Master's thesis

Trondheim, 2012

Per Ivar Bruheim

Master's thesis

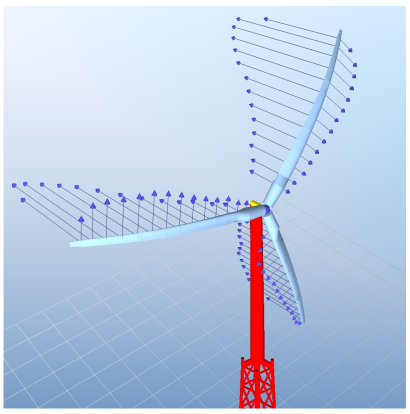
NTNU
Norwegian University of Science and Technology
Faculty of Engineering Science and Technology
Department of Civil and Transport Engineering

NTNU – Trondheim
Norwegian University of
Science and Technology

Per Ivar Bruheim

Development and validation of a finite element software facilitating large-displacement aeroelastic analysis of wind turbines

Trondheim, June 2012



NTNU – Trondheim
Norwegian University of
Science and Technology

Acknowledgements

The work resulting in this thesis was supervised by Dr. Paul E. Thomassen (Department of Civil Engineering, NTNU). I would like to thank him for involving me in the ASHES project and the interesting field of wind energy technology. As a result of his determination and vision, the further development of the ASHES software will even be my full-time employment after graduating.

The software developed during this work was based on the finite element framework developed by Dr. Jae Won Jang and Professor Gregory M. Miller at the University of Washington. Their sharing of the source code has been greatly appreciated.

The last year's work in the ASHES team has been a pleasant experience, and I would like to thank also Loup Suja for his enthusiasm during this period.

I also want to thank Dr. Bjørn Haugen (Department of Engineering Design and Materials, NTNU) for providing me with his doctoral dissertation and his valuable comments.

Lastly, I want to thank my family for their support.

"In the middle of difficulty lies opportunity."
–Albert Einstein

Per Ivar Bruheim — Trondheim, June 2012

Abstract

(English)
This thesis establishes necessary theory for the geometrically nonlinear dynamic analysis of spatial beam structures by the corotational formulation of finite elements. By extending an existing framework that is capable of finite element modeling as well as computing aerodynamic forces, a code is developed that facilitates the aeroelastic analysis of wind turbines, where the effects of large rotations and deflections of the blades are captured. Verification of the code is carried out first by basic tests, and then by comparing the computed response of a wind turbine model with other codes for wind turbine design. Results are shown to be in good agreement with other codes.

(Norsk)
Denne masteroppgaven etablerer nødvendig teori for geometrisk ikke-lineær dynamisk analyse av romlige bjelkekonstruksjoner ved å benytte en korotert formulering i elementmetoden. Med utgangspunkt i et eksisterende rammeverk som er i stand til å modellere med elementmetoden og å beregne aerodynamiske krefter, utvikles det et program som gjør det mulig å utføre aeroelastisk analyse av vindturbiner, hvor effekter som skyldes store rotasjoner og utbøyninger er inkludert. Verifisering av koden blir gjennomført først ved hjelp av enkle tester, så ved å sammenligne den beregnede responsen av en vindturbinmodell med andre programmer for design av vindturbiner. Resultatene er funnet å være i god overensstemmelse.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Principal objectives

This thesis presents the background theory, development and validation of a tool for analysis of structures undergoing large displacements, and its application to the aeroelastic simulation of wind turbines.

The principal objectives of this thesis are three:

1. Devise and investigate a suitable method for geometrically nonlinear analysis of beam structures for application to wind turbine rotors and blades.

In a finite element context, there exist several approaches to the inclusion of non-linear geometry, with the corotational formulation being one of them. In this work, the corotational formulation is chosen as a fitting candidate due to attractive features such as re-usability with other element types.

2. Develop and validate a software built on an existing framework that facilitates the large-displacement analysis of wind turbines.

The demand for codes that correctly and effectively predicts the response of wind turbines subjected to large displacements is clearly present. Today's wind turbine designs trend towards increased dimensions and scales, as well as lighter materials. Longer blades increase the energy output, but at the same time they become more slender and flexible. Thus, they are possibly subjected to moderate deflections.

An existing finite element framework will be extended to accommodate the non-linear dynamic analysis of a wind turbine rotor, and numerical results are further compared to similar results from other codes.

3. Compare the advantages and disadvantages of different methods and algorithms for computing the dynamic response of wind turbines subjected to arbitrary loads.

The behaviour of a typical wind turbine is determined by complex interactions between several components, which often are of a nonlinear nature. As the interest of the analyst varies greatly—from transient, time-local effects as e.g. controller behaviour; to the performance on a large time-scale, such as predicted power production and expected damage due to fatigue—there is a diverse set of appropriate methods. Existing software and tools for wind turbine design illustrate this, utilizing methods ranging from classical frequency-domain methods to time-domain methods based on finite elements or multibody systems, as well as using various reduction and simplification techniques. A comparison will be given, with special focus on determining the difference with the implementation of the previous objective.

1.2 Overview of literature

The theory of corotational finite elements has been studied by several authors. Simo and Vu-Quoc (1988) describe an isoparametric approach[42]. Crisfield (1990) describes corotational 3D beam elements including a consistent tangent stiffness [14]. Iura (1994) compares results using finite strain measures with corotated elements [22]. Crisfield and Shi (1994)[16] establish a scheme in a dynamic context, which is also considered by Crisfield et al. (1997)[15]. Haugen (1994) thoroughly presents different corotated formulations, while Haugen and Felippa (2005)[20] develop a unified formulation. Shabana et al. (2007)[40] pursue the integration of large deformation finite elements and mulitbody systems. References on the topic of rotations are e.g. Argyris (1982)[2].

In the context of wind turbine design, Quarton (1998)[35] gives an historical overview over design techniques. Rasmussen et al. (2003) describe the present status of aeroelastic modeling, including large blade deflections[36]. Kallesøe (2011) investigates the effect of large blade deflections on stability[25]. Vollan and Komzsik (2012) give an overview of computational techniques for rotor dynamics, but do not mention the corotational formulation[47]. Different methods for applying aeroelastic loads to finite elements are discussed by Knill (2005)[26]. For references on wind turbine design and wind energy technology, see e.g. [7, 17, 30].

For references on finite elements and dynamics, see e.g. [4,9–11]. The finite element framework that has been used as a basis for the present work has been described by Miller (1991)[32], Miller and Rucki (1996 and 1998)[37, 38] as well as Jang (2007)[23]. Later extensions related to wind turbines are described by Thomassen et al. (2011)[43].

1.3 Outline of thesis

Chapter 2 presents theory and formulas for the treatment of rotations in space. Chapter 3 discusses geometric nonlinearities and formulations, and details the commonly used Newmark-β incremental scheme. Chapter 4 presents the corotational formulation of finite elements and suggests a modified tangent stiffness. In Chapter 5, the background of the framework is outlined, and implementation specifics are presented. Chapter 6 is devoted to the benchmarking and validation of the implementation of corotated elements. Chapter 7 discusses different techniques for wind turbine analysis and their differences, and further presents simulation results of a full wind turbine model demonstrating the validity of the implementation. Finally, Chapter 8 gives conclusions and recommendations for future work.

2 Rotations in space

2.1 Introduction

The correct treatment of rotations in space is essential to the analysis of large deformations. An inherent property of finite (i.e. large) rotations in space is that they are not additive, in contrast to displacements, which behave like vectors. Hence, the motivation for this chapter is to establish theory and formulas necessary for dealing with rotations in the formulation and implementation of corotated elements in in later parts of this thesis.

As several ways of describing the rotation of a body in space exist, with each of them having its advantages and disadvantages, a brief comparison will be given in the following sections. The quaternion representation will be given special care due to its attractive features. It is pointed out that several other representations exist, but they will not be considered here. More detailed theory of quaternions can be found in [13, 41] and [46, Ch. 5] while rotation tensors are treated by Haugen and Felippa (2005)[20].

2.2 Rotation representations

2.2.1 Rotation vector and axis-angle

The rotation vector or axis-angle representations are perhaps the most intuitive ways of describing an arbitrary rotation in space. A rotation of a body in space can always be stated as a rotation about a unit direction vector $\mathbf{n}$ by an angle θ. This pair, consisting of a unit vector and a scalar, is referred to as the axis-angle representation,

$$\{\mathbf{n}, \theta\} \; . \tag{2.1}$$

Further, by multiplying the unit direction vector by the scalar-valued angle, the information can be contained in the form of a single rotation vector:

$$\boldsymbol{\theta} = \theta \mathbf{n} = \begin{Bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{Bmatrix} \quad \text{with} \quad \theta = |\boldsymbol{\theta}| \ . \tag{2.2}$$

This quantity can in an implementation be stored conveniently as a vector type. Still, it should be stressed that this vector does not adhere to the usual rules of vector algebra, as further discussed in Section 2.5.

2.2.2 Rotation tensor

A rotation tensor (i.e. a 3x3 matrix) is another possible representation of a rotation in space.

A vector $\mathbf{v}$ is rotated by the rotation described by the tensor $\mathbf{R}$ to $\mathbf{v}'$ by a pre-multiplication:

$$\mathbf{v}' = \mathbf{R}\mathbf{v} \tag{2.3}$$

A rotation tensor is orthogonal—that is, $\mathbf{R}^{-1} = \mathbf{R}^{\mathrm{T}}$. Its content is commonly computed by the use of direction cosines.

2.2.3 Euler angles

The Euler angle representation separates the total rotation into three: a pitch, roll and yaw angle. No universal rule exists for the order of rotations, which is not arbitrary, and they are also susceptible to gimbal lock when the pitch approaches $\pm 90°$. Hence, they are problematic when arbitrary rotations and orientations are involved, and are not suited for representing finite rotations.

The next section is devoted to the quaternion representation, including quaternion algebra and its application to rotations.

2.3 Quaternions

Quaternions, although appearing somewhat abstract, are well-suited for representing rotations. The concept of quaternions was discovered by Sir William Rowan Hamilton in 1843, in his attempt to find the 3D equivalent of complex numbers[41].

Although quaternions mathematically are based on complex numbers, their operations can be utilized without explicitly working with complex numbers. In this section, the operations that are useful for rotations will be presented. Each operation is followed by a short C++ code section that demonstrates how the operation might be implemented.

2.3.1 Quaternion algebra

A quaternion is basically a 4-tuple, which can be stated in multiple equivalent ways

$$\begin{aligned}\mathbf{q} &= \begin{bmatrix} x & y & z & w \end{bmatrix} \\ &= x\mathbf{i} + y\mathbf{j} + z\mathbf{k} + w \ ,\end{aligned}$$

although, since a quaternion in essence has a vector part and a scalar part, writing it as

$$\mathbf{q} = \begin{bmatrix} \mathbf{v} & w \end{bmatrix} \tag{2.4}$$

clearly shows the vector–scalar nature.

The basic quaternion operations and properties are given in the following.

Addition

$$\mathbf{q} + \mathbf{q}' = \begin{bmatrix} \mathbf{v} & w \end{bmatrix} + \begin{bmatrix} \mathbf{v}' & w' \end{bmatrix} = \begin{bmatrix} \mathbf{v} + \mathbf{v}' & w + w' \end{bmatrix}$$

```
Quat q3 = q1.Plus(q2);
q2.PlusEquals(q1);
```

Multiplication

The product of two quaternions is given by

$$\mathbf{q}\mathbf{q}' = \begin{bmatrix} \mathbf{v} & w \end{bmatrix} \begin{bmatrix} \mathbf{v}' & w' \end{bmatrix} = \begin{bmatrix} \mathbf{v} \times \mathbf{v}' + w\mathbf{v}' + w'\mathbf{v} & ww' - \mathbf{v} \cdot \mathbf{v}' \end{bmatrix} \ . \tag{2.5}$$

```
Quat q3 = q1.Mult(q2);
q2.MultEquals(q1);
```

In general, $\mathbf{q}\mathbf{q}' \neq \mathbf{q}'\mathbf{q}$.

The product of a quaternion $\mathbf{q}$ and a vector $\mathbf{v}$, yields a new quaternion, and is calculated in the same manner as (2.5) with the vector part of $\mathbf{q}$ set equal to $\mathbf{v}$ and the scalar part to zero, i.e.

$$\mathbf{q}\mathbf{v} = \mathbf{q}\begin{bmatrix}\mathbf{v} & 0\end{bmatrix} \tag{2.6}$$

Conjugation

$$\mathbf{q}^* = \begin{bmatrix}\mathbf{v} & w\end{bmatrix}^* = \begin{bmatrix}-\mathbf{v} & w\end{bmatrix}$$

```
Quat q2 = q.Conjugate();
```

Norm

The quaternion norm is given by

$$||\mathbf{q}|| = \mathbf{q}\mathbf{q}^* = \mathbf{v}\cdot\mathbf{v} + w^2 = x^2 + y^2 + z^2 + w^2 \ .$$

Note that in the literature, both the definition above and the square-root of it can be found for the quaternion norm. This is important to take notice of, because when normalizing a quaternion, one should divide the components by the square-root expression. Still, when checking if a quaternion is normalized, the fact that the norm is unity is sufficient, and avoids the square-root operation. This paper will use the definitions in e.g. [41], with the *magnitude* being the square-root of the norm, so that

$$|\mathbf{q}| = \sqrt{||\mathbf{q}||}$$

```
Scalar norm = q.Norm();
Scalar magnitude = q.Magnitude();
```

Inverse

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{||\mathbf{q}||}$$

```
Quat q2 = q.Inverse();
```

Normalization

$$\hat{\mathbf{q}} = \frac{\mathbf{q}}{|\mathbf{q}|} \quad \Rightarrow \quad |\mathbf{q}| = ||\mathbf{q}|| = 1 \tag{2.7}$$

```
q.Normalize();
```

2.3.1.1 Rotating a vector by a quaternion

Given the normalized quaternion $\mathbf{q}$, the vector resulting from applying the rotation of $\mathbf{q}$ to the vector $\mathbf{v}_1$ is given by the vector part of the product

$$\begin{bmatrix}\mathbf{v}_2 & w\end{bmatrix} = \mathbf{q}\mathbf{v}_1\mathbf{q}^{-1}\,. \tag{2.8}$$

This operation can conveniently be implemented as

```
Vector v2 = q.Rotate(v1);
```

2.4 Conversions

The conversion between different rotation representations is necessary in many of the procedures in later chapters. Many conversion rules exist in the literature for converting between the axis-angle, tensor and quaternion representations of rotations. The conversion rules given here are the ones used in the implementation of Chapter 5, and have seemed to give correct results and acceptable numerical performance

2.4.1 Axis-angle to quaternion

Given the arbitrary axis-angle rotation $\boldsymbol{\theta} = \begin{bmatrix}\theta_x & \theta_y & \theta_z\end{bmatrix}^{\mathrm{T}}$, where the rotation angle is $\theta = |\boldsymbol{\theta}|$, the quaternion equivalent can be calculated by

$$\mathbf{v} = \sin\left(\frac{\theta}{2}\right) \cdot \frac{\mathbf{n}}{\theta} \quad \text{and} \quad w = \cos\left(\frac{\theta}{2}\right) \tag{2.9}$$

2.4.2 Axis-angle to rotation tensor

The Rodrigues formula gives a rotation tensor representing the rotation of an axis-angle. Given the arbitrary axis-angle rotation $\boldsymbol{\theta} = \begin{bmatrix}\theta_x & \theta_y & \theta_z\end{bmatrix}^{\mathrm{T}}$, where the rotation angle is $\theta = |\boldsymbol{\theta}|$ and the unit rotation axis vector is $\mathbf{n} = \begin{bmatrix}n_1 & n_2 & n_3\end{bmatrix}^{\mathrm{T}} = \dfrac{\boldsymbol{\theta}}{\theta}$, the rotation tensor is given by

$$\mathbf{R} = \mathbf{I} + \mathbf{N}\sin\theta + \mathbf{N}^2\left(1 - \cos\theta\right) \tag{2.10}$$

$$\tag{2.11}$$

where $\mathbf{N}$ is the skew-symmetric tensor defined by

$$\mathbf{N} = \operatorname{Spin}\left(\mathbf{n}\right) = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

and

$$\mathbf{N}^2 = \mathbf{n}\mathbf{n}^{\mathrm{T}} - \mathbf{I} = \begin{bmatrix} n_1n_1 - 1 & n_1n_2 & n_1n_3 \\ n_2n_1 & n_2n_2 - 1 & n_2n_3 \\ n_3n_1 & n_3n_2 & n_3n_3 - 1 \end{bmatrix}.$$

By combining these equations, an explicit form can be obtained.

2.4.3 Rotation tensor to rotation vector

For a small absolute rotation, the conversion from a rotation tensor $\mathbf{R}$ to the equivalent rotation vector $\boldsymbol{\theta}$ is given by the following algorithm.

First, compute

$$\begin{aligned} d_1 &= \frac{1}{2}\left(R_{32} - R_{23}\right)\,, \\ d_2 &= \frac{1}{2}\left(R_{13} - R_{31}\right)\,, \\ d_3 &= \frac{1}{2}\left(R_{21} - R_{12}\right)\,, \end{aligned} \tag{2.12a}$$

Where index ij corresponds to column i, row j. The angle is then given by

$$\sin\theta = \sqrt{d_1^2 + d_2^2 + d_3^2}\,. \tag{2.12b}$$

The rotation vector can finally be obtained as

$$\boldsymbol{\theta} = \frac{\theta}{\sin\theta}\begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix}^{\mathrm{T}} . \tag{2.12c}$$

In a computer implementation, care must be taken when the angle approaches 0, which causes $\theta/\sin\theta \longrightarrow 0/0$. For small angles (e.g. $\theta < 10^{-8}$), the fraction $\theta/\sin\theta$ should therefore be set to unity [18, Eq. 2.3.20].

2.5 Accumulation of rotations

The accumulation of multiple finite rotations into a total rotation is essential when e.g. rotation variables are incremented in a large-deformation analysis, and will be treated in the following. With the Euler angle or rotation vector representations, no general method exists, so a conversion to either a tensor or quaternion is necessary.

2.5.1 Using tensors

If $\mathbf{R}_1$ and $\mathbf{R}_2$ are two arbitrary rotation tensors, and $\mathbf{v}$ is a vector in space, then the vector

$$\mathbf{v}' = \mathbf{R}_2\mathbf{R}_1\mathbf{v}$$

is the vector resulting from first applying the $\mathbf{R}_1$ and then the $\mathbf{R}_2$ rotation to $\mathbf{v}$. The compound rotation tensor can hence be stated as

$$\mathbf{R} = \mathbf{R}_2\mathbf{R}_1 \tag{2.13}$$

so that

$$\mathbf{v}' = \mathbf{R}\mathbf{v}$$

2.5.2 Using quaternions

If the quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ represent two arbitrary rotations in space and $\mathbf{v}$ is a vector, the new vector resulting from first rotating $\mathbf{v}$ by the rotation $\mathbf{q}_1$ and then by $\mathbf{q}_2$ is given by (2.8):

$$\mathbf{v}' = \mathbf{q}_2\left(\mathbf{q}_1\mathbf{v}\mathbf{q}_1^{-1}\right)\mathbf{q}_2^{-1} . \tag{2.14}$$

Hence, the total rotation is represented by the quaternion $\mathbf{q} = \mathbf{q}_2\mathbf{q}_1$. Rotating a vector by $\mathbf{q}$ is equivalent to first applying the rotation of $\mathbf{q}_1$ and then the rotation of $\mathbf{q}_2$ to the vector. Finally, the total rotation can be applied to a vector by (2.8).

