Short Comments on Continuum Mechanics: Theory and Computation

G. Linga, J. Mathiesen, B. F. Nielsen

Version: 13th March 2019

Contents

Introduction

Throughout the lectures, where not stated otherwise, I use the Einstein summation convention where an index appearing twice in a single term implies summation, for example a scalar product between two vectors is written

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{3} a_i b_i = a_i b_i, \tag{1}$$

where a_i and b_i are the i'th components of the vectors $\mathbf{a}$ and $\mathbf{b}$, respectively. A vector $\mathbf{a}$ can be written as

$$\mathbf{a} = a_i \mathbf{e}_i, \tag{2}$$

where $\mathbf{e}_i$ is a unit vector in a given Cartesian coordinate system. Similarly a tensor $\boldsymbol{\sigma}$ can be written on the form

$$\boldsymbol{\sigma} = \sigma_{ij} \mathbf{e}_i \otimes \mathbf{e}_j \tag{3}$$

or simply just as σ_{ij}. In matrix notation, the dyadic product $\mathbf{e_i} \otimes \mathbf{e_j}$ is equivalent to the (i, j)'th entry of the matrix. In "bra-ket" notation you could write a tensor entry $\mathbf{e}_i \otimes \mathbf{e}_j$ as

$$|e_i\rangle \langle e_j|$$

1 Week 1

1.1 Traction or stress vector

The stress vector or traction vector, $\mathbf{T}^{(\mathbf{n})}$, denotes the force acting on a cross-section, ΔS, with a surface normal $\mathbf{n}$ divided by the area of the cross section,

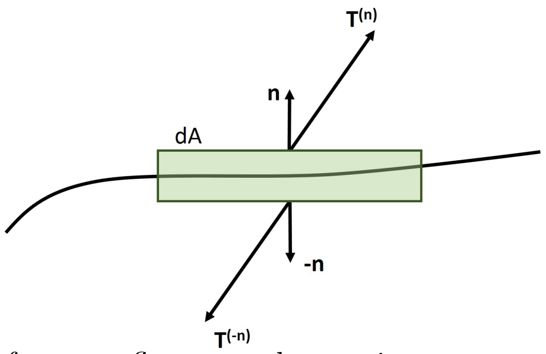
$$\mathbf{T}^{(\mathbf{n})} = \lim_{\Delta S \to 0} \frac{\Delta \mathbf{F}}{\Delta S} = \frac{\partial \mathbf{F}}{\partial S} \tag{4}$$

Consider a small rectangular box with vanishingly small side areas, mass dm, and top and bottom areas dA. The acceleration, $\mathbf{a}$ of the box must be proportional to the sum of all forces on the box, i.e.

$$\mathbf{a} dm = dA \left(\mathbf{T}^{(\mathbf{n})} + \mathbf{T}^{(-\mathbf{n})} \right). \tag{5}$$

In the case where the box is at rest, the sum of the traction on the right-hand side of the equation must vanish, i.e.

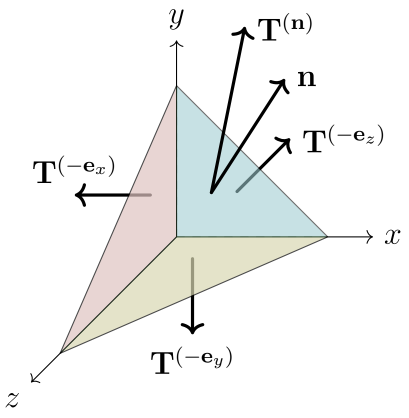
$$\mathbf{T}^{(\mathbf{n})} = -\mathbf{T}^{(-\mathbf{n})} \tag{6}$$



Therefore, the traction vectors on opposite sides of an interface or a flat mass element in rest must be of equal magnitude and point in opposite directions.

Cauchy's stress hypothesis

Instead of a thin box, we now consider the force balance on a small tetrahedron, where, in the case of no body forces, the surface forces acting on the four sides balances with the inertial force.


y
$\mathbf{T}^{(\mathbf{n})}$
$\mathbf{n}$
$\mathbf{T}^{(-\mathbf{e}_z)}$
$\mathbf{T}^{(-\mathbf{e}_x)}$
x
$\mathbf{T}^{(-\mathbf{e}_y)}$
z

We therefore achieve the relation

$$\begin{aligned}\rho dV\mathbf{a} &= \mathbf{T}^{(-\mathbf{e}_x)}dS_x + \mathbf{T}^{(-\mathbf{e}_y)}dS_y + \mathbf{T}^{(-\mathbf{e}_z)}dS_z + \mathbf{T}^{(\mathbf{n})}dS \\ &= -\mathbf{T}^{(\mathbf{e}_x)}dS_x - \mathbf{T}^{(\mathbf{e}_y)}dS_y - \mathbf{T}^{(\mathbf{e}_z)}dS_z + \mathbf{T}^{(\mathbf{n})}dS \\ &= \left(-\mathbf{T}^{(\mathbf{e}_x)}n_x - \mathbf{T}^{(\mathbf{e}_y)}n_y - \mathbf{T}^{(\mathbf{e}_z)}n_z + \mathbf{T}^{(\mathbf{n})}\right)dS. \qquad (7)\end{aligned}$$

Here ρ denotes the mass density and dV is the volume of the tetrahedron. Furthermore, we use the shorthand notation for the individual areas of the sides, $dS_i = \mathbf{e}_i \cdot \mathbf{n}dS$. If we shrink the volume equally in all directions such that we do not change it's shape and let the linear scale of the tetrahedron go to zero, we have that the surface forces will dominate over the volume forces. If we denote the approximate linear size of the tetrahedron ℓ, the area of a side element will be proportional to ℓ^2 and the volume proportional to ℓ^3, i.e. the volume divided by the area will be proportional to ℓ. Neglecting the volume terms in Eq. (7), we end up with the relation

$$\mathbf{T}^{(\mathbf{n})} = \mathbf{T}^{(\mathbf{e}_x)}n_x + \mathbf{T}^{(\mathbf{e}_y)}n_y + \mathbf{T}^{(\mathbf{e}_z)}n_z. \qquad (8)$$

In a more formal way, we could also write this as

$$\mathbf{T}^{(\mathbf{n})} = \underbrace{\left(\sum_i \mathbf{T}^{(\mathbf{e}_i)} \otimes \mathbf{e_i}\right)}_{\text{The stress tensor } \boldsymbol{\sigma}} \cdot\mathbf{n}, \qquad (9)$$

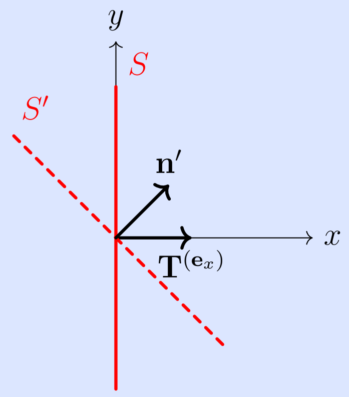
where it now becomes clear that the traction on an arbitrary interface with normal $\mathbf{n}$ is a sum over the tractions $\mathbf{T}^{(\mathbf{e}_i)}$ weighted with the projections $\mathbf{e}_i \cdot \mathbf{n}$. In index notation, we can write the traction vector,

$$T_i^{(\mathbf{n})} = \sigma_{ij}n_j \qquad (10)$$

Similarly, we can also compute the normal stress acting on the surface by projecting $\mathbf{T}^{(\mathbf{n})}$ onto $\mathbf{n}$, i.e.

$$\sigma_{nn} = \mathbf{n} \cdot \mathbf{T}^{(\mathbf{n})} = n_i T_i^{(\mathbf{n})} = n_i \sigma_{ij} n_j \tag{11}$$

Example: Traction vector in 2D



Assume that we have a traction, $\mathbf{T}^{(\mathbf{e}_x)}$, acting on a surface perpendicular to the x-axis.

$$\mathbf{T}^{(\mathbf{e}_x)} = \sigma_0 \mathbf{e}_x. \tag{12}$$

From the traction vector, we find that

$$\sigma_{xx} = \mathbf{e}_x \cdot \mathbf{T}^{(\mathbf{e}_x)} = \sigma_0, \qquad \sigma_{yx} = \mathbf{e}_y \cdot \mathbf{T}^{(\mathbf{e}_x)} = 0 \tag{13}$$

If we assume that no force is acting on the surface perpendicular to the y-axis, we achieve a stress tensor on the form

$$\sigma = \begin{pmatrix} \sigma_0 & 0 \\ 0 & 0 \end{pmatrix} \tag{14}$$

We now consider the corresponding traction vector acting on the surface S' (represented by the dashed line in the figure) with surface normal $\mathbf{e}_{x'} = 1/\sqrt{2}(\mathbf{e_x} + \mathbf{e_y})$ and tangent $\mathbf{e}_{y'} = 1/\sqrt{2}(-\mathbf{e_x} + \mathbf{e_y})$.
The traction can be computed by going back to first principles, i.e. using (4)

$$\mathbf{T}^{(\mathbf{e}_{x'})} = \lim_{\Delta S' \to 0} \frac{\sigma_0 \mathbf{e}_x \Delta S}{\Delta S'} = \frac{\sigma_0 \mathbf{e}_x}{\sqrt{2}}. \tag{15}$$

Alternatively we may use (9) and compute the traction $\mathbf{T}^{(\mathbf{e}_{y'})}$ as

$$\mathbf{T}^{(\mathbf{e}_{y'})} = \mathbf{T}^{(\mathbf{e}_x)}(\mathbf{e}_x \cdot \mathbf{e}_{y'}) + \mathbf{T}^{(\mathbf{e}_y)}(\mathbf{e}_y \cdot \mathbf{e}_{y'}) = -\frac{\sigma_0 \mathbf{e}_x}{\sqrt{2}}. \tag{16}$$

Expressing this in the frame of x' and y', we have that

$$\mathbf{T}^{(\mathbf{e}_{x'})} = \frac{\sigma_0}{2}\Big(\mathbf{e_{x'}} + \mathbf{e_{y'}}\Big), \quad \mathbf{T}^{(\mathbf{e}_{y'})} = \frac{\sigma_0}{2}\Big(\mathbf{e_{x'}} + \mathbf{e_{y'}}\Big). \tag{17}$$

In this new frame, we can also calculate the stress tensor components

$$\sigma_{x'x'} = \mathbf{e}_{x'} \cdot \mathbf{T}^{(\mathbf{e}_{x'})} = \frac{\sigma_0}{2}, \qquad \sigma_{y'x'} = \mathbf{e}_{y'} \cdot \mathbf{T}^{(\mathbf{e}_{x'})} = \frac{\sigma_0}{2}. \tag{18}$$

$$\sigma' = \frac{\sigma_0}{2} \left(\begin{array}{cc} 1 & 1 \\ 1 & 1 \end{array} \right) \tag{19}$$

By performing a rotation, $R(\theta)$, of frame by an angle $\theta = \pi/4$, one can verify (**do it**) that

$$\sigma = R(\theta)\sigma' R(-\theta) \tag{20}$$

Remember that

$$R(\theta) = \left(\begin{array}{cc} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{array} \right) \tag{21}$$

Exercise

For the following stress tensor calculate the traction on a surface with a normal $\mathbf{n}(\theta)$ for different angles θ. The angle should be measured between the normal and the positive x-axis,

$$\sigma = \frac{\sigma_0}{2} \left(\begin{array}{cc} 1 & 2 \\ 2 & 1 \end{array} \right) \tag{22}$$

1.2 Force balance

We write the total force acting on a body as a sum of body forces F_{body} and surface forces $F_{surface}$,

$$\mathbf{F}_{total} = \mathbf{F}_{surface} + \mathbf{F}_{body}. \tag{23}$$

The surface force is given by a surface integral of the traction vector $\mathbf{T}^{(\mathbf{n})}$

$$\mathbf{f}_{surface} = \oint_S \mathbf{T}^{(\mathbf{n})} dS \tag{24}$$

We now consider the force in the i'th direction and by inserting Eq. (10), the surface integral assumes the form

$$F_{surface,i} = \oint_S T_i^{(\mathbf{n})} dS = \oint_S \sigma_{ij} n_j dS. \tag{25}$$

By use of the divergence theorem the surface integral can be written as a volume integral,

$$F_{surface,i} = \int_V \partial_j \sigma_{ij} dV. \tag{26}$$

If we write the body force in terms of a force density

$$F_{body,i} = \int_V f_{body,i}\ dV, \tag{27}$$

we achieve the following expression for the total force in the i'th direction

$$F_{total,i} = \int_V f_{body,i} + \partial_j \sigma_{ij}\ dV, \tag{28}$$

If the body is everywhere in mechanical equilibrium, we must have that for any volume over which we integrate the total force vanishes, we therefore have the following force balance condition at mechanical equilibrium,

$$0 = f_{body,i} + \partial_j \sigma_{ij}, \tag{29}$$

Exercise

Pressure is always acting normal to a surface. For example if you rotate an object in your hand, the atmospheric pressure p acting on a given surface element will not change. The corresponding traction can therefore be written on the form

$$T_i^{(\mathbf{n})} = -pn_i$$

If we now have a constant pressure acting everywhere on an object and we neglect any body force, show that the $\mathbf{F}_{total} = 0$.

