$\left[1\right]$

Project work

Student Per Ivar Bruheim

Implementation of a modified inverse iteration algorithm in an object-oriented finite element framework

Trondheim, December 2011

NTNU
Norwegian University of Science and Technology
Faculty of Engineering Science and Technology
Department of Structural Engineering

NTNU
Norwegian University of Science and Technology

Preface

This report is the result of a project work at the Department of Structural Engineering at the Norwegian University of Science and Technology, and corresponds to 7.5 ECTS credits.

A main goal has been to get sufficiently familiarized with the finite element framework provided by my supervisor Paul E. Thomassen to understand what is happening "under the hood". With this knowledge aboard, implementing an eigenvalue solver was the other main objective.

I would like to thank my supervisor Paul E. Thomassen, Post. Doc. at the Department of Civil and Transport Engineering, NTNU, for involving me in an interesting area of research as well as for sharing with me his experience with the framework and providing me with a lot of relevant litterature. I also would like to thank Professor Gregory Miller and Michael Rucki at the University of Washington for providing me with the C++ framework used in this project. Michael Muskulus, Post. Doc., also pointed me to relevant topics.

Per Ivar Bruheim Trondheim, December 2011

Abstract

This report presents an overview over important theory related to the solution of the eigenvalue problem in structural dynamics. Some numerical algorithms are presented, while an elaborate discussion of the inverse iteration method is given. Further, an object-oriented framework featuring the concepts tensor-based computations and coordinate-free element formulations will be explored. An implementation of the inverse iteration method is carried out with the use of the above-mentioned framework, and the results and performance of the implementation are discussed.

Contents

1 Introduction

The purpose of this report is to investigate existing theory and algorithms concerning numerical solutions to the eigenvalue problem in structural dynamics. A description of the eigenvalue problem in a finite element context and its properties are given before the procedure of the *inverse iteration method* is demonstrated and implementated in a modified version, allowing the search for eigenvalue solutions in an arbitrary range.

The features of the object-oriented framework which the implementation was built on will further be explored, with special focus on the features that are relevant for the implementation of the eigenvalue solver. The concept of coordinate-free elements will be demonstrated, having a formulation disconnected from a global coordinate system. The report will not go to deeply into code statements, but rather present the ideas from a mathematical point-of-view. Still, it should be quite clear to the reader having some experience with object-oriented programming how the various operations can be implemented in terms of e.g. overloaded operators.

Traditionally, finite element techniques establish a global stiffness matrix on which the solution procedure operates. This report will briefly discuss how abandoning the global matrix in favor of *tensors*—that themselves reside solely in the objects representing the degrees-of-freedom—not only gives a "cleaner" code, but also demonstrably a *faster* code.

The efficiency of the implemented eigenvalue solver as well as the validity of the calculated results will be investigated. Finally, suitable applications for the solver and its limitations and weaknesses are discussed.

2 Theoretical background

2.1 The eigenvalue problem in structural dynamics

2.1.1 Importance

The concepts of natural frequencies and mode shapes play a central role in the analysis of dynamic response due to fluctuating loads acting on a structure. If the frequency content of the load matches the natural frequency of the structure, resonance phenomena will occur and the response will grow dramatically. The violent behaviour that results from resonance can eventually cause the structure to collapse, and must thus be considered during design. In nature, periodic loads appear as e.g. wind loads and earthquakes.

We mention that knowledge about the natural frequencies is particularly important for wind turbine structures, as they are being exposed to loads with a wide frequency spectrum. A wind turbine's dynamic response will be a key factor for design.

2.1.2 Definition and properties

The dynamic equation of motion for a general finite element system in free, undamped vibration can be stated as

$$\mathbf{M}\ddot{\mathbf{r}}\left(t\right) + \mathbf{K}\mathbf{r}\left(t\right) = \mathbf{0}\,, \tag{2.1}$$

where $\mathbf{M}$ and $\mathbf{K}$ is the system's mass and stiffness matrix, while $\mathbf{r}$ is the vector containing the unknown displacements of the N degrees-of-freedoms (DOFs). A solution to this differential equation is $\mathbf{r}\left(t\right) = \boldsymbol{\phi}\sin(\omega t)$, which inserted into

equation 2.1 yields the familiar eigenproblem in structural dynamics:

$$\left(\mathbf{K} - \omega^2 \mathbf{M}\right) \boldsymbol{\phi} = \mathbf{0}\,, \tag{2.2}$$

or equivalently, by left-multiplying by $\mathbf{M}^{-1}$, the *standard eigenproblem*

$$\left(\mathbf{A} - \omega^2 \mathbf{I}\right) \boldsymbol{\phi} = \mathbf{0} \qquad \text{where } \mathbf{A} = \mathbf{M}^{-1}\mathbf{K} \tag{2.3}$$

This system of equations generally has N solutions: N eigenvectors, $\boldsymbol{\phi}_n$; and their corresponding eigenvalues, $\omega_n^2 = \lambda_n$. Due to $\mathbf{K}$ and $\mathbf{M}$ being symmetric and positive semidefinite, the eigenvalues are real and non-negative. Note that to be precise, the term "eigenvalues of a matrix" means the solution to (2.3), with $\mathbf{A}$ as the matrix.

The resulting eigenvectors give the free vibration shapes, or the *mode shapes*. The corresponding value of $\omega = \sqrt{\lambda}$ is the mode's circular frequency or natural frequency (in rad/s). The mode shapes has the important property of being orthogonal with respect to both the stiffness and mass matrix, i.e.

$$\boldsymbol{\phi}_j^{\mathrm{T}} \mathbf{M} \boldsymbol{\phi}_i = 0 \qquad \text{when } i \neq j$$

$$\boldsymbol{\phi}_j^{\mathrm{T}} \mathbf{K} \boldsymbol{\phi}_i = 0 \qquad \text{when } i \neq j$$

This can be interpreted in the sense that there is no coupling between forces arising from a displacement $\boldsymbol{\phi}_i$ and forces arising from a displacement $\boldsymbol{\phi}_j$, both elastic and inertia forces[2, p.164]. The orthogonality property can further be used in modal analysis, where the response of each mode are calculated independently and superpositioned for the total response.

2.1.3 Eigenvalue shift

It can be shown[4] that the shifted matrix

$$\mathbf{A}_\mu = \mathbf{A} - \mu \mathbf{I}$$

has the same eigenvectors as $\mathbf{A}$ and the eigenvalues $\lambda_i - \mu$, where λ_i is the eigenvalues of $\mathbf{A}$ and μ is the *shift*. This is a property that will be useful in the discussion of the inverse iteration method (Section 2.3).

The equivalent shifted expression of (2.3) is

$$\left((\mathbf{A} - \mu \mathbf{I}) - \lambda \mathbf{I}\right) \mathbf{v} = \mathbf{0} \qquad \text{where } \mathbf{A} = \mathbf{M}^{-1}\mathbf{K}\,. \tag{2.4}$$

We note that the equation above can be written on the form

$$(\mathbf{A} - \mu\mathbf{I})\,\mathbf{v} = \lambda\mathbf{v}\,, \tag{2.5}$$

or equivalently, by left-multiplying by $\mathbf{M}$

$$\underbrace{(\mathbf{K} - \mu\mathbf{M})}_{\mathbf{K}_\mu}\,\mathbf{v} = \lambda\mathbf{M}\mathbf{v}\,. \tag{2.6}$$

2.2 Numerical eigenproblem algorithms

A lot of research has been made on the subject of finding reliable and efficient methods for solving the eigenvalue problem. Solution methods must in general be iterative, as solving the eigenvalue equation is basically equivalent to solving for the roots of a polynomial of degree equal to the number of degrees-of-freedom (i.e. the dimension of $\mathbf{v}$). As is known, no explicit formula exists for polynomials of order > 4.

When selecting a numerical algorithm for a special-purpose eigenvalue solver, several factors should be taken into account:

- Ease of implementation
- Rate of convergence and efficiency
- Numerical stability
- The properties of the problem to be solved: e.g. number of DOFs, number of modes and range of interest, as well as the structure of the coefficient matrices.

Several numerical algorithms exist, although each of them are suited to different applications. The following methods are mentioned in the literature:

- **Transformation methods**. Calculate *all* the eigenvalues simultaneously, and are therefore best suited to small systems, and when all eigenvalues are interesting. Examples are the *Jacobi method* and the *Householder-QR* method.
- **The Sturm sequence**. Determines the number of eigenvalues below a certain trial value λ. Useful for checking that an algorithm has not missed any solutions.