For correct results, the quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ *must be normalized* by (2.7) before calculating the product, by dividing each component by the quaternion magnitude.

2.6 Quaternion vs. tensor

2.6.1 Storage and performance

A tensor requires the storage of 9 numbers, the quaternion 4, while 3 numbers are sufficient for a rotation vector or the Euler angles. As previously mentioned, only the tensor and quaternion is suitable in an implementation where the accumulation of rotations is required. By counting the number of computer operations needed for the accumulation of two rotations and the application of the rotation to a vector, respectively, a crude estimate of their comparative efficiency is obtained. This comparison is given in Table 2.1. It is observed that, while quaternions require

Table 2.1: Comparison of number of operations.

Representation	Accumulation of rotations	Rotation of vector
Tensor	45	15
Quaternion	28	30

fewer operations than tensors for the accumulation of rotations, more are required when the rotation is applied to a vector. Thus, the most efficient candidate for representing rotations is not obvious.

2.6.2 Round-off and normalization

In a computer implementation, round-off errors will always be a concern when rotations are accumulated through either tensor or quaternion products. Due to these round-offs, a rotation tensor might loose its orthogonality property, leading to it representing an invalid rotation, further producing erroneous results. Thus, some procedure for orthonormalizing the tensor should be utilized, e.g. by the Gram-Schmidt orthogonalization algorithm (see [44, p. 56])

A quaternion does not suffer from round-offs, as it is not subjected to a constraint such as requiring orthogonality. The only requirement for a valid quaternion representation is that it is of unit magnitude (i.e. normalized).

2.7 Concluding remarks

- Finite rotations can not be treated in a vectorial manner, but rather by the use of tensors or quaternions.
- Euler angles and rotation vectors are not suited for accumulating arbitrary finite rotations.
- Finite rotations are accumulated through products of *either* tensors or quaternions. A rotation vector can be converted to a tensor or quaternion by explicit formulas.
- Computer round-off may lead to the loss of orthogonality, demanding procedures for re-orthonormalizing the rotation tensor.
- No general conclusion can be made with regards to which is the “best” choice of tensors and quaternions.

3 The nonlinear equation of motion

3.1 Introduction

In later chapters, the theory of corotated beam elements and the implementation in an existing finite element framework is presented. For the successful integration of the corotated procedures with the existing solver functionality—namely the Newmark-β scheme with Newton-Raphson iterations—a treatment of necessary theory is given in this chapter. Additionally, the inclusion of this theory makes referring to specific equations later simpler.

A brief comparison of methods for geometrically nonlinear analysis is here given. Due to the importance in the presence of aerodynamic loads, and hence to the analysis of wind turbine blades in Chapter 7, the load stiffness is also defined. More theory can be found in the finite element literature, e.g. [4, 9, 11].

The final section presents the conventional material and geometric stiffness, as well as the consistent-mass properties of the beam element, formulated in terms of tensors—which is necessary for use in the finite element framework used in this work.

3.2 Geometric nonlinearities

In a linear analysis, the equations of motion remain unchanged as the structure deforms. At some point, however, the magnitude of the deformation will affect the directions of applied forces and the element properties. Such effects are denoted geometric nonlinearities.

3.2.1 Non-conservative forces and load stiffness

The work done by a *conservative* force on a body moving through space is independent of the path taken. On the contrary, non-conservative forces does different work when different paths are taken. A *follower load* is a typical non-conservative load, with its direction following the rotation of the body that it acts upon. Aerodynamic forces, for instance, behave like follower loads.

Stated mathematically, an external conservative force $\mathbf{f}^{\text{ext}}$ acting on a body with displacement $\mathbf{d}$ has the property

$$\frac{\partial \mathbf{f}^{\text{ext}}}{\partial \mathbf{d}} = 0 \qquad \text{when } \mathbf{f}^{\text{ext}} \text{ is conservative} \tag{3.15a}$$

while for a non-conservative force, this gradient is in general non-zero. Hence, this gradient will give rise to a term in the linearized equation of motion, namely a *load stiffness*:

$$\mathbf{K}^{\text{ext}} = \frac{\partial \mathbf{f}^{\text{ext}}}{\partial \mathbf{d}} \neq 0 \qquad \text{when } \mathbf{f}^{\text{ext}} \text{ is non-conservative}\,. \tag{3.15b}$$

Using a variational notation, (3.15b) can equivalently be written as

$$\delta \mathbf{f}^{\text{ext}} = \mathbf{K}^{\text{ext}} \delta \mathbf{d} \tag{3.15c}$$

3.2.2 A note on the updated and total Lagrangian formulations

In the updated or total Lagrangian formulation, a finite strain measure for the element is usually employed. This strain measure gives a valid internal force state of the element for arbitrarily large rigid body displacements, and for arbitrarily large element strains. Although these two are equivalent, the equations of motion in the updated Lagrangian method is expressed in terms of the *current* configuration (i.e. using Eulerian coordinates), while the total Lagrangian method uses the *undeformed* configuration as the reference[39, p. 118]. The element in either the updated or total Lagrangian method will have to be developed and implemented with basis in such strain measures.

3.3 Solution of the non-linear equation of motion

3.3.1 The dynamic nonlinear dynamic equation of motion

The dynamic equation of motion at time $n+1$ is given as

$$\mathbf{M}\left\{\ddot{\mathbf{d}}\right\}_{n+1} + \mathbf{C}\left\{\dot{\mathbf{d}}\right\}_{n+1} + \left\{\mathbf{f}^{\text{int}}\right\}_{n+1} - \left\{\mathbf{f}^{\text{ext}}\right\}_{n+1} = \{\mathbf{r}\}_{n+1} = \mathbf{0}\,. \tag{3.16}$$

where vector $\{\mathbf{r}\}_{n+1}$ is the residual, or out-of-balance force, which is $\mathbf{0}$ when equilibrium is satisfied. By subtracting from (3.16) the equivalent equation of motion at time n, an incremental form is obtained:

$$\mathbf{M}\left\{\Delta\ddot{\mathbf{d}}\right\}_{n} + \mathbf{C}\left\{\Delta\dot{\mathbf{d}}\right\}_{n} + \left\{\Delta\mathbf{f}^{\text{int}}\right\}_{n} - \left\{\Delta\mathbf{f}^{\text{ext}}\right\}_{n} = \{\Delta\mathbf{r}\}_{n} = \mathbf{0}\,, \tag{3.17}$$

where $\left\{\Delta\dot{\mathbf{d}}\right\}_n = \left\{\dot{\mathbf{d}}\right\}_{n+1} - \left\{\dot{\mathbf{d}}\right\}_n$ etc. By a first-order linearization of the internal and external forces, we get

$$\left\{\Delta\mathbf{f}^{\text{int}}\right\}_n = \left[\frac{\partial\mathbf{f}^{\text{int}}}{\partial\mathbf{d}}\right]_n \{\Delta\mathbf{d}\}_n = \left[\mathbf{K}^{\text{int}}\right]_n \{\Delta\mathbf{d}\}_n \tag{3.18a}$$

$$\left\{\Delta\mathbf{f}^{\text{ext}}\right\}_n = \left[\frac{\partial\mathbf{f}^{\text{ext}}}{\partial\mathbf{d}}\right]_n \{\Delta\mathbf{d}\}_n = \left[\mathbf{K}^{\text{ext}}\right]_n \{\Delta\mathbf{d}\}_n\,, \tag{3.18b}$$

where $\left[\mathbf{K}^{\text{int}}\right]$ is called the consistent tangent stiffness and $\left[\mathbf{K}^{\text{ext}}\right]$ is the load stiffness. Inserting these linearized increments into (3.17) gives an equation where only the accelerations and velocities at $n+1$ are unknown remains:

$$\mathbf{M}\left\{\Delta\ddot{\mathbf{d}}\right\}_{n} + \mathbf{C}\left\{\Delta\dot{\mathbf{d}}\right\}_{n} + \left(\left[\mathbf{K}^{\text{int}}\right]_n - \left[\mathbf{K}^{\text{ext}}\right]_n\right)\{\Delta\mathbf{d}\}_n = \{\Delta\mathbf{r}\}_n\,. \tag{3.19}$$

In a non-linear context, $\mathbf{K}^{\text{int}}$ and $\mathbf{K}^{\text{ext}}$ are in general functions of the displacement. Equivalently, the gradients of the internal and external forces need not be linear. Thus, due to the linearizations in (3.18), the equilibrium equation at time $n+1$ is no longer exactly satisfied, giving a non-zero residual $\{\mathbf{r}\}_{n+1}$. In 3.3.3, a method for eliminating this residual is presented.

3.3.2 The Newmark-β integrators

The Newmark-β or Newmark's method is a commonly used implicit time-stepping algorithm for dynamic analysis (not necessarily non-linear) in the finite element method[4]. The method is actually a family of methods, with the two parameters β and γ defining the characteristics of the algorithm.

The Newmark approximations are derived by discretizing the dynamic equation of motion in time[31], yielding the following formulas for the increment in accelerations and velocities from time n to $n+1$:

$$\left\{\Delta\ddot{\mathbf{d}}\right\}_n = \frac{1}{\beta\left(\Delta t\right)^2}\left\{\Delta\mathbf{d}\right\}_n - \frac{1}{\beta\Delta t}\left\{\dot{\mathbf{d}}\right\}_n - \left(\frac{1}{2\beta}\right)\left\{\ddot{\mathbf{d}}\right\}_n \tag{3.20a}$$

$$\left\{\Delta\dot{\mathbf{d}}\right\}_n = \frac{\gamma}{\beta\Delta t}\left\{\Delta\mathbf{d}\right\}_n - \frac{\gamma}{\beta}\left\{\dot{\mathbf{d}}\right\}_n - \Delta t\left(\frac{\gamma}{2\beta}-1\right)\left\{\ddot{\mathbf{d}}\right\}_n . \tag{3.20b}$$

It is noted that knowledge about the displacement increment $\Delta\mathbf{d}_n$ is sufficient to compute the increments in velocity and acceleration. A commonly used choice for the parameters is $\beta = 1/4$ and $\gamma = 1/2$, implying the assumption of a constant average acceleration between each time step.

Substituting these expressions into the equation of motion in (3.19) gives the final incremental form of the implicit time-stepping scheme:

$$\left[\mathbf{K}^{\text{eff}}\right]_n\left\{\Delta\mathbf{d}\right\} = \left\{\Delta\mathbf{f}^{\text{eff}}\right\}_n \tag{3.21a}$$

where

$$\left[\mathbf{K}^{\text{eff}}\right]_n = \frac{1}{\beta\Delta t^2}\mathbf{M} + \frac{\gamma}{\beta\Delta t}\mathbf{C} + \left[\mathbf{K}^{\text{int}}\right]_n - \left[\mathbf{K}^{\text{ext}}\right]_n \tag{3.21b}$$

$$\begin{aligned}\left\{\Delta\mathbf{f}^{\text{eff}}\right\}_n &= \left\{\Delta\mathbf{r}\right\}_n \\ &\quad + \mathbf{M}\left[\frac{1}{\beta\Delta t}\left\{\dot{\mathbf{d}}\right\}_n + \left(\frac{1}{2\beta}-1\right)\left\{\ddot{\mathbf{d}}\right\}_n\right] \\ &\quad + \mathbf{C}\left[\left(\frac{\gamma}{\beta}-1\right)\left\{\dot{\mathbf{d}}\right\}_n + \Delta t\left(\frac{\gamma}{2\beta}-1\right)\left\{\ddot{\mathbf{d}}\right\}_n\right] .\end{aligned} \tag{3.21c}$$

By defining

$$\mathbf{A} = \frac{1}{\beta\Delta t}\mathbf{M} + \frac{\gamma}{\beta}\mathbf{C} \quad \text{and} \quad \mathbf{B} = \frac{1}{2\beta}\mathbf{M} + \Delta t\left(\frac{\gamma}{2\beta}-1\right)\mathbf{C} \tag{3.22}$$

the effective load residual in (3.21c) simplifies to

$$\left\{\Delta\mathbf{f}^{\text{eff}}\right\}_n = \left\{\Delta\mathbf{r}\right\}_n + \mathbf{A}\left\{\dot{\mathbf{d}}\right\}_n + \mathbf{B}\left\{\ddot{\mathbf{d}}\right\}_n \tag{3.23}$$

It is noted that $\mathbf{K}^{\text{eff}}$ acts as an effective tangent stiffness, and $\Delta\mathbf{f}^{\text{eff}}$ as an effective load residual.

In general, the mass and damping terms would also be function of the displacements. In that case, it should be noted that the equilibrium equation is stated at time $n+1$, and hence $\mathbf{M}_{n+1}$ and $\mathbf{C}_{n+1}$ should be used for the equation to be correct.

Once the incremental displacement $\Delta\mathbf{d}$ has been computed, the updated displacements, velocities and accelerations can be computed by the increments in (3.20).

3.3.3 Newton-Raphson iterations

As showed in the previous section, the Newmark-β scheme gives a linear algebraic system of equations which can be solved for the incremental displacement, $\mathbf{d}_{n+1}$. In a *linear* context, the tangent stiffness remains unchanged with respect to displacements, giving an exactly correct result for the displacement $\{\mathbf{d}\}_{n+1}$. However, in a non-linear context, the tangent stiffness changes with the displacements, so that the solution for the displacements at time $n+1$ is not exactly correct (due to the linearization in (3.18)). In order to control this error, the residual at time $n+1$ should hence be corrected by a set of iterations until a desired degree of tolerance is reached.

The Newton-Raphson iteration scheme corrects the residual by repeatedly computing a new incremental displacement and the corresponding residual forces—i.e. the internal and external forces resulting from the iterated displacement. Its basic procedure is given in e.g. Chopra (2006)[9].

3.4 Mass and stiffness tensors

The mass term $\mathbf{M}$ in the incremental equation of motion (Eq. 3.21) relates the nodal accelerations to the inertia forces. For the element, the lumped mass and consistent mass formulations are often employed. While a lumped mass formulation leads to terms only on the diagonal, and hence no coupling between inertia forces, a consistent mass formulation is derived with basis in the same shape functions used for establishing the stiffness, and gives coupled inertia forces. Generally, consistent masses produce more correct results at the expense of a more computationally demanding system of equations.

Conventionally the mass term on the element level for e.g. the beam element is expressed in terms of a matrix, which relates accelerations and inertia forces in terms of scalar degrees-of-freedom. In the present context, however, the element properties are instead defined in terms of tensors for use in the present finite element framework. These tensors relate vector quantities of acceleration to vector quantities of inertia forces. A complete derivation of the expressions in this section will not be given—rather they are established by inspection of tensor expressions derived by other authors[34, Eq. 46].

3.4.1 Material stiffness tensors

The tensor relations for the material stiffness is given by[33]

$$\mathbf{K}^e\mathbf{d} = \begin{bmatrix} \mathbf{K}_{fu} & \mathbf{K}_{f\theta} & -\mathbf{K}_{fu} & \mathbf{K}_{f\theta} \\ \mathbf{K}_{mu} & \mathbf{K}_{m\theta} & -\mathbf{K}_{mu} & \hat{\mathbf{K}}_{m\theta} \\ -\mathbf{K}_{fu} & -\mathbf{K}_{f\theta} & \mathbf{K}_{fu} & -\mathbf{K}_{f\theta} \\ \mathbf{K}_{mu} & \hat{\mathbf{K}}_{m\theta} & -\mathbf{K}_{mu} & \mathbf{K}_{m\theta} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \boldsymbol{\theta}_1 \\ \mathbf{u}_2 \\ \boldsymbol{\theta}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{m}_1 \\ \mathbf{f}_2 \\ \mathbf{m}_2 \end{bmatrix}, \tag{3.24}$$

where the individual tensors are given as

$$\mathbf{K}_{fu} = \frac{12E}{L^3}\hat{\mathbf{I}} + \frac{AE}{L}(\mathbf{n}\otimes\mathbf{n}) \tag{3.25a}$$

$$\mathbf{K}_{mu} = \mathbf{K}_{f\theta}^{\mathrm{T}} = \frac{6E}{L^2}(\mathbf{t}\otimes\mathbf{s} - \mathbf{s}\otimes\mathbf{t}) \tag{3.25b}$$

$$\mathbf{K}_{m\theta} = \frac{4E}{L}\hat{\mathbf{I}} + \frac{JG}{L}(\mathbf{n}\otimes\mathbf{n}) \tag{3.25c}$$

$$\hat{\mathbf{K}}_{m\theta} = \frac{2E}{L} - \frac{JG}{L}(\mathbf{n}\otimes\mathbf{n}) \ . \tag{3.25d}$$

It is noted that $\mathbf{K}_{mu}$ and $\hat{\mathbf{K}}_{m\theta}$ are symmetric tensors, and hence $\mathbf{K}^e$ remains symmetric even when the transpose operator has been left out in (3.24).

3.4.2 Consistent mass tensors

The following addresses tensors expressions for consistent nodal masses of a linear Euler-Bernoulli beam element, as well as for the geometric stiffness. The well-known, classical matrix equivalents can be found in e.g. the finite element literature, e.g. in [5, p. 141]. Instead of a formal derivation of these expression, they are instead devised by inspecting the stiffness expressions (3.24) and (3.25), and the matrix equivalents, accounting for the different symmetry of the mass matrix compared to the stiffness matrix.

