps) then
      bc_fix((i-1)*ndof+1)=.true. ; bc_val((i-1)*ndof+1)=0.d0
      bc_fix((i-1)*ndof+2)=.true. ; bc_val((i-1)*ndof+2)=0.d0
   endif
   if (y(i).gt.(Ly-eps) ) then
      bc_fix((i-1)*ndof+1)=.true. ; bc_val((i-1)*ndof+1)=0.d0
      bc_fix((i-1)*ndof+2)=.true. ; bc_val((i-1)*ndof+2)=0.d0
   endif
end do
```

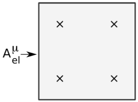
Example:



	bc_fix	bc_val	
u1	F		1
v1	F		2
u2	F		3
v2	F		4
u3	F		5
v3	F		6
u4	F		7
v4	T	9.876	8
u5	F		9
v5	F		10
u6	F		11
v6	F		12
u7	F		13
v7	F		14
u8	F		15
v8	F		16
u9	F		17
v9	F		18
u10	F		19
v10	F		20
u11	T	1.123	21
v11	F		22
u12	F		23
v12	F		24

Building the FEM matrix and rhs (1)

```
do iel=1,nel

 Ael=0.d0
 Bel=0.d0

 !=== integrate viscous term at 4 qpoints ===

 do iq=-1,1,2
 do jq=-1,1,2

    rq=iq/sqrt(3.d0) ; sq=jq/sqrt(3.d0) ; wq=1.d0

    ! compute N(1),N(2),N(3),N(4)

    ! compute dNdr(1),dNdr(2),dNdr(3),dNdr(4)

    ! compute jcb(2x2) = jacobian J

    ! compute jcob = |J|

    ! compute jcbi(2x2) = inverse of jacobian

    ! compute dNdx(1),dNdx(2),dNdx(3),dNdx(4)

    ! build B(3x8) matrix

    Ael=Ael+matmul(transpose(Bmat),matmul(viscosity*Cmat,Bmat))*wq*jcob

    do i=1,m
    i1=2*i-1
    i2=2*i
    Bel(i1)=Bel(i1)+N(i)*jcob*wq*density*gx
    Bel(i2)=Bel(i2)+N(i)*jcob*wq*density*gy
    end do

 end do
 end do
```



4 integration points

$$A_{el} = \int_{\Omega_e} \mu \mathbf{B}^T \cdot \mathbf{C} \cdot \mathbf{B} d\Omega$$

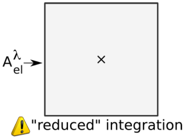
$$= \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j |J| \left(\mu \mathbf{B}^T \cdot \mathbf{C} \cdot \mathbf{B} \right)$$

$$B_{el} = \int_{\Omega_e} \mathbf{N}_b d\Omega$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j |J| (\mathbf{N}_b)$$

Building the FEM matrix and rhs (2)

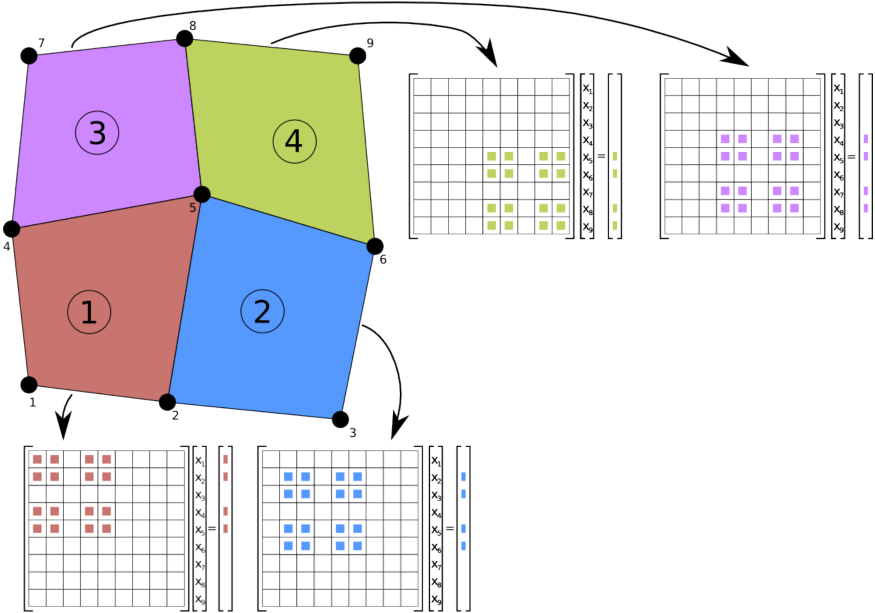
```
!=== integrate penalty term at one qpoint ===

rq=0.d0 ; sq=0.d0 ; wq=4.d0

! compute N(1),N(2),N(3),N(4)

! compute dNdr(1),dNdr(2),dNdr(3),dNdr(4)

! compute jcb(2x2) = jacobian J

! compute jcob = |J|

! compute jcbi(2x2) = inverse of jacobian

! compute dNdx(1),dNdx(2),dNdx(3),dNdx(4)

! build B(3x8) matrix

Ael=Ael+matmul(transpose(Bmat),matmul(penalty*Kmat,Bmat))*wq*jcob

!=== assemble ===

[...]

end do
```



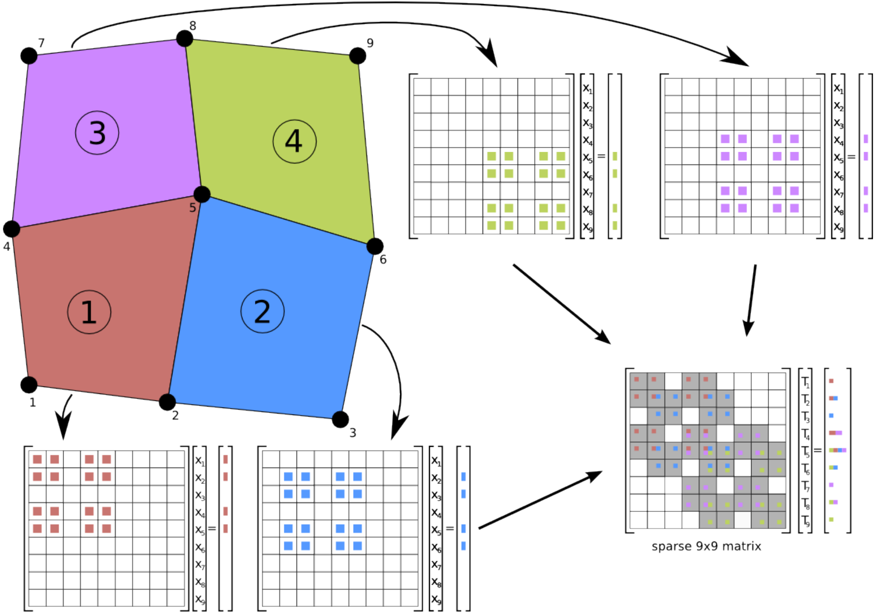
$$A_{el} = \int_{\Omega_e} \lambda \mathbf{B}^T \cdot \mathbf{K} \cdot \mathbf{B} d\Omega$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j |J| \left(\lambda \mathbf{B}^T \cdot \mathbf{K} \cdot \mathbf{B} \right)$$

assembly (1)



assembly (2)



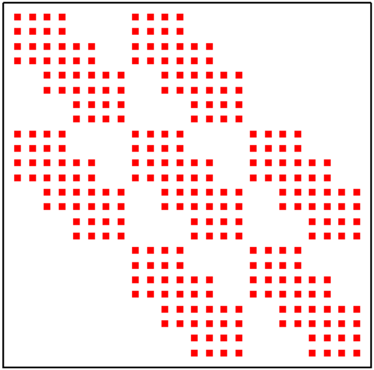
sparse 9x9 matrix

assembly (3)

