The Finite Element Method in Geodynamics

C. Thieulot

January 29, 2019

Contents

WARNING: this is work in progress

1 Introduction

1.1 Philosophy

This document was writing with my students in mind, i.e. 3rd and 4th year Geology/Geophysics students at Utrecht University. I have chosen to use jargon as little as possible unless it is a term that is commonly found in the geodynamics literature (methods paper as well as application papers). There is no mathematical proof of any theorem or statement I make. These are to be found in generic Numerical Analysic, Finite Element and Linear Algebra books.

The codes I provide here are by no means optimised as I value code readability over code efficiency. I have also chosen to avoid resorting to multiple code files or even functions to favour a sequential reading of the codes. These codes are not designed to form the basis of a real life application: Existing open source highly optimised codes shoud be preferred, such as ASPECT, CITCOM, LAMEM, PTATIN, PYLITH, ...

All kinds of feedback is welcome on the text (grammar, typos, ...) or on the code(s). You will have my eternal gratitude if you wish to contribute an example, a benchmark, a cookbook.

All the python scripts and this document are freely available at

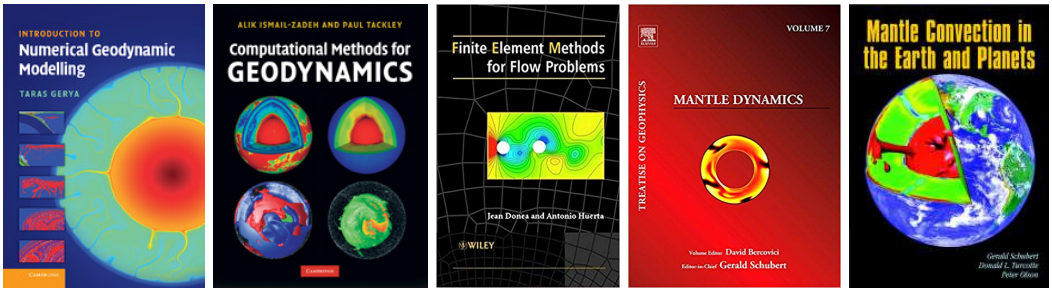
`https://github.com/cedrict/fieldstone`

1.2 Acknowledgments

I have benefitted from many discussions, lectures, tutorials, coffee machine discussions, debugging sessions, conference poster sessions, etc ... over the years. I wish to name these instrumental people in particular and in alphabetic order: Wolfgang Bangerth, Jean Braun, Philippe Fullsack, Menno Fraters, Anne Glerum, Timo Heister, Robert Myhill, John Naliboff, Lukas van de Wiel, Arie van den Berg, and the whole ASPECT family/team.

I wish to acknowledge many BSc and MSc students for their questions and feedback. and wish to mention Job Mos in particular who wrote the very first version of fieldstone as part of his MSc thesis. and Tom Weir for his contributions to the compressible formulations.

1.3 Essential literature



1.4 Installation

```
python3.6 -m pip install --user numpy scipy matplotlib
```

2 The physical equations of Fluid Dynamics

Symbol	meaning	unit
t	Time	s
x, y, z	Cartesian coordinates	m
$\boldsymbol{v}$	velocity vector	m$\cdot$ s^{-1}
ρ	mass density	kg/m^3
η	dynamic viscosity	Pa$\cdot$ s
λ	penalty parameter	Pa$\cdot$ s
T	temperature	K
$\boldsymbol{\nabla}$	gradient operator	m^{-1}
$\boldsymbol{\nabla}\cdot$	divergence operator	m^{-1}
p	pressure	Pa
$\dot{\boldsymbol{\varepsilon}}(\boldsymbol{v})$	strain rate tensor	s^{-1}
α	thermal expansion coefficient	K^{-1}
k	thermal conductivity	W/(m $\cdot$ K)
C_p	Heat capacity	J/K
H	intrinsic specific heat production	W/kg
β_T	isothermal compressibility	Pa^{-1}
$\boldsymbol{\tau}$	deviatoric stress tensor	Pa
$\boldsymbol{\sigma}$	full stress tensor	Pa

2.1 The heat transport equation - energy conservation equation

Let us start from the heat transport equation as shown in Schubert, Turcotte and Olson [**?**]:

$$\rho C_p \frac{DT}{Dt} - \alpha T \frac{Dp}{Dt} = \boldsymbol{\nabla} \cdot k \boldsymbol{\nabla} T + \Phi + \rho H$$

with D/Dt being the total derivatives so that

$$\frac{DT}{Dt} = \frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} T \qquad \frac{Dp}{Dt} = \frac{\partial p}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} p$$

Solving for temperature, this equation is often rewritten as follows:

$$\rho C_p \frac{DT}{Dt} - \boldsymbol{\nabla} \cdot k \boldsymbol{\nabla} T = \alpha T \frac{Dp}{Dt} + \Phi + \rho H$$

A note on the shear heating term Φ: In many publications, Φ is given by $\Phi = \tau_{ij} \partial_j u_i = \boldsymbol{\tau} : \boldsymbol{\nabla} \boldsymbol{v}$.

$$\begin{aligned}
\Phi &= \tau_{ij} \partial_j u_i \\
&= 2\eta \dot{\varepsilon}^d_{ij} \partial_j u_i \\
&= 2\eta \frac{1}{2} \left(\dot{\varepsilon}^d_{ij} \partial_j u_i + \dot{\varepsilon}^d_{ji} \partial_i u_j \right) \\
&= 2\eta \frac{1}{2} \left(\dot{\varepsilon}^d_{ij} \partial_j u_i + \dot{\varepsilon}^d_{ij} \partial_i u_j \right) \\
&= 2\eta \dot{\varepsilon}^d_{ij} \frac{1}{2} \left(\partial_j u_i + \partial_i u_j \right) \\
&= 2\eta \dot{\varepsilon}^d_{ij} \dot{\varepsilon}_{ij} \\
&= 2\eta \dot{\boldsymbol{\varepsilon}}^d : \dot{\boldsymbol{\varepsilon}} \\
&= 2\eta \dot{\boldsymbol{\varepsilon}}^d : \left(\dot{\boldsymbol{\varepsilon}}^d + \frac{1}{3} (\boldsymbol{\nabla} \cdot \boldsymbol{v}) \mathbf{1} \right) \\
&= 2\eta \dot{\boldsymbol{\varepsilon}}^d : \dot{\boldsymbol{\varepsilon}}^d + 2\eta \dot{\boldsymbol{\varepsilon}}^d : \mathbf{1} (\boldsymbol{\nabla} \cdot \boldsymbol{v}) \\
&= 2\eta \dot{\boldsymbol{\varepsilon}}^d : \dot{\boldsymbol{\varepsilon}}^d
\end{aligned} \tag{1}$$

Finally

$$\Phi = \boldsymbol{\tau} : \boldsymbol{\nabla} \boldsymbol{v} = 2\eta \dot{\boldsymbol{\varepsilon}}^d : \dot{\boldsymbol{\varepsilon}}^d = 2\eta \left((\dot{\varepsilon}^d_{xx})^2 + (\dot{\varepsilon}^d_{yy})^2 + 2(\dot{\varepsilon}^d_{xy})^2 \right)$$

2.2 The momentum conservation equations

Because the Prandlt number is virtually zero in Earth science applications the Navier Stokes equations reduce to the Stokes equation:

$$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \rho \boldsymbol{g} = 0$$

Since

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

it also writes

$$-\boldsymbol{\nabla} p + \boldsymbol{\nabla} \cdot \boldsymbol{\tau} + \rho \boldsymbol{g} = 0$$

Using the relationship $\boldsymbol{\tau} = 2\eta \dot{\boldsymbol{\varepsilon}}^d$ we arrive at

$$\boxed{-\boldsymbol{\nabla} p + \boldsymbol{\nabla} \cdot (2\eta \dot{\boldsymbol{\varepsilon}}^d) + \rho \boldsymbol{g} = 0}$$

2.3 The mass conservation equations

The mass conservation equation is given by

$$\frac{D\rho}{Dt} + \rho \boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$$

or,

$$\boxed{\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \boldsymbol{v}) = 0}$$

In the case of an incompressible flow, then $\partial\rho/\partial t = 0$ and $\boldsymbol{\nabla}\rho = 0$, i.e. $D\rho/Dt = 0$ and the remaining equation is simply:

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

2.4 The equations in ASPECT manual

The following is lifted off the ASPECT manual. We focus on the system of equations in a $d = 2$- or $d = 3$-dimensional domain Ω that describes the motion of a highly viscous fluid driven by differences in the gravitational force due to a density that depends on the temperature. In the following, we largely follow the exposition of this material in Schubert, Turcotte and Olson [**?**].

Specifically, we consider the following set of equations for velocity $\mathbf{u}$, pressure p and temperature T:

$$-\nabla \cdot \left[2\eta \left(\dot{\varepsilon}(\boldsymbol{v}) - \frac{1}{3}(\nabla \cdot \boldsymbol{v})\mathbf{1} \right) \right] + \nabla p = \rho \boldsymbol{g} \qquad \text{in } \Omega, \quad (2)$$

$$\nabla \cdot (\rho \boldsymbol{v}) = 0 \qquad \text{in } \Omega, \quad (3)$$

$$\begin{aligned} \rho C_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \nabla T \right) - \nabla \cdot k \nabla T &= \rho H \\ &+ 2\eta \left(\dot{\varepsilon}(\boldsymbol{v}) - \frac{1}{3}(\nabla \cdot \boldsymbol{v})\mathbf{1} \right) : \left(\dot{\varepsilon}(\boldsymbol{v}) - \frac{1}{3}(\nabla \cdot \boldsymbol{v})\mathbf{1} \right) \\ &+ \alpha T \left(\boldsymbol{v} \cdot \nabla p \right) \end{aligned} \qquad (4)$$

$$\text{in } \Omega,$$

where $\dot{\boldsymbol{\varepsilon}}(\mathbf{u}) = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)$ is the symmetric gradient of the velocity (often called the *strain rate*).

In this set of equations, (5) and (6) represent the compressible Stokes equations in which $\mathbf{v} = \mathbf{v}(\mathbf{x}, t)$ is the velocity field and $p = p(\mathbf{x}, t)$ the pressure field. Both fields depend on space $\mathbf{x}$ and time t. Fluid flow is driven by the gravity force that acts on the fluid and that is proportional to both the density of the fluid and the strength of the gravitational pull.

Coupled to this Stokes system is equation (7) for the temperature field $T = T(\mathbf{x}, t)$ that contains heat conduction terms as well as advection with the flow velocity $\mathbf{v}$. The right hand side terms of this equation correspond to

- internal heat production for example due to radioactive decay;
- friction (shear) heating;
- adiabatic compression of material;

In order to arrive at the set of equations that ASPECT solves, we need to

- neglect the $\partial p/\partial t$. WHY?
- neglect the $\partial \rho/\partial t$. WHY?

from equations above.

Also, their definition of the shear heating term Φ is:

$$\Phi = k_B(\boldsymbol{\nabla}\cdot\boldsymbol{v})^2 + 2\eta\dot{\boldsymbol{\varepsilon}}^d : \dot{\boldsymbol{\varepsilon}}^d$$

For many fluids the bulk viscosity k_B is very small and is often taken to be zero, an assumption known as the Stokes assumption: $k_B = \lambda + 2\eta/3 = 0$. Note that η is the dynamic viscosity and λ the second viscosity. Also,

$$\boldsymbol{\tau} = 2\eta\dot{\boldsymbol{\varepsilon}} + \lambda(\boldsymbol{\nabla}\cdot\boldsymbol{v})\mathbf{1}$$

but since $k_B = \lambda + 2\eta/3 = 0$, then $\lambda = -2\eta/3$ so

$$\boldsymbol{\tau} = 2\eta\dot{\boldsymbol{\varepsilon}} - \frac{2}{3}\eta(\boldsymbol{\nabla}\cdot\boldsymbol{v})\mathbf{1} = 2\eta\dot{\boldsymbol{\varepsilon}}^d$$

2.5 the Boussinesq approximation: an Incompressible flow

[from aspect manual] The Boussinesq approximation assumes that the density can be considered constant in all occurrences in the equations with the exception of the buoyancy term on the right hand side of (5). The primary result of this assumption is that the continuity equation (6) will now read

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

This implies that the strain rate tensor is deviatoric. Under the Boussinesq approximation, the equations are much simplified:

$$-\nabla \cdot [2\eta \dot{\boldsymbol{\varepsilon}}(\boldsymbol{v})] + \nabla p = \rho \boldsymbol{g} \qquad \text{in } \Omega, \tag{5}$$

$$\nabla \cdot (\rho \boldsymbol{v}) = 0 \qquad \text{in } \Omega, \tag{6}$$

$$\rho_0 C_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \nabla T \right) - \nabla \cdot k \nabla T = \rho H \qquad \text{in } \Omega \tag{7}$$

Note that all terms on the rhs of the temperature equations have disappeared, with the exception of the source term.

3 The building blocks of the Finite Element Method

3.1 Numerical integration

As we will see later, using the Finite Element method to solve problems involves computing integrals which are more often than not too complex to be computed analytically/exactly. We will then need to compute them numerically.

[wiki] In essence, the basic problem in numerical integration is to compute an approximate solution to a definite integral

$$\int_a^b f(x)dx$$

to a given degree of accuracy. This problem has been widely studied [**?**] and we know that if $f(x)$ is a smooth function, and the domain of integration is bounded, there are many methods for approximating the integral to the desired precision.

There are several reasons for carrying out numerical integration.

- The integrand $f(x)$ may be known only at certain points, such as obtained by sampling. Some embedded systems and other computer applications may need numerical integration for this reason.
- A formula for the integrand may be known, but it may be difficult or impossible to find an antiderivative that is an elementary function. An example of such an integrand is $f(x) = exp(-x^2)$, the antiderivative of which (the error function, times a constant) cannot be written in elementary form.
- It may be possible to find an antiderivative symbolically, but it may be easier to compute a numerical approximation than to compute the antiderivative. That may be the case if the antiderivative is given as an infinite series or product, or if its evaluation requires a special function that is not available.

3.1.1 in 1D - theory

The simplest method of this type is to let the interpolating function be a constant function (a polynomial of degree zero) that passes through the point $((a+b)/2, f((a+b)/2))$.

This is called the midpoint rule or rectangle rule.

$$\int_a^b f(x)dx \simeq (b-a)f(\frac{a+b}{2})$$

insert here figure

The interpolating function may be a straight line (an affine function, i.e. a polynomial of degree 1) passing through the points $(a, f(a))$ and $(b, f(b))$.

This is called the trapezoidal rule.

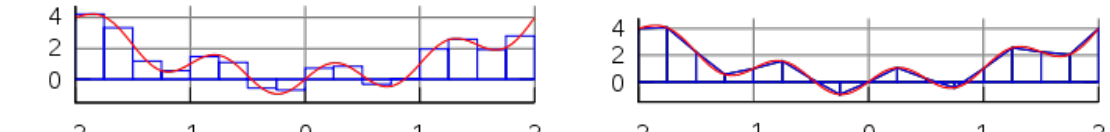
$$\int_a^b f(x)dx \simeq (b-a)\frac{f(a)+f(b)}{2}$$

insert here figure

For either one of these rules, we can make a more accurate approximation by breaking up the interval [a, b] into some number n of subintervals, computing an approximation for each subinterval, then adding up all the results. This is called a composite rule, extended rule, or iterated rule. For example, the composite trapezoidal rule can be stated as

$$\int_a^b f(x)dx \simeq \frac{b-a}{n}\left(\frac{f(a)}{2} + \sum_{k=1}^{n-1} f(a+k\frac{b-a}{n}) + \frac{f(b)}{2}\right)$$

where the subintervals have the form $[kh, (k+1)h]$, with $h = (b-a)/n$ and $k = 0, 1, 2, \ldots, n-1$.



a) b)

The interval $[-2, 2]$ is broken into 16 sub-intervals. The blue lines correspond to the approximation of the red curve by means of a) the midpoint rule, b) the trapezoidal rule.