The nodal force–acceleration relation can, similarly to 3.4.1, be stated as

$$\begin{bmatrix} \mathbf{M}_{fu} & \mathbf{M}_{f\theta} & \hat{\mathbf{M}}_{fu} & \hat{\mathbf{M}}_{f\theta} \\ \mathbf{M}_{mu} & \mathbf{M}_{m\theta} & \hat{\mathbf{M}}_{mu} & \hat{\mathbf{M}}_{m\theta} \\ \hat{\mathbf{M}}_{fu}^{\mathrm{T}} & \hat{\mathbf{M}}_{mu}^{\mathrm{T}} & \mathbf{M}_{fu} & -\mathbf{M}_{f\theta} \\ \hat{\mathbf{M}}_{f\theta}^{\mathrm{T}} & \hat{\mathbf{M}}_{m\theta}^{\mathrm{T}} & -\mathbf{M}_{f\theta}^{\mathrm{T}} & \mathbf{M}_{m\theta} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{u}}_1 \\ \ddot{\boldsymbol{\theta}}_1 \\ \ddot{\mathbf{u}}_2 \\ \ddot{\boldsymbol{\theta}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{m}_1 \\ \mathbf{f}_2 \\ \mathbf{m}_2 \end{bmatrix} \tag{3.26}$$

With M being the total mass of the element, we get the following consistent mass tensors

$$\mathbf{M}_{fu} = M\left[\frac{1}{3}\mathbf{n}\otimes\mathbf{n} + \frac{13}{35}(\mathbf{s}\otimes\mathbf{s} + \mathbf{t}\otimes\mathbf{t})\right] \tag{3.27a}$$

$$\hat{\mathbf{M}}_{fu} = M\left[\frac{1}{6}\mathbf{n}\otimes\mathbf{n} + \frac{9}{70}(\mathbf{s}\otimes\mathbf{s} + \mathbf{t}\otimes\mathbf{t})\right] \tag{3.27b}$$

$$\mathbf{M}_{mu} = \mathbf{M}_{f\theta}^{\mathrm{T}} = ML\left[\frac{11}{210}(\mathbf{t}\otimes\mathbf{s} - \mathbf{s}\otimes\mathbf{t})\right] \tag{3.27c}$$

$$\hat{\mathbf{M}}_{mu} = \hat{\mathbf{M}}_{f\theta} = ML\left[\frac{13}{420}(\mathbf{t}\otimes\mathbf{s} - \mathbf{s}\otimes\mathbf{t})\right] \tag{3.27d}$$

$$\mathbf{M}_{m\theta} = M\left[\frac{L^2}{105}\hat{\mathbf{I}} + \frac{I_p}{3A}(\mathbf{n}\otimes\mathbf{n})\right] \tag{3.27e}$$

$$\hat{\mathbf{M}}_{m\theta} = M\left[-\frac{L^2}{140}\hat{\mathbf{I}} + \frac{I_p}{6A}(\mathbf{n}\otimes\mathbf{n})\right] , \tag{3.27f}$$

3.4.3 Geometric stiffness tensor

In a similar manner as in the previous sections, we establish the tensor expressions equivalent to the ordinary geometric stiffness of a beam element. Let N be the axial force, with a tensile axial force being positive, then

$$\mathbf{K}_{fu}^{g} = N\frac{36}{30L}(\mathbf{s}\otimes\mathbf{s} + \mathbf{t}\otimes\mathbf{t}) \tag{3.28a}$$

$$\mathbf{K}_{mu}^{g} = N\frac{3}{30}(\mathbf{t}\otimes\mathbf{s} - \mathbf{s}\otimes\mathbf{t}) \tag{3.28b}$$

$$\mathbf{K}_{m\theta}^{g} = N\frac{4L}{30}(\mathbf{t}\otimes\mathbf{t} + \mathbf{s}\otimes\mathbf{s}) \tag{3.28c}$$

$$\hat{\mathbf{K}}_{m\theta}^{g} = -N\frac{L}{30}(\mathbf{t}\otimes\mathbf{t} + \mathbf{s}\otimes\mathbf{s}) , \tag{3.28d}$$

where it is implied that unmentioned tensor elements are zero.

The total element stiffness with the geometric stiffness included is then given as

$$\mathbf{K} = \mathbf{K}^{e} + \mathbf{K}^{g} . \tag{3.29}$$

4 Corotational formulation of beam elements

4.1 Introduction

The corotational formulation has been chosen as the approach for geometrically nonlinear analysis in this work, due to its attractive features. A neccessary theoretical treatment is presented in the following.

4.1.1 Background

The use of corotated elements for geometric nonlinear analysis has been studied in several papers. Crisfield [14] gives an algorithm for 3D beam elements, although only in a static context. Later, Crisfield gives a formulation for a beam element in a dynamic context[15]. Haugen and Felippa [20] establish a unified formulation, for arbitrary types of elements, and give a general expression for the consistent tangent stiffness. Another important reference is is by Simo and Vu-Quoc (1986)[42]. It is pointed out that the notations and derivations given in this section are mainly based on the dissertation of Haugen [18].

The use of corotated elements is motivated by the assumption of small strains, still allowing the large rigid-body displacements of the element. In contrast to e.g. a total lagrangian (TL) formulation, the ordinary linear strain measure of the element is conserved, where as a finite strain measure would have to be used in a total lagrangian formulation. With the corotated formulation, one can effectively re-use existing linear element libraries in a geometric nonlinear context.

Although most of the theory presented in this chapter is applicable to arbitrary finite element types, the main focus will be on the 3D Euler–Bernoulli beam element. Unless specified, all vectors are referred to in the global inertial coordinate system.

4.1.2 Basic concept

The basic concept of corotated finite elements is to split the (current) total translation and rotation into a rigid-body part that produces no forces or moments in the element, and a deformational part which produces the internal forces and moments of the element. At any point where information about the internal forces is required, the corotated element's position and orientation (which corresponds to the rigid-body part) is established. This orientation is also referred to as a shadow element or ghost element.

The element itself is assumed to be linear (i.e. small strains), so that the internal force–displacement relation remains linear. It is noted that the corotated (or shadow) element only exists for visualizing the corotated configuration, and is never established as an element in the conventional sense.

4.1.3 Configurations

The initial configuration of the element is denoted C_0, where there are no deformations. As the element deforms, it ends up in the deformed configuration, C_n. An illustration of these configurations for a two-dimensional beam element, as well as a possible placement of the corotated configuration, C_{0n} is illustrated in Figure 4.1.

Table 4.2: Configurations

C_0	Initial (e.g. when $t = 0$).
C_{0n}	Corotated. Is a rigid body motion of C_0.
C_n	Deformed. Is close to C_{0n}.

4.1.4 Directions and transformation

In the undeformed configuration, C_0, the orientation in space of an undeformed beam element is defined by its three orthogonal principal directions, or base vectors: $\mathbf{n}_0$, $\mathbf{s}_0$ and $\mathbf{t}_0$, where $\mathbf{n}_0$ is the initial direction of the element. The initial orientation can hence be defined by the orthonormal transformation tensor

$$\mathbf{T}_0 = \begin{bmatrix} \mathbf{n}_0^{\mathrm{T}} \\ \mathbf{s}_0^{\mathrm{T}} \\ \mathbf{t}_0^{\mathrm{T}} \end{bmatrix} , \tag{4.30}$$

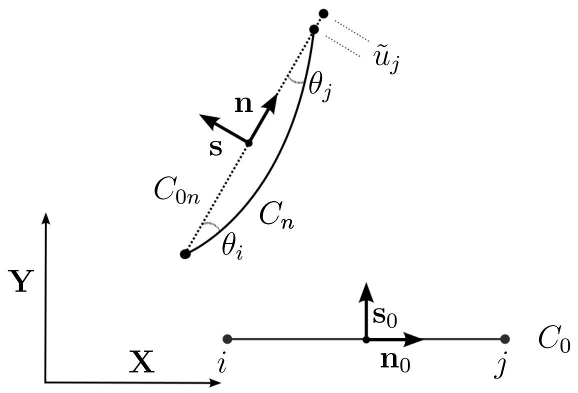


Figure 4.1: The initial, deformed, and corotated configuration of a 2D beam element. The deformations are exaggerated.

which remains constant as the element deforms.

The base vectors of the corotated element, C_{0n}, can similarly be collected into the orthonormal transformation tensor

$$\mathbf{T}_n = \begin{bmatrix} \mathbf{n}^{\mathrm{T}} \\ \mathbf{s}^{\mathrm{T}} \\ \mathbf{t}^{\mathrm{T}} \end{bmatrix} . \tag{4.31}$$

The determination of these corotated base vectors is further discussed in Section 4.2.3.

A vector $\mathbf{x}$ in the global coordinate system can be transformed to a vector $\tilde{\mathbf{x}}$ in the element's local $(\mathbf{n}, \mathbf{s}, \mathbf{t})$ system by the transformation rule

$$\tilde{\mathbf{x}} = \mathbf{T}_0 \mathbf{x} \quad \text{and} \quad \mathbf{x} = \mathbf{T}_0^{\mathrm{T}} \tilde{\mathbf{x}} . \tag{4.32}$$

4.2 Extraction of deformations

Let the finite displacement in C_n of node i in Figure 4.1 is described by the translation vector $\mathbf{u}_i$ and the rotation tensor $\mathbf{R}_i$. With knowledge about these quantities for both nodes of the element, an essential procedure in the corotational formulation is to extract from these the deformational translations and rotations of the beam element at its two nodes.

4.2.1 Extraction of deformational translations

By decomposing the total translation into a rigid-body translation and a deformational translation, the internal forces can be computed in terms of the latter component. The position and orientation of the corotated element (or shadow element), must therefore be established.

For arbitrary elements, some best-fit procedure should be used. For the beam element considered here, several placements are possible:

- On the straight line between the nodes.
- On a line giving no deformational rotation in one of the nodes.
- With its center of mass coinciding with the center of mass of the deformed element.

The first method, which corresponds to Figure 4.1, seems to be the simplest, and has been used in the present work. This gives an axial deformational translation only at node j, which is expressed in the element's local coordinate system as:

$$\tilde{\mathbf{u}}_d = \begin{bmatrix} \tilde{u} & 0 & 0 \end{bmatrix}^{\mathrm{T}} \quad \text{at node } j \tag{4.33}$$

$$\tilde{\mathbf{u}}_d = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}} \quad \text{at node } i \tag{4.34}$$

Note that the axial deformation could also have been divided equally between the two nodes, however, this would give a completely equivalent internal force of the element, as the *relative* displacement of the two nodes determines the axial force.

For a node having a translational displacement of $\mathbf{u}$, the displaced position is given by

$$\mathbf{x} = \mathbf{x}^0 + \mathbf{u} \ ,$$

The length of the deformed beam element is simply the distance between the two nodes, $l_n = |\mathbf{x}_j - \mathbf{x}_i|$, giving an axial elongation of

$$\tilde{u}_l = l_n - l_0 \ ,$$

where l_0 is the initial element length. To avoid numerical problems, the mid-point formula [14, Eq. (26)] should be adopted:

$$\tilde{u}_l = \frac{2}{l_n + l_0} \left[(\mathbf{x}_j - \mathbf{x}_i) + \frac{1}{2} (\mathbf{u}_j - \mathbf{u}_i) \right]^{\mathrm{T}} (\mathbf{u}_j - \mathbf{u}_i) \ . \tag{4.35}$$

Transforming the local axial displacement to global coordinates leads to

$$\mathbf{u}_d = \mathbf{T}_n^{\mathrm{T}} \tilde{\mathbf{u}}_d \, , \tag{4.36}$$

with $\tilde{\mathbf{u}}_d$ from (4.35) and $\mathbf{T}_n$ from (4.31).

4.2.2 Extraction of deformational rotations

In addition to the deformational translations, the deformational rotations must also be extracted, so that the internal elastic moments can be computed. As detailed in [18], the orientation of the shadow element (i.e. C_{0n}) can be obtained by a rigid-body rotation of the previously defined initial base vectors of the element:

$$\begin{aligned} \mathbf{n} &= \mathbf{R}_{0n}\mathbf{n}_0 \\ \mathbf{s} &= \mathbf{R}_{0n}\mathbf{s}_0 \\ \mathbf{t} &= \mathbf{R}_{0n}\mathbf{t}_0 \end{aligned} \tag{4.37}$$

where subscript $0n$ indicates that the tensor rotates the base vectors from the initial to the corotated configuration. These expressions can be expressed more compactly in terms of the tensors in equations 4.30 and 4.31:

$$\mathbf{T}_n^{\mathrm{T}} = \mathbf{R}_{0n}\mathbf{T}_0^{\mathrm{T}} \ , \tag{4.38}$$

which by post-multiplying by $\mathbf{T}_0$ gives

$$\mathbf{R}_{0n} = \mathbf{T}_n^{\mathrm{T}}\mathbf{T}_0 \ . \tag{4.39}$$

Further, if we assume that the total rotation of the node from C_0 to C_n can be expressed by a rotation tensor, $\mathbf{R}$, we proceed by decomposing this rotation into a rigid-body component and a deformational component. In contrast to translational components, which *are additive*, the total rotation will have to be decomposed into a *product* of a rigid-body rotation tensor and a deformational rotation tensor:

$$\mathbf{R} = \mathbf{R}_d\mathbf{R}_{0n} \ , \tag{4.40}$$

where the order of multiplication used here–which is *not* arbitrary–is in agreement with other authors[18].

The rigid-body rotation from the initial to the corotated configuration (i.e. $\mathbf{R}_{0n}$) was defined in (4.39), which substituted into the previous equation leads to the final expression for the deformational rotation at a node:

$$\mathbf{R}_d = \mathbf{R}\mathbf{R}_{0n}^{\mathrm{T}} = \mathbf{R}\mathbf{T}_0^{\mathrm{T}}\mathbf{T}_n \, . \tag{4.41}$$

With Figure 4.1 in mind, $\mathbf{R}_d$ for node j would be the tensor that describes a rotation about the out-of-plane axis by the angle θ_j, and similarly for node i (although a negative rotation). As seen in (4.41), if $\mathbf{R} = \mathbf{R}_{0n}$ then $\mathbf{R} = \mathbf{I}$, demonstrating that a pure rigid body motion of the element correctly leads to no resulting deformational rotations.

Due to the assumption of small strains in the corotational formulation, the rotation represented by $\mathbf{R}_d$ is also small. Due to this fact, the tensor can be converted to a rotation vector as in 2.4.3. Such a conversion is necessary e.g. when computing the interal moments, as the typical linear force–displacement relation (i.e. stiffness tensor) expresses the nodal moment in terms of the rotation *vector* (see (3.24)).

4.2.3 Establishing the corotated base vectors

As already mentioned, for arbitrary element types, the position and orientation of C_{0n} should be determined by som best-fit procedure. In this work, where only beam elements are considered, a simple geometric consideration is sufficient for identifying the directions of the corotated base vectors, which collectively define the transformation tensor $\mathbf{T}_n$.

By establishing the orientation of C_{0n} by placing the shadow element on the straight line between the deformed positions of the two nodes—$\mathbf{x}_i$ and $\mathbf{x}_j$, respectively—the direction of the corotated element can be easily computed as

$$\mathbf{n} = \frac{\mathbf{x}_j - \mathbf{x}_i}{||\mathbf{x}_j - \mathbf{r}_j||} \,. \tag{4.42a}$$

Let the current *total* rotations of node i and j of the element be $\mathbf{R}_i$ and $\mathbf{R}_j$, respectively. In general, the two rotations and their corresponding tensors need not be equal, due to the deformation of the element. By rotating the initial base vector $\mathbf{t}_0$ of the element by the average rotation of the two nodes, an approximate direction of the corotated base vector $\mathbf{t}$ is established:

$$\mathbf{t}^* = \mathbf{R}_i\mathbf{t}_0 + \mathbf{R}_j\mathbf{t}_0 \,. \tag{4.42b}$$

In general, $\mathbf{t}^*$ is not perpendicular to $\mathbf{n}$, so by first computing the $\mathbf{s}$ base vector by the cross-product

$$\mathbf{s} = \frac{\mathbf{t}^* \times \mathbf{n}}{|\mathbf{t}^* \times \mathbf{n}|} \,, \tag{4.42c}$$

and then finally $\mathbf{t}$ as

$$\mathbf{t} = \mathbf{n} \times \mathbf{s} \,, \tag{4.42d}$$

the tri-orthogonality of the base vectors is maintained. It is observed that this method requires $\mathbf{t}^*$ and $\mathbf{n}$ not to be parallel, although these would only be parallel if the rotation of the two nodes relative to eachother is very large, and the deformations of the element has here been assumed to be small.

4.3 The tangent stiffness

4.3.1 Consistent tangent stiffness

As shown in (3.19) and (3.18), the consistent tangent stiffness of the element is the gradient of the internal forces. For the corotational formulation of an arbitrary element, Haugen derives the consistent tangent stiffness by taking the variation of all terms occuring in the expression of the element's internal forces, leading to

$$\mathbf{K}^{\text{int}} = \mathbf{K}_{\text{GR}} + \mathbf{K}_{\text{GP}} + \mathbf{K}_{\text{GM}} + \mathbf{K}_{\text{M}}\,, \tag{4.43}$$

where the individual terms are named as following: $\mathbf{K}_{\text{GM}}$ is the moment-correction geometric stiffnes, $\mathbf{K}_{\text{GP}}$ is the equilibrium-projection geometric stiffness, and $\mathbf{K}_{\text{GR}}$ is the rotational geometric stiffness, while $\mathbf{K}_{\text{M}}$ is the material stiffness. Due to complexity, a detailed description of these terms is outside the scope of this thesis.

Bergan et al., on the assumption that the deformational rotations are small and that the axial elongation of the element is negligible with respect to the equilbrium of the element, disregard terms $\mathbf{K}_{\text{GP}}$ and $\mathbf{K}_{\text{GM}}$ from (4.43). Also, these assumptions lead to $\mathbf{K}_{\text{M}} = \mathbf{K}^{e}$, i.e. the linear element stiffness. The resulting formulation has been termed the *Consistent co-rotated formulation*, or abbreviated to *(C)*.

4.3.2 A modified tangent stiffness

As already mentioned, in circumstances where deformations are very small, terms of the consistent tangent stiffness may be neglected. Another simplification is here suggested, namely that also the rotational geometric stiffness $\mathbf{K}_{\text{GR}}$ be disregarded. Furthermore, the conventional linear geometric stiffness of the beam, giving stiffening due to axial strains, is included, giving

$$\mathbf{K}^{\text{int}} = \mathbf{K}^{e} + \mathbf{K}^{G}\,, \tag{4.44}$$

with the last term defined in (3.28d). It is pointed out that this modification (due to dependency on the axial force) leads to the element no longer being considered linear, at the advantage of capturing the stiffening or softening effect of a beam due to axial force. To achieve convergence towards the correct state of equilbrium, the stiffness in (4.44) must also be used in the computation of the interal forces. Since $\mathbf{K}^{G}$ is a function of the axial force, the "most recent" value of the axial force should be used in the computation.

The suggested tangent stiffness is utilized in the implementation presented later in this thesis. It is stressed that no formal justification is given for this modification, however, numerical results in later chapters show that the formulation behaves well. A similar tangent stiffness has been used by Hsiao and Jang with good results [21].

4.3.3 Remarks on the consistent tangent stiffness

To achieve a true second order convergence rate, the consistent tangent stiffness should be used in an incremental-iterative scheme. However, a sufficient rate of convergence can often be achieved with a simplified tangent stiffness. Although utilizing the full consistent stiffness minimizes the number of iterations needed for convergence, the efficiency in terms of running time need not be minimized due to more demanding operations, e.g. in the assembly of the tangent stiffness. Also, more effort of the programmer is required when implementing a complicated consistent stiffness compared to a simpler tangent stiffnes.

A final remark is that a converged solution is indeed equally correct (with respect to the internal forces), independent of the choice of the tangent stiffness.

4.4 Summary of procedure

In the previous sections, the necessary procedures for extracting the deformational state of the element was established. Here, a summary of the complete procedure in the context of an incremental solution process will be given in order to clearify the flow of the algorithm.

The tensor representation of rotations has been used consistently previously in this chapter, but other representations can be employed with equivalent results. As shown in 2.5.2, the decomposition of rigid body and deformational rotations in (4.40) could similarily be carried out with the use of quaternions instead of tensors.