- **The inverse iteration method**. Calculates the eigenpair closest to an initial estimate for the eigenvalue. This algorithm is described more detailed in the next section.
- **Subspace iteration**. A generalization of the inverse iteration method. Widely used to obtain a prescribed number of the lowest eigenpairs.[4]
- **Lanczos method**. Similar to subspace iteration, and is claimed to be from two to ten times faster.
- **Root-finding** of the characteristic polynomial. Generally not a practical method for large systems due to much computational effort required and sensitivity to numerical round-off errors.

Further details of these algorithms can be found in i.e. [4, p. 679][1].

2.3 The inverse iteration method

The inverse iteration method is a simple and effective iterative method for finding an eigenvector close to a guessed eigenvalue. By applying appropriate eigenvalue shifts, *any* eigenvector can be found. In addition, if estimates that are close to the eigenvalues are known a priori, a rapid convergence towards the eigenvectors is achieved. The eigevector's corresponding eigenvalue can further be calculated by the Rayleigh quotient.

The inverse iteration method is basically a modified version of the *power method* [6, Section 20.8], and where the power method converges to the *highest* eigenvalue, the inverse method gives the lowest, fundamental eigenvalue.

2.3.1 Procedure

As the name suggests, the method operates on the inverse of the matrix of which eigenvalues are wanted. In essence, the method repeatedly multiplies an estimated eigenvector by the inverse of the coefficient matrix (see (2.3)).

$$\mathbf{v}_{i+1} = \mathbf{A}^{-1}\lambda\mathbf{v}_i \,, \tag{2.7a}$$

or equivalently

$$\mathbf{v}_{i+1} = \mathbf{K}^{-1}\left(\lambda\mathbf{M}\mathbf{v}_i\right) \,, \tag{2.7b}$$

where $\mathbf{A} = \mathbf{M}^{-1}\mathbf{K}$ in general will be unsymmetric.

The factor λ can be set equal to 1, as scaling the eigenvector will not affect the final result. The initial guess for the eigenvector, $\mathbf{v}_0$ can be taken as *any* non-zero vector, with the only requirement that it is not orthogonal to the correct eigenvector. If $\mathbf{v}_0$ were to be orthogonal, convergence would not be achieved (in exact arithmetic). For simplicity, the initial guess can be taken as

$$\mathbf{v}_0 = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^{\mathrm{T}} ,$$

or some other non-zero random vector. The last equation will be the preferred form, as the symmetry of the stiffness matrix $\mathbf{K}$ is maintained. This will be beneficial when solving. Of course, calculating the inverse is expensive, so instead (2.7b) should be solved using i.e. Gaussian elimination or some other equation solving technique.

Between iterations, the new vector estimate is scaled by a factor so that its length doesn't grow too large or too small, which may cause numerical problems–typically so that the largest vector element is made equal to unity. By denoting the scaled vector by $\hat{\mathbf{v}}$, the scaling reads

$$\hat{\mathbf{v}}_{i+1} = \frac{\mathbf{v}_{i+1}}{\max\left(\mathbf{v}_{i+1}\right)} .$$

After a sufficient number of iterations, solving and scaling each time, this iteration scheme converges to the eigenvector corresponding to the lowest eigenvalue of $\mathbf{A}$, and thus the eigenvector which solves the eigenvalue equation. A proof of the convergence is given in 2.3.3.

Having calculated the eigenvector, an updated estimate for the corresponding eigenvalue can be computed by the Rayleigh quotient

$$\lambda_{i+1} = \frac{\hat{\mathbf{v}}_{i+1}^{\mathrm{T}}\mathbf{K}\hat{\mathbf{v}}_{i+1}}{\hat{\mathbf{v}}_{i+1}^{\mathrm{T}}\mathbf{M}\hat{\mathbf{v}}_{i+1}} = \frac{\mathbf{v}_{i+1}^{\mathrm{T}}\mathbf{M}\hat{\mathbf{v}}_{i}}{\mathbf{v}_{i+1}^{\mathrm{T}}\mathbf{M}\mathbf{v}_{i+1}} \tag{2.8}$$

and a criterion for determining sufficient convergence and aborting the iterations can be taken as

$$\frac{\lambda_{k+1} - \lambda_k}{\lambda_{k+1}} \leq \epsilon \tag{2.9}$$

with $\epsilon = 10^{-2s}$ if $2s$-digit accuracy as desired for the eigenvalue. The accuracy of the eigenvector will be s.

The algorithm for solving for the first eigenvalue is listed in Algorithm 2.1.

Algorithm 2.1 Inverse iteration, fundamental eigenvalue

Require: $\mathbf{v}_0 \neq \mathbf{0}$, $\lambda_0 = 0$

Require: $\mathbf{v}_0$ not orthogonal to correct solution

while $(\lambda_{k+1} - \lambda_k)/\lambda_{k+1} > \epsilon$ **do**

$\mathbf{v}_{k+1} \Leftarrow \mathbf{K}^{-1}\mathbf{M}\hat{\mathbf{v}}_k$

if $\max(\mathbf{v}_{k+1}) > 0$ **then**

$c \Leftarrow \max(\mathbf{v}_{k+1})$

else

$c \Leftarrow |\min(\mathbf{v}_{k+1})|$

end if

$\hat{\mathbf{v}}_{k+1} \Leftarrow \mathbf{v}_{k+1}/c$

$\lambda_{k+1} = \frac{\hat{\mathbf{v}}_k^{\mathrm{T}}\hat{\mathbf{v}}_k}{\mathbf{v}_{k+1}^{\mathrm{T}}\hat{\mathbf{v}}_k}$

end while

2.3.2 Inverse iteration with shifts

The basic inverse iteration method will only find the first eigenfrequency and eigenvector. However, by applying appropriate eigenvalue shifts, *any* solution can be found.

As shown in Section 2.1.3, an eigenvalue shift will produce a system with the same eigenvectors, whereas the new eigenvalues will be decrease by the same amount as the size of the shift. This property can be exploited, namely by first performing the subtraction $\mathbf{K}_\mu = \mathbf{K} - \mu\mathbf{M}$ and use this shifted stiffness matrix for the inverse iterations. Here μ is the shift, and subscript $_\mu$ is used for the shifted stiffness matrix. The procedure no longer converges to the lowest eigenvalue—instead it converges to the eigenvalue which is closest to the shift on the eigenvalue axis. Thus, if estimates to the eigenvalues are known, convergence to any of these can be reached. The eigenvalue is computed by adding the shift to the eigenvalue solution of the inverse iterations:

$$\lambda = \lambda_\mu + \mu \tag{2.10}$$

Finally, we note that if no knowledge of the eigenvalues exists, an "ad-hoc" possibility for enumerating several solutions is to gradually increase the shift until convergence to a new eigenvalue is reached. This modification is implemented and further discussed in Section 4.2.

2.3.3 Convergence and error

The convergence properties of the method depends mainly on where the shift is located relative to the eigenvalues. A worst-case situation is when the shifted position is exactly halfway between two solutions. In exact arithmetic, no amount of iterations will give a converged solution in this situation. See Section 4.3.2 for a study on the effect of the shift on the iteration count.