1.3 Strain

Deformation and stress go hand in hand, however, in order to formalize the relationship between them, we need a way to describe deformation. We shall in the following define strain for objects, which are only deformed minutely, say typically less than a percent. For example, many construction materials will only sustain a minute degree of deformation before yielding. For those materials it is often sufficient to only consider a linearized description of deformation. That being said, materials like rubber can sustain large deformation without yielding and do in fact have a highly non-trivial stress and strain relations. In the formalism of deformation, it is convenient to introduce a reference state, which we here assume to be undeformed

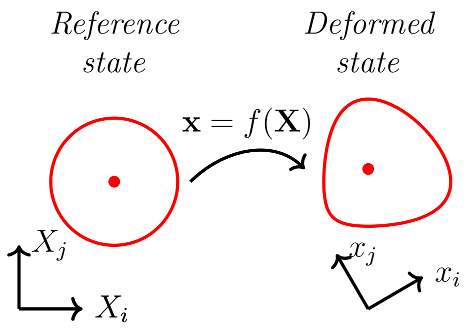


Figure 1: Mapping between a material reference state, described by a coordinate $\mathbf{X}$, and a deformed state described by a coordinate $\mathbf{x}$
.

and stress free[1]. The reference or material state is described by a Cartesian coordinate $\mathbf{X}$ and the deformed state with a coordinate $\mathbf{x}$. The deformation map (bijective function) between the two states is denoted

$$\mathbf{x} = f(\mathbf{X}). \tag{30}$$

If an object is moved and deformed relative to the reference state, each point will be displaced a new point in the deformed frame by a displacement vector field, which in the so-called Eulerian description has the form

$$\mathbf{u}(\mathbf{x}) \equiv \mathbf{x} - \mathbf{X}(\mathbf{x}) = \mathbf{x} - f^{-1}(\mathbf{x}). \tag{31}$$

Note that in this description, the reference coordinate is considered to be a function of the deformed coordinate and that the displacement vector points from the reference coordinate to the deformed coordinate.

1.4 The Euler Strain Tensor

During deformation, the distance vector between two infinitesimally close material points in the reference frame, e.g. $d\mathbf{X} = \mathbf{X}_2 - \mathbf{X}_1$, is changed into a new distance vector $d\mathbf{x} = \mathbf{x}_2 - \mathbf{x}_1$. The change in length of these elements $dL^2 = d\mathbf{X} \cdot d\mathbf{X}$ and $d\ell^2 = d\mathbf{x} \cdot d\mathbf{x}$ is computed by first performing an expansion to linear order of the map following Eq. (30)

$$dX_i \approx \frac{f_i^{-1}}{\partial x_\ell} dx_\ell.$$

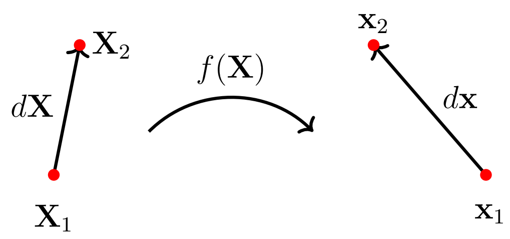


Figure 2: Transformation of a vector $d\mathbf{X}$ by the deformational map $f(\mathbf{X})$.

We now replace the individual coordinates in the configuration state by this expansion and achieve

$$\begin{aligned} dx_i dx_i - dX_i dX_i &= dx_i dx_i - \frac{\partial f_i^{-1}}{\partial x_\ell}\frac{\partial f_i^{-1}}{\partial x_k} dx_\ell dx_k \\ &= \left(\delta_{k\ell} - \frac{\partial f_i^{-1}}{\partial x_\ell}\frac{\partial f_i^{-1}}{\partial x_k}\right) dx_\ell dx_k \end{aligned}$$

If we further apply the derivative to Eq. (31) and replace the map by the displacement field, we obtain

$$\begin{aligned} dx_i dx_i - dX_i dX_i &= \left(\delta_{k\ell} - \left[\delta_{ik} - \frac{\partial u_i}{\partial x_k}\right]\left[\delta_{i\ell} - \frac{\partial u_i}{\partial x_\ell}\right]\right) dx_k dx_\ell \\ &= \left(\frac{\partial u_\ell}{\partial x_k} + \frac{\partial u_k}{\partial x_\ell} - \frac{\partial u_i}{\partial x_k}\frac{\partial u_i}{\partial x_\ell}\right) dx_k dx_\ell \\ &= 2u_{k\ell} dx_k dx_\ell \end{aligned} \tag{32}$$

[1]More generally, we could develop a description where the reference state can have stress and be deformed.

Where we in the latter have introduced the symmetric strain tensor given by

$$u_{ij} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} - \frac{\partial u_n}{\partial x_i}\frac{\partial u_n}{\partial x_j}\right) \tag{33}$$

Throughout the course, we will generally neglect the non-linear terms and only consider the linear strain tensor

$$u_{ij} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \tag{34}$$

2 Week 2

2.1 Hooke's law and the Navier-Cauchy equation

For linear elasticity, the stress is assumed to be a linear combination of strain components (u_{kl})

$$\sigma_{ij} = E_{ijkl}u_{kl} \tag{35}$$

The fourth order tensor, E_{ijkl}, relating the stress and strain tensor components is called the stiffness tensor. It tells you how a strain u_{kl} affects the stress component σ_{ij}. For a fully isotropic medium, in which an applied stress (or strain) will give rise to the same strain (or stress) state, regardless of the direction of application, the stiffness tensor can involve only two types of contributions:

- A pressure-like term $\lambda\delta_{ij}\delta_{kl}$ giving nonzero components of the form E_{1111}, E_{1122}, E_{2222} etc.
- A term of the form $\mu\delta_{ik}\delta_{jl}$ which ensures that e.g. a strain of the form u_{12} also gives rise to a nonzero stress σ_{12} but not a stress in some other direction (e.g. σ_{13}), which would be an example of anisotropy

This latter term must in fact be symmetrized since the stress and strain tensors are symmetric, so the full isotropic stiffness tensor becomes

$$E_{ijkl} = \mu(\delta_{ik}\delta_{jl} + \delta_{jk}\delta_{il}) + \lambda\delta_{ij}\delta_{kl}. \tag{36}$$

You may then verify that stress is related to the strain by

$$\sigma_{ij} = 2\mu u_{ij} + \lambda\delta_{ij}u_{kk} \tag{37}$$

If we now consider the case of infinitesimal strains, where the strain tensor is given by Eq. (34), we arrive at the Navier-Cauchy equation by combining Eqs. (29), (37) and (34),

$$0 = \mu\nabla^2\mathbf{u} + (\mu + \lambda)\nabla(\nabla\cdot\mathbf{u}) \tag{38}$$

In numerical solutions of the mechanical forces in a deformed elactic body, we can either try to solve directly for the stress tensor using Eq. (29) or solve for the displacement using Eq. (95)

2.2 Elastic energy

The change in the internal energy ϵ of our elastic system follows from thermodynamics and assumes the form

$$d\epsilon = Tds + \sigma_{ij}du_{ij} \tag{39}$$

The first term is the usual thermal contribution to the energy while the last term represents the work done when the material undergoes strain. Think of it as ”mechanical work equals force times distance”.
In solving real problems it is in many cases more convenient to consider the free energy, which assumes the form

$$f = \epsilon - Ts. \tag{40}$$

By combining Eqs.(39) and (40), we arrive at an expression for the change in free energy

$$df = -sdT + \sigma_{ij}du_{ij} \tag{41}$$

Under the assumption of constant temperature, we may calculate the stress by the following expression

$$\sigma_{ij} = \left(\frac{\partial f}{\partial u_{ij}}\right)_{T=const} \tag{42}$$

Next we assume that the free energy is always positive and isotropic and of maximum second order. For a general stiffness tensor E_{ijkl} this leads to the expression

$$f = \frac{1}{2}E_{ijkl}u_{ij}u_{kl}, \tag{43}$$

which can readily be checked to be in agreement with the linear stress-strain relation $\sigma_{ij} = E_{ijkl}u_{kl}$ by using (42).
For linear isotropic materials this reduces to the following expression for the free energy

$$f = \frac{1}{2}\lambda(u_{ii})^2 + \mu u_{ij}^2 = \frac{1}{2}\lambda\mathrm{Tr}(\mathbf{u})^2 + \mu u_{ij}^2 \tag{44}$$

Again, this is consistent with Hooke’s law when using (42). As is evident from the above equation, the free energy is a homogeneous function of second order.

Homogeneous functions

A function $f(x, y)$ is said to the homogeneous of order n if it satisfies the relation

$$f(tx, ty) = t^n f(x, y) \tag{45}$$

By differentiating on both sides with regards to t we get from the right-hand side

$$\frac{d}{dt}t^n f(x,y) = nt^{n-1}f(x,y) \tag{46}$$

and from the left-hand side

$$\frac{d}{dt}f(tx,ty) = \frac{\partial f}{\partial(tx)}\frac{\partial tx}{\partial t} + \frac{\partial f}{\partial(ty)}\frac{\partial ty}{\partial t} = x\frac{\partial f}{\partial(tx)} + y\frac{\partial f}{\partial(ty)} \tag{47}$$

By equating the two above equations and setting $t = 1$, we arrive at another relation satisfied by homogeneous function.

$$nf(x,y) = x\partial_x f(x,y) + y\partial_y f(x,y). \tag{48}$$

More generally, for a function f of several indexed variables x_i the relation reads

$$nf = x_i\frac{\partial f}{\partial x_i}. \tag{49}$$

Using that the free energy is of second order, we may use (49) to write

$$2f = u_{ij}\frac{\partial f}{\partial u_{ij}} \tag{50}$$

In combination with Eq. (42) we then achieve the final expression for the free energy density for an elastic material (under constant temperature)

$$f = \frac{1}{2}\sigma_{ij}u_{ij} \tag{51}$$

2.3 Linear elastodynamics from Harmonic Oscillators

In this section we will consider a long series of springs, but let us start by recalling the mechanics of combining springs in parallel and in series.
For two springs with spring constants k_1 and k_2 connected in parallel, the total force is the sum of the forces exerted by each spring

$$Kx = k_1x_1 + k_2x_2 \tag{52}$$

where K is the effective (total) spring constant.
When connected in parallel, the extension of each spring must equal the total extension of the system, i.e. $x_1 = x_2 = x$. We thus find

$$K = k_1 + k_2. \tag{53}$$

This is of course easily generalized to an arbitrary number N of springs:

$$K = \sum_{i=1}^{N} k_i, \quad \text{(Parallel)}. \tag{54}$$

For springs in series, each spring is pulled by the same force of magnitude F and their extension is then given by $x_i = F/k_i$. Furthermore the total extension of the spring system is the sum of the individual extensions:

$$F = Kx = K(x_1 + x_2) = K\left(\frac{F}{k_1} + \frac{F}{k_2}\right) \Rightarrow 1 = K\left(\frac{1}{k_1} + \frac{1}{k_2}\right). \tag{55}$$

