Implementation of a non-conforming rotated Q1 approximation on tetrahedron

Master’s Thesis in Computational Science and Engineering

Mahdieh Khanmohammadi
Mirza Cenanovic
Implementation of non-conforming rotated Q1 approximation on tetrahedron

Master’s thesis in Computational Science and Engineering

School of Information Science, Computer and Electrical Engineering
Halmstad University
Box 823, S-301 18 Halmstad, Sweden

June 2011
Acknowledgements

We would like to thank our supervisor Prof. Peter Hansbo and Prof. Bertil Nilsson for all the help and support that we received.
# Table of contents

Acknowledgements ........................................................................................................................................... iv  
Abstract .......................................................................................................................................................... 2  
1. Introduction ................................................................................................................................................... 3  
2. Background and Theory ............................................................................................................................ 3  
3. Method ............................................................................................................................................................ 4  
   3.1. Numerical Method using commercial software .................................................................................. 4  
   3.2. Nonconforming Elements $Q_1$ rotated approximation method ......................................................... 5  
   3.3. Analytically solvable eigenvalue problem ........................................................................................... 8  
      3.3.1. An ansatz for the eigenfunctions ................................................................................................. 9  
   3.4. Linear elasticity problem ...................................................................................................................... 11  
   3.5. Time dependent problem .................................................................................................................... 13  
4. Implementation of the method ..................................................................................................................... 14  
   4.1. Part 1 – Solving the problem with SolidWorks and COMSOL .......................................................... 14  
   4.2. Part 2 – Solving the problem numerically using the $Q_1$ method .................................................... 14  
   4.3. Part 3 – Solving the problem analytically ............................................................................................ 16  
   4.4. Part 4 – Solving the time dependent problem on a beam ................................................................. 16  
5. Results .......................................................................................................................................................... 18  
6. Conclusion ...................................................................................................................................................... 26  
7. References ..................................................................................................................................................... 27
Abstract

Our project consists of two parts (A and B). In part A we solve a linear elasticity problem with implementing a rotated $Q_1$ approximation method and simulating the problem in commercial softwares (COMSOL and SolidWorks). To evaluate the results we implement an analytical eigenvalue solver. As a simple case, we use a cube with side length of $L = 1m$ made of Alloy steel with density of $7850Kg/m^3$. In part B we implement a time dependent linear elasticity problem on a beam made of Alloy steel with density of $7850Kg/m^3$ with size of $1x 0.1x 0.01 m$. We use the implicit method to solve our problem. The frequency results in part A show that rotated $Q_1$ approximation method works more accurate than the commercial softwares.
1. Introduction

In this study we determine how well the nonconforming rotated Q1 method, using tetrahedral elements, behaves compared to the real solution and to the commercial software. The industry requires faster and more accurate solutions in order to cut lead times and costs. The goal is to present a comparison between different methods by implementing the rotated Q1 approximation on a linear elastic problem to compare the natural frequencies in a metal cube with frequencies derived with an analytical method and commercial software.

The purpose of this study is to make ground for future studies in this area to implement faster, more effective and cost efficient methods. The limitations in this study are that we only use a cube and a beam as our domain. We also only apply tetrahedron elements and we only solve the linear elasticity problem.

Our study consists of two parts, A and B.

In the part A, we consider the linear elasticity problem in three dimensions and we solve it using three different methods.

1. We solve the problem using COMSOL and SolidWorks simulation.
2. We develop the rotated $Q_1$ approximation in MATLAB and solve the problem.
3. We implement an analytically solvable eigenvalue method to solve the problem.

In all the methods we use an Alloy steel cube with side length of $L = 1m$. The next step is to compare the frequencies using both the numerical method (COMSOL and SolidWorks) and the rotated $Q_1$ approximation with the analytical results.

In the part B, we implement our method on a beam to solve a time dependent linear elasticity problem and animate the beam as it oscillates.

2. Background and Theory

Finite element analysis (FEA) is a numerical method for solving partial differential equations by splitting the computational domain into finite elements. It treats each element separately and then combines them into a whole solution. It is used in new
product development, construction and other areas. A company can cut lead times and lower costs by using computations and simulation prior to developing prototypes and testing, if needed at all. In case of structural failure, FEA may be used to help determine the design flaw. Regarding this fact, it is essential to have an accurate finite element method, accurate in the sense of being as close to the real world model as possible.

In general, there are two types of analysis that are used in industry: 2D (triangular or quadrilateral) modeling, and 3D (tetrahedral, hexahedral, etc.) modeling. Although 2D modeling is simpler to implement and allows the analysis to be run on a computer that is not high speed and it tends to yield less accurate results. 3D modeling, however, produces more accurate results while sacrificing the ability to run on all but the fastest computers effectively. Here, in this project we apply a 3D modeling using tetrahedral element.

Within each of 2D and 3D modeling schemes, the programmer can insert numerous algorithms (functions) which may make the system behave linearly or non-linearly. Linear systems are far less complex and generally do not take into account plastic deformation. In our project we do not account for plastic deformation and we use linear systems.

3. Method

3.1. Numerical Method using commercial software

In this part of the project we use COMSOL and SolidWorks to solve our problem numerically. COMSOL Multiphysics is a finite element analysis package including solver and simulation software for various physics and engineering applications, so called multiphysics. Also we use COMSOL to mesh our object with tetrahedron element and then we export the mesh data to MATLAB.

Besides, in finite element analysis, a problem is represented by a set of algebraic equations that must be solved simultaneously. There are two classes of solution methods: direct and iterative. Direct methods solve the equations using exact numerical
techniques. Iterative methods solve the equations using approximate techniques where, in each iteration, a solution is assumed and the associated errors are evaluated. The iterations continue until the errors become acceptable.

The SolidWorks software offers the following choices:

- **Automatic**: The software selects the solver based on the study type, analysis options, contact conditions, etc.
- **Direct Sparse**
- **FFEPlus (iterative)** [1]

The Automatic choice for a solver is the default option for Static, Frequency, and Thermal studies which in our case we use Automatic for frequency as well.

We have implemented the linear elasticity problem on a cube with side length of $L = 1m$ in COMSOL and SolidWorks. For this study the material data that is needed is the density, the modulus of elasticity and Poisson’s ratio. The mesh generation is done by tetrahedron elements and the solver in COMSOL uses the tetrahedron vertices to solve the problem (corresponding to a piecewise linear approximation).

### 3.2. Nonconforming Elements $Q_1$ rotated approximation method

In this study we are looking to model our object with tetrahedron which is a 3 dimensional element. In our nonconforming case we use a bi-linear approximation and we relax the continuing requirements. For this approximation we apply a 3D version of the Rannacher-Turek element which is piecewise linear but continuous in the midpoint of the side [3, 4]. To elaborate the method more clearly we start to explain our method by defining the base functions in 2 dimensions.

We have a bounded region $\Omega \subset \mathbb{R}^2$ into a geometrically conforming finite element which partitions $T_h = \{\bar{R}\}$ of $\Omega$ consisting of convex quadrilaterals. For each element $\bar{R} \in T_h$, we denote a coordinate system which is represented by joining the midpoint of each edges of tetrahedron called $(\xi, \eta)$. We have
Here we can use any type of continuity which we want. We decided to choose the continuity at the midpoints of the edges [5-6]. Based on existence of the parametric version of $Q_1^R(\bar{\mathcal{R}})$, there is a reference configuration $\bar{\mathcal{R}}$ for the approximation. We have the global coordinates $(x, y)$ and the parametric coordinates $(\xi, \eta)$ which a reference element defined for $0 \leq \xi \leq 1$, $0 \leq \eta \leq 1$. As illustrated in Figure.1 we use $(r, s)$ system to define a local base function for the approximation with $0 \leq r \leq 1$, $0 \leq s \leq 1$ which we draw the local basis as follows

$$\phi_1 = (1-r)(1-s), \quad \phi_2 = r(1-s), \quad \phi_3 = rs, \quad \phi_4 = (1-r)s$$

(3.2)

$\phi_i$ denotes the nonconforming basis function [4].