If the shift point is relatively close to the correct solution, it can be shown that the convergence rate is approximately linear. Defining $\epsilon = \mu - \lambda$, it can be shown that each iteration is expected to increase the accuracy of the eigenvector by a factor of

$$\frac{|\epsilon|}{|\lambda + \epsilon - \lambda_{\text{closest to } \lambda}|} \tag{2.11}$$

Proof of convergence. Taking the correct eigenvectors as $\boldsymbol{\phi}_n$, and acknowledging that these are orthogonal, the iterate v can be written as a linear combination of these eigenvectors. This sum is called the modal expansion of $\mathbf{v}$:

$$\mathbf{v} = \sum_{n=1}^{N} \boldsymbol{\phi}_n q_n = \sum_{n=1}^{N} \boldsymbol{\phi}_n \frac{\boldsymbol{\phi}^{\mathrm{T}} \mathbf{M} \mathbf{v}}{\boldsymbol{\phi}^{\mathrm{T}} \mathbf{M} \boldsymbol{\phi}_n} \tag{2.12a}$$

Substituting the modal expansion of $\mathbf{v}_0$ into (2.7b), the solution after the first inverse iteration is given as

$$\mathbf{v}_1 = \sum_{n=1}^{N} \mathbf{K}^{-1} \mathbf{M} \boldsymbol{\phi}_n q_n \tag{2.12b}$$

From (2.2) we find that $\mathbf{K}^{-1}\mathbf{M}\boldsymbol{\phi}_n = (1/\lambda_n)\,\boldsymbol{\phi}_n$, and by substituting into the equation above

$$\mathbf{v}_2 = \sum_{n=1}^{N} \frac{1}{\lambda_n} \boldsymbol{\phi}_n q_n = \frac{1}{\lambda_1} \sum_{n=1}^{N} \frac{\lambda_1}{\lambda_n} \boldsymbol{\phi}_n q_n \tag{2.12c}$$

For iteration number k, the sum reads

$$\mathbf{v}_{k+1} = \frac{1}{(\lambda_1)^k} \sum_{n=1}^{N} \left(\frac{\lambda_1}{\lambda_n} \right)^k \boldsymbol{\phi}_n q_n \,. \tag{2.12d}$$

Since $\lambda_1 < \lambda_n$ for $n > 1$, we acknowledge that $(\lambda_1/\lambda_n)^k \overset{k\to\infty}{\longrightarrow} 0$ for all $n \neq 1$. Thus $\mathbf{v}_{k+1}$ converges to a vector proportional to $\boldsymbol{\phi}_1$.

(2.12e)

□

A similar proof can be stated for the case where an eigenvalue shift is applied. For further details, see [3, Section 10.13.2].

2.3.4 Remarks

- The non-orthogonality requirement is rarely an issue, as in floating point arithmetic round-off errors will ensure that some component exists in the direction of the correct eigenvector, $\mathbf{v}$. If a random vector is generated as the initial guess, the probability of an orthogonal initial guess is in all practice zero.
- Repeated eigenvalues will not give two solutions, but rather the solution closest to the initial guess.
- Some eigenvalue problems may have an *eigenplane* as a solution instead of a unique eigenvector, as is the case with i.e. an axi-symmetric cantilever beam. In these cases, the inverse iterations will converge to the direction closest to the initial guess.
- The coefficient matrix resulting from applying an eigenvalue shift will have to be refactored before solving, which can be an expensive operation.
- The rate of convergence can be improved (one order) by shifting $\mathbf{K}$ by the most recent eigenvalue estimate after each iteration. The algorithm is in this case called *Rayleigh quotient iterations*, and has the disadvantage that the system will have to be refactored for every *iteration*. This method may also converge to some other eigenvalue than the one closest to the shift, if measures to avoid this is not employed.

2.4 The mass matrix

2.4.1 Introduction

Different mass matrix formulations can be employed for the eigenvalue problem. Where i.e. lumped masses yield a diagonal mass matrix which easily can be inverted, a consistent mass matrix is more accurate at the expense of more demanding calculations.

Elements with lumped masses will seem heavier than they really are, and hence will give a lower eigenfrequency than the exact value. However, with a fine mesh this effect will be negligible. A consistent mass matrix is derived using the element's shape functions, and thus gives an element where kinetic and potential energy is in balance. Consistent masses will in general give *upper* bounds on the natural frequencies, but not always[7, Section 5.4].

One problem arising when using an ordinary lumped-mass formulation is that the mass terms for the rotational DOFs usually will be set to zero. This will give rise to eigenmodes with no distinct solution for the eigenvalue (i.e. an infinite eigenvalue). To overcome this problem, the rotational DOFs must either be eliminated by the use of static condensation or, alternatively, by introducing some non-zero mass terms for the rotational DOFs. *Ad hoc* methods exist for including rotational terms, which is easier to implement than static condensation. The quite useful "HRZ" lumping method will be presented in the following.

2.4.2 HRZ lumping

The HRZ lumping technique, named after its authors, is an ad hoc method that appears to be quite successful for beam elements, yielding a diagonal mass matrix with non-zero and positive rotational terms. With a HRZ lumped mass matrix the resulting natural frequencies of the elements will be somewhere between the lumped and consistent formulation[4, p. 380][2].

The idea is to take only the diagonal terms from the consistent mass matrix, and scale them so that the total element mass is correct. For a 2D 6-DOF beam element

with total mass m and length L, the diagonal of the consistent mass matrix reads

$$\mathbf{M} = \frac{m}{420}\begin{bmatrix} 140 & 0 & 0 & 0 & 0 & 0 \\ 0 & 156 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4L^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 140 & 0 & 0 \\ 0 & 0 & 0 & 0 & 156 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4L^2 \end{bmatrix},$$

where the DOFs are $\mathbf{d} = \begin{bmatrix} u_1 & v_1 & \theta_{z1} & u_2 & v_2 & \theta_{z2} \end{bmatrix}^{\mathrm{T}}$. For a translational displacement in the v direction, the sum of the mass terms is $\frac{m}{420}(156 + 156) = \frac{312}{420}m$. By enforcing the total mass for any translational displacement to be m, we scale all the diagonal terms contributing to v displacement by the factor $\frac{420}{312}$. Similarily, in the u direction the sum of the terms is $\frac{m}{420}(140 + 140) = \frac{2}{3}m$. No other terms than the u translation terms will contribute to u displacement, so only the mass terms for the u direction itself will be scaled.

The resulting mass matrix is

$$\mathbf{M} = m\begin{bmatrix} 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & L^2/78 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & L^2/78 \end{bmatrix}.$$

For the 3D case, extra $1/2$ terms appear for the w direction, and $L^2/78$ terms for the θ_y direction. The mass terms corresponding to the torsional rotations (θ_x) will, however, depend on the element's cross-section. The corresponding consistent mass term is given as

$$M_{\theta_x\theta_x} = \frac{mI_p}{3}$$

where I_p is the polar moment of inertia.

For the case with a thin-walled axi-symmetric cross-section, the moment of inertia is $I = r^2$. For the full 12-DOF, 3D element with a thin-walled cross-section (e.g. a pipe) using HRZ lumping the mass matrix for one node then reads:

$$\mathbf{M}_1 = m\begin{bmatrix} 1/2 & 1/2 & 1/2 & r^2/3 & L^2/78 & L^2/78 \end{bmatrix} \tag{2.13}$$

where the nodal displacements for one node are taken as

$$\mathbf{d} = \begin{bmatrix} u_1 & v_1 & w_1 & \theta_x & \theta_y & \theta_z \end{bmatrix}^{\mathrm{T}} \tag{2.14}$$

3 An object-oriented finite element framework

3.1 Introduction

The finite element framework used for this research is the work done by Rucki[11], Miller[9] and Thomassen[12], and is a framework has been written and developed on an underlying object-oriented philosophy.

The object-oriented approach gives code which is easier to maintain, re-use and extend. In addition, since objects often represent some real-world entity, the structure of the code and the interaction between different code units will appear more clearly and intuitive than for a non-object-oriented language (i.e. a procedure-oriented language). In general, an object-oriented design gives potential for a higher level of abstraction—providing an interface for the developer more closely tied to the underlying mathematical or physical description.

A common conception is that, compared to traditional solvers, such high-level abstractions will at some point be detrimental to performance, if utilized too deeply into numerical calculations. However, research shows that performance on par with traditional solvers can be achieved with the approach used in this framework. Even an implementation on the Java platform, which usually performs worse than its C++ or Fortran counterparts, has been demonstrated to be competitive[10].

The key features of the framework are

- Tensors and vectors instead of matrices and scalars used as the fundamental quantities for defining the underlying mathematical problem.
- An isoparametric, coordinate-free formulation of the element.
- A high level of abstraction, in terms of objects, down to the core equation-solving level.

- Focus on allowing interactive manipulation of the model and analysis parameters in a rich graphics context.
- Providing real-time feedback of results (i.e. live modeling), avoiding the classical pre- and post-processing separation.

The features listed here are usually not seen in conventional finite element software.

This chapter will outline the important features of the framework, with focus on the topics concerning the extension of the framework done in Section 4.2.

3.2 A coordinate-free element formulation

3.2.1 Stiffness relations

Miller et al. presents a mathematical derivation[10] of an arbitrary element's stiffness properties, where no assumption is made on the properties of the global coordinate system, rendering expressions for the stiffness tensors with no explicit reference to global coordinates. The approach eliminates the need for explicit transformations between local and global coordinates, which instead will be implicitly handled in the coordinate-free expressions. The properties of the global coordinate system itself is encapsulated in the implementation of the vector and tensor classes and their mathematical operators.

The basic steps for obtaining coordinate-free expressions for nodal forces and subsequently for the stiffness tensors are here given for a straight beam element, only considering small deflections. More details can be found in [8].

The position of points within the beam element can be parametrized in terms of the parameter ξ as

$$\boldsymbol{\phi}_o(\xi) = \mathbf{u}_i + \xi(\mathbf{u}_j - \mathbf{u}_i) \tag{3.15}$$

where $\xi \in [0, 1]$ and $\mathbf{u}_{i,j}$ is the position of the nodes. By taking the derivative with respect to ξ, the basis in the physical space is found as

$$\mathbf{g}_1 = \frac{\partial \boldsymbol{\phi}_0}{\partial \xi} = \mathbf{u}_j - \mathbf{u}_i = L\mathbf{n} \tag{3.16}$$

where L is the element length and $\mathbf{n}$ is the unit direction vector of the element. The dual basis (see Appendix A) is simply $\mathbf{g}^1 = \frac{1}{L}\mathbf{n}$.

By introducing the usual shape functions of a beam, the parametrized displacement of the beam centerline due to bending can be expressed in terms of the nodal displacements $\mathbf{d}_{i,j}$ and $\boldsymbol{\theta}_{i,j}$ as

$$\begin{aligned}\mathbf{u}_0(\xi) = N_1(\xi)\left(\mathbf{d}_i - \mathbf{g}_1\left(\mathbf{g}^1 \cdot \mathbf{u}_1\right)\right) + N_2(\xi)\left(\boldsymbol{\theta}_i \times \mathbf{g}_1\right) + \\ N_3(\xi)\left(\mathbf{d}_j - \mathbf{g}_1\left(\mathbf{g}^1 \cdot \mathbf{u}_1\right)\right) + N_4(\xi)\left(\boldsymbol{\theta}_j \times \mathbf{g}_1\right)\end{aligned} \tag{3.17}$$

and the total displacement of an arbitrary point $\mathbf{x}$ can be written as

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}_0 + \left(\mathbf{g}^1 \times \mathbf{u}_0'\right) \times \mathbf{s} \tag{3.18}$$

where $\mathbf{s} = \mathbf{x} - \boldsymbol{\phi}_0(\xi)$. Further, the derivative $\partial\mathbf{u}/\partial\xi$ is found as

$$\frac{\partial \mathbf{u}}{\partial \xi} = \mathbf{u}_0' + \left(\mathbf{g}^1 \times \mathbf{u}_0''\right) \times \mathbf{s} \tag{3.19}$$

Substituting the above into the expression for the small strain tensor, noting that $\boldsymbol{\nabla} = \mathbf{g}^1 \frac{\partial}{\partial \xi}$ gives

$$\boldsymbol{\epsilon} = \frac{1}{2}\left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^{\mathrm{T}}\right] \tag{3.20}$$

and when adding the strain term for pure axial tension, the total strain in the axial direction simplifies to

$$\boldsymbol{\epsilon}_{nn} = \frac{1}{L^2}\left(\mathbf{u}_0'' \cdot \mathbf{s}\right) + \frac{1}{L}\left(\mathbf{u}_0' \cdot \mathbf{n}\right) \tag{3.21}$$

Space does not allow for the full derivation here. However, when having developed an expression for the strain tensor, assuming a linear material

$$\boldsymbol{\sigma}_{nn} = E\boldsymbol{\epsilon}_{nn}$$

and applying the principle of virtual work, the equivalent nodal forces and moments is found as

$$\begin{aligned}
\mathbf{f}_i &= \left[\frac{12E}{L^3}\hat{\mathbf{I}} + \frac{AE}{L}\mathbf{n}\otimes\mathbf{n}\right](\mathbf{u}_i - \mathbf{u}_j) + \frac{6E}{L^2}\left(\hat{\mathbf{I}} \times \mathbf{n}\right)(\boldsymbol{\theta}_i + \boldsymbol{\theta}_j) \\
\mathbf{f}_j &= \left[\frac{12E}{L^3}\hat{\mathbf{I}} + \frac{AE}{L}\mathbf{n}\otimes\mathbf{n}\right](\mathbf{u}_j - \mathbf{u}_i) - \frac{6E}{L^2}\left(\hat{\mathbf{I}} \times \mathbf{n}\right)(\boldsymbol{\theta}_i + \boldsymbol{\theta}_j) \\
\mathbf{m}_i &= \frac{6E}{L^2}\left(\hat{\mathbf{I}} \times \mathbf{n}\right)(\mathbf{u}_i - \mathbf{u}_j) + \frac{2E}{L}\left(\mathbf{n} \times \hat{\mathbf{I}} \times \mathbf{n}\right)(2\boldsymbol{\theta}_i + \boldsymbol{\theta}_j) + \frac{GJ}{L}\mathbf{n}\otimes\mathbf{n}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j) \\
\mathbf{m}_j &= \frac{6E}{L^2}\left(\hat{\mathbf{I}} \times \mathbf{n}\right)(\mathbf{u}_i - \mathbf{u}_j) + \frac{2E}{L}\left(\mathbf{n} \times \hat{\mathbf{I}} \times \mathbf{n}\right)(\boldsymbol{\theta}_i + 2\boldsymbol{\theta}_j) + \frac{GJ}{L}\mathbf{n}\otimes\mathbf{n}(\boldsymbol{\theta}_j - \boldsymbol{\theta}_i)
\end{aligned} \tag{3.22}$$

where $\hat{\mathbf{I}} = I_{ss}\,(\mathbf{s}\otimes\mathbf{s}) + I_{tt}\,(\mathbf{t}\otimes\mathbf{t})$ is the second moment of area tensor and J is the torsional stiffness. We see that the stiffness relations are expressed completely without any reference to a coordinate system.

By collecting terms, the equations may be written as

$$\begin{bmatrix} \mathbf{K}_{fu} & \mathbf{K}_{f\theta} & -\mathbf{K}_{fu} & -\mathbf{K}_{f\theta} \\ \mathbf{K}_{mu} & \mathbf{K}_{m\theta} & -\mathbf{K}_{mu} & -\hat{\mathbf{K}}_{m\theta} \\ -\mathbf{K}_{fu} & -\mathbf{K}_{f\theta} & \mathbf{K}_{fu} & \mathbf{K}_{f\theta} \\ -\mathbf{K}_{mu} & -\hat{\mathbf{K}}_{m\theta} & \mathbf{K}_{mu} & \mathbf{K}_{m\theta} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \boldsymbol{\theta}_1 \\ \mathbf{u}_2 \\ \boldsymbol{\theta}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{m}_1 \\ \mathbf{f}_2 \\ \mathbf{m}_2 \end{bmatrix} \tag{3.23}$$

where the individual stiffness tensors are given as

$$\mathbf{K}_{fu} = \frac{12E}{L^3}\hat{\mathbf{I}} + \frac{AE}{L}\,(\mathbf{n}\otimes\mathbf{n}) \tag{3.24a}$$

$$\mathbf{K}_{mu} = \mathbf{K}_{f\theta}^{\mathrm{T}} = \frac{6E}{L^2}\left(\mathbf{n}\times\hat{\mathbf{I}}\right) \tag{3.24b}$$

$$\mathbf{K}_{m\theta} = \frac{4E}{L}\hat{\mathbf{I}} + \frac{JG}{L}\,(\mathbf{n}\otimes\mathbf{n}) \tag{3.24c}$$

$$\hat{\mathbf{K}}_{m\theta} = \frac{2E}{L} - \frac{JG}{L}\,(\mathbf{n}\otimes\mathbf{n}) \tag{3.24d}$$

Again, no explicit references to coordinates appear.

For the 3D case, all vectors will have 3 components, and the tensors will have 9 components. It is noted that the geometric classes will eventually have to use some form of coordinates *internally*, in order to carry out the computations.

3.2.2 Example: Calculating nodal forces for a simple beam element

A simple example will be presented here, in order to demonstrate that the formulation given above essentially is equivalent to applying a standard coordinate transformation matrix to the local stiffness matrix.

Example 3.2.1. Take a simple beam element with its two nodes at positions $\mathbf{u}_1 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$ and $\mathbf{u}_2 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^{\mathrm{T}}$. The unit direction vector is then $\mathbf{n} = \mathbf{u}_2 - \mathbf{u}_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^{\mathrm{T}}$. The directions of the strong and weak axes can then be taken as $\mathbf{s} = \begin{bmatrix}0 & 0 & -1\end{bmatrix}^{\mathrm{T}}$ and $\mathbf{t} = \mathbf{s}\times\mathbf{n} = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^{\mathrm{T}}$, respectively.

First we se that the tensor products in (3.22) quite simply becomes

$$\begin{aligned}
\mathbf{n} \otimes \mathbf{n} = \mathbf{n}\mathbf{n}^{\mathrm{T}} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\mathbf{s} \otimes \mathbf{s} = \mathbf{s}\mathbf{s}^{\mathrm{T}} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
\mathbf{t} \otimes \mathbf{t} = \mathbf{t}\mathbf{t}^{\mathrm{T}} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{3.25}$$

By e.g. writing out the full 3x3 stiffness tensor $\mathbf{K}_{fu}$ (which relates $\mathbf{u}_1$ to $\mathbf{f}_1$):

$$\mathbf{K}_{fu} = \frac{12E}{L^3}\left(I_{ss} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + I_{tt} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) + \frac{AE}{L} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.26}$$

we find that a unit displacement of $\mathbf{u}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$ gives the nodal force in node 1

$$\mathbf{f}_1 = \mathbf{K}_{fu}\mathbf{u}_1 = \frac{EA}{L} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.27}$$

which is expected, as a pure axial displacement should result in a pure axial force. If the nodal displacement is taken to be $\mathbf{u}_1 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$, the nodal force in node 1 becomes

$$\mathbf{f}_1 = \mathbf{K}_{fu}\mathbf{u}_1 = \frac{12E}{L^3} I_{ss} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.28}$$

which gives a shear force in the y direction, as expected. By applying a translation containing components in both directions, the resulting nodal force will be a combination of the expressions above. We note that the tensor expressions automatically do what a traditional transformation matrix with direction cosines would do, however in an arguably more "clever" manner.

3.2.3 Expressions for the mass tensors

By pursuing the coordinate-free formulation of the element's stiffnes tensors, an equal approach should be used for the mass equivalent. If the HRZ lumping scheme (Section 2.4.2) is employed, the nodal acceleration–force relation reads

$$\begin{bmatrix} \mathbf{M}_{fu} & 0 & 0 & 0 \\ 0 & \mathbf{M}_{m\theta} & 0 & 0 \\ 0 & 0 & \mathbf{M}_{fu} & 0 \\ 0 & 0 & 0 & \mathbf{M}_{m\theta} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{u}}_1 \\ \ddot{\boldsymbol{\theta}}_1 \\ \ddot{\mathbf{u}}_2 \\ \ddot{\boldsymbol{\theta}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{m}_1 \\ \mathbf{f}_2 \\ \mathbf{m}_2 \end{bmatrix} \tag{3.29}$$

where the individual mass tensors are

$$\mathbf{M}_{fu} = \frac{m}{2} \left(\mathbf{n} \otimes \mathbf{n} + \mathbf{s} \otimes \mathbf{s} + \mathbf{t} \otimes \mathbf{t} \right) \tag{3.30a}$$

$$\mathbf{K}_{mu} = \frac{mL^2}{78} \left(\mathbf{s} \otimes \mathbf{s} + \mathbf{t} \otimes \mathbf{t} \right) + \frac{mr^2}{3} \left(\mathbf{n} \otimes \mathbf{n} \right) \tag{3.30b}$$

where the element has been assumed to have a thin-walled cross-section, giving a (mass) moment of inertia equal to mr^2.

4 Implementation and results

4.1 Introduction

A modified eigenvalue algorithm based on the inverse iteration method is here proposed and implemented by use of the framework described in Chapter 3. The results computed with the solver are verified by comparing to an exact expression for a cantilever beam problem. Further, the efficiency and convergence rate of the code is investigated. A short demonstration of how the code interacts with and extends the framework is given, as well as a demonstration of the visualization of the model's resulting eigenvectors.

4.2 Implementation of the inverse iteration algorithm

4.2.1 The modified algorithm

The basis for the algorithm is the inverse iteration method, outlined in 2.3. As shown earlier, this method in its basic form can only extract the lowest eigenvalue of the system at hand. In order to enumerate eigenpair solutions in some range, say $\lambda \in [0, \lambda_{max}]$, a "brute-force" approach is here suggested and implemented, where the shift point is incrementally moved to the right on the eigenvalue axis. After each shift increment, inverse iterations are performed. If the iterations converge forward/right, a new eigenvalue has been found, and the incrementing can start over again from this new point.

The initial increment of the shift is set by the analyst, and its value limits the lower resolution on the eigenvalue axis. If the shift increment is set too large solutions may therefore be missed. Exploiting the fact that the inverse iterations converge to the eigenvalue closest to the shift point, the size of the shift can be *doubled*

after each iteration when converging back to the previous value, without missing any solutions. The doubling of the shift significantly reduces the complexity of the algorithm, as will be seen in Section 4.2.2.

We further take the following parameters as inputs to the eigenvalue solver

- $\Delta\mu_0$, initial shift/minimum search resolution (on the eigenvalue axis).
- $\lambda_{\max}$, upper bound on λ.
- N, maximum number of solutions to find.
- $\mathbf{K}_0$, the initial stiffness matrix and $\mathbf{M}$, the mass matrix.

The essential steps of the algorithm is listed in Algorithm 4.2.

Algorithm 4.2 Eigenvalue search using inverse iterations with incrementing shifts

$\lambda_0 \Leftarrow 0$
for $i = 1 \rightarrow N$ **do**
  $\lambda_i \Leftarrow -\infty$
  $\mu \Leftarrow \Delta\mu_0$ (initial shift)
  while $\lambda_i - \lambda_{i-1} < \Delta\mu_0$ and $\lambda_{i-1} + \mu < \lambda_{\max}$ **do**
    $\mathbf{K} \Leftarrow \mathbf{K}_0 - (\lambda_{i-1} + \mu)\,\mathbf{M}$
    loop
      $\mathbf{v}_i \Leftarrow \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^{\mathrm{T}}$
      $\mathbf{v}_i \Leftarrow \mathbf{K}^{-1}(\mathbf{M}\mathbf{v}_i)$
      $c \Leftarrow \max\{|v_m|\}$ (largest absolute value of vector elements)
      $\hat{\mathbf{v}} \Leftarrow \mathbf{v}_i / c$
      $\lambda^* = \dfrac{\hat{\mathbf{v}}^{\mathrm{T}}\hat{\mathbf{v}}}{\mathbf{v}_i^{\mathrm{T}}\hat{\mathbf{v}}}$
      if $(\lambda^* - \lambda)/\lambda^* < \epsilon$ **then**
        $\lambda_i \Leftarrow \lambda^*$
        break loop (inverse iterations converged)
      end if
      $\lambda_i \Leftarrow \lambda^*$
    end loop
    $\mu \Leftarrow 2\mu$
  end while
end for

4.2.2 Time analysis

Assuming a solution λ_j has just been found and that the solver proceeds to gradually shift the system in order to find the next solution. Let the distance to the next eigenvalue be d and let the minimum search resolution be some multiple of this distance: $\Delta\mu_0 = c \cdot d$, with $c << 1$. Further, we assume that an upper bound, U, on the number of inverse iterations (for each shift increment) is enforced, so that if this limit is reached it can safely be assumed that the shift is not in the vicinity of a solution, and thus can be aborted. Although the number of iterations needed to reach the desired tolerance varies somewhat linearly with the distance from the shifted position to the eigenvalue (see 2.3.3), we can conservatively take this number equal to the maximum limit enforced above, U.

For a system with n degrees of freedom, presumably with a dense coefficient matrix, the cost of the different operations performed is

- LU decomposition of the system of equations: $n^3/3$ operations.
- Calculating the right hand side, $\mathbf{M}\mathbf{v}_i$: $\sim n$ operations.
- Solving the system of equations: $\sim n^2$ operations.
- Calculating the Rayleigh quotient: $\sim n^2$ operations.
- Resetting and shifting the stiffness matrix: $\sim n$ operations.

It is noted that for a system of equations with a mainly digonal structure, the expression above will be different.

We remind us that the iterations converge to the closest solution, so the critical point is half-way between the solutions. By doubling the shift for each increment, the number of increments k needed is found by

$$\begin{aligned} (\Delta\mu_0) \cdot 2^k &= \frac{d}{2} \\ \implies k &= \log_2\left(\frac{d}{2\Delta\mu_0}\right) = \log_2\left(\frac{d}{c \cdot 2d}\right) \\ &= \log_2\left(\frac{1}{2c}\right) \end{aligned} \tag{4.31}$$

The total number of operations needed for the next solution is then

$$\begin{aligned} T &= k\left[n^3/3 + U \cdot n + U \cdot n^2 + n\right] \\ &= log_2\left(\frac{1}{2c}\right)\left[n^3/3 + U \cdot n + U \cdot n^2 + n\right] \qquad (4.32) \\ &\approx log_2\left(\frac{1}{2c}\right)\left[n^3/3 + U \cdot n^2\right] \qquad (4.33) \end{aligned}$$

If N eigenvalue solutions are wanted, the total number of operations will be $\sim NT$, although c will vary if $\Delta\mu$ is kept constant (as will be the case when this parameter is selected by the analyst). As expected, the running time will depend on how the eigenvalues are distributed. Still, we note the logarithmic dependency on the ratio $d/\Delta\mu_0$.

4.2.3 Interface to the framework

Functionality in the framework for generating finite element models and generating the stiffness of the elements already existed. Some modifications were made in order to introduce well-behaving mass tensors, as discussed in Section 3.2.3. The framework's built-in Gaussian elimination solver were used to solve the system of equations in each iteration.

The framework's well-designed interfaces allowed for an easy implementation of new functionality, almost without modifying existing code. The major implementation steps were:

- Extension of the `GaussElimSolver` class to the new class `EigenvalueSolver`, in order to customize the solving functions and i.e. intercept the process of creating DOFs. All the eigenvalue solver code was implemented in this class.
- Extension of the `GaussElimDOF` class to the new class `EigenvalueDOF`, in order to add mass properties to the DOFs. Functions were added for shifting the stiffness tensor (see Section 2.1.3), and for storing and retrieving the displaced position vector for the DOF for each of the mode shapes that were found.
- Extension of the `GaussElimInteraction` class to `EigenvalueInteraction`, in order to add mass properties for the interactions between DOFs.

The declarations/headers of the implemented classes can be seen in Appendix B.

4.3 Results

4.3.1 Verification of solutions

A comparison between eigenvalues calculated by the solver and analytical values is here given for a cantilevered beam model. The eigenvectors were not specifically verified, except by visual inspection of animations of the shapes. Solutions for another, more complex model is also given, however due to time constraints these values could not be compared with other solvers.

- Cantilevered beam modelled with 50 3D elements having an axi-symmetric cross-section.
- The NREL 5-MW wind turbine model, with a tower consisting of a monopile on top of a truss tower, in addition to three blades.

Cantilevered beam
The eigenvalues of a cantilevered axi-symmetric beam was calculated using the following properties:

$$E = 210 \cdot 10^9\,\text{Pa},\ G = 80.2 \cdot 10^9\,\text{Pa},\ I = 3.927 \cdot 10^{-3}\,\text{m}^4$$
$$m = 246.6\,\text{kg/m},\ A = 2\pi rt = 84.8 \cdot 10^{-3}\,\text{m}^2 \text{ and } L = 10\,\text{m}\,.$$

The calculated values of the eigenfrequencies are given in Table 4.1, compared to exact values [2, p.75] for the bending modes. We see from table 4.1 that the solver gives almost identical results to the exact values. The error in the last digit is most likely caused by a round-off error, as the accuracy of the solver was set to approximately 4 digits ($\epsilon_\lambda = 10^{-7}$).

Table 4.1: Calculated eigenfrequencies for cantilever beam

n	f_n [Hz]	f_n^{exact} [Hz]	Mode identification
1	10.23	10.23	First bending
2	64.12	64.12	Second bending
3	80.19	-	First torsional
4	129.3	-	First axial
5	179.5	179.6	Third bending
6	240.6	-	Second torsional

NREL 5MW model
The 10 first eigenvalues of a the NREL 5-MW reference turbine[5] were calculated. Due to the mass distributions and stiffnesses of the blades and nacelle being wrong,

the results are not correct. Still, they will be presented here. The eigenvalues are for a standstill turbine with the blades set with one blade pointing downwards, parallel to the tower.

The values used for the tower were

$$E = 210 \cdot 10^9\,\text{Pa},\ G = 80.2 \cdot 10^9\,\text{Pa},\ d_{\text{bottom}} = 6\,\text{m},\ d_{\text{top}} = 3.87\,\text{m}$$
$$t_{\text{bottom}} = 0.027\,\text{m},\ t_{\text{top}} = 0.019\,\text{m},\ \rho = 8500\,\text{kg/m}^3$$

and a pipe cross-section were used. The tower was modelled with 11 elements. Results are presented in Table 4.2, although all the modes could not be identified.

Table 4.2: Calculated eigenfrequencies of NREL wind turbine model

n	f_n [Hz]	Mode identification
1	0.6241	Tower side-to-side
2	0.6329	Tower fore-aft
3	1.138	Tower torsion
4	1.219	–
5	1.257	–
6	1.314	–
7	1.370	–
8	1.422	–
9	3.620	Second tower fore-aft
10	3.633	Second tower side-to-side

4.3.2 Convergence

A plot demonstrating the effect of the shift point on the iteration count needed for convergence is shown in Figure 4.1. The convergence tolerance was set to $\epsilon_\lambda = 10^{-7}$, and the model used was a model of the NREL wind turbine. When the shift point approaches the mid-point between two eigenvalues, the number of iterations needed approaches inifinity—as is expected. The lower peaks correspond to the correct solutions, where only a few iterations is required.

Figure 4.2 shows how the solution time for the 5 first eigenvalues of the cantilever beam problem scales with the number of elements used. The discrepancy with (4.33) is most likely due to the fact that a tower structure will have a diagonal equation system, where as the numbers given in (4.33) is for a system assumed to be dense.

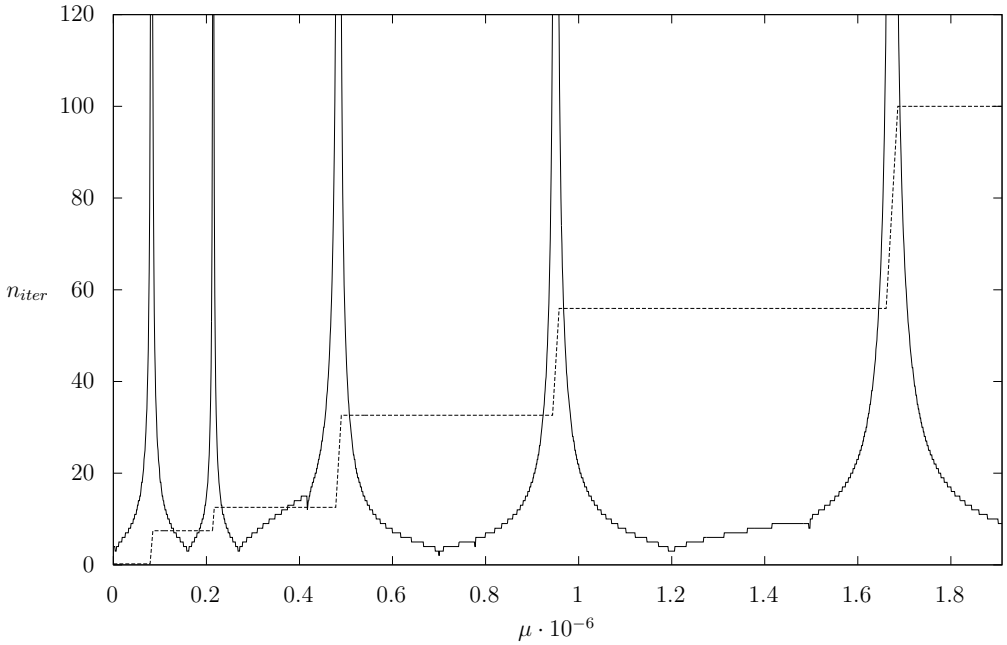


Figure 4.1: Number of inverse iterations needed for convergence vs. shift point. Dashed lines indicate the value of the converged solution, scaled as to fit in the plot. Lower peaks correspond to the correct solutions.

4.4 Visualization and user interface

The framework allowed for easy use of existing visualization functionality and code already existed (using *OpenGL*) for drawing the elements in their displaced configuration. Using the eigenvector solutions, animations of the shapes could quite easily be implemented. A simple user dialog was designed and coupled to the eigenvalue solver. At the time of implementation, the MFC (*Microsoft Foundation Classes*) framework was used for user interface design. A screenshot of the interface, with an animated free-vibration mode in the background is seen in Figure 4.3. Note that this is *not* the NREL reference turbine used earlier.

Snapshots from the animations are shown in Figure 4.4, displaying bending modes of a cantilever beam model.

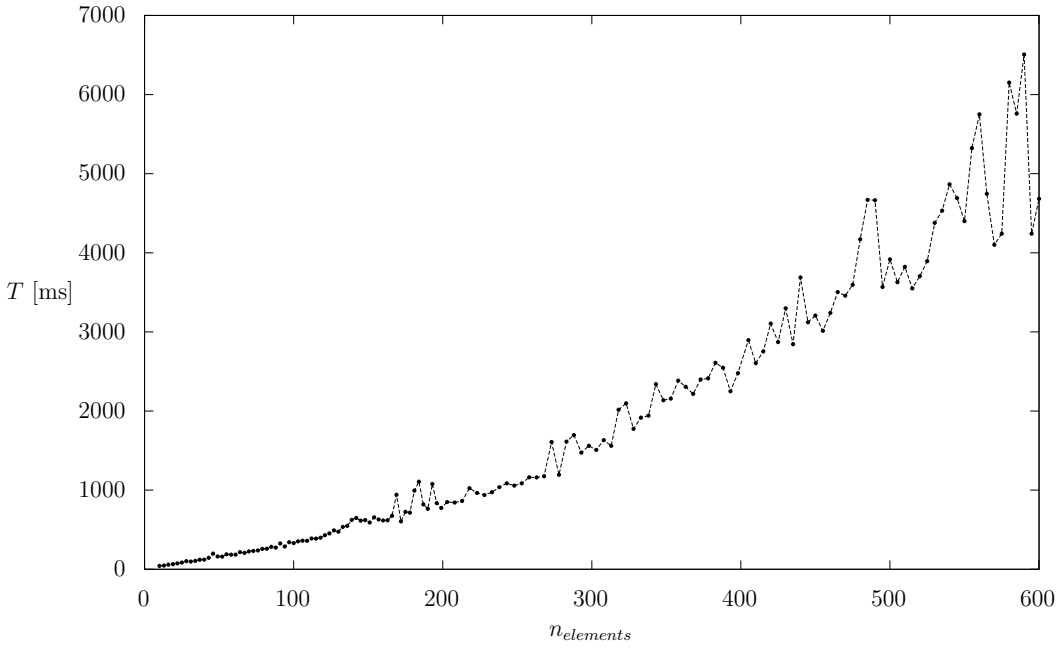


Figure 4.2: Solution time of 5 first modes for a cantilever beam compared to number of elements used. Parameters used: $\epsilon_\lambda = 10^{-7}$, $\Delta\mu_0 = 0.01\,\mathrm{s}^{-2}$

4.5 Discussion

The inverse iteration method will converge very fast if the shifted point is close to an eigenvalue. However, a "brute-force" search, which was suggested and implemented here will in general not be competitive to more sofisticated eigenmethods with respect to performance. If a large range is to be searched with a fine resolution, the solution time for a medium-sized problem will be on the scale of seconds. Combination with some other algorithm, e.g. the Sturm sequence, might improve the efficiency.

On the contrary, if only a few (e.g. 10) of the lowest modes are of interest, using a medium search range, the solution time will be quite fast. By allowing a simple implementation strategy where existing solver functionality can be used in-place, the need for writing an efficient equation solver code is eliminated. The algorithm may be a possible choice in such cases, where an easy and relatively uncomplex implementation is desirable.

As shown here, the results for the eigenvalues of the model problem are correct

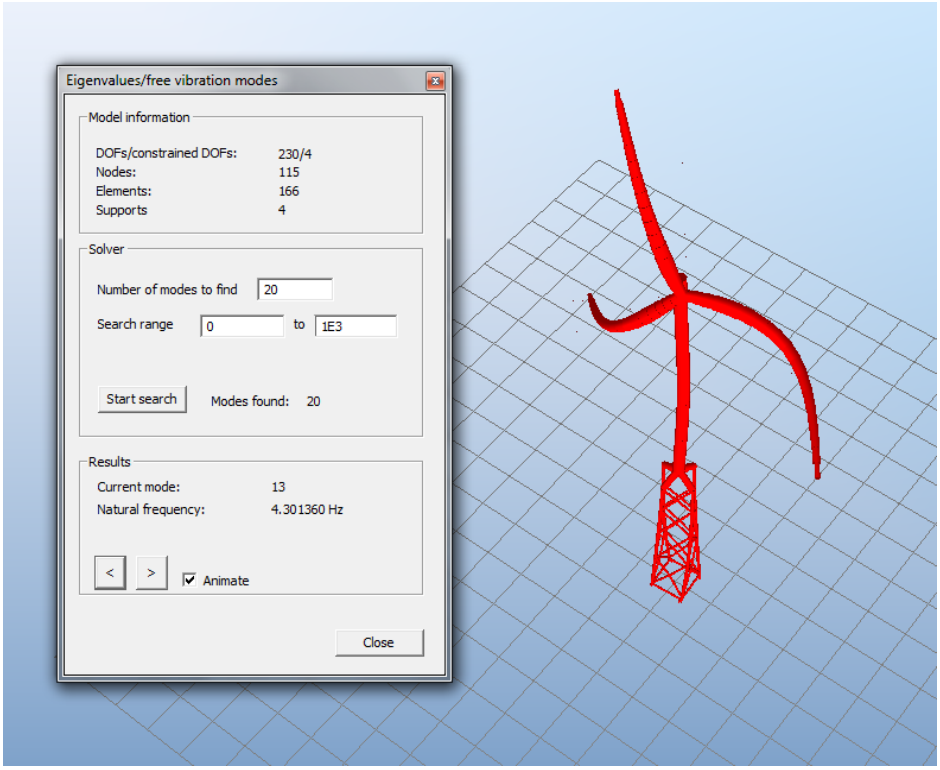
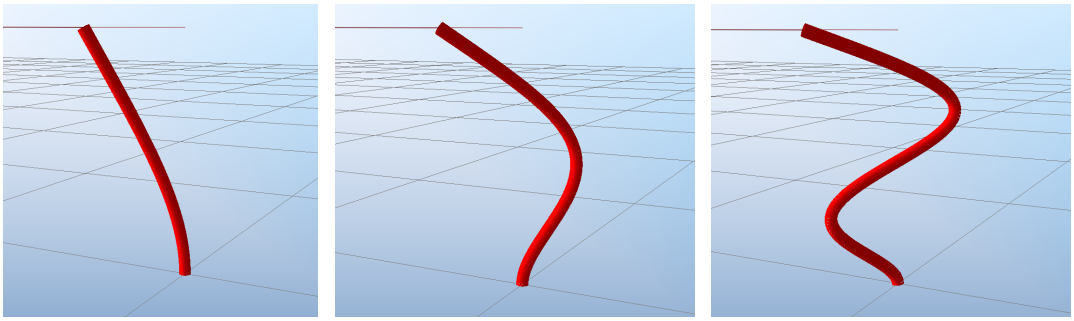


Figure 4.3: User dialog for the eigenvalue solver, showing the the shape of mode 13 of a wind turbine model.



(a) First mode (b) Second mode (c) Third mode

Figure 4.4: Bending modes of cantilever beam

when compared to exact values. By using different settings for the solver, one can effectively guarantee that all solutions within some range are found. The implementation is therefore reliable.

Having an eigenvalue solver implemented in close connection to the existing frame-

work gives possibilities for a more interactive and responsive interface to the user, as well as a potential for customizing the solver to new needs. On the contrary, if an external solver library were to be used these features might be harder to implement.

5 Conclusion

As stated in the introduction, the main focus in this paper has been to become familiar with efficient numerical methods for solving the eigenvalue problem in structural dynamics in an efficient and reliable manner. A modification to the inverse iteration method was developed in order to solve for eigenvalues in a range without any prior knowledge about their positions on the eigenvalue axis.

The implemented algorithm was developed in a more or less ad hoc manner, but provided to be well-suited to problems where either high efficiency was not imperative or a somewhat uncomplicated algorithm was preferable. Further investigation on the validity of the results and the behaviour of the solver for different parameter settings was done.

An even more interesting part has been the study of the arguably state-of-the-art features of the finite element framework that was used. In describing the concepts of coordinate-free elements and tensor-based formulations, the clarity and "cleanness" these features provided in an implementation context was experienced.

Appendices

Appendix A

Dual vector space

The mathematical concept of dual spaces may be useful for defining tensor relations needed in order to develop expressions for e.g. the stiffness tensors. For completeness, and due to its use in the deduction of element stiffness (Section 3.2.1), the concept is appended here.

In mathematics, any vector space , V, has a corresponding dual vector space consisting of *all* linear functionals on V. Stated mathematically, for any vector space V over a field F, the dual space, V^* is all linear maps

$$\varphi : V \to F$$

Given a basis $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ in V, a basis in the dual space V^*, $\{\mathbf{e}^1, \ldots, \mathbf{e}^n\}$ is defined such that

$$\boldsymbol{e}^i \cdot \boldsymbol{e}_j = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } \neq j \end{cases} \tag{A.1}$$

Example

By taking the basis in 2D space ($V = \mathbf{R}^2$), as $\{\boldsymbol{e}_1, \boldsymbol{e}_2\} = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$, a basis of its dual space $\mathbf{R}^{2*}$ is $\{\boldsymbol{e}^1, \boldsymbol{e}^2\} = \left\{ \begin{bmatrix} 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \end{bmatrix} \right\}$.

In 3D space, the dual basis can be calculated from the following formulas:

$$\boldsymbol{e}_1 = \frac{(\boldsymbol{e}_2 \times \boldsymbol{e}_3)}{(\boldsymbol{e}_2 \times \boldsymbol{e}_3) \cdot \boldsymbol{e}_1}, \quad \boldsymbol{e}_2 = \frac{(\boldsymbol{e}_3 \times \boldsymbol{e}_1)}{(\boldsymbol{e}_3 \times \boldsymbol{e}_1) \cdot \boldsymbol{e}_2}, \quad \boldsymbol{e}_3 = \frac{(\boldsymbol{e}_1 \times \boldsymbol{e}_2)}{(\boldsymbol{e}_1 \times \boldsymbol{e}_2) \cdot \boldsymbol{e}_3} \tag{A.2}$$

Appendix B

C++ class declarations

```
#include "GaussElimInteraction.h"

/**
 * An EigenvalueInteraction is no more than a basic Interaction.
 * Additionally, a mass tensor is defined for the interaction,
 * which is necessary for solving the eigenvalue problem.
 *
 * @author Per Ivar Bruheim
 * @date 15.10.2011
 *
 */
class EigenvalueInteraction : public GaussElimInteraction
{

public:
  EigenvalueInteraction();
  EigenvalueInteraction(DOF* who, GTensor &kij, GTensor &mij,
      Interaction *next);

  /// The mass tensor for this interaction
  GTensor mMassWhat;
  /// A copy of the original mass tensor, for use when resetting
  GTensor mOrigWhat;
};
```

:

```
#include "GaussElimDOF.h"
#include <map>

/**
 * An EigenvalueDOF is essentially a GaussElimDOF with mass
     properties
```

```
 * included and added functionality, i.e for shifting the stiffness
 * and other useful functions for the EigenvalueSolver.
 *
 * @author Per Ivar Bruheim
 */
class EigenvalueDOF : public GaussElimDOF
{
public:
  EigenvalueDOF();
  Interaction* MakeInteraction();

  // Overrides of DOF::
  virtual void Connect(DOF *dofj, GTensor &kij);
  virtual void Connect(DOF *dofj, GTensor &kij, GTensor &mij);
  virtual void SelfConnect(GSymmTensor &kij, GSymmTensor &mij);
  virtual void HandleConstraint();

  // Added functions
  Scalar GetOmega(int modeNumber) const;
  void SetDisplacement(GVector const& v);
  int SetDisplacementByMode(int modeNumber, Scalar scale = 1.0);
  void ResetKM();
  void ShiftStiffnes(Scalar mu);
  void StoreMode(int modeNumber, Scalar omega);
  GVector& GetInertialLoadFromDisplacement(GVector &v) const;

  /// This DOF's mass tensor
  GSymmTensor mSelfMass;
  GSymmTensor mOrigSelfInteraction;
  GVector mPrevDisplacement;

private:

  struct EigenPair { GVector v; Scalar omega; };
  /// A vector that is filled up with displacements resulting from
  // calculating each mode. The modes are ordered from lowest to
  // highes eigenfrequency.
  std::map<int, EigenPair> mModes;
  /// The mode number that is currently set
  int mCurrentMode;

};
```

:

```
#include "GaussElimSolver.h"
#include "EigenvalueDOF.h"

class EigenvalueSolver : public GaussElimSolver
```

```
5 {

7 public:
    EigenvalueSolver(int numModesToFind, Scalar rangeMin = 0.0, Scalar
        rangeMax = 1E6, Scalar convergenceTolerance = 1E-6);
9   ~EigenvalueSolver();

11  // Overrides
    virtual EigenvalueDOF* MakeDOF();
13  virtual void SetDOFs(DOF* dofs);
    virtual void SolveSystem();
15
    // Non-overrides
17  Scalar GetRayleighQuotient() const;
    int GetNumModesFound() const;
19  void SetConvergenceTolerance(Scalar eps);
    /// Function used for benchmarking and studying the effect of the
        initial step
21  void SetInitialShift(Scalar mu) { mMu = mu; }

23 private:
    Scalar mTolerance;
25  int mNumModesToFind;
    int mNumModesFound;
27  Scalar mShift;
    Scalar mRangeMax;
29  Scalar mRangeMin;
    Scalar mMu;
31
    Scalar SolveNext(Scalar prevLambda);
33  void ResetKM() const;
    void ShiftStiffness(Scalar lambda);
35  void StoreMode(int modeNumber, Scalar omega) const;

37  int GetDOFCount() const;

39 };
```

:

References

[1] K. J. Bathe. *Finite Element Procedures*. Prentice Hall, 1996.

[2] Pål G. Bergan, Per Kr. Larsen, and Egil Mollestad. *Svingning av konstruksjoner*. Tapir, second edition, 1985.

[3] Anil K. Chopra. *Dynamics of Structures: Theory and Applications to Earthquake Engineering*. Pearson, third edition, 2006.

[4] Robert D. Cook, David S. Malkus, Michael E. Plesha, and Robert J. Witt. *Concepts and Applications of Finite Element Analysis*. Wiley, fourth edition, 2001.

[5] Jonkman J., Butterfield S., Musial W., and Scott G. Definition of a 5-mw reference wind turbine for offshore system development, 2009.

[6] Erwin Kreyszig. *Advanced Engineering Mathematics*. Wiley, ninth edition, 2006.

[7] Ivar Langen and Ragnar Sigbjörnsson. *Dynamisk analyse av konstruksjoner*. Tapir, 1979.

[8] G. R. Miller and M. D. Rucki. A program architecture for interactive nonlinear dynamic analysis of structures. 1993.

[9] G.R. Miller. An object-oriented approach to structural analysis and design. *Computers & Structures*, 40(1):75 – 82, 1991.

[10] G.R. Miller, P. Arduino, J. Jang, and C. Choi. Localized tensor-based solvers for interactive finite element applications using c++ and java. *Computers and Structures*, 81(7):423–437, 2003.

[11] M.D. Rucki and G.R. Miller. An algorithmic framework for flexible finite element-based structural modeling. *Computer Methods in Applied Mechanics and Engineering*, 136(3-4):363–384, 1996.

[12] Paul E. Thomassen. *Development of a Direct Manipulation, Two-dimensional Finite Element Analysis Tool.* PhD thesis, University of Washington, 1995.

NTNU

Norwegian University of
Science and Technology