Lecture 1 - 3D linear elasticity

Introduction

Our starting point is the general equation of elasticity (not only linear!) in three dimensions:

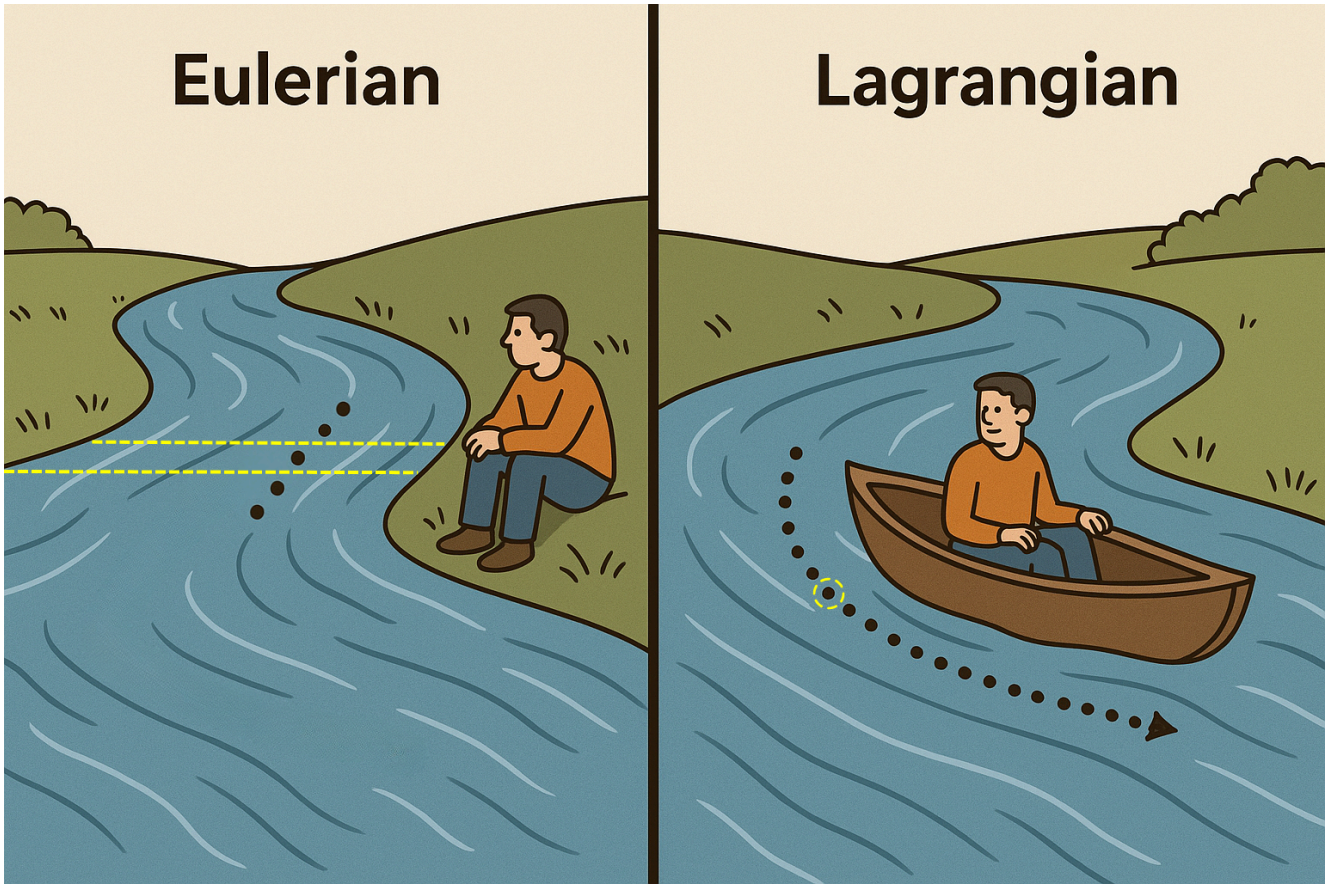
$$\boxed{\rho \frac{d^2 \mathbf{x}}{dt^2} = \mathbf{f} + \operatorname{div} \boldsymbol{\sigma}}$$

Here:

- $\rho = \rho(t, \mathbf{x})$ is the density
- $\mathbf{f} = \mathbf{f}(t, \mathbf{x})$ is a volumetric force, usually in units of N/m^3. The most common force we use is $\mathbf{f} = \rho \mathbf{g}$, with $\mathbf{g}$ the acceleration of gravity.
- $\operatorname{div} \boldsymbol{\sigma}$ is the divergence of the Cauchy stress tensor. $\boldsymbol{\sigma}$ has units of pressure, i.e. N/m^2.

This equation hides a few assumptions:

- It is an equation in the **Lagrangian** framework, meaning that it tracks how particles move over time. This is contrast with (most of) fluid mechanics, usually done in an **Eulerian** framework: we look at a specific region in space, and see how its properties change as the fluid moves around
- The divergence operator here is really a *covariant* derivative operator, so that this equation is valid under any changes of coordinates. This is not important now, but will be when we discuss shells.



Conventions and notation

We have no time to go over mechanical theory, so I will assume some familiarity with the concepts:

- What the *undeformed*, or *reference*, configuration is;

- What the *deformed*, or *current*, configuration is;
- The role of the Cauchy stress tensor.

We live in 3D Euclidean space, and are consistently using Cartesian coordinates x, y, z. In particular, the canonical orthonormal basis is $\{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$.

For this lecture, we will adopt index notation as follows:

- **Indices** $a, b, \ldots$ will be used to denote nodes and associated shape functions. Usually, they will range from 1 to the total number of nodes in an element. This choice of notation is a bit unusual, but I find that it simplifies comprehension.
- **Indices** $i, j, k, \ldots$ denote vector components in 3D space.
- **3-vectors**, in the lecture notes, will be written with **upright boldface**: $\mathbf{x}, \mathbf{y}, \mathbf{u}$ etc. In the white board, I will give preference to using arrows: $\vec{x}, \vec{u}$ etc.
- **3-tensors**, in the lecture notes, will be written in ***italic boldface***: $\boldsymbol{\sigma}, \boldsymbol{\varepsilon}$, etc. In the white board, I will give preference to using double underlines: $\underline{\underline{\sigma}}, \underline{\underline{\varepsilon}}$
- **Einstein summation notation**: a *vector* index repeated *twice* in an expression means it is being summed over, e.g.

$$a_i v_i \equiv \sum_{i=1}^{3} a_i v_i.$$

  Since the index is dummy, it can be changed to anything as long as we keep consistency:

$$a_i v_i = a_k v_k = v_j a_j, \quad \text{etc.}$$

  For indices a, b etc, we still write the summations explicitly.
  NOTE: in following lectures, when dealing with shells, this will only hold for pairs of *covariant* and *contravariant* indices. In this lecture, since we only deal with a fixed, globally Cartesian coordinate system, such distinction is irrelevant.

We will also follow common engineering notation in finite element texts. *Matrices* are written between square brackets, so $[a]$ may denote a matrix (with arbitrary size). *Vectors* are column vectors, and written between curly brackets: for example,

$$\{u\} \equiv \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \qquad \{u\}^T \equiv [u_1 \quad u_2 \quad u_3].$$

Voigt notation will be adopted for the symmetric tensors $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$. In this case, we write their 6D vectors using engineering notation, as $\{\sigma\}$ and $\{\varepsilon\}$.

Explicitly, for $\mathbf{u} = (u, v, w)$, the strain vector is defined as

$$\{\varepsilon\} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ 2\varepsilon_{xy} \\ 2\varepsilon_{yz} \\ 2\varepsilon_{xz} \end{bmatrix} = \begin{bmatrix} \partial_x u \\ \partial_y v \\ \partial_z w \\ \partial_y u + \partial_x v \\ \partial_z v + \partial_y w \\ \partial_x w + \partial_z u \end{bmatrix}$$

With the (confusing) factors of 2, their inner products are preserved between representations:

$$\boldsymbol{\sigma} : \boldsymbol{\varepsilon} \equiv \sigma_{ij}\varepsilon_{ij} = \{\sigma\} \cdot \{\varepsilon\} = \{\sigma\}^T\{\varepsilon\} = \{\varepsilon\}^T\{\sigma\}.$$

In this lecture, we only consider **small deformations**, meaning that:

- The displacement vector $\mathbf{u}$ is much smaller than the problem's characteristic length L:

$$\|\mathbf{u}\| \ll L$$

- The (dimensionless) strain tensor is small:

$$\sqrt{\varepsilon_{ij}\varepsilon_{ij}} \ll 1.$$

  For 3D metallic solids, this is the case as long as the maximum stress is below the material's initial yield stress.

For small deformations, the common practice is to model everything in the *deformed* configuration - the elastic equation is indeed for deformed coordinates x^i, as is the Cauchy stress tensor and the divergence. If we choose to deal with large deformations, it would then be advisable to do everything with respect to the *undeformed* coordinates, and consider the Piola-Kirchhoff stress tensor instead.

Calculus

We denote *partial* derivatives with respect to the deformed coordinates as any of the following:

$$u_{i,j} \equiv \partial_j u_i \equiv \frac{\partial u_i}{\partial x_j}$$

The first is common across mechanics and physics literature; the second makes the "I am a derivative!" fact more explicit, but messes the natural position of the indices; the last one is the most precise.

Derivatives with respect to time will use dot notation:

$$\dot{x} \equiv \frac{\partial x}{\partial t}, \quad \ddot{x} \equiv \frac{\partial^2 x}{\partial t^2}.$$

For integration, I will use the notations $d\mathbf{x}$ and $d\mathbf{s}$ for volume and surface measures respectively. These are the invariant integration measures, meaning that they include both the Jacobian (or metric determinant) and the differentials. As examples:

- $d\mathbf{x} = dx\, dy\, dz$ in standard Cartesian coordinates, but $d\mathbf{x} = r^2 \sin\theta\, dr\, d\theta\, d\varphi$ in spherical coordinates
- $d\mathbf{s} = R^2 \sin\theta\, d\theta\, d\varphi$ in spherical coordinates on the surface of a ball with radius R.

Weak forms

In a globally Cartesian coordinate system, we can write the problem of elasticity in coordinate notation as

$$\boxed{\rho \ddot{x}_i = f_i + \sigma_{ij,j} \quad \text{in } \Omega(t).}$$

using the fact that the divergence operator is

$$(\operatorname{div} \boldsymbol{\sigma})_i = \frac{\partial \sigma_{ij}}{\partial x_j} \equiv \sigma_{ij,j}.$$

Here, $\Omega(t)$ is the interior of a compact subset $\bar{\Omega}(t) \subset \mathbf{R}^3$. It denotes our object of interest. It evolves in time as the object deforms. The functions x_i are, really, functions of time and the reference (undeformed) coordinates, which we will denote by $\mathbf{X}$. Hence:

$$\mathbf{x} = \mathbf{x}(t, \mathbf{X}).$$

The initial conditions, for consistency, are then

$$\boxed{x_i(0, \mathbf{X}) = X_i, \quad \dot{x}_i(0, \mathbf{X}) = V_i(\mathbf{X})}$$

for some velocity field $\mathbf{V}$.

There are two types of boundary conditions to consider:

- Traction aka Neumann:

$$\boxed{\sigma_{ij} n_j = t_i} \quad (\text{ or } \quad \underline{\underline{\sigma}} \cdot \mathbf{n} = \mathbf{t} \quad \text{in index-free notation})$$

  on a patch $\Gamma_N \subseteq \partial\Omega$ where $\mathbf{n}$ is the unit normal to the surface

- Prescribed displacement aka Dirichlet:

$$\boxed{x_i = D_i} \quad (\text{ or } \quad \mathbf{x} = \mathbf{D} \quad \text{in index-free notation})$$

on a patch $\Gamma_D = \partial\Omega \backslash \Gamma_N$. Since the reference coordinate $\mathbf{X}$ for a given point $\mathbf{x}$ is fixed, we may alternatively define the Dirichlet condition in terms of displacement $u \equiv \mathbf{x} - \mathbf{X}$ as

$$u_i = U_i$$

with $U_i = D_i - X_i$.

All quantities above may depend on time.