Figure1. Reference element $\bar{\mathcal{R}}$ in different coordinates and rotated reference element $\bar{\mathcal{R}}$. [4]
The basis functions in the parametric domain of $\xi$ and $\eta$ can simply obtained from equation below

$$
\begin{bmatrix}
[1]
\end{bmatrix} = \begin{bmatrix}
\frac{-1}{2}
\frac{1}{2}
\end{bmatrix} + \begin{bmatrix}
1
1
\end{bmatrix} \begin{bmatrix}
\xi
\eta
\end{bmatrix},
$$

(3.3)

Leading to

$$
\phi_1 = \frac{3}{4} + \xi - \xi^2 - 2\eta + \eta^2, \quad \phi_2 = -\frac{1}{4} + \xi^2 + \eta - \eta^2,
\phi_3 = -\frac{1}{4} + \xi - \xi^2 + \eta^2, \quad \phi_4 = \frac{3}{4} - 2\xi + \xi^2 + \eta - \eta^2
$$

(3.4)

We define an isoparametric map $F : (\xi, \eta) \rightarrow (x, y)$ as follows:

$$(x, y) = F(\xi, \eta) := \sum \phi_i(\xi, \eta) m_i
$$

(3.5)

Where $m_i$ is the midpoints of the edges. Thus $m_1 = (x_a + x_b)/2$, $m_2 = (x_b + x_c)/2$, $m_3 = (x_c + x_d)/2$, and $m_4 = (x_d + x_a)/2$. Simply we can compute

$$x(\xi) = (x_b + x_c - x_a - x_d)\xi/2 + (x_c + x_d - x_a - x_b)\eta/2 + (3x_a + x_b - x_c + x_d)/4,
$$

(3.6)

In the case of 3D we map the vertices to the edges of a tetrahedron. We have a bounded region $\Omega \subset \mathbb{R}^3$ into a geometrically conforming finite element which partitions $T_h = \{K\}$ of $\Omega$ consisting of convex quadrilaterals. For each element $K \in T_h$, we denote a coordinate system which is represented by joining the midpoint of each edges of tetrahedron called $(\bar{\xi}, \bar{\eta}, \bar{\zeta})$. We have the rotated approximation as below:

$$Q_1^K(\tilde{K}) := \text{span}\{1, \xi, \eta, \bar{\zeta}, \xi^2 - \bar{\eta}^2, \bar{\zeta}^2 - \xi^2\}
$$

(3.7)

The nodal basis element associated with the reference element is as follows:

$$\phi_1 = 1/3 (2 + 2\xi - 7\eta + 2\zeta - 2\xi^2 + 4\eta^2 - 2\zeta^2),
\phi_2 = 1/3 (-1 - \xi + 2\eta + 2\zeta + 4\xi^2 - 2\eta^2 - 2\zeta^2),
\phi_3 = 1/3 (-1 + 2\xi - \eta + 2\zeta - 2\xi^2 + 4\eta^2 - 2\zeta^2),
\phi_4 = 1/3 (2 - 7\xi + 2\eta + 2\zeta + 4\xi^2 - 2\eta^2 - 2\zeta^2),
\phi_5 = 1/3 (2 + 2\xi + 2\eta - 7\zeta - 2\xi^2 - 24\eta^2 + 4\zeta^2),
\phi_6 = 1/3 (-1 + 2\xi + 2\eta - \zeta - 2\xi^2 - 2\eta^2 + 4\zeta^2).
$$

(3.8)
Figure 2 shows the physical element T. To map our reference configuration to a physical tetrahedron T we need to define an isoparametric map $F : (\xi, \eta, \zeta) \rightarrow (x, y, z)$ by

$$(x, y, z) = F(\xi, \eta, \zeta) := \sum \phi_i(\xi, \eta, \zeta) \mathbf{m}_i,$$

(3.9)

Where $\mathbf{m}_i$ elucidates the midpoints of the edges on tetrahedron. Based on the numbering which is showed in Figure 2 we have

$$x(\xi) = x_a + (x_b + x_c - x_a - x_d)\xi/2 + (x_c + x_d - x_a - x_b)\eta/2 + (-x_a + x_b - x_c + x_d)\zeta/2,$$

(3.10)

3.3. Analytically solvable eigenvalue problem

In particular transferring from discipline of structural engineering to a mathematical area is current and intense interest. This area involves examining the eigenvalues of systems of differential equations. In the structural analogy, the problem is that of finding the natural frequencies of vibration in an object. On the other hand, in the mathematical point
of view, the problem is that of finding a solution to the eigenvalue problem of linear elasticity equation (Navier’s equations) [7].

We consider the hexahedral domain \( \Omega = [0, L_x] \times [0, L_y] \times [0, L_z] \), and we define the boundary conditions:

\[
\begin{align*}
  u &= \sigma_{xy} = \sigma_{xz} = 0 \quad \text{if} \quad x \in \{0, L_x\}, \\
  v &= \sigma_{xy} = \sigma_{yz} = 0 \quad \text{if} \quad y \in \{0, L_y\}, \\
  w &= \sigma_{xz} = \sigma_{yz} = 0 \quad \text{if} \quad z \in \{0, L_z\}.
\end{align*}
\]

Equivalently we draw the strain equations in each direction as follows:

\[
\begin{align*}
  u &= \frac{\partial v}{\partial x} = \frac{\partial w}{\partial x} = 0 \quad \text{if} \quad x \in \{0, L_x\}, \\
  v &= \frac{\partial u}{\partial y} = \frac{\partial w}{\partial y} = 0 \quad \text{if} \quad y \in \{0, L_y\}, \\
  w &= \frac{\partial u}{\partial z} = \frac{\partial v}{\partial z} = 0 \quad \text{if} \quad z \in \{0, L_z\}.
\end{align*}
\]

Assuming the time dependent form of \( e^{i\omega t} \) the function \( u \) must satisfy the eigenvalue problem:

\[
(1 - 2\nu)\nabla^2 u + \nabla \cdot u = -u \psi \tag{3.13}
\]

Where \( \psi \) is the eigenvalue. Applying the boundary conditions in (3.12), the frequencies \( \omega \) are determined from

\[
\rho \omega^2 = \frac{E}{2(1+\nu)(1-2\nu)} \tag{3.14}
\]

3.3.1. An ansatz for the eigenfunctions

The eigenfunctions are the solution to the PDEs. But, before finding the eigenfunctions it is proper to figure out the basis for the functions that are satisfying the boundary conditions. Then we will find the eigenfunctions based on the simple representation of the basis functions [7].
Here we want to solve a second order PDE and we know the functions \((\sin(nx))_{n>0}\) are a basis for the smooth univariate functions vanishing at 0 and \(\pi\). On the other hand, \((\cos(nx))_{n>0}\) are a basis for the smooth univariate functions whose derivative vanishes 0 and \(\pi\) [7]. All the components of an arbitrary function \(u\) which satisfies equation (3.12) have the same bases. To make all the eight possibilities, we choose either \(\sin()\) or \(\cos()\) for each coordinate axis. Each interval is transformed to \([0,\pi]\) using,

\[\alpha_l = \frac{ln}{L_x}, \beta_m = \frac{m\pi}{L_y}, \gamma_n = \frac{n\pi}{L_z}, \quad l, m, n = 1, 2, 3, \ldots \quad (3.15)\]

We have

\[f_{lmn}(x) = \sin(\alpha_lx) \cos(\beta_my) \cos(\gamma_nz), \quad g_{lmn}(y) = \cos(\alpha_lx) \sin(\beta_my) \cos(\gamma_nz), \quad h_{lmn}(z) = \cos(\alpha_lx) \cos(\beta_my) \sin(\gamma_nz). \quad (3.16)\]

Where \(f_{lmn}(x), g_{lmn}(y),\) and \(f_{lmn}(x)\) are the bases for \(u, v\) and \(w\) respectively.

For all components we look for eigenfunctions with the same value of \((l, m, n)\). We have

\[f_{lmn} = (f_{lmn}, g_{lmn}, h_{lmn}), \quad (3.17)\]

For a nonzero constant \(q\) we make the ansatz that we may write each eigenfunctions like \(u = f_{lmn} \ast q\). The \(\ast\) symbol stands for component wise Hadamard product of two vectors (component wise multiplication). By substituting the ansatz into the equation (3.13), we find the eigenfunctions. For each nonzero \(PT = (\alpha_i, \beta_m, \gamma_n)\), \(q\) will satisfies

\[\psi q = (1 - 2v)(P^T P)q + (P^T q)P. \quad (3.18)\]

With nonzero \(P\) and \(q\) vectors we will get the eigenvector either if \(q \cdot P = 0\) or \(q = P\). These two families of eigenvectors called solenoidal and compressive respectively.