Consider one single element of an arbitrary finite element mesh. The initial principal axes/base vectors of the element are $\mathbf{n}_0$, $\mathbf{s}_0$ and $\mathbf{t}_0$, and these base vectors collectively define the element's initial transformation tensor $\mathbf{T}_0$, according to (4.30), which should be stored during the analysis process. Further, the initial position of each node is $\mathbf{x}^0$ and the initial translations and rotations are assumed zero.

Initialization (C_0).

For each node, set and store

$$\mathbf{u} := \mathbf{0} \quad \text{and} \quad \mathbf{R} := \mathbf{I}\,. \tag{4.45}$$

For each element, set

$$\mathbf{T}_0 := \begin{bmatrix} \mathbf{n}_0^{\mathrm{T}} \\ \mathbf{s}_0^{\mathrm{T}} \\ \mathbf{t}_0^{\mathrm{T}} \end{bmatrix} \quad \text{and} \quad \mathbf{R}_d := \mathbf{I} \tag{4.46}$$

Incrementing translations and rotations.
An incremental-iterative solver working on global coordinates typically solves for the increment (or iterate) in translations and rotations for each node. For a single node and arbitrary increment, these increments:

$$\Delta\mathbf{u} = \begin{bmatrix} \Delta u & \Delta v & \Delta w \end{bmatrix}^{\mathrm{T}} \tag{4.47}$$

$$\Delta\boldsymbol{\theta} = \begin{bmatrix} \Delta\theta_x & \Delta\theta_y & \Delta\theta_z \end{bmatrix}^{\mathrm{T}} \tag{4.48}$$

The accumulation of translations is performed by *adding* the displacement increment $\Delta\mathbf{u}^n$ to the current displacement vector. On the other hand, the increment in rotation, usually given by the solver as the rotation vector $\Delta\boldsymbol{\theta}^n$, can *not* be added to the current rotational displacement. Instead, the current rotational displacement should be stored as a tensor or quaternion (per node), and updated with the incremented rotation (converted to a tensor or quaternion), as outlined in Sec. 2.5. If a tensor representation is used for the total rotation, the tensor that represents the increment in (4.48) is computed by the Rodrigues formula in (2.11). Alternatively, using quaternions, the rotation increment is converted by (2.9).

Let $\mathbf{R}_\Delta = \mathbf{R}_\Delta\left(\Delta\boldsymbol{\theta}\right)$ be the tensor representing the increment in rotations, given by (2.11), then for each node the position and rotation is updated by

$$\mathbf{u} := \mathbf{u} + \Delta\mathbf{u} \tag{4.49}$$

$$\mathbf{R} := \mathbf{R}_\Delta\,\mathbf{R}\,. \tag{4.50}$$

Extraction of deformations.
In a nonlinear context, equilibrium iterations are usually performed in order to eliminate the out-of-balance forces, as discussed previously in Sec. 3.3.3. Such a procedure requires the computation of the current internal forces of the element, and hence requires the extraction of the deformational translations and rotations of the element. The necessary steps has been detailed in Sec. 4.2. It is noted

that such a computation is performed on an per-*element* basis, contrary to the per-node incrementation of displacements.

For each element, compute and store the corotated orientation defined by the tensor $\mathbf{T}_n$, by (4.42) and (4.31). Extract the deformational translations and rotations at the two nodes, as shown in Sec. 4.2.

Computing internal forces

By converting the tensor representations of the deformational rotations at the two nodes to vectors by (2.12), the internal forces are given by the linear stiffness relation of the beam element. Let $\mathbf{u}_d^i$, $\mathbf{u}_d^j$, $\boldsymbol{\theta}_d^i$, and $\boldsymbol{\theta}_d^j$ be the deformational translations and rotations at nodes i and j of the element. The internal forces and moments are then given by:

$$\begin{bmatrix} \mathbf{f}^i \\ \mathbf{m}^i \\ \mathbf{f}^j \\ \mathbf{m}^j \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{u}_d^i \\ \boldsymbol{\theta}_d^i \\ \mathbf{u}_d^j \\ \boldsymbol{\theta}_d^j \end{bmatrix} , \tag{4.51}$$

where $\mathbf{K}$ is the linear stiffness of the element in (3.24), computed in the orientation of the corotated configuration ($\mathbf{n}$, $\mathbf{s}$, $\mathbf{t}$). If the geometric stiffness is included, as suggested in (4.44), this term should also be included here.

5 Implementation of the application

5.1 Introduction

One of the main objectives of this work has been to develop a tool that facilitates the large-displacement analysis of wind turbines. The chosen approach was to use an existing framework as the starting point, and to modify and extend the source code with the necessary functionality. Due to previous developments having been carried out entirely on object-oriented principles and with extensibility in mind, such an approach was deemed feasible. This chapter presents the background of the framework, as well as extensions made.

5.2 Background of the framework

The framework used here was developed in C++ on object-oriented principles, as described by Miller[32] and Miller and Rucki (1996, 1998)[37,38] and further by the work of Jang[23]. In recent years, Thomassen et al. worked on the implementation of specialized code for the analysis of wind turbines[6, 43].

The framework, on the core level, provides functionality for the creation of finite element meshes and the static and dynamic time-simulation of the response. Instead of the classical approach, where matrices and scalars are used for e.g. stiffness and displacements of degrees-of-freedom, the underlying scalars are encapsulated within tensor and vector classes, allowing for formulating element properties in a coordinate-free manner[33]. Further, an important focus has been the development of a graphical user interface (GUI) that is closely coupled to the underlying framework, providing for an interactive, real-time user experience. All code is written in C++ on object-oriented principles.

The work by Thomassen, Suja, Bruheim and Frøyd[43] provided new features for the analysis of wind turbine design, such as the blade element momentum theory (BEM) that computes aerodynamic forces on the blades, flexible settings for the input of data to the code, as well as improvements to the GUI. Additionally, functionality for obtaining the eigenvalues of the model was implemented[6].

5.3 Extensions

To facilitate the implementation of corotated beam elements, a large set of extensions and modifications to the framework has been made. The most important are:

- Extension of the `FrameElt` class to a sub-class `FrameEltCorot`. Since the logic of the corotated element is different than the linear element, e.g. with respect to computation of internal forces and computation of stiffness and mass tensors, functions for such computations are overridden in `FrameEltCorot`.
- Definition of tool funtions for converting between e.g. rotation tensors and vectors.
- Modification of the DOF class, `NewmarkDOF`, to accommodate the corotational update procedure.
- Implementation of the `GQuat` class, which provides support for quaternion operations.
- Definition of the `Sensor` class, that provides the real-time display of relevant quantities and recording/outputting data from simulations. Implementation of e.g. `SensorNode` that records e.g. nodal displacements and velocities.
- Improvements of the graphical user interface, e.g. visualizing the deformed blades during real-time analysis.

It is noted that the term “frame” is used for the beam element classes. In Appendix A, excerpts from the source code is given for classes and functions related to the corotational procedures and the nonlinear solution process.

5.3.1 The `FrameElt` class

The existing implementation of the linear beam element is represented by the `FrameElt` class. Some necessary extensions and modifications made to this class were:

- Support for using consistent masses and including the geometric stiffnes when installing these in the DOFs in InstallMCK().
- The ComputeStiffness() function, giving the possibility for computing the set of stiffness tensors for the element in different coordinate systems, e.g. in element-local or global, in either the undeformed or deformed configuration. Also, the inclusion of the geometric stiffness could be toggled on or off.
- The ComputeMass() function. Similar to the previous modification, but for computing the mass tensors.
- The GetMomenta() function, computing the nodal momenta, i.e. the product of the nodal masses and velocities.

It should be noted that added functions were also declared as virtual functions in the top-level element class, Element, making the implementation of elements easy.

5.3.2 The FrameEltCorot class

The FrameEltCorot class inherits from the FrameElt class, and overrides neccessary functions of FrameElt for the corotational procedures. In the existing classes GTensor and GSymmTensor, necessary functions were provided for e.g. tensor multiplications etc.

- GetElasticForces() and GetStrainEnergy(), which for the corotated element must compute the internal forces and energy in terms of the *deformational* translations and rotations at the nodes.
- UpdateResistingForce(), called during equilbrium iterations, which must perform the corotational extraction of deformational translations and rotations and establish the base vectors before computing the internal forces and applying these to the DOFs.

5.3.3 Updating of DOFs

For the incorporation of corotated elements, procedures that update displacements, velocities and accelerations of the DOFs must be modified. Additionally, for linear elements, the usual additive update of variables should be used. For this, each Node stores a flag that signalizes if it is connected to a corotated element. See also the source code given in A.6.

5.4 Documentation of the framework

5.4.1 Documentation of the source code

The free software package Doxygen[45] provides functionality for automatically generating documentation of the source code in the form of html pages, by parsing the entire source code structure of the application. When working on the same code in a team, such documentation is obviously very helpful when e.g. interfacing with others' classes. This documentation might also serve as the main documentation library for a project involving many developers. In the context of wind turbine design, Quarton points out that a well-documented software is essential for the transition from a research code to a design tool[35, p. 19].

In the present work, this package has been successfully used for documenting all classes, including member variables and member functions. Due to active parallel development, the source code resides on a remote server with version control, which makes it possible to always have available an updated documentation of the current state of the source code.

In Figure 5.2, the syntax for documenting a member function is demonstrated.

5.4.2 Sequence diagrams

These figures illustrate the interaction between the different objects during the solution process. In Figure 5.3, the solver sequence is shown for a dynamic linear analysis, while Figure 5.4 shows a nonlinear analysis where equilibrium iterations are performed.

5.4.3 Corotational procedures

As seen in Figure 5.3, the `Element` has the responsibility of computing the internal forces and applying them to the `DOF`s during equilbrium iterations, through the `UpdateResistingForce` function. For the corotated element, this procedure must establish the corotated configuration and extract the internal deformations, as detailed in previous chapters.

```
/**
 * Calculates the nodal loads consistent with the current
 * displacement of the nodes/DOFs.
 *
 * Only the elastic forces are considered. The function
 * uses this element's current stiffness tensor members
 * for calculating, and does not consider the stiffness
 * properties of the DOFs and their interactions.
 *
 * All results are given in global coordinates. The function
 * has been tested and checked against basic beam formulas.
 *
 * @param fi Will be modified to the left force vector.
 * @param fj Will be modified to the right force vector.
 * @param mi Will be modified to the left moment vector.
 * @param mj Will be modified to the right moment vector.
 * @param NRiter (optional) If set to true, the displacement
 * is taken as the sum of the GetPrevDispl() and GetDisplTimestep(),
 * which will be the current accumulated displacement during
 * N-R iterations. The default is false, and the (usual) current
 * displacement of the DOFs will be used.
 *
 * @author Per Ivar Bruheim
 */
void FrameElt::GetElasticForces(GVector &fi, GVector &fj,
   GVector &mi, GVector &mj, bool NRiter) const
{
  /// @todo Clean up this section.
  /// @bug Left moment has wrong sign.
```

Figure 5.2: Example of a documentation header for a function, in the Doxygen syntax. The inlined `todo` and `bug` tags are also demonstrated.

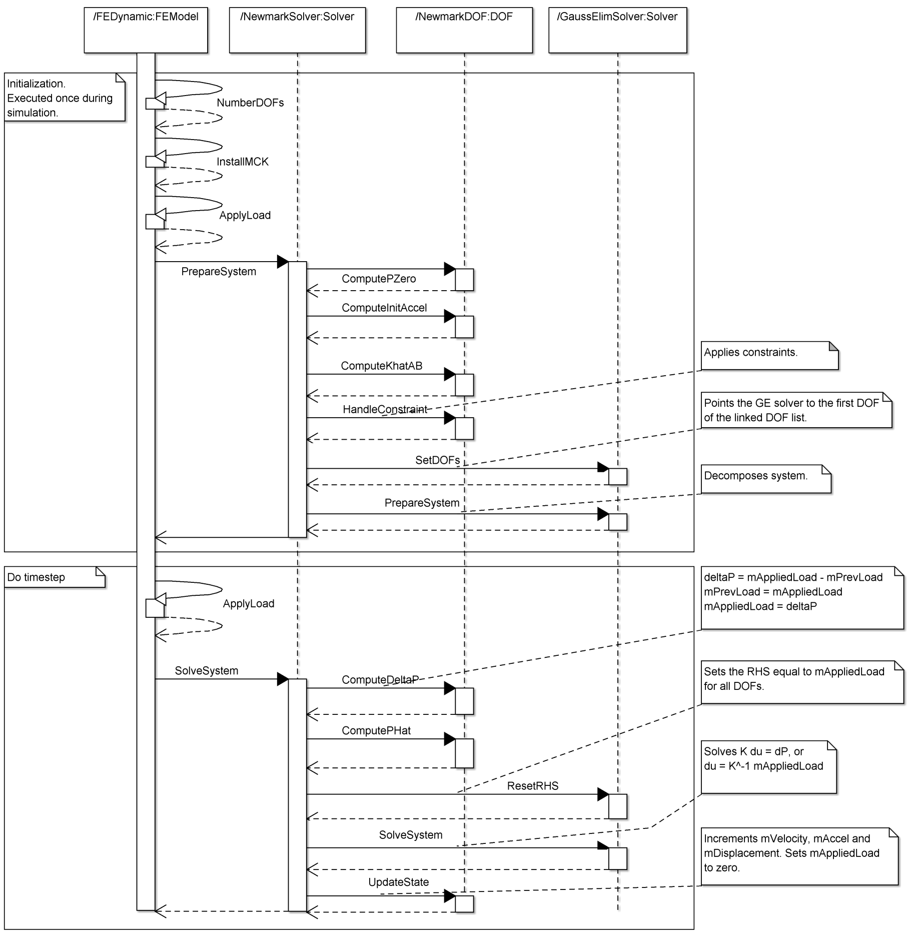


Figure 5.3: Newmark-β solution sequence in a linear analysis.

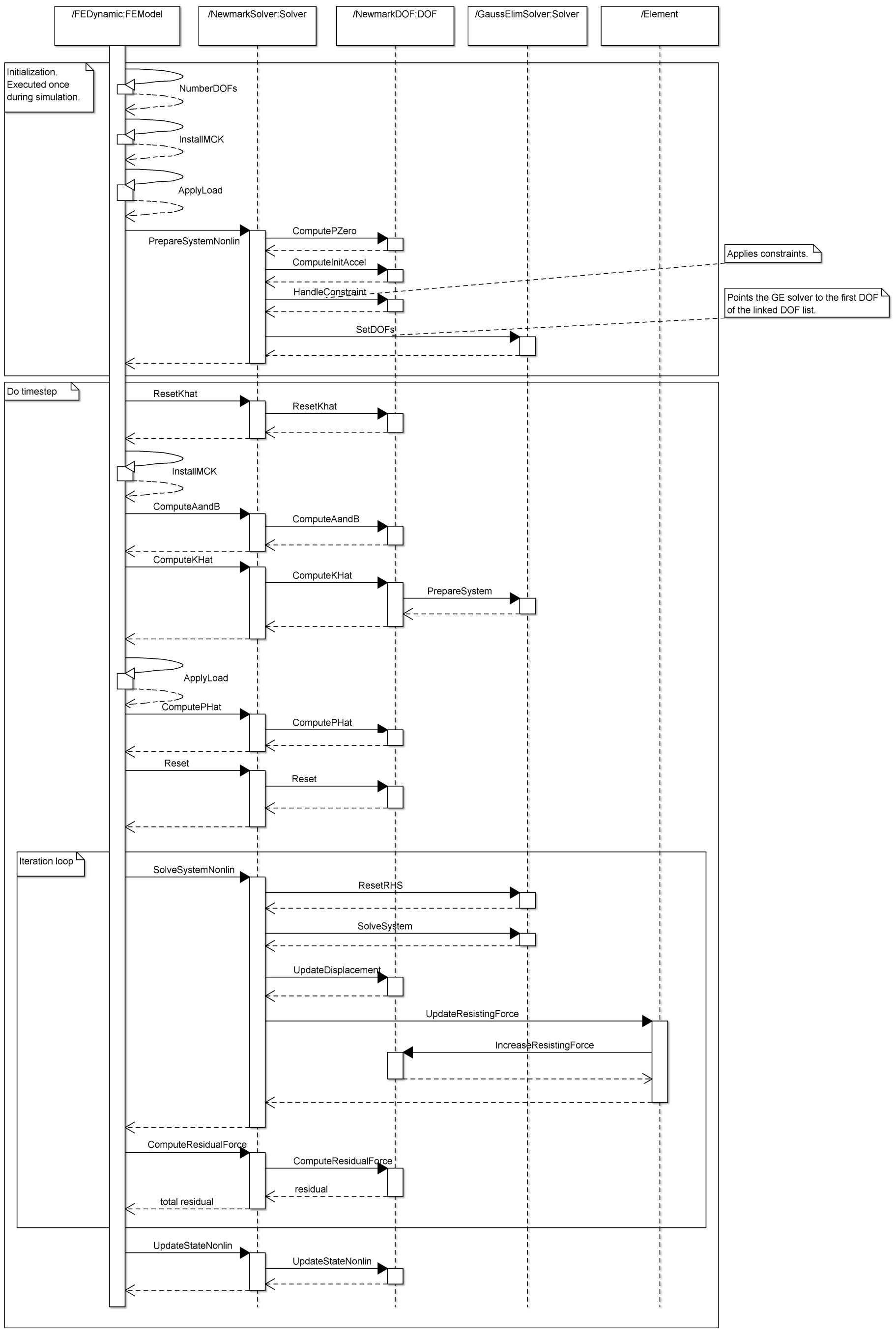


Figure 5.4: Newmark-β solution sequence in a non-linear analysis.

6 Validation of implementation

In the following, the present implementation of corotated beam elements is validated numerically by a set of test problems.

If not otherwise stated, the following convergence tolerances have been used during the Newton-Rapshon residual iterations:

$$\epsilon_{\text{load}} = 10^{-7} \tag{6.52}$$
$$\epsilon_{\text{displacement}} = 10^{-7} \tag{6.53}$$

6.1 Static problems

Here, the validity of the implementation will be investigated through a set of tests with pure static behaviour. This serves to reduce the set of potential errors down to:

- The extraction of deformations (4.2.1, 4.2.2)
- The computation of the tangent stiffness
- The time-stepping algorithm
- Handling of rotations

6.1.1 Cantilevered beam subjected to end force

A beam of length 100 m with its left end constrained against translations and rotations is subjected to a concentrated force of 20 kN on its right end in the lateral direction. This load does not change direction, and hence remains conservative, giving no load stiffness contribution. Further, the beam is modeled with 25 corotated beam elements, and the static response is computed by applying the load over 200 increments. Further, $E = 10^7\,\mathrm{Pa}$, $I = 1\,\mathrm{m}^4$ and $A = \sqrt{12}\,\mathrm{m}^2 = 3.464\,\mathrm{m}^2$. The full Newton-Raphson scheme is used with load and displacement tolerances of 10^{-5}, and the geometric stiffness is included. A reference solution was obtained with the ABAQUS[1] software using similar solver settings and mesh.