To obtain the weak form of the equation, we take its product with a "virtual velocity" v_i (with units of speed) and integrate over $\Omega(t)$:

$$\int_{\Omega(t)} \rho \ddot{x}_i v_i d\mathbf{x} = \int_{\Omega(t)} f_i v_i d\mathbf{x} + \int_{\Omega(t)} \sigma_{ij,j} v_i d\mathbf{x}.$$

The divergence theorem yields (writing $\Omega \equiv \Omega(t)$ to avoid clutter)

$$\begin{aligned}\int_{\Omega} \sigma_{ij,j} v_i d\mathbf{x} &= \int_{\Omega} \partial_j (\sigma_{ij} v_i) d\mathbf{x} - \int_{\Omega} \sigma_{ij} \partial_j v_i d\mathbf{x} \\ &= \oint_{\partial\Omega} \sigma_{ij} v_i n_j d\mathbf{s} - \int_{\Omega} \sigma_{ij} v_{i,j} d\mathbf{x}.\end{aligned}$$

We now add the choice / assumption that $\mathbf{v}$ is zero at the Dirichlet portion of the boundary. Then, only the Neumann part survives, where conveniently $\sigma_{ij} n_j = t_i$. Finally, we notice that, by symmetry of σ_{ij},

$$\sigma_{ij} v_{i,j} = \sigma_{ij} \frac{v_{i,j} + v_{j,i}}{2} \equiv \sigma_{ij} \varepsilon_{ij}(\mathbf{v})$$

so the infinitesimal strain tensor for $\mathbf{v}$ appears naturally. The final form of the equations is then

$$\int_{\Omega} \rho \ddot{x}_i v_i d\mathbf{x} + \int_{\Omega} \sigma_{ij} \varepsilon_{ij}(\mathbf{v}) d\mathbf{x} = \int_{\Omega} f_i v_i d\mathbf{x} + \int_{\Gamma_N} t_i v_i d\mathbf{s}.$$

This can also be written in index-free notation, as

$$\int_{\Omega} \rho \ddot{\mathbf{x}} \cdot \mathbf{v} d\mathbf{x} + \int_{\Omega} \boldsymbol{\sigma} : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} d\mathbf{x} + \int_{\Gamma_N} \mathbf{t} \cdot \mathbf{v} d\mathbf{s}.$$

From now on, we shall only consider *static* problems for which $\ddot{\mathbf{x}} = 0$. The first term then vanishes. We may further change the function of interest from $\mathbf{x}$ to $\mathbf{u}$, the displacement -- this will be convenient for specifying constitutive laws. Our final statement of the weak form is that the equation must hold for all "admissible" $\mathbf{v}$:

$$\boxed{\text{Find} \quad \mathbf{u} \in U \quad \text{such that} \quad \int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} d\mathbf{x} + \int_{\Gamma_N} \mathbf{t} \cdot \mathbf{v} d\mathbf{s} \quad \text{for all } \mathbf{v} \in U_0.}$$

Notice how we made the dependence of the stress tensor on $\mathbf{u}$ explicit. We have yet to define what the spaces U, U_0 are, and what it means for a function to be admissible.

(this next part is shallow, on purpose. I am jumping through a lot of functional analysis)

Nomenclature-wise, functions in U are called *trial functions*, and those in U_0 are called *test functions*. These are common terms (e.g. the `FEniCSx` library explicitly asks you to define trial/test spaces). In many applications, they will be the same, but this is not strictly required.

The definition of admissibility depends on the problem. Here, we require test functions to be (square-) integrable and to have a first derivative in the weak sense which is also integrable: these are mathematical requirements for the problem to be well-posed, with unique solution (the interested reader is recommended to check the Lax-Milgram theorem). Furthermore, functions in this space need to satisfy the boundary condition considered above, namely to be zero at the Dirichlet portion of the boundary. The suitable Sobolev space is

$$U_0 \equiv \{\mathbf{w} \in [H^1(\Omega)]^3 : \mathbf{w}\big|_{\Gamma_D} = \mathbf{0}\}$$

where $H^1(\Omega)$ denotes the square-integrable functions with square-integrable weak derivatives. The suitable trial function space is

$$U \equiv \{\mathbf{w} \in [H^1(\Omega)]^3 : \mathbf{w}\big|_{\Gamma_D} = \mathbf{D}\}.$$

In almost all of our applications, the prescribed displacement is just zero: $\mathbf{D} = \mathbf{0}$, hence the two spaces will match. For this lesson, I will consider them to be one and the same, for simplicity.

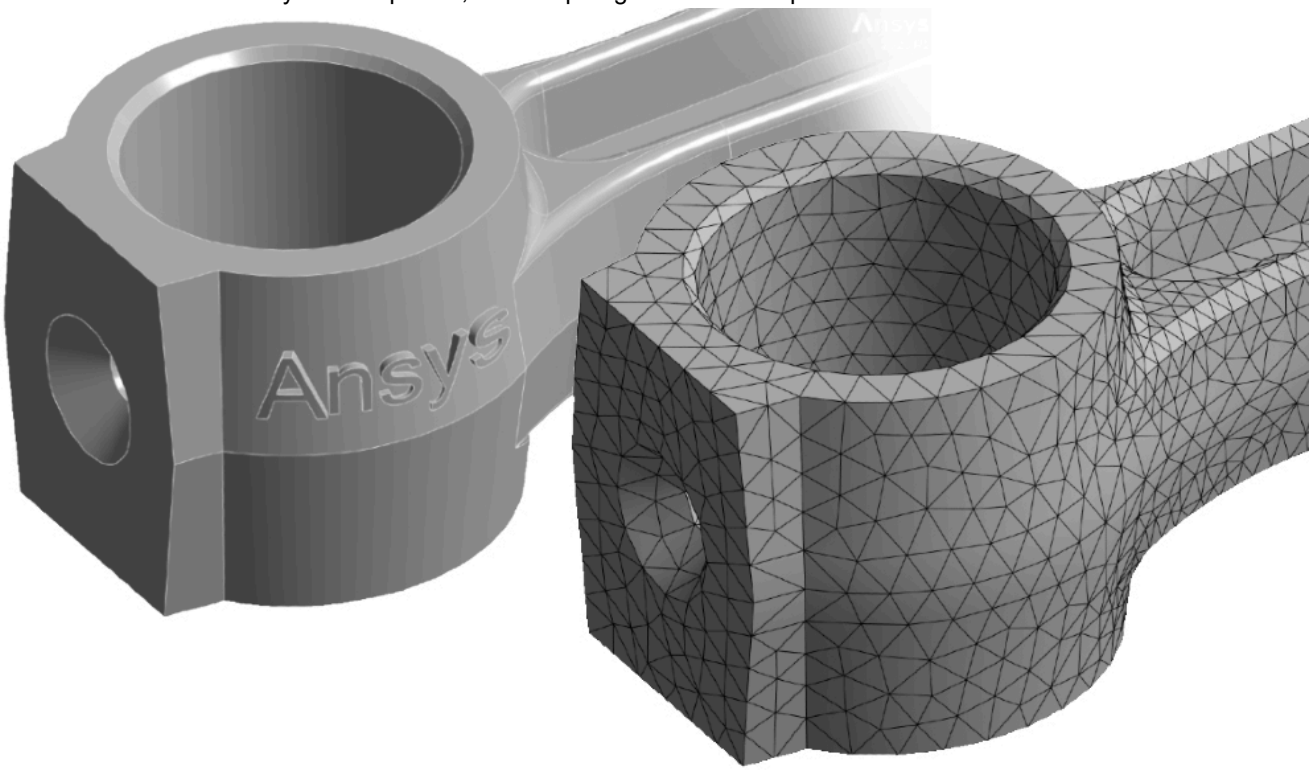
Finite subspaces & shape functions

The space U_0 has infinite dimensions, and hence cannot be represented on a computer. We then choose a finite-dimensional approximation. By definition, this means that we pick a finite function **basis** that is thought to provide a good approximation to the finite-dimensional one.

To avoid excess abstract math, I will go straight to implementation here.

We will represent our continuous domain Ω with a *mesh* composed of N **elements**: simpler geometric primitives endowed with a family of **interpolation functions**, defined in **natural coordinates**. Let us define these terms.

The notion of a mesh should be familiar to anyone working at Braid. It translates mathematically the idea of breaking up a smooth domain into many smaller pieces, with simpler geometrical shapes:

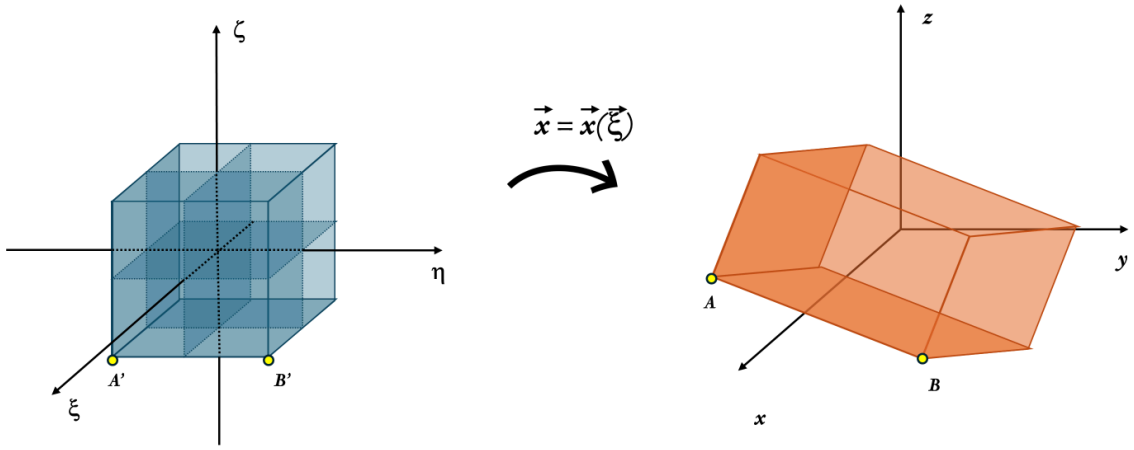


If we control the meshing process, we have access to the xyz coordinates of all mesh *nodes*.

Consider a generic "brick" (hexahedron) in 3D space (figure below on the right), which we will call K.

> In FEM literature, elements are usually called K; the more intuitive letter e is often reserved for edges. The mesh is represented with some funky notation like $\mathcal{T}_h = \bigcup_K K$.

There is a very natural way of setting coordinates ξ, η, ζ on the element. Pick one of the vertices, arbitrarily ($\mathbf{A}$), to be the one with $\xi = \eta = \zeta = -1$ (represented by $\mathbf{A}'$ in the figure below). The adjacent vertices are then mapped to coordinates where only one of the signs change: for example, vertex B will be given coordinates (-1, +1, -1) in ξ, η, ζ coordinates.



$\xi, \eta, \zeta \in [-1,1]$ K: hexahedron

Then, I claim that, with a *proper ordering of the vertices of the brick*, the following result follows: *any* point in the brick can be written as an interpolation

$$\boxed{\mathbf{x} = \mathbf{x}(\xi, \eta, \zeta) = \sum_{a=1}^{8} N_a(\xi, \eta, \zeta)\mathbf{x}_a}$$

where $\mathbf{x}_a$ are the brick's vertices. Here, we have defined a set of 8 functions $N_1, N_2, \ldots, N_8$, mapping any point in $[-1,1]^3$ into $\mathbf{R}$, as

$$\boxed{N_a(\xi, \eta, \zeta) = \frac{1}{8}(1 + \xi\xi_a)(1 + \eta\eta_a)(1 + \zeta\zeta_a),}$$

where the vertices coordinates ξ_a, η_a, ζ_a are ± 1, with the sign defined according to the table below:

a	ξ_a	η_a	ζ_a
1	-	-	-
2	+	-	-
3	+	+	-
4	-	+	-
5	-	-	+
6	+	-	+
7	+	+	+
8	-	+	+

Convince yourself that, for $\mathbf{A} = \mathbf{x}_1$, one has

$$N_1(\mathbf{A}') = 1, \quad N_b(\mathbf{A}') = 0 \ \text{ for all } \ b \neq 1$$

so that

$$\mathbf{x}(\mathbf{A}') = \sum_{a=1}^{8} N_a(\mathbf{A}')\mathbf{x}_a = \mathbf{x}_1.$$

Similarly, for $\mathbf{B} = \mathbf{x}_4$ and $\mathbf{B}' = (-1, 1, -1)$, we can show that

$$\mathbf{x}(\mathbf{B}') = \mathbf{x}_4.$$

In fact, the set of functions $\{N_a\}_{a=1}^8$ properly defines *shape functions* for representing a hexahedral element. Let us dive deeper into these below.

Curiously, there is no single, unambiguous definition of shape functions including their properties, at least that I could find. I will therefore introduce them "physics-style": give a definition, build intuition, then break the definition partly to accommodate more advanced features, and show that the original definition was just a special case.

Shape functions for position.
Consider an element K on a mesh. Assume the positions of n_K points are know, i.e. we have access to n_K vectors $\mathbf{x}_1, \dots, \mathbf{x}_{n_K}$ in the element.

Further assume there exists a *continuous bijection* $\boldsymbol{\phi} : \omega \to K$ between a compact set ω, the "natural domain", and the element K, such that every element $\mathbf{x} \in K$ can be written as $\mathbf{x} = \boldsymbol{\phi}(\boldsymbol{\xi})$ for some $\boldsymbol{\xi} \in \omega$, the "natural coordinates" of $\mathbf{x}$. Any such bijection naturally defines an *interpolation* between the sample points $\mathbf{x}_a$.