This then leads to the effective (total) spring constant K given by

$$K = \frac{1}{\frac{1}{k_1} + \frac{1}{k_2}}. \tag{56}$$

Again, this is easily generalized to an N-spring system

$$K = \frac{1}{\sum_{i=1}^{N} k_i^{-1}}, \quad \text{(Series)}. \tag{57}$$

For the special case of identical springs $k_1 = k_2 = \cdots = k_N := k$, we then just find

$$K = \frac{k}{N}, \quad \text{(Series, } N \text{ identical springs)}. \tag{58}$$

Consider now a series of springs, each of equilibrium length ℓ_{eq} and individual spring constants k. The equilibrium length of the entire spring system is L_{eq}. We may then use the above relation to find the effective spring constant:

$$K = \frac{k}{N} = \frac{\ell_{\text{eq}}}{L_{\text{eq}}} k. \tag{59}$$

The extension of the entire spring is of course proportional to the extensions of the individual springs:

$$\Delta L := L - L_{\text{eq}} = N(\ell - \ell_{\text{eq}}) =: N\Delta\ell. \tag{60}$$

We let the equilibrium positions of the springs be x_i, such that $x_{i+1} = x_i + \ell$. When perturbed, the springs will of course be displaced in a time-dependent manner. We let $u_i(x,t)$ be the displacement, i.e.

$$x_i \to x_i + u_i(x,t). \tag{61}$$

Then Newtons's second law combined with Hooke's Law gives

$$m\frac{\partial^2 u_i}{\partial t^2} = k(\Delta u_{i+1} - \Delta u_i), \tag{62}$$

where Δu_i means $u_i - u_{i-1}$. We thus find

$$m\frac{\partial^2 u_i}{\partial t^2} = k(u_{i+1} - 2u_i + u_{i-1}). \tag{63}$$

Now, recall that the second derivative can be computed as

$$f''(x) = \lim_{h\to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \tag{64}$$

We can take the continuum limit by treating u_i as a continuous variable and letting ℓ_{eq} be small. This means

$$u_{i+1} - 2u_i + u_{i-1} \to u(x+\ell_{\text{eq}}) - 2u(x) + u(x-\ell_{\text{eq}}) \approx \ell_{\text{eq}}^2 \frac{\partial^2 u}{\partial x^2}. \tag{65}$$

In the continuum limit we thus find

$$\frac{\partial^2 u}{\partial t^2} = \frac{k\ell_{\text{eq}}^2}{m}\frac{\partial^2 u}{\partial x^2}. \tag{66}$$

By comparison with the wave equation, we find a speed of sound (disturbance propagation speed) in this elastic medium of

$$c_s = \ell_{\text{eq}}\sqrt{\frac{k}{m}}. \tag{67}$$

If we assume each mass to have a cross-sectional area A, the mass density is $\rho = \frac{m}{A\ell_{\text{eq}}}$. In this case the speed of sound may be expressed as

$$c_s = \sqrt{\frac{E}{\rho}}. \tag{68}$$

where E is Young's Modulus which is – as we shall argue in a moment – a material parameter. Young's modulus is, in general, defined as the tensile stress per unit of relative strain. In the present case, the relation is simply

$$E = \frac{k\ell_{\text{eq}}}{A}. \tag{69}$$

2.3.1 Material Parameters in Linear Elasticity

There are two material parameters relevant to linear elasticity.

Young's Modulus describes how hard it is to stretch a material in the direction you apply the force (here: the x-direction):

$$E = \frac{\sigma_{xx}}{u_{xx}}. \tag{70}$$

This is a sensible material parameter because

- u_{xx} is independent of the absolute length (as opposed to e.g. u_x).
- σ_{xx} is 'force per area', but the effective spring constant $K \propto A$ and so σ_{xx} is independent of cross sectional area.

The relation between the spring constant and Young's modulus can be worked out as

$$E = \frac{\sigma_{xx}}{u_{xx}} = \frac{F/A}{\Delta x/L} = \frac{K\Delta x L}{A\Delta x} = \frac{KL}{A}. \tag{71}$$

Since $K \propto A$ and $K \propto 1/L$ it is evident that E is indeed a material parameter, depending only on the intrinsic properties of the material.

Poisson's Ratio parametrizes how much a material expands/contracts in the transverse direction when stretched. Imagine that the material is stretched by a strain u_{xx} and the transverse direction of interest is the y-direction. Then Poisson's ratio ν is given by

$$\nu = -\frac{u_{yy}}{u_{xx}}. \tag{72}$$

In a linearly elastic material, both the denominator and numerator will be proportional to F, which then drops out and leaves us with a sensible material parameter.

2.4 Reynold's Transport Theorem

Consider a global quantity Q which can be computed from an integral of a corresponding volumetric density $q(\mathbf{x}, t)$

$$Q = \int_{V(t)} q(\mathbf{x}, t) dV. \tag{73}$$

We will now consider the temporal change of Q when both the volume over which we integrate and $q(\mathbf{x}, t)$ change with time, i.e.

$$\frac{dQ}{dt} = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left[\int_{V(t+\Delta t)} q(\mathbf{x}, t + \Delta t) dV - \int_{V(t)} q(\mathbf{x}, t) dV \right] \tag{74}$$

Since Δt will be small, we can expand to leading order the change in q,

$$q(\mathbf{x}, t + \Delta t) \approx q(\mathbf{x}, t) + \frac{\partial q}{\partial t}\Delta t \tag{75}$$

If the volume that we consider, V(t), moves and deforms with a velocity $\mathbf{v}(\mathbf{x}, t)$, we can for small time steps, assume that the change in volume can be computed by an integration along the surface of the volume ∂V. In other words, a small segment dS of the boundary gives rise to a change in volume dV

$$dV = (\mathbf{v} \cdot \mathbf{n})\Delta t dS, \tag{76}$$

where $\mathbf{n}$ is a normal to the surface. Note that we multiply with Δt because $\mathbf{v}\Delta t$ is the distance that the boundary moves in Δt. We can now rewrite the first integral in Eq. (74)

$$\int_{V(t+\Delta t)} q(\mathbf{x}, t + \Delta t) dV = \int_{V(t)} \left(q(\mathbf{x}, t) + \frac{\partial q}{\partial t}\Delta t \right) dV + \int_{\Delta V} \left(q(\mathbf{x}, t) + \frac{\partial q}{\partial t}\Delta t \right) dV, \tag{77}$$

where ΔV is the difference between the volumes $V(t)$ and $V(t + \Delta t)$. If we further use Eq. (76) to rewrite the second integral on the right-hand side of Eq. (77) and disregard the term of order $\mathcal{O}(\Delta t^2)$, we get

$$\int_{V(t+\Delta t)} q(\mathbf{x}, t + \Delta t) dV = \int_{V(t)} \left(q(\mathbf{x}, t) + \frac{\partial q}{\partial t}\Delta t \right) dV + \oint_{\partial V} q(\mathbf{x}, t)(\mathbf{v} \cdot \mathbf{n})\Delta t dS. \tag{78}$$

By inserting this expression into Eq. (74), we get

$$\frac{dQ}{dt} = \int_{V(t)} \left(\frac{\partial q(\mathbf{x}, t)}{\partial t} \right) dV + \oint_{\partial V} q(\mathbf{x}, t)(\mathbf{v} \cdot \mathbf{n}) dS, \tag{79}$$

where all the Δt cancels out. Now by the divergence theorem (on the second integral), we arrive at Reynold's transport theorem.

$$\frac{dQ}{dt} = \int_{V(t)} \left(\frac{\partial q}{\partial t} + \nabla \cdot (q\mathbf{v}) \right) dV. \tag{80}$$

Example: Change in volume element

We can apply Reynold's Transport Theorem in order to figure out the change in a volume element under deformation. Let Q be the volume $Q := V$. Then q is a volume density and thus necessarily $q = 1$.
We consider the displacement to be $u_i = v_i \mathrm{d}t$ and thus find, for a small volume,

$$\delta(\mathrm{d}V) = \int_{V(t)} (\nabla \cdot \mathbf{v}\mathrm{d}t)\mathrm{d}V = (\nabla \cdot \mathbf{u})\mathrm{d}V. \tag{81}$$

In other words, the relative change in volume is given by the gradient of the displacement field:

$$\frac{\mathrm{d}V' - \mathrm{d}V}{\mathrm{d}V} = \frac{\delta(\mathrm{d}V)}{\mathrm{d}V} = \nabla \cdot \mathbf{u}. \tag{82}$$

Example: Mass Conservation

We now consider the change in mass

$$M = \int_{V(t)} \rho(\mathbf{x}, t)\mathrm{d}V. \tag{83}$$

If the volume $V(t)$ is co-moving with the material, i.e. the volume moves with same velocity as the material, the mass must be conserved,

$$0 = \frac{\mathrm{d}M}{\mathrm{d}t} = \int_{V(t)} \left(\frac{\partial \rho}{\partial t} + \nabla \cdot \big(\rho \mathbf{v}\big) \right) \mathrm{d}V. \tag{84}$$

Since this has to be true for any choice of co-moving volume, we must have

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \big(\rho \mathbf{v}\big) = 0. \tag{85}$$

This equation is also known as the mass-continuity equation.

3 Week 3

3.1 Balance laws

We will throughout this course primarily be concerned about the balance laws for mass and momentum (and less about energy balance and other balance laws). The balance law for momentum follows from Reynolds transport theorem by considering the local momentum density ρv_i, where ρ is the mass density and v_i the velocity component in the i'th direction. From Newton's second law, we have that the change in total momentum $\mathbf{P}$ is equal to the sum of all forces $\mathbf{F}$,

$$\frac{d\mathbf{P}}{dt} = \mathbf{F}. \tag{86}$$

If we use that

$$P_i = \int \rho v_i dV \tag{87}$$

and use Eq. (28), we achieve from Reynolds transport theorem (assuming that the velocity of the representative volume is the co-moving velocity) Eq. (80) that

$$\partial_t(\rho v_i) + \partial_j(\rho v_i v_j) = f_{body,i} + \partial_j \sigma_{ij}. \tag{88}$$

We now take the derivative of individual terms and arrive at

$$v_i \partial_t \rho + \rho \partial_t v_i + v_i \partial_j(\rho v_j) + \rho v_j \partial_j v_i = f_{body,i} + \partial_j \sigma_{ij}, \tag{89}$$

which we can write on the form

$$v_i[\partial_t \rho + \partial_j(\rho v_j)] + \rho[\partial_t v_i + v_j \partial_j v_i] = f_{body,i} + \partial_j \sigma_{ij}, \tag{90}$$

If we invoke mass balance, the first bracket vanishes and we arrive at the following two fundamental balance laws

$$\begin{aligned} \partial_t \rho + \partial_j(\rho v_j) &= 0 \\ \rho[\partial_t v_i + v_j \partial_j v_i] &= f_{body,i} + \partial_j \sigma_{ij} \end{aligned} \tag{91}$$

We often use the short-hand notation

$$\frac{D}{Dt} \equiv \partial_t + v_j \partial_j$$

for the so-called co-moving derivative. For example consider the mass conservation, which using the co-moving derivative can be written on the form

$$\frac{D\rho}{Dt} = \rho \partial_j v_j. \tag{92}$$

This equation states that the temporal change in density inside a co-moving volume (the left-hand side) is due to the compression or de-compression of the volume element (right-hand side, see Eq. (82)).