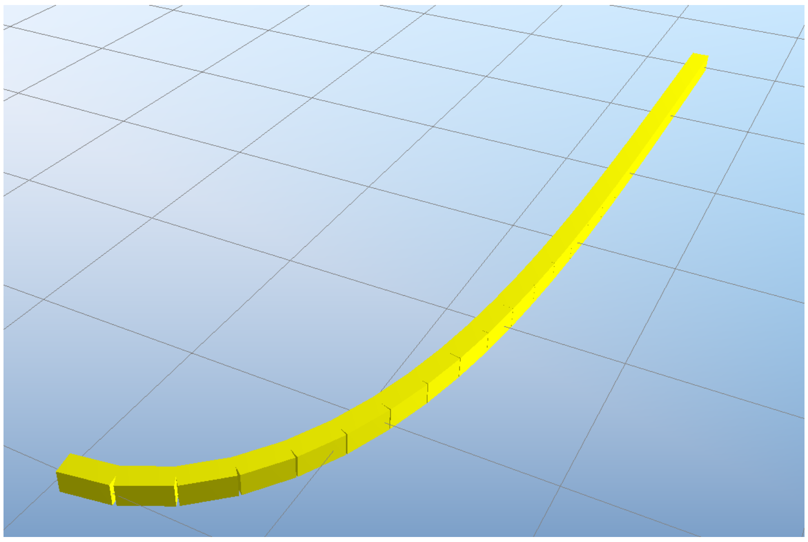


Figure 6.5: Beam with large deformations due to end load.

As seen in Figure 6.6, the load–displacement curve deviates from the linear relation when the displacements increase, as expected. The solution is in virtually perfect agreement with the reference solution, numerically with a maximum devation (per increment) of

$$\max\left\{\left|\frac{\Delta_{\text{present}} - \Delta_{\text{reference}}}{\Delta_{\text{present}}}\right|_n\right\} < 0.0045\,.$$

When computing the solution without the inclusion of the geometric stiffness, convergence problems were observed.

6.1.2 Buckling in axial compression

The critical buckling load, N_{cr} of a perfectly straight column with one end constrained against translations and rotations is given by the Euler load:

$$N_{cr} = \frac{\pi^2 EI}{4L^2}\,, \tag{6.54}$$

where L is the length of the column, and EI is the bending stiffness[28].

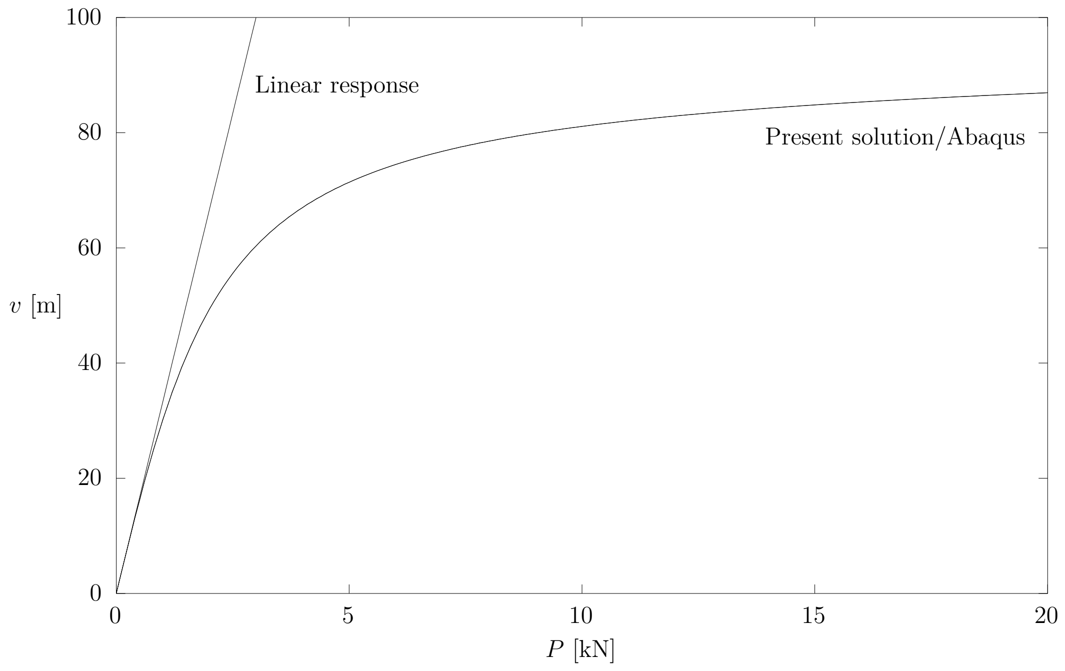


Figure 6.6: Lateral displacement of a 100 m long cantilevered beam subjected to a concentrated force on its free end.

The corresponding instability in the finite element formulation of the column occurs when the tangent stiffness becomes singular, that is

$$\det\left(\mathbf{K_m} + \mathbf{K_g}\right) = 0\,, \tag{6.55}$$

where the terms are the material and geometric stiffness matrices, respectively. The geometric stiffness matrix increases proportionally to the element's internal axial load, and the convention used here is that a compressive axial force has a negative sign.

Applying an increasing axial load to a beam element will indicate if the stiffness implementation is correct, for example if the sign of the axial force is correct. The same properties of the beam as in 6.1.1 is used. With these properties

$$N_{cr} = \frac{\pi^2 \cdot 1 \cdot 10^7 \cdot 31.4 \cdot 10^{-3}}{4 \cdot 10^2}\,\mathrm{N} = 7.75\,\mathrm{kN}\,.$$

The results indicate the the geometric stiffness is correct, with a buckling point very close to N_{cr}, even when using only one element.

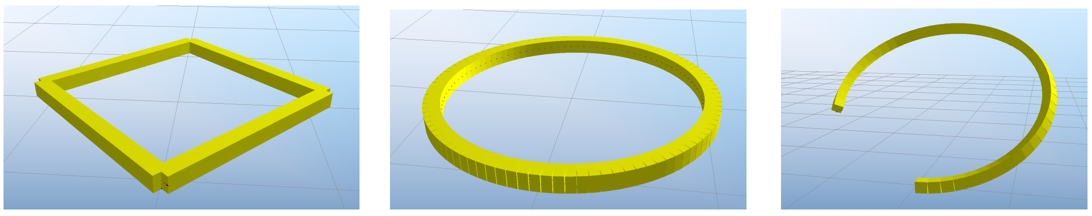

(a) Equilbrium, using 4 elements **(b)** Equilibrium, using 100 elements. **(c)** Load level 0.8, alternative direction of end-moment.

Figure 6.7: Beam after having been forced into a circular shape by an end-moment. Note in 6.7a that the curvature of the elements themselves is not shown graphically.

6.1.3 Cantilever subjected to end-moment

The cantilever, initially straight, is subjected to an end-moment that forces it to bend into a full circle. An analytic value of the required moment is given as [14]:

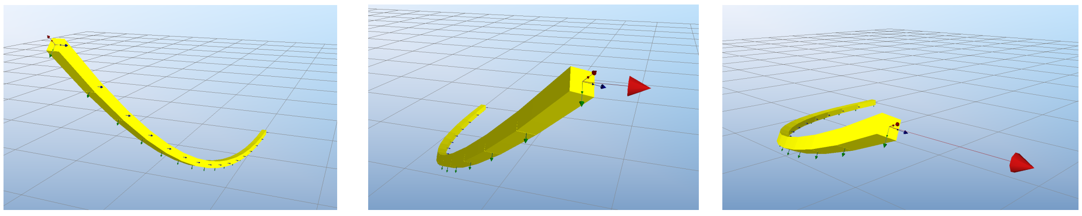
$$M = \frac{2\pi EI}{L}\,. \tag{6.56}$$

The implementation shows excellent results, with the final position of the beam-end being close to the other end, within a distance of $L \cdot 10^{-4}$. However, when the geometric stiffness was included, convergence was not obtained. Using different increment sizes shows that even applying the full load in one increment resulted in the correct solution. This seemed to be independent on the number of elements used. Both using a mesh with 4 elements and 400 elements produced the correct solution in one increment, although the last required over 1000 equilbrium iterations, with relative tolerances of $1 \cdot 10^{-7}$.

Further, applying the moment in arbitrary directions (still normal to the direction of the beam) also produced a correct solution, as illustrated in Figure 6.7c.

Discussion

These results strongly indicate the correct extraction of internal strains from the deformed position of the elements. Also, the procedures that accumulate rotations seem to be correct and invariant to the coordinate system. Finally, the correctly converged mesh of 4 elements demonstrates the remarkably good behaviour of the linear beam element even at relative end-rotations approaching 90°.



(a) Initial geometry **(b)** $P = 200$ **(c)** $P = 600$

Figure 6.8: Forty-five degree bend subjected to a lateral load (red arrow). The end furthest away is constrained against translations and rotations.

6.1.4 Forty-five degree bend

The following example demonstrates a full three-dimensional response, and results for comparison have been given several places[8,14]. A complete description of the problem is given by Crisfield (1990)[14].

As shown in Figure 6.8a, the beam geometry follows the path of a circle with radius $R = 100$, giving a $45°$ bend. A load with constant direction is applied to one end in eight equal load increments, up to the final load level of $P = 600$. The other end is constrained against translations and rotations, and a mesh of eight elements is used. The beam cross-section is set to unity, which gives $I_{ss} = I_{tt} = 1/12$ and $A = 1$, while the elasticity and shear moduli are $E = 1 \cdot 10^7$ and $G = E/2$, respectively. It seems that the cited sources give no clear definition of the torsional stiffness, I_{nn}, so it is here assumed that $I_{nn} = 2I_{ss} = 1/6$. Within each increment, the residual tolerance was set to $1 \cdot 10^{-6}$ both for the displacement and load.

The minimum mesh size that would converge in all increments when limiting the maximum number of iterations to $1 \cdot 10^3$ was found to be 15. Increasing the iteration limit to $1 \cdot 10^6$ did not improve this. Using different increment sizes seemed to make no difference, with no convergence achieved at around $P \approx 150$. With 15 elements, the lowest number of increments needed was 30. In Table 6.3, the position of the beam tip at three load levels is given, compared to solutions given by other authors.

6.2 Dynamic problems

In the previous section, the static behaviour of the model problems were analyzed, neglecting any damping and inertia forces. Here, further investigation will be made to the present implementation when inertia forces are included in the equations of

Table 6.3: Position of beam tip (x, y, z).

	Load level, P		
	300	450	600
Present	58.79, 40.19, 22.26	52.25, 48.50, 18.53	47.17, 53.48, 15.71
Bathe and Bolourchi[3]	59.2, 39.5, 22.5	—	47.2, 53.4, 15.9
Simo and Vu-Quoc[42]	58.84, 40.08, 22.33	52.32, 48.39, 18.62	47.23, 53.37, 15.79
Cardona and Geradin	58.64, 40.35, 22.14	52.11, 48.59, 18.38	47.04, 53.50, 15.55
Crisfield	58.53, 40.53, 22.16	51.93, 48.79, 18.43	46.84, 53.71, 15.61

motion. Damping, both structural and numerical, is disregarded in these problems.

6.2.1 Centripetal force

A body rotating at a constant rotational velocity about a rotation center will experience a centripetal force. The centripetal force acting on a point mass m rotating with a constant angular velocity ω along a path with radius r is given by:

$$F = mr\omega^2 \tag{6.57}$$

The rotating point mass is modeled using a single corotated beam element with lumped masses, which is fixed on one end. The element is slowly accelerated by applying an end-moment to one end, and the axial force N in the beam element is recorded. The results are given in Table 6.4, compared to (6.57). The values used are $m = 25.13\,\text{kg}$ and $r = 10\,\text{m}$. Loading the element slowly minimizes unwanted fluctuations due to bending deformations in the element. We see that the axial

Table 6.4: Axial force vs. angular velocity

ω	N	F
0.1109	3.063 kN	3.091 kN
0.3089	24.23 kN	23.98 kN
0.4503	51.00 kN	50.96 kN
0.8289	178.0 kN	172.7 kN

force matches reasonably well with 6.57. However, since the beam element is flexible, the increasing rotational velocity will give rise to axial deformation in the element. The discrepancies in the results are most likely caused by oscillations due to these fluctuations.

6.2.2 Energy conservation vs. element mass

In these problems, the internal (linear) strain energy and kinetic energy of the beam element is computed by the relations

$$E_u = \frac{1}{2}\left(\mathbf{f}^{\text{int}}\right)^{\text{T}}\mathbf{d} = \frac{1}{2}\mathbf{d}^{\text{T}}\mathbf{K}\mathbf{d} \tag{6.58a}$$

$$E_k = \frac{1}{2}\underbrace{\left(\mathbf{p}^{\text{int}}\right)}_{\text{Momentum}}^{\text{T}}\dot{\mathbf{d}} = \frac{1}{2}\dot{\mathbf{d}}^{\text{T}}\mathbf{M}\dot{\mathbf{d}} \tag{6.58b}$$

$$\tag{6.58c}$$

where **K** is the linear stiffness, **M** either the lumped or consistent mass **d** the nodal displacement vector, while $\mathbf{f}^{\text{int}}$ and $\mathbf{p}^{\text{int}}$ are the internal elastic force and internal momentum vectors, respectively.

As no explicit expression of the internal strain energy can be given when the geometric stiffness is included, it is disregarded in the following examples. The total strain and kinetic energy of the model is found by an element-wise summation. Physically speaking, if the energy flux of the system is zero, i.e. when there are no external loads or dissipation due to damping, the following should hold true:

$$\mathbf{E}_{\text{tot}} = \sum_{i=1}^{n_{\text{elt}}}\left(\mathbf{E}_u + \mathbf{K}_k\right)_i = \text{const.} \tag{6.59}$$

where n_{elt} are the total number of elements. As the external load is constant, the work done on the system is simply

$$W_P = P\Delta \quad \text{where } \Delta \text{ is the lateral displacement of free end.} \tag{6.60}$$

A single beam element of length 10 is constrained against rotations and translations in one end, while the other end is free. On the free end, a lateral load of constant direction and magnitude $P = 1 \cdot 10^3$ is acting while $t \leq 1$, after which the load is removed and the element is left in an in-plane free-vibration motion. Further, $E = 1E7$, $\rho = 1E2$, $I = 1/12$, and $A = 1$, while the time-step size is set to $\Delta t = 0.01$.

Discussion

As can be seen in Figure 6.10, the total energy is correctly conserved with lumped masses after the external force is removed. Computing the external work by (6.60)) verifies that the total energy is equal to work done by the applied force. However,

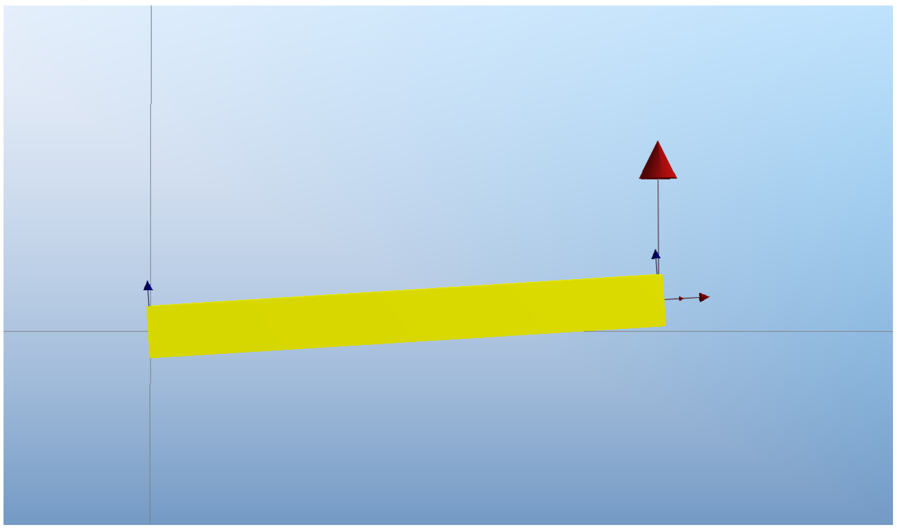


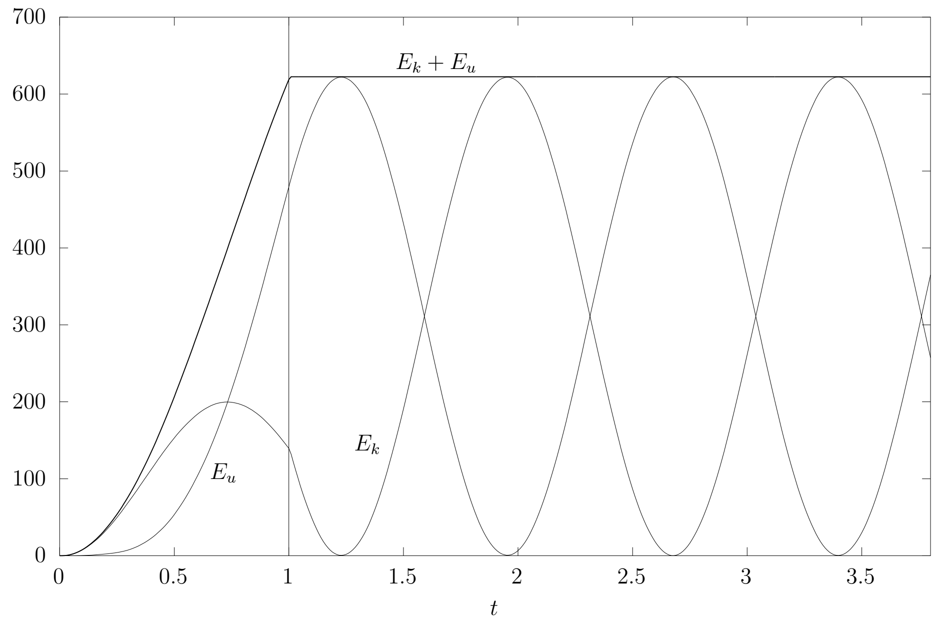
Figure 6.9: Displaced element at $t = 1$ due to lateral load of $P = 1 \cdot 10^3$ (red arrow). Note that the rotation of the constrained left end is actually zero, but this is not showed graphically.

when using consistent masses, the element is observed to be gaining and losing some energy during the simulation. Additionally, when even larger rotations are allowed, the total energy is found to vary more significantly—and at some point the beam is even observed to be turning around and going backwards. This issue might be attributed to the part of the code dealing with the non-linear geometry, as further tests show that when using a linear element, the energy is exactly conserved with consistent masses.