A set of n_K functions $N_a : \omega \to \mathbf{R}$ will be called **shape functions** for the interpolation $\boldsymbol{\phi}$ if the latter can be expressed as

$$\boldsymbol{\phi}(\boldsymbol{\xi}) = \sum_{a=1}^{n_K} N_a(\boldsymbol{\xi}) \mathbf{x}_a, \qquad \text{for all } \boldsymbol{\xi} \in \omega.$$

If this is a bijection, we may in particular label the pre-images of our nodes, $\boldsymbol{\xi}_a = \boldsymbol{\phi}^{-1}(\mathbf{x}_a)$. It must hold true that $\mathbf{x}_a = \boldsymbol{\phi}(\boldsymbol{\phi}^{-1}(\mathbf{x}_a))$, or

$$\boxed{\mathbf{x}_a = \sum_{b=1}^{n_K} N_b(\boldsymbol{\xi}_a) \mathbf{x}_b.}$$

> OBS: this is the most general definition of a shape function I could come up with. It is consistent with everything I have read so far.

> In the example above, of the brick element, the domain of natural coordinates is $\omega = [-1, 1]^3$. The bijection $\boldsymbol{\phi}$ itself is never explicitly written -- it is whatever can be generated with the combination of the shape functions.

A *sufficient, but not necessary* condition for the last equality to hold is that

$$\boxed{N_b(\boldsymbol{\xi}_a) = \delta_{ab} \equiv \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \text{'} \end{cases}}$$

which is called the **Kronecker delta condition**.

Shape functions for other quantities.
Now, assume that you are interested in interpolating another physical quantity -- displacement, stress, temperature etc. -- as it varies spatially over the mesh. Let us use the letter u here, which can mean either displacement or temperature.

In fact, assume that you have observations u_a on n_K nodes. Since the element itself can be mapped to natural coordinates $\boldsymbol{\xi}$, one may ask: can we interpolate u too? The answer is affirmative: the simplest such construction is in fact to write

$$u(\boldsymbol{\xi}) = \sum_{a=1}^{n_K} N_a(\boldsymbol{\xi}) u_a$$

for the same shape functions used in position (this has a fancy name: *isoparametric elements*). However, there is no uniqueness requirement here -- if we are given *another set of shape functions*, say, M_a, we may as well use them for interpolation: $u(\boldsymbol{\xi}) = \sum_a M_a(\boldsymbol{\xi}) u_a$.

Discussing properties:

- Let's discuss the **Kronecker delta** property a bit more.
  Again, if we are given position-value pairs $(\mathbf{x}_1, u_1)$, ..., $(\mathbf{x}_{n_K}, u_{n_K})$, we may consider a very general interpolation

function to be

$$\hat{u}(\boldsymbol{\xi}) = \sum_{a=1}^{n_K} N_a(\boldsymbol{\xi}) c_a,$$

where c_a's are constants which we must find. *If* the shape functions satisfy the Kronecker delta property, then the c_a 's can be found trivially: $c_a = u_a$, since

$$\hat{u}(\boldsymbol{\xi}_b) = \sum_a N_a(\boldsymbol{\xi}_b) c_a = \sum_a \delta_{ab} c_b = c_b$$

and $\hat{u}(\boldsymbol{\xi}_b) = u_b$ by definition.

If this property is not satisfied, we can still build a proper interpolation, but now one must invert the system of equations

$$\sum_a N_a(\boldsymbol{\xi}_b) c_a = u_b, \quad b = 1, \ldots, n_K.$$

This is a linear system with n_K variables and n_K equations, and hence is solvable. We see that the Kronecker delta property is the particular "Gaussian-elimination" case of this system.

- **Partition of unity**: this is the property

$$\sum_{a=1}^{n_K} N_a(\boldsymbol{\xi}) = 1, \quad \forall \boldsymbol{\xi}.$$

This property is particularly important for (a) **position shape functions** and (b) being able to represent **element-wise constant fields**.

The reason for (a) is so that we can represent *rigid body motion*. Suppose I move my whole mesh by a constant spatial displacement $\mathbf{d}$, so that $\mathbf{x}_a$ is mapped to $\mathbf{x}_a + \mathbf{d}$. Consider how this would change the interpolation:

$$\mathbf{x}_{\text{new}} = \sum_a N_a(\mathbf{x}_a + \mathbf{d}) = \sum_a N_a \mathbf{x}_a + \mathbf{d} \sum_a N_a = \mathbf{x}_{\text{old}} + \mathbf{d} \sum_a N_a.$$

If the partition of unity property is satisfied, then $\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \mathbf{d}$, i.e. the interpolation and rigid displacement commute.

Being able to represent element-wise constant fields consistently is also important. Assume our readings of a function u are all constant on the nodes, i.e. $u_a = c$ for all a. Then, we expect the interpolation to also be constant:

$$\hat{u}(\boldsymbol{\xi}) = \sum_a N_a(\boldsymbol{\xi}) c = c \left(\sum_a N_a(\boldsymbol{\xi}) \right).$$

The right-hand side will be just c for any $\boldsymbol{\xi}$ if and only if the partition of unity property is satisfied.

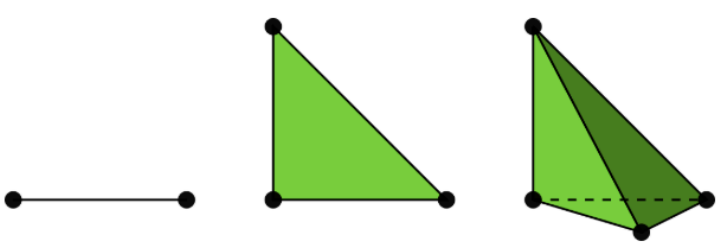
Let us illustrate these properties with actual finite elements.

Lagrange polynomials

In numerical analysis, given a set of points, a Lagrange polynomial is the lowest-order polynomial which perfectly interpolates them. Reference

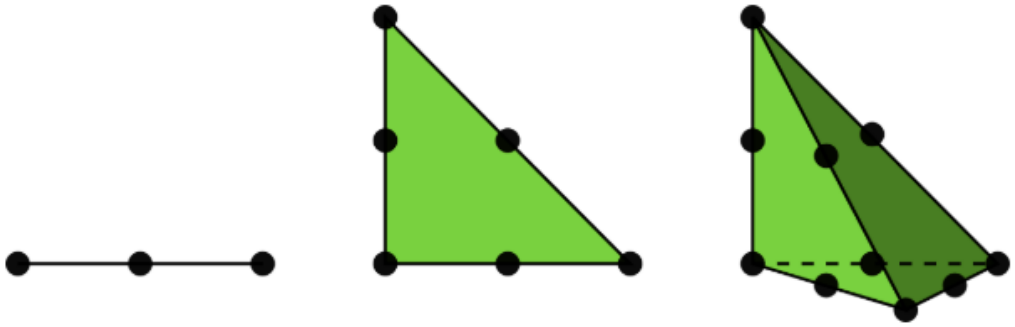
- In general, we denote by P_1 the shape functions made from linear polynomials. In n dimensions, they are linear functions defined on the n-simplex.

Figure 42: P1 elements in 1D, 2D, and 3D.

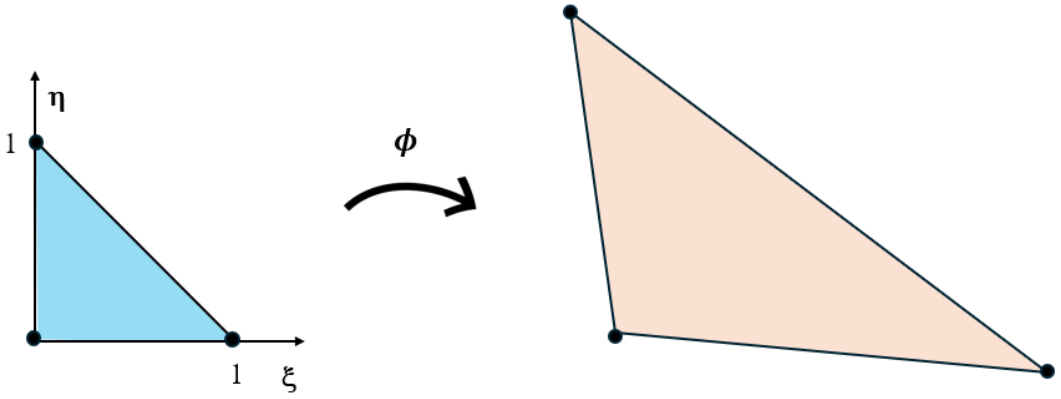


- Similarly, P_2 shape functions are defined on elements with higher number of nodes. They will be quadratic.

Figure 43: P2 elements in 1D, 2D, and 3D.



- Let us explicitly provide the expressions for Lagrange polynomials. Going to 2D for a second, any triangle can be parameterized in natural coordinates via the following set of P1 Lagrange polynomials:

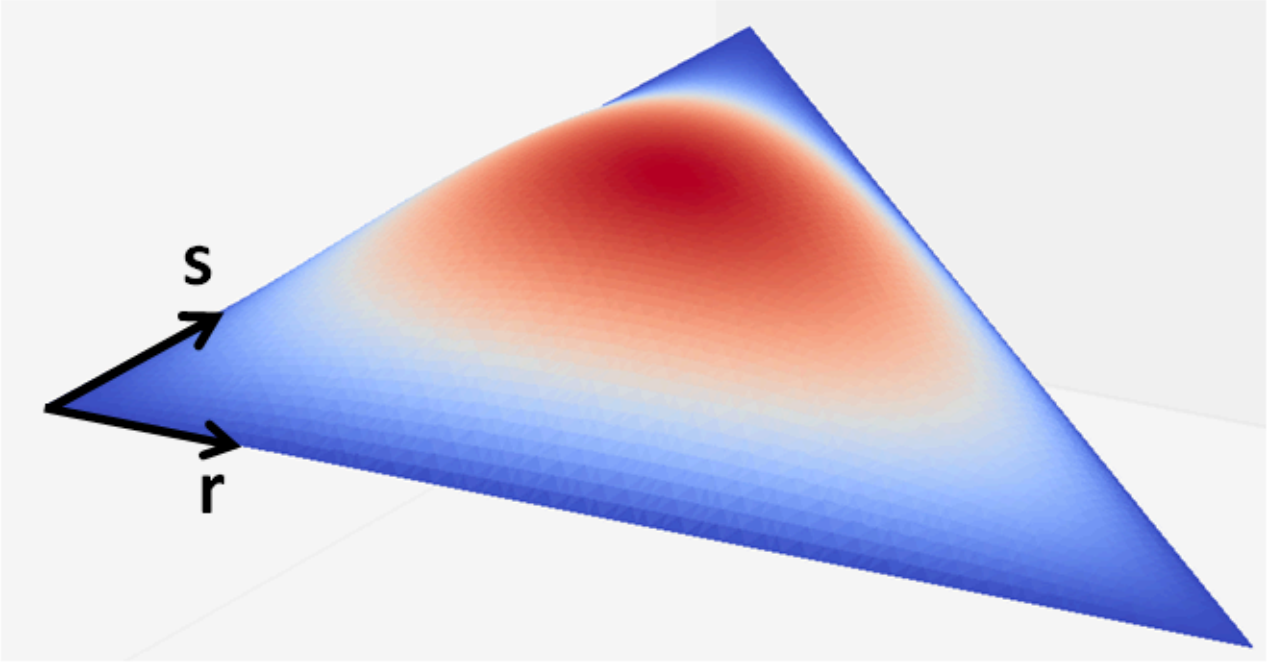


$$
\begin{aligned}
N_1(\xi,\eta) &= 1-\xi-\eta \\
N_2(\xi,\eta) &= \xi \qquad\qquad , \quad \xi,\eta \in [0,1] \\
N_3(\xi,\eta) &= \eta
\end{aligned}
$$

- Lagrange polynomials are the most basic and most used type of shape functions in FEM. They satisfy, by construction, both the Kronecker delta and partition of unity property (check it for yourself in the example above).
- However, one often *extends* a set of shape functions. An example is the so-called **bubble function** used for thin shell modeling: using the P1 functions above, the bubble is defined as