```
do k1=1,m
   ik=icon(k1,iel)
   do i1=1,ndof
      ikk=ndof*(k1-1)+i1
      m1=ndof*(ik-1)+i1
      do k2=1,m
         jk=icon(k2,iel)
         do i2=1,ndof
            jkk=ndof*(k2-1)+i2
            m2=ndof*(jk-1)+i2
            A(m1,m2)=A(m1,m2)+Ael(ikk,jkk)
            C(m1,m2)=.true.
         end do
      end do
      B(m1)=B(m1)+Bel(ikk)
   end do
end do
```

(ikk,jkk) : coordinates in local matrix
(m1,m2) : coordinates in global matrix

FEM matrix sparsity pattern :



Nfem= ndof $\times$np $= 2\times4\times3 = 24$

imposing boundary conditions (1)

Let us consider an algebraic system of the form $\mathbf{A}.\mathbf{x} = \mathbf{B}$:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{44} \\ A_{31} & A_{32} & A_{33} & A_{44} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{pmatrix}$$

Let us assume that we want to impose $x_3 = x^{bc}$ on the third node. Then system writes

imposing boundary conditions (1)

Let us consider an algebraic system of the form $\mathbf{A}.\mathbf{x} = \mathbf{B}$:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{44} \\ A_{31} & A_{32} & A_{33} & A_{44} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{pmatrix}$$

Let us assume that we want to impose $x_3 = x^{bc}$ on the third node. Then system writes

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{44} \\ 0 & 0 & 1 & 0 \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ x^{bc} \\ B_4 \end{pmatrix}$$

imposing boundary conditions (1)

Let us consider an algebraic system of the form $\mathbf{A}.\mathbf{x} = \mathbf{B}$:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{44} \\ A_{31} & A_{32} & A_{33} & A_{44} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{pmatrix}$$

Let us assume that we want to impose $x_3 = x^{bc}$ on the third node. Then system writes

$$\begin{pmatrix} A_{11} & A_{12} & 0 & A_{13} \\ A_{21} & A_{22} & 0 & A_{23} \\ 0 & 0 & 1 & 0 \\ A_{41} & A_{42} & 0 & A_{43} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 - A_{13}x^{bc} \\ B_2 - A_{23}x^{bc} \\ x^{bc} \\ B_4 - A_{43}x^{bc} \end{pmatrix}$$

imposing boundary conditions (1)

Let us consider an algebraic system of the form $\mathbf{A}.\mathbf{x} = \mathbf{B}$:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{44} \\ A_{31} & A_{32} & A_{33} & A_{44} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{pmatrix}$$

Let us assume that we want to impose $x_3 = x^{bc}$ on the third node. Then system writes

$$\begin{pmatrix} A_{11} & A_{12} & 0 & A_{13} \\ A_{21} & A_{22} & 0 & A_{23} \\ 0 & 0 & A_{33} & 0 \\ A_{41} & A_{42} & 0 & A_{43} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} B_1 - A_{13} x^{bc} \\ B_2 - A_{23} x^{bc} \\ A_{33} * x^{bc} \\ B_4 - A_{43} x^{bc} \end{pmatrix}$$

imposing boundary conditions (2)

```
do i=1,Nfe}m
   if (bc_fix(i)) then
      Aref=A(i,i)
      do j=1,Nfem
         B(j)=B(j)-A(i,j)*bc_val(i)
         A(i,j)=0.d0
         A(j,i)=0.d0
      end do
      A(i,i)=Aref
      B(i)=Aref*bc_val(i)
   end if
end do
```

Solving the system

⇒ `www.netlib.org`

- dgeco factors a double precision matrix by gaussian elimination and estimates the condition of the matrix.
- dgesl solves the double precision system using the factors computed by dgeco or dgefa.

```
job=0
allocate(work(Nfem))
allocate(ipvt(Nfem))
call DGECO (A, Nfem, Nfem, ipvt, rcond, work)
call DGESL (A, Nfem, Nfem, ipvt, B, job)
deallocate(ipvt)
deallocate(work)
```

The solution is stored in B, we must now split it into `u` and `v` arrays :

```
do i=1,np
   u(i)=B((i-1)*ndof+1)
   v(i)=B((i-1)*ndof+2)
end do
```

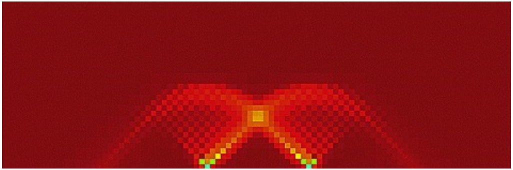
Compute pressure (1)

```
do iel=1,nel

   [ compute N(1:4),dNdx(1:4),dNdy(1:4)
     at center of element]

   xq=0.d0
   yq=0.d0
   exxq=0.d0
   eyyq=0.d0
   do k=1,m
      xq=xq+N(k)*x(icon(k,iel))
      yq=yq+N(k)*y(icon(k,iel))
      exxq=exxq+ dNdx(k)*u(icon(k,iel))
      eyyq=eyyq+ dNdy(k)*v(icon(k,iel))
   end do

   press(iel)=-penalty*(exxq+eyyq)

end do
```

$$
\begin{aligned}
p &= -\lambda \boldsymbol{\nabla} \cdot \mathbf{v} \\
  &= -\lambda \left(\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy}\right)
\end{aligned}
$$

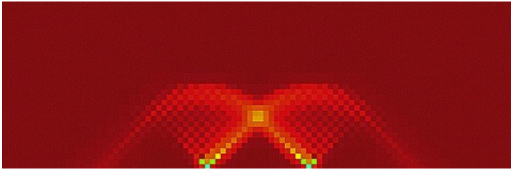
Compute pressure (2)

⚠ pressure field likely to display checkerboard pattern :

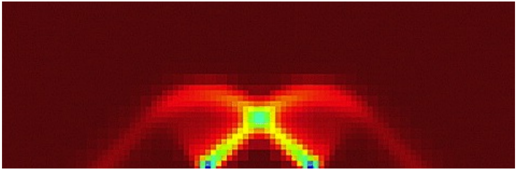


Compute pressure (2)

⚠ pressure field likely to display checkerboard pattern :



$\downarrow$ Smoothing/Filtering



Thieulot et al, JGR, 2008

Compiling and running the code

Compilation