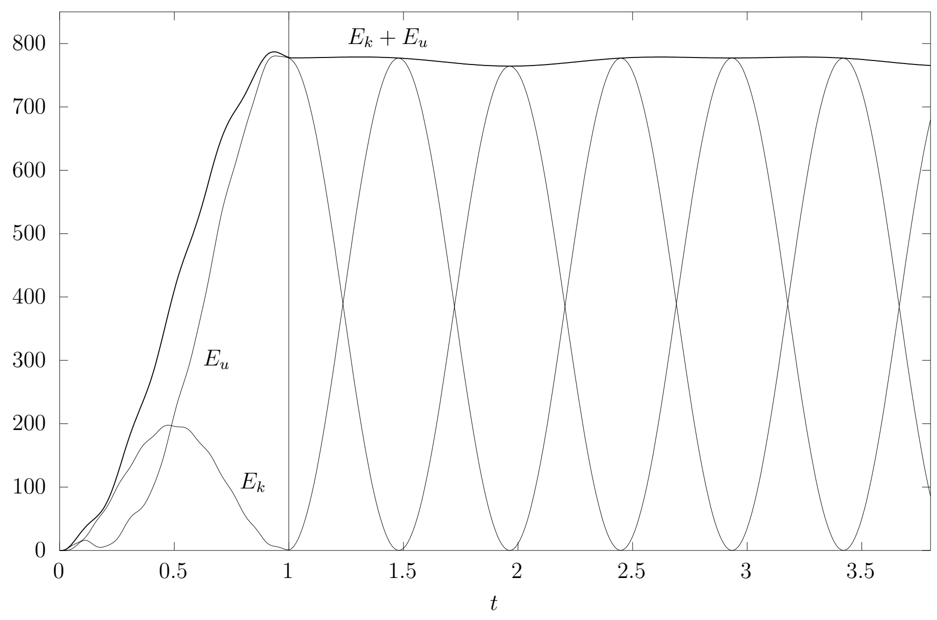
Another observation made is the shorter period with consistent masses. As expected, the lumped masses will lead to an element appearing heavier than with consistent masses[11].

6.2.3 Complex problem

The analysis of a large model of a wind turbine subjected to large rotations and deformations is presented in 7.5, with comparison to other codes. This model serves to demonstrate the full three-dimensional behaviour in a dynamic analysis.



(a) Using lumped masses.



(b) Using consistent masses.

Figure 6.10: Strain, kinetic and total energy of a single consistent-mass element subjected to a constant lateral force while $t \leq 1$.

6.3 Discussion

The static tests in this chapter have demonstrated that the corotated element produces results very close to other authors. The implementation was shown to be coordinate-invariant. Application of an axial force to a beam showed that the implementation of the geometric stiffness was correct. Simple dynamic problems demonstrate a correct centripetal force for a rotating element.

Still, some limitations are present. Consistent element masses lead to erronous results, where elements pick up energy. However, lumped masses produce good results as the mesh is refined.

The application of external nodal moments seems to give problems in combination with the linear geometric stiffness, most likely due to the simplifications made to the tangent stiffness.

Although not closely investigated, it is added that the observed speed of the code was approximately 2.3 times as fast as the real time of the simulation when the large model of 6.2.3 was simulated (with an Intel i5-2500K 3.30 GHz CPU). Further, the number of equilbrium iterations per increment was found to be around 7 (with tolerances of 10^{-6}).

7 Application to aeroelastic analysis of wind turbines

7.1 Introduction

The field of aeroelasticity studies the interaction between elastic and inertia forces of a structure and external aerodynamic forces. In the analysis and design of wind turbines, the understanding of such interactions is of great importance. As of today, many software tools exist that accommodate such analyses, yet their underlying methods are very diverse. Due to modern wind turbines trending towards increased size and slenderness, and hence flexibility, large-deflection effects play an increasingly important role in the overall response and should be captured in modern tools.

In this chapter, the different methods and techniques used in wind turbine codes are given a brief comparison in an attempt to expose their advantages and disvantages. Further, the present implementation is utilized for such analyses where the objective is to establish the validity of its results. The large-deflection analysis is performed with the finite element method by the use of corotated beam elements, as detailed in previous chapters.

A more general treatment of wind turbine design is outside the scope of this thesis, and can be found elsewhere[7,17,30].

7.2 Techniques for analysis of wind turbines

Different approaches to the aeroelastic analysis of wind turbines exist, each having their strength and weaknesses and areas of application. The basic design of a turbine can be accomplished by some crude method, where the primary objective is to predict long-term effects, e.g. fatigue damage and power production. Establishing the optimized design, however, requires a much more detailed and complex model of the underlying physics. Thus, different approaches are suited to different aspects of the design process. References in this section include Rasmussen et al. (2003) and Cordle (2010)[12, 36].

The commonly used methods can be classified as either time-domain based or frequency-domain based. A time-domain method produces a time-series of the response from a given time-series of external loading, and allows for the inclusion of transient events . As an example, a wind turbine's control system will give rise to transient effects that are important to capture in an analysis.

A frequency-domain method typically computes a spectrum of the response given the spectrum of the loading. However, such methods have important limitations that make them unsuitable as a general-purpose technique for wind turbine analysis namely that nonlinear effects and transient events can not be included[35].

7.2.1 Modal approaches

In a conventional modal method, a combination of the free-vibration shapes of the discretized structure is used to express the spatial displacements of the structure. A displacement in *all* spatial degrees-of-freedom (DOFs) can effectively be described with sufficient accuracy by a smaller set of modes, acting as generalized DOFs. As these mode shapes are orthogonal with respect to the mass and stiffness properties of the system, a linear multiple-degree-of-freedom (MDOF) system of equations (in time) can be converted into an uncoupled set of single-degree-of-freedom (SDOF) equations, which greatly reduces the computational effort required in the solution process. A modal method is, however, only applicable to a linear system, where the superposition principle holds and the mode shapes remain unchanged with time. (It is possible to move nonlinear terms to the right-hand side of the equation, but the method will no longer resemble the conventional modal technique[11, p. 398].)

Malcolm (2002) establishes the modes of a simple five-DOF model of a horizontal-axis wind turbine (HAWT), operating at an arbitrary rotational speed. The use of the Coleman transformation yields a set of equations free of the periodic terms arising due to the rotation of the blades. By incorporating the mode shapes of the stationary turbine, the complex operating modes of the system are obtained. The solution is further applied to both the aerodynamic loading and the displacement response[29]. Such a procedure is restricted to linear analysis.

Of the wind turbine simulation codes available, many use a modal method. As mentioned, the method is attractive due to its speed, but it is limited with respect to capturing large deflections (i.e. nonlinear behavior).

7.2.2 Finite element method

In a finite element approach, the structure is discretized into a mesh of elements, each having their properties (i.e. stiffness, damping, mass). Different types of elements, e.g. plate and shell elements, can be connected, allowing for a great flexibility in modeling. Due to the possibly large number of degrees-of-freedom in a complex model, the finite element method is computationally more expensive than a modal method. Techniques exist for reducing the number of the degrees-of-freedom[27]. Nonlinear geometry can be included by finite strain measures or the corotated formulation, as detailed in previous chapters.

7.2.3 Multibody systems

The multibody system approach divides the structure into either rigid or flexible elements, and allows for large translations and rotations. The joints—i.e. the connection between elements—can be configured to constrain the elements in arbitrary directions or types of relative motion. With this approach, the equation of motion is generally formulated in a nonincremental manner[40], as is not the case in the finite element method.

7.2.4 Discussion

Mainly three methods are present in existing codes, with the modal method being unable to capture nonlinearities due to large deflections. The finite element method with corotated elements provides the most flexibility in modeling. An inherent feature of multibody systems is the capability for modeling arbitrary joints between elements. In closing, it is mentioned that there is ongoing research on the integration of finite element and multibody systems methods in engineering applications, via *gluing algorithms*[40].

7.3 The effects of large blade deflections

Today's wind turbine blades are long and slender structures that can be subjected to moderate deflections, and aeroelastic effects resulting from the interaction of the aerodynamic forces and the structural deflections will be important to capture in an analysis. Additionally, there is ongoing research on new blade designs, e.g. pre-bent or swept blades, where large deflection effects are of great importance.

The large deflection of blades gives rise to several physical phenomena, mainly three[36]:

- Reduced effective rotor diameter and cone angle, leading to a lower power production than estimated with linear computations.
- Structural coupling between edgewise and torsional motions, affecting aerodynamic damping and pitching moments at the blade root.
- Increased flapwise stiffness due to geometric nonlinearities.

In a paper, Kallesøe, considering only the steady-state flapwise deflection of the blades (similar to Sec. 7.5), shows that the edgewise aerodynamic damping can be decreased by half at worst. This negative damping effect occurs due to a geometrically nonlinear coupling between the edgewise and torsional blade motion when the blade is subjected to a flapwise deflection. In turn, this coupling affects the angle of attack as the blade twists, and hence the loads on the blade, resulting in a lowered aerodynamic damping for the edgewise motion[25]. As a reduction of damping lowers the stability, such effects are important to capture.

7.4 Modeling the OC4 reference turbine

The wind turbine model considered here is defined in papers by Jonkman et al. and Vorpahl et al. [24, 49], as a part of the OC4 (Offshore Code Comparison Collaboration Continuation) project. In this project, results by several research teams and available computational codes for wind turbines are compared. The load cases (e.g. parameters defining the wind conditions) are specified in [48].

Here, the OC4 reference turbine is modeled using the functionality of the framework and the corotational element implementation described in previous chapters. A geometrically nonlinear behaviour is allowed for the parts of the structure that are subjected to large displacements and rotations, i.e. the rotor with blades and the shaft. All parts are modeled with beam elements, with a total of 249 nodes and 299 elements. All elements have an axi-symmetric cross-section, except the blades, which have a distinct weak and strong axis. Parts whose properties are not explicitly defined in the references, such as the shaft–tower connection etc., are modeled with a high stiffness and negligible mass.

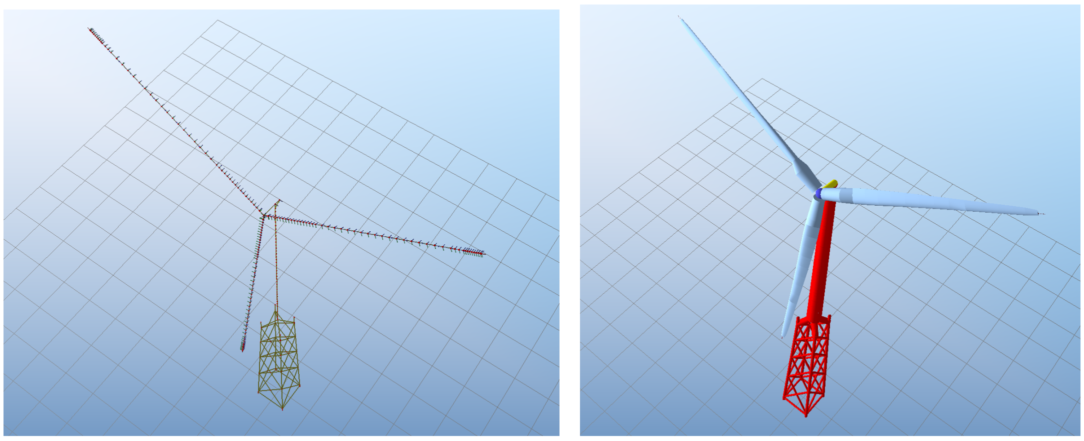

(a) Element model. **(b)** Graphical view.

Figure 7.11: The finite element model of the full structure. The arrows on the blades in (a) are the elements' principal axes.

The main goal of this section is to obtain results of the dynamic response that demonstrate the validity of the present implementation.

7.4.1 Model overview

7.4.1.1 Support structure

The support structure consists of a bottom truss structure of height 50 m (a jacket), followed by a circular tower of height 88.15 m, giving a total height of 138.15 m.

7.4.1.2 Shaft and bearings

The shaft transfers forces and moments from the rotor to the tower, but is free to rotate about its axis due to it being connected to a set of bearings. The bearings are modeled by having no connection of the rotational degrees-of-freedom of the shaft with the tower (node 1 and 2 in Figure 7.12). A triangle-shaped structure transfers the overturning moments to the tower, through two stiff elements (1–T and 2–T). A tilt angle of 5° from the horizontal direction is used.

It is noted that the element 2–H is the actual shaft of approx. 5 m, as defined in Jonkman et al.

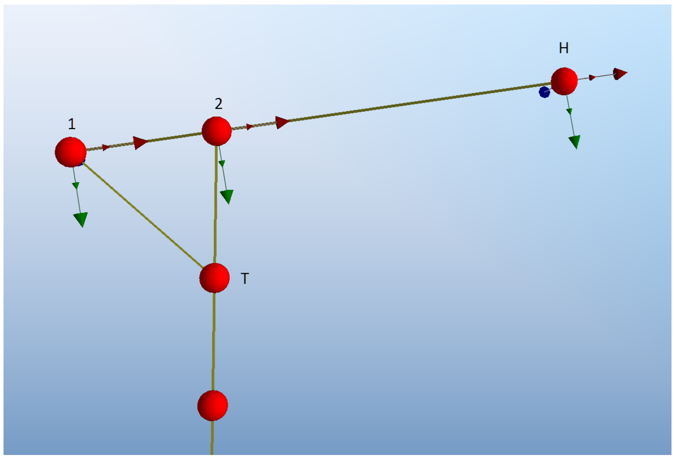


Figure 7.12: The shaft (1–2–H) and its connection to the tower as seen from the side. 1–2–H is free to rotate about all axes in connections 1 and 2, while the connection to the tower (T) is moment-stiff. H is the hub center, i.e. the center of the rotor.

7.4.1.3 Rotor and blades

The rotor consists of three equal blades of length 61.5 m, each with a mass of 17,740 kg. The root of the blade is located 1.5 m from the hub node along the blade pitch axis. The hub–blade connection is modeled with beam elements having a high stiffness and negligible translational mass, while a rotational inertia of 115,926 kg m^2 about the shaft direction is added to the hub node. All blades have an upwind pre-cone of 2.5°, i.e. the blades are directed slightly out of the rotor plane. The distributed structural properties of the blades are found in the previous references. The beam elements in the blade are modeled with corotated elements, allowing them to undergo large displacements and rotations.

Separate from the structural element mesh, a mesh of aerodynamic blade elements is defined along the blade. Each of these elements contains information about the aerodynamic coefficients, the amount of aerodynamic twist, and the chord length at the given distance from the blade root, and are input to the BEM computations. The distributed aerodynamic properties of the blade are found in the previous references.

7.4.1.4 Loads

The turbine is subjected to a non-turbulent homogeneous wind field of constant speed $V_{\text{hub}} = 8\,\text{m/s}$ in the horizontal direction, corresponding to load case 3.2 in [48]. There is no yaw misalignment. All elements are subjected to gravity forces.

The blade element momentum method (BEM) is used to compute the aerodynamic forces on the blade during the simulation. These forces are computed in terms of the relative velocity seen by the blade, with the velocity of the blade itself being included. A tower-shadow correction (by the use of potential flow) to the aerodynamic loads is effective when a blade is close to the tower. Further, the generator applies a torque to the main shaft and an opposite moment to the tower connection (node 2 and T in Figure 7.12, respectively), as described in the next section.

7.4.1.5 Controller and generator

The turbine operates in a variable-speed, variable pitch-to-feather configuration. A variable-speed turbine is allowed to rotate at varying rotational speeds, where instead the generator controls the torque it applies to the shaft. The objective is to maximize the power output when it is below the rated power (power, torque and rotational speed is related by $P = T\omega$). Above the rated power (due to high wind speeds), a pitch controller responds by pitching the blades—effectively altering the angle of attack and hence the aerodynamic loads.

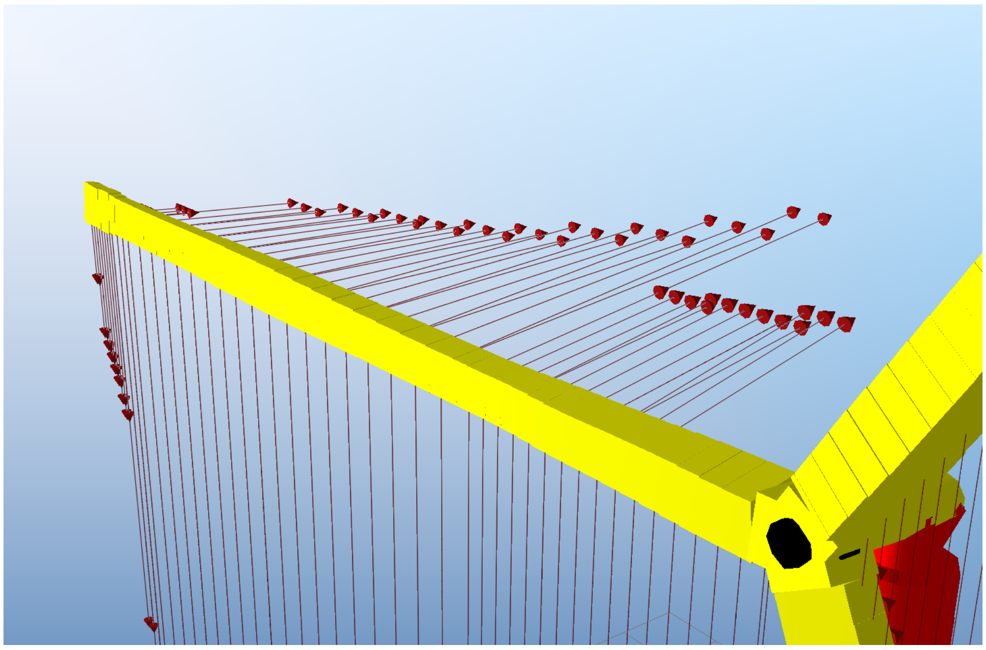
While a pitch-to-feather system *reduces* the angle of attack, an active stall system would reduce the loads by pitching in the opposite direction, so that the angle of attack *increases* and progresses into the stalled region. Such a setup will not be considered here. Additionally, due to the low wind speed in the present load case, the pitch controller will not be activated during the simulation. The controller implementation follows the specifications in [24].

7.4.2 Directions in the deformed state

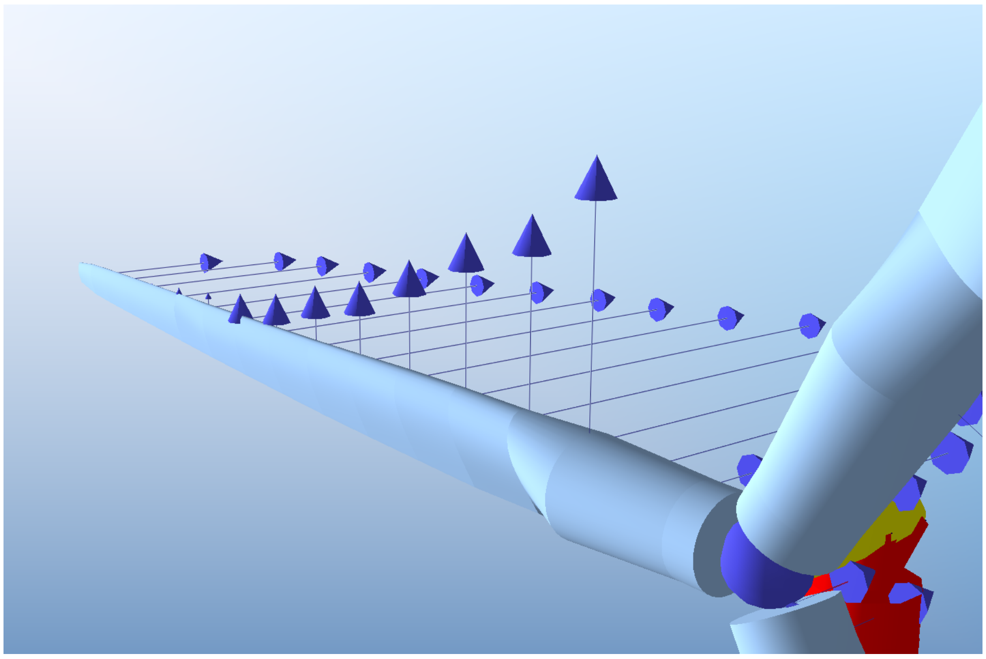
Due to the structure being allowed to undergo large arbitrary rotations and translation in space, directions that are assumed constant in a small-displacement analysis must be computed in the deformed configuration of the finite element mesh. As an example: A yawing motion of the rotor would alter the rotor plane, which e.g. is a necessary plane of reference for values such as the out-of-plane blade deflection. Here, a consistent way of extracting these from the finite element configuration is devised. All vectors are referred to the global inertial coordinate system.