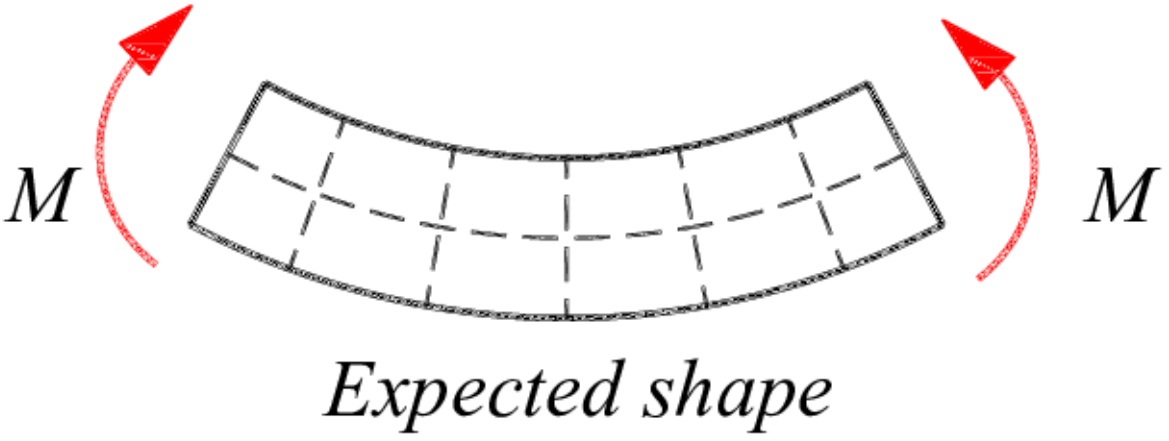
$$N_b(\xi,\eta) = 27 N_1(\xi,\eta) N_2(\xi,\eta) N_3(\xi,\eta).$$

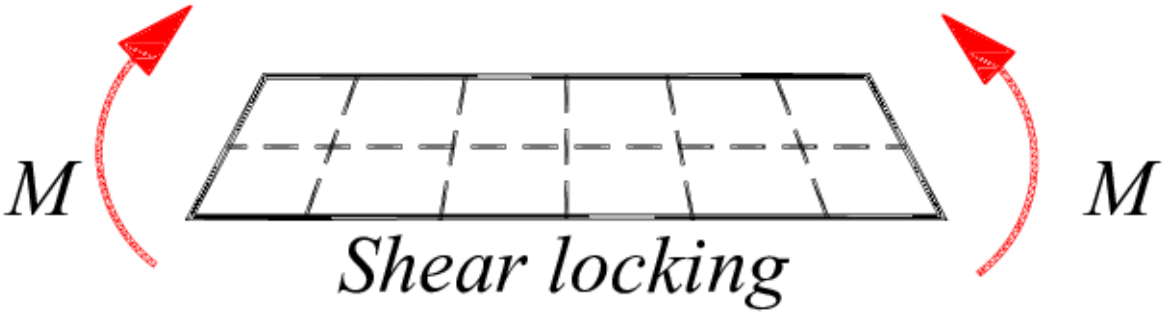
This is a 3rd order polynomial which is zero on the element boundaries, but takes a maximum at its barycenter.



Reference here

Why on Earth would one choose to add this to the set of shape functions? The idea is that the linear shape functions are too limited -- they cannot fully express complex functions on the element, and this makes them too stiff (a phenomenon called "shear locking" in beams and shells).





Adding a new non-linear function to the shape functions allows more complex behaviors to be captured. The downside is that the basis, as a whole, stops satisfying the Kronecker delta condition.

A word on integration

Suppose we have a function $f : K \to \mathbf{R}$ which we want to *integrate*, i.e. we seek

$$\int_K f(\mathbf{y}) d\mathbf{y}.$$

This is, by itself, a well-defined quantity. To compute it numerically, there are two things we can do:

1. **Use the parametrization**. By the change of variables formula, if $\mathbf{x} : \omega \to K$ is a diffeomorphism,

   $$\int_K f(\mathbf{y}) d\mathbf{y} = \int_\omega f(\mathbf{x}(\boldsymbol{\xi})) \left| \det \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \right| d\boldsymbol{\xi}$$

   where $\partial \mathbf{x} / \partial \boldsymbol{\xi}$ is the Jacobian matrix. The determinant is often simply called J. Suppose we use shape functions defined on a simple domain, like the unit cube $[-1, 1]^3$ used before. Then, this can be explicitly written as

   $$\int_K f(\mathbf{y}) d\mathbf{y} = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 f(\mathbf{x}(\xi, \eta, \zeta)) |J(\xi, \eta, \zeta)| d\xi \, d\eta \, d\zeta.$$

   In this way, I am able to use the interpolation constructed from the shape functions to bring my integrals to a simpler domain. The "price" I pay is to add the weight $|J|$ to my measure.
2. **Use Gaussian integration**. This is a common technique for performing numerical integration. The idea is that any integral can be broken down into a *weighted sum over* a few sample points in the domain. Explicitly:

   $$\int_{-1}^1 \int_{-1}^1 \int_{-1}^1 g(\xi, \eta, \zeta) d\xi \, d\eta \, d\zeta \approx \sum_{i=1}^{n_q} \sum_{j=1}^{n_q} \sum_{k=1}^{n_q} w_i w_j w_k g(\xi_i, \eta_j, \zeta_k).$$

   The number of *quadrature points*, n_q, is in principle chosen by the user - the more there are, the more precise the procedure is, but it is also costlier and, in some cases, can be less precise. Try taking a look at *reduced vs. full integration* and issues like *hourglassing* for more information.

   Finally, the specific locations of the points, ξ_i, η_j etc, as well as the associated weights w_i, can be found in tables.

Armed with these tools, any integrals we come across can be tackled. We are ready to face the FEM equations for elasticity.

Towards finite element equations

Consider the weak-form equation in the form

$$\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \, d\mathbf{x} + \int_{\Gamma_N} \mathbf{t} \cdot \mathbf{v} \, d\boldsymbol{s} \quad \forall v.$$

Now, let us break the domain -- which we make into a mesh, and still denote by Ω for simplicity -- into a union of disjoint elements. Specifically,

$$\Omega = \bigcup_{i=1}^N K_i$$

for N elements K_i. Then, by the fact that

$$\int_{A \cup B} = \int_A + \int_B \quad \text{if } A \text{ and } B \text{ are disjoint},$$

we can break down the equation as

$$\sum_{i=1}^N \int_{K_i} \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} + \sum_{i=1}^N \int_{K_i} \mathbf{f} \cdot \mathbf{v} \, d\mathbf{x} = \sum_{j=1}^S \int_{S_j} \mathbf{t} \cdot \mathbf{v} \, d\boldsymbol{s}.$$

On the right-hand side, we broke the surface term analogously to what we did for the bulk: as a sum over disjoint surface elements.

The linearity of sum allowed us to decompose a large integral into the sum of many integrals over smaller elements. We can then focus our attention on one element at a time.

> Hint #1 for finite elements: always do all your considerations at *element* level first, and then *assemble* everything together for the whole mesh.

First, for an element K with $n \equiv n_K$ nodes, consider the left-hand side of the linear elastic equation:

$$\int_K \sigma_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})d\mathbf{x}.$$

We know that our degrees of freedom here are the 3 components of $\mathbf{u}$. If we pick an isoparametric approach, where displacement shares the same shape functions as position, we can then write

$$\mathbf{u}(\boldsymbol{\xi}) = \sum_{a=1}^{n} N_a(\boldsymbol{\xi})\mathbf{u}_a$$

for $\mathbf{u}_a$ defined at each node.

In linear theory, the constitutive law relates the Cauchy stress and microscopic strain tensor via a constant 4th-order tensor

$$\sigma_{ij}(\mathbf{u}) = C_{ijkl}(\varepsilon_{kl}(\mathbf{u}) - \varepsilon^t_{kl}) + \sigma^0_{ij}$$

or, in index-free notation

$$\boxed{\boldsymbol{\sigma}(\mathbf{u}) = \mathbb{C} : (\boldsymbol{\varepsilon}(\mathbf{u}) - \boldsymbol{\varepsilon}^t) + \boldsymbol{\sigma}^0}$$

where $\boldsymbol{\sigma}^0$ denotes any pre-existing stresses, and $\boldsymbol{\varepsilon}^t = \alpha(T - T_0)\mathbf{1}$ is the free thermal strain for an isotropic conductor. For our current applications, we neglect thermal expansion and consider bodies with no pre-stress, hence the constitutive law reduces to

$$\sigma_{ij}(\mathbf{u}) = C_{ijkl}\varepsilon_{kl}(\mathbf{u}) \quad \text{with} \quad \varepsilon_{ij}(\mathbf{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

In general (not only for isotropic elasticity), C has several symmetries:

- $C_{ijkl} = C_{jikl}$ (symmetry in first indices). This corresponds to the symmetry of the stress tensor
- $C_{ijkl} = C_{ijlk}$ (symmetry in the last indices). This corresponds to the symmetry of the strain tensor.
- For elastic materials, $C_{ijkl} = C_{klij}$ (major symmetry). This corresponds to the existence of a strain-energy potential from which stress derives.

From the first two symmetries, we can, roughly speaking, think of the pair ij as a single index A and similarly for kl. Each index can take $3 \times (3+1)/2 = 6$ distinct values, and so there exists a bijection between C_{ijkl} and a 6x6 symmetric tensor C_{AB}, itself with $6 \times (6+1)/2 = 21$ independent components.

There are two paths we can choose now:

- Do everything "math-style", getting precise equations with lots of indices dancing around - in the end, we will need to convert these to matrix operations to make them efficient in a program
- Do everything "matrix-style" from this point: this makes the underlying physics less clear, but provides equations that can be easily parsed to a programming language later.

Both work, and have their pros and cons. I will go with "math-style" first; skip this if you prefer.

Index gymnastics-based approach

Write the stress as

$$\sigma_{ij}(\mathbf{u}) = C_{ijkl}\frac{\partial u_k}{\partial x_l}$$

leveraging the symmetry of C_{ijkl} in the last two indices. At the same time, we know that the k-th component of the

displacement can be written as

$$u_k(\boldsymbol{\xi}) = \sum_{a=1}^{n} N_a(\boldsymbol{\xi})(u_a)_k;$$

hence

$$\sigma_{ij}(\mathbf{u}) = \sum_{a=1}^{n} C_{ijkl}(u_a)_k \frac{\partial N_a(\boldsymbol{\xi})}{\partial x_l}.$$

As said before, we need the chain rule here:

$$\frac{\partial N_a(\boldsymbol{\xi})}{\partial x_l} = \frac{\partial \xi^m}{\partial x_l}\frac{\partial N_a(\boldsymbol{\xi})}{\partial \xi_m} \equiv (J^{-1})_{lm}\frac{\partial N_a(\boldsymbol{\xi})}{\partial \xi_m}$$

where we have identified the Jacobian matrix J and its inverse,

$$J_{ml} = \frac{\partial x_l}{\partial \xi_m}, \quad (J^{-1})_{lm} = \frac{\partial \xi_m}{\partial x_l}.$$

Hence, the stress becomes

$$\sigma_{ij}(\mathbf{u}) = \sum_{a=1}^{n} C_{ijkl}(u_a)_k (J^{-1})_{lm}\frac{\partial N_a(\boldsymbol{\xi})}{\partial \xi_m}.$$

We also need the term $\varepsilon_{ij}(\mathbf{v})$ for the weak form equation. Since it is contracted with the symmetric σ_{ij}, we can just use $\partial v_i/\partial x_j$.

Expressing the test function v_i in the same way as we did for u, namely

$$v_i(\boldsymbol{\xi}) = \sum_{b=1}^{n} N_b(\boldsymbol{\xi})(v_b)_i \quad \Rightarrow \quad \frac{\partial v_i}{\partial x_j} = \sum_{b=1}^{n}(J^{-1})_{jp}\frac{\partial N_b}{\partial \xi_p}(v_b)_i$$

we can put everything together to obtain the first term in the weak-form equation:

$$\int_K \sigma_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})d\mathbf{x} = \int_K \sum_{a=1}^{n}\sum_{b=1}^{n} C_{ijkl}(u_a)_k(v_b)_i(J^{-1})_{lm}(J^{-1})_{jp}\frac{\partial N_a(\boldsymbol{\xi})}{\partial \xi_m}\frac{\partial N_b(\boldsymbol{\xi})}{\partial \xi_p}d\mathbf{x}.$$

To clean up a bit, let us define

$$(B_a)_l := (J^{-1})_{lm}\frac{\partial N_a}{\partial \xi_m}$$

so the expression becomes

$$\int_K \sigma_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})d\mathbf{x} = \sum_{a=1}^{n}\sum_{b=1}^{n}\int_K (B_b)_j C_{ijkl}(B_a)_l(u_a)_k(v_b)_i d\mathbf{x}.$$

Finally, we need the Jacobian determinant J to write the integration measure. The result is

$$\int_K \sigma_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})d\mathbf{x} = \sum_{a=1}^{n}\sum_{b=1}^{n}\left[\int_\omega (B_b(\boldsymbol{\xi}))_j C_{ijkl}(B_a(\boldsymbol{\xi}))_l |J(\boldsymbol{\xi})| d\boldsymbol{\xi}\right](u_a)_k(v_b)_i.$$

where the integral on the right-hand side is performed over the natural coordinates of the element.

