TMA4220 Numerical Solution of Differential Equations by element methods

Part 2: Vibration analysis

Trygve Bærland, Geir Amund Svan Hasle & Eirik Ingebrigtsen

November 22, 2013

Abstract

In this project we've developed a finite element method for solving the free vibration equation and finding a geometry's natural frequencies and modes of vibration. We've used linear elements, giving satisfactory qualitative results.

Introduction

With the displacement of spatial points in x_1- and x_2-direction represented as

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

the strain on each point is

$$\begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} \\ \frac{\partial u_2}{\partial x_2} \\ \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \end{bmatrix}$$

as many notes on linear elasticity will let you know.[1]

[1] Note however that this is the form usually called engineer strain.

The connection between the strain ε and the stress σ is

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} = \frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1-\nu}{2}\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{12}\end{bmatrix}$$
$$\boldsymbol{\sigma} = C\boldsymbol{\varepsilon}$$

where E is the Young's modulus and ν is the Poisson's ratio. Young's modulus characterises the solid's stiffness, i.e. large E means that you need a large force to deform the solid. Poisson's ratio is the ratio between the strains in x_1- and x_2-direction when submitting the solid to a stress in only one of the directions. To clarify: Consider a stress in only the x_1-direction direction[2], then $\nu > 0$ will say that the solid contracts in the x_2-direction and elongates in the x_1-direction. Worth noting is that $\nu < 0$ is possible, and some materials actually have this property. Weird. During the course of this analysis we will let E and ν completely describe a solid's properties. No stress or strain due to difference in temperature.

The equation and a variational formulation

In continuum mechanics

$$\rho\ddot{\boldsymbol{u}} = \nabla\sigma(\boldsymbol{u}) \tag{1}$$

describes the spatial displacement due to internal stresses alone. It is the extension of Hook's law and Newton's second law into a continuous medium. So rather than total mass we look at mass density, here denoted ρ.

To eventually arrive at variational formulation of (1) we should spend some time with the right hand side of this equation to better understand it. For that reason we will take a slight detour and start our journey by considering the equation

$$\nabla\sigma(\boldsymbol{u}) = -\boldsymbol{f} \tag{2}$$
$$\left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}\right]\begin{bmatrix}\sigma_{11} & \sigma_{12}\\ \sigma_{12} & \sigma_{22}\end{bmatrix} = -[f_1, f_2]$$

working on some domain Ω, together with some boundary conditions, which in its general form looks a little something like

$$\boldsymbol{u} = \boldsymbol{g}, \quad \text{on } \partial\Omega_D$$
$$\sigma(\boldsymbol{u})\cdot\hat{\boldsymbol{n}} = \boldsymbol{h}, \quad \text{on } \partial\Omega_N.$$

As is usual procedure when trying to derive a variational formulation, we multiply (2) with some test function $\boldsymbol{v} \in V$, where the function space V will be determined from what seems useful during the derivation. Then we integrate over the domain to get

$$\int_\Omega (\nabla\sigma(\boldsymbol{u}))\cdot\boldsymbol{v}d\Omega = -\int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}d\Omega. \tag{3}$$

[2] σ_{11} being the only stress different from zero.

We aim at applying Green's formula to the left hand side, and since Green's formula states that

$$\int_\Omega \nabla \cdot \boldsymbol{a} d\Omega = \int_{\partial\Omega} \boldsymbol{a} \cdot \hat{\boldsymbol{n}} dS$$

we should come up with a good candidate for $\boldsymbol{a}$. A good guess seems to be $\boldsymbol{a} = \sigma(\boldsymbol{u})\boldsymbol{v}$. Now, this isn't exactly the form the integrand on the left in (3) takes, so we need to do some calculations and find a relation between the two. Disclaimer for the following calculation: It is long, tedious and not extremely easy to read. In addition, for convenience's sake we use the notation $\frac{\partial}{\partial x_i} = \partial_{x_i}$.

$$\begin{aligned}
\nabla \cdot (\sigma(\boldsymbol{u})\boldsymbol{v}) &= \nabla \cdot \begin{bmatrix} \sigma_{11}v_1 + \sigma_{12}v_2 \\ \sigma_{12}v_1 + \sigma_{22}v_2 \end{bmatrix} \\
&= \partial_{x_1}(\sigma_{11}v_1 + \sigma_{12}v_2) + \partial_{x_2}(\sigma_{12}v_1 + \sigma_{22}v_2) \\
&= v_1\partial_{x_1}\sigma_{11} + \sigma_{11}\partial_{x_1}v_1 + v_2\partial_{x_1}\sigma_{12} + \sigma_{12}\partial_{x1}v_2 \\
&+ v_1\partial_{x_2}\sigma_{12} + \sigma_{12}\partial_{x2}v_1 + v_2\partial_{x_2}\sigma_{22} + \sigma_{22}\partial_{x_2}v_2.
\end{aligned}$$

Up to this point we've just done a vector dot product and some differentiation using the product rule. Let's continue by rearranging the terms to

$$\begin{aligned}
\nabla \cdot (\sigma(\boldsymbol{u})\boldsymbol{v}) = v_1(\partial_{x_1}\sigma_{11} + \partial_{x_2}\sigma_{12}) + v_2(\partial_{x_1}\sigma_{12} + \partial_{x_2}\sigma_{22}) \\
+ \sigma_{11}\partial_{x_1}v_1 + \sigma_{12}(\partial_{x_2}v_1 + \partial_{x_1}v_2) + \sigma_{22}\partial_{x_2}v_2.
\end{aligned}$$

We're definitely closing in on something. To make it even more easy on the eye we go back to the integrand on the left in (3) and see that

$$(\nabla\sigma(\boldsymbol{u})) \cdot \boldsymbol{v} = v_1(\partial_{x_1}\sigma_{11} + \partial_{x_2}\sigma_{12}) + v_2(\partial_{x_1}\sigma_{12} + \partial_{x_2}\sigma_{22}),$$

which corresponds nicely with the first part of what we have after rearranging terms. The last part we deal with by looking back at how the strain vector was defined. Using these two things we end up with

$$\nabla \cdot (\sigma(\boldsymbol{u})\boldsymbol{v}) = (\nabla\sigma(\boldsymbol{u})) \cdot \boldsymbol{v} + \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot \boldsymbol{\sigma}(\boldsymbol{u}). \tag{4}$$

The result of putting this into (3) is

$$\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot \boldsymbol{\sigma}(\boldsymbol{u}) d\Omega - \int_\Omega \nabla \cdot (\sigma(\boldsymbol{u})\boldsymbol{v}) d\Omega = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} d\Omega,$$

and we've put ourselves in position to employ Green's formula on the second term on the left hand side. What we end up with is

$$\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot \boldsymbol{\sigma}(\boldsymbol{u}) d\Omega = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} d\Omega + \int_{\partial\Omega} (\sigma(\boldsymbol{u})\boldsymbol{v}) \cdot \hat{\boldsymbol{n}} dS,$$

and can be further simplified by letting $V = H^1_{\Gamma_D} = \{v \in H^1 : v = 0 \text{ on } \partial\Omega_D\}$. Then

$$\begin{aligned}
\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot \boldsymbol{\sigma}(\boldsymbol{u}) d\Omega &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} d\Omega + \int_{\partial\Omega_N} \boldsymbol{v} \cdot \sigma\hat{\boldsymbol{n}} dS \\
&= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} d\Omega + \int_{\partial\Omega_N} \boldsymbol{v} \cdot \boldsymbol{h} dS
\end{aligned}$$

We are now prepared to make a variational formulation of (2):
Find $u \in \{H^1 : u = g, \text{ on } \partial\Omega_D\}$ so that

$$\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot C\boldsymbol{\varepsilon}(\boldsymbol{u}) d\Omega = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} d\Omega + \int_{\partial\Omega_N} \boldsymbol{v} \cdot \boldsymbol{h} dS \tag{5}$$

for all $v \in V$. Here we have now substituted $C\boldsymbol{\varepsilon}$ for $\boldsymbol{\sigma}$.

Galerkin projection

Since both function spaces we're looking at is infinite dimensional it is a good idea to let the test function reside in the function space $X_h \subset V$ of piecewise linear functions on a triangulation of Ω. Furthermore, let $X_h = \text{span}(\varphi_1, \ldots, \varphi_n)$ where φ_i are basis function with some compact support. Since the test functions v are now vector functions, there will for each node $\hat{i}$ be two test functions

$$\varphi_{\hat{i},1} = \begin{bmatrix} \varphi_{\hat{i}} \\ 0 \end{bmatrix}$$
$$\varphi_{\hat{i},2} = \begin{bmatrix} 0 \\ \varphi_{\hat{i}} \end{bmatrix},$$

and there should be some connection between the basis number i, node number $\hat{i}$ and $d = 1, 2$. If the total number of nodes are N, then total number of basis functions should be $2N$. A good function between the two can be $i = 2\hat{i} + (d-2)$. Then i ranges from 1 to $2N$. To build up a linear system of what we have so far we state the Galerkin formulation of the problem: Find $u \in X_h$ so that

$$\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot C\boldsymbol{\varepsilon}(\boldsymbol{u}) d\Omega = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} d\Omega + \int_{\partial\Omega_N} \boldsymbol{v} \cdot \boldsymbol{h} dS \tag{6}$$

for all $v \in X_h$.

It is sufficient to find a u where this is true for all the basis frunctions since the right hand side is a bilinear form and the right hand side is a linear functional. That means we can let $\boldsymbol{v} = \varphi_i$ with $i \in \{1, 2, \ldots, 2N\}$ and $u = \sum_j u_j \varphi_j$. We put this into our Galerkin formulation as

$$\begin{aligned} \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{\varphi_i}) \cdot C\boldsymbol{\varepsilon} \left(\sum_j u_j \boldsymbol{\varphi_j} \right) d\Omega &= \int_\Omega \sum_j u_j \boldsymbol{\varepsilon}(\boldsymbol{\varphi_i}) \cdot C\boldsymbol{\varepsilon}(\boldsymbol{\varphi_j}) d\Omega \\ &= \sum_j u_j \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{\varphi_i}) \cdot C\boldsymbol{\varepsilon}(\boldsymbol{\varphi_j}) d\Omega \\ &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{\varphi_i} d\Omega + \int_{\partial\Omega_N} \boldsymbol{\varphi_i} \cdot \boldsymbol{h} dS. \end{aligned}$$

Taking this to be true over all i we get the linear system

$$A\boldsymbol{u} = \boldsymbol{b},$$

where

$$A_{ij} = \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{\varphi_i}) \cdot C\boldsymbol{\varepsilon}(\boldsymbol{\varphi_j}) d\Omega$$
$$b_i = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{\varphi_i} d\Omega + \int_{\partial\Omega_N} \boldsymbol{\varphi_i} \cdot \boldsymbol{h} dS.$$

This linear system is what we will be implementing.

A simple test case

When using a finite element implementation it's in most cases a sound strategy to have compared to a test case where the exact solution is known. This is to build some confidence in the implementation. As a test example we are going to consider the following:

$$\begin{cases} \nabla\boldsymbol{\sigma}(\boldsymbol{u}) & = -\boldsymbol{f}, \quad \text{in } \Omega \\ \boldsymbol{u} & = \boldsymbol{0} \quad \text{on } \partial\Omega \end{cases} \tag{7}$$

where $\Omega = \{(x, y) \in \mathbb{R}^2 : |x|, |y| < 1\}$ and $\boldsymbol{f}$ is given as

$$f_1 = \frac{E}{1-\nu^2}\left(-2y^2 - x^2 + \nu x^2 - 2\nu xy - 2xy + 3 - \nu\right)$$
$$f_2 = \frac{E}{1-\nu^2}\left(-2x^2 - y^2 + \nu y^2 - 2\nu xy - 2xy + 3 - \nu\right).$$

In addition we are given

$$\boldsymbol{u} = \begin{bmatrix} (x^2-1)(y^2-1) \\ (x^2-1)(y^2-1) \end{bmatrix},$$

and all we need to do is verify that this is in fact the solution of (7). So let's do just that. To begin with we notice easily that the boundary condition is satisfied, so we proceed by calculating the strains as

$$\begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix} = \begin{bmatrix} 2x(y^2-1) \\ 2y(x^2-1) \\ 2y(x^2-1) + 2x(y^2-1) \end{bmatrix}.$$

Now we make use of the affine relation between σ and ε to deduce that

$$\begin{aligned} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} &= \frac{E}{1-\nu^2} \begin{bmatrix} \varepsilon_{11} + \nu\varepsilon_{22} \\ \nu\varepsilon_{11} + \varepsilon_{22} \\ \frac{1-\nu}{2}\varepsilon_{12} \end{bmatrix} \\ &= \frac{E}{1-\nu^2} \begin{bmatrix} 2x(y^2-1) + 2\nu y(x^2-1) \\ 2\nu x(y^2-1) + 2y(x^2-1) \\ (1-\nu)(y(x^2-1) + x(y^2-1)) \end{bmatrix} \end{aligned}$$

What follows is a tedious calculation, so brace yourself. The gradient of the stress is

$$\begin{aligned}\nabla \cdot \sigma(\boldsymbol{u}) &= \begin{bmatrix}\partial_x \sigma_{11} + \partial_y \sigma_{12} \\ \partial_x \sigma_{12} + \partial_y \sigma_{22}\end{bmatrix} \\ &= \frac{E}{1-\nu^2}\begin{bmatrix}2(y^2-1) + 4\nu xy + (1-\nu)(x^2 - 1 + 2xy) \\ (1-\nu)(2xy + y^2 - 1) + 4\nu xy + 2(x^2-1)\end{bmatrix} \\ &= \frac{-E}{1-\nu^2}\begin{bmatrix}-2y^2 - x^2 + \nu x^2 - 2\nu xy - 2xy + 3 - \nu \\ -2x^2 - y^2 + \nu y^2 - 2\nu xy - 2xy + 3 - \nu\end{bmatrix} \\ &= -\boldsymbol{f}\end{aligned}$$

So we now have a simple problem we know the analytic solution of and we can therefore use this to compare with what will become our finite element implementation.

Implementation in 2D

Every implementation of the finite element method can roughly be separated into three steps: Preprocessing, solving the system and postprocessing the data. As for our test example given by (7) the triangulated grid is given to us by the matlab script *getPlate.m*, so the next step is constructing both stiffness matrix and the loading vector. As is standard procedure for finite element implementations we go through each element and calculate that element's contribution to both the stiffness matrix and loading vector. Keep in mind that we are using triangular, linear elements, so every nodal basis function restricted to an element T_k can be written $\varphi_{\hat{i}}\big|_{T_k} = c_0 + c_1 x + c_2 y$ for some coefficients c_0, c_1 and c_2. Furthermore, since we will restrict ourselves to using a Lagrangian nodal basis, the function evaluation of some basis function $\varphi_{\hat{i}}$ in the node $\mathbf{x}_{\hat{j}}$ is $\varphi_{\hat{i}}(\mathbf{x}_{\hat{j}}) = \delta_{\hat{i}\hat{j}}$.[3]

To determine these coefficients, let's say the domain of an element is the triangle with vertices p_1, p_2 and p_3, then the vector $[c_0, c_1, c_2]^T$ solves

$$\begin{bmatrix}1 & p_1^T \\ 1 & p_2^T \\ 1 & p_3^T\end{bmatrix}\begin{bmatrix}c_0 \\ c_1 \\ c_2\end{bmatrix} = \begin{bmatrix}1 \\ 0 \\ 0\end{bmatrix}$$

if the vector corresponds to the coefficients of the basis function $\varphi\big|_{T_k}$ that is equal to 1 in p_1. Notice that the right hand side is the first column vector of the three dimensional identity matrix. Similarly we can determine the coefficients of the basis function evaluating to one in the other points by using other column vectors of the identity matrix on the right hand side of the system.

Reiterating that our basis functions are vector valued (see the section on Galerkin projection) we have that

$$\boldsymbol{\varepsilon}\left(\varphi_{\hat{i},d}\right) = \begin{bmatrix}c_1 \\ 0 \\ c_2\end{bmatrix}, \quad or \begin{bmatrix}0 \\ c_2 \\ c_1\end{bmatrix}$$

[3] The Kronecker delta.

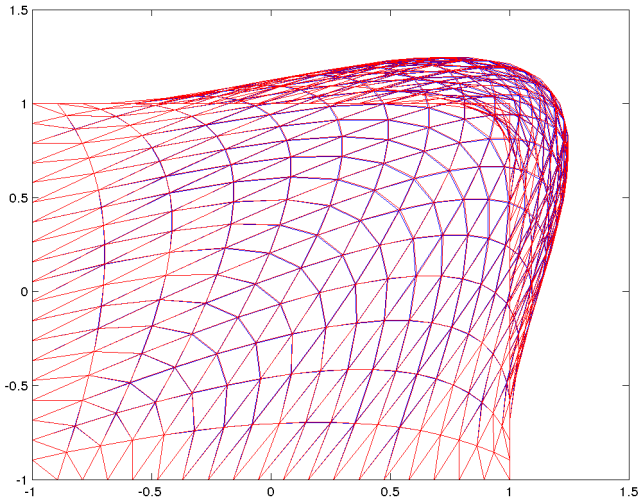

Figure 1: A comparison between the analytical and numerical solution of (7)

depending on whether $d = 1$ or 2 respectively. However, both these vectors are constant over the element's domain, and so the contribution of element T_k to the stiffness matrix in row i, column j is simply

$$\boldsymbol{\varepsilon}(\varphi_i) \cdot C\boldsymbol{\varepsilon}(\varphi_j) \cdot \text{area}(T_k),$$

which is easy to compute.

As for the contributions to the loading vector the product $\varphi_i \cdot \boldsymbol{f}$ is not in general constant over any triangle, so we don't end up with as nice of a result as in the stiffness contribution case. To make it simpler for ourselves we resort to using Gaussian quadrature. We use the same approach as for the stiffness matrix to determine the coefficients for the basis functions. Then simply taking the dot product of $\boldsymbol{f}$ and φ_i to make a new function handler in the code we are in a position to use the 2D numerical integrator we made in part 1 of this project.

So to sum up thus far: We go through each element on our global domain and calculate the coefficients of each basis function not vanishing on that element. Then we use these coefficients to calculate the contributions to the stiffness matrix and loading vector according to the above description. When this is done, the last thing we need to before solving the resulting system is to remove the rows and columns that corresponds to the Dirichlet boundary. After solving we put in 0-values in the right place of the solution vector. See figure 1 for a comparison between the analytical and numerical solution of (7). Instead of delving into analysing the error of our method we conclude from the figure that it seems to be working satisfactory.

Back to our main problem and moving up to 3D

Finally we shall return to (1) and consider a three-dimensional body. We have now that

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{23} \end{bmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ -\nu & 1 & -\nu & 0 & 0 & 0 \\ -\nu & -\nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(1+\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 2(1+\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(1+\nu) \end{bmatrix} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{bmatrix} = C^{-1}\boldsymbol{\sigma},$$

and so the stress-strain relation we will be using is $\boldsymbol{\sigma} = C\boldsymbol{\varepsilon}$ with

$$C = \begin{bmatrix} \frac{E}{2\nu^2+\nu-1} \begin{bmatrix} (\nu-1) & -\nu & -\nu \\ -\nu & (\nu-1) & -\nu \\ -\nu & -\nu & (\nu-1) \end{bmatrix} & 0_{3\times 3} \\ 0_{3\times 3} & \frac{E}{2(1+\nu)} I_{3\times 3} \end{bmatrix},$$

where $0_{3\times 3}$ is to be read as a 3×3-matrix consisting only of 0-entries and $I_{3\times 3}$ is the 3×3 identity matrix.

Using a very similar approach as in the 2D-case we have the variational formulation: Find $u \in V$ so that

$$\rho \int_\Omega \boldsymbol{v} \cdot \boldsymbol{u} d\Omega = -\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot C\boldsymbol{\varepsilon}(\boldsymbol{u}) d\Omega \tag{8}$$

for all $\boldsymbol{v} \in V$.

Again, rather than looking for a solution in the infinitely dimensional function space V we let the test function reside in $X_h \subset V$. X_h will be defined as before, only that now, since we are working in three dimensions,

$$\varphi_{\hat{i},1} = \begin{bmatrix} \varphi_{\hat{i}} \\ 0 \\ 0 \end{bmatrix}, \quad \varphi_{\hat{i},2} = \begin{bmatrix} 0 \\ \varphi_{\hat{i}} \\ 0 \end{bmatrix}, \quad \varphi_{\hat{i},3} = \begin{bmatrix} 0 \\ 0 \\ \varphi_{\hat{i}} \end{bmatrix}.$$

In addition the local-to-global function between the indices must now be rdefined to $i = 3\hat{i} + (d-3)$. With N nodes we look for a solution of the form $u_h = \sum_{i=1}^{3N} u_{h,i}\varphi_i$ and say that (8) must hold for all basis functions φ_j. What we end up with is the system

$$M\ddot{\boldsymbol{u}} = -A\boldsymbol{u} \tag{9}$$

where

$$M_{ij} = \rho \int_\Omega \varphi_i \cdot \varphi_j d\Omega$$
$$A_{ij} = \int_\Omega \boldsymbol{\varepsilon}(\varphi_i) \cdot C\boldsymbol{\varepsilon}(\varphi_j) d\Omega$$

and the matrices are called the mass– and stiffness matrix respectively.

The implementation of the stiffness matrix is the exact analogue of the 2D case and therefore doesn't need much of an elaboration. So let's spend a bit more time on one possible way of implementing the mass matrix M. As with the stiffness matrix we go through the domain Ω element by element and sum up the contributions. In contrast to how we calculated the contributions to loading vector we now transform the tetrahedral domain into a reference tetrahedron and calculate the contributions there. So consider an arbitrary tetrahedron T_k with vertices p_1, p_2, p_3 and p_4. Then for any $x \in T_k$ the mapping

$$\boldsymbol{\xi} = B(\boldsymbol{x} - p_1)$$

maps $\boldsymbol{x} = (x, y, z)^T$ to $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)^T$ which lies in the tetrahedron with vertices $(0,0,0)^T, (1,0,0)^T, (0,1,0)^T$ and $(0,0,1)^T$. Worth remarking on is that

$$B = \begin{bmatrix} (p_2 - p_1) & (p_3 - p_1) & (p_4 - p_1) \end{bmatrix}. \tag{10}$$

This tetrahedron we'll in the following call S and it has has basis functions

$$\begin{aligned} \mathcal{H}_1(\boldsymbol{\xi}) = 1 - \xi_1 - \xi_2 - \xi_3 \quad \mathcal{H}_2(\boldsymbol{\xi}) = \xi_1 \\ \mathcal{H}_3(\boldsymbol{\xi}) = \xi_2 \quad \mathcal{H}_4(\boldsymbol{\xi}) = \xi_3. \end{aligned}$$

Notice now that the contribution to the mass matrix is

$$\begin{aligned} \int_{T_k} \rho \varphi_i(\boldsymbol{x}) \cdot \varphi_j(\boldsymbol{x}) d\Omega &= \int_{T_k} \rho \varphi_{\hat{i},d}(\boldsymbol{x}) \cdot \varphi_{\hat{j},d}(\boldsymbol{x}) d\Omega \\ &= \rho \cdot \det(B) \int_S \mathcal{H}_\alpha(\boldsymbol{\xi}) \mathcal{H}_\beta(\boldsymbol{\xi}) dS. \end{aligned}$$

In the first equality we have taken advantage of the fact that $M_{ij} = 0$ unless $\varphi_{\hat{i}}$ and $\varphi_{\hat{j}}$ are placed on the same element in the vector, i.e. when the d in the local-to-global index mapping is the same for both i and j. $\alpha, \beta = 1, 2, 3, 4$ and corresponds to which of the vertices the original basis function should evaluate to one for. If we can calculate the mass matrix for the reference element S we can use this to calculate the contributions. Let's call this mass matrix the prototype mass matrix and denote it M^P. The calculation isn't all that interesting so we just state the result, which is

$$M^P = \frac{1}{60} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}.$$

And so the elements contribution to M is simply

$$\rho \cdot \det(B) M^P_{\alpha,\beta}. \tag{11}$$

Eigenfrequencies and –modes

After taking in some geometry, meshing it[4] and assembled the mass– and stiffness matrices we are done with what we can call the preprocessing part of the implementation. Now comes the solving of the system: We are interested in periodic solutions of the form

$$\boldsymbol{u} = \boldsymbol{u}e^{i\omega t}.$$

We put this into our system and get

$$\omega^2 M\boldsymbol{u} = A\boldsymbol{u}. \tag{12}$$

We see that this type of solutions is periodic with frequency ω and we call them eigenfrequencies because ω^2 are generalized eigenvalues of the system. Their corresponding vectors $\boldsymbol{u}$ are called eigenmodes. Since the meshing of our domain can contain a lot of nodes the matrices A and M will have a high number of rows and columns. This will yield a lot of eigenvalues, most of which we aren't all that interested in. Fortuitously Matlab has the 'eigs'-function at its disposal which can give out only the, say, $20, 30, 40$ or k smallest eigenvalues and their corresponding eigenvectors. To maybe better understand why we are only interested in the k *smallest* eigenvalues consider e.g. a piano. When hitting one of the keys the sound you hear is the superposition of many of that tones harmonics. In particular, the one you hear most clearly is the first harmonic, and then the second harmonic and so forth, and it is the smallest frequencies (lowest tones) that correspond to the first harmonics. This is analogous to any structure being put to vibrate by some means. The lowest frequencies will be the most expressed in the total vibration, and so naturally we would be most interested in them.

Actually, we are done with the solving part and all that remains is post processing.

Results

After using the 'eigs'-function in matlab we end up with k eigenvalues and –vectors. For the post processing part we used Paraview, and so we need a script for converting our results into something Praview can read. Happily, this could easily be gotten from the internet. Legally, I might add. Although a structure's natural modes of vibration aren't well-represented in a picture refer to fig. 2 to see what the 14'th mode of vibration for a chess pawn look like according to our findings. Note that the displacement is scaled for easier appreciation from the viewer. Since the chess pawn already has been analysed by Kjetil Johannesen it would be natural to compare our modes to those he found and submitted to youtube. All that can be stated here is that our results for the 7'th, 10'th and 14'th modes corresponded well with his results.

This method lends itself not only to the geometries of chess pawn, but to a wide variety of geometries. Considering the experiment depicted in *http://www.youtube.com/watch?v=wvJAgrUBF4w* we sought to recreate some of these patterns numerically. See fig. 3 for an example of a

[4] In this case we've either been given a geometry and its mesh or created one ourselves using gmsh.

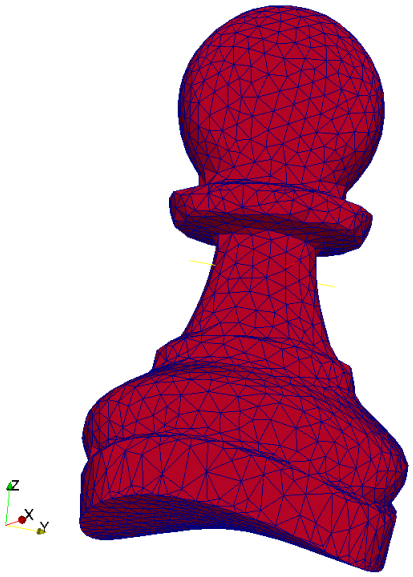

Figure 2: Vibration mode 14 of a chess pawn.

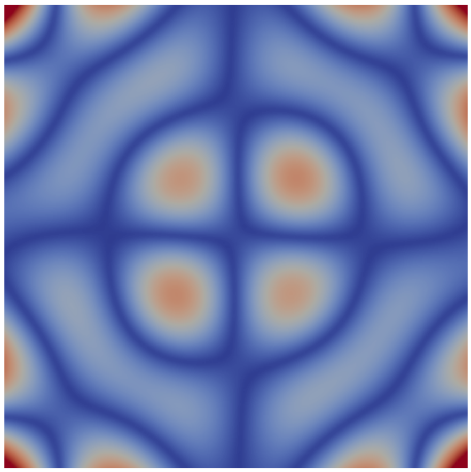

Figure 3: An eigenmode pattern for a uniform, thin plate.

pattern we managed to create. Here the colour coding corresponds to the magnitude of $\boldsymbol{u}$ in that point. Where it's blue there are small to no displacements and this is where the salt in the experiment would have settled. A slight problem was that we didn't have the time, nor means, to recreate the experiment, and the material of the plate in the video is up to discussion. It would seem that our choice of material (reflected in numerical values of E and ν) is a bit stiffer than that in the video. However, that is not the biggest problem: Say that a plate has k eigenfrequencies in some interval. If k is large enough and the meshing of the plate is coarse you would only get out a few eigenfrequencies in that interval, and they would be crude approximations of any of the actual frequencies. So a coarse mesh of linear elements will not reflect reality well, and in some engineering settings where accuracy is important this can have dire consequences. A natural remedy would of course be to make the mesh finer, and that will yield better results, but this will lengthen the computational time considerably. Fig. 3 is the result of a fine mesh that took about a week to process and assemble the mass– and stiffness matrix of. So better results come at the price of greatly increasing the computation time.

Adding some boundary conditions

Looking back at (12) it's worth noting the lack of boundary conditions. In our system it isn't necessary, but it can be interesting to consider it nonetheless. Since (12) doesn't have any loading vector in which we can put contributions from a possible Neumann or Robin condition these types of conditions seems out of the question. However Dirichlet boundary condition, and especially homogenous Dirichlet seems attractive. If the domain has homogenous Dirichlet conditions on some part of the boundary, this translates to keeping that part immovable (by e.g. welding it to some large solid). In the way our code has been structured homogenous Dirichlet is also quite easy to implement. Just identify the nodes that are part of the Dirichlet boundary and remove the corresponding rows and columns from M and A, and insert the boundary values in the right places of the resulting eigenvectors. See fig. 4 for an example. Here we've taken homogenous boundary conditions on the bottom part of the chess pawn, resulting in some interesting, new vibration modes.

Discussion

We've seen that the code generated from what has been outlined in this text is flexible in the sense that we can easily take in new, complex geometries and study their vibrational modes. Although not done in this project it would be possible to extend the code so that a geometry can be made up of more than one uniform material. The code is also flexible in that we can impose Dirichlet boundary conditions on the domain with relative ease. Other types of boundary conditions make little sense in the way we've formulated the problem (eq. 12).

It also seems that using linear elements, easy as the are to make into code, are a bit naive. They do give nice results when the mesh is fine, but at the cost of a long

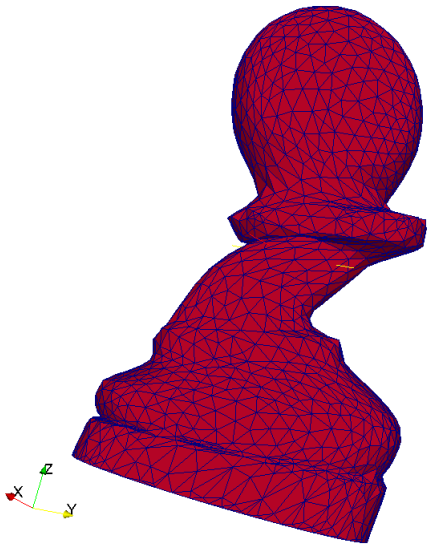

Figure 4: A vibration mode for a chess pawn with homogenous Dirichlet condition on the bottom of it.

computation time. As a conclusion we can say that the code gives good qualitative results and can make for good insights in that regard, but if it's fast and accurate results you're after, more refined methods are advised. There is definitely a potential for improving the code, e.g. using parallellization, more advanced element types or possibly some other methods outside of the practical scope of the TMA4220 course.