The **rotor plane normal** is taken as the deformed direction vector of the shaft, further rotated by the deformational rotations of the shaft at the node connected to the hub:

$$\mathbf{n}_{\text{rotor}} = \mathbf{R}_d^{\text{shaft element}} \, \mathbf{n}_{\text{shaft}} \, ,$$



(a) Beam elements, illustrating the proportions of the flapwise to the edgewise stiffness. Red arrows are gravity (down) and aerodynamic loads.



(b) Graphical blades with thrust and tangential aerodynamic forces. One load arrow corresponds to one aerodynamic blade element.

Figure 7.13: Layouts of blade meshes.

while the rotor plane is further determined by the location of the hub node. This definition allows for arbitrary yawing and tilting motions, while still having a reference plane, for e.g. the out-of-plane blade deflections. Although the rotational deformations of the shaft in general remain small, they are included to avoid loss of generality, and due to the fact that even small rotations can give moderate changes in the position of the tip of the blades.

The **blade direction** is taken as the deformed direction of the element connecting the blade-root node to the hub node,

$$\mathbf{n}_{\text{blade}} = \mathbf{n}_{\text{hub–blade element}} \,,$$

which gives the deformed direction independent on large yaw and tilt, as well as the rotation about the shaft of the blades themselves. The line determined by this axis and the position of the hub node can be used for e.g. computing the blade in-plane deflection.

7.5 Simulation results and comparisons

Time simulations of the full finite element model described in Section 7.4 is here carried out where large-displacements are allowed by using corotated elements for the blades, hub and shaft.

The settings of the nonlinear solver used in these simulations are summarized in Table 7.5.

Table 7.5: Simulation settings

Δt	$0.05\,\mathrm{s}$
α	-0.025
ϵ	10^{-7}
Mass	Lumped

7.5.1 Startup simulation

Initially, at $t = 0\,\mathrm{s}$, the turbine is at stand-still, with zero pitch, and zero generator torque. The aerodynamic forces are sufficient to start the turbine, and at approximately $t = 77\,\mathrm{s}$, at a rotational speed of $6.4\,\mathrm{rounds/min}$, the controller commands the generator to apply the counteracting torque. As seen in Figure 7.14, a transient response is observed, where the acceleration of the rotor is arrested by the

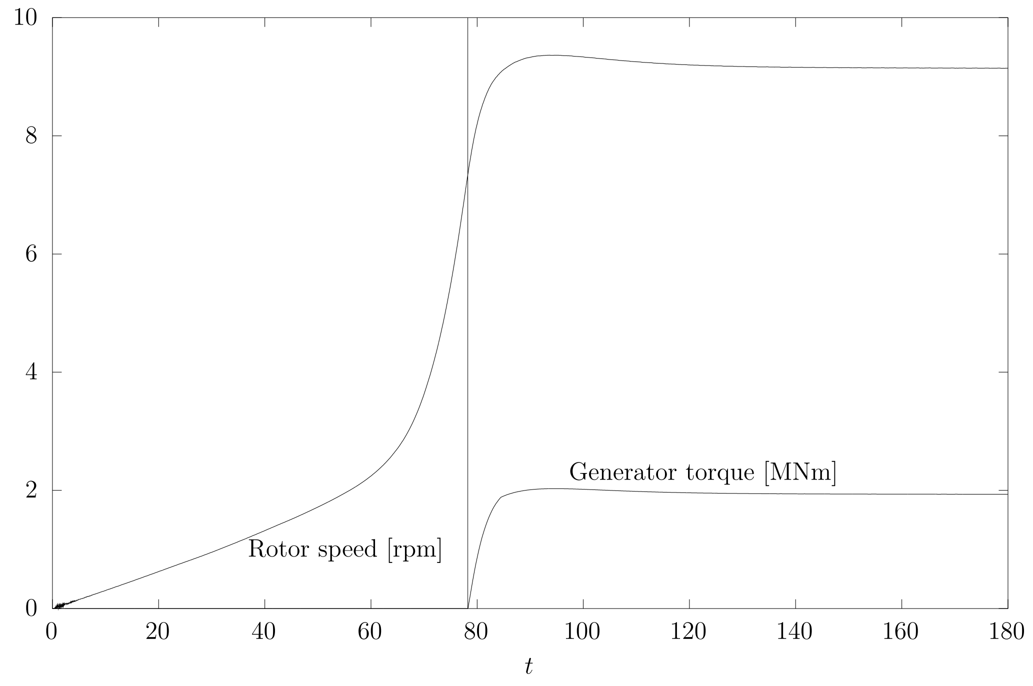


Figure 7.14: Rotor speed and applied generator torque from start-up to the steady state.

counteracting torque, and then slightly decelerated, until a steady-state behaviour is adopted at $t \approx 150\,\mathrm{s}$.

As the rotor speed increases, the thrust force increases, causing the blades to bend out of the rotor plane, as can be seen in Figure 7.15. Initially, some fluctuation is seen in the tip deflection—this is caused by the sudden application of the aerodynamic loads on the stand-still turbine. However, this is damped out by a combination of the aerodynamic and algorithmic damping as the simulation progresses.

In the steady state, the thrust and hence the tip deflection experiences periodic fluctuations. This occurs each time the blade passes the tower, when the thrust is reduced due to the tower shadow effect.

7.5.2 Comparison with other codes

Time histories of the steady-state behaviour (Figure 7.15) for other codes have been submitted to the OC4 project, and serves as a good basis for investigating the present results. A total of eight other codes are compared to the present results, including the following:

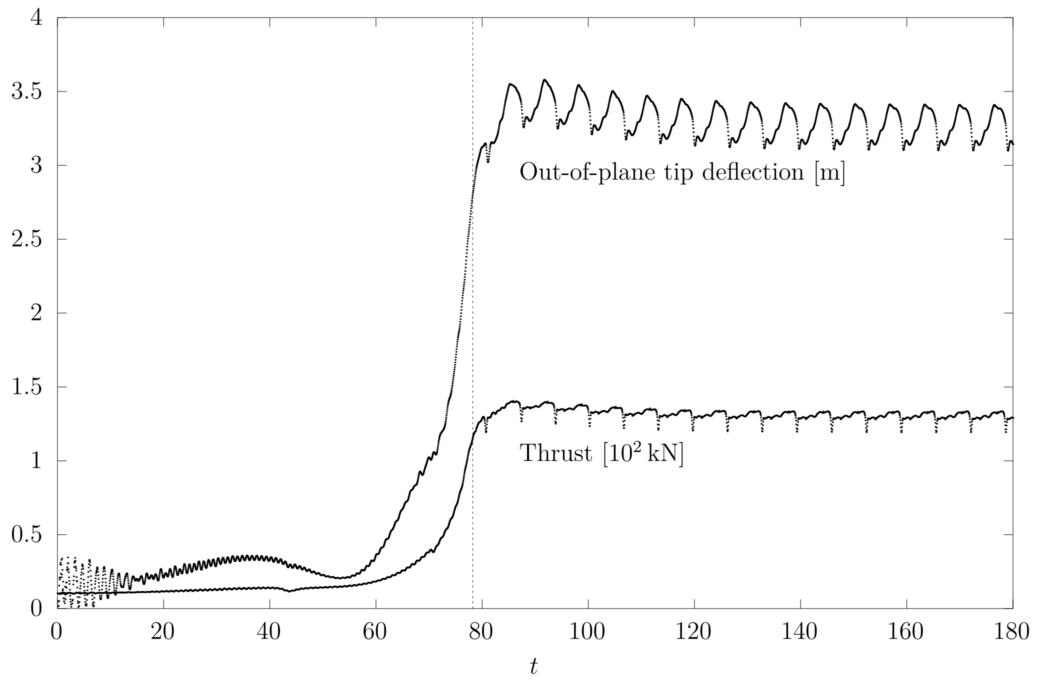


Figure 7.15: Out-of-plane tip deflection and total thrust force on one blade from start-up to the steady state. The vertical line marks the application of the generator torque.

1. FAST-ANSYS
2. USFOS-vpOne
3. FEDEM Windpower
4. GAST
5. Bladed V4
6. Flex5
7. HAWC2
8. ADCoS-Offshore

Here, the steady state response is studied, by considering the 25 second time-span $t \in [150\,\mathrm{s},\ 175\,\mathrm{s}]$ in the simulation of 7.5.1. To avoid clutter, the figures do not explicitly label the graphs of each code—this information can be found in the references given previously. The resulting out-of-plane tip deflection and tip twist of one blade is shown in Figure 7.16 and Figure 7.17.

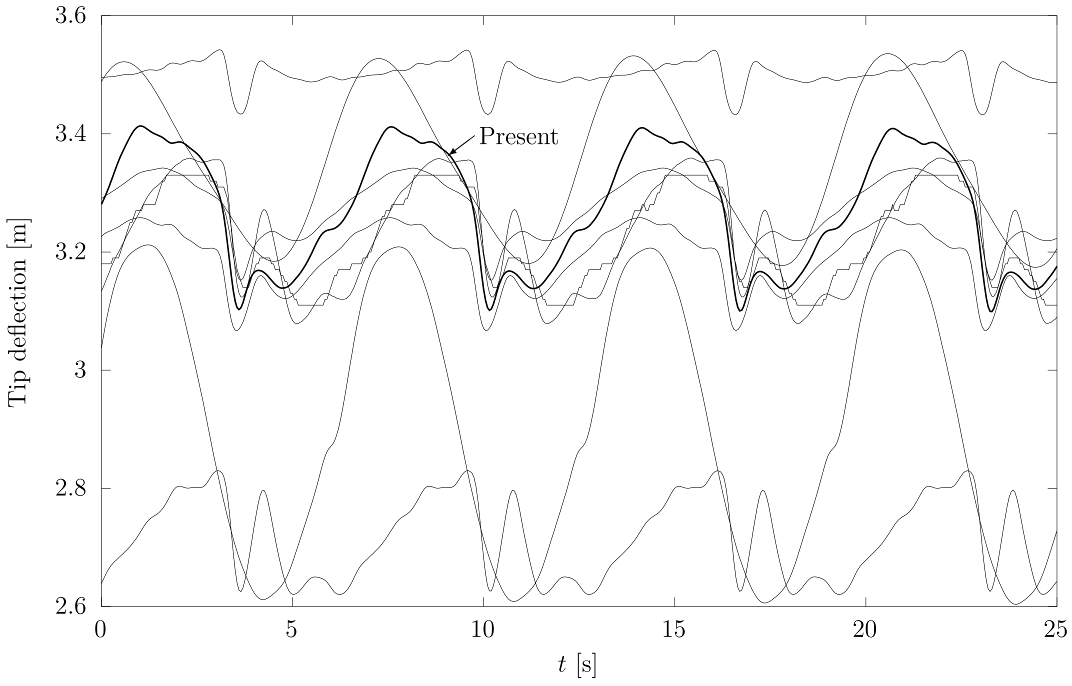


Figure 7.16: Out-of-plane tip deflection in the steady state, compared to other codes. The present results are the highlighted curve.

7.5.3 Remarks

The results obtained with the present implementation are in good agreement with other codes. Differences observed are attributed to differences in the model input or the aerodynamic (i.e. BEM) computations, and not to the corotational finite element implementation itself. It is observed that only four codes include the blade twisting, and none of these are based on a modal method, but rather the finite element method or multibody systems.

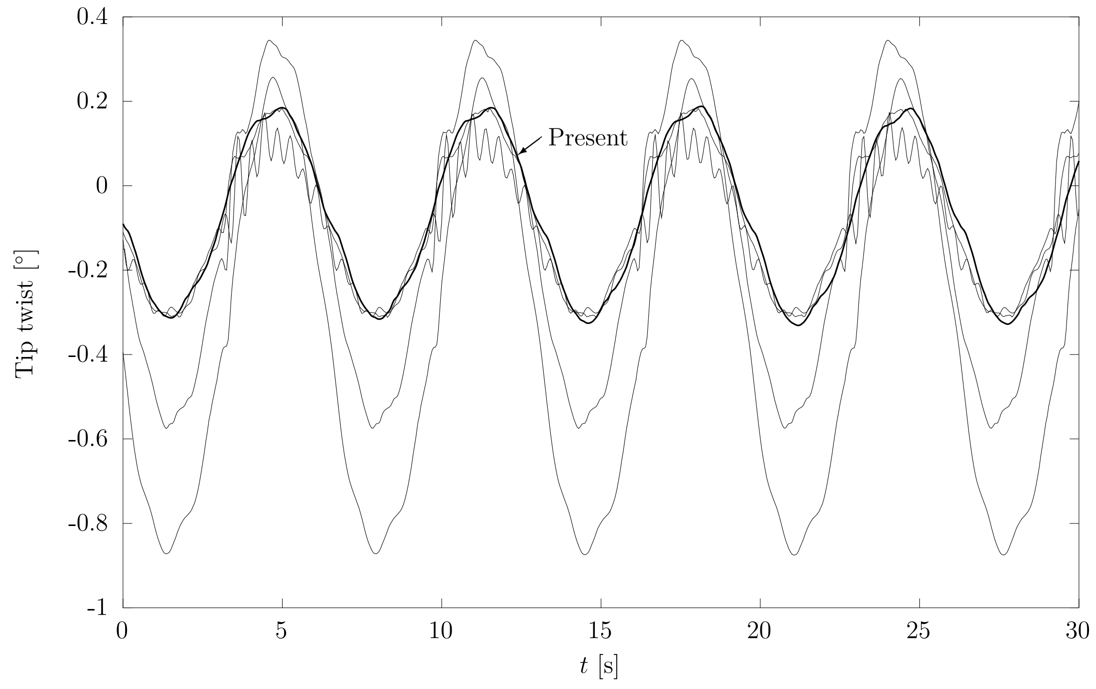


Figure 7.17: Tip twist.

Figure 7.18: Tip twist in the steady state, compared to other codes. The present results are the highlighted curve.

8 Conclusion

8.1 Summary of thesis

A formulation has been established that enables the large deformation (i.e. geometrically nonlinear) analysis of structures modeled with finite elements. The theory and procedures of the corotational formulation in the context of three-dimensional finite beam elements was established, where the necessary treatment of rotations in space was given special care. A commonly used algorithm for the solution of the nonlinear dynamic equation of motion was presented, namely Newmark-β with Newton-Raphson iterations.

A previously developed finite element framework based on object-oriented principles was extended with new functionality supporting the corotional formulation. The interaction of the existing framework source code and the corotational procedures was demonstrated. The thorough benchmarking and validation of the implementation produced excellent results compared to other authors, although some limitations of the implementation was discovered.

A discussion of different methods for the aeroelastic analysis of wind turbines, including their major differences, was given. Further, due to current trend in wind turbine designs, the need for aeroelastic codes that capture large blade deflections was identified. With the use of the existing BEM code, time simulations of a full model of the OC4 reference turbine were carried out, where the goal was to study the behaviour of the rotating flexible blades modeled with corotated beam elements.

The resulting responses were in good agreement with results produced by other codes, particularly those with a similar underlying algorithm, and discrepancies were most likely due to differences in the model and the computations of the aerodynamic loads and not the corotational implemention itself.

8.2 Conclusion

The conclusions of this thesis were mainly that:

- The corotational formulation is an attractive approach for allowing nonlinear geometry in small-strain applications.
- Geometrically nonlinear analysis in 3D requires non-trivial treatments of rotations utilizing tensors or quaternions.
- Corotated beam elements are well-suited to the analysis of wind turbines, while modal methods are unable to capture large deflections.
- Less flexibility in modeling is achieved with the multibody systems approach compared to the corotated formulation, however multibody systems are more flexible in defining joints.
- The present implementation was successful, and produced good results in comparison with other codes for wind turbine analysis, although further benchmarking should be performed.

8.3 Suggestions for further development

Further work on the software developed here should include various improvements. Improving the efficiency of simulations is suggested, and would call for research on e.g.: (1) Adaptive time-step algorithms and their implications to the BEM computations in varying wind conditions, (2) substructuring/localized nonlinearity[10, p 332] (for instance, if the tower is known in advance to behave linearly, simplifications in the solution process can be made), as well as (3) other reduction techniques[27]. As the analyst often would be interested in running a large set of simulations in one single batch, as fast as possible, such improvements are obviously important. However, as pointed out earlier, in the present state the code runs over twice as fast as the real time of the simulation, easily allowing for a *real-time* user experience.

Further work should include structural damping with the corotated elements, and ensure correct behaviour (which is known to be problematic[19]). The application of BEM loads to the finite element mesh also requires further work. As of now, the BEM loads do not account for the nodal velocities in a correct manner, as velocities are not incremented during equilbrium iterations—and in a true aeroelastic analysis, the BEM loads would depend on the structural velocities. Problems with consistent masses observed in Chapter should also be investigated. Finally, the internal Coriolis force of the element has not been discussed, (and rather seems to be neglected in the present implementation), and could be of importance when a coarse mesh is used.

References

[1] *ABAQUS/Standard user's manual.* Hibbitt, Karlsson & Sorensen, 2001.

[2] John Argyris. An excursion into large rotations. *Computer Methods in Applied Mechanics and Engineering*, 32(1–3):85–155, September 1982.

[3] K-J. Bathe and S. Bolourchi. Large displacement analysis of three-dimensional beam structures. *Numerical methods in engineering*, 14:961–986, 1979.

[4] Ted Belytschko, Wing Kam Liu, and Brian Moran. *Nonlinear Finite Elements for Continua and Structures.* Wiley, 1 edition, September 2000.

[5] Pål G. Bergan, Per Kr. Larsen, and Egil Mollestad. *Svingning av konstruksjoner.* Tapir forlag, second edition, 1986.

[6] Per Ivar Bruheim. Implementation of a modified inverse iteration algorithm in an object-oriented finite element framework. Technical report, NTNU, Trondheim, Norway, 2011.

[7] Tony Burton, Nick Jenkins, David Sharpe, and Ervin Bossanyi. *Wind Energy Handbook.* Wiley, 2 edition, June 2011.

[8] A. Cardona and M. Geradin. Beam finite element non-linear theory with finite rotations. *International Journal for Numerical Methods in Engineering*, 26(11):2403–2438, 1988.

[9] Anil K. Chopra. *Dynamics of Structures.* Prentice Hall, 3 edition, September 2006.

[10] Ray W. Clough and Joseph Penzien. *Dynamics of structures.* McGraw-Hill, second edition edition, 1993.

[11] Robert D. Cook, David S. Malkus, Michael E. Plesha, and Robert J. Witt. *Concepts and applications of finite element analysis.* John Wiley & Sons, Ltd, fourth edition edition, 2001.