Let us call

$$\mathcal{K}_{(bi)(ak)} \equiv \int_\omega (B_b(\boldsymbol{\xi}))_j C_{ijkl}(B_a(\boldsymbol{\xi}))_l |J(\boldsymbol{\xi})| d\boldsymbol{\xi}$$

the element **stiffness** tensor. Then, we simplify the expression to the following one, making explicit the Einstein summations too:

$$\int_K \sigma_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})d\mathbf{x} = \sum_{\substack{a=1,\dots,n\\k=1,2,3}}\sum_{\substack{b=1,\dots,n\\i=1,2,3}}(v_b)_i\mathcal{K}_{(bi)(ak)}(u_a)_k$$

This is the first term of our weak-form equation. It is... a bit complicated. It has 4 sums which just really look like 2 sums over "multi-indices". In fact, we can formally define new indices A, B from the following table

A	a	k
1	1	1
2	1	2
3	1	3
4	2	1
5	2	2
6	2	3
...	...	...

(analogously for B, b and i) or, explicitly,

$$A = 3(a-1) + k, \quad B = 3(b-1) + i.$$

Based on these, we can write

$$\int_K \sigma_{ij}(\mathbf{u}) \varepsilon_{ij}(\mathbf{v}) d\mathbf{x} = \sum_A \sum_B v_B \mathcal{K}_{BA} u_A$$

which looks exactly like a matrix product $\{v\}^T [\mathcal{K}] \{u\}$ in an expanded space. In the next section, we will

Before moving on to other terms, it is best to see how we would have obtained the equivalent term from a more matrix-based approach. We will see that this heavily simplifies the math above.

Matrix-based approach
As before, we need to express the stress tensor as a function of the displacement. First, we will rewrite the equation

$$\mathbf{x} = \mathbf{x}(\boldsymbol{\xi}) = \sum_{a=1}^{n} N_a(\boldsymbol{\xi}) \mathbf{x}_a$$

as a single matrix equation. Define a vector $\{X\} \in \mathbf{R}^{3n \times 1}$, i.e. with 3n rows and 1 column, from each of the nodes:

$$\{X\} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}.$$

Define the matrix of shape functions $[N] = [N(\boldsymbol{\xi})] \in \mathbf{R}^{3 \times 3n}$ as

$$[N] = \begin{bmatrix} N_1 & & & N_2 & & & & N_n & \\ & N_1 & & & N_2 & & \cdots & & N_n \\ & & N_1 & & & N_2 & & & & N_n \end{bmatrix}$$

(all unspecified elements are zero). This matrix can also be written in block form as

$$[N] = [N_1 \mathbf{1} \quad N_2 \mathbf{1} \quad \cdots \quad N_m \mathbf{1}]$$

where $\mathbf{1}$ is the 3x3 identity matrix.

Then, convince yourself of the following identity:

$$\boxed{\mathbf{x}(\boldsymbol{\xi}) = [N(\boldsymbol{\xi})]\{X\} = \sum_{a=1}^{n} N_a(\boldsymbol{\xi}) \mathbf{x}_a.}$$

Check how this works from a matrix product point of view: $[N] \in \mathbf{R}^{3 \times 3n}$ and $\{X\} \in \mathbf{R}^{3n \times 1}$ implies that the product is a 3D vector.

We can do the exact same thing for displacements: given a list of nodal displacements $\mathbf{u}_1, \dots, \mathbf{u}_n$, we can interpolate any displacement as

$$\mathbf{u}(\boldsymbol{\xi}) = \sum_{a=1}^{n} N_a(\boldsymbol{\xi})\mathbf{u}_a = [N(\boldsymbol{\xi})]\{U\}$$

where $\{U\}$ is defined analogously to the stack $\{X\}$ of positions.

Now, we proceed to computing the strain. Notice how, formally, the strain vector $\{\varepsilon\}$ representing the tensor $\boldsymbol{\varepsilon}(\mathbf{u})$, for $\mathbf{u} = [u\ v\ w]$, can be written as

$$\{\varepsilon\} = \begin{bmatrix} \partial_x u \\ \partial_y v \\ \partial_z w \\ \partial_y u + \partial_x v \\ \partial_z v + \partial_y w \\ \partial_x w + \partial_z u \end{bmatrix} = \begin{bmatrix} \partial_x & & \\ & \partial_y & \\ & & \partial_z \\ \partial_y & \partial_x & \\ & \partial_z & \partial_y \\ \partial_z & & \partial_x \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

or simply

$$\boxed{\{\varepsilon\} = [L]\mathbf{u}}.$$

The matrix L is really an operator from differentiable, vector-valued functions into $\mathbf{R}^6$.

We can then join the two results $\mathbf{u}(\boldsymbol{\xi}) = [N(\boldsymbol{\xi})]\{U\}$ and $\{\varepsilon\} = [L]\mathbf{u}$ into a single matrix equation:

$$\boxed{\{\varepsilon\} = [B]\{U\}, \quad \text{with} \quad [B] = [L][N]}$$

The matrix $[B]$ is called the **strain-displacement** matrix (for obvious reasons). Note its explicit form as a $6 \times 3n$ matrix,

$$[B] = [B_1 \ \ B_2 \ \ \cdots \ \ B_n], \quad B_a = [L]N_a = \begin{bmatrix} \partial_x N_a & 0 & 0 \\ 0 & \partial_y N_a & 0 \\ 0 & 0 & \partial_z N_a \\ \partial_y N_a & \partial_x N_a & 0 \\ 0 & \partial_z N_a & \partial_y N_a \\ \partial_z N_a & 0 & \partial_x N_a \end{bmatrix}.$$

To compute each of these components, we will need to use the Jacobian.

Moving on to stress. The standard relation $\sigma_{ij} = C_{ijkl}\varepsilon_{kl}$ can be re-written in Voigt form as

$$\boxed{\{\sigma\} = [D]\{\varepsilon\}, \quad [D] \in \mathbf{R}^{6\times 6}.}$$

Notice how D is 6x6, translating the idea we discussed above of the unique degrees of freedom of $\mathbb{C}$ being able to fit into a symmetric 6x6 matrix. Voigt notation efficiently translates this fact into matrix algebra.

We can now substitute this into the weak form. Notice that

$$\begin{aligned} \int_K \sigma_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})d\mathbf{x} &= \int_K \{\varepsilon(\mathbf{v})\}^T\{\sigma(\mathbf{u})\}d\mathbf{x} \\ &= \int_K \big([B]\{V\}\big)^T[D]\{\varepsilon(\mathbf{u})\}d\mathbf{x} \\ &= \int_K \{V\}^T[B]^T[D][B]\{U\}d\mathbf{x} \\ &= \{V\}^T\left[\int_K [B]^T[D][B]d\mathbf{x}\right]\{U\} \\ &= \{V\}^T\left[\int_\omega [B]^T[D][B]J\,d\boldsymbol{\xi}\right]\{U\} \end{aligned}$$

We call the term

$$\boxed{[\mathcal{K}] \equiv \int_\omega [B]^T[D][B]J\,d\boldsymbol{\xi}}$$

the **stiffness matrix** for element K. Notice how all the dependence on $\boldsymbol{\xi}$ lives in the matrix

$$[B(\boldsymbol{\xi})] = [L][N(\boldsymbol{\xi})] = [LN_1(\boldsymbol{\xi}) \;\; LN_2(\boldsymbol{\xi}) \;\; \cdots LN_n(\boldsymbol{\xi})].$$

Hence, this term can be pre-computed, regardless of the values of the trial/test functions $\{U\}$ and $\{V\}$. The stiffness matrix has shape $3n \times 3n$ in 3 dimensions.

A brief summary

So far, we have the following:

- Weak-form equation of elasticity (static):

$$\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \, d\mathbf{x} + \int_{\Gamma_N} \mathbf{t} \cdot \mathbf{v} \, d\boldsymbol{s} \quad \forall v.$$

- Linear elastic constitutive law:

$$\boldsymbol{\sigma}(\mathbf{u}) = \mathbb{C} : \boldsymbol{\varepsilon}(\mathbf{u})$$

- Element stiffness (index form):

$$\int_K \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \sum_{\substack{a=1,\ldots,n \\ k=1,2,3}} \sum_{\substack{b=1,\ldots,n \\ i=1,2,3}} (v_b)_i \mathcal{K}_{(ak)(bi)} (u_a)_k$$

  with

$$(B_a)_k = \frac{\partial N_a}{\partial x_k} = (J^{-1})_{km} \frac{\partial N_a}{\partial \xi_m}$$

  and stiffness matrix

$$\mathcal{K}_{(ak)(bi)} \equiv \int_\omega (B_b(\boldsymbol{\xi}))_j C_{ijkl} (B_a(\boldsymbol{\xi}))_l |J(\boldsymbol{\xi})| d\boldsymbol{\xi}$$

- Element stiffness (matrix form):

$$\int_K \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \{V\}^T [\mathcal{K}] \{U\}$$

  where the stiffness matrix is

$$[\mathcal{K}] \equiv \int_\omega [B]^T [D] [B] J \, d\boldsymbol{\xi}.$$

  with

$$[B] = [B_1 \;\; B_2 \;\; \cdots \;\; B_n], \quad B_a = [L] N_a = \begin{bmatrix} \partial_x N_a & 0 & 0 \\ 0 & \partial_y N_a & 0 \\ 0 & 0 & \partial_z N_a \\ \partial_y N_a & \partial_x N_a & 0 \\ 0 & \partial_z N_a & \partial_y N_a \\ \partial_z N_a & 0 & \partial_x N_a \end{bmatrix}.$$

Global stiffness

> Due to the simplicity of the matrix form, we will continue with it from now on.

The *element* stiffness $\{V\}^T [\mathcal{K}] \{U\}$ that we got is only for a specific element, so strictly speaking the matrices need a label. To avoid another K appearing here, let us change notation to use e for an element, so the element stiffness will be written as

$$\int_e \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \{V^e\}^T [\mathcal{K}^e] \{U^e\}$$

In principle, we need only sum over e to get the proper left-hand side of the weak-form equation.

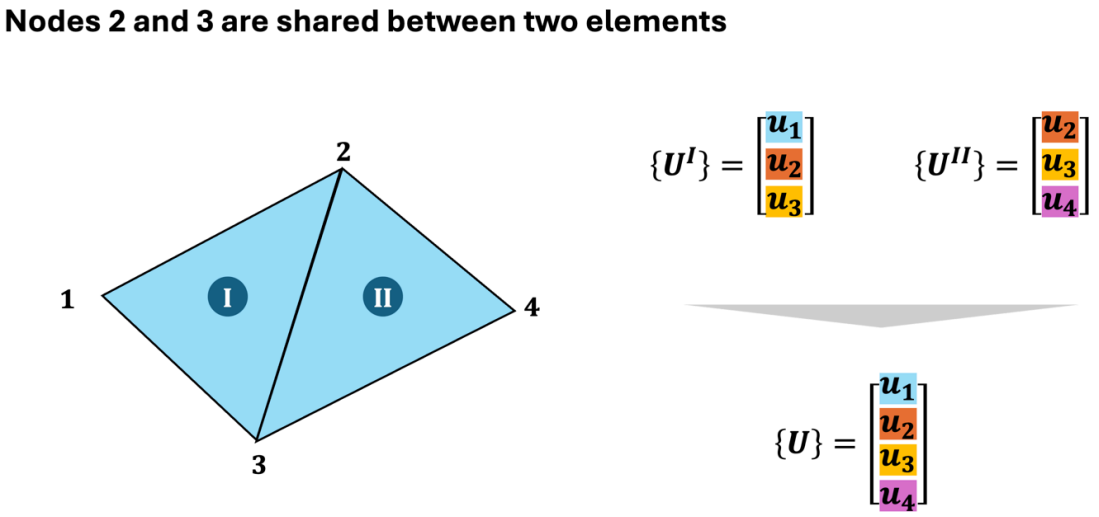
However, there is a catch: in general, a node is not part of a single element, but of many. Hence, the same unknown displacement u of a node will be present inside more than one $\{U^e\}$. To properly convert the sum of the element stiffnesses into a single matrix operation, we need a *bookkeeping* mechanism.

> Ignore the test degrees of freedom $\{V\}$ in what follows. As we will show in the end, they will not matter.

Consider the simple 2D example below, where we have two triangular elements sharing nodes labeled as 2 and 3. The displacement vector $\{U^I\}$ in element I, for example, is

$$\{U^I\} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix}$$

and similarly for $\{U^{II}\}$, except it will have the vectors for nodes 2, 3 and 4.