Thus the solenoidal eigenfunctions are spanned with the functions below

\[
\begin{align*}
A_{lmn}(x) &= (0, \gamma_n g_{lmn}(x), -\beta_m h_{lmn}(x)), \\
B_{lmn}(x) &= (\gamma_n f_{lmn}(x), 0, -\alpha_i h_{lmn}(x)), \\
C_{lmn}(x) &= (\beta_m f_{lmn}(x), -\alpha_i g_{lmn}(x), 0).
\end{align*}
\quad (3.19)
\]
Each group of eigenfunctions have zero divergence. The eigenvalues are
\[ \psi_{lmn}^{(c)} = (1 - 2\nu)(\alpha_l^2 + \beta_m^2 + \gamma_n^2). \] (3.20)

If \( l = 0 \), then \( \alpha_l = f_{ml} = 0 \).

The compressive eigenfunctions are as follows:
\[ D_{lmn}(x,y,z) = (\alpha_l f_{lmn}(x,y,z), \beta_m g_{lmn}(x,y,z), \gamma_n h_{lmn}(x,y,z)), \] (3.21)

For \((l,m,n) \neq 0\) have nonzero divergence. The eigenfunction with \((l,m,n) = 0\) is omitted and the eigenvalues are:
\[ \psi_{lmn}^{(c)} = 2(1 - \nu)(\alpha_l^2 + \beta_m^2 + \gamma_n^2), \quad l^2 + m^2 + n^2 \neq 0, \] (3.22)

If at least one of the component of \((l,m,n)\) is nonzero the \( \psi_{lmn}^{(c)} \) is an eigenvalue, if two components of \((l,m,n)\) are nonzero then \( \psi_{lmn}^{(c)} \) is an eigenvalue, and if all the components of \((l,m,n)\) are nonzero \( \psi_{lmn}^{(c)} \) is a double eigenvalue [7].

3.4. Linear elasticity problem
The physical model for linear elastic problems consists of equations, which express a relation between the stress and strain tensors, and the equilibrium equation. This first-order partial differential system is called the stress-displacement formulation. Substituting the stress into the equilibrium equation leads to a second-order elliptic partial differential system called the pure displacement formulation. The stress-displacement formulation is sometimes preferable to the pure displacement formulation for some important practical problems, e.g., modeling of nearly incompressible or incompressible materials and modeling of plastic materials where the elimination of the stress tensor is difficult. In addition, the stress is usually a physical quantity of primary interest. It can be obtained in the pure displacement method by differentiating displacement, but this degrades the order of the approximation [4]. However, it is more than the pure displacement method, which is more common in practical applications.
In this study we consider the pure displacement formulation of the linear elasticity problem in a domain $\Omega$ in $\mathbb{R}^3$. In the problem we are looking for the displacement vector $\mathbf{u} = [u_i]_{i=1}^3$ and the symmetric stress tensor $\sigma = [\sigma_{ij}]_{i,j=1}^3$ such that

$$
\begin{align*}
\sigma &= \lambda \nabla \cdot \mathbf{u} + 2\mu \varepsilon(\mathbf{u}) \quad \text{in } \Omega, \\
-\nabla \cdot \sigma &= \mathbf{f} \quad \text{in } \Omega, \\
\mathbf{u} &= \mathbf{0} \quad \text{on } \partial \Omega_D, \\
\mathbf{n} \cdot \sigma &= \mathbf{h} \quad \text{on } \partial \Omega_N.
\end{align*}
$$

(3.23)

The displacement vector $\mathbf{u}$ has $i$th component $u_i$ and the linearized strain tensor is a $3 \times 3$ symmetric matrix defined by components below:

$$
\varepsilon_{ij}(\mathbf{u}) = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right),
$$

(3.24)

The linearized stress tensor is also a $3 \times 3$ matrix defined by components below:

$$
\sigma_{ij} = 2\mu \left( \varepsilon_{ij} - \frac{1}{3} \delta_{ij} \varepsilon_{kk} + K \delta_{ij} \varepsilon_{kk} \right) = 2\mu \varepsilon_{ij} + \lambda \delta_{ij} \varepsilon_{kk},
$$

(3.25)

Where $\varepsilon_{kk}$ is the trace of the strain matrix, $\lambda = K - \frac{2\mu}{3}$ and $\mu$ are named Lame’s constants, and $K$ is called the modulus of compression. According to the Hooke’s law we implement the equilibrium equation for the strain and external body forces $\mathbf{f} \ dV$ like this: $\text{div} \sigma + \mathbf{f} = 0$

Based on the Newton’s law we have

$$
\rho \frac{\partial^2 \mathbf{u}}{\partial t^2} + \mathbf{f} = 0
$$

(3.26)

It is more common to state the stress tensor using Young’s modulus $E$ and Poisson’s ratio $\nu$ which with respect to Lame’s constant and $K$ we draw

$$
E = \frac{9K\mu}{3K + \mu} = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}, \quad \text{and} \quad \nu = \frac{1}{2} \frac{3K - 2\mu}{3K + \mu} = \frac{\lambda}{2(\lambda + \mu)}
$$

(3.27)

So,

$$
\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} , \quad \text{and} \quad \mu = \frac{E}{2(1+\nu)}
$$

(3.28)
3.5. Time dependent problem

Here we are solving a second order time dependent derivative problem using an implicit and explicit method. We have

\[
\begin{align*}
\dot{v} &= \frac{du}{dt} \\
M\frac{dv}{dt} + Su &= f'
\end{align*}
\]  (3.29)

Assume that we have discrete times \( t_n \) on the interval \( t_n \leq t \leq t_{n+1} \). Then we have

\[ k_n = t_{n+1} - t_n. \]

Considering same approximation for \( v \) and \( u \) and collecting the same point we have

\[
\begin{align*}
\frac{u_{n+1} - u_n}{k_n} &= \theta v_{n+1} + (1 - \theta)v_n \\
M\frac{v_{n+1} - v_n}{k_n} + S(\theta u_{n+1} + (1 - \theta)u_n) &= \theta f_{n+1} + (1 - \theta)f_n
\end{align*}
\]  (3.30)

Where for implicit method \( \theta = 1 \) and for explicit method \( \theta = 0 \).

If we solve the equations for \( v_{n+1} \) we get

\[
(M + k_n^2\theta^2S)\dot{v}_{n+1} = (M - k_n^2(1 - \theta^2)S)v_n + k_n(f_n(1 - \theta) + f_{n+1}\theta) - k_nSu_n. \]  (3.31)

After finding the solution we animate the displacement.

To solve a time dependent linear elastic problem, both implicit and explicit methods can be used. In the explicit method we use a lumped mass matrix, by doing the row-sum trick [8]. Because of the fact that the lumped mass matrix is diagonal we don’t need to inverse the matrix to solve the problem. This method is stable under a time step restriction only. However, the method is fast and cost effective, and may for very large problems be the only option.

As it is well known, in a P2 method we cannot lump the mass matrix to be able to use an explicit method. That is because of the fact that in a P2 method we obtain a non positive definite lumped mass matrix which cannot be used in an explicit time stepping scheme. In a Q1 method, however, the lumped mass matrix is positive definite.
4. Implementation of the method

Our implementation is divided into four parts.

1. SolidWorks and COMSOL simulation
2. Implementation of the rotated Q1 method on a cube
3. Implement the analytical solver in MATLAB
4. Implementation of the time dependent problem on a beam

4.1. Part 1 – Solving the problem with SolidWorks and COMSOL

In both SolidWorks and COMSOL we create a cube with the side length of 1 meter and apply an alloy steel material to it. For this study the material data that is needed is the density, the modulus of elasticity and the Poisson’s ratio. We used the “standard” values: density (rho) = 7850 Kg/m³, modulus of elasticity (E) = 200*10^9 N/mm² (MPa), Poisson’s ratio = 0.33.

To our knowledge all commercial software that use tetrahedral elements use the corner points to define basis functions.

We run a static study and apply rollers as boundary conditions to get a greased wall on all sides of the cube. We apply the alloy steel material with the properties mentioned above. We create both a coarse and a fine mesh to see how the results differ, which you can find in the result section of this report.

4.2. Part 2 – Solving the problem numerically using the Q1 method

Here is the flowchart for the solution:
Start

Use imported mesh data from COMSOL to get the coordinates and nodes.

Redefine the boundary on the cube because of numerical errors that cause infinite element.

Compute the edges of the elements and the midpoints of the edges.

Implement the material data.

Loop over each element.

Use the basis functions according to the C1 method.

Compute Gauss-points for each element.

Loop over each Gauss-point.

Implement the numerical method to solve the linear elastic problem.

Assemble the local stiffness and mass matrix.

Looped over all Gauss-points?

No

Yes

Assemble the global stiffness and mass matrix.

Looped over all elements?

No

Yes

Apply boundary conditions.

Use eigenvalue solver to get eigenvalues and eigenvectors.

Project the midpoint displacements on the corners.

Visualize the displacements for the lowest frequency mode.
4.3. Part 3 – Solving the problem analytically

We implemented the analytical method in MATLAB using the same material properties and getting the results that we need for comparison.

Based on the theoretical part we have discussed in section 3.3.1 we define the parameter \( \alpha = \pi / L \) with \( L = 1m^3 \). Assuming the value of \( l, m \) and \( n \) are in the same interval of \([0, \text{neig}]\) (where \( \text{neig} \) is the number of eigenvalues that we are looking for) we have \( \alpha = \beta = \gamma \). Creating different combinations we can get the eigenvalues. The loops are implemented based on the fact that if only one of the components \( l, m \) and \( n \) is nonzero the \( \psi^{(c)}_{lmn} \) is an eigenvalue, and if two components of \( (l, m, n) \) are nonzero then \( \psi^{(s)}_{lmn} \) is also an eigenvalue, and if all the components of \( (l, m, n) \) are nonzero \( \psi^{(s)}_{lmn} \) is a double eigenvalue. We omit all the duplicates and sort them. The last step is to customize these eigenvalues to fit our particular problem.

4.4. Part 4 – Solving the time dependent problem on a beam

As a part B of this study we decided to visualize a beam oscillating in its natural mode. We create a model of a simple beam in SolidWorks with the dimensions 1m * 0.1m * 0.01m. The mesh is created using COMSOL and imported to MATLAB where we implement our code.

To visualize the beam oscillation we have to implement the time dependent theory. We use the implicit time dependent method because it is the most stable one and in this case accuracy is not a priority, we just want to see it oscillate in its normal mode. The flowchart of the algorithm is shown below.
5. Results

We have done this study in two parts. In part A we have implemented a rotated Q1 approximation on a linear elasticity problem. For simplicity, we have used a cube made of Alloy steel with side length of 1m and density of 7850 kg/m$^3$ modulus of elasticity (E) = 200*10$^9$ N/mm$^2$, Poisson's ratio = 0.33. We have simulated the linear elastic problem on the same cube in SolidWorks and COMSOL. To compare our results, we have implemented an analytical method to solve an eigenvalue problem. In this section you can see the results for all the methods that we have implemented and simulated. We have five tables that include the frequency results for the various methods that we implement comparing to the analytical results (Table 1- Table 5). We have also estimated the displacements for both the coarse and the fine meshes using the full and lumped mass matrices. The results are shown in Figure 3.

We have compared the time to compute the eigenvalues using the full and lumped mass matrices. The results can be seen in Table 6.

In part B of the study we have applied the rotated Q1 approximation to solve a time dependent linear elastic problem. We have used the same material properties. The oscillation results of the five different modes of the beam can be seen in Figure 4.

SolidWorks results

Table 1. The frequency results using coarse mesh comparing with the analytical results in five modes.

<table>
<thead>
<tr>
<th>Mode Number</th>
<th>Frequency(Hertz)</th>
<th>Analytical solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2238.3</td>
<td>2188.4 Hz</td>
</tr>
<tr>
<td>2</td>
<td>2769.5</td>
<td>2680.2 Hz</td>
</tr>
<tr>
<td>3</td>
<td>3054.4</td>
<td>3072.0 Hz</td>
</tr>
</tbody>
</table>
Table 2. The frequency results using fine mesh comparing with the analytical results in five modes.

<table>
<thead>
<tr>
<th>Mode Number</th>
<th>Frequency (Hertz)</th>
<th>Analytical solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2206.3</td>
<td>2188.4 Hz</td>
</tr>
<tr>
<td>2</td>
<td>2709.6</td>
<td>2680.2 Hz</td>
</tr>
<tr>
<td>3</td>
<td>3084.4</td>
<td>3072.0 Hz</td>
</tr>
<tr>
<td>4</td>
<td>3482.3</td>
<td>3460.1 Hz</td>
</tr>
<tr>
<td>5</td>
<td>3824.3</td>
<td>3790.4 Hz</td>
</tr>
</tbody>
</table>

**COMSOL** results:

Table 3. The frequency results using coarse mesh comparing with the analytical results in five modes.

<table>
<thead>
<tr>
<th>Mode Number</th>
<th>Frequency (Hertz)</th>
<th>Analytical solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2472.5</td>
<td>2188.4 Hz</td>
</tr>
<tr>
<td>2</td>
<td>2502.3</td>
<td>2680.2 Hz</td>
</tr>
<tr>
<td>3</td>
<td>3158.0</td>
<td>3072.0 Hz</td>
</tr>
<tr>
<td>4</td>
<td>3300.0</td>
<td>3460.1 Hz</td>
</tr>
<tr>
<td>5</td>
<td>4503.2</td>
<td>3790.4 Hz</td>
</tr>
</tbody>
</table>

Table 4. The frequency results using fine mesh comparing with the analytical results in five modes.
<table>
<thead>
<tr>
<th>Mode Number</th>
<th>Frequency (Hertz)</th>
<th>Analytical solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2202.0</td>
<td>2188.4 Hz</td>
</tr>
<tr>
<td>2</td>
<td>2704.7</td>
<td>2680.2 Hz</td>
</tr>
<tr>
<td>3</td>
<td>3076.6</td>
<td>3072.0 Hz</td>
</tr>
<tr>
<td>4</td>
<td>3512.8</td>
<td>3460.1 Hz</td>
</tr>
<tr>
<td>5</td>
<td>3858.5</td>
<td>3790.4 Hz</td>
</tr>
</tbody>
</table>

*Rotated $Q_1$ approximation* results:

**Table 5.** The frequency results using fine mesh comparing with the analytical results in five modes.

<table>
<thead>
<tr>
<th>Eigenvalues using full mass matrix</th>
<th>Eigenvalues using lumped mass matrix</th>
<th>Analytical solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2186.8 Hz</td>
<td>2180.9 Hz</td>
<td>2188.4 Hz</td>
</tr>
<tr>
<td>2674.4 Hz</td>
<td>2664.3 Hz</td>
<td>2680.2 Hz</td>
</tr>
<tr>
<td>3066.9 Hz</td>
<td>3063.1 Hz</td>
<td>3072.0 Hz</td>
</tr>
<tr>
<td>3454.7 Hz</td>
<td>3430.2 Hz</td>
<td>3460.1 Hz</td>
</tr>
<tr>
<td>3783.5 Hz</td>
<td>3752.1 Hz</td>
<td>3790.4 Hz</td>
</tr>
</tbody>
</table>
Figure 3. The displacements are shown in both coarse and fine meshes. To find the displacement we use full and lumped matrixes.

Figure 3. Coarse mesh with full mass matrix

Figure 3. Coarse mesh with lumped mass matrix
Figur 3. Fine mesh with full mass matrix

Figur 3. Fine mesh with lumped mass matrix
Figure 4. The metal beam oscillating in its 5 different modes.
Figure 5. Estimated error for the frequencies using full and lumped mass matrices.
As it is illustrated in the figure, the errors of the full mass matrix are much lower than the errors of the lumped mass matrix.
**Figure 6.** Frequency results using the full and lumped mass matrices.

In this figure we have compared both results with the analytical results, as it is shown, the frequencies that are derived from the full mass matrix are closer to the analytical ones.

Comparing the time to compute the eigenvalues with the use of the full and lumped mass matrices:

**Table 6. The time results using the fine and coarse mesh**

<table>
<thead>
<tr>
<th>Mesh Type</th>
<th>Time using full mass matrix</th>
<th>Time using lumped mass matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse mesh</td>
<td>0.46236 seconds</td>
<td>0.3486 seconds</td>
</tr>
<tr>
<td>Fine mesh</td>
<td>10.7637 seconds</td>
<td>9.8909 seconds</td>
</tr>
</tbody>
</table>
6. Conclusion

In this thesis, we have examined the feasibility of using a nonconforming rotated Q1 method on a linear elasticity problem using tetrahedron elements. We have designed and implemented the algorithm for solving a linear elastic problem using the rotated Q1 method and compared the results with the analytical results, SolidWorks and COMSOL results. The conclusion we can make is that the rotated Q1 method generates better frequency results compared to SolidWorks and it generates a little bit lower frequencies than the analytical eigenvalue method and the other commercial software (COMSOL). We also estimated the displacements using full and lumped mass matrices. The displacements and frequencies show that the full mass matrix is more accurate but takes more time to run. In the second part of the thesis, we have implemented a time dependent linear elastic method on a beam in order to see how the different explicit and implicit methods work. We have seen that in the rotated Q1 approximation, the explicit method works. That is because in the Q1 approximation the lumped mass matrix is positive definite. This method certainly deserves more attention and should be used for future computations.
7. References


8. Appendix

The code for part A of thesis is as follows:

```matlab
%% -----------------------------------------------
%% Main program
%% -----------------------------------------------

clear all
clc
global nu E lambda mu rho
rho = 7850;
E = 200e9;
nu = 0.33;

lambda=nu*E/(1+nu)/(1-2*nu);mu=E/2/(1+nu);

%% Mesh import
nodes = importdata('nodes.txt')+1;
coord = importdata('coord.txt'); %The textfile is a part of a exported meshfile from comsol

% Boundary
ind1min=find(coord(:,1) <= 1.e-9);
ind1max=find(coord(:,1) >= 1-1.e-9);

ind2min=find(coord(:,2) <= 1.e-9);
ind2max=find(coord(:,2) >= 1-1.e-9);

ind3min=find(coord(:,3) <= 1.e-9);
ind3max=find(coord(:,3) >= 1-1.e-9);

coord(ind1min,1)=0;
coord(ind1max,1)=1;
coord(ind2min,2)=0;
coord(ind2max,2)=1;
coord(ind3min,3)=0;
coord(ind3max,3)=1;

%%
tris3d = nodes;

x3d=coord(:,1);y3d=coord(:,2);z3d=coord(:,3);

[xed,yed,zed,trisq]=edges3d(tri3d,x3d,y3d,z3d);

[uxFull,uyFull,uzFull,uxLumped,uyLumped,uzLumped,eig]=solveeigen(trisq,x3d,y3d,zd3d)
```
3d, xed, yed, zed;

%% Viz Full mass
disp(' ')
disp('Solving displacements...')

eps=2;
c2=coord;
c2(:,1)=c2(:,1)+eps*uxFull;
c2(:,2)=c2(:,2)+eps*uyFull;
c2(:,3)=c2(:,3)+eps*uzFull;
disp('Done!')

figure(1);clf,tetramesh(nodes,c2),title('Displacements using full matrix'),hold on

%% Viz Lumped mass
disp(' ')
disp('Solving displacements...')

eps=2;
c2=coord;
c2(:,1)=c2(:,1)+eps*uxLumped;
c2(:,2)=c2(:,2)+eps*uyLumped;
c2(:,3)=c2(:,3)+eps*uzLumped;
disp('Done!')

figure(2);clf,tetramesh(nodes,c2),title('Displacements using lumped mass matrix'),hold on

%% -----------------------------------------------
%% solveeigen
%% -----------------------------------------------

function [uxFull,uyFull,uzFull,uxLumped,uyLumped,uzLumped,eigenvalue]=solveeigen(nodes,xnod,ynod,znod,xed,yed,zed)
global nu E rho
bweps=zeros(9,3*6);
 bw=zeros(3,3*6);
bwdiv=zeros(1,3*6);
lambda=nu*E/(1+nu)/(1-2*nu);
mu=E/2/(1+nu);
nu=lambda/(2*(lambda+mu));
nele=size(nodes,1);
neq=3*length(xed);
nsize=6*3*nele;
 f=zeros(neq,1);
row = zeros(nsise, 1);
col = row;
val = row;
up = 0;
valm = row;

alpha = pi/(max(xnod) - min(xnod));

disp(' ')
disp(['Number of elements: ', num2str(nele)])
disp(' ')

%% Assembling Stiffness and Mass matrices
disp('Assembling Stiffness and Mass matrices...                ')

nv = length(xnod);
sq2 = sqrt(2);
for iel = 1:nele
    a = (iel*100)/nele;
    fprintf('%3.0f%% done...', a)

    iv = nodes(iel, 5:end) - nv; ivv = nodes(iel, 1:4);
    ieqs = [3*iv(1) - 2, 3*iv(1) - 1, 3*iv(1), 3*iv(2) - 2, 3*iv(2) - 1, ... 
            3*iv(2), 3*iv(3) - 2, 3*iv(3) - 1, 3*iv(3), 3*iv(4) - 2, 3*iv(4) - 1, 3*iv(4), ... 
            3*iv(5) - 2, 3*iv(5) - 1, 3*iv(5), 3*iv(6) - 2, 3*iv(6) - 1, 3*iv(6)];
    [gcx, gcy, gcz, gv] = tetgauc(xnod(ivv), ynod(ivv), znod(ivv), 3);
    % [gcx, gcy, gcz, gv] = tetraquad(5, [xnod(ivv), ynod(ivv), znod(ivv)]);
    mele = zeros(3*6); sele = zeros(3*6);
    for igau = 1:length(gv)
        [fi, fix, fiy, fiz, detj] = basehansbo(gcx(igau), gcy(igau), gcz(igau), xnod(ivv), ynod(ivv), znod(ivv));
        bweps(1, 1:3:end) = fix;
        bweps(2, 2:3:end) = fiy;
        bweps(3, 3:3:end) = fiz;
        bweps(4, 1:3:end) = fiy/sq2;
        bweps(4, 2:3:end) = fix/sq2;
        bweps(5, 1:3:end) = fiz/sq2;
        bweps(5, 3:3:end) = fix/sq2;
        bweps(6, 2:3:end) = fiz/sq2;
        bweps(6, 3:3:end) = fiy/sq2;
        bwdiv(1:3:end) = fix;
        bwdiv(2:3:end) = fiy;
        bwdiv(3:3:end) = fiz;
        bw(1, 1:3:end) = fi;
bw(2,2:3:end)=fi;
bw(3,3:3:end)=fi;

sele=sele+gv(igau)*detj*(2*mu*bweps'*bweps+lambda*bwdiv'*bwdiv);
mele=mele+gv(igau)*rho*detj*bw'*bw;
end

len = length(ieqs);
X = ieqs(:, ones(1, len));
Y = X';
nn = len * len;
lo = up + 1;
up = up + nn;
row(lo:up) = X(:);
col(lo:up) = Y(:);
val(lo:up) = sele(:);
valm(lo:up) = mele(:);
end
disp('Done!')
S = sparse(row(1:up), col(1:up), val(1:up), neq, neq);
M = sparse(row(1:up), col(1:up), valm(1:up), neq, neq);

%% Boundary condition
ind1=find(xed==min(xed) | xed==max(xed));
ind2=find(yed==min(yed) | yed==max(yed));
ind3=find(zed==min(zed) | zed==max(zed));
ind=[3*ind1-2;3*ind2-1;3*ind3];

free=setdiff([1:neq],ind);
S=S(free,free);
M=M(free,free);

%% Lumped mass
MLumped = diag(sum(M'));

%% Analytical
alpha = pi / (max(xnod)-min(xnod));
neig= 5;
e=analyteig(alpha,neig);
omega = sort(e*(E/(2*(1+nu))/(1-2*nu)/rho));
disp(' ')
disp(' ')
disp(' ')
disp('Analytical solution')
freqAnalytical = (sqrt(omega)/(2*pi))'
%% solving eigenvalue, full mass

disp('')
disp('Solving eigenvalues with full mass matrix...') tic
[v,d,flag]=eigs(S,M,20,'SM');
timeFull  = toc;
a=diag(d);
eigcomp=sort(a');
ind=find(a>0);
a=a(ind);v=v(:,ind);
ind=find(a==min(a));ind=ind(1); v=v(:,ind);
eigenvalue=min(a);

omega = sqrt(a); %Eigenvalues or the natural frequencies in rad/sec
disp('')
disp('Eigenvalues using full mass matrix')
FreqFull = omega/(2*pi)
MinFreq = min(FreqFull)
save('FullMassMatrixResultsFull.mat','FreqFull');
disp(['time for Full mass matrix: ',num2str(timeFull),', seconds.'])

%% solving eigenvalue, lumped mass

disp('')
disp('')
disp('')
disp('')
disp('')
disp('Solving eigenvalues with lumped mass matrix...')
tic
[vLumped,d,flag]=eigs(S,MLumped,20,'SM');
timeLumped  = toc;
aLumped=diag(d);
omega = sqrt(aLumped); %Eigenvalues or the natural frequencies in rad/sec
disp('')
disp('Eigenvalues using lumped mass matrix')
FreqLumped = omega/(2*pi)
MinFreqLumped = min(FreqLumped)
save('LumpedMassMatrixResultsFull.mat','FreqLumped');
disp(['time for Lumped mass matrix: ',num2str(timeLumped),', seconds.'])

ind=find(aLumped>0);
aLumped=aLumped(ind);vLumped=vLumped(:,ind);
The code for part A of thesis is as follows:

```matlab
%% Main program

clear all
clc
global nu E lambda mu rho

rho = 7850;
E = 200e9;
nu = 0.33;

lambda=nu*E/(1+nu)/(1-2*nu);mu=E/2/(1+nu);

%% Mesh import
nodes = importdata('nodes_p.txt')+1;
coord = importdata('coord_p.txt'); %The textfile is a part of a exported meshfile from comsol

% Boundary
ind1min=find(coord(:,1) <= 1.e-9);
ind1max=find(coord(:,1) >= 1-1.e-9);
```
coord(ind1min,1)=0;
coord(ind1max,1)=1;

tri3d = nodes;

x3d=coord(:,1);y3d=coord(:,2);z3d=coord(:,3);

[xed,yed,zed,trisq]=edges3d(tri3d,x3d,y3d,z3d);

[uxFull,uyFull,uzFull,uxLumped,uyLumped,uzLumped,eigFull,U,N]=solveeigen_PMetal(
    trisq,x3d,y3d,z3d,xed,yed,zed);

%% Implicit timendependent viz

for n = 1:2
    eps = 0.5;
    c2=coord;

    for i = 1:N+1
        ux = U(:,1,i);
        uy = U(:,2,i);
        uz = U(:,3,i);
        c2(:,1)=c2(:,1)+eps*ux;
        c2(:,2)=c2(:,2)+eps*uy;
        c2(:,3)=c2(:,3)+eps*uz;
        fig = figure(5);
        clf,tetramesh(nodes,c2), axis off;
    end
end

function
[uxFull,uyFull,uzFull,uxLumped,uyLumped,uzLumped,eigFull,U,N]=solveeigen_PMetal(nodes,x
    nod,ynod,znod,xed,yed,zed)
    global nu E rho
    bweps=zeros(9,3*6);
    bw=zeros(3,3*6);
    bwdiv=zeros(1,3*6);
    lambda=nu*E/(1+nu)/(1-2*nu);mu=E/2/(1+nu);
    nu=nu/lambda/(2*(lambda+mu));
    nele=size(nodes,1);
    neq=3*length(xed);
    nsize=6*3*nele;
    f=zeros(neq,1);
    row = zeros(nsize, 1);
col = row;
val = row;
up = 0;
valm = row;

alpha = pi / (max(xnod) - min(xnod));

disp(' ')
disp(['Number of elements: ', num2str(nele)])
disp(' ')

%% Assembling Stiffness and Mass matrices
disp('Assembling Stiffness and Mass matrices...')

nv = length(xnod);
sq2 = sqrt(2);
for iel = 1:nele
    a = (iel*100)/nele;
    fprintf('%3.0f%% done...', a)
    iv = nodes(iel, 5:end) - nv; ivv = nodes(iel, 1:4);
    ieqs = [3*iv(1)-2, 3*iv(1)-1, 3*iv(1), 3*iv(2)-2, 3*iv(2)-1, ...
            3*iv(2), 3*iv(3)-2, 3*iv(3)-1, 3*iv(3), 3*iv(4)-2, 3*iv(4)-1, 3*iv(4), ...
            3*iv(5)-2, 3*iv(5)-1, 3*iv(5), 3*iv(6)-2, 3*iv(6)-1, 3*iv(6)]';
    [gcx, gcy, gcz, gv] = tetgauc(xnod(ivv), ynod(ivv), znod(ivv), 3);
    % [gcx, gcy, gcz, gv] = tetraquad(5, [xnod(ivv), ynod(ivv), znod(ivv)]);
    mele = zeros(3*6); sele = zeros(3*6);
    for igau = 1:length(gv)

        [fi, fix, fiy, fiz, detj] = basehansbo(gcx(igau), gcy(igau), gcz(igau), xnod(ivv), ynod(ivv), znod(ivv)));

        bweps(1, 1:3:end) = fix;
        bweps(2, 2:3:end) = fiy;
        bweps(3, 3:3:end) = fiz;
        bweps(4, 1:3:end) = fiy/sq2;
        bweps(4, 2:3:end) = fix/sq2;
        bweps(5, 1:3:end) = fiz/sq2;
        bweps(5, 3:3:end) = fix/sq2;
        bweps(6, 2:3:end) = fiz/sq2;
        bweps(6, 3:3:end) = fiy/sq2;
        bwdiv(1:3:end) = fix;
        bwdiv(2:3:end) = fiy;
        bwdiv(3:3:end) = fix;

        bw(1, 1:3:end) = fi;
bw(2,2:3:end)=fi;
bw(3,3:3:end)=fi;

sele=sele+gv(igau)*detj*(2*mu*bweps'*bweps+lambda*bwdv'*bwdiv);
% sele=sele+gv(igau)*detj*((1-2*nu)*bweps'*bweps+bwdiv'*bwdiv);
mele=mele+gv(igau)*rho*detj*bw'*bw;
end

len = length(ieqs);
X = ieqs(:, ones(1, len));
Y = X';
nn = len * len;
lo = up + 1;
up = up + nn;
row(lo:up) = X(:);
col(lo:up) = Y(:);
val(lo:up) = sele(:);
valm(lo:up) = mele(:);
end
disp('Done!')
S = sparse(row(1:up), col(1:up), val(1:up), neq, neq);
M = sparse(row(1:up), col(1:up), valm(1:up), neq, neq);

%% Boundary condition
ind1=find(xed==min(xed) | xed==max(xed));
ind=sort([3*ind1-2;3*ind1-1;3*ind1]);
free=setdiff(1:neq,ind);
S=S(free,free);
M=M(free,free);

%% solving eigenvalue, full mass
disp(' ') disp('Solving eigenvalues with full mass matrix...') tic
[v,d,flag]=eigs(S,M,20,'SM');
timeFull = toc;
a=diag(d);
eigFull=sort(a');
ind=find(a>0);
a=a(ind);v=v(:,ind);
ind=find(a==min(a))+6;ind=ind(1);%Next mode
%ind=7;
v=v(:,ind);
eigenvalue=min(a);
omega = sqrt(a); %Eigenvalues or the natural frequencies in rad/sec
T = 2*pi/min(omega);
disp(' ')
disp('Eigenvalues using full mass matrix')
MinFreq = min(omega/(2*pi))
Freq = omega/(2*pi)
disp(['time for Full mass matrix: ',num2str(timeFull),', seconds.'])

%% L2-projection, full mass

disp(' ')
disp(' ')
disp(' ')
disp('Projecting midpoint-displacements to corners using full mass matrix...')

[uxFull,uyFull,uzFull]=l2Project_PMetal(v,neq,free,nodes,xnod,ynod,znod);
disp('Done!')

eps = 2;
coord = [xnod,ynod,znod];
c2=coord;

ux = uxFull;
uy = uyFull;
uz = uzFull;
c2(:,1)=c2(:,1)+eps*ux;
c2(:,2)=c2(:,2)+eps*uy;
c2(:,3)=c2(:,3)+eps*uz;
figure(3);clf,tetramesh(nodes,c2);

%% TimeDependent
kncrit = 2/sqrt(max(a));
N = ceil(T/kncrit);
% N = 1;

w=zeros(neq,1);
w(free)=v;
[ux,uy,uz]=l2Project_PMetal_timeD(w,neq,free,nodes,xnod,ynod,znod);
eps=2;
coord = [xnod,ynod,znod];
c2=coord;

c2(:,1)=c2(:,1)+eps*ux;
c2(:,2)=c2(:,2)+eps*uy;
c2(:,3)=c2(:,3)+eps*uz;
figure(5);tetramesh(nodes,c2)

% iterations - implicit
velocity = zeros(neq,1);
v0 = 0;
w0 = 0;
v0n = v;
vel_n = 0*vn;
kn = T/N;

Theta = 1/2;

U = zeros(length(ux),3,N+1);

U(:,:,1) = [ux,uy,uz];
disp(' ')
disp(' ')
disp(['Computing ', num2str(N), ' displacements using l2project... '])

for n = 1:N
    a = (n*100)/N;
    fprintf('%3.0f%% done...', a)
    vtemp = (M+(kn^2)*(Theta^2)*S)/(M-((kn^2)*(Theta-Theta^2))*S)*vel_n - kn*S*vn);
    vn = vn + kn * (Theta * vtemp + (1 - Theta) * vel_n);
    vel_n = vtemp;
    w(free) = vn;
    velocity(free) = vel_n;
    [ux,uy,uz]=l2Project_PMetal_timeD(w,neq,free,nodes,xnod,ynod,znod);
    U(:,:,n+1)=[ux,uy,uz];
end

The code for part B of thesis is as follows:

clear all
clc
global nu E lambda mu rho

%nu=0.3;E=1e3;
rho = 7850;
E = 200e9;
nu = 0.33;

lambda=nu*E/(1+nu)/(1-2*nu);mu=E/2/(1+nu);

%% Mesh import
nodes = importdata('nodes_q.txt')+1;
coord = importdata('coord_q.txt'); %The textfile is a part of a exported meshfile from comsol

% Boundary
ind1min=find(coord(:,1) <= 1.e-9);
% ind2max=find(coord(:,2) >= 1-e-9);
% ind2min=find(coord(:,2) <= 1.e-9);
% ind3min=find(coord(:,3) <= 1.e-9);
% ind3max=find(coord(:,3) >= 1-1.e-9);

coord(ind1min,1)=0;
coord(ind1max,1)=1;
% coord(ind2min,2)=0;
% coord(ind2max,2)=1;
% coord(ind3min,3)=0;
% coord(ind3max,3)=1;
%

tri3d = nodes;
x3d=coord(:,1);y3d=coord(:,2);z3d=coord(:,3);
[xed,yed,zed,trisq]=edges3d(tri3d,x3d,y3d,z3d);

U=solveeigen_PMetal(trisq,x3d,y3d,z3d,xed,yed,zed);

displ = U;
save('displMode5LeapFrog1.mat','displ');

The code for the subfunctions which we have used for the thesis is as follows:

```matlab
def function U=solveeigen_PMetal(nodes,xnod,ynod,znod,xed,yed,zed)
  global nu E rho
  bweps=zeros(9,3*6);
  bw=zeros(3,3*6);
  bwdiv=zeros(1,3*6);
  lambda=nu*E/(1+nu)/(1-2*nu);mu=E/2/(1+nu);
  nu=lambda/(2*(lambda+mu));
  nele=size(nodes,1);
  neq=3*length(xed);
  nsize=6*3*nele;
  f=zeros(neq,1);
  row = zeros(nsize, 1);
  col = row;
  val = row;
  up = 0;
  valm = row;
  alpha=pi/(max(xnod)-min(xnod));
  disp(' ')
  disp(['Number of elements: ', num2str(nele)])
  disp(' ')
  % Assembling Stiffness and Mass matrices
  disp('Assembling Stiffness and Mass matrices...
  
  nvs=length(xnod);
  sq2=sqrt(2);
  for iel=1:nele
    a = (iel*100)/nele;
    fprintf('

    iv=nodes(iel,5:end)-nv;ivv=nodes(iel,1:4);
    ieqs=[3*iv(1)-2,3*iv(1)-1,3*iv(1),3*iv(1)+1,3*iv(2)-2,3*iv(2)-1,...
    3*iv(2),3*iv(3)-2,3*iv(3)-1,3*iv(3),3*iv(3)+1,3*iv(4)-2,3*iv(4)-1,3*iv(4),...
```
\[3 \cdot \text{iv}(5) - 2, 3 \cdot \text{iv}(5) - 1, 3 \cdot \text{iv}(6) - 2, 3 \cdot \text{iv}(6) - 1, 3 \cdot \text{iv}(6)\]  
\[
\text{gcx}, \text{gcy}, \text{gcz}, \text{gv} = \text{tetgauc}([\text{xnod}(\text{ivv}), \text{ynod}(\text{ivv}), \text{znod}(\text{ivv})], 3); \\
\text{mele} = \text{zeros}(3 \cdot 6); \text{sele} = \text{zeros}(3 \cdot 6); \\
\text{for } \text{igau} = 1 : \text{length(gv)} \\
\]
\[
[\text{fi}, \text{fix}, \text{fiy}, \text{fiz}, \text{detj}] = \text{basehansbo(gcx(igau), gcy(igau), gcz(igau), xnod(ivv), ynod(ivv), znod(ivv))}; \\
\text{bweps}(1, 1:3:end) = \text{fix}; \\
\text{bweps}(2, 2:3:end) = \text{fiy}; \\
\text{bweps}(3, 3:3:end) = \text{fiz}; \\
\text{bweps}(4, 1:3:end) = \text{fiy} / \sqrt{2}; \\
\text{bweps}(4, 2:3:end) = \text{fix} / \sqrt{2}; \\
\text{bweps}(5, 1:3:end) = \text{fiz} / \sqrt{2}; \\
\text{bweps}(5, 3:3:end) = \text{fiy} / \sqrt{2}; \\
\text{bweps}(6, 2:3:end) = \text{fiz} / \sqrt{2}; \\
\text{bweps}(6, 3:3:end) = \text{fiy} / \sqrt{2}; \\
\text{bwdiv}(1:3:end) = \text{fix}; \\
\text{bwdiv}(2:3:end) = \text{fiy}; \\
\text{bwdiv}(3:3:end) = \text{fiz}; \\
\text{bw}(1, 1:3:end) = \text{fi}; \\
\text{bw}(2, 2:3:end) = \text{fi}; \\
\text{bw}(3, 3:3:end) = \text{fi}; \\
\text{sele} = \text{sele} + \text{gv}(\text{igau}) \cdot \text{detj} \cdot (2 \cdot \mu \cdot \text{bweps}' \cdot \text{bweps} + \lambda \cdot \text{bwdiv}' \cdot \text{bwdiv}); \\
\text{mele} = \text{mele} + \text{gv}(\text{igau}) \cdot \rho \cdot \text{detj} \cdot \text{bw}' \cdot \text{bw}; \\
\text{end} \\
\]
\[
\text{len} = \text{length(ieqs)}; \\
\text{X} = \text{ieqs(:, ones(1, len))}; \\
\text{Y} = \text{X'}; \\
\text{nn} = \text{len} \cdot \text{len}; \\
\text{lo} = \text{up} + 1; \\
\text{up} = \text{up} + \text{nn}; \\
\text{row(lo:up)} = \text{X}(::); \\
\text{col(lo:up)} = \text{Y}(::); \\
\text{val(lo:up)} = \text{sele}(::); \\
\text{valm(lo:up)} = \text{mele}(::); \\
\text{end} \\
\text{disp('Done!')} \\
\text{S} = \text{sparse(row(1:up), col(1:up), val(1:up), neq, neq);} \\
\text{M} = \text{sparse(row(1:up), col(1:up), valm(1:up), neq, neq);} \\
\%
\text{M} = \text{sum(M, 2);} \\
\%
\text{M} = \text{sparse(diag(M));} \\
\%
\text{Boundary condition} \\
\%
\text{indlmin} = \text{find(xed <= 1.e-9)}; \\
\%
\text{indlmax} = \text{find(xed >= (1-1.e-9)}; \\
\text{ind1} = \text{find(xed == min(xed) | xed == max(xed));} \\
\%
\text{ind2} = \text{find(yed == min(yed) | yed == max(yed));} \\
\%
\text{ind3} = \text{find(zed == min(zed) | zed == max(zed));} \\
\%
\text{ind1} = [\text{indlmin}; \text{indlmax}]; \\
\text{ind} = \text{sort([3*ind1-2; 3*ind1-1; 3*ind1]);} \\
\%
\text{free} = \text{setdiff(1:neq, ind)}; \\
\%
\text{S} = \text{S(free, free)}; \\
\%
\text{M} = \text{M(free, free)}; \\
\%
\text{solving eigenvalue, full mass} \\
\text{disp(' ')} \\
\text{disp('Solving eigenvalues with full massmatrix...')} \\
\%
\text{tic} \\
\%
\text{[v, d] = eig(full(S), full(M)); flag = 0; }
[v, d, flag] = eigs(S, M, 20, 'SM');
timeFull = toc;
a = diag(d);
eigFull = sort(a);
ind = find(a > 0);
a = a(ind);
v = v(:, ind);
ind = find(a == min(a)) + 0; ind = ind(1); % first mode
% ind = 7;
v = v(:, ind);
eigenvalue = min(a);

omega = sqrt(a); % Eigenvalues or the natural frequencies in rad/sec
T = 2 * pi / min(omega);
disp('')
disp('Eigenvalues using full mass matrix')
MinFreq = min(omega / (2 * pi))
Freq = omega / (2 * pi)
disp(['time for Full mass matrix: ', num2str(timeFull), ', seconds.'])

% dev
uxFull = 0; uyFull = 0; uzFull = 0;
uxLumped = 0; uyLumped = 0; uzLumped = 0;
U = 1; N = 1;

% Pre viz
w = zeros(neq, 1); % displacements
w(free) = v; % free displacements in the midpoints
[ux, uy, uz] = l2Project_PMetal_timeD(w, neq, free, nodes, xnod, ynod, znod);

coord = importdata('coord_q.txt');
ind1min = find(coord(:, 1) <= 1.0e-9);
ind1max = find(coord(:, 1) >= 1.0e-9);
coord(ind1min, 1) = 0;
coord(ind1max, 1) = 1;
c2 = coord;
esp = 2;
c2(:, 1) = c2(:, 1) + esp * ux;
c2(:, 2) = c2(:, 2) + esp * uy;
c2(:, 3) = c2(:, 3) + esp * uz;
fig = figure(1);
c1f, tetramesh(nodes, c2), axis off;

% TimeDependent
kncrit = 2 / sqrt(max(a));
% kncrit = 1.0e-11;
N = ceil(T / kncrit);
% N = 30;

% Start values
w = zeros(neq, 1); % displacements
w(free) = v; % free displacements in the midpoints
[ux, uy, uz] = l2Project_PMetal_timeD(w, neq, free, nodes, xnod, ynod, znod); % Displacements in the corners of the tetrahedron

% start values
un = v; % start displacements
vn = 0 * un; % start velocity
kn = kncrit; % kritical timestep
% M = diag(sum(M')); % Use if explicit method
% Theta = 1; % implicit
% % Theta = 1/2; % C-N
% % Theta = 0; % explicit

% Leapfrog method
% start values
u0 = v; %start displacement
% u-1 is needed to compute u1.

% acceleration at time t = 0;
% M * a0 = f0 - S * u0
% f0 = 0
% a0 = M \ (-S * u0)
a0 = M \ (-S * u0);

v0 = 0 * u0; %the velocity is zero at t = 0;
% u-1 = u0 - kn * v0 + (kn^2/2)*a0
% u-1 is u prev
un_prev = u0 - kn * v0 + (kn^2/2)*a0;

un = u0;
v0 = v0;

U = zeros(length(ux),3,N+1); %displacements matrix
U(:,:,1) = [ux,uy,uz]; % Start displacement
disp(' ')
for n = 1:N
a = (n*100)/N;
fprintf('%3.0f%% done...',a)

% vn1 = (M+(kn^2)*(Theta^2)*S)/(M-((kn^2)*(Theta-Theta^2))*S)*vn - kn*S*un);
% un1 = un + kn * Theta * vn1 + (1 - Theta) * vn;
% un = un1;
% vn = vn1;
% w(free) = un;

% Leapfrog method - multistep method
% M * un1 = kn^2*fn+(2*M-kn^2*S)*un-M*un_prev
% fn = 0; so
% M * un1 = (2*M-kn^2*S)*un-M*un_prev
% and
% un1 = M \ ((2*M-kn^2*S)*un-M*un_prev)
% vn = (un1 - un_prev)/(2*kn);
% un_prev=un;
% un = un1;

w(free) = un;
[ux,uy,uz]=l2Project_PMetal_timeD(w,neq,free,nodes,xnod,ynod,znod);
U(:,:,n+1)=[ux,uy,uz];% Saving the displacements into a 3D matrix to visualize later
end

The code for the subfunctions which we have used for the thesis is as follows:

function [fi,fixy,fiy,fiz,vol]=basehansbo(x,y,z,xc,yc,zc)
% Scalar evaluation of basis functions
x1=[xc(1);yc(1);zc(1)];x2=[xc(2);yc(2);zc(2)];x3=[xc(3);yc(3);zc(3)];x4=[xc(4);y c(4);zc(4)];
A=[(x2-x1+x3-x4)/2,(x3-x1-x2+x4)/2,(x2-x1-x3+x4)/2];J=A';
xsiv=A\([x;y;z]-x1); xsiv=A\([x;y;z]-x1); xsi=xsiv(1);eta=xsiv(2);zeta=xsiv(3);
fi=[(2 + 2*xsi - 2*xsi^2 - 7*eta + 4*eta^2 + 2*zeta - 2*zeta^2)/3;
\[
\begin{align*}
(-1 - \xi + 4\xi^2 + 2\eta - 2\eta^2 + 2\zeta - 2\zeta^2)/3; \\
(-1 + 2\xi - 2\xi^2 - \eta + 4\eta^2 + 2\zeta - 2\zeta^2)/3; \\
(-7\xi + 4\xi^2 + 2(1 + \eta - \eta^2 + \zeta - \zeta^2))/3; \\
(2 + 2\xi - 2\xi^2 + 2\eta - 2\eta^2 - 7\zeta + 4\zeta^2)/3; \\
(-1 + 2\xi - 2\xi^2 + 2\eta - 2\eta^2 - \zeta + 4\zeta^2)/3
\end{align*}
\]

\[
\begin{align*}
\text{fixi} &= \left[2 - 4\xi, -1 + 8\xi, 2 - 4\xi, -7 + 8\xi, 2 - 4\xi, 2 - 4\xi\right]/3; \\
\text{fieta} &= \left[-7 + 8\eta, 2 - 4\eta, -1 + 8\eta, 2 - 4\eta, 2 - 4\eta, 2 - 4\eta\right]/3; \\
\text{fizeta} &= \left[2 - 4\zeta, 2 - 4\zeta, 2 - 4\zeta, 2 - 4\zeta, -7 + 8\zeta, -1 + 8\zeta\right]/3;
\end{align*}
\]

\[
\begin{align*}
\text{xv} &= \left[(x(1)+x(2))/2, (x(2)+x(3))/2, (x(4)+x(3))/2, (x(1)+x(4))/2, (x(1)+x(3))/2, (x(2)+x(4))/2\right]; \\
\text{yv} &= \left[(y(1)+y(2))/2, (y(2)+y(3))/2, (y(4)+y(3))/2, (y(1)+y(4))/2, (y(1)+y(3))/2, (y(2)+y(4))/2\right]; \\
\text{zv} &= \left[(z(1)+z(2))/2, (z(2)+z(3))/2, (z(4)+z(3))/2, (z(1)+z(4))/2, (z(1)+z(3))/2, (z(2)+z(4))/2\right];
\end{align*}
\]

\[
\begin{align*}
\text{fiv} &= \text{J} \left[\text{fixi}; \text{fieta}; \text{fizeta}\right]; \\
\text{fiy} &= \text{fiv}(2,:); \\
\text{fix} &= \text{fiv}(3,:); \\
\text{fix} &= \text{fiv}(1,:);
\end{align*}
\]

\[
\begin{align*}
\text{v} &= \text{cross}([x(2)-x(1), y(2)-y(1), z(2)-z(1)], [x(3)-x(1), y(3)-y(1), z(3)-z(1)]); \\
\text{vol} &= \text{abs} \left(\text{dot}(\text{v}, [x(4)-x(1), y(4)-y(1), z(4)-z(1)])\right)/6;
\end{align*}
\]

% --------------------------------------------
% edges3d
% --------------------------------------------

function [xed,yed,zed,trisq]=edges3d(tri,xnod,ynod,znod)
nele=size(tri,1);
trisq=zeros(nele,10);
trisq(:,1:4)=tri;
nno=length(xnod);
for iel=1:nele
    ed=sort([tri(iel,1:2);tri(iel,2:3);tri(iel,4),tri(iel,3);tri(iel,1),tri(iel,4);tri(iel,1),tri(iel,3);tri(iel,2),tri(iel,4)],2);
    if(iel==1)
        edges=ed;
    else
        edges=[edges,ed];
    end
    trisq(iel,5:10)=nno+[1:6];
end
for i=1:6
    e=ed(i,:);  
    ind=find(edges(:,1)==e(1) & edges(:,2)==e(2));
    if(isempty(ind))
        edges=[edges;e];
        trisq(iel,4+i)=nno+size(edges,1);
    else
        trisq(iel,4+i)=nno+ind;
    end
end
xed=(xnod(edges(:,1))+xnod(edges(:,2)))/2;
yed=(ynod(edges(:,1))+ynod(edges(:,2)))/2;
zed=(znod(edges(:,1))+znod(edges(:,2)))/2;