3.2 The temporal elasticity equation

If we insert Hooke's law, $\sigma_{ij} = 2\mu u_{ij} + \lambda \delta_{ij} u_{kk}$, in the momentum balance law Eq. (91), and note that velocity field follows from the displacement field $\mathbf{v} = \partial_t \mathbf{u}$, we arrive at

$$\rho[\partial_t^2 u_i + v_j \partial_j v_i] = f_{body,i} + \mu \partial_{kk}^2 u_i + (\mu + \lambda)\partial_i \partial_k u_k \tag{93}$$

Here, we shall consider only small deformations and small amplitude elastic waves, such that the advection term in the equation can be neglected. We therefore have that

$$\rho \partial_t^2 u_i = f_{body,i} + \mu \partial_{kk}^2 u_i + (\mu + \lambda)\partial_i \partial_k u_k \tag{94}$$

In vector notation the equation has the form

$$\rho\partial_t^2\mathbf{u} = \mathbf{f}_{body} + \mu\nabla^2\mathbf{u} + (\mu+\lambda)\nabla(\nabla\cdot\mathbf{u}). \tag{95}$$

If we now use the vector identify

$$\nabla\times\nabla\times\mathbf{u} = \nabla(\nabla\cdot\mathbf{u}) - \nabla^2\mathbf{u} \tag{96}$$

We can write Eq. (95) as

$$\rho\partial_t^2\mathbf{u} = \mathbf{f}_{body} - \mu\nabla\times\nabla\times\mathbf{u} + (2\mu+\lambda)\nabla(\nabla\cdot\mathbf{u}). \tag{97}$$

Helmholtz decomposition

Any sufficiently smooth vector field, $\mathbf{u}$ can be decomposed in an irrotational (curl-free), $\mathbf{u}_L$, and a solonoidal (divergence free) field, $\mathbf{u}_T$,

$$\mathbf{u} = -\nabla\varphi + \nabla\times\mathbf{\Psi} = \mathbf{u}_L + \mathbf{u}_T \tag{98}$$

If we decompose in irrotational and divergence free fields and neglect body forces, we can write Eq. (100) as two seperate equations

$$\rho\partial_t^2\mathbf{u}_L = (2\mu+\lambda)\nabla^2\mathbf{u}_L. \tag{99}$$

and

$$\rho\partial_T^2\mathbf{u}_T = \mu\nabla^2\mathbf{u}_T. \tag{100}$$

4 Week 4

4.1 Ideal and nearly ideal flow

This week, we will consider inviscid fluids, i.e. fluids where the viscosity is equal to zero. As usual, we start from the mass and momentum balance equations. In addition, we assume that the only contribution to the stress tensor comes from a pressure field. In that case, we can write

$$\sigma_{ij} = -p\delta_{ij} \tag{101}$$

If we insert the stress tensor in the momentum balance law, we arrive at the Euler equation for fluid flow accompanied by mass balance,

$$\begin{aligned}\rho(\partial_t + v_j\partial_j)v_i &= f_i^{body} - \partial_i p\\ \partial_t\rho + \partial_j(\rho v_j) &= 0\end{aligned} \tag{102}$$

In the incompressible case, we replace the mass balance equation with $\partial_j v_j = 0$. There is an inherent symmetry in the Euler equation, which becomes apparent after some rewriting. We therefore first define the vorticity of the flow

$$\omega \equiv \nabla \times \mathbf{v} \tag{103}$$

We now use the identity

$$\mathbf{v} \times \omega = \frac{1}{2}\nabla v^2 - (\mathbf{v} \cdot \nabla)\mathbf{v} \tag{104}$$

We insert this identify in the Euler equation and assume that the body forces are conservative and can be written on the form $\mathbf{f}^{body} = -\rho\nabla\Phi$. We now have

$$\partial_t \mathbf{v} - \mathbf{v} \times \omega = -\nabla\left(\frac{1}{2}v^2 + \frac{p}{\rho} + \Phi\right) \tag{105}$$

For a steady-state (meaning $\partial_t \mathbf{v} = 0$) and irrotational flow (the vorticity is zero), we have Bernoulli's law

$$\nabla\left(\frac{1}{2}v^2 + \frac{p}{\rho} + \Phi\right) = 0. \tag{106}$$

By integrating once, we see that the quantity

$$H \equiv \frac{1}{2}v^2 + \frac{p}{\rho} + \Phi \tag{107}$$

is conserved when following a small fluid volume.

4.2 Potential flow

Consider an irrotational and incompressible flow, i.e.

$$\nabla \times \mathbf{v} = 0, \quad \text{and} \quad \nabla \cdot \mathbf{v} = 0. \tag{108}$$

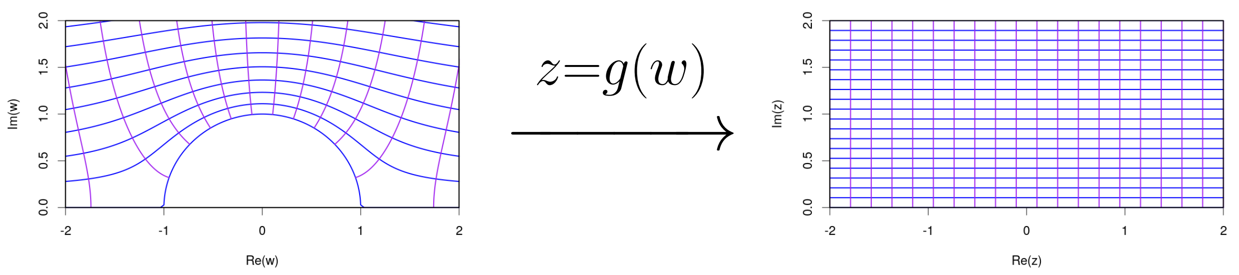
If this case, the flow field is determined by the gradient of a scalar field, which by the incompressibility satisfies the Laplace equation

$$\mathbf{v} = -\nabla\psi, \quad \text{and} \quad \nabla \cdot \mathbf{v} = -\nabla^2\psi = 0 \tag{109}$$

Two-dimensional potential flow over a half-cylinder

Potential problems in two-dimensions are often conveniently solved in the complex plane. For example, consider a cut of a half-cylinder, which becomes a circle, in the upper complex plane with a coordinate $w = r + is$ (where u and v are real numbers). The outside of the circle is mapped to the full upper plane by a transformation (check!)

$$z = g(w) = w + 1/w \tag{110}$$



In the coordinate w, we can write the Laplace equation

$$\nabla^2\psi = (\partial_r^2 + \partial_s^2)\psi = (\partial_r + i\partial_s)(\partial_r - i\partial_s)\psi = 4\partial_{\bar{w}}\partial_w\psi \tag{111}$$

Similarly, we can write the Laplace equation in the plane z without the circle, i.e. we just solve for a parallel flow to the x-axis. We call the potential in the full plane for ϕ. We can calculate this potential either by using complex or real variables. Here we use the complex variables (the real variable case can be seen in the book). We now have

$$4\partial_z\partial_{\bar{z}}\phi = 0,$$

which implies that

$$\phi(z,\bar{z}) = h_1(z) + h_2(\bar{z}),$$

where h_1 and h_2 are some functions which must satisfy the boundary conditions. For an undisturbed ideal flow parallel to the x-axis, we have that $\mathbf{v} = v_0\mathbf{e}_x = -\nabla\phi$ or similarly in the complex variables $-2\partial_{\bar{z}}\phi = v_0$. We therefore have that

$$h_2'(\bar{z}) = v_0/2. \tag{112}$$

Integrating once, we obtain the complex potential

$$\phi(z,\bar{z}) = v_0\bar{z}/2 + \alpha, \tag{113}$$

where α is some constant. Due to the conformal invariance of the Laplace operator $\phi(w,\bar{w}) = \phi(z,\bar{z})$, the potential for the flow around the disc is "simply" given by

$$\psi(w,\bar{w}) = \phi\Big(g(w), \overline{g(w)}\Big) = v_0\overline{g(w)}/2 + \alpha. \tag{114}$$

From the potential, we now obtain the velocity

$$\begin{aligned} v &= 2\partial_{\bar{w}}\psi(w,\bar{w}) \\ &= v_0\partial_{\bar{w}}\overline{g(w)} \\ &= v_0\partial_{\bar{w}}(\bar{w} + 1/\bar{w}) \\ &= v_0(1 - 1/\bar{w}^2) \end{aligned} \tag{115}$$

If we neglect gravity, we obtain from Bernoulli's law that

$$const = \frac{p^\infty}{\rho} + \frac{1}{2}|v|^2 = \frac{p^\infty}{\rho} + \frac{1}{2}v_0^2\left[1 - 2\mathrm{Re}\left(\frac{1}{w^2}\right) + \frac{1}{|w|^4}\right] \tag{116}$$

If we next set $w = re^{i\theta}$ and assume that the pressure at infinity is p^∞, we get

$$\frac{p^\infty}{\rho} + \frac{1}{2}v_0^2 = \frac{p}{\rho} + \frac{1}{2}v_0^2\left[1 - \left(\frac{4\cos^2(\theta) - 2}{r^2}\right) + \frac{1}{r^4}\right] \tag{117}$$

5 Week 5

From the inviscid flows, we now turn to dissipative flows. Following the same arguments leading to Hooke's law Eq. (37), viscous forces arise due to gradient in the flow field, or due to compression/decompression. Fluids homogeneous, isotropic and linear (Newtonian) in the gradient of the velocity field has a stress tensor on the form (including an additional pressure term)

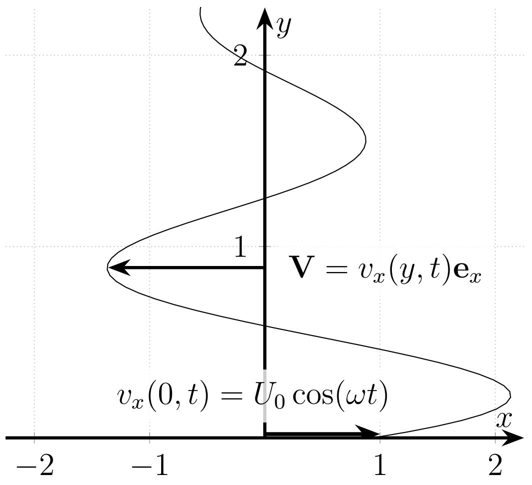
$$\sigma_{ij} = -p\delta_{ij} + \eta(\partial_i v_j + \partial j v_i) + \lambda(\partial_k v_k)\delta_{ij} \tag{118}$$

If we insert this expression in the momentum balance equation, we arrive at the Navier-Stokes equations

$$\rho(\partial_t + v_j\partial_j)v_i = f_i^{body} - \partial_i p + \eta\partial_j^2 v_i + (\eta + \lambda)\partial_i\partial_k v_k.$$

Note the similarity with the elasticity equation. If the flow is incompressible, the last term vanishes in the equation, and we have the incompressible Navier-Stokes eqauation.

5.1 Solutions to Stokes' First and Second Problem


y
2
1
$\mathbf{V} = v_x(y,t)\mathbf{e}_x$
$v_x(0,t) = U_0\cos(\omega t)$
x
-2
-1
1
2

We consider a shear flow parallel to the x-z plane, which only depends on the y coordinate, $\mathbf{v} = v_x(y,t)\mathbf{e}_x$. In this case $\nabla \cdot \mathbf{v} = \partial_x v_x(y,t) = 0$. We further assume that the pressure gradient in the system is zero. The flow is driven by an oscillatory motion $U_0 \cos(\omega t)\mathbf{e}_x$ of the x-z plane in the x direction for $y = 0$, i.e. we have the boundary condition for the flow velocity.

$$\text{Boundary conditions} \begin{cases} v_x(0,t) & = & U_0 \cos(\omega t) \\ v_x(y,0) & = & 0, \quad \text{for } y > 0 \\ v_x(y,t) & \overset{y\to\infty}{\to} & 0 \end{cases} \tag{119}$$

The Navier-Stokes equation reduces in this case to the diffusion equation along the y-axis

$$\partial_t v_x(y,t) = \nu \partial_y^2 v_x(y,t). \tag{120}$$

If we apply the Laplace transform with regard to time, we get

$$s\hat{v}_x(y,s) - v_x(y,0) - \nu \partial_y^2 \hat{v}_x(y,s) = 0. \tag{121}$$

If we use that initially the fluid is at rest everywhere in the system, $v_x(y,0) = 0$, we get the simple second order differential equation

$$0 = \left(\frac{s}{\nu} - \partial_y^2\right) \hat{v}_x(y,s) = \left(\sqrt{\frac{s}{\nu}} + \partial_y\right)\left(\sqrt{\frac{s}{\nu}} - \partial_y\right) \hat{v}_x(y,s). \tag{122}$$

The general solution is

$$\hat{v}_x(y,s) = A(s)e^{-\sqrt{\frac{s}{\nu}}y} + B(s)e^{\sqrt{\frac{s}{\nu}}y}, \tag{123}$$

where $A(s)$ and $B(s)$ are functions depending on s. We see that the second term is not compatible with the boundary condition $v_x \to 0$ for $y \to \infty$ and therefore set $B(s) = 0$.

We find the function $A(s)$ from the boundary condition $v_x(0,t) = U_0 \cos(\omega t)$, by using Eq. (123)

$$\hat{v}_x(0,s) = A(s), \tag{124}$$

which again is

$$\hat{v}_x(0,s) = \int_0^\infty U_0 \cos(\omega t) e^{-st} dt = \frac{U_0 s}{s^2 + \omega^2}. \tag{125}$$

The complete solution to Stokes' problems is therefore in the Laplace transformed time

$$\hat{v}_x(0,s) = \frac{U_0 s}{s^2 + \omega^2} e^{-\sqrt{\frac{s}{\nu}}y}. \tag{126}$$

Now in order to get the solution in the real time variable, we have to apply the inverse Laplace transform. Things get more easy though, by writing the solution, Eq. (126), as a product of the two terms $\hat{g}_1(s) = U_0 s/(s^2 + \omega^2)$ and $\hat{g}_2(s) = \exp(-\sqrt{s/\nu y})$, i.e.

$$\hat{v}_x(y,s) = \hat{g}_1(s)\hat{g}_2(y,s) \tag{127}$$

If we now use the convolution theorem for the Laplace transform, we get

$$\mathcal{L}^{-1}\left[\hat{g}_1(s)\hat{g}_2(y,s)\right] = \int_0^t g_1(t-\tau)g_2(\tau)d\tau, \tag{128}$$

where g_1 and g_2 are the inverse Laplace transformations of $\hat{g}_1$ and $\hat{g}_2$, respectively. Note that the variable y has no influence on the transform. The inverse transform of $\hat{g}_1$ follows directly from Eq. (125) and is $U_0\cos(\omega t)$. The inverse transform of $\hat{g}_2$ is slightly more difficult to compute,

$$\mathcal{L}^{-1}\left[\hat{g}_2(y,s)\right] = \frac{1}{2\pi i}\int_{\lambda-i\infty}^{\lambda+i\infty} e^{-\sqrt{\frac{s}{\nu}}y}e^{st}ds = \frac{\nu}{2\pi i y^2}\int_{\lambda-i\infty}^{\lambda+i\infty} e^{-\sqrt{s}}e^{\frac{\nu t}{y^2}s}ds. \tag{129}$$

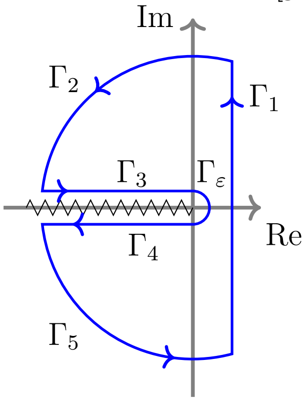


Figure 3: Integration path

After the second equality we have redefined the integration variable s using that y and ν are always positive. The problem in performing the inverse Laplace transform is that the integrand has a branch cut due to the square root in the exponent of the first exponential. We therefore have to be somewhat careful in choosing the contour along which we perform the integration. We choose the negative real axis as the branch cut and let the argument of the complex numbers run from $-\pi$ to π. We choose an integration path avoiding the branch cut as shown in Fig. 3. We note that as usual for $t > 0$ the integration along the path elements Γ_2 and Γ_5 vanish, we therefore have that the inverse Laplace transform can be written along the remaining path elements

$$0 = \frac{\nu}{2\pi i y^2}\int_{\Gamma_1+\Gamma_3+\Gamma_4} e^{-\sqrt{s}}e^{\frac{\nu t}{y^2}s}ds. \tag{130}$$

The combined integral vanishes since no poles exist inside the integration path. If we for simplicity define $\tilde{t} = \nu t/y^2$, we can write

$$\begin{aligned}
\int_{\Gamma_1} e^{-\sqrt{s}}e^{\tilde{t}s}ds &= -\int_{\Gamma_3} e^{-\sqrt{s}}e^{\tilde{t}s}ds - \int_{\Gamma_4} e^{-\sqrt{s}}e^{\tilde{t}s}ds \\
&= -e^{i\pi}\int_{\infty}^{0} e^{-\sqrt{r\exp(i\pi)}}e^{r\exp(i\pi)\tilde{t}}dr - e^{-i\pi}\int_0^{\infty} e^{-\sqrt{r\exp(-i\pi)}}e^{r\exp(-i\pi)\tilde{t}}dr \\
&= -\int_0^{\infty} e^{-i\sqrt{r}}e^{-r\tilde{t}}dr + \int_0^{\infty} e^{i\sqrt{r}}e^{-r\tilde{t}}dr \\
&= -2\int_0^{\infty} e^{-iu}e^{-u^2\tilde{t}}udu + 2\int_0^{\infty} e^{iu}e^{-u^2\tilde{t}}udu \\
&= 2\int_0^{\infty}\left(e^{iu}-e^{-iu}\right)ue^{-u^2\tilde{t}}du \\
&= \int_{-\infty}^{\infty}\left(e^{iu}-e^{-iu}\right)ue^{-u^2\tilde{t}}du \\
&= 2i\mathrm{Im}\left[\int_{-\infty}^{\infty} e^{iu}ue^{-u^2\tilde{t}}du\right],
\end{aligned} \tag{131}$$

where we along the way have made the substitution $r = u^2$ and used the symmetry of the integrand to expand the integration to the whole real axis. We have further taken the imaginary part of the complex exponential outside, noting that the exponential involving u^2 is real. We can now complete the square, perform the Gaussian-like integral and arrive at the result

$$\frac{\nu}{2\pi i y^2}\int_{\Gamma_1} e^{-\sqrt{s}}e^{\tilde{t}s}ds = \frac{\nu}{\pi y^2}\text{Im}\left[\int_{-\infty}^{\infty} e^{iu}ue^{-u^2\tilde{t}}du\right] = \frac{\nu}{\sqrt{4\pi}y^2\tilde{t}^{3/2}}e^{-1/(4\tilde{t})} = \frac{y}{\sqrt{4\pi\nu}t^{3/2}}e^{-\frac{y^2}{4\nu t}} \tag{132}$$

Coming back to Eq. (128), we arrive at the final solution in real space, which has the form

$$V_x(y,t) = \mathcal{L}^{-1}\left[\hat{g}_1(s)\hat{g}_2(y,s)\right] = \frac{U_0 y}{\sqrt{4\pi\nu}}\int_0^t \cos[\omega(t-\tau)]\tau^{-3/2}e^{-\frac{y^2}{4\nu\tau}}d\tau. \tag{133}$$

We now have solutions to the two Stokes problems, which corresponds to the cases where either the bottom plate at $y = 0$ moves at steady speed U_0 or oscillates with a frequency ω. Stokes' first problem, the steady motion of the plate, is solved by letting $\omega = 0$ in Eq. (133) and then by performing the integration,

$$V_x(y,t) = \frac{U_0 y}{\sqrt{4\pi\nu}}\int_0^t \tau^{-3/2}e^{-\frac{y^2}{4\nu\tau}}d\tau = \text{erfc}\left(\frac{y}{2\sqrt{\nu t}}\right), \tag{134}$$

where erfc denotes the complementary error function. Stokes second problem corresponds to the steady state oscillation of the bottom plate. The steady state solution is found by letting time approach infinity, i.e. let the system evolve long enough that any transient dies out.

$$\begin{aligned}
V_x(y,t) &= \frac{U_0 y}{\sqrt{4\pi\nu}}\int_0^t \cos[\omega(t-\tau)]\tau^{-3/2}e^{-\frac{y^2}{4\nu\tau}}d\tau \\
&= \frac{U_0 y}{\sqrt{4\pi\nu}}\left[\int_0^\infty \cos[\omega(t-\tau)]\tau^{-3/2}e^{-\frac{y^2}{4\nu\tau}}d\tau - \int_t^\infty \cos[\omega(t-\tau)]\tau^{-3/2}e^{-\frac{y^2}{4\nu\tau}}d\tau\right] \\
. &\overset{t\gg 1}{\approx} \frac{U_0 y}{\sqrt{4\pi\nu}}\int_0^\infty \cos[\omega(t-\tau)]\tau^{-3/2}e^{-\frac{y^2}{4\nu\tau}}d\tau \\
&= e^{-\sqrt{\frac{\omega}{2\nu}}y}\cos\left(\omega t - \sqrt{\frac{\omega}{2\nu}}y\right)
\end{aligned} \tag{135}$$

6 Week 6

6.1 Stokes flow

In this section, we consider the numerical computation of steady viscous flow in various 2D geometries. We briefly present some background on the finite element method and the FEniCS framework.

The section contains three exercises, which are given with increasing difficulty: (1) Laminar pipe flow, (2) force calculation from flow around objects, and (3) lid-driven cavity flow.

Incompressible flow is in general described by the Navier–Stokes equations,

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u}\right) - \eta\boldsymbol{\nabla}^2\mathbf{u} = \mathbf{f} - \boldsymbol{\nabla}p, \tag{136a}$$

$$\boldsymbol{\nabla}\cdot\mathbf{u} = 0, \tag{136b}$$

where $\mathbf{u}(\mathbf{x}, t)$ is the velocity field, $\mathbf{f}(\mathbf{x}, t)$ a body force, ν the kinematic viscosity, and $p(\mathbf{x}, t)$ the pressure field.

In many settings the inertial forces are small compared to viscous forces, i.e. the Reynolds number $\mathrm{Re} = |\mathbf{u}|\ell/\nu \ll 1$, where ℓ is a characteristic length scale of the flow. In this case, eq. (136a) reduces to the time-independent equation[2]

$$\eta\boldsymbol{\nabla}^2\mathbf{u} = \boldsymbol{\nabla}p - \mathbf{f}. \tag{137}$$

If we further neglect body forces and rescale $p \to \eta p$, we get the equation system

$$\boldsymbol{\nabla}^2\mathbf{u} = \boldsymbol{\nabla}p, \tag{138a}$$

$$\boldsymbol{\nabla}^2 p = 0, \tag{138b}$$

where eq. (138b) is found by taking the gradient of eq. (138a) and using the incompressibility condition eq. (136b).

The system of eqs. (138a) and (138b) are in a domain $\mathbf{x} \in \Omega$. For simplicity, we will assume prescribed velocity boundary conditions,

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}_0(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \partial\Omega_{\mathbf{u}}, \tag{139}$$

and prescribed pressure boundary conditions (normal stress),

$$p(\mathbf{x}) = p_0(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \partial\Omega_p, \tag{140}$$

where $\partial\Omega = \partial\Omega_{\mathbf{u}} \cup \partial\Omega_p$ is the boundary of Ω.

6.2 Finite Element Method

The finite element method (FEM) is an efficient method for solving partial differential equations (PDEs) in complex geometries. Basically, it consists in approximating the *solution* to a PDE rather than approximating the *equation itself* (as you do with finite difference methods and finite volume methods). This is obtained by employing calculus of variations. We will not present the FEM

[2]On very short time scales, however, the time-derivative term may be important.

theory in detail here, just a direct explanation of how our program will work. For a thorough introduction, see e.g. [1, 5].

The quick and dirty recipe of FEM is the following: The domain is discretized by dividing it into many subdomains, called elements. In our case, the elements are triangles. On each element, we approximate the solution by linear combinations of basis functions defined on each element. The element-wise equations defining the coefficients in these linear combinations are assembled for the entire system, yielding a (large, and often sparse) set of algebraic equations to be solved.

In order to use FEM to solve our equations, we must formulate the problem in a weak (variational) form. The basis functions belong to the function spaces V (velocity space) and P (pressure space)

By multiplying our equations (unknowns = trial functions $(\mathbf{u}, p)$) by the test functions $(\mathbf{v}, q)$ and successively integrating by parts, we may obtain the following (weak) problem formulation:

Weak formulation: *Find* $(\mathbf{u}, p) \in W$ *such that*

$$a((\mathbf{u}, p), (\mathbf{v}, q)) = L((\mathbf{v}, q)) \quad \text{for all} \quad (\mathbf{v}, q) \in W, \tag{141a}$$

$$\text{and} \quad \mathbf{u} = u_0 \quad \text{on} \quad \partial\Omega, \tag{141b}$$

where $W = V \times P$ *is a mixed function space, and*

$$a((\mathbf{u}, p), (\mathbf{v}, q)) = \int_\Omega (\boldsymbol{\nabla}\mathbf{u} : \boldsymbol{\nabla}\mathbf{v} - p\boldsymbol{\nabla} \cdot \mathbf{v} + q\boldsymbol{\nabla} \cdot \mathbf{u})\, \mathrm{d}V, \tag{141c}$$

$$L((\mathbf{v}, p)) = -\int_{\partial\Omega_p} p_0 \mathbf{n} \cdot \mathbf{v} \mathrm{d}S. \tag{141d}$$

This problem formulation is all we need. Now, to avoid most of the technicalities of FEM (actually constructing the basis functions, solving the integral equations on the element level, assembling the system matrix, and enforcing the boundary conditions, . . .) and get straight to the physics, we use the FEniCS framework.

6.3 FEniCS/DOLFIN

FEniCS is a collection of software for automated solution of PDEs using FEM [7]. The component of FEniCS we will use to communicate with the FEniCS engine via Python is called DOLFIN [8]. Some of the functionality we will use is

- Making a simple mesh
- Setting up subspaces and basis functions
- Applying boundary conditions
- Solving the system

- Visualizing the solution

The bottom line is, all we need to do is to supply the variational form and boundary conditions and the order of the elements (i.e. the polynomial degree of the basis functions)—FEniCS does the rest (and it's quite fast!).

Tip: If you want to get really dirty with FEniCS, you should consider following their excellent tuturial [6].

6.4 Getting started with FEniCS

If you are a student at University of Copenhagen, you should already have access to FEniCS through ERDA using your University of Copenhagen login on `http://erda.dk`. If it is the first time you log in, you may have to go through a few steps to activate your user. Once you are logged into ERDA you have the possibility to start python through a jupyter notebook by 1) clicking "jupyter" in menu, 2) "start DAG" and 3) start a server with "Finite element notebook using FEniCS". If you prefer to have FEniCS installed locally on your own computer (only recommended if you plenty of time), you can follow the instructions on `http://fenicsproject.org/download/`. We would recommend using either Docker or Ubuntu.

In the following, you will be asked to run simulations for different input parameters. If you are an advanced Python programmer, you may want to make automated scripts, and systematically store the input and output values to a data file running python in a terminal. However, here, we keep the plotting to the minimal possible within a jupyter notebook. We will also for simplicity restrict our attention to flow in two dimensions.

In general, we suggest that you break your program down in small units (functions) which you can test individually. If the components themselves work, for any input, it is likely that the whole thing will work.

6.4.1 Pipe flow

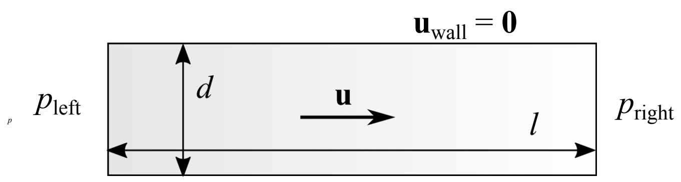


Figure 4: Pipe flow setup.

The first case concerns simulation of pipe flow. In this case we will go step-by-step through an example program. We shall consider a pipe of diameter $d = 4$ and length $l = 10$, as shown in fig. 4.

On the left side, we prescribe the pressure $p(0, y) = 1$ and on the right $p(l, y) = 0$. Along the pipe walls we use the no-slip condition $\mathbf{u}(x, 0) = \mathbf{u}(x, D) = \mathbf{0}$.

Load DOLFIN: First, we need to load all the capabilities of FEniCS using the Python module DOLFIN, and the numerical tools package Numpy. This can be done using

```
from dolfin import *
import numpy as np
```

where we have for convenience adopted the entire `dolfin` namespace.

Domain and mesh: We now have to define and discretize our domain. To create the domain, we can use the function `RectangleMesh(p0, p1, nx, ny, diagonal="right")` which draws a `nx`×`ny` rectangle between the points `p0` and `p1`.

This can e.g. be done as

```
# Define domain and discretization
delta = 0.5
length = 10.0
diameter = 4.0

# Create mesh
p0 = Point(np.array([0.0, 0.0]))
p1 = Point(np.array([length, diameter]))
nx = int(length/delta)
ny = int(diameter/delta)

mesh = RectangleMesh(p0, p1, nx, ny)
```

You can interactively visualize the mesh using

```
plot(mesh, title="Mesh")
interactive()
```

Mark the boundary: In order to apply boundary conditions, we must mark the different parts of the boundary correctly. We may first define some labels,

```
# Making a mark dictionary
# Note: the values should be UNIQUE identifiers.
mark = {"generic": 0,
        "wall": 1,
        "left": 2,
        "right": 3 }
```

Now we define a function marking the subdomains of the mesh. We first mark the entire domain as “generic”.

```
subdomains = MeshFunction("size_t", mesh, 1)
subdomains.set_all(mark["generic"])
```

The subdomain function should be marked with the correct labels on the correct parts of the mesh. Now, we define the `Left` part of the boundary, which inherits the `SubDomain` class:

```
class Left(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], 0)
```

Here, `on_boundary` is a flag indicating if a node is on the boundary, and `near(a, b)` is a function returning true if `a` and `b` are approximately equal (difference below some threshold). Note that `x[0]` refers to the first coordinate of `x`.

♠ Write the classes `Right` and `Wall` for the remaining boundaries.

Now we apply the mark defined previously for the boundaries:

```
left = Left()
left.mark(subdomains, mark["left"])
```

Now the `left` boundary has been marked.

♠ Mark the boundaries `right` and `wall` as well.

Finally, you can visualize the `subdomains` mesh function using the command

```
plot(subdomains, title="Subdomains")
interactive()
```

♠ Do the boundaries have the correct value according to the `mark` labels defined above?

Define function spaces: We now move away from the boundary for a little while. We define our function spaces, effectively the set of basis functions on our elements. To avoid stability issues which can be a problem for mixed spaces (look up the Babuska–Brezzi condition), we use Taylor–Hood elements to resolve the problem. This means that we use second-order continuous-Galerkin (CG) basis functions for the velocity and first-order CG for the pressure.

This can be implemented as follows, by first defining the elements and constructing a function space from this:

```
# Define function spaces
V = VectorElement("CG", triangle, 2)
P = FiniteElement("CG", triangle, 1)
W = FunctionSpace(mesh, V*P)
```

Here, `W` is the mixed space.

Define variational problem: We now define trial and test functions living in the space `W`:

```
# Define variational problem
(u, p) = TrialFunctions(W)
(v, q) = TestFunctions(W)
```

We also define measures for our integration (matrix assembly). We supply `subdomains` to divide the domain according to which `mark` it has (see below).

```
dx = Measure("dx", domain=mesh, subdomain_data=subdomains)   # Volume integration
ds = Measure("ds", domain=mesh, subdomain_data=subdomains)   # Surface integration
```

To assemble the surface integral in the variational form, we need the surface normal, the inlet/outlet pressures, and a (dummy) body force:

```
# Surface normal
n = FacetNormal(mesh)

# Pressures. First define the numbers (for later use):
p_left = 1.0
p_right = 0.0

# ...and then the DOLFIN constants:
pressure_left = Constant(p_left)
pressure_right = Constant(p_right)

# Body force:
force = Constant((0.0, 0.0))
```

Finally, we are ready to feed DOLFIN the variational form, cf. eqs. (141a) to (141d):

```
a = inner(grad(u), grad(v))*dx - p*div(v)*dx + q*div(u)*dx
L = inner(force, v) * dx \
    - pressure_left * inner(n, v) * ds(mark["left"]) \
    - pressure_right * inner(n, v) * ds(mark["right"])
```

Apply (Dirichlet) boundary conditions: We now return to the boundary. The no-slip (zero velocity) boundary condition can be defined as:

```
noslip = Constant((0.0, 0.0))
bc_wall = DirichletBC(W.sub(0), noslip, subdomains, mark["wall"])
```

Here, `W.sub(0)` refers to the first subspace of `W`, namely `V`. Similarly, `W.sub(1)` refers to `P`.

♠ Implement also the *pressure* boundary conditions `bc_left` and `bc_right`, using, respectively, `pressure_left` and `pressure_right`. Remember that this must be applied to the pressure subspace using `W.sub(1)`.

Finally, we collect the boundary conditions into one vector:

```
bcs = [bc_wall, bc_left, bc_right]
```

Solve variational problem: The problem is now fully specified, and all that remains is to solve it. We define the temporary solution vector `w` (corresponding to the mixed space `W`), and solve the variational problem with the supplied Dirichlet BCs `bcs` using the function `solve(problem, w, bcs)`. If `problem` is of the form `a == L`, as below, DOLFIN automatically deduces that it is a linear problem.

```
# Compute solution
w = Function(W)
solve(a == L, w, bcs)
```

Finally, we split the solution vector `w` into its velocity and pressure parts:

```
# Split using deep copy
(u, p) = w.split(True)
```

This should give you the solution for velocity and pressure, respectively through `u` and `p`.

Visualize the solution: The straightforward plotting tool is easy to use:

```
# Plot solution
plot(u, title="Velocity")
plot(p, title="Pressure")
interactive()
```

You might want to visualize the speed $|\mathbf{u}|$. It is therefore useful to implement the function `magnitude`:

```
# Magnitude function
def magnitude(vec):
    return sqrt(vec**2)
```

Then you can plot the speed by:

```
plot(magnitude(u), "Speed")
```

Tip: For more sophisticated interactive plotting, you can use ParaView[4]. Export the solution using the following commands:

```
# Save solution in VTK format
ufile = File("velocity.pvd")
ufile << u
pfile = File("pressure.pvd")
pfile << p
```

Now you can open the `.pvd`-files using ParaView.

Assess the solution: You are now asked to inspect the solution.

- ♠ How does the solution for the velocity field $\mathbf{u}$ look? What about the pressure p?
- ♠ Vary and interchange the values of `p_left` and `p_right`. How does the solution change?
- ♠ How does the numerical solution, for $\mathbf{u}$ and p, compare to the analytical solution?

One way of doing this quantitatively is the following: Construct first the analytical solution in the domain. This is done by constructing an `Expression` (which evaluates C code in the domain; therefore we have to pass variables in a special way—see code below), and interpolating it onto the `V` subspace. For both `p` and `u`:

```
# Define coefficient
coeff = (p_left-p_right)/(2*length)
# Define expressions
u_analytic = Expression(("C*x[1]*(2.0*rad - x[1])", "0.0"),
                        C=coeff, rad=0.5*diameter, degree=2)
p_analytic = Expression("p_l - 2*C*x[0]", p_l=p_left, C=coeff,
                        degree=1)
# Project onto domains
u_analytic = interpolate(u_analytic, W.sub(0).collapse())
p_analytic = interpolate(p_analytic, W.sub(1).collapse())
```

Subtract the analytical solution from the original one and take the magnitude:

```
u_error_local = magnitude(u - u_analytic)
```

Integrate (assemble) over the whole domain to get the global error:

```
u_error = assemble(u_error_local*dx)
print "u_error =", u_error
```

- ♠ How do the errors in $\mathbf{u}$ and p change as you refine the grid (i.e. change `delta`)?

Normally, we would expect that the error decreases with refined grid according to the order of the elements.

- ♠ Why may this not happen here? **Hint:** Look at the polynomial order of the function spaces `V` and `P` and of the analytical solution for $\mathbf{u}$ and p.

The divergence of the velocity field is found by

```
div_u = div(u)
div_u = project(div_u, W.sub(1).collapse()) # Project on the pressure subspace since it is scalar order
```

and it should, of course, be zero for infinite numerical precision.

♠ How does the error in $\boldsymbol{\nabla} \cdot \mathbf{u}$ change as you refine the grid?

6.5 Flow around objects

In this exercise, you will learn to calculate the drag force F_{drag} and lift force F_{lift} on arbitrarily shaped objects in an otherwise uniform creeping flow, as sketched in fig. 5.

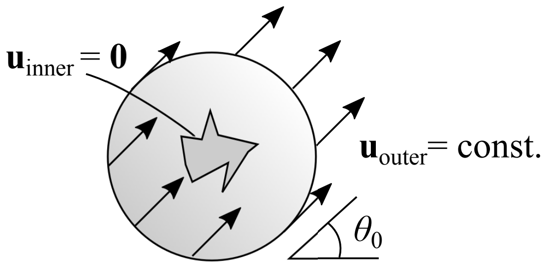

$\mathbf{u}_{\text{inner}} = \mathbf{0}$
$\mathbf{u}_{\text{outer}}$= const.
θ_0

Figure 5: Setup for flow around objects.

We will use a constant velocity condition on the outer boundary,

$$\mathbf{u}(\mathbf{x}) = U_0 \cos\theta_0 \hat{\mathbf{e}}_x + U_0 \sin\theta_0 \hat{\mathbf{e}}_y, \tag{142}$$

where θ_0 is a fixed angle. The final aim of this exercise is to calculate F_{drag} and F_{lift} as a function of rotation angle θ_0, and thereby, among other things, find the most "aerodynamic" angle in low Reynolds number flow.

♠ You can now open your favourite text editor and create a file named, e.g., `flowaround.py`.

It is a good idea to base the program on what you learned from the previous exercise, and reuse some functions where possible. For example, the function spaces, test functions and trial functions will be the same as before. In the variational problem, the only change is that `L` becomes

```
L = inner(force, v) * dx
```

since we no longer have pressure boundary conditions.

Meshes: In this case, we will supply you with meshes (or you can make your own, if you prefer and are able to).

The following meshes are provided:

- An obstacle-free mesh (http://www.nbi.dk/~mathies/free_2d.xml.gz),
- Circular obstacle (http://www.nbi.dk/~mathies/circle_2d.xml.gz),
- NACA airfoil [2] obstacle (http://www.nbi.dk/~mathies/naca_2d.xml.gz).

Download the mesh files and upload them to your ERDA folder. You can then load the pre-generated mesh by the following command:

```
mesh = Mesh("free_2d.xml.gz")
```

Mark the boundary: In this case, we have two boundaries: outer and inner boundary.

♠ Mark the boundary in a similar manner as in the previous exercise, but now with the boundary mark labels `"inner_boun"` and `"outer_boun"`.

Hint: The command

```
rad2 = x[0]**2 + x[1]**2
```

gives the squared distance from the center, and the flag `on_boundary` yields (as before) `True` if a point is on the boundary or `False` if not. The meshes we provide you with are centered at the origo. **Note:** You should not use the name `inner` on a variable, as this name is reserved for the inner product function in DOLFIN.

♠ Visualize the `subdomains` function to ensure that the boundary is correctly marked.

Apply (Dirichlet) boundary conditions: Similarly as in the previous exercise, we will apply Dirichlet boundary conditions on the velocity field. For prescribed, constant boundary velocity, this can be achieved by the commands:

```
u_outer_boun = Constant((U_0*cos(theta), U_0*sin(theta)))
bc_outer_boun = DirichletBC(W.sub(0), u_outer_boun, subdomains, mark["outer_boun"])
```

♠ Implement and apply the (no-slip) inner boundary condition as well.

To have a well-posed problem you need to fix the pressure at one node to some reference value. Take e.g. the node to the left:

```
bc_p = DirichletBC(W.sub(1), 0, "x[0] < -1.0 + DOLFIN_EPS", "pointwise")
```

Analyse the solution: You now have all the information needed to calculate the flow in any of these domains.

First, to verify that the solver is working correctly, you can run a simulation using the mesh `free_2d.xml.gz`, where the domain is free of obstacles.

♠ Does the solution match the (trivial) analytical solution?

Now, we switch to more complex geometries. The following tasks should be done for the mesh `circle_2d.xml.gz` (flow around a circle) and `naca_2d.xml.gz` (NACA airfoil).

♠ Visualize the solution for an angle of choice θ_0. Where is the pressure and velocity highest and lowest? Why?

The *second deviatoric stress invariant*, J_2, which measures how sheared the liquid is, is given by

$$J_2 = \frac{1}{2}\mathrm{tr}(\boldsymbol{\sigma}^2_{\mathrm{visc}}). \tag{143}$$

Calculation of the viscous stress tensor, total stress tensor and J_2 can be implemented as follows:

```
# Viscous stress
# sym returns the symmetric part of a matrix
stress_visc = 2*sym(grad(u))

# Total stress
stress = -p*Identity(2) + stress_visc

# Second deviatoric (viscous) stress invariant
J_2 = 0.5*tr(stress_visc*stress_visc)
```

♠ Calculate the viscous and total stress in the fluid, and visualize J_2.

Tip: If you want to visualize it in ParaView, you can export the tensor field $\boldsymbol{\sigma}$ by the following commands:

```
T = TensorFunctionSpace(mesh, "CG", 1) # Order 1, as it is a deriv. of V (order 2)
stress = project(stress, T)
stress_file = File("stress.pvd")
stress_file << stress
```

We will now calculate the traction on the inner boundary, and decompose it into a drag force and a lift force by using the direction of the imposed flow, `n_flow`. A way to do this is the following, assuming you have already calculated the boundary normal `n` and the surface measure `ds` (see previous exercise if you haven't):

```
# Imposed flow direction:
n_flow = Constant((cos(theta), sin(theta)))
n_flow = project(n_flow, W.sub(0).collapse())
```

```
# Perpendicular to n_flow:
t_flow = Constant((sin(theta), -cos(theta)))
t_flow = project(t_flow, W.sub(0).collapse())

# Compute traction vector
traction = dot(stress, n)

# Integrate decomposed traction to get total F_drag and F_lift
F_drag = assemble(dot(traction, n_flow) * ds(mark["inner"]))
F_lift = assemble(dot(traction, t_flow) * ds(mark["inner"]))
```

Now you should be able to calculate the lift force F_{lift} and drag force F_{drag} for any imposed flow angle θ_0.

- ♠ Systematically vary the imposed flow angle $\theta_0 \in [0, 2\pi]$ and plot F_{lift} and F_{drag} as functions of θ_0. Locate the extremal points.
- ♠ Do you recognize the symmetry of the geometry (and the equations) in the plotted graphs?

Extra task for the most ambitious: As you might be aware of, "unbounded" Stokes flow can not occur in 2D (see Stokes' paradox, e.g. [3]). This makes our numerical experiments only applicable to confined flows, and our calculated drag coefficients would not converge to a finite value as we increased the domain and kept the obstacle size fixed. To add some realism, you are therefore invited to include the inertia term in your calculations. The necessary information for doing so is given in the next exercise.

- ♠ Plot the drag force F_{drag} as a function of Re for different angles. How does it change with increasing Re? It has been proposed that $F_{\text{drag}} \sim \text{Re}^{-1}$ for small Re. Can you see this from your simulations?

6.6 Lid-driven cavity

You are quickly becoming an expert in FEniCS. We shall now consider one of the classic test cases in computational fluid dynamcs (CFD), namely that of lid-driven cavity flow. We consider a square geometry (cavity) with side length $l = 1$. The left, right and bottom boundaries have no-slip conditions and the top (lid) is driven with a velocity $U_0 = 1$ in the x direction. This results in a circulation in the cavity. The setup is sketched in fig. 6.

- ♠ Open your favourite text editor and create a file named, e.g., `cavity.py`.
- ♠ Create a mesh for the cavity, in the area $[0, l] \times [0, l]$. You can start with discretization resolution $N = 32$ (=`nx`=`ny`) points along each axis.
- ♠ Create the classes `Wall` and `Lid` to mark the boundary of the cavity.

In this case, we shall explore the effect of including the advection term in the solver, i.e. investigate $\text{Re} > 0$. We will still restrict our scope to seeking steady-state solutions. The equations describing

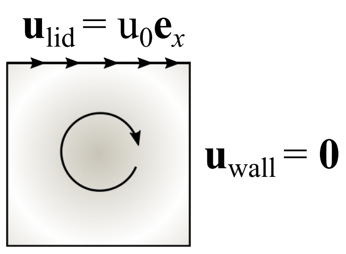

$\mathbf{u}_{\text{lid}} = \text{u}_0\mathbf{e}_x$
$\mathbf{u}_{\text{wall}} = \mathbf{0}$

Figure 6: Setup for lid-driven cavity flow.

the problem are then, disregarding the body force,

$$\nu\boldsymbol{\nabla}^2\mathbf{u} - (\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u} = \boldsymbol{\nabla}p, \tag{144a}$$

$$\boldsymbol{\nabla}\cdot\mathbf{u} = 0 \tag{144b}$$

so that the weak form of the problem becomes: Find $(\mathbf{u}, p) \in W$ such that

$$\int_\Omega (\nu\boldsymbol{\nabla}\mathbf{u} : \boldsymbol{\nabla}\mathbf{v} + (\mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u})\cdot\mathbf{v} - p\boldsymbol{\nabla}\cdot\mathbf{v} + q\boldsymbol{\nabla}\cdot\mathbf{u})\,\mathrm{d}V = 0 \quad \text{for all} \quad (\mathbf{v}, q) \in W. \tag{145}$$

The non-linear functional F describing the problem can be impemented as

```
F = inner(grad(u)*u, v)*dx \
    + nu*inner(grad(u), grad(v))*dx \
    - p*div(v)*dx \
    - q*div(u)*dx
```

Here, some variables (w, u, p, nu) have to be defined beforehand, by

```
Reynolds = 1.0  # or any other value
nu = 1./Reynolds

# Solution vectors
w = Function(W)
u, p = (as_vector((w[0], w[1])), w[2])
```

The inertia term is non-linear, so a standard linear solver will not do in solving this problem. To optimally use a non-linear solver, we supply the Jacobian with respect to the (mixed) field w that we are solving for. This is implemented as the following:

```
J = derivative(F, w)  # Jacobian
```

Then the non-linear solver can be invoked:

```
solve(F == 0, w, bcs, J=J)
```

Here, the `solve()` function automatically recognizes the form `F == 0` as a non-linear problem. Newton's method is the default non-linear solver, and works well for relatively small meshes. You can set the reference pressure BC at the bottom-left-most node by

```
bc_p = DirichletBC(W.sub(1), 0, "x[0] < DOLFIN_EPS && x[1] < DOLFIN_EPS", "pointwise")
```

Remember that you have to include `bc_p` in the `bcs` vector.

♠ Implement the above, along with the described boundary conditions, to solve the steady-state Navier–Stokes equations for lid-driven cavity flow. Visualize the resulting fields.

The most straightforward way to get the streamlines of the flow field we can place "particles" at random in the domain and passively advect them, i.e. integrate up their velocity. The script `trace.py` we provide you with does this. First, you have to export the mesh and velocity field to `.xml.gz` format.

```
u_file = File("u_Re" + str(Reynolds) + "_N" + str(N) + ".xml.gz")
mesh_file = File("mesh.xml.gz")
u_file << u
mesh_file << mesh
```

Then you can open a new terminal and run the script by:

```
python trace.py mesh.xml.gz u_Re10_N100.xml.gz trace_Re10_N100.dat -ds 0.005 -S 5 -n 100
```

This will create a data file `trace_Re10_N100.dat` with the trajectories of 100 tracers, where the integration step length is 0.005 and total path length is 5. The latter file is structured as blocks (one block per tracer) of space-separated data in the following order: tracer ID, time t, x, y, u_x, u_y, i.e. you need to plot column 3 and 4. In Gnuplot, this is straightforward:

```
pl 'cavity_data/trace_Re10_N100.dat' u 3:4 w l
```

♠ Plot streamlines of the flow for some values of $\mathrm{Re} < 1000$ (changing Re amounts to changing the viscosity ν). How does the flow change?

Tip: You may have to increase the grid resolution N to achieve convergence.

A Tensor subtleties (covered in week 2)

This section describes some of the subtleties of tensor calculus which are in general very important, but which can be "brushed under the carpet" in most of this course since we will be discussing sufficiently simple situations. In particular we can usually make do with the somewhat more restricted concept of a *Cartesian* tensor.

Contravariant and covariant tensors In a general theory, one will need to distinguish between two separate classes of indices when specifying tensors, namely the *contravariant* indices and the *covariant* ones. Contravariant indices are written as superscripts:

V^i (Components of a contravariant rank-1 tensor – a vector. Also called a (1,0) tensor)

Covariant indices, on the other hand, are written as subscripts:

ω_i (Components of a covariant rank-1 tensor – a co-vector. Also called a (0,1) tensor)

"Why the need for different types of indices?" you may ask.
Well, the short answer is "Because $|V\rangle$ and $\langle V|$ are not the same!".
The difference is probably most directly appreciated via an example. Consider the following two quantities

$$V^i := \frac{\mathrm{d}x^i}{\mathrm{d}t} = v^i, \tag{146}$$

$$\omega_i := \frac{\partial f}{\partial x^i}, \tag{147}$$

where f is some smooth function of the coordinates x^i and v^i is a velocity vector of some particle. What differs between these two quantities are their **transformation properties** – in other words how they behave under change of coordinates.
A change of coordinates is a set of new coordinates given as functions of the old coordinates:

$$x^i \to \tilde{x}^i = \tilde{x}^i(x) \tag{148}$$

One example could be a rotation around the z axis by an angle α:

$$x^i \to \tilde{x}^i = R^i{}_j x^j, \tag{149}$$

where $R^i{}_j$ is the following rotation matrix:

$$R^i{}_j = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{150}$$

The vector V^i changes in the following way under such a change of coordinates:

$$\tilde{V}^i = \frac{\mathrm{d}\tilde{x}^i}{\mathrm{d}t} = \frac{\partial \tilde{x}^i}{\partial x^j}\frac{\mathrm{d}x^j}{\mathrm{d}t} = \frac{\partial \tilde{x}^i}{\partial x^j} V^j = R^i{}_j V^j. \tag{151}$$

Whereas ω_i changes *in the opposite manner*! To see this, we simply use the chain rule:

$$\tilde{\omega}_i = \frac{\partial f}{\partial \tilde{x}^i} = \frac{\partial x^j}{\partial \tilde{x}^i}\frac{\partial f}{\partial x^j} = (R^{-1})_i{}^j \omega_j. \tag{152}$$

For a rotation matrix it holds that $R^{-1} = R^T$ so that

$$\tilde{\omega}_i = (R^T)_i^{\ j} \omega_j. \tag{153}$$

This last part was of course particular to rotations, but the more general take-home message is that contravariant and covariant tensors transform oppositely:

$$\begin{aligned} \tilde{V}^i &= \frac{\partial \tilde{x}^i}{\partial x^j} V^j, \\ \tilde{\omega}_i &= \frac{\partial x^j}{\partial \tilde{x}^i} \omega_j. \end{aligned} \tag{154}$$

There is a simple physical argument for why it has to be so. Imagine a simple transformation corresponding to a change of units. If you decide to use a 10cm measuring rod instead of a meter stick, the components of a vector such as V^i become *ten times as large*, numerically. On the other hand, a gradient such as ω_i becomes *ten times smaller*, numerically, since it represents how fast the function f changes as you move one unit in some direction.
In certain situations we can neglect to distinguish between covariant and contravariant indices. More on this later. First we shall introduce a special tensor, the metric.

The metric When working in ordinary Euclidean space or on flat surfaces, we're used to computing distances by using Pythagoras:

$$\mathrm{d}s^2 = \mathrm{d}x^2 + \mathrm{d}y^2, \tag{155}$$

where $\mathrm{d}s^2$ is the length of the shortest path from points (x, y) to $(x + \mathrm{d}x, y + \mathrm{d}y)$. Note that we're specializing to 2 dimensions here, to keep things simple. Using the Kronecker delta, δ_{ij}, this can also be written as

$$\mathrm{d}s^2 = \mathrm{d}x^i \delta_{ij} \mathrm{d}x^j, \tag{156}$$

where x^i is the vector of coordinates (x and y in this case). Recall that the Einstein summation convention is in force, so the above represents a sum over i as well as j.
However, as soon as we're working on a curved surface, such as the surface of a sphere, Pythagoras doesn't hold anymore. On a sphere we would usually prefer to use the azimuthal and polar angles (ϕ, θ) to describe things. In this case the (infinitesimal) distance should be computed as

$$\mathrm{d}s^2 = R^2 \mathrm{d}\theta^2 + R^2 \sin^2(\theta) \mathrm{d}\phi^2. \tag{157}$$

Exercise

Show that an infinitesimal line element between points (ϕ, θ) and $(\phi + \mathrm{d}\phi, \theta + \mathrm{d}\theta)$ on a sphere of radius R is given by (157).

We wish to mimic what we did above with the Kronecker delta. In other words, we wish to find a tensor g_{ij} such that the distances can be computed as

$$\mathrm{d}s^2 = \mathrm{d}x^i g_{ij} \mathrm{d}x^j, \tag{158}$$

where x^i are our chosen coordinates, namely (ϕ, θ). In that case we must choose

$$g_{\theta\theta} = R^2, \quad g_{\phi\phi} = R^2 \sin^2(\theta), \quad g_{\theta\phi} = g_{\phi\theta} = 0. \tag{159}$$

In matrix notation this reads

$$g_{ij} = \begin{bmatrix} R^2 \sin^2(\theta) & 0 \\ 0 & R^2 \end{bmatrix} \tag{160}$$

This is the metric tensor of a sphere. It essentially contains all the geometrical information about the space and – concretely – allows us to compute distances in the space.
Note that this tensor could also have been obtained from it's Euclidean counterpart, which is δ_{ij}, as described above. To do this, we just need to use the tensor transformation law. We saw in (154) how vectors and co-vectors transform. To get the transformation law for the tensor g_{ij} you just transform each of the indices according to the same law!
In other words:

$$\tilde{g}_{ij} = \frac{\partial x^k}{\partial \tilde{x}^i} \frac{\partial x^l}{\partial \tilde{x}^j} g_{kl}. \tag{161}$$

In going from Cartesian to spherical coordinates, the 'old' metric is δ_{ij}, so we write

$$\tilde{g}_{ij} = \frac{\partial x^k}{\partial \tilde{x}^i} \frac{\partial x^l}{\partial \tilde{x}^j} \delta_{kl}. \tag{162}$$

The coordinates are of course related by

$$\begin{aligned} x &= R\cos(\phi)\sin(\theta), \\ y &= R\sin(\phi)\sin(\theta), \\ z &= R\cos(\theta). \end{aligned}$$

Thus we find the relations

$$\begin{aligned}
\frac{\partial x}{\partial \phi} &= -R\sin(\phi)\sin(\theta), \\
\frac{\partial x}{\partial \theta} &= R\cos(\phi)\cos(\theta), \\
\frac{\partial y}{\partial \phi} &= R\cos(\phi)\sin(\theta), \\
\frac{\partial y}{\partial \theta} &= R\sin(\phi)\cos(\theta), \\
\frac{\partial z}{\partial \phi} &= 0, \\
\frac{\partial z}{\partial \theta} &= -R\cos(\theta).
\end{aligned}$$

So to compute a component of the spherical metric, we would simply write

$$\begin{aligned}
\tilde{g}_{\phi\phi} &= \frac{\partial x^k}{\partial \phi}\frac{\partial x^l}{\partial \phi}\delta_{kl} = \left(\frac{\partial x}{\partial \phi}\right)^2 + \left(\frac{\partial y}{\partial \phi}\right)^2 + \left(\frac{\partial z}{\partial \phi}\right)^2 \\
&= R^2\left(\sin^2(\theta)\cos^2(\phi) + \sin^2(\theta)\sin^2(\phi) + 0\right) = R^2\sin^2(\theta). \qquad (163)
\end{aligned}$$

We see that the result fits with what we found in (160).

The metric and the strain tensor Recall that in 1.4 we found the Euler Strain tensor by considering a deformation given by $x = f(X)$ where X^i are the reference coordinates and x^i are the deformed coordinates.
There we found the expression

$$dx_i dx_i - dX_i dX_i = \left(\delta_{k\ell} - \frac{\partial f_i^{-1}}{\partial x_\ell}\frac{\partial f_i^{-1}}{\partial x_k}\right) dx_\ell dx_k \qquad (164)$$

We see that the quantity in the brackets is just the difference between the undeformed and deformed metrics! So the Euler strain tensor just tells you by how much the metric has changed.

References

[1] Wikipedia: Finite element method. https://en.wikipedia.org/wiki/Finite_element_method, 2016. Online; accessed 13 March 2016.

[2] Wikipedia: NACA airfoil. https://en.wikipedia.org/wiki/NACA_airfoil, 2016. Online; accessed 13 March 2016.

[3] Wikipedia: Stokes' paradox. https://en.wikipedia.org/wiki/Stokes%27_paradox, 2016. Online; accessed 13 March 2016.

[4] Utkarsh Ayachit. The ParaView guide: A parallel visualization application. *Kitware, Inc.*, 2015.

[5] Young W. Kwon and Hyochoong Bang. *The finite element method using MATLAB*. CRC press, 2000.

[6] Hans Petter Langtangen and Anders Logg. Solving PDEs in Minutes - The FEniCS Tutorial Volume I. https://fenicsproject.org/pub/tutorial/html/ftut1.html, 2016. Online; accessed 13 March 2016.

[7] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

[8] Anders Logg, Garth N Wells, and Johan Hake. DOLFIN: A C++/Python finite element library. In *Automated Solution of Differential Equations by the Finite Element Method*, pages 173–225. Springer, 2012.