```
> gfortran blas_routines.f90 linpack_d.f90 simplefem.f90 -o simplefem
```

Run

```
> ./simplefem
```

Produce figures :

```
> cd OUT
> gnuplot script
```

Analytical stokes benchmark

- square domain $\Omega = [0,1] \times [0,1]$,
- boundary conditions $\mathbf{v} = \mathbf{0}$ on Γ
- viscosity μ=1
- body force

$$
\begin{aligned}
b_x &= (12-24y)x^4 + (-24+48y)x^3 + (-48y+72y^2-48y^3+12)x^2 \\
&\quad +(-2+24y-72y^2+48y^3)x + 1 - 4y + 12y^2 - 8y^3 \\
b_y &= (8-48y+48y^2)x^3 + (-12+72y-72y^2)x^2 \\
&\quad +(4-24y+48y^2-48y^3+24y^4)x - 12y^2 + 24y^3 - 12y^4
\end{aligned}
$$

Exact solution is

$$
\begin{aligned}
u(x,y) &= x^2(1-x)^2(2y-6y^2+4y^3) \\
v(x,y) &= -y^2(1-y)^2(2x-6x^2+4x^3) \\
p(x.y) &= x(1-x)
\end{aligned}
$$

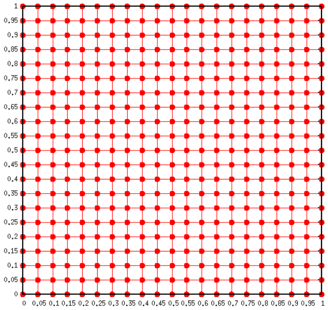
Benchmark setup

Lx=1
Ly=1

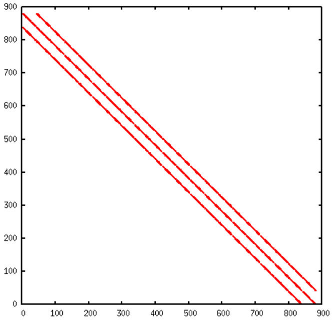
nnx=21
nny=21
np=nnx∗nny =441

nelx=nnx−1=20
nely=nny−1=20
nel=nelx∗nely=400

Nfem=ndof∗np=882

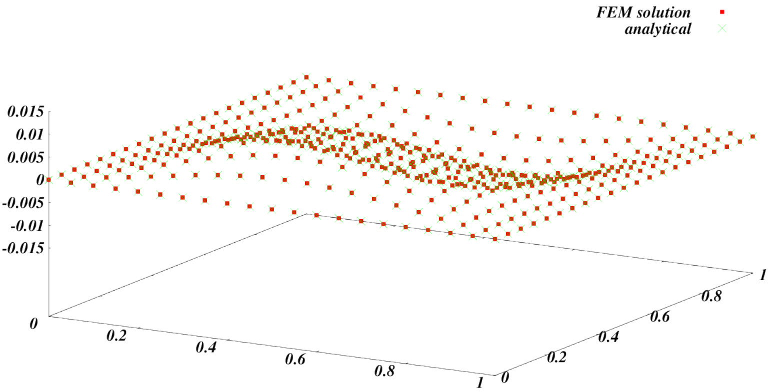


grid

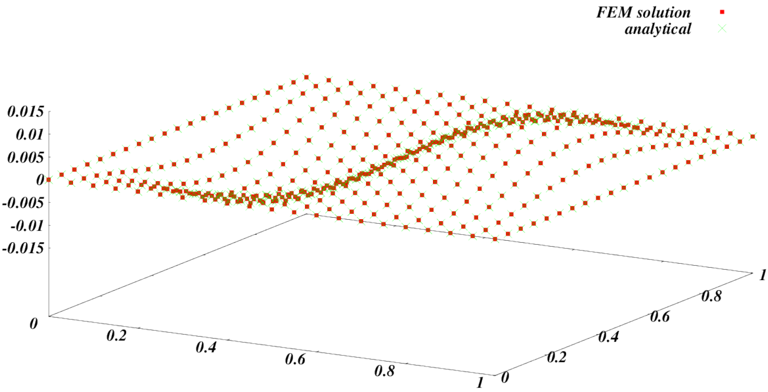


matrix