In 2D, the stiffness matrix has size $2n \times 2n$ for an element with n nodes, hence it is 6x6. Schematically, if we say that the stiffness matrix for the first element is $[\mathcal{K}^I] = [\mathbf{k}_1\, \mathbf{k}_2\, \mathbf{k}_3]$ where each $\mathbf{k}_a$ is a 6x2 matrix, we can write

$$[\mathcal{K}^I]\{U^I\} = \mathbf{k}_1\mathbf{u}_1 + \mathbf{k}_2\mathbf{u}_2 + \mathbf{k}_3\mathbf{u}_3$$

(the matrix products $\mathbf{k}_i\mathbf{u}_i$ have dimension 6x1, as expected). Similarly, if we say that the stiffness matrix for the second element is $[\mathcal{K}^{II}] = [\mathbf{m}_1\, \mathbf{m}_2\, \mathbf{m}_3]$, then

$$[\mathcal{K}^{II}]\{U^{II}\} = \mathbf{m}_1\mathbf{u}_2 + \mathbf{m}_2\mathbf{u}_3 + \mathbf{m}_3\mathbf{u}_4$$

Note that the degrees of freedom $\mathbf{u}_2$ and $\mathbf{u}_3$ appear in both expressions. In fact, if we *assemble* the global system, we will get

$$\sum_e [\mathcal{K}^e]\{U^e\} = \mathbf{k}_1\mathbf{u}_1 + (\mathbf{k}_2 + \mathbf{m}_1)\mathbf{u}_2 + (\mathbf{k}_3 + \mathbf{m}_2)\mathbf{u}_3 + \mathbf{m}_3\mathbf{u}_4.$$

This can be effectively transformed into a single matrix expression if we define the *global* vector of degrees of freedom

$$\{U^g\} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \mathbf{u}_4 \end{bmatrix}$$

and the global stiffness matrix

$$[\mathcal{K}^g] = [\mathbf{k}_1 \mid \mathbf{k}_2 + \mathbf{m}_1 \mid \mathbf{k}_3 + \mathbf{m}_2 \mid \mathbf{m}_3]$$

so that

$$\boxed{[\mathcal{K}^g]\{U^g\} = \sum_e [\mathcal{K}^e]\{U^e\}.}$$

This whole procedure depended on us knowing how to map each node from their element "labels" to global labels.

Reintroducing the test function. Recall that our element-wise stiffness term was $\{V^e\}^T[\mathcal{K}^e]\{U^e\}$. By the same bookkeeping procedure, we can consider a global $\{V^g\}$ at the global node indices, such that this whole term can be written as

$$\text{Global stiffness} = \{V^g\}^T[\mathcal{K}^g]\{U^g\}.$$

Here, if the mesh has N nodes, both $\{U^g\}$ and $\{V^g\}$ are vectors in $\mathbf{R}^{3N}$, and the global stiffness matrix is in $\mathbf{R}^{3N\times 3N}$.

Other terms of the equation

We now consider the right-hand side of the linear elastic equation,

$$\int_\Omega \mathbf{f}\cdot\mathbf{v}\,d\mathbf{x} + \int_{\Gamma_N} \mathbf{t}\cdot\mathbf{v}\,d\boldsymbol{s}.$$

First examine the term $\mathbf{f}\cdot\mathbf{v}$. Here, the force $\mathbf{f}(\mathbf{x})$ is *known*: we know it at every point, and do not need to interpolate it. Then, following the same procedure as above,

$$\begin{aligned}
\int_\Omega \mathbf{f}\cdot\mathbf{v}\,d\mathbf{x} &= \sum_e \int_e \mathbf{v}^T\mathbf{f}\,d\mathbf{x} \\
&= \sum_e \int_e ([N]\{V^e\})^T\mathbf{f}(\mathbf{x})d\mathbf{x} \\
&= \sum_e \{V^e\}^T \int_e [N]^T[f]J\,d\boldsymbol{\xi} \\
&\equiv \sum_e \{V^e\}^T\{F^e\}
\end{aligned}$$

where the element force vector is

$$\boxed{\{F^e\} = \int_e [N(\boldsymbol{\xi})]^T[f(\mathbf{x}(\boldsymbol{\xi}))]J(\boldsymbol{\xi})d\boldsymbol{\xi}.}$$

Of course, we can assemble the *global* force vector such that

$$\{V^g\}^T\{F^g\} = \sum_e \{V^e\}^T\{F^e\}.$$

Finally, for the boundary term, we proceed similarly but for surface elements. Notice that the shape functions themselves do not care whether they are interpolating within a surface or a 3D element, so we can use the same shape functions N and matrix $[N]$ whose size is still $3\times 3n$, with n the number of nodes of the surface element. Then

$$\begin{aligned}
\int_{\Gamma_N} \mathbf{t}\cdot\mathbf{v}\,d\boldsymbol{s} &= \sum_s \int_s \mathbf{v}^T\mathbf{t}\,d\boldsymbol{s} \\
&= \sum_s \int_s ([N]\{V^s\})^T[t(\mathbf{x})]d\boldsymbol{s} \\
&= \sum_s \{V^s\}^T \int_s [N]^T[t]Jd\boldsymbol{\xi} \\
&\equiv \sum_s \{V^s\}^T\{T^s\}, \quad \{T^s\} \equiv \int_s [N]^T[t]Jd\boldsymbol{\xi} \\
&= \{V^g\}^T\{T^g\}
\end{aligned}$$

already constructing the element-level and global tractions.

Putting everything together

We have shown that the elasticity equation

$$\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\mathbf{x} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \, d\mathbf{x} + \int_{\Gamma_N} \mathbf{t} \cdot \mathbf{v} \, d\boldsymbol{s} \quad \forall \mathbf{v}$$

can be cast to the matrix form

$$\{V^g\}^T [\mathcal{K}^g] \{U^g\} = \{V^g\}^T \{F^g\} + \{V^g\}^T \{T^g\}, \quad \forall \{V^g\}.$$

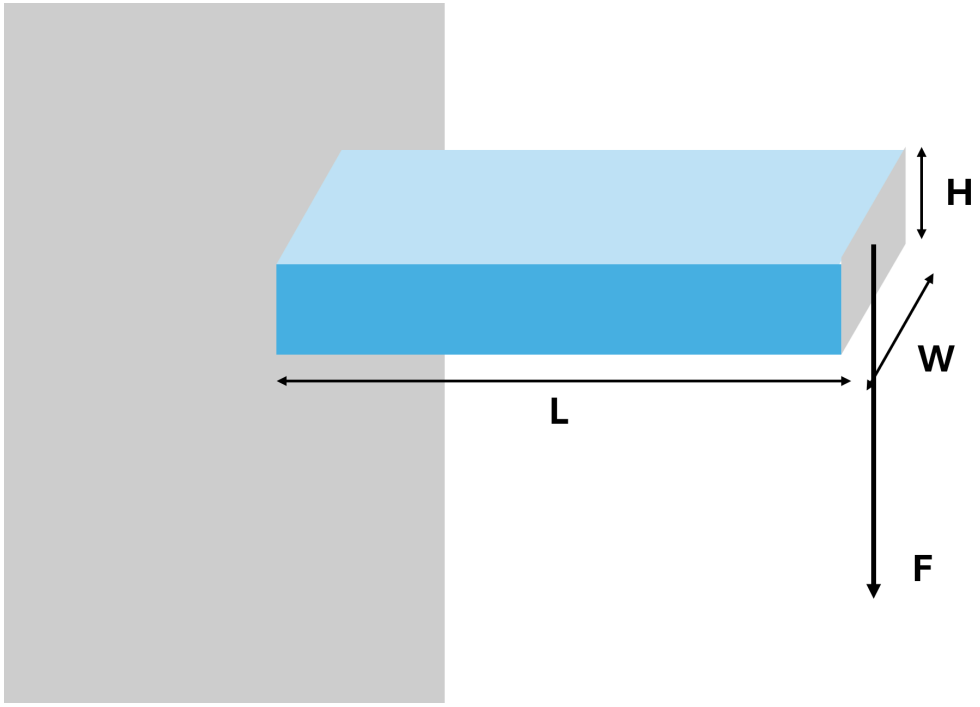
The fact that this must hold for all $\{V^g\}$ means that we must have

$$\boxed{[\mathcal{K}^g]\{U^g\} = \{F^g\} + \{T^g\}}$$

This is a system of equations with $3N$ unknowns and $3N$ equations.

Worked example

Consider an aluminum cantilever with dimensions $L = 100$, $W = 50$ and $H = 5$ (in millimeters).



We will model this with a **single hexahedral element**, using the family of shape functions introduced in the beginning:

$$N_a(\xi, \eta, \zeta) = \frac{1}{8}(1 + \xi\xi_a)(1 + \eta\eta_a)(1 + \zeta\zeta_a).$$

We want to emulate doing things by hand. The simplest way is to use the Sympy library, which does symbolic math for us and allows to convert results to Numpy later for actual numerical calculations.

Let us summarize the procedure:

- We want to solve for

$$[\mathcal{K}^g]\{U^g\} = \{F^g\} + \{T^g\}$$

- Since, in this case, we only have one element, it is easier - no global assembly necessary.
- Now,

$$[\mathcal{K}] \equiv \int_\omega [B]^T [D] [B] J \, d\boldsymbol{\xi}$$

  so we need:

1. The matrix $[D]$ containing the material properties
2. The Jacobian determinant J between natural and physical coordinates -- hence we need the mapping $\boldsymbol{\xi} \to \mathbf{x}(\boldsymbol{\xi})$
3. The matrices $[LN_a]$ which contain the partials $\partial N_a / \partial x_i$ - for this, we will again need the mapping $\boldsymbol{\xi} \to \mathbf{x}(\boldsymbol{\xi})$ and the Jacobian matrix
4. Once the integrand is in place, we need the Gaussian quadrature positions and weights.

For this case, the map between natural and physical coordinates is very easy:

$$x = \frac{L}{2}(\xi + 1), \quad y = \frac{W}{2}(\eta + 1), \quad z = \frac{H}{2}(\zeta + 1).$$

The Jacobian will then be trivial,

$$\frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} = \begin{bmatrix} L/2 & 0 & 0 \\ 0 & W/2 & 0 \\ 0 & 0 & H/2 \end{bmatrix}$$

```
import sympy as sp
import numpy as np
import pyvista as pv
import matplotlib.pyplot as plt
```

Setting up problem parameters:

```
# Cuboid dimensions
L, W, H = sp.Integer(100.0), sp.Integer(50.0), sp.Integer(5.0)

# Material properties (Young's modulus, Poisson's ratio)
E_val = 210e9
v_val = 0.3
```

Setting up variables:

```
ξ, η, ζ = sp.symbols('ξ η ζ')
E, v = sp.symbols('E v')
```

Constructing the $[N]$ matrix:

```
# all combinations of ±1 for (ξ_a, η_a, ζ_a)

nodes_natural_coordinates = [
    [-1, -1, -1],
    [ 1, -1, -1],
    [ 1,  1, -1],
    [-1,  1, -1],
    [-1, -1,  1],
    [ 1, -1,  1],
    [ 1,  1,  1],
    [-1,  1,  1],
]

N = [(1 + ξ*ξa)*(1 + η*ηa)*(1 + ζ*ζa)/sp.Integer(8) for (ξa, ηa, ζa) in nodes_natural_coordinates]

N
```

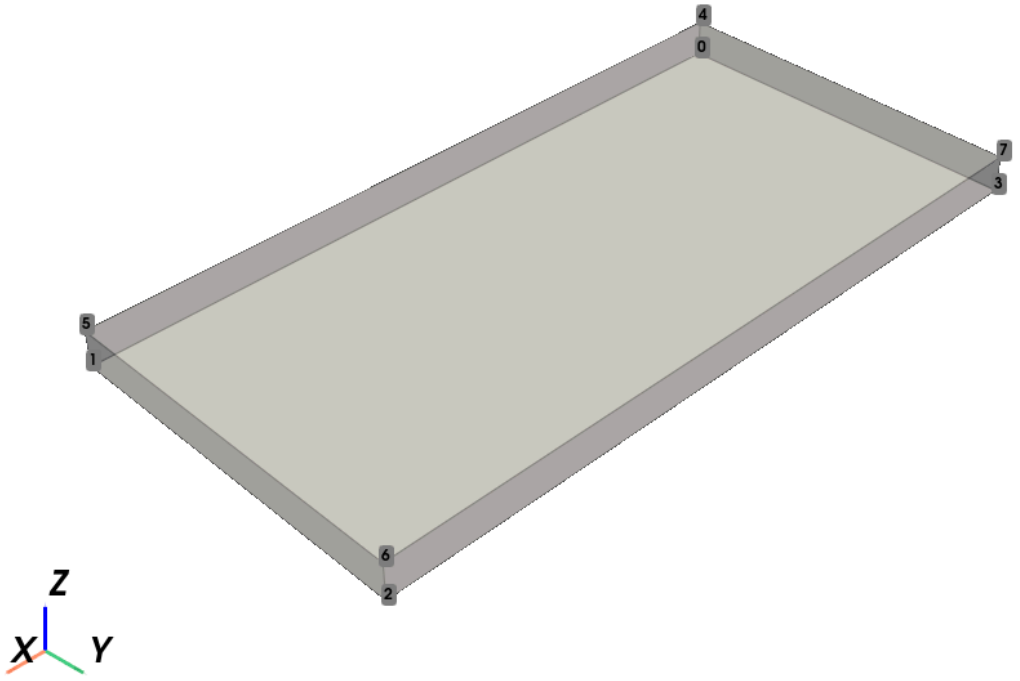
```
[(1 - ζ)*(1 - η)*(1 - ξ)/8,
 (1 - ζ)*(1 - η)*(ξ + 1)/8,
 (1 - ζ)*(η + 1)*(ξ + 1)/8,
 (1 - ζ)*(1 - ξ)*(η + 1)/8,
 (1 - η)*(1 - ξ)*(ζ + 1)/8,
 (1 - η)*(ζ + 1)*(ξ + 1)/8,
 (ζ + 1)*(η + 1)*(ξ + 1)/8,
 (1 - ξ)*(ζ + 1)*(η + 1)/8]
```