[12] Andrew Cordle. State-of-the-art in design tools for floating offshore wind tur-bines. Technical report, Project UpWind, 2010.

[13] Evangelos A. Coutsias and Louis Romero. The quaternions with an application to rigid body dynamics. Technical report, University of New Mexico, Albuquerque, 1999.

[14] M. A Crisfield. A consistent co-rotational formulation for non-linear, three-dimensional, beam-elements. *Computer Methods in Applied Mechanics and Engineering*, 81(2):131–150, August 1990.

[15] M. A. Crisfield, U. Galvanetto, and G. Jelenić. Dynamics of 3-D co-rotational beams. *Computational Mechanics*, 20(6):507–519, November 1997.

[16] M. A Crisfield and J. Shi. A co-rotational element/time-integration strategy for non-linear dynamics. *International Journal for Numerical Methods in Engineering*, 37(11):1897–1913, June 1994.

[17] Martin O. L. Hansen. *Aerodynamics of Wind Turbines.* Routledge, 2nd edition, December 2007.

[18] Bjørn Haugen. *Buckling and stability problems for thin shell structures using high performance finite elements.* Dissertation, University of Colorado, Colorado, 1994.

[19] Bjørn Haugen. Private conversation, May 2012.

[20] Bjørn Haugen and Carlos A. Felippa. A unified formulation of Small-Strain corotational finite elements: I. theory. Technical report, University of Colorado, 2005.

[21] Kuo-Mo Hsiao and Jing-Yuh Jang. DYNAMIC ANALYSIS OF PLANAR FLEXIBLE MECHANISMS BY CO-ROTATIONAL FORMULATION. Technical report, National Chiao Tung University, 1989.

[22] Masashi Iura. Effects of coordinate system on the accuracy of corotational formulation for Bernoulli-Euler's beam. *International Journal of Solids and Structures*, 31(20):2793–2806, 1994.

[23] Jae Won Jang. *Characterization of live modeling performance boundaries for computational structural mechanics.* Ph.D. thesis, University of Washington, 2007.

[24] J. Jonkman, S. Butterfield, W. Musial, and G. Scott. Definition of a 5-MW reference wind turbine for offshore system development. Technical Report NREL/TP-500-38060, National Renewable Energy Laboratory, 2009.

[25] B. S. Kallesøe. Effect of steady deflections on the aeroelastic stability of a turbine blade. *Wind Energy*, 14(2):209–224, 2011.

[26] Timothy Knill. The application of aeroelastic analysis output load distributions to finite element models of wind. *Wind Engineering*, 29(2):153–168, March 2005.

[27] P. Krysl, S. Lall, and J. E. Marsden. Dimensional model reduction in nonlinear finite element dynamics of solids and structures. *International Journal for Numerical Methods in Engineering*, 51(4):479–504, March 2001.

[28] Per Kr. Larsen, Arild. H Clausen, and Arne Aalberg. *Stålkonstruksjoner. Profiler of formler.* Tapir akademisk forlag, Trondheim, 3. utgave edition, 2003.

[29] David J. Malcolm. Modal response of 3-Bladed wind turbines. *Journal of Solar Energy Engineering*, 124(4):372–377, November 2002.

[30] James F. Manwell, Jon G. McGowan, and Anthony L. Rogers. *Wind Energy Explained: Theory, Design and Application.* Wiley, 2 edition, February 2010.

[31] Kjell Magne Mathisen. Lecture 8, nonlinear finite element analysis. lecture notes from the course TKT4197 - nonlinear finite element analysis at NTNU, 2011.

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

[33] G.R. Miller. Coordinate-free isoparametric elements. *Computers and Structures*, 49(6):1027–1035, 1993.

[34] 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.

[35] D. C. Quarton. The evolution of wind turbine design analysis—a twenty year progress review. *Wind Energy*, 1(S1):5–24, 1998.

[36] Flemming Rasmussen, Morten Hartvig Hansen, Kenneth Thomsen, Torben Juul Larsen, Franck Bertagnolio, Jeppe Johansen, Helge Aagaard Madsen, Christian Bak, and Anders Melchior Hansen. Present status of aeroelasticity of wind turbines. *Wind Energy*, 6(3):213–228, 2003.

[37] 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.

[38] M.D. Rucki and G.R. Miller. An adaptable finite element modelling kernel. *Computers and Structures*, 69(3):399–409, 1998.

[39] Ahmed A. Shabana. *Computational Continuum Mechanics.* Cambridge University Press, March 2008.

[40] Ahmed A. Shabana, Oliver A. Bauchau, and Gregory M. Hulbert. Integration of large deformation finite element and multibody system algorithms. *Journal of Computational and Nonlinear Dynamics*, Vol 2:351–359, 2007.

[41] Ken Shoemake. Quaternions.

[42] J. C. Simo and L. Vu-Quoc. On the dynamics of flexible beams under large overall motions-The plane case : part 1. *J. Appl. Mech. ASME 53,849-858.*, 1986.

[43] Paul E. Thomassen, Per Ivar Bruheim, Loup Suja, and Lars Frøyd. A novel tool for FEM analysis of offshore wind turbines with innovative visualization techniques. Trondheim, 2011.

[44] Lloyd Nicholas Trefethen and David Bau. *Numerical Linear Algebra.* SIAM, June 1997.

[45] Dimitri van Heesch. Doxygen, February 2012.

[46] John Vince. *Geometric Algebra for Computer Graphics.* Springer, Bornemouth University, 2008.

[47] Arne Vollan and Louis Komzsik. *Computational Techniques of Rotor Dynamics with the Finite Element Method.* CRC Press, March 2012.

[48] Fabian Vorpahl and Wojciech Popko. Description of the load cases and output sensors to be simulated in the OC4 project under IEA wind annex XXX, 2011.

[49] Fabian Vorpahl, Wojciech Popko, and Daniel Kaufer. Description of a basic model of the "UpWind reference jacket" for code comparison in the OC4 project under IEA wind annex XXX, 2011.

Appendices

Appendix A

Excerpt of C++ code

A.1 FrameEltCorot.h

```cpp
#include "FrameElt.h"
#include "GQuat.h"

/**
 * The FrameEltCorot class implements a frame element using the corotated
 * formulation.
 *
 * By using the non-linear solver with this element, arbitrarily large
 * finite displacements and rotations are supported. The major difference
 * to a FrameElt is that the UpdateResistingForce() function extracts the
 * deformational displacements and rotations, after establishing the corotated
 * configuration (which is one the line between the displaced nodal posistions).
 *
 * The other difference lies in the NewmarkSolver itself, in how it
 * updates/accumulates the rotations after each iteration and increment.
 * See for example NewmarkSolver::UpdateStateNonlin()
 *
 * @author PIB
 */
class FrameEltCorot : public FrameElt
{

public:
  FrameEltCorot(
    Node *n1,
    Node *n2,
    Material *m,
    CrossSection *xsect);

  virtual ~FrameEltCorot();

  // FrameElt overrides //
  virtual void InstallMCK(bool lumped = true);
  virtual void SetWeakAxisPerpendicularTo(GVector v);
  virtual void Twist(Scalar angleDeg);
  // Do not use the functions below.
  virtual void InstallStiffness() { assert(false); };
  virtual void InstallMass(bool lumped = true) { assert(false); };
  virtual void InstallDamping(bool lumped = true) { assert(false); };
  virtual Element::ElementType GetType() const;

  virtual void UpdateResistingForce();
  virtual void PostIterations();
  virtual Scalar GetStrainEnergy() const;
  virtual void GetElasticForces(GVector &fi, GVector &fj, GVector &mi, GVector &
     mj, bool NRiter = false) const;
  // End FrameElt overrides //

private:
  void GetElasticForcesCorot(GVector &fi, GVector &fj, GVector &mi, GVector &mj)
     const;
  void UpdateStateCorotational();
  void ComputeSdirTdirCorot(GTensor const& roti, GTensor const& rotj);
  void ComputeSdirTdirCorot(GQuat const& roti, GQuat const& rotj);
  void StoreInitDirections();

  /// Initial (t = 0) directions
  GVector mInitDirection, mInitSDirection, mInitTDirection;
  /// Initial transformation tensor
  GTensor mT0;

  // The current deformational translations and rotations
  GVector mDefTransi;
  GVector mDefTransj;
  GVector mDefRoti;
  GVector mDefRotj;


};
```

:

A.2 GQuat.h

```
#include "Def.h"
#include "GTensor.h"
class GQuatRing;

/**
 * An implementation of a quaternion type.
 *
 * @author PIB
 */
class GQuat
{

public:
  GQuat();
  GQuat(Scalar i, Scalar j, Scalar k, Scalar r);
  GQuat(GVector const& rotAxis, Scalar angle);
  explicit GQuat(GVector const& rotVector);
  explicit GQuat(GTensor const& rotTensor);
  Scalar X() const { return mx; }
  Scalar Y() const { return my; }
  Scalar Z() const { return mz; }
  Scalar W() const { return mw; }

  Scalar Magnitude() const;
  Scalar Norm() const;
  Scalar Abs() const;
  GVector Rotate(GVector const& vec) const;

  GTensor& ToTensor() const;
  void Normalize();

  GQuat& Negate();
  GQuat& Conjugate();
  GQuat& Square();
  GQuat& Invert();

  GQuat& operator+=(const GQuat& b);
  GQuat& operator-=(const GQuat& b);
  GQuat& operator*=(const Scalar b);
  GQuat& operator/=(const Scalar b);
  GQuat& operator*=(const GQuat& b);
  GQuat& operator/=(const GQuat& b);

  friend int operator!=(const GQuat& a, const GQuat& b);
  friend GQuat operator-(GQuat& x);
  friend GQuat operator+(const GQuat& x, const GQuat& y);
  friend GQuat operator-(const GQuat& x, const GQuat& y);
  friend GQuat operator*(const GQuat& x, const Scalar y);
  friend GQuat operator*(const GQuat& x, const GQuat& y);
  friend GQuat operator/(const GQuat& x, const GQuat& y);
  friend std::ostream& operator<<(std::ostream& ost, const GQuat& x);

private:
  Scalar mx, my, mz, mw;

  static GTensorRing sWorkTensorRing;
  /// Epsilon
  static const Scalar sEPS;
};
```

:

A.3 NewmarkDOF.h

```
#include "GaussElimDOF.h"
#include "GQuat.h"

class NewmarkDOF : public GaussElimDOF
{
 (...)

  GTensor GetRotDispAccumT() const;
  GQuat GetRotDispAccumQ() const;

private:
  /// Flag enabling finite rotations for this DOF.
  bool mIsCorotational;

  /// Similar to mPrevDisp, but used for finite rotations.
  GTensor mPrevRotDispT;
  /// Similar to mDisplacement, but used for finite rotations.
  GTensor mRotDispT;

  /// Similar to mPrevDisp, but used for finite rotations.
  GQuat mPrevRotDispQ;
  /// Similar to mDisplacement, but used for finite rotations.
  GQuat mRotDispQ;
```

:

A.4 Nonlinear timestep function

```
/**
 * Does one time step using Newton−Raphson (NR) equilibrium iterations
 * until the residual is sufficiently small.
 *
 * When this function is completed, the internal loads of all elements
 * will be in equilbrium with the applied external loads. For a dynamic
 * analysis, the modified effective stiffness matrix is used (Khat),
 * so that inertia and damping forces are taken into account.
 *
 * If AnalysisInitNonlinear() has not been called, it will be called once.
 *
 * @note Only dynamic analyses are assumed.
 *
 * @throws NRIterationsNotConvergingException if the iteration counter
 *  passed some limit without converging.
 *
 * @author PIB
 */
void FEDynamic::DoTimeStepNonlinear()
{
  bool doIterations = true;
  bool useFullNR = true;
  int nMaxIter = 100;

  if (!mAnalysisInit)
  {
    AnalysisInitNonlinear();
    mAnalysisInit = true;
  }

  assert(dynamic_cast<NewmarkSolver*>(mSolver) != NULL);
  NewmarkSolver* solver = (NewmarkSolver*)mSolver;

  Scalar startTime = mCurrTime;
  mCurrTime += (1 + solver->GetAlpha()) * solver->GetDt(); // used in ApplyLoad

  solver->ResetKhat();
  InstallMCK();
  ApplyConstraint();
  solver->ComputeAandB();
  solver->PrepareKhatInv();
  ApplyLoad();
  solver->ResetFs();
  solver->ComputePHat();

  solver->ZeroDispTimeStep(); // NB! Do this before InitDeltaR.
  solver->InitDeltaR();

  int iterCount = 0;
  bool converged = false;
  while (!converged)
  {
    // If full Newton-Raphson is used, update the stiffness for each iteration,
    // except first iteration, where the computation was already done.
    if (useFullNR && iterCount > 0)
    {
      solver->ResetKhat();
      InstallMCK();
      ApplyConstraint();
      solver->ComputeAandB();
      solver->PrepareKhatInv();
    }

    solver->SolveSystemNonlin();

    if (!doIterations)
      break;

    solver->ResetFs();

    for (unsigned int i = 0; i < mElements.size(); i++)
    {
      mElements[i]->UpdateResistingForce();
    }

    for (unsigned int i = 0; i < mLoads.size(); i++)
    {
      mLoads[i]->UpdateLoadNonlin();
    }

    solver->ComputeResidualForce();
    Scalar loadResidual = solver->GetResidual(NewmarkDOF::eLOAD);
    Scalar displResidual = solver->GetResidual(NewmarkDOF::eDISPLACEMENT);

    Scalar convTolLoad = 1E-6;
    Scalar convTolDispl = 1E-6;
    converged = (loadResidual < convTolLoad && displResidual < convTolDispl);

    iterCount++;

    if (iterCount > nMaxIter)
    {
      // Allow the first increment to not be fully convergent
      if (startTime == 0)
      {
        converged = true;
        break;
      }

      throw NRIterationsNotConvergingException();
    }
    }

    solver->UpdateStateNonlin();
    solver->ZeroDispTimeStep();
    mCurrTime = startTime + solver->GetDt();

    Model::UpdateSensors(mCurrTime);
    if (mSensorSolution != NULL)
      mSensorSolution->LogIncrement(0, iterCount, mCurrTime, solver->GetDt(),
          mAlpha, true);

}
```

:

A.5 Corotational update procedure

```
/**
 * Extracts the deformational state of this element from the current (iterated)
     displacement.
 * Will also establish the corotated base vectors.
 *
 * @author PIB
 */
void FrameEltCorot::UpdateStateCorotational()
{
  Node* ni = GetNodei();
  Node* nj = GetNodej();

  GVector const& xi = ni->GetLoc();
  GVector const& xj = nj->GetLoc();
  GVector ui = ((NewmarkDOF*)ni->GetDOF(DOF::eTranslation))->GetPrevDisp() + ((
      NewmarkDOF*)ni->GetDOF(DOF::eTranslation))->GetDispTimeStep();
  GVector uj = ((NewmarkDOF*)nj->GetDOF(DOF::eTranslation))->GetPrevDisp() + ((
      NewmarkDOF*)nj->GetDOF(DOF::eTranslation))->GetDispTimeStep();

  // "displ" = displaced
  GVector xidispl = xi + ui;
  GVector xjdispl = xj + uj;

  // Establish the deformed local CR system by simply taking the ghost element
  // directly between the (displaced) position of the nodes
  GVector newDirection = (xjdispl - xidispl).Direction();
  Scalar newLength = (xjdispl - xidispl).Magnitude();
  Scalar axialElong = 2.0/(mLength+newLength) * (xj - xi + 0.5*(uj-ui)).Dot(uj-ui
      );

  mDirection = newDirection;  // Update the element direction
  assert(mDirection.Magnitude() > 0.0);

  // Compute deformational translations (i.e. axial) in local C.S!
  mDefTransi.SetComponents(0, 0, 0);
  mDefTransj.SetComponents(axialElong, 0, 0);

  // Get finite rotations and set principal axes.
  GTensor finiteRoti, finiteRotj;
  finiteRoti = ((NewmarkDOF*)ni->GetDOF(DOF::eRotation))->GetRotDispAccum();
  finiteRotj = ((NewmarkDOF*)nj->GetDOF(DOF::eRotation))->GetRotDispAccum();

  ComputeSdirTdirCorot(finiteRoti, finiteRotj);

  // Establish transformation matrices //
  GTensor Tn(mDirection.vx, mDirection.vy, mDirection.vz,
    mSDirection.vx, mSDirection.vy, mSDirection.vz,
    mTDirection.vx, mTDirection.vy, mTDirection.vz);  // Using the updated
        directions

  // Compute deformational rotations (local)
  GTensor const& T0transposed = mT0.Transpose();
  GTensor const& Rd_i = Tn.Dot(finiteRoti.Dot(T0transposed));
  GTensor const& Rd_j = Tn.Dot(finiteRotj.Dot(T0transposed));
  // Save the deformational rotation vectors (local)
  CorotTools::TensorToVector(Rd_i, mDefRoti);
  CorotTools::TensorToVector(Rd_j, mDefRotj);
}
```

:

A.6 Accumulation of rotations

```
void NewmarkDOF::UpdateStateNonlin(const Scalar *factr)
{
  if(mConstrainedp)
    return;

  ( *** )

  if (this->GetTag() == DOF::eTranslation)
  {

      GVector duDot;
      GVector duDotDot;

      // velocity and acceleration increments
      duDot = mDispTimeStep*dampKhatFactor - mVelocity*dampaFactor -
          mAcceleration*dampbFactor;
      duDotDot = mDispTimeStep*massKhatFactor - mVelocity*massaFactor -
          mAcceleration*massbFactor;

      mVelocity += duDot;
      mAcceleration += duDotDot;
      mPrevDisp += mDispTimeStep;
      mDisplacement = mPrevDisp;

  }
  else if (GetTag() == DOF::eRotation)
  {
    if (mIsCorotational)
    {
      GVector duDot;
      GVector duDotDot;

      // velocity and acceleration increments
```

```
        duDot = mDispTimeStep*dampKhatFactor - mVelocity*dampaFactor -
            mAcceleration*dampbFactor;
        duDotDot = mDispTimeStep*massKhatFactor - mVelocity*massaFactor -
            mAcceleration*massbFactor;

        // Tensor update
        CorotTools::IncrementTensor(mDispTimeStep, mPrevRotDispT);
        mRotDispT = mPrevRotDispT;
        CorotTools::TensorToVector(mPrevRotDispT, mPrevDisp);
        mDisplacement = mPrevDisp;

        mVelocity += duDot;
        mAcceleration += duDotDot;
      }
    }
    else  // Default (additive) updating
    {
      GVector duDot;
      GVector duDotDot;

      // velocity and acceleration increments
      duDot = mDispTimeStep*dampKhatFactor - mVelocity*dampaFactor - mAcceleration*
          dampbFactor;
      duDotDot = mDispTimeStep*massKhatFactor - mVelocity*massaFactor -
          mAcceleration*massbFactor;

      // increment the actual values
      mVelocity += duDot;
      mAcceleration += duDotDot;
      mPrevDisp += mDispTimeStep;
      mDisplacement = mPrevDisp;

    }

    mAppliedLoad.Zero();

}
```

: