STUDY OF THE VIBRATION BEHAVIOUR OF ROTATORY BLADES USING THE FINITE ELEMENT METHOD

FINAL DEGREE THESIS
AEROSPACE TECHNOLOGIES ENGINEERING

by

Pere Frontera Pericàs

Director: Hernández Ortega, Joaquín A.
Co-Director: Cante Teran, Juan Carlos
Project duration: February 13 – June 10, 2019

— Report attachment —
# Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong> Theoretical frame</td>
<td>5</td>
</tr>
<tr>
<td>A.1 Introduction to elasticity</td>
<td>5</td>
</tr>
<tr>
<td>A.1.1 The elastic problem</td>
<td>5</td>
</tr>
<tr>
<td>A.1.2 Generalised Hooke's law</td>
<td>7</td>
</tr>
<tr>
<td>A.2 Classical analysis methods</td>
<td>9</td>
</tr>
<tr>
<td>A.2.1 Energy methods</td>
<td>9</td>
</tr>
<tr>
<td>A.2.2 Graphical methods</td>
<td>11</td>
</tr>
<tr>
<td>A.2.3 Matrix methods</td>
<td>12</td>
</tr>
<tr>
<td><strong>B</strong> Elemental matrix computation</td>
<td>13</td>
</tr>
<tr>
<td>B.1 The static stiffness matrix</td>
<td>13</td>
</tr>
<tr>
<td>B.2 The force vector</td>
<td>14</td>
</tr>
<tr>
<td>B.2.1 Body forces</td>
<td>14</td>
</tr>
<tr>
<td>B.2.2 Traction forces</td>
<td>14</td>
</tr>
<tr>
<td>B.3 The mass matrix</td>
<td>16</td>
</tr>
<tr>
<td>B.4 The static damping matrix</td>
<td>16</td>
</tr>
<tr>
<td>B.5 The rotation matrices</td>
<td>16</td>
</tr>
<tr>
<td><strong>C</strong> Example of vectorised assembly</td>
<td>17</td>
</tr>
<tr>
<td>C.1 Mass matrix vectorised assembly</td>
<td>17</td>
</tr>
<tr>
<td>C.2 Generalisation to rotation matrices</td>
<td>20</td>
</tr>
<tr>
<td><strong>D</strong> Raw simulation results</td>
<td>21</td>
</tr>
<tr>
<td>D.1 Two-dimensional cantilever beam</td>
<td>21</td>
</tr>
<tr>
<td>D.1.1 Natural modes and frequencies</td>
<td>21</td>
</tr>
<tr>
<td>D.1.2 Constant angular velocity</td>
<td>22</td>
</tr>
<tr>
<td>D.1.3 Constant angular acceleration</td>
<td>29</td>
</tr>
<tr>
<td>D.2 Three-dimensional cantilever beam</td>
<td>34</td>
</tr>
<tr>
<td>D.2.1 Natural modes and frequencies</td>
<td>34</td>
</tr>
<tr>
<td>D.2.2 Constant angular velocity</td>
<td>35</td>
</tr>
<tr>
<td>D.3 Starting of a rotor blade</td>
<td>41</td>
</tr>
<tr>
<td><strong>E</strong> Developed code</td>
<td>43</td>
</tr>
<tr>
<td>E.1 One-dimensional stationary solver</td>
<td>43</td>
</tr>
<tr>
<td>E.2 Modal analysis: free undamped vibrations</td>
<td>48</td>
</tr>
<tr>
<td>E.3 Numeric integration: Newmark $\beta$</td>
<td>49</td>
</tr>
<tr>
<td>E.4 SVD implementation</td>
<td>57</td>
</tr>
<tr>
<td>E.5 Auxiliary files</td>
<td>59</td>
</tr>
<tr>
<td>E.6 Others</td>
<td>61</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>64</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>General stress state for a 3D deformable body</td>
<td>6</td>
</tr>
<tr>
<td>A.2</td>
<td>Simplified equation for long rotating beams</td>
<td>11</td>
</tr>
<tr>
<td>C.1</td>
<td>2D mesh composed by two quadrilateral elements.</td>
<td>17</td>
</tr>
<tr>
<td>D.1</td>
<td>Natural modes associated to the eight lowest natural frequencies of a rectangular cantilever beam.</td>
<td>21</td>
</tr>
<tr>
<td>D.2</td>
<td>Left side vectors (predominant modes) associated to the eight higher singular values.</td>
<td>22</td>
</tr>
<tr>
<td>D.3</td>
<td>Right side vectors (predominant oscillations) associated to the eight higher singular values.</td>
<td>22</td>
</tr>
<tr>
<td>D.4</td>
<td>Left side vectors (predominant modes) associated to the eight higher singular values.</td>
<td>23</td>
</tr>
<tr>
<td>D.5</td>
<td>Right side vectors (predominant oscillations) associated to the eight higher singular values.</td>
<td>23</td>
</tr>
<tr>
<td>D.6</td>
<td>Left side vectors (predominant modes) associated to the eight higher singular values.</td>
<td>24</td>
</tr>
<tr>
<td>D.7</td>
<td>Right side vectors (predominant oscillations) associated to the eight higher singular values.</td>
<td>24</td>
</tr>
<tr>
<td>D.8</td>
<td>Mode intensities for increasing values of $\Omega$.</td>
<td>25</td>
</tr>
<tr>
<td>D.9</td>
<td>Reaction torque for increasing values of $\Omega$.</td>
<td>25</td>
</tr>
<tr>
<td>D.10</td>
<td>Local transverse displacements at the tip of the beam for different values of $\Omega$. Numeric results and its SVD approximation are compared.</td>
<td>25</td>
</tr>
<tr>
<td>D.11</td>
<td>Power spectral density estimation of the sum of the right side vectors obtained for different rotation speeds: 50 (a), 100 (b) and 200 (c) rad/s.</td>
<td>26</td>
</tr>
<tr>
<td>D.12</td>
<td>Local horizontal displacements at the tip of the beam for different values of $\Omega$: 50 (a), 100 (b) and 200 (c) rad/s. Numeric results and its SVD approximation are compared.</td>
<td>26</td>
</tr>
<tr>
<td>D.13</td>
<td>Deformed shape ($\times$500) and $\sigma_{zz}$ distribution (colour map, in KPa) at 50 rad/s. Time from 0 to 14 ms.</td>
<td>28</td>
</tr>
<tr>
<td>D.14</td>
<td>Left side vectors (predominant modes) associated to the eight higher singular values.</td>
<td>29</td>
</tr>
<tr>
<td>D.15</td>
<td>Right side vectors (predominant oscillations) associated to the eight higher singular values.</td>
<td>29</td>
</tr>
<tr>
<td>D.16</td>
<td>Left side vectors (predominant modes) associated to the eight higher singular values.</td>
<td>30</td>
</tr>
<tr>
<td>D.17</td>
<td>Right side vectors (predominant oscillations) associated to the eight higher singular values.</td>
<td>30</td>
</tr>
<tr>
<td>D.18</td>
<td>Mode intensities for increasing values of $\alpha$.</td>
<td>31</td>
</tr>
<tr>
<td>D.19</td>
<td>Temporal evolution of vertical displacements at the tip of a cantilever beam under constant angular acceleration.</td>
<td>31</td>
</tr>
<tr>
<td>D.20</td>
<td>Reaction torque for increasing values of $\alpha$.</td>
<td>31</td>
</tr>
<tr>
<td>D.21</td>
<td>Deformed shape ($\times$200) and $\sigma_{zz}$ distribution (colour map, in KPa) at $\alpha = 50$ rad/s$^2$ and $\Omega_0 = 50$ rad/s. Time from 0 to 25 ms.</td>
<td>33</td>
</tr>
<tr>
<td>D.22</td>
<td>Natural modes associated to the eight lowest natural frequencies of a 3D cantilever beam.</td>
<td>34</td>
</tr>
<tr>
<td>D.23</td>
<td>Left side vectors (predominant modes) associated to the eight higher singular values.</td>
<td>35</td>
</tr>
<tr>
<td>D.24</td>
<td>Right side vectors (predominant oscillations) associated to the eight higher singular values.</td>
<td>35</td>
</tr>
</tbody>
</table>
D.25 Left side vectors (predominant modes) associated to the eight higher singular values. 36
D.26 Right side vectors (predominant oscillations) associated to the eight higher singular values. ................................................................. 36
D.27 Left side vectors (predominant modes) associated to the eight higher singular values. 37
D.28 Right side vectors (predominant oscillations) associated to the eight higher singular values. ................................................................. 37
D.29 Campbell diagram: comparison between modal prediction and SVD predominant frequencies. Subscripts \(a\) and \(b\) refer to pairs of equal modes that activate either on \(\rho\) or \(z\). ................................................................. 38
D.30 Temporal evolution of \(\rho\) displacements at the tip of a cantilever beam under constant angular velocity. ................................................................. 39
D.31 Temporal evolution of \(z\) displacements at the tip of a cantilever beam under constant angular velocity. ................................................................. 39
D.32 Reaction torque for increasing values of \(\Omega\). ................................................................. 39
D.33 Deformed shape (\(\times 500\)) and \(\sigma_{zz}\) distribution (colour map, in KPa) at \(\Omega_0 = 50\) rad/s. Time from 0 to 17 ms (up-down, left-right). ................................................................. 40
D.34 Temporal evolution of the angular speed and the local axial displacements at the tip. 41
D.35 Temporal evolution of the angular acceleration and elastic reaction torque. ................. 41
D.36 Temporal evolution of the maximum values of stress. ................................................................. 42
D.37 Right side vectors of the four most predominant oscillations. ................................................................. 42
Theoretical frame

A.1. Introduction to elasticity

A.1.1. The elastic problem

Fundamentals of Theory of Elasticity date from the seventeenth century, back when Robert Hooke posed the law of elasticity, which states that the stretching of a solid is proportional to the force applied to it. This law was latter generalised to three dimensional bodies by Cauchy, introducing six components of stress $\sigma$, linearly related to six components of strain $\varepsilon$. Cauchy was the first to introduce the notion of stress at a point, given by the tractions per unit area across all plane elements through the point.

This linear behaviour of deformable bodies is governed by 15 coupled partial differential equations, first derived by Navier. Even today, no closed form of the solution exists, even for simple structures. Elasticians have used energy principles to tackle this problem, which have become a fundamental tool to study elastic bodies, including those problems involving vibrations (see A.2). The Navier-Cauchy equations are [1]:

Equilibrium equations:

\[
\begin{align*}
\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} + b_x &= 0 \\
\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} + b_y &= 0 \\
\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + b_z &= 0
\end{align*}
\] (A.1)
Strain displacement relations:

\[ \varepsilon_{xx} = \frac{\partial u}{\partial x}, \quad \varepsilon_{yy} = \frac{\partial v}{\partial y}, \quad \varepsilon_{zz} = \frac{\partial w}{\partial z}, \quad \varepsilon_{yx} = \frac{1}{2} \left( \frac{\partial w}{\partial y} + \frac{\partial u}{\partial z} \right) = \varepsilon_{yx} \]

\[ \varepsilon_{yy} = \frac{\partial v}{\partial y}, \quad \varepsilon_{zz} = \frac{\partial w}{\partial z}, \quad \varepsilon_{xy} = \frac{1}{2} \left( \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) = \varepsilon_{xy} \]

\[ \varepsilon_{zz} = \frac{\partial w}{\partial z}, \quad \varepsilon_{yy} = \frac{1}{2} \left( \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) = \varepsilon_{yx} \]

\[ \varepsilon_{xx} = \frac{\partial u}{\partial x}, \quad \varepsilon_{yy} = \frac{\partial v}{\partial y}, \quad \varepsilon_{zz} = \frac{\partial w}{\partial z}, \quad \varepsilon_{xy} = \frac{1}{2} \left( \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) = \varepsilon_{xy} \]

(A.2)

Compatibility relations:

\[ \frac{\partial^2 \varepsilon_{xx}}{\partial y^2} + \frac{\partial^2 \varepsilon_{yy}}{\partial x^2} = 2 \frac{\partial^2 \varepsilon_{xy}}{\partial x \partial y}, \quad \frac{\partial^2 \varepsilon_{xx}}{\partial y \partial z} = \frac{\partial}{\partial x} \left( -\frac{\partial \varepsilon_{yz}}{\partial x} + \frac{\partial \varepsilon_{xz}}{\partial y} + \frac{\partial \varepsilon_{xy}}{\partial z} \right) \]

\[ \frac{\partial^2 \varepsilon_{yy}}{\partial z^2} + \frac{\partial^2 \varepsilon_{zz}}{\partial y^2} = 2 \frac{\partial^2 \varepsilon_{yz}}{\partial y \partial z}, \quad \frac{\partial^2 \varepsilon_{yy}}{\partial z \partial x} = \frac{\partial}{\partial y} \left( -\frac{\partial \varepsilon_{yz}}{\partial x} + \frac{\partial \varepsilon_{xz}}{\partial y} + \frac{\partial \varepsilon_{xy}}{\partial z} \right) \]

\[ \frac{\partial^2 \varepsilon_{zz}}{\partial x^2} + \frac{\partial^2 \varepsilon_{xx}}{\partial z^2} = 2 \frac{\partial^2 \varepsilon_{zx}}{\partial z \partial x}, \quad \frac{\partial^2 \varepsilon_{zz}}{\partial x \partial y} = \frac{\partial}{\partial z} \left( -\frac{\partial \varepsilon_{yz}}{\partial x} + \frac{\partial \varepsilon_{xz}}{\partial y} - \frac{\partial \varepsilon_{xy}}{\partial z} \right) \]

(A.3)

Equilibrium equations are obtained considering a general state of stress at an arbitrary point in the deformable body through an infinitesimal approach (see figure below). Note that shear stresses are represented with a \( \tau \). If one approximates the spatial variations of the stresses in terms of first-order Taylor series, the resulting equilibrium equations are the ones posed in equation A.1. The strain-displacement equations link both strain and displacement field, defining the second as the derivative of the first. However, this is not enough, as another condition needs to be met: the body is still \textit{continuous} after the deformation. The latter is ensured by the compatibility relations, which involve second derivatives.