Now, we can map to global (physical) coordinates:

```
# Map from natural to physical coordinates
x, y, z = L/2 * (ξ + 1), W/2 * (η + 1), H/2 * (ζ + 1)
```

Create a mesh, and show the node labels

```
# Create a pv.UnstructuredGrid from nodes_natural_coordinates and the functions x, y, z
# Use a single hex element connectivity
nodes = np.array([[x.subs({ξ: ξa, η: ηa, ζ: ζa}).evalf(),
                   y.subs({ξ: ξa, η: ηa, ζ: ζa}).evalf(),
                   z.subs({ξ: ξa, η: ηa, ζ: ζa}).evalf()] for (ξa, ηa, ζa) in
nodes_natural_coordinates], dtype=np.float64)

cells = np.hstack([[8], np.arange(8)])  # 8 nodes per hexahedron
cell_types = np.array([pv.CellType.HEXAHEDRON])
mesh = pv.UnstructuredGrid(cells, cell_types, nodes)

# Point labels
points = mesh.points  # shape (n_nodes, 3)
labels = [str(i) for i in range(points.shape[0])]

# Plot
pl = pv.Plotter()
pl.add_mesh(mesh, show_edges=True, color='lightgray', opacity=0.5)
pl.add_point_labels(points, labels,
                    point_size=5,       # size of the point glyph
                    font_size=12,       # font size of the labels
                    text_color='black',
                    italic=False, bold=True,
                    always_visible=True)

pl.add_axes(line_width=3,
            cone_radius=0.06,
            shaft_length=0.9,
            tip_length=0.3,
            label_size=(0.5, 0.2))
pl.show()
```



The nodes 0, 3, 4 and 7 are touching the wall.

Let's find the Jacobian:

```
# find the Jacobian matrix d(x,y,z)/d(ξ,η,ζ)
J = sp.Matrix([[sp.diff(x, ξ), sp.diff(x, η), sp.diff(x, ζ)],
               [sp.diff(y, ξ), sp.diff(y, η), sp.diff(y, ζ)],
               [sp.diff(z, ξ), sp.diff(z, η), sp.diff(z, ζ)]])
J
```

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & \frac{5}{2} \end{bmatrix}$$

and its inverse:

```
J_inv = J.inv()
J_inv
```

$$\begin{bmatrix} \frac{1}{50} & 0 & 0 \\ 0 & \frac{1}{25} & 0 \\ 0 & 0 & \frac{2}{5} \end{bmatrix}$$

We will also need the derivatives of N_a, both with respect to the $\boldsymbol{\xi}$ coordinates but also (via the Jacobian) with respect to the global coordinates:

```
# dN_i/ dx = (dN_i / dξ) * (dξ / dx), etc.
dN_dξ = [sp.diff(Na, ξ) for Na in N]
dN_dη = [sp.diff(Na, η) for Na in N]
dN_dζ = [sp.diff(Na, ζ) for Na in N]

scale_x = 2 / L
scale_y = 2 / W
scale_z = 2 / H

dN_dx = [scale_x * dN_dξ[i] for i in range(8)]
dN_dy = [scale_y * dN_dη[i] for i in range(8)]
dN_dz = [scale_z * dN_dζ[i] for i in range(8)]
dN_dx  # entries are dN_i/dx
```

```
[-(1 - ζ)*(1 - η)/400,
 (1 - ζ)*(1 - η)/400,
 (1 - ζ)*(η + 1)/400,
 -(1 - ζ)*(η + 1)/400,
 -(1 - η)*(ζ + 1)/400,
 (1 - η)*(ζ + 1)/400,
 (ζ + 1)*(η + 1)/400,
 -(ζ + 1)*(η + 1)/400]
```

Moving on, we build $[B]$:

```
# Build B matrix as function of (ξ,η,ζ)
def B_sym():
    B = sp.zeros(6, 3*8)
    for a in range(8):
        i = 3*a
        B[0, i+0] = dN_dx[a]
        B[1, i+1] = dN_dy[a]
        B[2, i+2] = dN_dz[a]
        B[3, i+0] = dN_dy[a]
        B[3, i+1] = dN_dx[a]
        B[4, i+1] = dN_dz[a]
        B[4, i+2] = dN_dy[a]
        B[5, i+0] = dN_dz[a]
        B[5, i+2] = dN_dx[a]
    return B
B_sym_expr = B_sym()
B_sym_expr
```

$$\begin{bmatrix} -\frac{(1-\zeta)(1-\eta)}{400} & 0 & 0 & \frac{(1-\zeta)(1-\eta)}{400} & 0 & 0 & \frac{(1-\zeta)(\eta+1)}{400} & 0 & 0 & -\frac{(1-\zeta)(\eta+1)}{400} \\ 0 & -\frac{(1-\zeta)(1-\xi)}{200} & 0 & 0 & -\frac{(1-\zeta)(\xi+1)}{200} & 0 & 0 & \frac{(1-\zeta)(\xi+1)}{200} & 0 & 0 \\ 0 & 0 & -\frac{(1-\eta)(1-\xi)}{20} & 0 & 0 & -\frac{(1-\eta)(\xi+1)}{20} & 0 & 0 & -\frac{(\eta+1)(\xi+1)}{20} & 0 \\ -\frac{(1-\zeta)(1-\xi)}{200} & -\frac{(1-\zeta)(1-\eta)}{400} & 0 & -\frac{(1-\zeta)(\xi+1)}{200} & \frac{(1-\zeta)(1-\eta)}{400} & 0 & \frac{(1-\zeta)(\xi+1)}{200} & \frac{(1-\zeta)(\eta+1)}{400} & 0 & \frac{(1-\zeta)(1-\xi)}{200} \\ 0 & -\frac{(1-\eta)(1-\xi)}{20} & -\frac{(1-\zeta)(1-\xi)}{200} & 0 & -\frac{(1-\eta)(\xi+1)}{20} & -\frac{(1-\zeta)(\xi+1)}{200} & 0 & -\frac{(\eta+1)(\xi+1)}{20} & \frac{(1-\zeta)(\xi+1)}{200} & 0 \\ -\frac{(1-\eta)(1-\xi)}{20} & 0 & -\frac{(1-\zeta)(1-\eta)}{400} & -\frac{(1-\eta)(\xi+1)}{20} & 0 & \frac{(1-\zeta)(1-\eta)}{400} & -\frac{(\eta+1)(\xi+1)}{20} & 0 & \frac{(1-\zeta)(\eta+1)}{400} & -\frac{(1-\xi)(\eta+1)}{20} \end{bmatrix}$$

Of course, it doesn't fit the screen.

We now define the matrix $[D]$ which encodes the constitutive law:

```
# 4) define elasticity matrix D (isotropic, e.g. E=1, nu=0.3 for simplicity)
lam = E*v / ((1+v)*(1-2*v))
mu  = E / (2*(1+v))
D = sp.zeros(6,6)

for i in range(3):
```

```
    D[i,i] = lam + 2*mu
    for j in range(3):
        if i!=j:
            D[i,j] = lam

for i in range(3,6):
    D[i,i] = mu
D
```

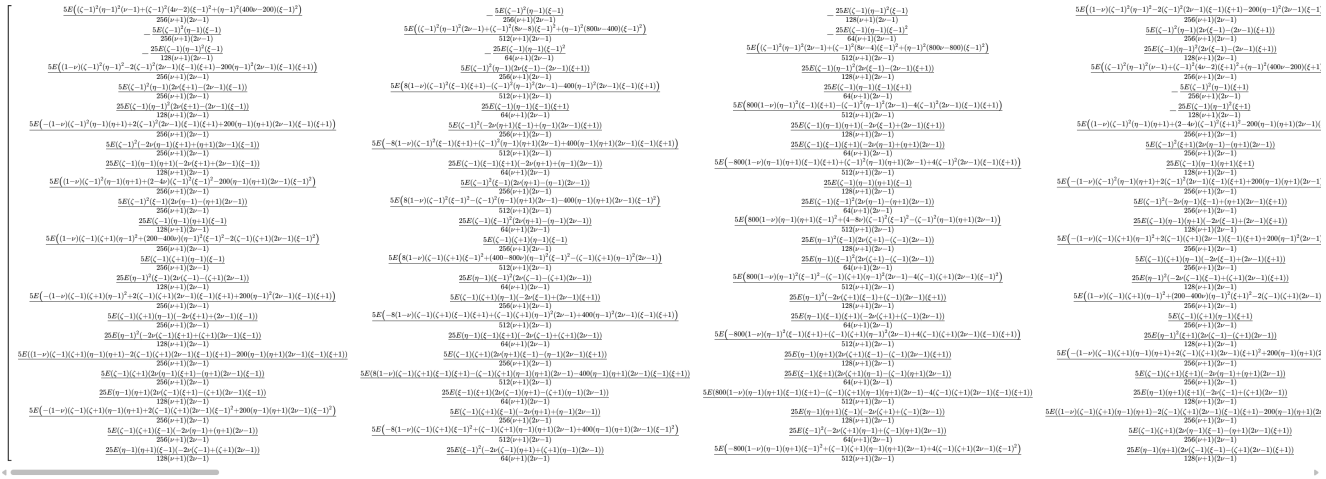
$$\begin{bmatrix} \frac{E\nu}{(1-2\nu)(\nu+1)}+\frac{2E}{2\nu+2} & \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)} & 0 & 0 & 0 \\ \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)}+\frac{2E}{2\nu+2} & \frac{E\nu}{(1-2\nu)(\nu+1)} & 0 & 0 & 0 \\ \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)}+\frac{2E}{2\nu+2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{E}{2\nu+2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{E}{2\nu+2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{E}{2\nu+2} \end{bmatrix}$$

We have almost all the elements to build the stiffness integrand $[B]^T[D][B]J$, all that is left is the determinant:

```
# 5) form integrand for stiffness: integrand = B^T D B * detJ --> I am adding detJ here for convenience
detJ = sp.det(J)
detJ # outputs 3125
```

Build the integrand:

```
# Build integrand
K_sym = (B_sym_expr.T * D * B_sym_expr) * detJ
K_sym.simplify()
K_sym # 24 x 24 symbolic stiffness matrix
```



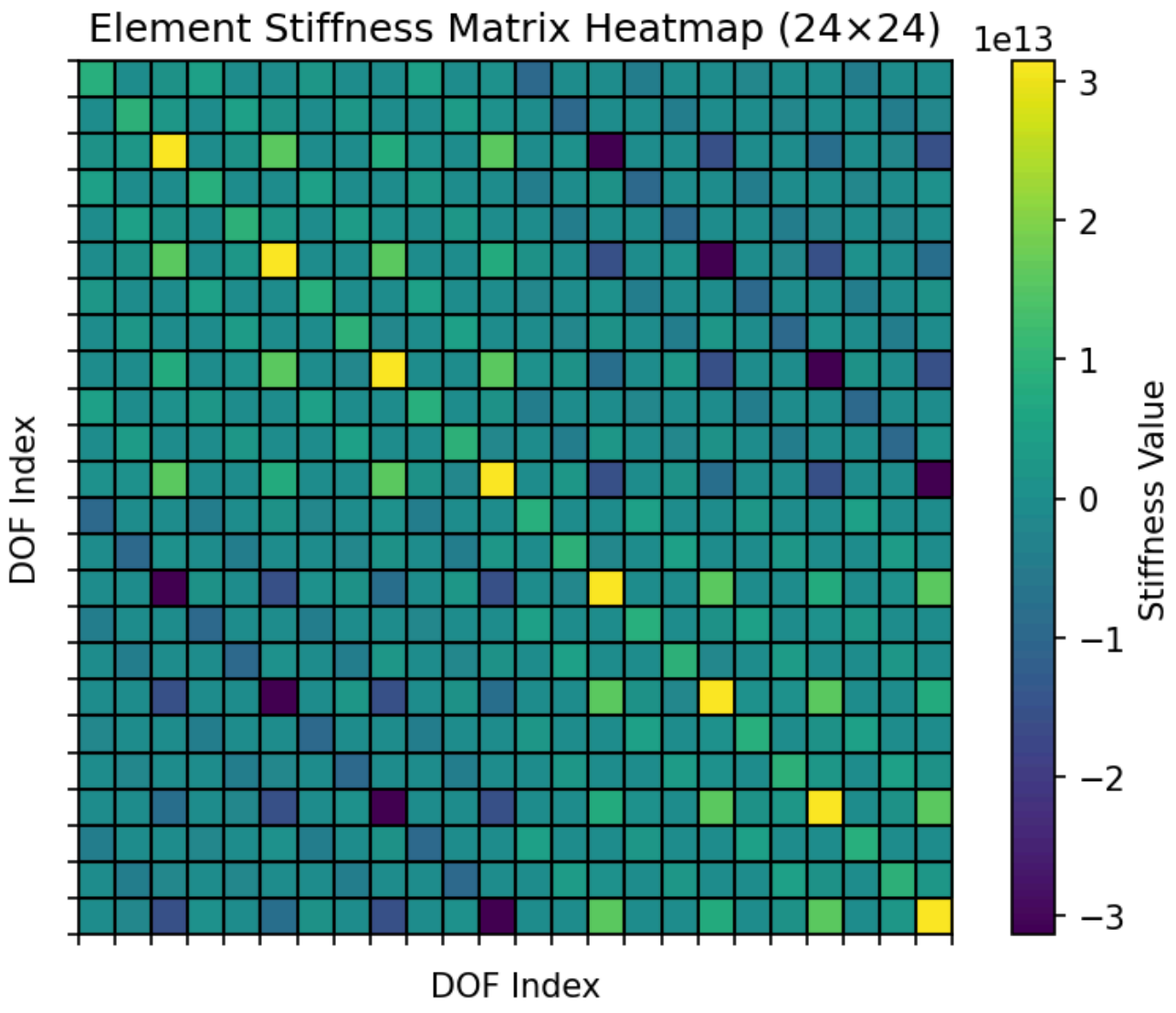
Thank God we are not doing this by hand.

Now, we want to integrate. We need to actually make this monstrosity into a usable function taking values in `numpy` arrays:

```
# Replace E and nu with numeric values
K_sym = K_sym.subs({E: E_val, v: v_val})

# 6) lambdify K integrand for numeric evaluation
f_K = sp.lambdify((ξ,η,ζ), K_sym, 'numpy')
```

We are ready to integrate. Let me show you how stiffness matrix looks like once integrated:

```python
def integrate_K(n_gauss=2):

    # get Gauss points and weights for 1D
    if n_gauss == 1:
        points = [0.0]
        weights = [2.0]
    elif n_gauss == 2:
        p = 1/np.sqrt(3)
        points = [-p, p]
        weights = [1.0, 1.0]
    else:
        raise ValueError("extend as needed")

    K_e = np.zeros((24,24))
    for ξi, wξ in zip(points, weights):
        for ηj, wη in zip(points, weights):
            for ζk, wζ in zip(points, weights):
                K_e += f_K(ξi, ηj, ζk) * (wξ*wη*wζ)
    return K_e

# Example: compute for E=210e9, nu=0.3 with 2×2×2 Gauss
K_num = integrate_K(n_gauss=2)
plot_matrix_heatmap(K_num)
```



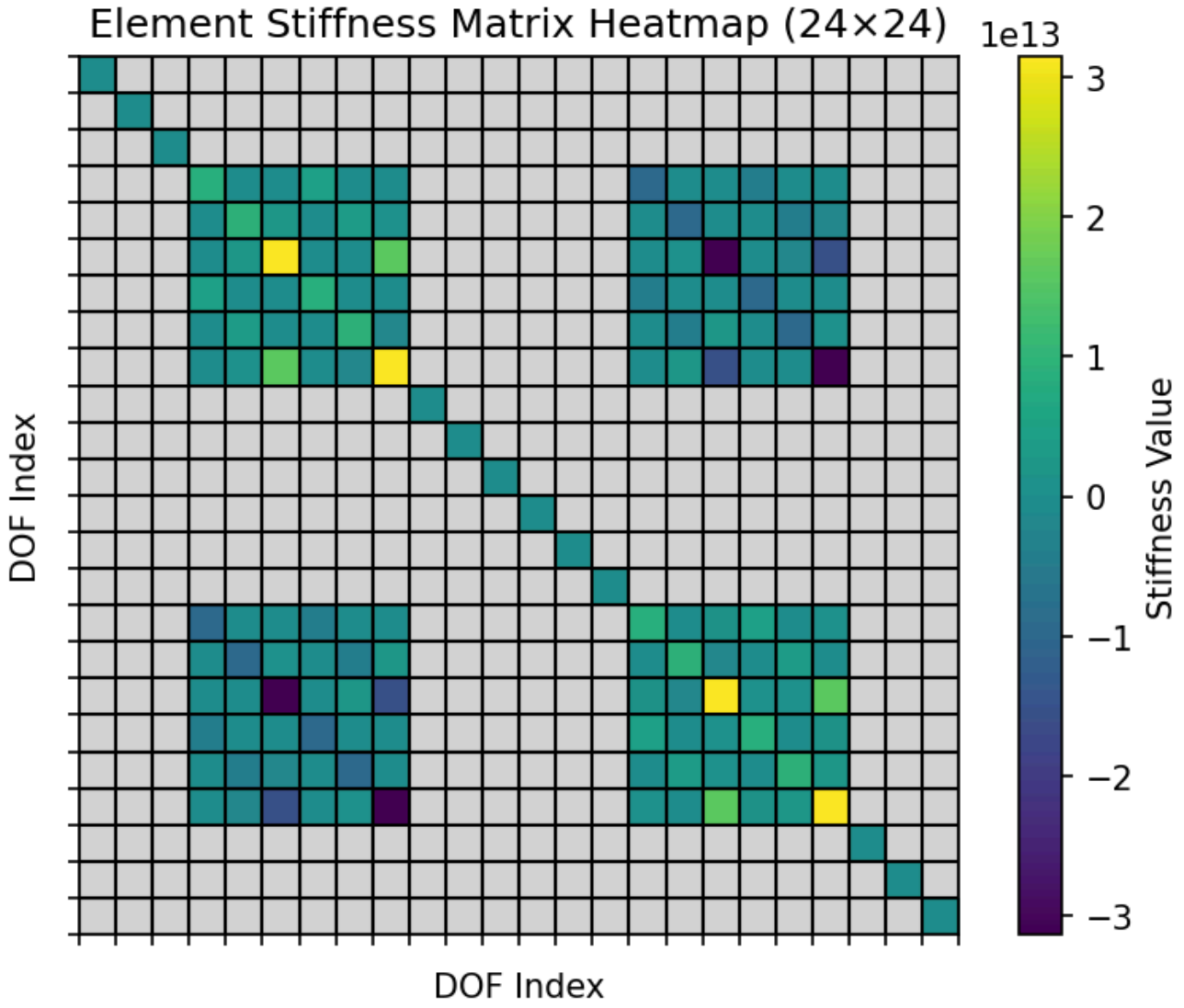
Notice how it is sparse.

Now, we build the loads vector:

```
# Construct the loads vector as needed

loads_vec = np.zeros((24, 1))  # WIP: actually make the surface integral
# e.g., apply a force of 1000 N in the z-direction at nodes 1 and 2 (counting from 0);
# since each node has 3 DOFs, the z-direction DOF for node i is at index 3*i + 2

loads_vec[5, 0] = 10000.0  # Node 1
loads_vec[8, 0] = 10000.0  # Node 2
```

Finally, we add Dirichlet boundary conditions:

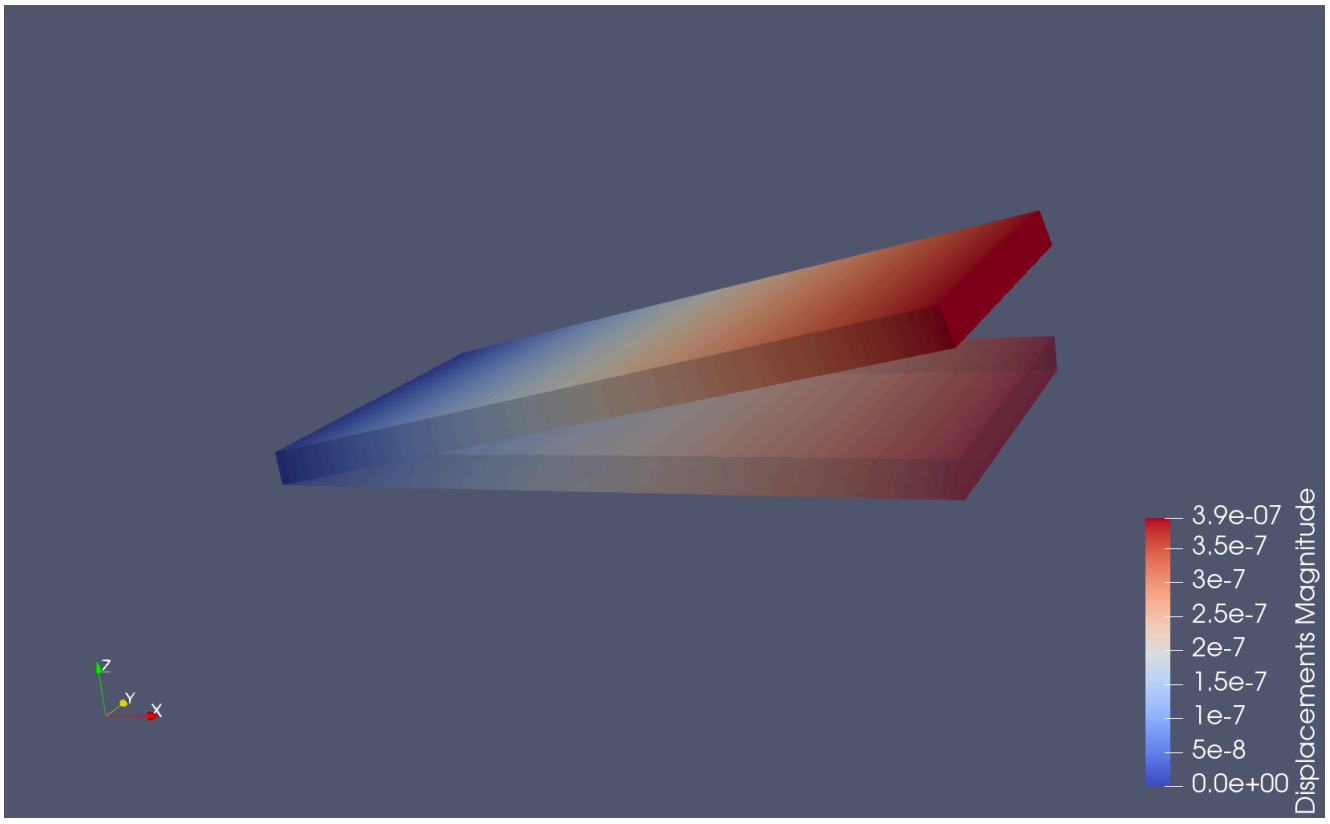
```
# Add boundary conditions
# Suppose node numbering 0-7 internally; boundary_nodes = [0, 3, 4, 7]  # (1,4,5,8 in 1-based)
boundary_nodes = [0, 3, 4, 7]

# Convert to DOF indices
fixed_dofs = []

for n in boundary_nodes:
    fixed_dofs += [3*n + 0, 3*n + 1, 3*n + 2]  # x,y,z DOFs

for dof in fixed_dofs:
    # zero out row and column
    K_num[dof, :] = 0
    K_num[:, dof] = 0

    # set diagonal to 1
    K_num[dof, dof] = 1

    # zero force
    loads_vec[dof] = 0

plot_matrix_heatmap(K_num)
```



Finally, solve the equations and visualize the results:

```
# Solve for displacements: K u = f
displacements = np.linalg.solve(K_num, loads_vec)

# Add displacements to the pyvista grid for visualization
mesh.point_data['Displacements'] = displacements.reshape(-1, 3)
mesh.save('deformed_mesh.vtk')
```

Opening in Paraview (warping by 10^7):



References

O.C. Zienkiewicz, R.L. Taylor and J.Z. Zhu, *The Finite Element Method: Its Basis and Fundamentals,* Sixth edition. Elsevier (2005)

This book is the standard bible on the field. It pretty much covers everything.

Allan F. Bower, *Applied Mechanics of Solids*.
https://solidmechanics.org/contents.php

Best theory book containing full details of FEM and shell theory using differential geometry.

Carlos A. Felippa, *Introduction to Finite Element Methods*. University of Colorado Boulder, Colorado 80309-0429
https://dynadata.com/ITVER/Documentos/FEM%20Book.pdf

This is another famous reference, with very step by step construction of the FEM equations.

R.W. Ogden, *Elements of the theory of finite elasticity*. In: *Nonlinear Elasticity: Theory and Applications* (edited by Y.B. Fu and R.W. Ogden) Cambridge University Press (2001), pp. 1-58 1. Available at link

The standard reference for finite-displacement elasticity