There are several algorithms for numerical integration (also commonly called 'numerical quadrature', or simply 'quadrature') . Interpolation with polynomials evaluated at equally spaced points in $[a, b]$ yields the NewtonCotes formulas, of which the rectangle rule and the trapezoidal rule are examples. If we allow the intervals between interpolation points to vary, we find another group of quadrature formulas, such as the Gauss(ian) quadrature formulas. A Gaussian quadrature rule is typically more accurate than a NewtonCotes rule, which requires the same number of function evaluations, if the integrand is smooth (i.e., if it is sufficiently differentiable).

An $n-$point Gaussian quadrature rule, named after Carl Friedrich Gauss, is a quadrature rule constructed to yield an exact result for polynomials of degree $2n-1$ or less by a suitable choice of the points x_i and weights w_i for $i = 1, \ldots, n$.

The domain of integration for such a rule is conventionally taken as $[-1, 1]$, so the rule is stated as

$$\int_{-1}^{+1} f(x)dx = \sum_{i_q=1}^{n} w_{i_q} f(x_{i_q})$$

In this formula the x_{i_q} coordinate is the i-th root of the Legendre polynomial $P_n(x)$.

It is important to note that a Gaussian quadrature will only produce good results if the function $f(x)$ is well approximated by a polynomial function within the range $[-1, 1]$. As a consequence, the method is not, for example, suitable for functions with singularities.

Number of points, n	Points, x_i	Weights, w_i
1	0	2
2	$\pm\sqrt{\frac{1}{3}}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
4	$\pm\sqrt{\frac{3}{7}-\frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18+\sqrt{30}}{36}$
	$\pm\sqrt{\frac{3}{7}+\frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18-\sqrt{30}}{36}$
5	0	$\frac{128}{225}$
	$\pm\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	$\frac{322+13\sqrt{70}}{900}$
	$\pm\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	$\frac{322-13\sqrt{70}}{900}$

Gauss-Legendre points and their weights.

As shown in the above table, it can be shown that the weight values must fulfill the following condition:

$$\sum_{i_q} w_{i_q} = 2 \tag{8}$$

and it is worth noting that all quadrature point coordinates are symmetrical around the origin.

Since most quadrature formula are only valid on a specific interval, we now must address the problem of their use outside of such intervals. The solution turns out to be quite simple: one must carry out a change of variables from the interval $[a, b]$ to $[-1, 1]$.

We then consider the reduced coordinate $r \in [-1, 1]$ such that

$$r = \frac{2}{b-a}(x-a) - 1$$

This relationship can be reversed such that when r is known, its equivalent coordinate $x \in [a,b]$ can be computed:

$$x = \frac{b-a}{2}(1+r) + a$$

From this it follows that

$$dx = \frac{b-a}{2}dr$$

and then

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{+1} f(r)dr \simeq \frac{b-a}{2}\sum_{i_q=1}^{n} w_{i_q} f(r_{i_q})$$

3.1.2 in 1D - examples

example 1 Since we know how to carry out any required change of variables, we choose for simplicity $a=-1$, $b=+1$. Let us take for example $f(x)=\pi$. Then we can compute the integral of this function over the interval $[a,b]$ exactly:

$$I = \int_{-1}^{+1} f(x)dx = \pi\int_{-1}^{+1} dx = 2\pi$$

We can now use a Gauss-Legendre formula to compute this same integral:

$$I_{gq} = \int_{-1}^{+1} f(x)dx = \sum_{i_q=1}^{n_q} w_{i_q} f(x_{i_q}) = \sum_{i_q=1}^{n_q} w_{i_q}\pi = \pi \underbrace{\sum_{i_q=1}^{n_q} w_{i_q}}_{=2} = 2\pi$$

where we have used the property of the weight values of Eq.(8). Since the actual number of points was never specified, this result is valid for all quadrature rules.

example 2 Let us now take $f(x)=mx+p$ and repeat the same exercise:

$$I = \int_{-1}^{+1} f(x)dx = \int_{-1}^{+1} (mx+p)dx = [\frac{1}{2}mx^2 + px]_{-1}^{+1} = 2p$$

$$I_{gq} = \int_{-1}^{+1} f(x)dx = \sum_{i_q=1}^{n_q} w_{i_q} f(x_{i_q}) = \sum_{i_q=1}^{n_q} w_{i_q}(mx_{i_q}+p) = m\underbrace{\sum_{i_q=1}^{n_q} w_{i_q}x_{i_q}}_{=0} + p\underbrace{\sum_{i_q=1}^{n_q} w_{i_q}}_{=2} = 2p$$

since the quadrature points are symmetric w.r.t. to zero on the x-axis. Once again the quadrature is able to compute the exact value of this integral: this makes sense since an n-point rule exactly integrates a $2n-1$ order polynomial such that a 1 point quadrature exactly integrates a first order polynomial like the one above.

example 3 Let us now take $f(x)=x^2$. We have

$$I = \int_{-1}^{+1} f(x)dx = \int_{-1}^{+1} x^2 dx = [\frac{1}{3}x^3]_{-1}^{+1} = \frac{2}{3}$$

and

$$I_{gq} = \int_{-1}^{+1} f(x)dx = \sum_{i_q=1}^{n_q} w_{i_q} f(x_{i_q}) = \sum_{i_q=1}^{n_q} w_{i_q} x_{i_q}^2$$

- $n_q = 1$: $x_{iq}^{(1)} = 0$, $w_{i_q} = 2$. $I_{gq} = 0$
- $n_q = 2$: $x_q^{(1)} = -1/\sqrt{3}$, $x_q^{(2)} = 1/\sqrt{3}$, $w_q^{(1)} = w_q^{(2)} = 1$. $I_{gq} = \frac{2}{3}$
- It also works $\forall n_q > 2$!

3.1.3 in 2D/3D - theory

Let us now turn to a two-dimensional integral of the form

$$I = \int_{-1}^{+1} \int_{-1}^{+1} f(x,y) dx dy$$

The equivalent Gaussian quadrature writes:

$$I_{gq} \simeq \sum_{i_q=1}^{n_q} \sum_{j_q}^{n_q} f(x_{i_q}, y_{j_q}) w_{i_q} w_{j_q}$$

3.2 The mesh

3.3 A bit of FE terminology

We introduce here some terminology for efficient element descriptions [**?**]:

- For triangles/tetrahedra, the designation $P_m \times P_n$ means that each component of the velocity is approximated by continuous piecewise complete Polynomials of degree m and pressure by continuous piecewise complete Polynomials of degree n. For example $P_2 \times P_1$ means

  $$u \sim a_1 + a_2 x + a_3 y + a_4 xy + a_5 x^2 + a_6 y^2$$

  with similar approximations for v, and

  $$p \sim b_1 + b_2 x + b_3 y$$

  Both velocity and pressure are continuous across element boundaries, and each triangular element contains 6 velocity nodes and three pressure nodes.

- For the same families, $P_m \times P_{-n}$ is as above, except that pressure is approximated via piecewise *discontinuous* polynomials of degree n. For instance, $P_2 \times P_{-1}$ is the same as $P_2 P_1$ except that pressure is now an independent linear function in each element and therefore discontinuous at element boundaries.

- For quadrilaterals/hexahedra, the designation $Q_m \times Q_n$ means that each component of the velocity is approximated by a continuous piecewise polynomial of degree m *in each direction* on the quadrilateral and likewise for pressure, except that the polynomial is of degree n. For instance, $Q_2 \times Q_1$ means

  $$u \sim a_1 + a_2 x + a_3 y + a_4 xy + a_5 x^2 + a_6 y^2 + a_7 x^2 y + a_8 xy^2 + a_9 x^2 y^2$$

  and

  $$p \sim b_1 + b_2 x + b_3 y + b_4 xy$$

- For these same families, $Q_m \times Q_{-n}$ is as above, except that the pressure approximation is not continuous at element boundaries.

- Again for the same families, $Q_m \times P_{-n}$ indicates the same velocity approximation with a pressure approximation that is a discontinuous complete piecewise polynomial of degree n (not of degree n in each direction !)

- The designation P_m^+ or Q_m^+ means that some sort of bubble function was added to the polynomial approximation for the velocity. You may also find the term 'enriched element' in the literature.

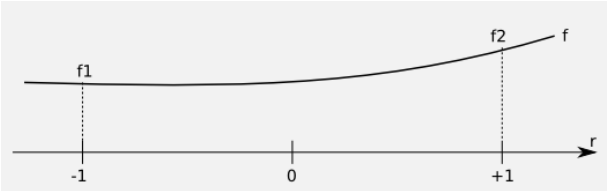
- Finally, for $n = 0$, we have piecewise-constant pressure, and we omit the minus sign for simplicity.

Another point which needs to be clarified is the use of so-called 'conforming elements' (or 'non-conforming elements'). Following again [**?**], conforming velocity elements are those for which the basis functions for a subset of H^1 for the continuous problem (the first derivatives and their squares are integrable in Ω). For instance, the rotated $Q_1 \times P_0$ element of Rannacher and Turek (see section **??**) is such that the velocity is discontinous across element edges, so that the derivative does not exist there. Another typical example of non-conforming element is the Crouzeix-Raviart element [**?**].

3.4 Elements and basis functions in 1D

3.4.1 Linear basis functions (Q_1)

Let $f(r)$ be a C^1 function on the interval $[-1:1]$ with $f(-1) = f_1$ and $f(1) = f_2$.



Let us assume that the function $f(r)$ is to be approximated on $[-1, 1]$ by the first order polynomial

$$f(r) = a + br \tag{9}$$

Then it must fulfill

$$\begin{aligned} f(r=-1) &= a - b = f_1 \\ f(r=+1) &= a + b = f_2 \end{aligned}$$

This leads to

$$a = \frac{1}{2}(f_1 + f_2) \qquad b = \frac{1}{2}(-f_1 + f_2)$$

and then replacing a, b in Eq. (9) by the above values on gets

$$f(r) = \left[\frac{1}{2}(1-r)\right] f_1 + \left[\frac{1}{2}(1+r)\right] f_2$$

or

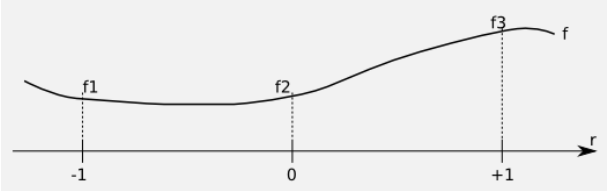
$$f(r) = \sum_{i=1}^{2} N_i(r) f_1$$

with

$$\begin{aligned} N_1(r) &= \frac{1}{2}(1-r) \\ N_2(r) &= \frac{1}{2}(1+r) \end{aligned} \tag{10}$$

3.4.2 Quadratic basis functions (Q_2)

Let $f(r)$ be a C^1 function on the interval $[-1:1]$ with $f(-1) = f_1$, $f(0) = f_2$ and $f(1) = f_3$.



Let us assume that the function $f(r)$ is to be approximated on $[-1, 1]$ by the second order polynomial

$$f(r) = a + br + cr^2 \tag{11}$$

Then it must fulfill

$$\begin{aligned} f(r=-1) &= a - b + c = f_1 \\ f(r=0) &= a \qquad\quad = f_2 \\ f(r=+1) &= a + b + c = f_3 \end{aligned}$$

This leads to

$$a = f_2 \quad b = \frac{1}{2}(-f1 + f3) \quad c = \frac{1}{2}(f_1 + f_3 - 2f_2)$$

and then replacing a, b, c in Eq. (11) by the above values on gets

$$f(r) = \left[\frac{1}{2}r(r-1)\right] f_1 + (1-r^2)f_2 + \left[\frac{1}{2}r(r+1)\right] f_3$$

or,

$$f(r) = \sum_{i=1}^{3} N_i(r) f_i$$

with

$$\begin{aligned} N_1(r) &= \frac{1}{2}r(r-1) \\ N_2(r) &= (1-r^2) \\ N_3(r) &= \frac{1}{2}r(r+1) \end{aligned} \tag{12}$$

3.4.3 Cubic basis functions (Q_3)

The 1D basis polynomial is given by

$$f(r) = a + br + cr^2 + dr^3$$

with the nodes at position -1,-1/3, +1/3 and +1.

$$\begin{aligned} f(-1) &= a - b + c - d = f_1 \\ f(-1/3) &= a - \frac{b}{3} + \frac{c}{9} - \frac{d}{27} = f_2 \\ f(+1/3) &= a - \frac{b}{3} + \frac{c}{9} - \frac{d}{27} = f_3 \\ f(+1) &= a + b + c + d = f_4 \end{aligned}$$

Adding the first and fourth equation and the second and third, one arrives at

$$f_1 + f_4 = 2a + 2c \qquad f_2 + f_3 = 2a + \frac{2c}{9}$$

and finally:

$$a = \frac{1}{16}(-f_1 + 9f_2 + 9f_3 - f_4)$$

$$c = \frac{9}{16}(f_1 - f_2 - f_3 + f_4)$$

Combining the original 4 equations in a different way yields

$$2b + 2d = f_4 - f_1 \qquad \frac{2b}{3} + \frac{2d}{27} = f_3 - f_2$$

so that

$$b = \frac{1}{16}(f_1 - 27f_2 + 27f_3 - f_4)$$

$$d = \frac{9}{16}(-f_1 + 3f_2 - 3f_3 + f_4)$$

Finally,

$$
\begin{aligned}
f(r) &= a + b + cr^2 + dr^3 \\
&= \frac{1}{16}(-1 + r + 9r^2 - 9r^3)f_1 \\
&+ \frac{1}{16}(9 - 27r - 9r^2 + 27r^3)f_2 \\
&+ \frac{1}{16}(9 + 27r - 9r^2 - 27r^3)f_3 \\
&+ \frac{1}{16}(-1 - r + 9r^2 + 9r^3)f_4 \\
&= \sum_{i=1}^{4} N_i(r) f_i
\end{aligned}
$$

where

$$
\begin{aligned}
N_1 &= \frac{1}{16}(-1 + r + 9r^2 - 9r^3) \\
N_2 &= \frac{1}{16}(9 - 27r - 9r^2 + 27r^3) \\
N_3 &= \frac{1}{16}(9 + 27r - 9r^2 - 27r^3) \\
N_4 &= \frac{1}{16}(-1 - r + 9r^2 + 9r^3)
\end{aligned}
$$

Verification:

- Let us assume $f(r) = C$, then

$$\hat{f}(r) = \sum N_i(r) f_i = \sum_i N_i C = C \sum_i N_i = C$$

  so that a constant function is exactly reproduced, as expected.