Figure A.1: General stress state for a 3D deformable body [2]
Robert Hooke was one of the first scientists to notice elastic behaviour of materials. He studied springs under traction and compression, and tried to pose an equation linking both the applied force and the elongation of the spring. He discovered that, for small deformations $x$, the required force was $F = kx$, where $k$ is a constant characteristic of the spring. This law was later generalised to describe elastic objects in general, posing that the strain $\varepsilon$ (that is, nondimensional deformation) is proportional to the stress $\sigma$ (that is, force per unit area) applied to it.

Remembering the equations from the previous section, one can realise that general stresses and strains have multiple independent components, and thus the proportionality factor is no longer a single value, but a tensor. The tensorial form of Hooke’s law introduces a concept that might not be intuitive: some elastic bodies deform in one direction when subjected to a force with a different direction. In such cases, proportionality will hold: if the directions are not changed, the deformation will increase proportional to the applied force. For that cases, the formulation in three dimensions has the following vectorial form:

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (A.4)$$

When moving from a discrete spring to continuous media, the formulation changes a little as the strain and stress state around a point cannot longer be described by a vector as all pushing, pulling and shearing effects have to be taken into account. This complexity is captured using a second order tensor, and in the tensorial formulation, forces and displacements are replaced by stresses and strains, respectively. In a Cartesian base, stresses and strains are represented with a first order tensor each:

$$\varepsilon = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{bmatrix} \quad \sigma = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{bmatrix} \quad (A.5)$$

Stress and strain tensors vary from a point to another inside continuum materials: $\varepsilon$ gives information about the displacements in the neighbourhood of the point, while $\sigma$ stands for the forces per unit area that neighbouring parcels exert on each other. The tensor linking $\varepsilon$ and $\sigma$ is known as the elasticity tensor $C$, represented by a matrix of 81 real numbers $C_{ijkl}$:

$$\sigma_{ij} = \sum_{k=1}^{3} \sum_{l=1}^{3} C_{ijkl} \varepsilon_{kl} \quad (A.6)$$

Due to inherent symmetries of the model, the elasticity tensor $C$ can be reduced to 21 independent coefficients, while $\varepsilon$ and $\sigma$ are found to be symmetrical matrices ($\sigma_{ij} = \sigma_{ji}$ and $\varepsilon_{kl} = \varepsilon_{lk}$), having thus only 6 independent values. In this context, the Voigt notation is introduced: strain and stress tensors are substituted by vectors including only independent coefficients. For the case of shear strains, they are substituted by engineering strains $\gamma$ in Voigt notation ($2\varepsilon_{kl} = \gamma_{kl}$):

$$\sigma \Rightarrow \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{yz} \\ \tau_{xz} \\ \tau_{xy} \end{bmatrix} \quad \varepsilon = \nabla^T u \Rightarrow \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_x}{\partial x} \\ \frac{\partial u_y}{\partial y} \\ \frac{\partial u_z}{\partial z} \\ \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \\ \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \\ \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \end{bmatrix} \quad (A.7)$$
In the above equation, \( \nabla^s u \) stands for the **symmetric gradient** operator, which in 3D Cartesian coordinates adopts the following form:

\[
\nabla^s := \begin{bmatrix}
\frac{\partial}{\partial x} & 0 & 0 \\
0 & \frac{\partial}{\partial y} & 0 \\
0 & 0 & \frac{\partial}{\partial z} \\
\frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} \\
\frac{\partial}{\partial z} & \frac{\partial}{\partial x} & 0
\end{bmatrix}
\]  

(A.8)

For the case of **isotropic materials**—that is, uniform properties in all directions—it can be found that the elasticity matrix depends only on two coefficients, the **Young's Modulus** \( E \) (standing for the material stiffness or resistance to be elastically deformed) and the **Poisson's coefficient** \( \nu \). The latter is the negative of the ratio of transverse strain (unit lateral contraction) to axial strain, and the responsible for the Possion effect: the phenomenon in which a given material tends to expand in directions perpendicular to the direction of compression, and to contract in the directions transverse to the direction of stretching [2].

The Young's Modulus has units of pressure, while the Poisson's coefficient is dimensionless, and theoretically limited to the interval \(-1 < \nu < 0.5\), although in reality only positive values of \( \nu \) are found. However, \( C \) is commonly expressed as a function of the Lame parameters \( \lambda \) and \( \mu \). In indicial form:

\[
C_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}), \quad \lambda = \frac{\nu E}{(1 + \nu)(1 - 2\nu)}, \quad \mu = \frac{E}{2(1 + \nu)}
\]  

(A.9)

In Voigt notation and for isotropic materials, Hooke's law adopts the form:

\[
\begin{bmatrix}
\sigma_x \\
\sigma_y \\
\sigma_z \\
\tau_{yz} \\
\tau_{xz} \\
\tau_{xy}
\end{bmatrix}
= \begin{bmatrix}
2\mu + \lambda & \lambda & \lambda & 0 & 0 & 0 \\
\lambda & 2\mu + \lambda & \lambda & 0 & 0 & 0 \\
\lambda & \lambda & 2\mu + \lambda & 0 & 0 & 0 \\
0 & 0 & 0 & \mu & 0 & 0 \\
0 & 0 & 0 & 0 & \mu & 0 \\
0 & 0 & 0 & 0 & 0 & \mu
\end{bmatrix}
\begin{bmatrix}
\varepsilon_x \\
\varepsilon_y \\
\varepsilon_z \\
\gamma_{yz} \\
\gamma_{xz} \\
\gamma_{xy}
\end{bmatrix}
\]  

(A.10)

The same can be deduced analogously for orthotropic and even anisotropic materials, with the difference that the constants \( E \) and \( \nu \) are no longer unique, but have a different value for each set of directions. Even thermal effects and other phenomena can be added into the equations describing the model. However, although Hooke's law is widely used, it presents great limitations as nonlinearities emerge when the deformations are big enough to be no longer considered small. Thus, nonlinear models have to be used, and the phenomena of **plasticity** has to be taken into account.
A.2. Classical analysis methods

The study of the vibration behaviour of rotatory blades is a well-known problem and the main
subject of rotordynamics, a specialised branch of applied mechanics. In this section, a review on the
main methods and models used over the last century for the analysis of rotating structures will be
addressed, following the key concepts presented in [1].

A.2.1. Energy methods

One of the most fundamentals principles in Physics states that energy in the Universe is conserved.
Energy can adopt many forms, but in the study of rotordynamics the most relevant are kinetic and
potential (strain) energy. If energy changes from potential to kinetic repetitively in time, then vi-
bration appears. In this section, the main methods used to describe the beam behaviour from an
energetic point of view will be briefly presented.

Euler-Lagrange equations

The following equations take advantage of variational calculus to illustrate the behaviour of a one
dimensional idealization of a cantilever beam. Let’s consider a function that would represent the
beam problem:

\[ I = \int_{x_1}^{x_2} F(x, y, y', y'') \, dx \]  \hspace{1cm} (A.11)

with \( x \) as independent variable and \( y \) and its derivatives as an admissible path. Considering the
varied path defined by \( \tilde{y} = y + \epsilon \eta \), being \( \epsilon \) a small parameter and \( \eta \) a differentiable function zero
valued at \( x_1 \) and \( x_2 \), equation A.11 over the varied path reads as:

\[ \tilde{I} = \int_{x_1}^{x_2} F(x, y + \epsilon \eta, y' + \epsilon \eta', y'' + \epsilon \eta'') \, dx \]  \hspace{1cm} (A.12)

If the function above is expanded:

\[ \tilde{I} - I = \frac{\partial \tilde{I}}{\partial \epsilon} \bigg|_{\epsilon=0} \epsilon + \frac{1}{2!} \frac{\partial^2 \tilde{I}}{\partial \epsilon^2} \bigg|_{\epsilon=0} \epsilon^2 + \cdots \]  \hspace{1cm} (A.13)

Neglecting high order terms, one can state that \( \frac{\partial \tilde{I}}{\partial \epsilon} \bigg|_{\epsilon=0} = 0 \). Expanding and integrating by parts,
the equation admits two solutions: or either \( \eta \) and \( \eta' \) are zero at \( x_1 \) and \( x_2 \) (Euler-Lagrange equa-
tion, eq. A.14) or its value is prescribed (boundary conditions, eq. A.15).

\[ \frac{d^2}{dx^2} \left( \frac{\partial F}{\partial y''} \right) - \frac{d}{dx} \left( \frac{\partial F}{\partial y'} \right) + \frac{\partial F}{\partial y} = 0 \]  \hspace{1cm} (A.14)

\[ \left( -\frac{d}{dx} \left( \frac{\partial F}{\partial y''} \right) + \frac{\partial F}{\partial y'} \right)_{x_1}^{x_2} = 0 \]  \hspace{1cm} (A.15)

These equations describe one of the simplest structures, a one dimensional idealization of a can-
tilever beam. The greatness of this equation lies in converting the complex problem of solving 15
coupled differential equations (the Elasticity Problem) into a simple set of equations which are pre-
cise enough for engineering applications. For this particular case, the function \( F \) is found by com-
puting the potential energy of the beam after a load is applied, and reads as:

\[ F = \frac{1}{2} EI \left( \frac{d^2 w}{dx^2} \right)^2 - qw \]  \hspace{1cm} (A.16)
Including eq. A.16 into eq. A.14 yields to
\[
\frac{d^2}{dx^2} \left[ EI \left( \frac{d^2 w}{dx^2} \right) \right] = q \tag{A.17}
\]

Better known as the **beam equation**: \( w \) states for lateral deflection, \( x \) for the independent coordinate along the beam axis, \( E \) for the Young's modulus, \( I \) for the second moment of area of the cross-section about the \( yy \) axis and \( q \) for a distributed load. An advantage of using energy methods instead of the Newtonian approach is that boundary conditions are completely fixed by equation A.15, whereas with classic methods they can be difficult to visualise.

**Other methods**

There are plenty of methods that use conservation of energy as basic principle to describe the behaviour of beams. The **Lagrange method**, for example, assumes that the solution of the equation satisfies the kinematic boundary conditions. For cantilever boundary conditions, in fact, a fourth grade polynomial is found to satisfy boundary conditions in (A.15). The solution is then found by minimizing total potential energy.

Lagrange also studied time-dependent structures, that is, the phenomena of vibration. The conservation of energy for a dynamic system states as
\[
\frac{d}{dt} \left( \frac{\partial T}{\partial q} \right) - \frac{\partial U}{\partial q} = 0, \quad T \text{ representing the kinetic energy and } U \text{ the strain energy. If one uses } n \text{ terms to define both kinetic and strain energy, the following matrix system is obtained, which leads to an eigenvalue problem that was not quite practical to solve until the 20th century:}
\]
\[
[M] \ddot{\eta} + [K] \eta = 0, \quad [M] = m \int_0^l \sum f_i f_j dz, \quad [K] = EI \int_0^l \sum f_i'' f_j'' dz \tag{A.18}
\]

**Rayleigh's approach**, on the other hand, consisted in finding a method for calculating fundamental natural frequencies with a formulation that could be **tabulated** and used in industry as a routine numerical operations, so the designs could be faster developed. To do so, Rayleigh used the conservation principle which states that, for a vibrating conservative system in simple harmonic motion, the maximum potential energy at maximum displacement is equal to the maximum kinetic energy when the system passes trough its equilibrium with maximum velocity.

Another great contributor to this field was Grigoryevich **Galerkin**, who in 1915 published an article on how to solve differential equations boundary problems with an approximate solution method. Differential equations of the type of equation A.14 cannot be solved except in a very few special cases. The key here is to multiply equation A.14 by \( \eta(x) \), which is assumed to be the solution itself, and then integrating this product from \( x_1 \) to \( x_2 \). The solution is found by making this integral zero. In the selection of \( \eta(x) \), the Galerkin method uses \( \dot{y} \) itself. As in the Lagrange method, if one aims to solve the beam equation, it will be necessary to approximate the deflection first.

Modern Finite Element Methods rely, however, on **Hamilton's Principle**, the most general form for all vibrating systems, and one of the most powerful when applied to rotodynamics. A general equation for beam rotodynamics is found in terms of this principle for constant angular acceleration \( \alpha \):
\[
EI_{xx} y''' + \rho A \left[ \dot{y} - (\omega_0 + \alpha t)^2 \left\{ y + R_2 y'' - (R + z) y' \right\} + 2(\omega_0 + \alpha t) \left\{ \dot{y} y' - y'' \int_0^z \dot{y} dz - \int_0^z y'' y' dz \right\} \right] - \left( \rho I_{xx} + \frac{kpEI_{xx}}{G} \right) \ddot{y} = -\rho A \alpha (R + z) \tag{A.19}
\]
Neglecting the effect of shear \((k = 0\) for long blades, as in helicopters), the equation can be simplified and some terms identified. The following equation is solved using approximate methods, and the Galerkin method is a powerful tool to put it in terms of Finite Element Method. As can be seen, Coriolis forces and angular acceleration add non-linearities to the equation. A stiffness **stiffening** and **softening** due to rotation can also be spotted.

\[
\begin{align*}
\ddot{y} + \frac{1}{A} \frac{E I}{\rho A} \dddot{y} + \frac{1}{A} \frac{E I}{\rho A} \dddot{y} + \left(\omega_0^2 + \alpha \right)^2 \left(\zeta + Z\right) y' - R \dot{y}^2 y' &= 0 \\
+ 2\left(\omega_0 + \alpha\right) \left[jy' - y^2 \int_0^x y'dz - \frac{1}{2} y^2 \int_0^x y'dz\right] &= \text{Nonlinear Coriolis Forces} \\
+ \alpha \left[yv' - \frac{1}{2} y^2 \int_0^x y'dz - y^2 \int_0^x y'dz\right] &= \alpha(R + Z) = 0
\end{align*}
\]

**A.2.2. Graphical methods**

At the beginning of the 20th, a rapid growth of rotating machinery made the need of rapid designs in the industrial environment arise. To handle this problem, graphical methods were used to determine natural frequencies of rotating structures. The goal was to create methods to be used by semi-skilled engineers and handled in the shortest possible time.

At that time, beam theory was already developed and used in many fields. A very common method to handle vibration problems consisted on drawing both the shear force and bending moment diagrams. With those, the deflection diagram could also be drawn and the integration was converted into graphical summation. This is a graphical way to interpret the Stodola-Viannello method, which considered deflection itself as a proportional load that produces the deflection. However, this process of drawing the diagrams is rather slow and depends on the accuracy of the drawings. That is why the Stodola-Viannello method became popular in the industrial world not in its graphical form, but as a tabulated iterative process.

In 1922, Holzer proposed another tabulated iterative method, based on Stodola's, which was able to capture higher torsional modes and its frequency. This method was widely used, and for instance a Railway Diesel-Electric Drive was designed. The main drawback of the Holzer method was the impossibility to predict deflection, slope, shear and bending moments.

The first graphical method used in aircraft wings and turbine blades was Myklestad's method, widely spread during World War II. This tabular iterative method follows Holzer's in the way of capturing high torsional modes, but also computes deflection, slope, shear and bending moments.
Phrol’s method is quite similar to Myklestad’s, but addressed to shafts instead of aircraft structures. The main advantage of the latter was that the analysis included the rotatory inertia of the disks, thus adding the so-called **gyroscopic effect**.

This method was widely used until the first half of the past century. With the advanced computational facilities, finite element methods gradually overtook these tabular and energy methods.

**A.2.3. Matrix methods**

As it has already been addressed in equation A.18, the Lagrange method enables the transformation of the differential equations governing the dynamics of free vibration systems into homogeneous algebraic equations. This leads to a well known **eigenvalue problem**, were the key is to find the vibration mode shapes (eigenvectors) and the natural frequencies (eigenvalues). However, the computational needs for practical systems were not present until the late 20\(^{th}\) century.

However, some methods were developed to handle these kind of systems with relative ease: Jacobsen and Ayre’s, Gräffe’s, Prieb’s and Hahn’s are just a few examples of methods that dealt with matrix systems to solve structures.

As matrix methods proven to be a good promise for vibration problems, Duncan and Collar developed an iterative method (essentially an extension of Stodola-Vianello’s) that enabled to handle relatively large matrices. **Transfer matrix** methods also gained popularity because they were easier to adapt for computerization, as they only involve simple matrix multiplications, and the overall transfer matrix is always small. The latter method was extensively used in rotor dynamics calculations for determining critical speeds and unbalanced response. This method was extended even for transient whirl analysis.

But as processing power of computers started to grow, tabular and matrix methods lost their place and nowadays have no use in modern designs.
Elemental matrix computation

When the physical system of study is posed in terms of the FEM, spatial coordinates are discretised and the resulting equations are ruled by a set of global matrices (see in fact equation 3.14). That is why matrix computation is always an important step of every FEM solver. However, those matrices involve integrals over the domain which have no closed solution unless they are split into local or elemental coordinates. Under those conditions, the global matrices are decomposed into smaller, elemental matrices. If mapped into the parent domain, the integrals can be approximated using the Gauss Quadrature method. The aim of this chapter is to review the process of assembly: how global matrices are built from elemental ones. The approximation of the latter will also be assessed.

B.1. The static stiffness matrix

From equation 3.13, the expression of the stiffness matrix for static structures \( K_{\text{static}} \) is, in terms of the FEM:

\[
K_{\text{static}} = \int_\Omega B^T C B \, d\Omega
\]  

(B.1)

The above integral is to be evaluated over the domain \( \Omega \), which becomes infeasible unless local formulation is adopted. Using the formulation presented in section 3.26, the above integral can be splitted into elemental integrals as follows:

\[
K_{\text{static}} = \sum_{e=1}^{n\text{el}} L^e_t^T K_{\text{static}}^e L^e_t, \quad \text{with} \quad K_{\text{static}}^e = \int_\Omega B^e_t^T C B^e_t \, d\Omega^e  
\]  

(B.2)

Where \( L^e \) is a \( n_d n^e \times n_d n_{pt} \) matrix known as boolean connectivity matrix, consisting of the integers 0 and 1 relating element nodal variables \( d^e \in \mathbb{R}^{n_d n^e} \) with the global nodal vector \( (d^e = L^e d) \). With all, the problem of computing a single \( n_d n_{pt} \times n_d n_{pt} \) global matrix reduces to the computation of \( n_{el} \) \( n_d n^e \times n_d n^e \) elemental matrices. For the sake of simplicity, the assembly operation is expressed as:

\[
K_{\text{static}} = \sum_{e=1}^{n\text{el}} L^e_t K_{\text{static}}^e  
\]  

(B.3)

The only thing left is finding a reliable method to compute \( K_{\text{static}}^e \). As integration boundaries may differ from one element to another, the best method to evaluate elemental matrices is by transforming them into the parent domain, where their geometry is normalised and thus Gauss Quadrature can be applied in the same way for all elements. Using the formulation presented in 3.3.1:

\[
K_{\text{static}}^e = \int_\Omega B^e_t^T C B^e_t \, d\tilde{\Omega}^e = \int_{\tilde{\Omega}} J^e B^e_t^T C B^e_t \, d\tilde{\Omega} = \sum_{g=1}^{m} w_g \left( J^e B^e_t^T C B^e_t \right)_{\tilde{\xi} = \xi_g}  
\]  

(B.4)
The number of Gauss points \( m \) required for the integration depends on the order of the employed shape functions, and thus is to be specified for each type of element.

B.2. The force vector

In equation 4.1, \( \mathbf{F} \) stands for the force vector accounting for the effect of the body forces \( \mathbf{F}_b \), boundary tractions \( \mathbf{F}_t \), and centrifugal forces \( \mathbf{F}_\text{rot} \). The methodology used to compute them is the same as in the previous case: splitting the global vector into elemental vectors and mapping them into the parent domain, where Gauss Quadrature can be applied.

B.2.1. Body forces

Recall from equation 3.13 that body and rotation force vectors can be expressed in terms of the FEM as:

\[
\mathbf{F}_b = \int_\Omega \mathbf{N}^T \mathbf{f} \, d\Omega, \quad \mathbf{F}_\text{rot} = -\int_\Omega \mathbf{N}^T \rho \left( \ddot{\mathbf{r}}_i + \ddot{\alpha} \right) d\Omega
\]  

(B.5)

Where \( \mathbf{f} \) stands for the vector of external body forces acting on the structure. Notice that the shape of both \( \mathbf{F}_b \) and \( \mathbf{F}_\text{rot} \) is quite similar, and so they can be written as:

\[
\mathbf{F}_{(b + \text{rot})} = \mathbf{F}_b + \mathbf{F}_\text{rot} = \int_\Omega \mathbf{N}^T \left( \mathbf{f} - \rho \left( \ddot{\mathbf{r}}_i + \ddot{\alpha} \right) \right) d\Omega = \int_\Omega \mathbf{N}^T \mathbf{\tilde{f}} \, d\Omega,
\]  

(B.6)

Being \( \mathbf{\tilde{f}} \) the total contribution of body and rotation forces. As in the previous case, the global vector can be splitted into elemental force vectors using the assembly operator:

\[
\mathbf{F}_{(b + \text{rot})} = \sum_{e=1}^{n_e} \int_\Omega \mathbf{N}^e_T \mathbf{\tilde{f}}^e \, d\Omega^e = \sum_{e=1}^{n_e} \int_\Omega \mathbf{N}^e_T \mathbf{\tilde{f}} \, d\Omega^e
\]  

(B.7)

The problem lies now in how to define the vector \( \mathbf{\tilde{f}} \) along each element. A common approach is to replace it by an approximation constructed by interpolation of its nodal values \( \mathbf{\tilde{f}}^e \) using the same shape functions as for the displacement field:

\[
\mathbf{\tilde{f}} \big|_{\mathbf{x} \in \Omega^e} \approx \mathbf{N}^e \mathbf{\tilde{f}}^e, \quad \text{with} \quad \mathbf{\tilde{f}}^e = \begin{bmatrix} \mathbf{\tilde{f}}^e(x_{e1}) \\ \mathbf{\tilde{f}}^e(x_{e2}) \\ \vdots \\ \mathbf{\tilde{f}}^e(x_{en_e}) \end{bmatrix}
\]  

(B.8)

Note that this approximation can induce errors if, for example, two neighbouring elements have different densities, as density would be a nodal property rather than an elemental one. Using Gauss Quadrature, the sum of the elemental body and rotation force vectors is expressed as:

\[
\mathbf{F}_{(b + \text{rot})}^e = \int_\Omega \mathbf{N}^e_T \left( \mathbf{N}^e \mathbf{\tilde{f}}^e \right) d\Omega^e = \sum_{g=1}^{m_g} w_g \left( \mathbf{J}^e \mathbf{N}^e_T \left( \mathbf{N}^e \mathbf{\tilde{f}}^e \right) \right)_{\xi_g}
\]  

(B.9)

B.2.2. Traction forces

When determining the external forces due to boundary traction conditions \( \mathbf{F}_t \), it is convenient to distinguish between point loads \( \mathbf{F}_{\text{pnt}} \) and distributed loads \( \mathbf{F}_{\text{dis}} \). The determination of \( \mathbf{F}_{\text{pnt}} \) is straightforward: if a point force \( \mathbf{Q} \in \mathbb{R}^{n_d} \) is applied to node \( \mathbf{A} \), then \( \left( \mathbf{F}_{\text{pnt}} \right)_A = \mathbf{Q} \). Point loads are a particular case of distributed loads, where the force function is the Dirac delta operator: zero valued everywhere but \( \mathbf{A} \), where it presents a spike.
B.2. The force vector

The study of distributed loads requires a more meticulous approach, and a new formulation regarding boundary elements is to be posed. To start with, let’s define \( \mathbf{E}_b^l \) as the set of elements lying on the Neumann boundary \( \Gamma_{\partial}^l \), with \( l = 1, \ldots, n_d \):

\[
\mathbf{E}_b^l := \left\{ e \in \{1, 2, \ldots, n_{el}\} | \Gamma_{\partial}^{e, l} = \Gamma_{\partial}^l \cap \Gamma_{\partial}^l \neq \emptyset \right\} \tag{B.10}
\]

Being \( n_{b}^e \) the number of nodes per boundary element, and assuming for simplicity it is the same for all elements, one can define \( \mathbf{CNB}^l \) as the boundary connectivity matrix, a \( n_{el}^{b, l} \times n_{b}^e \) matrix containing, in each row, the global indices of the boundary nodes of each patch \( \Gamma_{\partial}^{e, l} (e \in \mathbf{E}_b^l) \). For each boundary element \( \Gamma_{\partial}^{e, l} \), a matrix of boundary shape functions is defined:

\[
\mathbf{N}^e := \begin{bmatrix} \mathbf{N}_1^e & \mathbf{N}_2^e & \cdots & \mathbf{N}_{n_b}^e \end{bmatrix} \tag{B.11}
\]

Notice that the boundary shape functions are different from the ones used to interpolate the displacements, as the dimension of the boundary is always smaller than the dimension of the elements, \( \mathbf{N}^e \): \( \Gamma_{\partial}^l \to \mathbb{R} \). Recall the definition of the global vector of prescribed boundary tractions (equation 4.1), whose decomposition into elemental integrals reads as:

\[
\mathbf{F}_{\text{dis}} = \sum_{l=1}^{n_d} \int_{\Gamma_{\partial}^l} \mathbf{N}_i^T \mathbf{t}^l d\Gamma = \sum_{l=1}^{n_d} \sum_{e \in \mathbf{E}_b^l} \left( \mathbf{L}^{e, l} \right)^T \int_{\Gamma_{\partial}^l} \mathbf{N}_i^T \mathbf{t}^l d\Gamma \tag{B.12}
\]

Where \( \mathbf{L}^{e, l} \) is the Boolean operator linking the global \( \mathbf{c} \in \mathbb{R}^{n_d n_{b}^e} \) and local \( \mathbf{c}^{e, l} \in \mathbb{R}^{n_{b}^e} \) vectors of test coefficients. The above expression can be expressed as:

\[
\mathbf{F}_{\text{dis}} = \sum_{l=1}^{n_d} \mathbf{F}_l^{\text{dis}} = \sum_{l=1}^{n_d} \sum_{e \in \mathbf{E}_b^l} \left( \mathbf{L}^{e, l} \right)^T \mathbf{F}_{\text{dis}}^{e, l} = \sum_{l=1}^{n_d} \sum_{e \in \mathbf{E}_b^l} \left( \mathbf{L}^{e, l} \right)^T \int_{\Gamma_{\partial}^l} \mathbf{N}_i^T \mathbf{t}^l d\Gamma \tag{B.13}
\]

Where \( \mathbf{F}_{\text{dis}}^{e, l} \in \mathbb{R}^{n_{b}^e} \) is the elemental traction vector due to distributed loads along direction \( \mathbf{i} \), \( \mathbf{F}_l^{\text{dis}} \in \mathbb{R}^{n_d n_{b}^e} \) is the global vector due to distributed loads along direction \( \mathbf{l} \) and \( \mathbf{F}_{\text{dis}} \in \mathbb{R}^{n_d n_{b}^e} \) is the global vector due to distributed loads. As for the case of body forces, the input function \( \mathbf{b}^l \) is commonly replaced by an approximation constructed by interpolation of its nodal values \( \mathbf{b}^{e, l} \) using the shape functions employed for interpolating the boundary test functions:

\[
\mathbf{b}^{e, l} = \mathbf{N}^e \mathbf{b}^l, \quad \text{with} \quad \mathbf{b}^{e, l} = \begin{bmatrix} \mathbf{b}^l(x_1^e) \\ \mathbf{b}^l(x_2^e) \\ \vdots \\ \mathbf{b}^l(x_{n_b}^e) \end{bmatrix} \tag{B.14}
\]

Using this interpolation, and taking advantage of the assembly operator, the global vector due to distributed loads along direction \( \mathbf{i} \) results in:

\[
\mathbf{F}_l^{\text{dis}} = \mathbf{A} \mathbf{F}_{\text{dis}}^{e, l} = \sum_{e \in \mathbf{E}_b^l} \int_{\Gamma_{\partial}^l} \mathbf{N}^e^T \left( \left[ \mathbf{N}^e \mathbf{b}^l \right]^{e, l} \right) d\Gamma \tag{B.15}
\]

If mapped into the parent domain, the computation of the elemental traction vector due to distributed loads along direction \( \mathbf{i} \) can be approximated using the Gauss Quadrature method:

\[
\mathbf{F}_{\text{dis}}^{e, l} = \int_{\Gamma_{\partial}^l} \mathbf{N}^e^T \left( \left[ \mathbf{N}^e \mathbf{b}^l \right]^{e, l} \right) d\Gamma = \sum_{g=1}^{m_b} \mathbf{J}_{b}^{g, e, l} \mathbf{N}^e^T \left( \left[ \mathbf{N}^e \mathbf{b}^l \right]^{e, l} \right)_{\xi = \xi^g} \tag{B.16}
\]
As it has already been stressed out, $\Gamma_{\alpha}^{e,i} \subset R^{n_{e}-1}$ and thus the shape functions and weights appearing in the above expressions are different from those computed when assessing the stiffness matrix and the body forces vector. For 3D problems, the boundary elements are 2D, and therefore a different Gauss rule is applied.

### B.3. The mass matrix

It is found from equation 3.13 that the mass matrix can be expressed in terms of the FEM as:

$$M = \int_{\Omega} N^T \rho N \, d\Omega$$  \hspace{1cm} (B.17)

The process of splitting the above integral into elemental ones, and approximating the latter using the Gauss Quadrature method is identical as for the static stiffness matrix case. Using the assembly operator:

$$M = \sum_{e=1}^{n_{el}} M^e, \quad M^e = \int_{\Omega^e} N^e^T \rho N^e \, d\Omega^e$$  \hspace{1cm} (B.18)

The elemental matrices are then mapped into the parent domain and approximated using Gauss Quadrature:

$$M^e = \int_{\Omega} N^e^T \rho N^e \, d\Omega = \int_{\Omega} J^e N^e^T \rho N^e \, d\Omega^e = \sum_{g=1}^{m} w_g \left( J^e N^e^T \rho N^e \right)_{\xi=\xi_g}$$  \hspace{1cm} (B.19)

### B.4. The static damping matrix

It is found from equation 3.13 that the static damping matrix can be expressed in terms of the FEM as:

$$D_{\text{static}} = \int_{\Omega} N^T \mu N \, d\Omega + \int_{\Omega} B^T B \, d\Omega$$  \hspace{1cm} (B.20)

Using the approximation presented in equation 3.15, the above integral is transformed into:

$$D_{\text{static}} = \bar{\alpha} M + \bar{\beta} K_{\text{static}}$$  \hspace{1cm} (B.21)

Where $M$ and $K_{\text{static}}$ are known matrices, while $\bar{\alpha}$ and $\bar{\beta}$ are constant parameters.

### B.5. The rotation matrices

Recall that the rotation of the structure induces new matrices to appear in equation 3.13:

$$K_{\text{rot}} = \int_{\Omega} N^T \rho \left( \Omega^2 + \alpha \right) N \, d\Omega, \quad D_{\text{rot}} = 2 \int_{\Omega} N^T \rho \Omega N \, d\Omega$$  \hspace{1cm} (B.22)

Its computation method is not so different from the mass matrix case:

$$K_{\text{rot}} = \sum_{e=1}^{n_{el}} K_{\text{rot}}^e \Rightarrow K_{\text{rot}}^e = \int_{\Omega^e} N^e^T \rho \left( \Omega^2 + \alpha \right) N^e \, d\Omega^e$$  \hspace{1cm} (B.23)

$$D_{\text{rot}} = \sum_{e=1}^{n_{el}} D_{\text{rot}}^e \Rightarrow D_{\text{rot}}^e = 2 \int_{\Omega^e} N^e^T \rho \Omega N^e \, d\Omega^e$$  \hspace{1cm} (B.24)

Using the Gauss Quadrature method to approximate the elemental integrals:

$$K_{\text{rot}}^e = \int_{\Omega^e} J^e N^e^T \rho \left( \Omega^2 + \alpha \right) N^e \, d\Omega^e = \sum_{g=1}^{m} w_g \left( J^e N^e^T \rho \left( \Omega^2 + \alpha \right) N^e \right)_{\xi=\xi_g}$$  \hspace{1cm} (B.25)

$$D_{\text{rot}}^e = 2 \int_{\Omega^e} J^e N^e^T \rho \Omega N^e \, d\Omega^e = 2 \sum_{g=1}^{m} w_g \left( J^e N^e^T \rho \Omega N^e \right)_{\xi=\xi_g}$$  \hspace{1cm} (B.26)
Example of vectorised assembly

The FEM is able to solve boundary value problems along very complex geometries because it splits the domain into a finite number of smaller partitions called elements. In numerical terms, this implies that global matrices are not computed all at once, but element by element. Then, all the elemental matrices need to be assembled, but traditional assembly algorithms can be very slow. For this reason a vectorised assembly algorithm has been used in the present project. The code was provided by prof. J. Hernández and adapted to the case of rotating structures. The goal of the present chapter is to illustrate how vectorised assembly works by developing, step by step, the assembly of the mass matrix for two linear quadrilateral elements.

C.1. Mass matrix vectorised assembly

To start with, recall the definition of the mass matrix and its decomposition into elemental components from the previous chapter:

\[
M = \int_{\Omega} N^T \rho N \, d\Omega
\]

\[
M = \sum_{e=1}^{n_e} M^e, \quad M^e = \int_{\Omega^e} N^T \rho N \, d\Omega^e
\]

(C.1)

To keep the example as simple as possible, let’s consider the following geometry made from two linear quadrilateral elements:

![Figure C.1: 2D mesh composed by two quadrilateral elements.](image-url)
Remember that for 2D elements, the numeration order is not important as well as it follows \textbf{counter-clockwise} direction. In the above figure, numbers inside a circle represent local numeration. When the geometric domain is not as trivial as the presented one, keeping track of the global numeration is not an easy task. On the other hand, local numeration is always performed in the same way. Hence, one may suspect that some properties remain \textit{undisturbed} in the local frame.

Following the previous idea, the vectorised assembly takes advantage of the fact that in the parent domain, all the elements are geometrically identical and so are their shape functions. Hence, it is only necessary to compute the elemental matrix of one element and use it to construct a matrix including all the elemental shape functions matrices. The latter matrix, referred as \( \mathbf{N}_{\text{elem}} \), is actually a \((n_g \times n_d) \times (n_e \times n_d)\) matrix.

Remember that, for vector fields, the elemental matrix of shape functions \( \mathbf{N}^e \) was defined as:

\[
\mathbf{N}^e = \{N_1 \mathbf{I} \ N_2 \mathbf{I} \ \cdots \ N_{n_e} \mathbf{I}\}
\]  \hspace{1cm} (C.2)

Where \( \mathbf{I} \) is the \( n_d \times n_d \) identity matrix and \( N_A \) is the scalar-valued shape function of node \( A \). Let's define now \( \mathbf{N}_E \), a \((n_g \times n_d) \times (n_e \times n_d)\) matrix that incorporates the value of \( \mathbf{N}^e \) at every Gauss Point:

\[
\mathbf{N}_E = \begin{bmatrix}
N_{\xi_1}^1 & N_{\xi_1}^2 & \cdots & N_{\xi_1}^{n_e} \\
N_{\xi_2}^1 & N_{\xi_2}^2 & \cdots & N_{\xi_2}^{n_e} \\
\vdots & \vdots & \ddots & \vdots \\
N_{\xi_{ng}}^1 & N_{\xi_{ng}}^2 & \cdots & N_{\xi_{ng}}^{n_e}
\end{bmatrix} = \begin{bmatrix}
\mathcal{N}^1 \\
\mathcal{N}^2 \\
\vdots \\
\mathcal{N}^{n_{pt}}
\end{bmatrix}
\]  \hspace{1cm} (C.3)

Where \( \mathcal{N} \) is a \( n_g \times n_d \) matrix. Taking advantage of the fact that \( \mathbf{N}_E \) is the same for all elements, \( \mathbf{N}_{\text{elem}} \) can be easily constructed by repeating \( \mathbf{N}_E \) a number of \( n_{el} \) times:

\[
\mathbf{N}_{\text{elem}} = [\mathbf{N}_E \ \mathbf{N}_E \ \cdots \ \mathbf{N}_E]^T_{n_g \times n_d \times n_{el}}
\]  \hspace{1cm} (C.4)

The next step is to built \( \mathbf{N}_{st} \), a \((n_g \times n_d) \times (n_{pt} \times n_d)\) matrix that takes into account the global formulation and is constructed from \( \mathbf{N}_{\text{elem}} \) by using the connectivity matrix. For instance, for the presented geometry and with \( n_e = 4 \) and \( n_g = 4 \), \( \mathbf{N}_{st} \) adopts the following form:

\[
\mathbf{N}_{st} = \begin{bmatrix}
\mathcal{N}^1 & \mathcal{N}^2 & 0 & 0 & \mathcal{N}^3 & \mathcal{N}^4 \\
0 & \mathcal{N}^1 & \mathcal{N}^2 & \mathcal{N}^3 & \mathcal{N}^4 & 0
\end{bmatrix}_{n_1 \ n_2 \ n_3 \ n_4 \ n_5 \ n_6}
\]  \hspace{1cm} (C.5)

The grace of this approach is that once \( \mathbf{N}_E \) is computed, the transformations that follow are performed using optimised in-built functions such as \textit{sparse}, \textit{reshape} and \textit{repmat}:

\begin{verbatim}
1 m = ndim*nelem*ngaus ; % Number of rows
2 n = nnode*ndim ; % Number of columns
3 nzmaxLOC = m*nnodeE*ndim ; % Maximum number of zeros
4 Nst = sparse([],[],[],m,n,nzmaxLOC) ; % Allocating memory for Bst
5 for inode = 1:nnodeE % Loop over number of nodes at each element
6    setnodes = CONNECT(:,inode) ; % Global numbering of the
7        % inode-th node of each element
8    end
\end{verbatim}
for idime = 1:ndim % loop over spatial dimensions
    DOFloc = (inode-1)*ndim+idime; % Corresponding indices
    s = Nelems(:,DOFloc); % Corresponding column
    i = [1:m]'; % All rows
    DOFglo = (setnodes-1)*ndim+idime; % b) Columns
    j = repmat(DOFglo', ndim*ngaus,1);
    j = reshape(j,length(i),1);
    % Assembly process (as sum of matrix with sparse representation)
    Nst = Nst + sparse(i,j,s,m,n,m);
end

Until now, the Gauss weights and Jacobians were not considered. So let’s define \( \text{WSTs} \), a \( n_{el} \times n_g \) vector containing the product of weights and Jacobians at all Gauss points:

\[
\text{WSTs} = \begin{bmatrix} s^1 \\ s^2 \\ \vdots \\ s^{n_{el}} \end{bmatrix}, \quad s^e = \begin{bmatrix} w_1 \cdot J_{\xi_1}^e \\ w_2 \cdot J_{\xi_2}^e \\ \vdots \\ w_{ng} \cdot J_{\xi_{ng}}^e \end{bmatrix}
\]

The previous matrix is to be transformed into diagonal. To do so, let’s first define the following auxiliary matrices:

\[
w^e_i = (w_i \cdot J_{\xi_i}^e) \mathbf{I}, \quad \mathbf{W}_d^e = \begin{bmatrix} w^e_1 \ [0] \ [0] \ [0] \\ [0] \ w^e_2 \ [0] \ [0] \\ \vdots \ \vdots \ \vdots \\ [0] \ \ldots \ \ldots \ w^e_{ng} \end{bmatrix}
\]

And now the diagonalized version of \( \text{WSTs} \) reads as:

\[
\text{W}_d^{\text{diag}} = \begin{bmatrix} \mathbf{W}_d^1 \ [0] \ \ldots \ [0] \\ [0] \ \mathbf{W}_d^2 \ \ldots \ [0] \\ \vdots \ \vdots \ \vdots \\ [0] \ \ldots \ \ldots \ \mathbf{W}_d^{n_{el}} \end{bmatrix}
\]

To close down the section, let’s take density into consideration. Recall that density is given as a vector input, having a value for each element:

\[
density = \{\rho^1 \rho^2 \ldots \rho^{n_{el}}\}^T
\]

The next step is to transform the previous expression into a global matrix, accounting for each dimension and Gauss point. To do so, let’s consider the following matrices:

\[
\text{dens}^e = \rho^e \mathbf{I}, \quad \text{dens}_G^e = \begin{bmatrix} \text{dens}^e \ [0] \ \ldots \ [0] \\ [0] \ \text{dens}^e \ \ldots \ [0] \\ \vdots \ \vdots \ \vdots \\ [0] \ \ldots \ \ldots \ \text{dens}^e \\ [0] \ [0] \ \ldots \ \text{dens}^e \end{bmatrix}_n
\]
And finally, the global density matrix is defined as:

$$\text{densglo} = \begin{bmatrix}
\text{dens}_G^1 & [0] & \cdots & [0] \\
[0] & \text{dens}_G^2 & \cdots & [0] \\
\vdots & \vdots & \ddots & \vdots \\
[0] & [0] & \cdots & \text{dens}_G^{n_{st}}
\end{bmatrix} \quad (C.11)$$

After all this abstract formulation, the mass matrix can be simply assembled with no nested loops at all:

$$M = N_{st}^T \cdot \text{densglo} \cdot W_{diag} \cdot N_{st} \quad (C.12)$$

### C.2. Generalisation to rotation matrices

During the development of the FEM method for rotating structures, it was found that new matrices emerged by the effect of rotation. These matrices have the form:

$$P = \int_{\Omega} \rho \mathbf{A} \mathbf{N} \, d\tilde{\Omega} \quad (C.13)$$

Where $\mathbf{A}$ is a $n_d \times n_d$ matrix accounting for the rotation effect, mostly Coriolis, centrifugal and tangential forces. The vectorised assembly of $P$ is almost straightforward, as most of the process is done exactly as in the case of the mass matrix. The only difference has to do with $\mathbf{A}$, which has to be globally defined:

$$\mathbf{A}_{G}^e = \underbrace{\begin{bmatrix}
\mathbf{A}^e & [0] & \cdots & [0] \\
[0] & \mathbf{A}^e & \cdots & [0] \\
\vdots & \vdots & \ddots & \vdots \\
[0] & [0] & \cdots & \mathbf{A}^e
\end{bmatrix}}_{(n_d \times n_d \times n_g)} \quad , \quad \mathbf{A}_{glo} = \begin{bmatrix}
\mathbf{A}_G^1 & [0] & \cdots & [0] \\
[0] & \mathbf{A}_G^2 & \cdots & [0] \\
\vdots & \vdots & \ddots & \vdots \\
[0] & [0] & \cdots & \mathbf{A}_G^{n_{st}}
\end{bmatrix} \quad (C.14)$$

Finally, the global matrix is assembled as:

$$P = N_{st}^T \cdot \text{densglo} \cdot \mathbf{A}_{glo} \cdot W_{diag} \cdot N_{st} \quad (C.15)$$
In this chapter, results obtained from the developed simulations will be displayed. The discussion and analysis of the present results has already been addressed in the report (sections 6 and 7).

D.1. Two-dimensional cantilever beam

D.1.1. Natural modes and frequencies

<table>
<thead>
<tr>
<th>Modal Analysis: Natural Frequencies (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.96</td>
</tr>
<tr>
<td>3942.17</td>
</tr>
</tbody>
</table>

Figure D.1: Natural modes associated to the eight lowest natural frequencies of a rectangular cantilever beam.

Table D.1: The twenty lowest natural frequencies of a rectangular cantilever beam.
D.1.2. Constant angular velocity

Case 1: $\Omega = 50 \text{ rad/s}$

Figure D.2: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.3: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Case 2: $\Omega = 100$ rad/s

Figure D.4: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.5: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Case 3: $\Omega = 200$ rad/s

Figure D.6: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.7: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Comparative figures

Figure D.8: Mode intensities for increasing values of $\Omega$

Figure D.9: Reaction torque for increasing values of $\Omega$

Figure D.10: Local transverse displacements at the tip of the beam for different values of $\Omega$. Numeric results and its SVD approximation are compared.
D. Raw simulation results

Figure D.11: Power spectral density estimation of the sum of the right side vectors obtained for different rotation speeds: 50 (a), 100 (b) and 200 (c) rad/s.

Figure D.12: Local horizontal displacements at the tip of the beam for different values of $\Omega$: 50 (a), 100 (b) and 200 (c) rad/s. Numeric results and its SVD approximation are compared.
D.1. Two-dimensional cantilever beam

<table>
<thead>
<tr>
<th>$\Omega$ = 50 rad/s</th>
<th>Predominant frequency (Hz)</th>
<th>$U_1$</th>
<th>$U_2$</th>
<th>$U_3$</th>
<th>$U_4$</th>
<th>$U_5$</th>
<th>$U_6$</th>
<th>$U_7$</th>
<th>$U_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predominant frequency (Hz)</td>
<td>40.17</td>
<td>637.21</td>
<td>245.66</td>
<td>647.72</td>
<td>1177.60</td>
<td>1797.41</td>
<td>2476.19</td>
<td>1910.56</td>
</tr>
<tr>
<td></td>
<td>Normalised singular value</td>
<td>6.36 E-01</td>
<td>3.53 E-01</td>
<td>9.50 E-03</td>
<td>8.19 E-04</td>
<td>1.70 E-04</td>
<td>5.94 E-05</td>
<td>2.72 E-05</td>
<td>2.23 E-05</td>
</tr>
<tr>
<td>$\Omega$ = 100 rad/s</td>
<td>Predominant frequency (Hz)</td>
<td>637.01</td>
<td>37.09</td>
<td>245.23</td>
<td>647.61</td>
<td>1177.72</td>
<td>1797.32</td>
<td>1910.35</td>
<td>2476.13</td>
</tr>
<tr>
<td></td>
<td>Normalised singular value</td>
<td>6.56 E-01</td>
<td>3.37 E-01</td>
<td>4.33 E-03</td>
<td>3.66 E-04</td>
<td>8.07 E-04</td>
<td>2.87 E-05</td>
<td>2.06 E-05</td>
<td>1.14 E-05</td>
</tr>
<tr>
<td>$\Omega$ = 200 rad/s</td>
<td>Predominant frequency (Hz)</td>
<td>636.44</td>
<td>25.70</td>
<td>243.53</td>
<td>647.18</td>
<td>1177.32</td>
<td>1910.30</td>
<td>1797.98</td>
<td>2475.88</td>
</tr>
<tr>
<td></td>
<td>Normalised singular value</td>
<td>7.82 E-01</td>
<td>2.15 E-01</td>
<td>1.33 E-03</td>
<td>1.17 E-04</td>
<td>2.40 E-04</td>
<td>1.67 E-05</td>
<td>8.52 E-06</td>
<td>3.37 E-06</td>
</tr>
</tbody>
</table>

Table D.2: Predominant frequencies and normalised singular values of the presented left side vectors.

**Deformed shapes and stress distribution for $\Omega = 50$ rad/s**

[Images of deformed shapes and stress distribution for $\Omega = 50$ rad/s]
Figure D.13: Deformed shape (×500) and $\sigma_{cc}$ distribution (colour map, in KPa) at 50 rad/s. Time from 0 to 14 ms.
D.1.3. Constant angular acceleration

Case 1: $\alpha = 5 \text{ rad/s}^2$

Figure D.14: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.15: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Case 2: $\alpha = 50 \text{ rad/s}^2$

Figure D.16: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.17: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Double two-dimensional cantilever beam

Comparative figures

Figure D.18: Mode intensities for increasing values of $\alpha$.

Figure D.19: Temporal evolution of vertical displacements at the tip of a cantilever beam under constant angular acceleration.

Figure D.20: Reaction torque for increasing values of $\alpha$. 
Table D.3: Predominant frequencies and normalised singular values of the presented left side vectors.

<table>
<thead>
<tr>
<th></th>
<th>( \omega_c = 50 \text{ rad/s} )</th>
<th>Predominant frequency (Hz)</th>
<th>Normalised singular value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega = 5 \text{ rad/s} )</td>
<td>40.15</td>
<td>40.15</td>
<td>245.66</td>
</tr>
</tbody>
</table>

Deformed shapes and stress distribution for \( \alpha = 50 \text{ rad/s}^2 \)
Figure D.21: Deformed shape (∗200) and $\sigma_{yy}$ distribution (colour map, in KPa) at $\alpha = 50$ rad/s$^2$ and $\Omega_0 = 50$ rad/s. Time from 0 to 25 ms.
D.2. Three-dimensional cantilever beam

D.2.1. Natural modes and frequencies

Figure D.22: Natural modes associated to the eight lowest natural frequencies of a 3D cantilever beam.

Table D.4: Fifty lowest natural frequencies of a 3D cantilever beam.

<table>
<thead>
<tr>
<th>Modal Analysis: Natural Frequencies (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>59.03</td>
</tr>
<tr>
<td>638.64</td>
</tr>
<tr>
<td>1365.43</td>
</tr>
</tbody>
</table>
D.2.2. Constant angular velocity

Case 1: $\Omega = 25 \text{ rad/s}$

Figure D.23: Left side vectors (predominant modes) associated to the eight higher singular values.

$U_1 \cdot \sigma_1 = 0.55$
$U_2 \cdot \sigma_2 = 0.44$
$U_3 \cdot \sigma_3 = 2.2 \times 10^{-3}$
$U_4 \cdot \sigma_4 = 3.9 \times 10^{-4}$

$U_5 \cdot \sigma_5 = 1.3 \times 10^{-4}$
$U_6 \cdot \sigma_6 = 1.9 \times 10^{-5}$
$U_7 \cdot \sigma_7 = 4.7 \times 10^{-6}$
$U_8 \cdot \sigma_8 = 2.1 \times 10^{-6}$

Figure D.24: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Case 2: $\Omega = 50 \text{ rad/s}$

Figure D.25: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.26: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Case 3: $\Omega = 100$ rad/s

$\mathbf{U}_1 \cdot \sigma_1 = 0.92$

$\mathbf{U}_2 \cdot \sigma_2 = 0.07$

$\mathbf{U}_3 \cdot \sigma_3 = 3.8 \times 10^{-4}$

$\mathbf{U}_4 \cdot \sigma_4 = 2.9 \times 10^{-5}$

$\mathbf{U}_5 \cdot \sigma_5 = 2.1 \times 10^{-5}$

$\mathbf{U}_6 \cdot \sigma_6 = 3.3 \times 10^{-6}$

$\mathbf{U}_7 \cdot \sigma_7 = 1.4 \times 10^{-6}$

$\mathbf{U}_8 \cdot \sigma_8 = 7.9 \times 10^{-7}$

Figure D.27: Left side vectors (predominant modes) associated to the eight higher singular values.

Figure D.28: Right side vectors (predominant oscillations) associated to the eight higher singular values.
Campbell diagram

Figure D.29: Campbell diagram: comparison between modal prediction and SVD predominant frequencies. Subscripts \( a \) and \( b \) refer to pairs of equal modes that activate either on \( \varphi \) or \( z \).
D.2. Three-dimensional cantilever beam

Comparative figures

Figure D.30: Temporal evolution of \( \rho \) displacements at the tip of a cantilever beam under constant angular velocity.

Figure D.31: Temporal evolution of \( z \) displacements at the tip of a cantilever beam under constant angular velocity.

Figure D.32: Reaction torque for increasing values of \( \Omega \).
Deformed shapes and stress distribution for $\Omega = 50$ rad/s

Figure D.33: Deformed shape ($\times 500$) and $\sigma_{zz}$ distribution (colour map, in KPa) at $\Omega_0 = 50$ rad/s. Time from 0 to 17 ms (up-down, left-right).
D.3. Starting of a rotor blade

Figure D.34: Temporal evolution of the angular speed and the local axial displacements at the tip.

Figure D.35: Temporal evolution of the angular acceleration and elastic reaction torque.
Figure D.36: Temporal evolution of the maximum values of stress.

Figure D.37: Right side vectors of the four most predominant oscillations.
The modelled FEM has been coded in MATLAB. Some of the code regarding the singular value decomposition, the vectorised assembly and the multidimensional static solver has been provided by professor J. Hernández. The present chapter collects some of the most representative functions that were created from scratch for the context of rotating structures analysis.

E.1. One-dimensional stationary solver

Main file

```
%% Code to solve the elastostatic equations for a 1D case
clc
clear all

%% Constants definition
ur = 0; % Prescribed displacements
x0=0; % Initial axial position
xf=4; % Final axial position
n_el=70; % Number of elements to carry the simulation
S=1; % Cross section area (m^2)
E=70e9; % Young Modulus (Pa)
rho = 2700; % Linear density (Kg / m^3)
omega=60; % Angular speed (rad/s)

%% Analytic solution
R=xf; %Radius
k = sqrt(rho*omega^2 / (E));
xi = linspace(0,R,1000); %Axial coordinates
for i=1:length(xi)
    u(i)=(1/k) * (sin(k*xi(i))/cos(k*R)) - xi(i); % Displacements
    sigma2(i) = E*(cos(k*xi(i))/cos(k*R)-1); % Stresses
end

%% Numeric solution
```

43
Developed code

%% 1D finite element method solver
function [d, coor, sigma] = FEM(x0, xf, n_el, E, S, ur, rho, omega)
    %% Geometry definition
    coor = zeros(n_el + 1, 1); % Coordinate matrix
    for i = 1:(n_el + 1)
        coor(i) = x0 + (i - 1)*(xf - x0)/n_el;
    end
    n_nod = size(coor, 1); % Number of nodes

    CN = zeros(n_el, 2); % Connectivity matrix
    for i = 1:(n_el)
        CN(i, 1) = i;
        CN(i, 2) = i + 1;
    end

    %% Stiffness matrix
    K = AssemblyK(coor, CN, S, E, rho, omega);

    %% Force vector
    F = AssemblyF(CN, coor, rho, omega);

    %% Algebraic system resolution
    r = 1; % Nodes with restricted displacement
    l = 2:n_nod; % Nodes with unknown displacement
    A = K(l, l);
    B = (F(l) - K(l, r)*ur);
    dl = A\B;
    R = K(r, l)*dl + K(r, r)*ur - F(r); % Reaction forces

    %% Displacement vector assembly
    d = zeros(n_nod, 1);
    for i = 2:n_nod
        d(i) = dl(i - 1);
    end
    d(1) = ur;

    %% Stress computation
    sigma = stress(coor, CN, E, d, S);
end
**Stiffness matrix assembly**

```matlab
function K = AssemblyK(coor, CN, S, E, rho, omega)
%% Constants
n_el=size(CN,1);
n_nod=size(coor,1);
n_nod_el=size(CN,2);

%% K computation
K=sparse(n_nod,n_nod); % Global stiffness matrix

for e=1:n_el
    n_nod_e=CN(e,:); % Element e nodes
    coor_e=coor(n_nod_e); % Nodal coordinates
    he=coor_e(2)-coor_e(1); % Element length
    Ke=S*(E/he)*[1 -1; -1 1] - he*(1/6)*rho*omega^2*[2 1; 1 2]; % Elemental matrix
    for a=1:n_nod_el % Traditional assembly
        for b=1:n_nod_el
            A=CN(e,a);
            B=CN(e,b);
            K(A,B)=K(A,B)+Ke(a,b);
        end
    end
end
```

**Force vector assembly**

```matlab
function Ff = AssemblyF(CN, coor, rho, omega)
%% Constants
n_el=size(CN,1);
n_nod=size(coor,1);
n_nod_e=size(CN,2);

%% F computation
Ff=sparse(n_nod,1); % Global forces vector

for e=1:n_el
    Fe=ComputeFe(coor, e, CN, rho, omega); % Local forces vector
    for a=1:n_nod_e % Traditional assembly
        A=CN(e,a);
        Ff(A)=Ff(A)+Fe(a);
    end
end
```
Elemental force vector computation

```matlab
function Ff = AssemblyF(CN, coor, rho, omega)
% Constants
n_el=size(CN,1);
n_nod=size(coor,1);
n_nod_e=size(CN,2);

% F computation
Ff=sparse(n_nod,1); %Global forces vector
for e=1:n_el
    Fe=ComputeFe(coor,e,CN,rho,omega); %Local forces vector
    for a=1:n_nod_e %Traditional assembly
        A=CN(e,a);
        Ff(A)=Ff(A)+Fe(a);
    end
end
```

Stresses and strains computation

```matlab
function sigma = stress(coor, CN, E,d, S)
% Constants
n_el=size(CN,1);
n_nod=size(coor,1);
n_nod_el=size(CN,2);

% Stress computation
sigma = zeros(n_el,1);
for e=1:n_el
    n_nod_e=CN(e,:); %Element e nodes
    coor_e=coor(n_nod_e); %Nodal coordinates
    d1=d(n_nod_e(1));
    d2=d(n_nod_e(2));
    he=coor_e(2)-coor_e(1); %Element length
    sigma(e)= (E*S/he)*[-1,1]*[d1; d2]; %stress at element e
end
```

Error computation

```matlab
function [GQ, GQ_diff] = GaussQuad(coor, uexact,uapprox)
GQ=0; %Total error
GQ_diff=0; %Total error of the derivative
n_el=size(coor,1)-1; %Number of elements

% Weights and positions of Gauss points
w(1)=8/9;
w(2)=5/9;
w(3)=5/9;
```
E.1. One-dimensional stationary solver

\[ x_i(1) = 0; \]
\[ x_i(2) = \sqrt{3/5}; \]
\[ x_i(3) = -\sqrt{3/5}; \]

%% Error computation

\begin{verbatim}
for e = 1:n_el
    x = [coor(e); coor(e+1)];  % Physical coordinates of element e
    d = [uapprox(e); uapprox(e+1)]; % Nodal value of u
    h_el = x(2) - x(1); % Element size
    sum = 0; % u error
    sum_diff = 0; % u' error

    for i = 1:3 % For each Gauss point
        N = 0.5*[1-xi(i), 1+xi(i)]; % Shape function
        B = (1/h_el)*[-1, 1]; % Derivative of the shape function
        x_pos = N*x; % Change of variable in the coordinates of element e

        uap = N*d; % Nodal u value (FEM)
        uapdiff = B*d; % Nodal u' value (FEM)

        uex = double(subs(uexact, x_pos)); % Exact u value
        uexdiff = double(subs(diff(uexact), x_pos)); % Exact u' value

        q = (uex - uap)^2; % Squared difference of the functions
        q_diff = (uexdiff - uapdiff)^2;

        sum = sum + w(i)*q; % Values at each Gauss point are added
        sum_diff = sum_diff + w(i)*q_diff;
    end
    GQ = GQ + ((h_el/2)*sum); % Computation of the associated error
    GQ_diff = GQ_diff + ((h_el/2)*sum_diff);
end
GQ = sqrt(GQ); % Error in u
GQ_diff = sqrt(GQ_diff); % Error in u'
\end{verbatim}
### E.2. Modal analysis: free undamped vibrations

```matlab
%% Program to compute dynamic system response using modal analysis
clc
clear all
if exist('ElemBnd')==0
    addpath('ROUTINES_AUX');
end
%% Load the data from the FEM program
load('Inputs','M','K', 'dL', 'COOR', 'CN','TypeElement', 'posgp', ...
     'NameFileMesh', 'DATA','DOFl');

%% Initial assignments
M=M(DOFl, DOFl); % M \rightarrow M(LL)
K=K(DOFl, DOFl); % K \rightarrow K(LL)
n_nodes=size(COOR,1);
n_dim=size(COOR,2);
n_dofs=n_dim*n_nodes;
n_eig=8; %Number of modes to compute

%% Natural modes and frequencies
[MODES, FREQ] = UndampedFREQ(M,K,n_eig); %Eigenvalue problem
FREQ_HZ=FREQ./(2*pi); % Convert rad/s to Hz

%% Initial conditions
v_0=zeros(length(DOFl),1); %Initial velocities = 0
d_0=dL; %Initial displacements
q0 = MODES'*M*d_0;
q0_2 = MODES'*M*v_0;
fact=abs(q0./(norm(q0))); %Initial amplitude normalized

%% Plot normalised initial displacements
figure
subplot(2,1,1);
bar(fact);
title('Normalized initial amplitude vs Number of mode')
ylabel('Normalized initial amplitude')

%% Solver
TF = 0.2; % Final time of simulation
m=10000; % Number of divisions of the timespan
AT = TF/m; % Timestep
Fs = 1/AT; % Sampling frequency
time=linspace(0,TF,m); %Time vector
DYN_DIS=zeros(length(DOFl),length(time)); %Matrix of displacement vectors
for i=1:length(time)
    t=time(i);
```
E.3. Numeric integration: Newmark $\beta$

Main file

```plaintext
%% Elastodynamic solver using the Newman-beta method
clc
clear all

addpath('ROUTINES_AUX','DATA','FEM','INPUTS','K_VECT','M_K_D_STANDARD','M_VECT','MODAL','ROTATION_STUFF') ;

%% Inputs
% Extracts information from the mesh file
NAME_INPUT_DATA = 'BEAM2D' ;

[COOR,CN,TypeElement,TypeElementB, celasglo, DOFr,dR,...
 Tnod,CNb,fNOD,Fpnt,NameFileMesh,typePROBLEM,...
 celasgloINV,densglo,DATA] = ReadInputDataFile(NAME_INPUT_DATA) ;

%% Numeric data
tf = 5; %Final time
AT = 0.000005; %timestep
beta = 1/4; %Newmark beta
gamma = 1/2; %Numerical damping
TIME = linspace(0,tf,(tf/AT)+1); %Vector containing time values

%% MISCELLANEOUS data
%Rayleigh damping
aa=0.25;
bb=0.000025;
```

```matlab
DIS=0;
for j=1:n_eig %Sum the effect of all the eigenvectors
    harmonic = q0(j)*cos(FREQ(j)*t) ... + (q0_2(j)/FREQ(j)) * sin(FREQ(j)*t);
    DIS= DIS + MODES(:,j)*harmonic;
end
DYN.DIS(:,i)=DIS;
end
d_dyn=zeros(n_dofs,length(time));
d_dyn(DOFl,:)=DYN.DIS;

%% GiD postprocessing
GidPostProcessModes(COOR,CN,TypeElement,...
    MODES,posgp,NameFileMesh,n_eig,DOFl);
GidPostProcessDynamic(COOR,CN,TypeElement,d_dyn,'DYN_SOL',posgp,...
    NameFileMesh,time);
```
timeElapsed = 0; %Time spent during the integration process
n_div=40; %Compute dynamic variables 1/40 of the time

%% Aeronautics-related case data
T0=0; % Initial torque generated by the device in kN m
TMAX = 25; % Nominal value of the torque in kN m
P_MAX = 50; % Nominal value of power in kW
rho_air=1.225e-3; % Air density in kKg/m^3
cd=0.008; % Drag coefficient
TA = 0.025; %Time at which torque generated stabilises
TB = 0.05; %Time at which engine turns on
tC = 0.12; %Time at which power generated stabilises

%% Initial rotation conditions
theta0 = 0; % Initial angular position
vel0 = 0; % Initial angular speed
alpha0=0; % Initial guess, used to approximate rotation matrices

%% PREPROCESS and initial matrix computation
% To be executed once
nnode = size(COOR,1); ndim = size(COOR,2);
nelem = size(CN,1); nnodeE = size(CN,2) ;
DOFl = 1:nnode*ndim ;
DOFl(DOFr) = [] ;

rnod = zeros(length(DOFr),2); %This vector contains the node number
%at which pertains each restricted degree of freedom, and the degree of
%freeness which is restricted (whether is X, Y or Z)
for i=1:length(rnod)
    rnod(i,1) = ceil(DOFr(i)/ndim);
    for n=1:ndim
        if rem(DOFr(i),n) == 0
            rnod(i,2) = n;
        end
    end
end

%Transform COOR into a vector
COOR_VECT = zeros(nnode*ndim,1);
for n=1:nnode
    for d=1:ndim
        COOR_VECT(ndim*(n-1)+d)= COOR(n,d);
    end
end

%Constant matrices
disp('Computing M Matrix');
E.3. Numeric integration: Newmark $\beta$

```matlab
M= MassMatrix(densglo,COOR,CN,TypeElement);
M_dyn = M(DOFl,DOFl);

disp('Computing K Matrix');
K2 = StiffnessMatrixGlobal(COOR,CN,TypeElement,celasglo,[],[],); %Vectorised!
Ftrac = FtracCOMP(COOR,CNb,...
    TypeElementB,Fpnt,Tnod); %Traction forces (constant)

%Rigid body modes (2D)
RIGIDMODES = RBMODES(COOR_VECT, nnode);
P = RIGIDMODES'*M*RIGIDMODES;
inertia = P(3,3);

%Gauss Points Positions
TypeIntegrand = 'K';
[~,posgp,~,~] = ComputeElementShapeFun(TypeElement,nnodeE,TypeIntegrand) ;
[Nst,NstW,ngaus]= VECTORISED_DATA(COOR,CN,TypeElement);

%% Initial conditions
%Displacement, velocity and acceleration at t=0 (FREE NODES)
load('Static_d','d'); % Load initial displacements from static FEM code
d0 = d(DOFl); % Initial local displacements
v0 = zeros(length(DOFl),1); % Initial local velocities

% Kinematic variables at constarined nodes
vR = zeros(length(DOFr),1);
aR = zeros(length(DOFr),1);

for j=1:2 % Iteration to guess initial angular velocity
    if ndim==3
        VEL0 = [0 −vel0 0; vel0 0 0; 0 0 0];
        ALPHA0 = [0 −alpha0 0; alpha0 0 0; 0 0 0];
    else
        VEL0 = [0 −vel0; vel0 0];
        ALPHA0 = [0 −alpha0; alpha0 0];
    end

    % Compute initial local acceleration
    [K_rot0,D_rot0,Fb0 Mass _Rotation(densglo,COOR,COOR_VECT,CN,...
        TypeElement,ALPHA0,VEL0,0);
    K0 = K2 + K_rot0; % Initial rotation matrix
    D0 = D_rot0 + aa*M + bb*K2; % Initial damping matrix
    F0 = Fb0 + Ftrac; % Initial force vector
    a0 = M_dyn\(F0(DOFl)−D0(DOFl,DOFl)*...
```
\( v_0 - K_0(D\text{OF}_l, D\text{OF}_l) \cdot d_0 \); % Initial local acceleration

% Initial reactions
\( R_0 = \text{DYNT_REACT}(d_0, v_0, a_0, dR, vR, ... \)
aR, M, K_0, D_0, F_0, DOFl, DOFr); % Initial elastic reactions
Total_R_0 = Total_Reactions(R_0, COOR, rnod); % Total reactions in each axis
W_0 = Total_R_0(6); % Initial elastic reaction torque in Z
alpha_0 = \((1/\text{inertia}) \cdot (T_0 - W_0)\); % Initial angular acceleration

end

% First step corresponds to initial conditions
d_n = d_0;
v_n = v_0;
a_n = a_0;
acc = alpha_0;
vel = vel_0;

% Matrix definition
% Vector to store local displacements at a specific t
d = zeros(nnode*ndim,1);
% Vector to store local velocities at a specific t
v = zeros(nnode*ndim,1);
% Vector to store local accelerations at a specific t
a = zeros(nnode*ndim,1);
DYN_DIS = zeros(nnode*ndim,floor(length(TIME)/n_div));
DYN_VEL = zeros(nnode*ndim,floor(length(TIME)/n_div));
DYN_ACC = zeros(nnode*ndim,floor(length(TIME)/n_div));
% Matrix to store local displacements for each t
DYN_DIS_LOCAL = zeros(nnode*ndim,floor(length(TIME)/n_div));
% Matrix to store local reactions for each t
R_all = zeros(length(DOFr),floor(length(TIME)/n_div));
% Matrix to store nodal contribution of the forces at a specific t
F = zeros(nnode*ndim,1);
% Vector to store rotation variables for each t
angular_vel = zeros(1,floor(length(TIME)/n_div));
angular_acc = zeros(1,floor(length(TIME)/n_div));
torque = zeros(1,floor(length(TIME)/n_div));
power = zeros(1,floor(length(TIME)/n_div));
drag = zeros(1,floor(length(TIME)/n_div));

% SOLVER
for i=1:length(TIME)
tic;
t = TIME(i); % Actual time

% Vector of local displacements at tn
d(DOFl) = d_n;
d(DOFr) = dR;

% Vector of local velocities at tn
v(DOFl) = v_n;
v(DOFr) = vR;

% Vector of local accelerations at tn
a(DOFl) = a_n;
a(DOFr) = aR;

%Rotation matrices
if ndim==3
    Q = [cos(theta) -sin(theta) 0; ...
         sin(theta) cos(theta) 0; ...
         0 0 1];  %Rotation matrix
    VEL = [0 -vel 0; vel 0 0; 0 0 0];
    ACC = [0 -acc 0; acc 0 0; 0 0 0];
    VEL2 = [0 -(vel^2) 0; vel^2 0 0; 0 0 0];
else
    Q = [cos(theta) -sin(theta)]; ...
        sin(theta) cos(theta)];  %Rotation matrix
    VEL = [0 -vel; vel 0];
    ACC = [0 -acc; acc 0];
    VEL2 = [0 -(vel^2); vel^2 0];
end

% Compute matrices that depend on angular variables
[K_rot,D_rot,Fb,Fdrag] = Mass_Rotation(densglo,COOR,COOR_VECT,...
    CN,ACC,VEL,VEL2,rho_air,cd,Nst,NstW,ngaus);

F = Fb+Ftrac+Fdrag;
K = K2 + K_rot;
D = D_rot + aa*M + bb*K2;

% Compute matrices for dynamic solving
K_dyn = K(DOFl,DOFl);
D_dyn = D(DOFl,DOFl);
F_dyn = F(DOFl) - (M(DOFl,DOFr)*aR + D(DOFl,DOFr)*vR + K(DOFl,DOFr)*dR);

%Compute bar (~) vectors and matrices
dn_bar = d_n+ AT*v_n + AT^2 *0.5*(1-2*beta)*a_n;
vn_bar = v_n + (1-gamma)*AT*a_n;
M_bar = (M_dyn + gamma*AT*D_dyn)/(beta*AT^2);

%Solve the system
d_n1 = (M_bar + K_dyn)\(F_dyn - D_dyn*vn_bar + M_bar*dn_bar);
a_n1 = (1/(beta*AT^2))*(d_n1 - dn_bar);
v_n1 = vn_bar + gamma*AT*a_n1;

%Next time-step
d_n = d_n1;
v_n = v_n1;
a_n = a_n1;

%Elastic reactions at tn due to local displacements
R = DYN_REACT(d_n1, v_n1, a_n1, dR, vR, aR, M, K , D, F,DOFl,DOFr);
Total_R = Total_Reactions(R,COOR,rnod);
W_n=Total_R(6); %Elastic reaction torque in Z

% Power and torque injected by the engine
if t<tA
    T = T0+TMAX*sin(t*pi/(2*tA));
    POWER = T*vel;
elseif t<tB && t>tA
    T = TMAX;
    POWER = T*vel;
    P_INI=T*vel;
elseif t<tC && t>tB
    CONST1=pi/(tC-tB);
    CONST2=pi*(0.5-tC/(tC-tB));
    POWER = 0.5*(P_INI+P_MAX)+0.5*(P_MAX-P_INI)*sin(CONST1*t+CONST2);
    T=POWER/vel;
else
    T=P_MAX/vel;
    POWER = T*vel;
end

%Implicit scheme for angular acceleration
velbar = vel+(1-gamma)*AT*acc;
acc = (1/inertia)*(T-W_n);
vel = velbar+gamma*AT*acc;

%Variables to post-process
if rem(i,n_div)==0 %Save 1/n_div of the timesteps
    % Inertial variables
    [DYN_DIS(:,j),DYN_VEL(:,j),DYN_ACC(:,j)] = ROTATED_VECT...
        (d,v,a,Q,COOR_VECT,ndim,nnode, vel,acc);
    % Local displacements


DYN_DIS_LOCAL(:,j)=d;

% Local reactions
R_all(:,j) = R;

% Stresses and strains
D(Fl)= d_n;
D(Fr) = dR;
v(Fl)= v_n;
v(Fr) = vR;

[~, stressGLO(4*j-3:4*j,:), ~ ]= ...
  StressStrainsTest(COOR,CN,TypeElement,celasglo,d,...
  typePROBLEM,celasglo_INV,bb,DATA,v);

% Maximum value of stress
stress_x_max(j) = max(abs(stressGLO(4*j-3,:)));
stress_y_max(j) = max(abs(stressGLO(4*j-2,:)));
stress_xy_max(j) = max(abs(stressGLO(4*j-1,:)));

%Angular variables
angular_vel(j) = vel;
angular_acc(j) = acc;
torque(j) = T;
power(j) = POWER;
drag(j) = sum(Fdrag);
newT(j) = t;

%Display integration status
timeElapsed = timeElapsed + toc;
PERC = 100*(i/length(TIME));  % Completion percentage
if timeRemaining =(100-PERC)*timeElapsed/(PERC);
if mod(i-1,1+round(5*length(TIME)/1000))==0
  clc
disp(['%%%%%%%%%%%%%%%%%%%%%%%%%']);
disp(['INTEGRATION IN PROCESS']);
disp(['beta = ',num2str(beta), ', gamma =', num2str(gamma)]);
disp(['%%%%%%%%%%%%%%%%%%%%%%%%%']);
disp(['num2str(PERC), ' % completed']);
if timeElapsed<60
disp(['seconds elapsed']);
else
disp(['minutes elapsed']);
end
if timeRemaining<60
disp(['%%%%%%%%%%%%%%%%%%%%%%%%%']);
disp(['SAVING RESULTS']);
disp(['%%%%%%%%%%%%%%%%%%%%%%%%%']);

%% Postprocess
GidPostProcessDynamic2(COOR,CN,TypeElement,Post_dis,'DYN_SOL',posgp,...
NameFileMesh,newT,stressGLO,[]);

%% Save for the SVD
save('Results','M', 'COOR', 'CN','TypeElement', 'posgp', ...
'NameFileMesh','DOFl','newT','Post_dis');
disp(['DONE!']);

Vectorised assembly

%%% Computes rotation vectors and matrices

function [K_rot,D_rot,Fb,Fdrag]= Mass_Rotation(DENSGLO,COOR,COOR_VECT,...
CN,ACC,VEL,VEL2,rho_air,cd,Nst,NstW,ngaus)

if nargin == 0
    load('tmp.mat')
end

%%% Constants
nnode = size(COOR,1); ndim = size(COOR,2);
nelem = size(CN,1); nnodeE = size(CN,2);
MATK = VEL*VEL+ACC; %For computing rotation stiffness
MATD = 2*VEL; %Coriolis

%%% Vectorised computation
%%% Gyroscopic
matkglo = repmat(MATK,ngaus*nelem,1);
MATKGLO=ConvertBlockDiag(matkglo);
%%% Coriolis
matdglo = repmat(MATD,ngaus*nelem,1);
MATDGLO=ConvertBlockDiag(matdglo);
%%% Nodal forces
matfglo = repmat(MATK,nnode,1);
MATFGLO = ConvertBlockDiag(matfglo);
% Drag forces
matdragglo = repmat(VEL2,nnode,1);
MATDRAGGLO = ConvertBlockDiag(matdragglo);

% Density
densGLO = repmat(DENSGLO',ngaus,1);
densGLO = densGLO(:);
densGLO = CompWeightDiag(densGLO,ndim);

% Rotation matrices
K_rot = densGLO*MATKGLO*Nst;
K_rot = NstW'*K_rot;
D_rot = densGLO*MATDGLO*Nst;
D_rot = NstW'*D_rot;

% Compute fb_nod
rho_material = densGLO(1,1); % Constant density ...
fb_nod = -rho_material.*MATFGLO * COOR_VECT;
NF = Nst*fb_nod;
Fb = 2*NstW'*NF;

% Compute drag
D0 = 0.5*cd*rho_air;
fdrag_nod = -D0.*MATDRAGGLO *(COOR_VECT.*COOR_VECT);
ND = Nst*fdrag_nod;
Fdrag = 2*NstW'*ND;
end

clc
clear all

%% Include the library developed by Hernandez J.
addpath('SVDlibrary')
addpath('INPUTS')
addpath('SVDlibraryPARTITIONED')
addpath('ROUTINES_AUX')

%% Load the data matrix
load('DATA','M', 'COOR', 'CN','TypeElement', 'posgp', ...
   'NameFileMesh','DOFl','d', 'AT');

%% Definitions
TIME = AT:AT:AT*size(d,2); % Time vector
Fs = 1/AT; % Sample frequency
FMAX = 2400; % Max freq. to compute when applying Fourier
Developed code

% Divisions of FMAX
div = 100000;

%% SVD execution
TOL = 4e-11; % Tolerance of the truncated method
[U,S,V] = WRSVDT(y,M,TOL);
% U --> Matrix of modes
% S --> Intensity of each mode (Singular Values)
S_norm = S/sum(S); % Normalised singular values

%% Predominant frequencies: Method 1 (find local peaks)
AppF_A = zeros(size(V,2),1); % Approximate dominant frequencies of RSV
for j=1:size(V,2) % For each RSV
    [~,loc]=findpeaks(V(:,j)); % Search for local peaks in the RSV
    if length(loc)<=1
        AppF_A(j) = Fs/(2*loc);
    else
        AppF_A(j) = Fs/mean(diff(loc)); % Compute an averaged frequency
    end
end

%% Predominant frequencies: Method 2 (discrete Fourier transform)
AppF_B = zeros(size(V,2),1);
h = 80; % Intensity of the high frequency filter
for j=1:size(V,2) % For each RSV
    % Do a periodogram of the RSV and find the peaks in there
    [Pxx,F] = periodogram(V(:,j),[],linspace(0,FMAX,div),Fs);
    for j=1:length(F)-h % High frequency filter
        FilteredPxx(j)=mean(Pxx(j:j+h));
    end
    [~,loc]=max(FilteredPxx); % Find the local maxima
    AppF_B(j)=F(loc);
end

%% RSV plots
for i=1:length(S)
    subplot(4,2,i);
    plot( linspace(0,AT*length(V(:,i)), length(V(:,i))),V(:,i));
    xlabel('Time (s)');
    ylabel(['V_',num2str(i)]);
end

%% GiD postprocess
GidPostProcessModes(COOR,CN,TypeElement,U(DOFl,:),... posgp,'svd3d25',size(U,2),DOFl);
E.5. Auxiliary files

Rigid body modes

```matlab
function [MODES] = RBMODES(COOR_VECT, nnode)
% Computes Rigid body modes using vectorised approach:
% Rotation about (0,0) Z axis, 2D
MAT = [0 -1; 1 0];
matglo = repmat(MAT,nnode,1);
MATGLO=ConvertBlockDiag(matglo);
MODE3 = MATGLO*COOR_VECT;
MODE1 = repmat([1;0],nnode,1);
MODE2 = repmat([0;1],nnode,1);
MODES = [MODE1, MODE2, MODE3];
end
```

Dynamic elastic reactions

```matlab
function R = DYN_REACT(dl, vl, al, dr, vr, ar, M, K, D, DOFl, DOFr)
% Computes reactions for the dynamic case: local case
R = M(DOFr,DOFl)*al + D(DOFr,DOFl)*vl + K(DOFr,DOFl)*dl + ... 
  M(DOFr,DOFr)*ar + D(DOFr,DOFr)*vr + K(DOFr,DOFr)*dr - F(DOFr);
end
```

Total contributions of elastic reactions

```matlab
function [REACTIONS] = Total_Reactions(R,COOR,rNOD)
ndim=size(COOR,2);
n=size(rNOD,1);
% Initialise reactions
RX=0;
RY=0;
RZ=0;
% Initialise moments
MX=0;
MY=0;
MZ=0;
% 3D case
if ndim==3
    for i=1:n
        e=rNOD(i,1); % Node at which reactions are computed
        axis = rNOD(i,2); % Degree of freedom: X Y or Z
        Rx=0;
        Ry=0;
        Rz=0;
    end
```
```matlab
if axis==1
    Rx=R(i);
elseif axis==2
    Ry=R(i);
elseif axis ==3
    Rz=R(i);
end
RX=RX+Rx;
RY=RY+Ry;
RZ=RZ+Rz;
mx=−Ry*COOR(e,3)+Rz*COOR(e,2);
my=Rx*COOR(e,3)−Rz*COOR(e,1);
mz=−Rx*COOR(e,2)+Ry*COOR(e,1);
MX=MX+mx;
MY=MY+my;
MZ=MZ+mz;
end
end
REACTIONS = [RX, RY, RZ, MX, MY, MZ];
end
```

### Coordinate transformation

```matlab
function [d_inertial,v_inertial,acc_inertial] = ROTATED_VEC(d_local,...```

---

E. Developed code
The input is the local displacements vector, which is rotated according to the rotation matrix. The displacements in inertial frame d_inertial = zeros(nnode,1); %Displacements in inertial frame v_inertial = zeros(nnode,1); %Velocities in inertial frame acc_inertial = zeros(nnode,1); %Accelerations in inertial frame

%% Angular spin matrices
if ndim==2
VEL = [0; -w; w 0];
ACC = [0; -a; a 0];
end
if ndim==3
VEL = [0; -w 0; w 0 0; 0 0 0];
ACC = [0; -a 0; a 0 0; 0 0 0];
end
%% Trasformation
for n=1:nnode %For each node
a = (ndim*(n-1)+1);
b = (n*ndim);
u = d_local(a:b); %Local position
u2 = v_local(a:b); %Local velocity
u3 = acc_local(a:b); %Local acceleration
x0 = COOR_VECT(a:b);
d_inertial(a:b) = (Q-eye(ndim))*x0 + Q*u;
v_inertial(a:b) = Q*VEL*(x0+u)+Q*u2;
acc_inertial(a:b) = Q*(VEL*VEL*(x0+u)+ACC*(x0+u)+2*VEL*u2 + u3);
end

Symbolic 1D FEM functions

%% Function that computes 1D shape functions of order n
clc
clear all

%% Initial definitions
grade = 2; %Shape functions grade
n_f = grade+1; %Number of nodes
v = zeros(n_f^2,1);

%% Computation
for i=1:n_f
j=(i-1)*(n_f+1)+1;
v(j) = 1;
end
x = linspace(-1,1,n_f);
B=zeros(n_f,n_f);
for i=1:n_f
    for j=1:n_f
        B(i,j)=x(i)^(n_f-j);
    end
end
A=zeros(n_f^2,n_f^2);
for i=1:n_f
    for j=1:n_f
        q(j)=n_f*(i-1) + j;
    end
    A(q,q)=B;
end
sol = A\v;
COEF = zeros(n_f,n_f);
for i=1:n_f
    for j=1:n_f
        COEF(i,j) = sol((i-1)*n_f +j);
    end
end
syms xi
shape=xi*zeros(n_f,1);
for i=1:n_f
    for j=1:n_f
        Z = shape(i,1)+COEF(i,j)*xi^(n_f-j);
        shape(i,1)=Z;
    end
end
%% Plots
fplot(shape,[−1 1]);
legend('show','Location','best')
xlabel('\xi');
title('Quadratic shape functions');
hold on
plot(x,0,'ko','MarkerFaceColor',[0,0,0],'HandleVisibility','off');
figure
dershape=diff(shape);
fplot(dershape,[−1 1]);
title('dN / d\xi');
 Rotation tensor computation

```matlab
%% Function to compute the rotation tensor T
clc
clear all

%% Definitions
syms xx xy xz yy yz zz o; %Corotational coordinates
syms XX XY XZ YY YZ ZZ; %Inertial coordinates
Q = [cos(o) -sin(o) 0; sin(o) cos(o) 0; 0 0 1]; % Rotation matrix

sigma = [xx xy xz; xy yy yz; xz yz zz]; % Stress tensor in local frame
sigma_voigt=[xx yy zz yz xz xy].'; %Voigt stress vector in local frame

%% Computation
sigma2=Q*sigma*Q.'; % Rotation tensor in the inertial frame
XX= sigma2(1,1);
YY= sigma2(2,2);
ZZ= sigma2(3,3);
XY= sigma2(1,2);
XZ= sigma2(1,3);
YZ= sigma2(2,3);
sigma2_voigt=[XX YY ZZ YZ XY].'; %Voigt stress vector in global frame

T = [ cos(o)^2 sin(o)^2 0 0 0 0; -2*sin(o)*cos(o); sin(o)^2 cos(o)^2 0 0 0 2*sin(o)*cos(o); 0 0 1 0 0 0; 0 0 0 cos(o) sin(o) 0; 0 0 0 -sin(o) cos(o) 0; sin(o)*cos(o) -sin(o)*cos(o) 0 0 0 (cos(o)^2-sin(o)^2)];
```
Bibliography