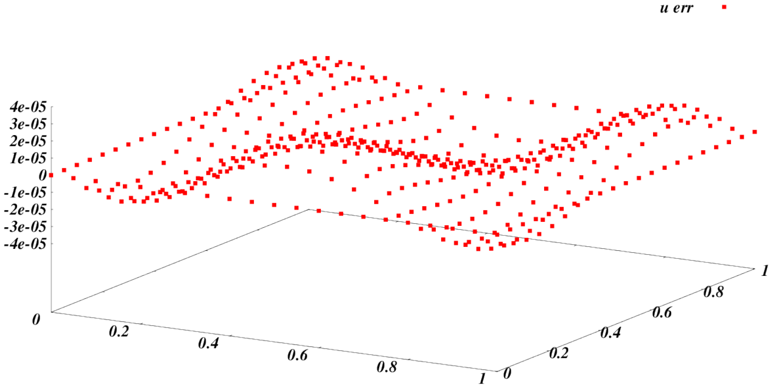
Results (1) : u field



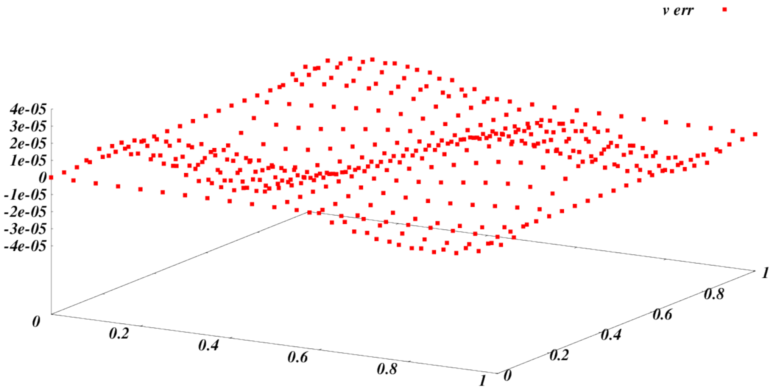
Results (2) : v field



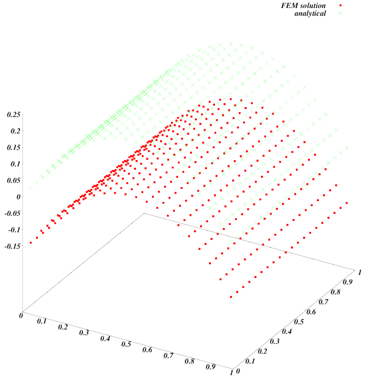
Results (3) : Velocity field error ($u - u_{th}$)

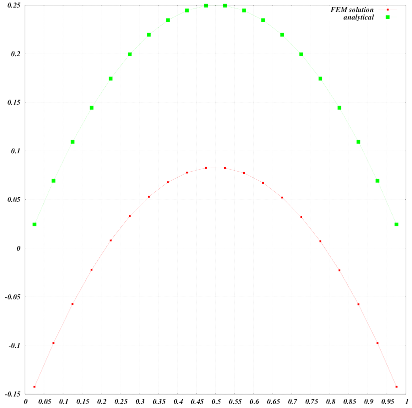


Results (4) : Velocity field error ($v - v_{th}$)



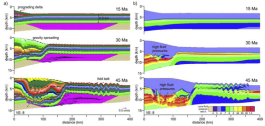
Results (5) : p field





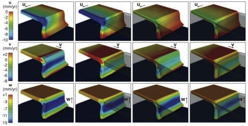
Applications to geodynamics

- **SOPALE** (2D)



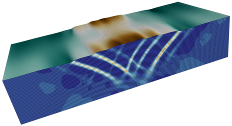
http://geodynamics.oceanography.dal.ca/sopaledoc.html

- **DOUAR** (3D)



http://www.cedricthieulot.net/douar.html

- **FANTOM** (2D,3D)



http://www.cedricthieulot.net/fantom.html

homework

- optimise
- try other benchmark
- extend to 3D
- implement non-penalised (i.e. mixed) formulations
- implement other element
- compile with optimised BLAS library of your computer OS
- implement clever storage of matrix terms (CSR/CSC)
  $\rightarrow$change solver to Pardiso or MUMPS
- implement particle-in-cell technique