- Let us assume $f(r) = r$, then $f_1 = -1$, $f_2 = -1/3$, $f_3 = 1/3$ and $f_4 = +1$. We then have

$$
\begin{aligned}
\hat{f}(r) &= \sum N_i(r) f_i \\
&= -N_1(r) - \frac{1}{3}N_2(r) + \frac{1}{3}N_3(r) + N_4(r) \\
&= [-(-1 + r + 9r^2 - 9r^3) \\
&\quad -\frac{1}{3}(9 - 27r - 9r^2 - 27r^3) \\
&\quad +\frac{1}{3}(9 + 27r - 9r^2 + 27r^3) \\
&\quad +(-1 - r + 9r^2 + 9r^3)]/16 \\
&= [-r + 9r + 9r - r]/16 + ...0... \\
&= r
\end{aligned}
\tag{13}
$$

The basis functions derivative are given by

$$
\begin{aligned}
\frac{\partial N_1}{\partial r} &= \frac{1}{16}(1 + 18r - 27r^2) \\
\frac{\partial N_2}{\partial r} &= \frac{1}{16}(-27 - 18r + 81r^2) \\
\frac{\partial N_3}{\partial r} &= \frac{1}{16}(+27 - 18r - 81r^2) \\
\frac{\partial N_4}{\partial r} &= \frac{1}{16}(-1 + 18r + 27r^2)
\end{aligned}
$$

Verification:

- Let us assume $f(r) = C$, then

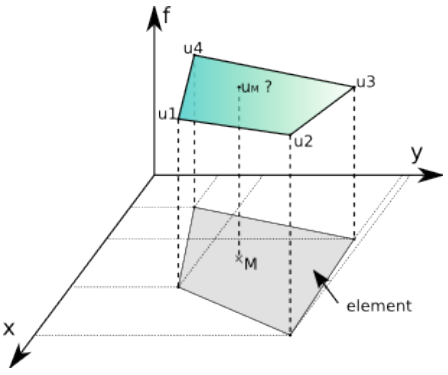
$$
\begin{aligned}
\frac{\partial \hat{f}}{\partial r} &= \sum_i \frac{\partial N_i}{\partial r} f_i \\
&= C \sum_i \frac{\partial N_i}{\partial r} \\
&= \frac{C}{16}[(1 + 18r - 27r^2) \\
&\quad +(-27 - 18r + 81r^2) \\
&\quad +(+27 - 18r - 81r^2) \\
&\quad +(-1 + 18r + 27r^2)] \\
&= 0
\end{aligned}
$$

- Let us assume $f(r) = r$, then $f_1 = -1$, $f_2 = -1/3$, $f_3 = 1/3$ and $f_4 = +1$. We then have

$$
\begin{aligned}
\frac{\partial \hat{f}}{\partial r} &= \sum_i \frac{\partial N_i}{\partial r} f_i \\
&= \frac{1}{16}[-(1 + 18r - 27r^2) \\
&\quad -\frac{1}{3}(-27 - 18r + 81r^2) \\
&\quad +\frac{1}{3}(+27 - 18r - 81r^2) \\
&\quad +(-1 + 18r + 27r^2)] \\
&= \frac{1}{16}[-2 + 18 + 54r^2 - 54r^2] \\
&= 1
\end{aligned}
$$

3.5 Elements and basis functions in 2D

Let us for a moment consider a single quadrilateral element in the xy-plane, as shown on the following figure:



Let us assume that we know the values of a given field u at the vertices. For a given point M inside the element in the plane, what is the value of the field u at this point? It makes sense to postulate that $u_M = u(x_M, y_M)$ will be given by

$$u_M = \phi(u_1, u_2, u_3, u_4, x_M, y_M)$$

where ϕ is a function to be determined. Although ϕ is not unique, we can decide to express the value u_M as a weighed sum of the values at the vertices u_i. One option could be to assign all four vertices the same weight, say $1/4$ so that $u_M = (u_1 + u_2 + u_3 + u_4)/4$, i.e. u_M is simply given by the arithmetic mean of the vertices values. This approach suffers from a major drawback as it does not use the location of point M inside the element. For instance, when $(x_M, y_M) \to (x_2, y_2)$ we expect $u_M \to u_2$.

In light of this, we could now assume that the weights would depend on the position of M in a continuous fashion:

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

where the N_i are continous ("well behaved") functions which have the property:

$$N_i(x_j, y_j) = \delta_{ij}$$

or, in other words:

$$N_3(x_1, y_1) = 0 \tag{14}$$
$$N_3(x_2, y_2) = 0 \tag{15}$$
$$N_3(x_3, y_3) = 1 \tag{16}$$
$$N_3(x_4, y_4) = 0 \tag{17}$$

The functions N_i are commonly called basis functions.

Omitting the M subscripts for any point inside the element, the velocity components u and v are given by:

$$\hat{u}(x, y) = \sum_{i=1}^{4} N_i(x, y)\ u_i \tag{18}$$

$$\hat{v}(x, y) = \sum_{i=1}^{4} N_i(x, y)\ v_i \tag{19}$$

Rather interestingly, one can now easily compute velocity gradients (and therefore the strain rate tensor) since we have assumed the basis functions to be "well behaved" (in this case differentiable):

$$\dot{\epsilon}_{xx}(x, y) = \frac{\partial u}{\partial x} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial x}\ u_i \tag{20}$$

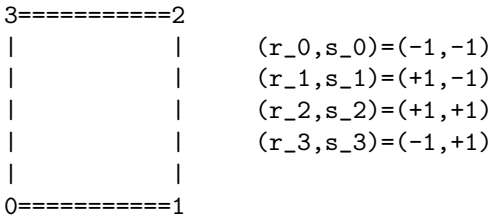
$$\dot{\epsilon}_{yy}(x, y) = \frac{\partial v}{\partial y} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial y}\ v_i \tag{21}$$

$$\dot{\epsilon}_{xy}(x, y) = \frac{1}{2}\frac{\partial u}{\partial y} + \frac{1}{2}\frac{\partial v}{\partial x} = \frac{1}{2}\sum_{i=1}^{4} \frac{\partial N_i}{\partial y}\ u_i + \frac{1}{2}\sum_{i=1}^{4} \frac{\partial N_i}{\partial x}\ v_i \tag{22}$$

How we actually obtain the exact form of the basis functions is explained in the coming section.

3.5.1 Bilinear basis functions in 2D (Q_1)

In this section, we place ourselves in the most favorables case, i.e. the element is a square defined by $-1 < r < 1$, $-1 < s < 1$ in the Cartesian coordinates system (r, s):



This element is commonly called the reference element. How we go from the (x, y) coordinate system to the (r, s) once and vice versa will be dealt later on. For now, the basis functions in the above reference element and in the reduced coordinates system (r, s) are given by:

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

The partial derivatives of these functions with respect to r ans s automatically follow:

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

Let us go back to Eq.(19). And let us assume that the function $v(r, s) = C$ so that $v_i = C$ for $i = 1, 2, 3, 4$. It then follows that

$$\hat{v}(r,s) = \sum_{i=1}^{4} N_i(r,s)\ v_i = C \sum_{i=1}^{4} N_i(r,s) = C[N_1(r,s) + N_2(r,s) + N_3(r,s) + N_4(r,s)] = C$$

This is a very important property: if the v function used to assign values at the vertices is constant, then the value of $\hat{v}$ *anywhere* in the element is exactly C. If we now turn to the derivatives of v with respect to r and s:

$$\frac{\partial \hat{v}}{\partial r}(r,s) = \sum_{i=1}^{4} \frac{\partial N_i}{\partial r}(r,s)\ v_i = C \sum_{i=1}^{4} \frac{\partial N_i}{\partial r}(r,s) = C\left[-0.25(1-s) + 0.25(1-s) + 0.25(1+s) - 0.25(1+s)\right] = 0$$

$$\frac{\partial \hat{v}}{\partial s}(r,s) = \sum_{i=1}^{4} \frac{\partial N_i}{\partial s}(r,s)\ v_i = C \sum_{i=1}^{4} \frac{\partial N_i}{\partial s}(r,s) = C\left[-0.25(1-r) - 0.25(1+r) + 0.25(1+r) + 0.25(1-r)\right] = 0$$

We reassuringly find that the derivative of a constant field anywhere in the element is exactly zero.

If we now choose $v(r,s) = ar + bs$ with a and b two constant scalars, we find:

$$\begin{aligned}
\hat{v}(r,s) &= \sum_{i=1}^{4} N_i(r,s)\, v_i && (23)\\
&= \sum_{i=1}^{4} N_i(r,s)(ar_i + bs_i) && (24)\\
&= \underbrace{a\sum_{i=1}^{4} N_i(r,s) r_i}_{r} + \underbrace{b\sum_{i=1}^{4} N_i(r,s) s_i}_{s} && (25)\\
&= a\,[0.25(1-r)(1-s)(-1) + 0.25(1+r)(1-s)(+1) + 0.25(1+r)(1+s)(+1) + 0.25(1-r)(1+s)(-1)]\\
&+ b\,[0.25(1-r)(1-s)(-1) + 0.25(1+r)(1-s)(-1) + 0.25(1+r)(1+s)(+1) + 0.25(1-r)(1+s)(+1)]\\
&= a\,[-0.25(1-r)(1-s) + 0.25(1+r)(1-s) + 0.25(1+r)(1+s) - 0.25(1-r)(1+s)]\\
&+ b\,[-0.25(1-r)(1-s) - 0.25(1+r)(1-s) + 0.25(1+r)(1+s) + 0.25(1-r)(1+s)]\\
&= ar + bs && (26)
\end{aligned}$$

verify above eq. This set of bilinear shape functions is therefore capable of exactly representing a bilinear field. The derivatives are:

$$\begin{aligned}
\frac{\partial \hat{v}}{\partial r}(r,s) &= \sum_{i=1}^{4} \frac{\partial N_i}{\partial r}(r,s)\, v_i && (27)\\
&= a\sum_{i=1}^{4} \frac{\partial N_i}{\partial r}(r,s) r_i + b\sum_{i=1}^{4} \frac{\partial N_i}{\partial r}(r,s) s_i && (28)\\
&= a\,[-0.25(1-s)(-1) + 0.25(1-s)(+1) + 0.25(1+s)(+1) - 0.25(1+s)(-1)]\\
&+ b\,[-0.25(1-s)(-1) + 0.25(1-s)(-1) + 0.25(1+s)(+1) - 0.25(1+s)(+1)]\\
&= \frac{a}{4}\,[(1-s) + (1-s) + (1+s) + (1+s)]\\
&+ \frac{b}{4}\,[(1-s) - (1-s) + (1+s) - (1+s)]\\
&= a && (29)
\end{aligned}$$

Here again, we find that the derivative of the bilinear field inside the element is exact: $\frac{\partial \hat{v}}{\partial r} = \frac{\partial v}{\partial r}$.

However, following the same methodology as above, one can easily prove that this is no more true for polynomials of degree strivtly higher than 1. This fact has serious consequences: if the solution to the problem at hand is for instance a parabola, the Q_1 shape functions cannot represent the solution properly, but only by approximating the parabola in each element by a line. As we will see later, Q_2 basis functions can remedy this problem by containing themselves quadratic terms.

3.5.2 Biquadratic basis functions in 2D (Q_2)

Inside an element the local numbering of the nodes is as follows:

```
3=====6=====2
|     |     |   (r_0,s_0)=(-1,-1)   (r_4,s_4)=( 0,-1)
|     |     |   (r_1,s_1)=(+1,-1)   (r_5,s_5)=(+1, 0)
7=====8=====5   (r_2,s_2)=(+1,+1)   (r_6,s_6)=( 0,+1)
|     |     |   (r_3,s_3)=(-1,+1)   (r_7,s_7)=(-1, 0)
|     |     |                       (r_8,s_8)=( 0, 0)
0=====4=====1
```

The velocity shape functions are then given by:

$$
\begin{aligned}
N_0(r,s) &= \frac{1}{2}r(r-1)\frac{1}{2}s(s-1) \\
N_1(r,s) &= \frac{1}{2}r(r+1)\frac{1}{2}s(s-1) \\
N_2(r,s) &= \frac{1}{2}r(r+1)\frac{1}{2}s(s+1) \\
N_3(r,s) &= \frac{1}{2}r(r-1)\frac{1}{2}s(s+1) \\
N_4(r,s) &= (1-r^2)\frac{1}{2}s(s-1) \\
N_5(r,s) &= \frac{1}{2}r(r+1)(1-s^2) \\
N_6(r,s) &= (1-r^2)\frac{1}{2}s(s+1) \\
N_7(r,s) &= \frac{1}{2}r(r-1)(1-s^2) \\
N_8(r,s) &= (1-r^2)(1-s^2)
\end{aligned}
$$

and their derivatives by:

$$
\begin{aligned}
\frac{\partial N_0}{\partial r} &= \frac{1}{2}(2r-1)\frac{1}{2}s(s-1) & \frac{\partial N_0}{\partial s} &= \frac{1}{2}r(r-1)\frac{1}{2}(2s-1) \\
\frac{\partial N_1}{\partial r} &= \frac{1}{2}(2r+1)\frac{1}{2}s(s-1) & \frac{\partial N_1}{\partial s} &= \frac{1}{2}r(r+1)\frac{1}{2}(2s-1) \\
\frac{\partial N_2}{\partial r} &= \frac{1}{2}(2r+1)\frac{1}{2}s(s+1) & \frac{\partial N_2}{\partial s} &= \frac{1}{2}r(r+1)\frac{1}{2}(2s+1) \\
\frac{\partial N_3}{\partial r} &= \frac{1}{2}(2r-1)\frac{1}{2}s(s+1) & \frac{\partial N_3}{\partial s} &= \frac{1}{2}r(r-1)\frac{1}{2}(2s+1) \\
\frac{\partial N_4}{\partial r} &= (-2r)\frac{1}{2}s(s-1) & \frac{\partial N_4}{\partial s} &= (1-r^2)\frac{1}{2}(2s-1) \\
\frac{\partial N_5}{\partial r} &= \frac{1}{2}(2r+1)(1-s^2) & \frac{\partial N_5}{\partial s} &= \frac{1}{2}r(r+1)(-2s) \\
\frac{\partial N_6}{\partial r} &= (-2r)\frac{1}{2}s(s+1) & \frac{\partial N_6}{\partial s} &= (1-r^2)\frac{1}{2}(2s+1) \\
\frac{\partial N_7}{\partial r} &= \frac{1}{2}(2r-1)(1-s^2) & \frac{\partial N_7}{\partial s} &= \frac{1}{2}r(r-1)(-2s) \\
\frac{\partial N_8}{\partial r} &= (-2r)(1-s^2) & \frac{\partial N_8}{\partial s} &= (1-r^2)(-2s)
\end{aligned}
$$

3.5.3 Bicubic basis functions in 2D (Q_3)

Inside an element the local numbering of the nodes is as follows:

```
12===13===14===15   (r,s)_{00}=(-1,-1)      (r,s)_{08}=(-1,+1/3)
||   ||   ||   ||   (r,s)_{01}=(-1/3,-1)    (r,s)_{09}=(-1/3,+1/3)
08===09===10===11   (r,s)_{02}=(+1/3,-1)    (r,s)_{10}=(+1/3,+1/3)
||   ||   ||   ||   (r,s)_{03}=(+1,-1)      (r,s)_{11}=(+1,+1/3)
04===05===06===07   (r,s)_{04}=(-1,-1/3)    (r,s)_{12}=(-1,+1)
||   ||   ||   ||   (r,s)_{05}=(-1/3,-1/3)  (r,s)_{13}=(-1/3,+1)
00===01===02===03   (r,s)_{06}=(+1/3,-1/3)  (r,s)_{14}=(+1/3,+1)
                    (r,s)_{07}=(+1,-1/3)    (r,s)_{15}=(+1,+1)
```

The velocity shape functions are given by:

$$N_1(r) = (-1 + r + 9r^2 - 9r^3)/16 \qquad N_1(t) = (-1 + t + 9t^2 - 9t^3)/16$$
$$N_2(r) = (+9 - 27r - 9r^2 + 27r^3)/16 \qquad N_2(t) = (+9 - 27t - 9t^2 + 27t^3)/16$$
$$N_3(r) = (+9 + 27r - 9r^2 - 27r^3)/16 \qquad N_3(t) = (+9 + 27t - 9t^2 - 27t^3)/16$$
$$N_4(r) = (-1 - r + 9r^2 + 9r^3)/16 \qquad N_4(t) = (-1 - t + 9t^2 + 9t^3)/16$$

$$\begin{aligned}
N_{01}(r,s) &= N_1(r)N_1(s) = (-1 + r + 9r^2 - 9r^3)/16 * (-1 + t + 9s^2 - 9s^3)/16 \\
N_{02}(r,s) &= N_2(r)N_1(s) = (+9 - 27r - 9r^2 + 27r^3)/16 * (-1 + t + 9s^2 - 9s^3)/16 \\
N_{03}(r,s) &= N_3(r)N_1(s) = (+9 + 27r - 9r^2 - 27r^3)/16 * (-1 + t + 9s^2 - 9s^3)/16 \\
N_{04}(r,s) &= N_4(r)N_1(s) = (-1 - r + 9r^2 + 9r^3)/16 * (-1 + t + 9s^2 - 9s^3)/16 \\
N_{05}(r,s) &= N_1(r)N_2(s) = (-1 + r + 9r^2 - 9r^3)/16 * (9 - 27s - 9s^2 + 27s^3)/16 \\
N_{06}(r,s) &= N_2(r)N_2(s) = (+9 - 27r - 9r^2 + 27r^3)/16 * (9 - 27s - 9s^2 + 27s^3)/16 \\
N_{07}(r,s) &= N_3(r)N_2(s) = (+9 + 27r - 9r^2 - 27r^3)/16 * (9 - 27s - 9s^2 + 27s^3)/16 \\
N_{08}(r,s) &= N_4(r)N_2(s) = (-1 - r + 9r^2 + 9r^3)/16 * (9 - 27s - 9s^2 + 27s^3)/16 \\
N_{09}(r,s) &= N_1(r)N_3(s) = \qquad (30) \\
N_{10}(r,s) &= N_2(r)N_3(s) = \qquad (31) \\
N_{11}(r,s) &= N_3(r)N_3(s) = \qquad (32) \\
N_{12}(r,s) &= N_4(r)N_3(s) = \qquad (33) \\
N_{13}(r,s) &= N_1(r)N_4(s) = \qquad (34) \\
N_{14}(r,s) &= N_2(r)N_4(s) = \qquad (35) \\
N_{15}(r,s) &= N_3(r)N_4(s) = \qquad (36) \\
N_{16}(r,s) &= N_4(r)N_4(s) = \qquad (37)
\end{aligned}$$

4 Solving the Stokes equations with the FEM

In the case of an incompressible flow, we have seen that the continuity (mass conservation) equation takes the simple form $\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$. In other word flow takes place under the constraint that the divergence of its velocity field is exactly zero eveywhere (solenoidal constraint), i.e. it is divergence free.

We see that the pressure in the momentum equation is then a degree of freedom which is needed to satisfy the incompressibilty constraint (and it is not related to any constitutive equation) [**?**]. In other words the pressure is acting as a Lagrange multiplier of the incompressibility constraint.

Various approaches have been proposed in the literature to deal with the incompressibility constraint but we will only focus on the penalty method (section 4.2) and the so-called mixed finite element method 4.3.

4.1 strong and weak forms

The strong form consists of the governing equation and the boundary conditions, i.e. the mass, momentum and energy conservation equations supplemented with Dirichlet and/or Neumann boundary conditions on (parts of) the boundary.

To develop the finite element formulation, the partial differential equations must be restated in an integral form called the weak form. In essence the PDEs are first multiplied by an arbitrary function and integrated over the domain.

4.2 The penalty approach

In order to impose the incompressibility constraint, two widely used procedures are available, namely the Lagrange multiplier method and the penalty method [**?**, **?**]. The latter is implemented in ELEFANT, which allows for the elimination of the pressure variable from the momentum equation (resulting in a reduction of the matrix size).

Mathematical details on the origin and validity of the penalty approach applied to the Stokes problem can for instance be found in [**?**], [**?**] or [**?**].

The penalty formulation of the mass conservation equation is based on a relaxation of the incompressibility constraint and writes

$$\boldsymbol{\nabla} \cdot \boldsymbol{v} + \frac{p}{\lambda} = 0 \tag{38}$$

where λ is the penalty parameter, that can be interpreted (and has the same dimension) as a bulk viscosity. It is equivalent to say that the material is weakly compressible. It can be shown that if one chooses λ to be a sufficiently large number, the continuity equation $\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$ will be approximately satisfied in the finite element solution. The value of λ is often recommended to be 6 to 7 orders of magnitude larger than the shear viscosity [**?**, **?**].

Equation (38) can be used to eliminate the pressure in Eq. (**??**) so that the mass and momentum conservation equations fuse to become :

$$\boldsymbol{\nabla} \cdot (2\eta\dot{\varepsilon}(\boldsymbol{v})) + \lambda\boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \boldsymbol{v}) = \rho\boldsymbol{g} = 0 \tag{39}$$

[**?**] have established the equivalence for incompressible problems between the reduced integration of the penalty term and a mixed Finite Element approach if the pressure nodes coincide with the integration points of the reduced rule.

In the end, the elimination of the pressure unknown in the Stokes equations replaces the original saddle-point Stokes problem [**?**] by an elliptical problem, which leads to a symmetric positive definite (SPD) FEM matrix. This is the major benefit of the penalized approach over the full indefinite solver with the velocity-pressure variables. Indeed, the SPD character of the matrix lends itself to efficient solving stragegies and is less memory-demanding since it is sufficient to store only the upper half of the matrix including the diagonal [**?**] .

list codes which use this approach

The stress tensor $\boldsymbol{\sigma}$ is symmetric (*i.e.* $\sigma_{ij} = \sigma_{ji}$). For simplicity I will now focus on a Stokes flow in two dimensions.

Since the penalty formulation is only valid for incompressible flows, then $\dot{\boldsymbol{\epsilon}} = \dot{\boldsymbol{\epsilon}}^d$ so that the d superscript is ommitted in what follows. The stress tensor can also be cast in vector format:

$$\begin{aligned}
\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} &= \begin{pmatrix} -p \\ -p \\ 0 \end{pmatrix} + 2\eta \begin{pmatrix} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{pmatrix} \\
&= \lambda \begin{pmatrix} \dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ 0 \end{pmatrix} + 2\eta \begin{pmatrix} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{pmatrix} \\
&= \left[\lambda \underbrace{\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{\boldsymbol{K}} + \eta \underbrace{\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{C}} \right] \cdot \begin{pmatrix} \frac{\partial u}{\partial x} \\ \\ \frac{\partial v}{\partial y} \\ \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{pmatrix}
\end{aligned}$$

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

$$\begin{pmatrix} \frac{\partial u}{\partial x} \\ \\ \frac{\partial v}{\partial y} \\ \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial y} \\ \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial x} \end{pmatrix}}_{\boldsymbol{B}} \cdot \underbrace{\begin{pmatrix} u1 \\ v1 \\ u2 \\ v2 \\ u3 \\ v3 \\ u4 \\ v4 \end{pmatrix}}_{\boldsymbol{V}}$$

Finally,

$$\vec{\sigma} = \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = (\lambda \boldsymbol{K} + \eta \boldsymbol{C}) \cdot \boldsymbol{B} \cdot \boldsymbol{V}$$

We will now establish the weak form of the momentum conservation equation. We start again from

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

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

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

We can integrate by parts and drop the surface term[1]:

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

or,

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

[1] We will come back to this at a later stage

Let $i = 1, 2, 3, 4$ and stack the resulting four equations on top of one another.

$$\int_{\Omega_e} \begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_1}{\partial y} \\ & & \\ 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} d\Omega = \int_{\Omega_e} N_1 \begin{pmatrix} b_x \\ b_y \end{pmatrix} d\Omega \tag{40}$$

$$\int_{\Omega_e} \begin{pmatrix} \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_2}{\partial y} \\ & & \\ 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} d\Omega = \int_{\Omega_e} N_i \begin{pmatrix} b_x \\ b_y \end{pmatrix} d\Omega \tag{41}$$

$$\int_{\Omega_e} \begin{pmatrix} \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_3}{\partial y} \\ & & \\ 0 & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} d\Omega = \int_{\Omega_e} N_3 \begin{pmatrix} b_x \\ b_y \end{pmatrix} d\Omega \tag{42}$$

$$\int_{\Omega_e} \begin{pmatrix} \frac{\partial N_4}{\partial x} & 0 & \frac{\partial N_4}{\partial y} \\ & & \\ 0 & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} d\Omega = \int_{\Omega_e} N_4 \begin{pmatrix} b_x \\ b_y \end{pmatrix} d\Omega \tag{43}$$

We easily recognize $\boldsymbol{B}^T$ inside the integrals! Let us define

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

then we can write

$$\int_{\Omega_e} \boldsymbol{B}^T \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} d\Omega = \int_{\Omega_e} \boldsymbol{N}_b d\Omega$$

and finally:

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

Since V contains the velocities at the corners, it does not depend on the x or y coordinates so it can be taking outside of the integral:

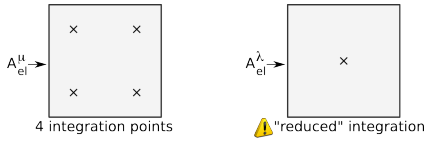
$$\underbrace{\left(\int_{\Omega_e} \boldsymbol{B}^T \cdot [\lambda \boldsymbol{K} + \eta \boldsymbol{C}] \cdot \boldsymbol{B} d\Omega \right)}_{A_{el}(8\times 8)} \cdot \underbrace{\boldsymbol{V}}_{(8x1)} = \underbrace{\int_{\Omega_e} \boldsymbol{N}_b d\Omega}_{B_{el}(8\times 1)}$$

or,

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

INTEGRATION - MAPPING
reduced integration [**?**]

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



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

4.3 The mixed FEM

5 Solving the elastic equations with the FEM

6 Additional techniques

6.1 Picard and Newton

6.2 The SUPG formulation for the energy equation

6.3 Tracking materials and/or interfaces

6.4 Dealing with a free surface

6.5 Convergence criterion for nonlinear iterations

6.6 Static condensation

6.7 The method of manufactured solutions

The method of manufactured solutions is a relatively simple way of carrying out code verification. In essence, one postulates a solution for the PDE at hand (as well as the proper boundary conditions), inserts it in the PDE and computes the corresponding source term. The same source term and boundary conditions will then be used in a numerical simulation so that the computed solution can be compared with the (postulated) true analytical solution.

Examples of this approach are to be found in [**?**, **?**, **?**].

- ▷ *python_codes/fieldstone*
- ▷ *python_codes/fieldstone_saddlepoint*
- ▷ *python_codes/fieldstone_saddlepoint_q2q1*
- ▷ *python_codes/fieldstone_saddlepoint_q3q2*
- ▷ *python_codes/fieldstone_burstedde*

6.7.1 Analytical benchmark I

Taken from [**?**].

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

Note that the pressure obeys $\int_\Omega p\, d\Omega = 0$

The corresponding components of the body force $\boldsymbol{b}$ are prescribed as

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

With this prescribed body force, the exact solution is

6.7.2 Analytical benchmark II

Taken from [**?**]

$$u(x,y) = x+x^2-2xy+x^3-3xy^2+x^2y \quad (44)$$

$$v(x,y) = -y-2xy+y^2-3x^2y+y^3-xy^2 \quad (45)$$

$$p(x,y) = xy+x+y+x^3y^2-4/3 \quad (46)$$

Note that the pressure obeys $\int_\Omega p\, d\Omega = 0$

$$b_x = -(1+y-3x^2y^2) \quad (47)$$

$$b_y = -(1-3x-2x^3y) \quad (48)$$

6.8 Assigning values to quadrature points

As we have seen in Section **??**, the building of the elemental matrix and rhs requires (at least) to assign a density and viscosity value to each quadrature point inside the element. Depending on the type of modelling, this task can prove more complex than one might expect and have large consequences on the solution accuracy.

Here are several options:

- The simplest way (which is often used for benchmarks) consists in computing the 'real' coordinates (x_q, y_q, z_q) of a given quadrature point based on its reduced coordinates (r_q, s_q, t_q), and passing these coordinates to a function which returns density and/or viscosity at this location. For instance, for the Stokes sphere:

```
def rho(x,y):
    if (x-.5)**2+(y-0.5)**2<0.123**2:
       val=2.
    else:
       val=1.
    return val

def mu(x,y):
    if (x-.5)**2+(y-0.5)**2<0.123**2:
       val=1.e2
    else:
       val=1.
    return val
```

  This is very simple, but it has been shown to potentially be problematic. In essence, it can introduce very large contrasts inside a single element and perturb the quadrature. Please read section 3.3 of [**?**] and/or have a look at the section titled "Averaging material properties" in the ASPECT manual.

- another similar approach consists in assigning a density and viscosity value to the nodes of the FE mesh first, and then using these nodal values to assign values to the quadrature points. Very often ,and quite logically, the shape functions are used to this effect. Indeed we have seen before that for any point (r, s, t) inside an element we have

$$f_h(r, s, t) = \sum_i^m f_i N_i(r, s, t)$$

  where the f_i are the nodal values and the N_i the corresponding basis functions.

  In the case of linear elements (Q_1 basis functions), this is straightforward. In fact, the basis functions N_i can be seen as moving weights: the closer the point is to a node, the higher the weight (basis function value).

  However, this is quite another story for quadratic elements (Q_2 basis functions). In order to illustrate the problem, let us consider a 1D problem. The basis functions are

$$N_1(r) = \frac{1}{2}r(r-1) \qquad N_2(r) = 1 - r^2 \qquad N_3(r) = \frac{1}{2}r(r+1)$$

  Let us further assign: $\rho_1 = \rho_2 = 0$ and $\rho_3 = 1$. Then

$$\rho_h(r) = \sum_i^m \rho_i N_i(r) = N_3(r)$$

  There lies the core of the problem: the $N_3(r)$ basis function is negative for $r \in [-1, 0]$. This means that the quadrature point in this interval will be assigned a negative density, which is nonsensical and numerically problematic!

  use 2X Q1. write about it !

The above methods work fine as long as the domain contains a single material. As soon as there are multiple fluids in the domain a special technique is needed to track either the fluids themselves or their interfaces. Let us start with markers. We are then confronted to the infernal trio (a *menage a trois*?) which is present for each element, composed of its nodes, its markers and its quadrature points.

Each marker carries the material information (density and viscosity). This information must ultimately be projected onto the quadrature points. Two main options are possible: an algorithm is designed and projects the marker-based fields onto the quadrature points directly or the marker fields are first projected onto the FE nodes and then onto the quadrature points using the techniques above.

At a given time, every element e contains n^e markers. During the FE matrix building process, viscosity and density values are needed at the quadrature points. One therefore needs to project the values carried by the markers at these locations. Several approaches are currently in use in the community and the topic has been investigated by [**?**] and [**?**] for instance.

ELEFANT adopts a simple approach: viscosity and density are considered to be elemental values, i.e. all the markers within a given element contribute to assign a unique constant density and viscosity value to the element by means of an averaging scheme.

While it is common in the literature to treat the so-called arithmetic, geometric and harmonic means as separate averagings, I hereby wish to introduce the notion of generalised mean, which is a family of functions for aggregating sets of numbers that include as special cases the arithmetic, geometric and harmonic means.

If p is a non-zero real number, we can define the generalised mean (or power mean) with exponent p of the positive real numbers a_1, ... a_n as:

$$M_p(a_1, ...a_n) = \left(\frac{1}{n} \sum_{i=1}^{n} a_i^p \right)^{1/p} \tag{49}$$

and it is trivial to verify that we then have the special cases:

$$M_{-\infty} = \lim_{p \to -\infty} M_p = \min(a_1, ...a_n) \quad \text{(minimum)} \tag{50}$$

$$M_{-1} = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n}} \qquad \text{(harm. avrg.)} \tag{51}$$

$$M_0 = \lim_{p \to 0} M_p = \left(\prod_{i=1}^{n} a_i \right)^{1/n} \qquad \text{(geom. avrg.)} \tag{52}$$

$$M_{+1} = \frac{1}{n} \sum_{i=1}^{n} a_i \qquad \text{(arithm. avrg.)} \tag{53}$$

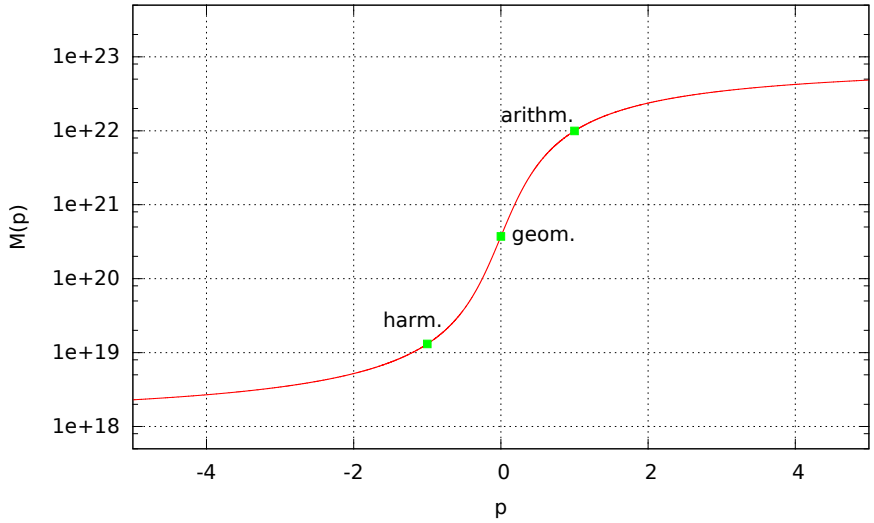
$$M_{+2} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} a_i^2} \qquad \text{(root mean square)} \tag{54}$$

$$M_{+\infty} = \lim_{p \to +\infty} M_p = \max(a_1, ...a_n) \quad \text{(maximum)} \tag{55}$$

Note that the proofs of the limit convergence are given in [**?**].

An interesting property of the generalised mean is as follows: for two real values p and q, if $p < q$ then $M_p \leq M_q$. This property has for instance been illustrated in Fig. 20 of [**?**].

One can then for instance look at the generalised mean of a randomly generated set of 1000 viscosity values within $10^{18} Pa.s$ and $10^{23} Pa.s$ for $-5 \leq p \leq 5$. Results are shown in the figure hereunder and the arithmetic, geometric and harmonic values are indicated too. The function M_p assumes an arctangent-like shape: very low values of p will ultimately yield the minimum viscosity in the array while very high values will yield its maximum. In between, the transition is smooth and occurs essentially for $|p| \leq 5$.



▷ *python_codes/fieldstone_markers_avrg*

6.9 Matrix (Sparse) storage

The FE matrix is the result of the assembly process of all elemental matrices. Its size can become quite large when the resolution is being increased (from thousands of lines/columns to tens of millions).

One important property of the matrix is its sparsity. Typically less than 1% of the matrix terms is not zero and this means that the matrix storage can and should be optimised. Clever storage formats were designed early on since the amount of RAM memory in computers was the limiting factor 3 or 4 decades ago. [**?**]

There are several standard formats:

- compressed sparse row (CSR) format
- compressed sparse column format (CSC)
- the Coordinate Format (COO)
- Skyline Storage Format
- ...

I focus on the CSR format in what follows.

6.9.1 2D domain - One degree of freedom per node

Let us consider again the 3×2 element grid which counts 12 nodes.

```
8=======9======10======11
|       |       |       |
|  (3)  |  (4)  |  (5)  |
|       |       |       |
4=======5=======6=======7
|       |       |       |
|  (0)  |  (1)  |  (2)  |
|       |       |       |
0=======1=======2=======3
```

In the case there is only a single degree of freedom per node, the assembled FEM matrix will look like this:

$$\begin{pmatrix}
X & X & & & X & X & & & & & & \\
X & X & X & & X & X & X & & & & & \\
 & X & X & X & & X & X & X & & & & \\
 & & X & X & & & X & X & & & & \\
X & X & & & X & X & & & X & X & & \\
X & X & X & & X & X & X & & X & X & X & \\
 & X & X & X & & X & X & X & & X & X & X \\
 & & X & X & & & X & X & & & X & X \\
 & & & & X & X & & & X & X & & \\
 & & & & X & X & X & & X & X & X & \\
 & & & & & X & X & X & & X & X & X \\
 & & & & & & X & X & & & X & X
\end{pmatrix}$$

where the X stand for non-zero terms. This matrix structure stems from the fact that

- node 0 sees nodes 0,1,4,5
- node 1 sees nodes 0,1,2,4,5,6
- node 2 sees nodes 1,2,3,5,6,7
- ...

- node 5 sees nodes 0,1,2,4,5,6,8,9,10
- ...
- node 10 sees nodes 5,6,7,9,10,11
- node 11 sees nodes 6,7,10,11

In light thereof, we have

- 4 corner nodes which have 4 neighbours (counting themselves)
- 2(nnx-2) nodes which have 6 neighbours
- 2(nny-2) nodes which have 6 neighbours
- (nnx-2)×(nny-2) nodes which have 9 neighbours

In total, the number of non-zero terms in the matrix is then:

$$NZ = 4 \times 4 + 4 \times 6 + 2 \times 6 + 2 \times 9 = 70$$

In general, we would then have:

$$NZ = 4 \times 4 + [2(nnx - 2) + 2(nny - 2)] \times 6 + (nnx - 2)(nny - 2) \times 9$$

Let us temporarily assume $nnx = nny = n$. Then the matrix size (total number of unknowns) is $N = n^2$ and

$$NZ = 16 + 24(n - 2) + 9(n - 2)^2$$

A full matrix array would contain $N^2 = n^4$ terms. The ratio of NZ (the actual number of reals to store) to the full matrix size (the number of reals a full matrix contains) is then

$$R = \frac{16 + 24(n - 2) + 9(n - 2)^2}{n^4}$$

It is then obvious that when n is large enough $R \sim 1/n^2$.

CSR stores the nonzeros of the matrix row by row, in a single indexed array A of double precision numbers. Another array COLIND contains the column index of each corresponding entry in the A array. A third integer array RWPTR contains pointers to the beginning of each row, which an additional pointer to the first index following the nonzeros of the matrix A. A and COLIND have length NZ and RWPTR has length N+1.

In the case of the here-above matrix, the arrays COLIND and RWPTR will look like:

$$COLIND = (0, 1, 4, 5,\ 0, 1, 2, 4, 5, 6,\ 1, 2, 3, 5, 6, 7, ..., 6, 7, 10, 11)$$

$$RWPTR = (0, 4, 10, 16, ...)$$

6.9.2 2D domain - Two degrees of freedom per node

When there are now two degrees of freedom per node, such as in the case of the Stokes equation in two-dimensions, the size of the $\mathbb{K}$ matrix is given by

```
NfemV=nnp*ndofV
```

In the case of the small grid above, we have `NfemV=24`. Elemental matrices are now 8×8 in size.

We still have

- 4 corner nodes which have 4 ,neighbours,
- 2(nnx-2) nodes which have 6 neighbours
- 2(nny-2) nodes which have 6 neighbours

- (nnx-2)x(nny-2) nodes which have 9 neighbours,

but now each degree of freedom from a node sees the other two degrees of freedom of another node too. In that case, the number of nonzeros has been multiplied by four and the assembled FEM matrix looks like:

$$\begin{pmatrix}
X & X & X & X & & & & & X & X & X & X & & & & & & & & & & & & \\
X & X & X & X & & & & & X & X & X & X & & & & & & & & & & & & \\
X & X & X & X & X & X & & & X & X & X & X & X & X & & & & & & & & & & \\
X & X & X & X & X & X & & & X & X & X & X & X & X & & & & & & & & & & \\
 & & X & X & X & X & X & X & & & X & X & X & X & X & X & & & & & & & & \\
 & & X & X & X & X & X & X & & & X & X & X & X & X & X & & & & & & & & \\
 & & & & X & X & X & X & & & & & X & X & X & X & & & & & & & & \\
 & & & & X & X & X & X & & & & & X & X & X & X & & & & & & & & \\
X & X & X & X & & & & & X & X & X & X & & & & & X & X & X & X & & & & \\
X & X & X & X & & & & & X & X & X & X & & & & & X & X & X & X & & & & \\
X & X & X & X & X & X & & & X & X & X & X & X & X & & & X & X & X & X & X & X & & \\
X & X & X & X & X & X & & & X & X & X & X & X & X & & & X & X & X & X & X & X & & \\
 & & X & X & X & X & X & X & & & X & X & X & X & X & X & & & X & X & X & X & X & X \\
 & & X & X & X & X & X & X & & & X & X & X & X & X & X & & & X & X & X & X & X & X \\
 & & & & X & X & X & X & & & & & X & X & X & X & & & & & X & X & X & X \\
 & & & & X & X & X & X & & & & & X & X & X & X & & & & & X & X & X & X \\
 & & & & & & & & X & X & X & X & & & & & X & X & X & X & & & & \\
 & & & & & & & & X & X & X & X & & & & & X & X & X & X & & & & \\
 & & & & & & & & X & X & X & X & X & X & & & X & X & X & X & X & X & & \\
 & & & & & & & & X & X & X & X & X & X & & & X & X & X & X & X & X & & \\
 & & & & & & & & & & X & X & X & X & X & X & & & X & X & X & X & X & X \\
 & & & & & & & & & & X & X & X & X & X & X & & & X & X & X & X & X & X \\
 & & & & & & & & & & & & X & X & X & X & & & & & X & X & X & X \\
 & & & & & & & & & & & & X & X & X & X & & & & & X & X & X & X
\end{pmatrix}$$

Note that the degrees of freedom are organised as follows:

$$(u_0, v_0, u_1, v_1, u_2, v_2, ...u_{11}, v_{11})$$

In general, we would then have:

$$NZ = 4\left[4 \times 4 + [2(nnx-2) + 2(nny-2)] \times 6 + (nnx-2)(nny-2) \times 9\right]$$

and in the case of the small grid, the number of non-zero terms in the matrix is then:

$$NZ = 4\left[4 \times 4 + 4 \times 6 + 2 \times 6 + 2 \times 9\right] = 280$$

In the case of the here-above matrix, the arrays COLIND and RWPTR will look like:

$$COLIND = (0, 1, 2, 3, 8, 9, 10, 11,\ 0, 1, 2, 3, 8, 9, 10, 11,\ ...)$$

$$RWPTR = (0, 8, 16, 28, ...)$$

6.9.3 in fieldstone

The majority of the codes have the FE matrix being a full array

```
a_mat = np.zeros((Nfem,Nfem),dtype=np.float64)
```

and it is converted to CSR format on the fly in the solve phase:

```
sol = sps.linalg.spsolve(sps.csr_matrix(a_mat),rhs)
```

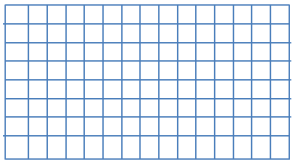
Note that linked list storages can be used (lil_matrix). Substantial memory savings but much longer compute times.

6.10 Mesh generation

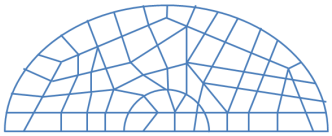
Before basis functions can be defined and PDEs can be discretised and solved we must first tesselate the domain with polygons, e.g. triangles and quadrilaterals in 2D, tetrahedra, prisms and hexahedra in 3D.

When the domain is itself simple (e.g. a rectangle, a sphere, ...) the mesh (or grid) can be (more or less) easily produced and the connectivity array filled with straightforward algorithms [**?**]. However, real life applications can involve etremely complex geometries (e.g. a bridge, a human spine, a car chassis and body, etc ...) and dedicated algorithms/softwares must be used (see [**?**, **?**, **?**]).

We usually distinguish between two broad classes of grids: structured grids (with a regular connectivity) and unstructured grids (with an irregular connectivity).

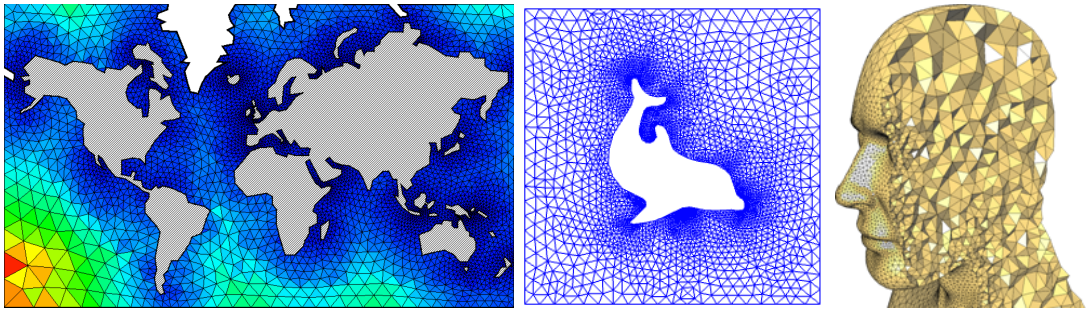


Structured Grid



Unstructured Grid

On the following figure are shown various triangle- tetrahedron-based meshes. These are obviously better suited for simulations of complex geometries:



add more examples coming from geo

Let us now focus on the case of a rectangular computational domain of size `Lx` × `Ly` with a regular mesh composed of `nelx`×`nely`=`nel` quadrilaterals. There are then `nnx`×`nny`=`nnp` grid points. The elements are of size `hx`×`hy` with `hx`=`Lx`/`nelx`.

We have no reason to come up with an irregular/ilogical node numbering so we can number nodes row by row or column by column as shown on the example hereunder of a 3×2 grid:

```
8=======9======10======11       2=======5=======8======11
|       |       |       |       |       |       |       |
|  (3)  |  (4)  |  (5)  |       |  (1)  |  (3)  |  (5)  |
|       |       |       |       |       |       |       |
4=======5=======6=======7       1=======4=======7======10
|       |       |       |       |       |       |       |
|  (0)  |  (1)  |  (2)  |       |  (0)  |  (2)  |  (4)  |
|       |       |       |       |       |       |       |
0=======1=======2=======3       0=======3=======6=======9

     "row by row"                   "column by column"
```

The numbering of the elements themselves could be done in a somewhat chaotic way but we follow the numbering of the nodes for simplicity. The row by row option is the adopted one in **fieldstone**and the coordinates of the points are computed as follows:

```
x = np.empty(nnp, dtype=np.float64)
y = np.empty(nnp, dtype=np.float64)
```

```
counter = 0
for j in range(0,nny):
    for i in range(0,nnx):
        x[counter]=i*hx
        y[counter]=j*hy
        counter += 1
```

The inner loop has `i` ranging from `0` to `nnx-1` first for `j`=0, 1, ... up to `nny-1` which indeed corresponds to the row by row numbering.

We now turn to the connectivity. As mentioned before, this is a structured mesh so that the so-called connectivity array, named `icon` in our case, can be filled easily. For each element we need to store the node identities of its vertices. Since there are `nel` elements and `m=4` corners, this is a `m`×`nel` array. The algorithm goes as follows:

```
icon =np.zeros((m,nel),dtype=np.int16)
counter = 0
for j in range(0,nely):
    for i in range(0,nelx):
        icon[0,counter] = i + j * nnx
        icon[1,counter] = i + 1 + j * nnx
        icon[2,counter] = i + 1 + (j + 1) * nnx
        icon[3,counter] = i + (j + 1) * nnx
        counter += 1
```

In the case of the 3×2 mesh, the `icon` is filled as follows:

element id→ / node id↓	0	1	2	3	4	5
0	0	1	2	4	5	6
1	1	2	3	5	6	7
2	5	6	7	9	10	11
3	4	5	6	8	9	10

It is to be understood as follows: element #4 is composed of nodes 5, 6, 10 and 9. Note that nodes are always stored in a counter clockwise manner, starting at the bottom left. This is very important since the corresponding basis functions and their derivatives will be labelled accordingly.

In three dimensions things are very similar. The mesh now counts `nelx`×`nely`×`nelz`=`nel` elements which represent a cuboid of size `Lx`×`Ly`×`Lz`. The position of the nodes is obtained as follows:

```
x = np.empty(nnp,dtype=np.float64)
y = np.empty(nnp,dtype=np.float64)
z = np.empty(nnp,dtype=np.float64)
counter=0
for i in range(0,nnx):
    for j in range(0,nny):
        for k in range(0,nnz):
            x[counter]=i*hx
            y[counter]=j*hy
            z[counter]=k*hz
            counter += 1
```

The connectivity array is now of size `m`×`nel` with `m=8`:

```
icon =np.zeros((m,nel),dtype=np.int16)
counter = 0
for i in range(0,nelx):
    for j in range(0,nely):
        for k in range(0,nelz):
            icon[0,counter]=nny*nnz*(i  )+nnz*(j  )+k
            icon[1,counter]=nny*nnz*(i+1)+nnz*(j  )+k
            icon[2,counter]=nny*nnz*(i+1)+nnz*(j+1)+k
            icon[3,counter]=nny*nnz*(i  )+nnz*(j+1)+k
            icon[4,counter]=nny*nnz*(i  )+nnz*(j  )+k+1
            icon[5,counter]=nny*nnz*(i+1)+nnz*(j  )+k+1
            icon[6,counter]=nny*nnz*(i+1)+nnz*(j+1)+k+1
            icon[7,counter]=nny*nnz*(i  )+nnz*(j+1)+k+1
            counter += 1
```

produce drawing of node numbering

6.11 Visco-Plasticity

6.11.1 Tensor invariants

Before we dive into the world of nonlinear rheologies it is necessary to introduce the concept of tensor invariants since they are needed further on. Unfortunately there are many different notations used in the literature and these can prove to be confusing.

Given a tensor $\boldsymbol{T}$, one can compute its (moment) invariants as follows:

- first invariant :

$$\begin{aligned} T_I|^{2D} &= Tr[\boldsymbol{T}] = T_{xx} + T_{yy} \\ T_I|^{3D} &= Tr[\boldsymbol{T}] = T_{xx} + T_{yy} + T_{zz} \end{aligned}$$

- second invariant :

$$\begin{aligned} T_{II}|^{2D} &= \frac{1}{2} Tr[\boldsymbol{T}^2] = \frac{1}{2} \sum_{ij} T_{ij} T_{ji} = \frac{1}{2}(T_{xx}^2 + T_{yy}^2) + T_{xy}^2 \\ T_{II}|^{3D} &= \frac{1}{2} Tr[\boldsymbol{T}^2] = \frac{1}{2} \sum_{ij} T_{ij} T_{ji} = \frac{1}{2}(T_{xx}^2 + T_{yy}^2 + T_{yy}^2) + T_{xy}^2 + T_{xz}^2 + T_{yz}^2 \end{aligned}$$

- third invariant :

$$T_{III} = \frac{1}{3} Tr[\boldsymbol{T}^3] = \frac{1}{3} \sum_{ijk} T_{ij} T_{jk} T_{ki}$$

The implementation of the plasticity criterions relies essentially on the second invariants of the (deviatoric) stress $\boldsymbol{\tau}$ and the (deviatoric) strainrate tensors $\dot{\boldsymbol{\varepsilon}}$:

$$\begin{aligned} \tau_{II}|^{2D} &= \frac{1}{2}(\tau_{xx}^2 + \tau_{yy}^2) + \tau_{xy}^2 \\ &= \frac{1}{4}(\sigma_{xx} - \sigma_{yy})^2 + \sigma_{xy}^2 \\ &= \frac{1}{4}(\sigma_1 - \sigma_2)^2 \end{aligned}$$

$$\begin{aligned} \tau_{II}|^{3D} &= \frac{1}{2}(\tau_{xx}^2 + \tau_{yy}^2 + \tau_{zz}^2) + \tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2 \\ &= \frac{1}{6}\left[(\sigma_{xx} - \sigma_{yy})^2 + (\sigma_{yy} - \sigma_{zz})^2 + (\sigma_{xx} - \sigma_{zz})^2\right] + \sigma_{xy}^2 + \sigma_{xz}^2 + \sigma_{yz}^2 \\ &= \frac{1}{6}\left[(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_1 - \sigma_3)^2\right] \end{aligned}$$

$$\begin{aligned} \varepsilon_{II}|^{2D} &= \frac{1}{2}\left[(\dot{\varepsilon}_{xx}^d)^2 + (\dot{\varepsilon}_{yy}^d)^2\right] + (\dot{\varepsilon}_{xy}^d)^2 \\ &= \frac{1}{2}\left[\frac{1}{4}(\dot{\varepsilon}_{xx} - \dot{\varepsilon}_{yy})^2 + \frac{1}{4}(\dot{\varepsilon}_{yy} - \dot{\varepsilon}_{xx})^2\right] + \dot{\varepsilon}_{xy}^2 \\ &= \frac{1}{4}(\dot{\varepsilon}_{xx} - \dot{\varepsilon}_{yy})^2 + \dot{\varepsilon}_{xy}^2 \end{aligned}$$

$$\begin{aligned} \varepsilon_{II}|^{3D} &= \frac{1}{2}\left[(\dot{\varepsilon}_{xx}^d)^2 + (\dot{\varepsilon}_{yy}^d)^2 + (\dot{\varepsilon}_{zz}^d)^2\right] + (\dot{\varepsilon}_{xy}^d)^2 + (\dot{\varepsilon}_{xz}^d)^2 + (\dot{\varepsilon}_{yz}^d)^2 \\ &= \frac{1}{6}\left[(\dot{\epsilon}_{xx} - \dot{\epsilon}_{yy})^2 + (\dot{\epsilon}_{yy} - \dot{\epsilon}_{zz})^2 + (\dot{\epsilon}_{xx} - \dot{\epsilon}_{zz})^2\right] + \dot{\epsilon}_{xy}^2 + \dot{\epsilon}_{xz}^2 + \dot{\epsilon}_{yz}^2 \end{aligned}$$

Note that these (second) invariants are almost always used under a square root so we define:

$$\underline{\tau}_{II} = \sqrt{\tau_{II}} \qquad \underline{\dot{\varepsilon}}_{II} = \sqrt{\dot{\varepsilon}_{II}}$$

Note that these quantities have the same dimensions as their tensor counterparts, i.e. Pa for stresses and s^{-1} for strain rates.

6.11.2 Scalar viscoplasticity

This formulation is quite easy to implement. It is widely used, e.g. [**?**, **?**, **?**], and relies on the assumption that a scalar quantity η_p (the 'effective plastic viscosity') exists such that the deviatoric stress tensor

$$\boldsymbol{\tau} = 2\eta_p \dot{\boldsymbol{\varepsilon}} \tag{56}$$

is bounded by some yield stress value Y. From Eq. (56) it follows that $\underline{\tau}_{II} = 2\eta_p \underline{\dot{\varepsilon}}_{II} = Y$ which yields

$$\eta_p = \frac{Y}{2\underline{\dot{\varepsilon}}_{II}}$$

This approach has also been coined the Viscosity Rescaling Method (VRM) [**?**].

insert here the rederivation 2.1.1 of spmw16

It is at this stage important to realise that (i) in areas where the strainrate is low, the resulting effective viscosity will be large, and (ii) in areas where the strainrate is high, the resulting effective viscosity will be low. This is not without consequences since (effective) viscosity contrasts up to 8-10 orders of magnitude have been observed/obtained with this formulation and it makes the FE matrix very stiff, leading to (iterative) solver convergence issues. In order to contain these viscosity contrasts one usually resorts to viscosity limiters η_{min} and η_{max} such that

$$\eta_{min} \leq \eta_p \leq \eta_{max}$$

Caution must be taken when choosing both values as they may influence the final results.

▷ *python_codes/fieldstone_indentor*

6.11.3 about the yield stress value Y

In geodynamics the yield stress value is often given as a simple function. It can be constant (in space and time) and in this case we are dealing with a von Mises plasticity yield criterion. . We simply assume $Y_{vM} = C$ where C is a constant cohesion independent of pressure, strainrate, deformation history, etc ...

Another model is often used: the Drucker-Prager plasticity model. A friction angle ϕ is then introduced and the yield value Y takes the form

$$Y_{DP} = p \sin\phi + C \cos\phi$$

and therefore depends on the pressure p. Because ϕ is with the range $[0°, 45°]$, Y is found to increase with depth (since the lithostatic pressure often dominates the overpressure).

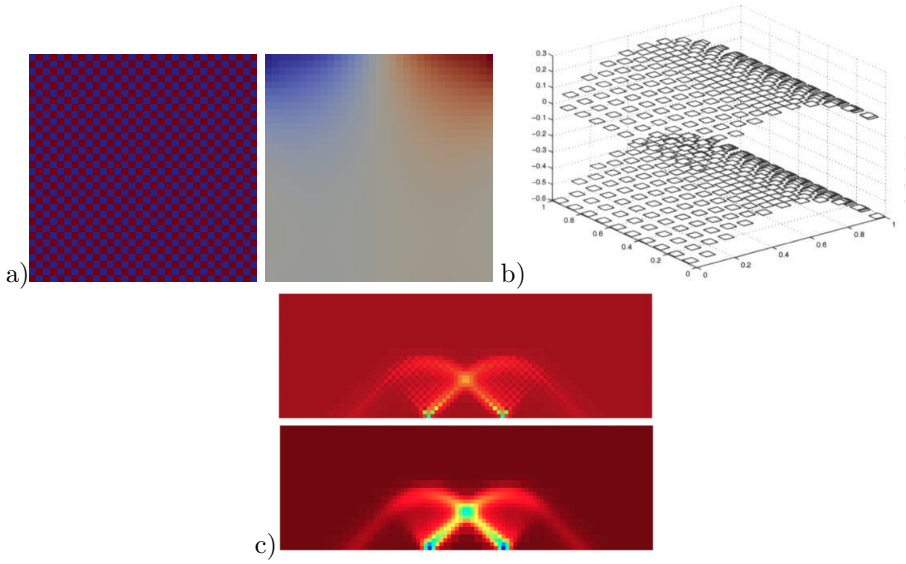
Note that a slightly modified verion of this plasticity model has been used: the total pressure p is then replaced by the lithostatic pressure p_{lith}.

6.12 Pressure smoothing

It has been widely documented that the use of the $Q_1 \times P_0$ element is not without problems. Aside from the consequences it has on the FE matrix properties, we will here focus on another unavoidable side effect: the spurious pressure checkerboard modes.

These modes have been thoroughly analysed [**?**, **?**, **?**, **?**]. They can be filtered out [**?**] or simply smoothed [**?**].

On the following figure (a,b), pressure fields for the lid driven cavity experiment are presented for both an even and un-even number of elements. We see that the amplitude of the modes can sometimes be so large that the 'real' pressure is not visible and that something as simple as the number of elements in the domain can trigger those or not at all.



a) element pressure for a 32x32 grid and for a 33x33 grid;
b) image from [**?**, p307] for a manufactured solution;
c) elemental pressure and smoothed pressure for the punch experiment [**?**]

The easiest post-processing step that can be used (especially when a regular grid is used) is explained in [**?**]: "The element-to-node interpolation is performed by averaging the elemental values from elements common to each node; the node-to-element interpolation is performed by averaging the nodal values element-by-element. This method is not only very efficient but produces a smoothing of the pressure that is adapted to the local density of the octree. Note that these two steps can be repeated until a satisfying level of smoothness (and diffusion) of the pres- sure field is attained."

In the codes which rely on the $Q_1 \times P_0$ element, the (elemental) pressure is simply defined as

```
p=np.zeros(nel,dtype=np.float64)
```

while the nodal pressure is then defined as

```
q=np.zeros(nnp,dtype=np.float64)
```

The element-to-node algorithm is then simply (in 2D):

```
count=np.zeros(nnp,dtype=np.int16)
for iel in range(0,nel):
    q[icon[0,iel]]+=p[iel]
    q[icon[1,iel]]+=p[iel]
    q[icon[2,iel]]+=p[iel]
    q[icon[3,iel]]+=p[iel]
    count[icon[0,iel]]+=1
    count[icon[1,iel]]+=1
    count[icon[2,iel]]+=1
    count[icon[3,iel]]+=1
q=q/count
```

Pressure smoothing is further discussed in [**?**].

produce figure to explain this

link to proto paper

link to least square and nodal derivatives

6.13 Pressure scaling

As perfectly explained in the step 32 of deal.ii[2], we often need to scale the $\mathbb{G}$ term since it is many orders of magnitude smaller than $\mathbb{K}$, which introduces large inaccuracies in the solving process to the point that the solution is nonsensical. This scaling coefficient is η/L. We start from

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P} \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix}$$

and introduce the scaling coefficient as follows (which in fact does not alter the solution at all):

$$\begin{pmatrix} \mathbb{K} & \frac{\eta}{L}\mathbb{G} \\ \frac{\eta}{L}\mathbb{G}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \frac{L}{\eta}\mathcal{P} \end{pmatrix} = \begin{pmatrix} f \\ \frac{\eta}{L}h \end{pmatrix}$$

We then end up with the modified Stokes system:

$$\begin{pmatrix} \mathbb{K} & \underline{\mathbb{G}} \\ \underline{\mathbb{G}}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \underline{\mathcal{P}} \end{pmatrix} = \begin{pmatrix} f \\ \underline{h} \end{pmatrix}$$

where

$$\underline{\mathbb{G}} = \frac{\eta}{L}\mathbb{G} \qquad \underline{\mathcal{P}} = \frac{L}{\eta}\mathcal{P} \qquad \underline{h} = \frac{\eta}{L}h$$

After the solve phase, we recover the real pressure with $\mathcal{P} = \frac{\eta}{L}\mathcal{P}'$.

[2] https://www.dealii.org/9.0.0/doxygen/deal.II/step_32.html

6.14 Pressure normalisation

When Dirichlet boundary conditions are imposed everywhere on the boundary, pressure is only present by its gradient in the equations. It is thus determined up to an arbitrary constant. In such a case, one commonly impose the average of the pressure over the whole domain or on a subsect of the boundary to be have a zero average, i.e.

$$\int_\Omega p dV = 0$$

Another possibility is to impose the pressure value at a single node.

Let us assume that we are using $Q_1 \times P_0$ elements. Then the pressure is constant inside each element. The integral above becomes:

$$\int_\Omega p dV = \sum_e \int_{\Omega_e} p dV = \sum_e p_e \int_{\Omega_e} dV = \sum_e p_e A_e =$$

where the sum runs over all elements e of area A_e. This can be rewritten

$$\mathbb{L}^T \mathcal{P} = 0$$

and it is a constraint on the pressure. As we have seen before **??**, we can associate to it a Lagrange multiplier λ so that we must solve the modified Stokes system:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} & 0 \\ \mathbb{G}^T & 0 & \mathbb{L} \\ 0 & \mathbb{L}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P} \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ h \\ 0 \end{pmatrix}$$

When higher order spaces are used for pressure (continuous or discontinuous) one must then carry out the above integration numerically by means of (usually) a Gauss-Legendre quadrature.

- ▷ *python_codes/fieldstone_stokes_sphere_3D_saddlepoint/*
- ▷ *python_codes/fieldstone_burstedde/*
- ▷ *python_codes/fieldstone_busse/*
- ▷ *python_codes/fieldstone_RTinstability1/*
- ▷ *python_codes/fieldstone_saddlepoint_q2q1/*
- ▷ *python_codes/fieldstone_compressible2/*
- ▷ *python_codes/fieldstone_compressible1/*
- ▷ *python_codes/fieldstone_saddlepoint_q3q2/*

6.15 The choice of solvers

Let us first look at the penalty formulation. In this case we are only solving for velocity since pressure is recovered in a post-processing step. We also know that the penalty factor is many orders of magnitude higher than the viscosity and in combination with the use of the $Q_1 \times P_0$ element the resulting matrix condition number is very high so that the use of iterative solvers in precluded. Indeed codes such as SOPALE [**?**], DOUAR [**?**], or FANTOM [**?**] relying on the penalty formulation all use direct solvers (BLKFCT, MUMPS, WSMP).

The main advantage of direct solvers is used in this case: They can solve ill-conditioned matrices. However memory requirements for the storage of number of nonzeros in the Cholesky matrix grow very fast as the number of equations/grid size increases, especially in 3D, to the point that even modern computers with tens of Gb of RAM cannot deal with a 100^3 element mesh. This explains why direct solvers are often used for 2D problems and rarely in 3D with noticeable exceptions [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**].

In light of all this, let us start again from the (full) Stokes system:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P} \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix}$$

We need to solve this system in order to obtain the solution, i.e. the $\mathcal{V}$ and $\mathcal{P}$ vectors. But how?

Unfortunately, this question is not simple to answer and the 'right' depends on many parameters. How big is the matrix ? what is the condition number of the matrix $\mathbb{K}$? Is it symmetric ? (i.e. how are compressible terms handled?).

6.15.1 The Schur complement approach

Let us write the above system as two equations:

$$\mathbb{K}\mathcal{V} + \mathbb{G}\mathcal{P} = f \quad (57)$$
$$\mathbb{G}^T\mathcal{V} = h \quad (58)$$

The first line can be re-written $\mathcal{V} = \mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P})$ and can be inserted in the second:

$$\mathbb{G}^T\mathcal{V} = \mathbb{G}^T[\mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P})] = h$$

or,

$$\mathbb{G}^T K^{-1}\mathbb{G}\mathcal{P} = \mathbb{G}^T\mathbb{K}^{-1}f - h$$

The matrix $\mathbb{S} = \mathbb{G}^T\mathbb{K}^{-1}\mathbb{G}$ is called the Schur complement. It is Symmetric (since $\mathbb{K}$ is symmetric) and Postive-Definite[3] (SPD) if $Ker(\mathbb{G}) = 0$. look in donea-huerta book for details Having solved this equation (we have obtained $\mathcal{P}$), the velocity can be recovered by solving $\mathbb{K}\mathcal{V} = f - \mathbb{G}\mathcal{P}$. For now, let us assume that we have built the $\mathbb{S}$ matrix and the right hand side $\tilde{f} = \mathbb{G}^T\mathbb{K}^{-1}f - h$. We must solve $\mathbb{S}\mathcal{P} = \tilde{f}$.

Since $\mathbb{S}$ is SPD, the Conjugate Gradient (CG) method is very appropriate to solve this system. Indeed, looking at the definition of Wikipedia: *In mathematics, the conjugate gradient method is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems.*

The Conjugate Gradient algorithm is as follows:

[3] M positive definite $\iff x^T Mx > 0 \; \forall \; x \in \mathbb{R}^n \setminus \mathbf{0}$

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
if $\mathbf{r}_0$ is sufficiently small, then return $\mathbf{x}_0$ as the result
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small, then exit loop
$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
$k := k + 1$
end repeat
return $\mathbf{x}_{k+1}$ as the result

Algorithm obtained from `https://en.wikipedia.org/wiki/Conjugate_gradient_method`

Let us look at this algorithm up close. The parts which may prove to be somewhat tricky are those involving the matrix (in our case the Schur complement). We start the iterations with a guess pressure $\mathcal{P}_0$ (and an initial guess velocity which could be obtained by solving $\mathbb{K}\mathcal{V}_0 = f - \mathbb{G}\mathcal{P}_0$).

$$\begin{aligned} r_0 &= \tilde{f} - \mathbb{S}\mathcal{P}_0 && (59)\\ &= \mathbb{G}^T\mathbb{K}^{-1}f - h - (\mathbb{G}^T\mathbb{K}^{-1}\mathbb{G})\mathcal{P}_0 && (60)\\ &= \mathbb{G}^T\mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P}_0) - h && (61)\\ &= \mathbb{G}^T\mathbb{K}^{-1}\mathbb{K}\mathcal{V}_0 - h && (62)\\ &= \mathbb{G}^T\mathcal{V}_0 - h && (63)\\ & && (64) \end{aligned}$$

We now turn to the α_k coefficient:

$$\alpha_k = \frac{r_k^T r_k}{p_k \mathbb{S} p_k} = \frac{r_k^T r_k}{p_k \mathbb{G}^T\mathbb{K}^{-1}\mathbb{G} p_k} = \frac{r_k^T r_k}{(\mathbb{G}p_k)^T\mathbb{K}^{-1}(\mathbb{G}p_k)}$$

We then define $\tilde{p}_k = \mathbb{G}p_k$, so that α_k can be computed as follows:

1. compute $\tilde{p}_k = \mathbb{G}p_k$
2. solve $\mathbb{K}d_k = \tilde{p}_k$
3. compute $\alpha_k = (r_k^T r_k)/(\tilde{p}_k^T d_k)$

Then we need to look at the term $\mathbb{S}p_k$:

$$\mathbb{S}p_k = \mathbb{G}^T\mathbb{K}^{-1}\mathbb{G}p_k = \mathbb{G}^T\mathbb{K}^{-1}\tilde{p}_k = \mathbb{G}^T d_k$$

We can then rewrite the CG algorithm as follows [**?**]:

- $r_0 = \mathbb{G}^T\mathcal{V}_0 - h$
- if r_0 is sufficiently small, then return $(\mathcal{V}_0, \mathcal{P}_0)$ as the result
- $p_0 = r_0$
- $k = 0$
- repeat
  - compute $\tilde{p}_k = \mathbb{G}p_k$
  - solve $\mathbb{K}d_k = \tilde{p}_k$
  - compute $\alpha_k = (r_k^T r_k)/(\tilde{p}_k^T d_k)$

- $\mathcal{P}_{k+1} = \mathcal{P}_k + \alpha_k p_k$
- $r_{k+1} = r_k - \alpha_k \mathbb{G}^T d_k$
- if r_{k+1} is sufficiently small, then exit loop
- $\beta_k = (r_{k+1}^T r_{k+1})/(r_k^T r_k)$
- $p_{k+1} = r_{k+1} + \beta_k p_k$
- $k = k + 1$

- return $\mathcal{P}_{k+1}$ as result

We see that we have managed to solve the Schur complement equation with the Conjugate Gradient method without ever building the matrix $\mathbb{S}$. Having obtained the pressure solution, we can easily recover the corresponding velocity with $\mathbb{K}\mathcal{V}_{k+1} = f - \mathbb{G}\mathcal{P}_{k+1}$. However, this is rather unfortunate because it requires yet another solve with the $\mathbb{K}$ matrix. As it turns out, we can slightly alter the above algorithm to have it update the velocity as well so that this last solve is unnecessary.

We have

$$\mathcal{V}_{k+1} = \mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P}_{p+1}) = \mathbb{K}^{-1}(f - \mathbb{G}(\mathcal{P}_k + \alpha_k p_k)) = \mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P}_k) - \alpha_k \mathbb{K}^{-1}\mathbb{G}p_k = \mathcal{V} - \alpha_k \mathbb{K}^{-1}\tilde{p}_k = \mathcal{V}_k - \alpha_k d_k$$

and we can insert this minor extra calculation inside the algorithm and get the velocity solution nearly for free. The final CG algorithm is then

solver_cg:

- compute $\mathcal{V}_0 = \mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P}_0)$
- $r_0 = \mathbb{G}^T\mathcal{V}_0 - h$
- if r_0 is sufficiently small, then return $(\mathcal{V}_0, \mathcal{P}_0)$ as the result
- $p_0 = r_0$
- $k = 0$
- repeat
  - compute $\tilde{p}_k = \mathbb{G}p_k$
  - solve $\mathbb{K}d_k = \tilde{p}_k$
  - compute $\alpha_k = (r_k^T r_k)/(\tilde{p}_k^T d_k)$
  - $\mathcal{P}_{k+1} = \mathcal{P}_k + \alpha_k p_k$
  - $\mathcal{V}_{k+1} = \mathcal{V}_k - \alpha_k d_k$
  - $r_{k+1} = r_k - \alpha_k \mathbb{G}^T d_k$
  - if r_{k+1} is sufficiently small ($|r_{k+1}|_2/|r_0|_2 < tol$), then exit loop
  - $\beta_k = (r_{k+1}^T r_{k+1})/(r_k^T r_k)$
  - $p_{k+1} = r_{k+1} + \beta_k p_k$
  - $k = k + 1$
- return $\mathcal{P}_{k+1}$ as result

This iterative algorithm will converge to the solution with a rate which depends on the condition number of the $\mathbb{S}$ matrix, which is not easily obtainable since $\mathbb{S}$ is never built. Also, we know that large viscosity contrasts in the domain will influence this too. Finally, we notice that this algorithm requires one solve with matrix $\mathbb{K}$ per iteration but says nothing about the method employed to do so.

One thing we know improves the convergence of any iterative solver is the use of a preconditioner matrix and therefore use the Preconditioned Conjugate Gradient method:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{z}_0 := \mathbf{M}^{-1}\mathbf{r}_0$
$\mathbf{p}_0 := \mathbf{z}_0$
$k := 0$
repeat
$\alpha_k := \dfrac{\mathbf{r}_k^\mathsf{T}\mathbf{z}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small **then** exit loop **end if**
$\mathbf{z}_{k+1} := \mathbf{M}^{-1}\mathbf{r}_{k+1}$
$\beta_k := \dfrac{\mathbf{z}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{z}_k^\mathsf{T}\mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$
$k := k + 1$
end repeat
The result is $\mathbf{x}_{k+1}$

Algorithm obtained from `https://en.wikipedia.org/wiki/Conjugate_gradient_method`

In the algorithm above the preconditioner matrix M has to be symmetric positive-definite and fixed, i.e., cannot change from iteration to iteration.

We see that this algorithm introduces an additional vector z and a solve with the matrix M at each iteration, which means that M must be such that solving $Mx = f$ where f is a given rhs vector must be cheap. Ultimately, the PCG algorithm applied to the Schur complement equation takes the form:

solver_pcg:

- compute $\mathcal{V}_0 = \mathbb{K}^{-1}(f - \mathbb{G}\mathcal{P}_0)$
- $r_0 = \mathbb{G}^T\mathcal{V}_0 - h$
- if r_0 is sufficiently small, then return $(\mathcal{V}_0, \mathcal{P}_0)$ as the result
- $z_0 = M^{-1}r_0$
- $p_0 = z_0$
- $k = 0$
- repeat
  - compute $\tilde{p}_k = \mathbb{G}p_k$
  - solve $\mathbb{K}d_k = \tilde{p}_k$
  - compute $\alpha_k = (r_k^T z_k)/(\tilde{p}_k^T d_k)$
  - $\mathcal{P}_{k+1} = \mathcal{P}_k + \alpha_k p_k$
  - $\mathcal{V}_{k+1} = \mathcal{V}_k - \alpha_k d_k$
  - $r_{k+1} = r_k - \alpha_k\mathbb{G}^T d_k$
  - if r_{k+1} is sufficiently small ($|r_{k+1}|_2/|r_0|_2 < tol$), then exit loop
  - $z_{k+1} = M^{-1}r_{k+1}$
  - $\beta_k = (z_{k+1}^T r_{k+1})/(z_k^T r_k)$
  - $p_{k+1} = z_{k+1} + \beta_k p_k$
  - $k = k + 1$
- return $\mathcal{P}_{k+1}$ as result

how to compute M for the Schur complement ?

6.16 The GMRES approach

6.17 The consistent boundary flux (CBF)

The Consistent Boundary Flux technique was devised to alleviate the problem of the accuracy of primary variables derivatives (mainly velocity and temperature) on boundaries, where basis function (nodal) derivatives do not exist. These derivatives are important since they are needed to compute the heat flux (and therefore the NUsselt number) or dynamic topography and geoid.

The idea was first introduced in [**?**] and later used in geodynamics [**?**]. It was finally implemented in the CitcomS code [**?**] and more recently in the ASPECT code (dynamic topography postprocessor). Note that the CBF should be seen as a post-processor step as it does not alter the primary variables values.

The CBF method is implemented and used in **??**.

6.17.1 applied to the Stokes equation

We start from the strong form:

$$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} = \boldsymbol{b}$$

and then write the weak form:

$$\int_\Omega N \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} dV = \int_\Omega N \boldsymbol{b} dV$$

where N is any test function. We then use the two equations:

$$\boldsymbol{\nabla} \cdot (N\boldsymbol{\sigma}) = N\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \boldsymbol{\nabla} N \cdot \boldsymbol{\sigma} \qquad \text{(chain rule)}$$

$$\int_\Omega (\boldsymbol{\nabla} \cdot \boldsymbol{f})\, dV = \int_\Gamma \boldsymbol{f} \cdot \boldsymbol{n}\, dS \qquad \text{(divergence theorem)}$$

Integrating the first equation over Ω and using the second, we can write:

$$\int_\Gamma N\boldsymbol{\sigma} \cdot \boldsymbol{n}\, dS - \int_\Omega \nabla N \cdot \boldsymbol{\sigma}\, dV = \int_\Omega N \boldsymbol{b} dV$$

On Γ, the traction vector is given by $\boldsymbol{t} = \boldsymbol{\sigma} \cdot \boldsymbol{n}$:

$$\int_\Gamma N \boldsymbol{t} dS = \int_\Omega \nabla N \cdot \boldsymbol{\sigma} dV + \int_\Omega N \boldsymbol{b} dV$$

Considering the traction vector as an unknown living on the nodes on the boundary, we can expand (for Q_1 elements)

$$t_x = \sum_{i=1}^{2} t_{x|i} N_i \qquad t_y = \sum_{i=1}^{2} t_{y|i} N_i$$

on the boundary so that the left hand term yields a mass matrix M'. Finally, using our previous experience of discretising the weak form, we can write:

$$M' \cdot \mathcal{T} = -\mathbb{K}\mathcal{V} - \mathbb{G}\mathcal{P} + f$$

where $\mathcal{T}$ is the vector of assembled tractions which we want to compute and $\mathcal{V}$ and $\mathcal{T}$ are the solutions of the Stokes problem. Note that the assembly only takes place on the elements along the boundary.

Note that the assembled mass matrix is tri-diagonal can be easily solved with a Conjugate Gradient method. With a trapezoidal integration rule (i.e. Gauss-Lobatto) the matrix can even be diagonalised and the resulting matrix is simply diagonal, which results in a very cheap solve [**?**].

6.17.2 applied to the heat equation

We start from the strong form of the heat transfer equation (without the source terms for simplicity):

$$\rho c_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} T \right) = \boldsymbol{\nabla} \cdot k \boldsymbol{\nabla} T$$

The weak form then writes:

$$\int_\Omega N \rho c_p \frac{\partial T}{\partial t} dV + \rho c_p \int_\Omega N \boldsymbol{v} \cdot \boldsymbol{\nabla} T dV = \int_\Omega N \boldsymbol{\nabla} \cdot k \boldsymbol{\nabla} T dV$$

Using once again integration by parts and divergence theorem:

$$\int_\Omega N\rho c_p \frac{\partial T}{\partial t} dV + \rho c_p \int_\Omega N\boldsymbol{v} \cdot \boldsymbol{\nabla} T dV = \int_\Gamma Nk\boldsymbol{\nabla} T \cdot \boldsymbol{n} d\Gamma - \int_\Omega \boldsymbol{\nabla} N \cdot k\boldsymbol{\nabla} T dV$$

On the boundary we are interested in the heat flux $\boldsymbol{q} = -k\boldsymbol{\nabla} T$

$$\int_\Omega N\rho c_p \frac{\partial T}{\partial t} dV + \rho c_p \int_\Omega N\boldsymbol{v} \cdot \boldsymbol{\nabla} T dV = -\int_\Gamma N\boldsymbol{q} \cdot \boldsymbol{n} d\Gamma - \int_\Omega \boldsymbol{\nabla} N \cdot k\boldsymbol{\nabla} T dV$$

or,

$$\int_\Gamma N\boldsymbol{q} \cdot \boldsymbol{n} d\Gamma = -\int_\Omega N\rho c_p \frac{\partial T}{\partial t} dV - \rho c_p \int_\Omega N\boldsymbol{v} \cdot \boldsymbol{\nabla} T dV - \int_\Omega \boldsymbol{\nabla} N \cdot k\boldsymbol{\nabla} T dV$$

Considering the normal heat flux $q_n = \boldsymbol{q} \cdot \boldsymbol{n}$ as an unknown living on the nodes on the boundary,

$$q_n = \sum_{i=1}^{2} q_{n|i} N_i$$

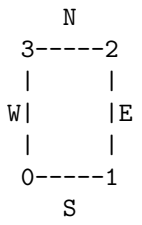
so that the left hand term becomes a mass matrix for the shape functions living on the boundary. We have already covered the right hand side terms when building the FE system to solve the heat transport equation, so that in the end

$$M' \cdot \mathcal{Q}_n = -M \cdot \frac{\partial \boldsymbol{T}}{\partial t} - K_a \cdot \boldsymbol{T} - K_d \cdot \boldsymbol{T}$$

where $\mathcal{Q}_n$ is the assembled vector of normal heat flux components.

Note that in all terms the assembly only takes place over the elements along the boundary.

What follows only applies to the reference element.



We start from

$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{0-1}} N_i \boldsymbol{t} dS + \int_{\Gamma_{1-2}} N_i \boldsymbol{t} dS + \int_{\Gamma_{2-3}} N_i \boldsymbol{t} dS + \int_{\Gamma_{3-0}} N_i \boldsymbol{t} dS \tag{65}$$

for $i = 0, 3$. Let us start with N_0, then

$$\begin{aligned}
\int_\Gamma N_0 \boldsymbol{t} dS &= \int_{\Gamma_{0-1}} N_0 \boldsymbol{t} dS + \int_{\Gamma_{1-2}} N_0 \boldsymbol{t} dS + \int_{\Gamma_{2-3}} N_0 \boldsymbol{t} dS + \int_{\Gamma_{3-0}} N_0 \boldsymbol{t} dS \\
&= \int_{\Gamma_{0-1}} N_0 (N_0^\Gamma \boldsymbol{t}_0 + N_1^\Gamma \boldsymbol{t}_1) dS \\
&+ \int_{\Gamma_{1-2}} N_0 (N_1^\Gamma \boldsymbol{t}_1 + N_2^\Gamma \boldsymbol{t}_2) dS \\
&+ \int_{\Gamma_{2-3}} N_0 (N_2^\Gamma \boldsymbol{t}_2 + N_3^\Gamma \boldsymbol{t}_3) dS \\
&+ \int_{\Gamma_{3-0}} N_0 (N_3^\Gamma \boldsymbol{t}_3 + N_0^\Gamma \boldsymbol{t}_0) dS \\
&= \left(\int_{\Gamma_{0-1}} N_0 N_0^\Gamma dS \right) \boldsymbol{t}_0 + \left(\int_{\Gamma_{0-1}} N_0 N_1^\Gamma dS \right) \boldsymbol{t}_1 \\
&+ \left(\int_{\Gamma_{1-2}} N_0 N_1^\Gamma dS \right) \boldsymbol{t}_1 + \left(\int_{\Gamma_{1-2}} N_0 N_2^\Gamma dS \right) \boldsymbol{t}_2 \\
&+ \left(\int_{\Gamma_{2-3}} N_0 N_2^\Gamma dS \right) \boldsymbol{t}_2 + \left(\int_{\Gamma_{2-3}} N_0 N_3^\Gamma dS \right) \boldsymbol{t}_3 \\
&+ \left(\int_{\Gamma_{3-0}} N_0 N_3^\Gamma dS \right) \boldsymbol{t}_3 + \left(\int_{\Gamma_{3-0}} N_0 N_0^\Gamma dS \right) \boldsymbol{t}_0
\end{aligned}$$

In what follows we will make use of

$$\int_{-1}^{+1} \frac{1}{4}(1-x)(1-x)dx = 2/3$$

$$\int_{-1}^{+1} \frac{1}{4}(1+x)(1+x)dx = 2/3$$

$$\int_{-1}^{+1} \frac{1}{4}(1+x)(1-x)dx = 1/3$$

$$
\begin{aligned}
\int_{\Gamma_{0-1}} N_0(r,s=-1)N_0^{\Gamma}(r)dS &= \int_{-1}^{+1}\frac{1}{2}(1-r)\cdot\frac{1}{2}(1-r)dr = 2/3\\
\int_{\Gamma_{0-1}} N_0(r,s=-1)N_1^{\Gamma}(r)dS &= \int_{-1}^{+1}\frac{1}{2}(1-r)\cdot\frac{1}{2}(1+r)dr = 1/3\\
\int_{\Gamma_{1-2}} N_0(r=+1,s)N_1^{\Gamma}(s)dS &= \int_{-1}^{+1}\frac{1}{4}(0)(1-s)\cdot\frac{1}{2}(1-s)ds = 0\\
\int_{\Gamma_{1-2}} N_0(r=+1,s)N_2^{\Gamma}(s)dS &= \int_{-1}^{+1}\frac{1}{4}(0)(1-s)\cdot\frac{1}{2}(1+s)ds = 0\\
\int_{\Gamma_{2-3}} N_0(r,s=+1)N_2^{\Gamma}(r)dS &= -\int_{-1}^{+1}\frac{1}{4}(1-r)(0)\cdot\frac{1}{2}(1+r)dr = 0\\
\int_{\Gamma_{2-3}} N_0(r,s=+1)N_3^{\Gamma}(r)dS &= -\int_{-1}^{+1}\frac{1}{4}(1-r)(0)\cdot\frac{1}{2}(1-r)dr = 0\\
\int_{\Gamma_{3-0}} N_0(r=-1,s)N_3^{\Gamma}(s)dS &= -\int_{-1}^{+1}\frac{1}{2}(1-s)\cdot\frac{1}{2}(1+s)ds = -1/3\\
\int_{\Gamma_{3-0}} N_0(r=-1,s)N_0^{\Gamma}(s)dS &= -\int_{-1}^{+1}\frac{1}{2}(1-s)\cdot\frac{1}{2}(1-s)ds = -2/3
\end{aligned}
$$

$$
\begin{aligned}
\int_{\Gamma_{0-1}} N_1(r,s=-1)N_0^{\Gamma}(r)dS &= \int_{-1}^{+1}\frac{1}{2}(1+r)\cdot\frac{1}{2}(1-r)dr = 1/3\\
\int_{\Gamma_{0-1}} N_1(r,s=-1)N_1^{\Gamma}(r)dS &= \int_{-1}^{+1}\frac{1}{2}(1+r)\cdot\frac{1}{2}(1+r)dr = 2/3\\
\int_{\Gamma_{1-2}} N_1(r=+1,s)N_1^{\Gamma}(s)dS &= \int_{-1}^{+1}\frac{1}{2}(1-s)\cdot\frac{1}{2}(1-s)ds = 2/3\\
\int_{\Gamma_{1-2}} N_1(r=+1,s)N_2^{\Gamma}(s)dS &= \int_{-1}^{+1}\frac{1}{2}(1-s)\cdot\frac{1}{2}(1+s)ds = 1/3\\
\int_{\Gamma_{2-3}} N_1(r,s=+1)N_2^{\Gamma}(r)dS &= -\int_{-1}^{+1}\frac{1}{4}(1+r)(0)\cdot\frac{1}{2}(1+r)dr = 0\\
\int_{\Gamma_{2-3}} N_1(r,s=+1)N_3^{\Gamma}(r)dS &= -\int_{-1}^{+1}\frac{1}{4}(1+r)(0)\cdot\frac{1}{2}(1-r)dr = 0\\
\int_{\Gamma_{3-0}} N_1(r=-1,s)N_3^{\Gamma}(s)dS &= -\int_{-1}^{+1}\frac{1}{4}(0)(1-s)\cdot\frac{1}{2}(1+s)ds = 0\\
\int_{\Gamma_{3-0}} N_1(r=-1,s)N_0^{\Gamma}(s)dS &= -\int_{-1}^{+1}\frac{1}{4}(0)(1-s)\cdot\frac{1}{2}(1-s)ds = 0
\end{aligned}
$$

$$
\begin{aligned}
\int_{\Gamma_{0-1}} N_2(r, s=-1) N_0^{\Gamma}(r) dS &= \int_{-1}^{+1} \frac{1}{4}(1+r)(0) \cdot \frac{1}{2}(1-r) dr = 0 \\
\int_{\Gamma_{0-1}} N_2(r, s=-1) N_1^{\Gamma}(r) dS &= \int_{-1}^{+1} \frac{1}{4}(1+r)(0) \cdot \frac{1}{2}(1+r) dr = 0 \\
\int_{\Gamma_{1-2}} N_2(r=+1, s) N_1^{\Gamma}(s) dS &= \int_{-1}^{+1} \frac{1}{2}(1+s) \cdot \frac{1}{2}(1-s) ds = 1/3 \\
\int_{\Gamma_{1-2}} N_2(r=+1, s) N_2^{\Gamma}(s) dS &= \int_{-1}^{+1} \frac{1}{2}(1+s) \cdot \frac{1}{2}(1+s) ds = 2/3 \\
\int_{\Gamma_{2-3}} N_2(r, s=+1) N_2^{\Gamma}(r) dS &= -\int_{-1}^{+1} \frac{1}{2}(1+r) \cdot \frac{1}{2}(1+r) dr = -2/3 \\
\int_{\Gamma_{2-3}} N_2(r, s=+1) N_3^{\Gamma}(r) dS &= -\int_{-1}^{+1} \frac{1}{2}(1+r) \cdot \frac{1}{2}(1-r) dr = -1/3 \\
\int_{\Gamma_{3-0}} N_2(r=-1, s) N_3^{\Gamma}(s) dS &= -\int_{-1}^{+1} \frac{1}{4}(0)(1+s) \cdot \frac{1}{2}(1+s) ds = 0 \\
\int_{\Gamma_{3-0}} N_2(r=-1, s) N_0^{\Gamma}(s) dS &= -\int_{-1}^{+1} \frac{1}{4}(0)(1+s) \cdot \frac{1}{2}(1-s) ds = 0
\end{aligned}
$$

$$
\begin{aligned}
\int_{\Gamma_{0-1}} N_3(r, s=-1) N_0^{\Gamma}(r) dS &= \int_{-1}^{+1} \frac{1}{4}(1-r)(0) \cdot \frac{1}{2}(1-r) dr = 0 \\
\int_{\Gamma_{0-1}} N_3(r, s=-1) N_1^{\Gamma}(r) dS &= \int_{-1}^{+1} \frac{1}{4}(1-r)(0) \cdot \frac{1}{2}(1+r) dr = 0 \\
\int_{\Gamma_{1-2}} N_3(r=+1, s) N_1^{\Gamma}(s) dS &= \int_{-1}^{+1} \frac{1}{4}(0)(1+s) \cdot \frac{1}{2}(1-s) ds = 0 \\
\int_{\Gamma_{1-2}} N_3(r=+1, s) N_2^{\Gamma}(s) dS &= \int_{-1}^{+1} \frac{1}{4}(0)(1+s) \cdot \frac{1}{2}(1+s) ds = 0 \\
\int_{\Gamma_{2-3}} N_3(r, s=+1) N_2^{\Gamma}(r) dS &= -\int_{-1}^{+1} \frac{1}{2}(1-r) \cdot \frac{1}{2}(1+r) dr = -1/3 \\
\int_{\Gamma_{2-3}} N_3(r, s=+1) N_3^{\Gamma}(r) dS &= -\int_{-1}^{+1} \frac{1}{2}(1-r) \cdot \frac{1}{2}(1-r) dr = -2/3 \\
\int_{\Gamma_{3-0}} N_3(r=-1, s) N_3^{\Gamma}(s) dS &= -\int_{-1}^{+1} \frac{1}{2}(1+s) \cdot \frac{1}{2}(1+s) ds = -2/3 \\
\int_{\Gamma_{3-0}} N_3(r=-1, s) N_0^{\Gamma}(s) dS &= -\int_{-1}^{+1} \frac{1}{2}(1+s) \cdot \frac{1}{2}(1-s) ds = -1/3
\end{aligned}
$$

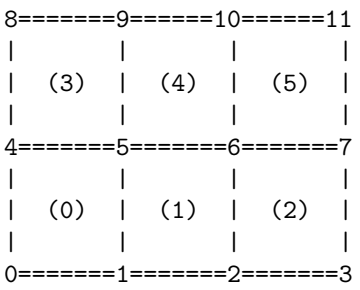
so finally

$$\begin{aligned}
\int_\Gamma N_0 \boldsymbol{t} dS &= \frac{1}{3}\boldsymbol{t}_1 - \frac{1}{3}\boldsymbol{t}_3 \\
\int_\Gamma N_1 \boldsymbol{t} dS &= \frac{1}{3}\boldsymbol{t}_0 + \frac{4}{3}\boldsymbol{t}_1 + \frac{1}{3}\boldsymbol{t}_2 \\
\int_\Gamma N_2 \boldsymbol{t} dS &= \frac{1}{3}\boldsymbol{t}_1 - \frac{1}{3}\boldsymbol{t}_3 \\
\int_\Gamma N_3 \boldsymbol{t} dS &= -\frac{1}{3}\boldsymbol{t}_0 - \frac{1}{3}\boldsymbol{t}_2 - \frac{4}{3}\boldsymbol{t}_3
\end{aligned}$$

$$\frac{1}{3}\begin{pmatrix}
. & . & 1 & . & . & . & -1 & . \\
. & . & . & 1 & . & . & . & -1 \\
1 & . & 4 & . & 1 & . & . & . \\
. & 1 & . & 4 & . & 1 & . & . \\
. & . & 1 & . & . & . & -1 & . \\
. & . & . & 1 & . & . & . & -1 \\
-1 & . & . & . & -1 & . & -4 & . \\
. & -1 & . & . & . & -1 & . & -4
\end{pmatrix} \cdot \begin{pmatrix}
t_{x,0} \\ t_{y,0} \\ t_{x,1} \\ t_{y,1} \\ t_{x,2} \\ t_{y,2} \\ t_{x,3} \\ t_{y,3}
\end{pmatrix} = \begin{pmatrix} \\ rhs \\ \\ \end{pmatrix}$$

Note that the resulting matrix is symmetric.

Let us start with a small example, a 3x2 element FE grid:



$$\text{Element 0:} \quad \int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{0-1}} N_i \boldsymbol{t} dS + \int_{\Gamma_{1-5}} N_i \boldsymbol{t} dS + \int_{\Gamma_{5-4}} N_i \boldsymbol{t} dS + \int_{\Gamma_{4-0}} N_i \boldsymbol{t} dS \tag{66}$$

$$\text{Element 1:} \quad \int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{1-2}} N_i \boldsymbol{t} dS + \int_{\Gamma_{2-6}} N_i \boldsymbol{t} dS + \int_{\Gamma_{6-5}} N_i \boldsymbol{t} dS + \int_{\Gamma_{5-1}} N_i \boldsymbol{t} dS \tag{67}$$

$$\text{Element 2:} \quad \int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{2-3}} N_i \boldsymbol{t} dS + \int_{\Gamma_{3-7}} N_i \boldsymbol{t} dS + \int_{\Gamma_{7-6}} N_i \boldsymbol{t} dS + \int_{\Gamma_{6-2}} N_i \boldsymbol{t} dS \tag{68}$$

$$\text{Element 3:} \quad \int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{4-5}} N_i \boldsymbol{t} dS + \int_{\Gamma_{5-9}} N_i \boldsymbol{t} dS + \int_{\Gamma_{9-8}} N_i \boldsymbol{t} dS + \int_{\Gamma_{8-4}} N_i \boldsymbol{t} dS \tag{69}$$

$$\text{Element 4:} \quad \int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{5-6}} N_i \boldsymbol{t} dS + \int_{\Gamma_{6-10}} N_i \boldsymbol{t} dS + \int_{\Gamma_{10-9}} N_i \boldsymbol{t} dS + \int_{\Gamma_{9-5}} N_i \boldsymbol{t} dS \tag{70}$$

$$\text{Element 5:} \quad \int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{6-7}} N_i \boldsymbol{t} dS + \int_{\Gamma_{7-11}} N_i \boldsymbol{t} dS + \int_{\Gamma_{11-10}} N_i \boldsymbol{t} dS + \int_{\Gamma_{10-6}} N_i \boldsymbol{t} dS \tag{71}$$

We see that the integral $\int_{\Gamma_{1-5}} N_i \boldsymbol{t} dS$ of element 0 is exactly the opposite[4] of the the integral $\int_{\Gamma_{5-1}} N_i \boldsymbol{t} dS$ of element 1, so that their contributions to the assembled matrix would actually cancel out. Likewise, any edge common to two elements will see in the expressions above two integrals of opposite sign, shich ultimately will not contribute to the assembled matrix.

Let us then remove the integrals over edges 1-5, 2-6, 4-5, 5-6, 6-7, 5-9 and 6-10 off the equations above:

[4] these are line integrals, one is going from node 1 to 5, the other from 5 to 1

Element 0:
$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{0-1}} N_i \boldsymbol{t} dS + \int_{\Gamma_{4-0}} N_i \boldsymbol{t} dS \tag{72}$$
$$= \int_{\Gamma_{0-1}} N_i (N_0 \boldsymbol{t}_0 + N_1 \boldsymbol{t}_1) dS + \int_{\Gamma_{4-0}} N_i (N_0 \boldsymbol{t}_0 + N_4 \boldsymbol{t}_4) dS \tag{73}$$
$$= \int_{\Gamma_{0-1}} N_i N_0 dS \quad \boldsymbol{t}_0 + \int_{\Gamma_{0-1}} N_i N_1 dS \quad \boldsymbol{t}_1 + \int_{\Gamma_{4-0}} N_i N_0 dS \quad \boldsymbol{t}_0 + \int_{\Gamma_{4-0}} N_i N_4 dS \quad \boldsymbol{t}_4$$

Element 1:
$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{1-2}} N_i \boldsymbol{t} dS \tag{74}$$
$$= \int_{\Gamma_{1-2}} N_i (N_1 \boldsymbol{t}_1 + N_2 \boldsymbol{t}_2) dS \tag{75}$$

Element 2:
$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{2-3}} N_i \boldsymbol{t} dS + \int_{\Gamma_{3-7}} N_i \boldsymbol{t} dS \tag{76}$$

Element 3:
$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{9-8}} N_i \boldsymbol{t} dS + \int_{\Gamma_{8-4}} N_i \boldsymbol{t} dS \tag{77}$$

Element 4:
$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{10-9}} N_i \boldsymbol{t} dS \tag{78}$$
$$= \int_{\Gamma_{10-9}} N_i (N_{10} \boldsymbol{t}_{10} + N_9 \boldsymbol{t}_9) dS \tag{79}$$

Element 5:
$$\int_\Gamma N_i \boldsymbol{t} dS = \int_{\Gamma_{7-11}} N_i \boldsymbol{t} dS + \int_{\Gamma_{11-10}} N_i \boldsymbol{t} dS \tag{80}$$

We see that the