Aeroelastic Response of High-Rise Buildings

Master Thesis by Anders Trondal Svendsen and Allan Michaelsen

Department of Civil Engineering - Aalborg University - 2008

Aeroelastic Response of High-Rise Building

Master Thesis by

Anders Trondal Svendsen & Allan Michaelsen

Department of Civil Engineering, Aalborg University
September 2007 - June 2008

Anders Trondal Svendsen

Allan Michaelsen

Abstract

The present report deals with the aeroelastic response of high-rise buildings. The wind flow around an assumed square cylindrical high-rise building is modeled using the commercial CFD-program Ansys CFX 11. To model the Fluid-Structure Interaction (FSI) a FEM beam model is used to represent the structure while the dynamic model is set up using a modal representation.

A method for generating 3D meshes suitable for exterior flow around a rectangular cylindrical building is presented, with basis in an algorithm for generating 2D meshes using a hyperbolic grid generation scheme. Some simple mechanisms to ensure a smoother mesh are presented as well. The difference between using structured and unstructured meshes are analyzed, and it is concluded that use of structured meshes is superior with respect to computation time.

An analysis is then performed to see, which methods for governing mesh stiffness during mesh deformations in Ansys-CFX is best, and on basis of 2D analyses it is concluded that the use of the reciprocal of the wall distance as stiffness parameter, or use of the built-in function 'Increase Near Boundaries' show superior performance.

Due to limited availability of computational power in the project, analyses are made determining the effects of using simulation settings that are less than ideal. In these analyses it is shown that these delimitations show little influence on the loads on the building in the streamwise direction, but that the loads in the cross stream direction show strong dependency.

The effect of modeling the aeroelastic response of a high-rise building is determined, by comparing the response of two different simulations, one where the structure is allowed to move freely, and one where it is stationary. It is concluded that there is an obvious difference between the two methods.

Finally the aeroelastic load response of a high-rise building are obtained by use of flutter derivatives, and it is shown that this method gives qualitatively good results, and seems applicable to use on high-rise buildings. The response is found using the two first eigenmodes only, and the effects of including more eigenmodes is not analyzed.

Preface

This Master Thesis is written by Allan Michaelsen and Anders Trondal Svendsen during the 3rd and 4th semester of the Masters of Structural and Civil Engineering Programme at the Department of Civil Engineering, Aalborg University. The theme of the thesis is "Aeroelastic Response of High-Rise Buildings".

The thesis consists of a report and a DVD. The report is divided into a main part and an appendix. The DVD contains the thesis in .tex and .pdf format, as well as the programs, simulation setups, and data files produced throughout the course of the project period. The files on the DVD are generated using the following programs:

Ansys-CFX 10.0	CFD program
Ansys-CFX 11.0	CFD program
CorelDRAW 9	Graphics program
MatLab R2007b	Math program
Microsoft Visio	Graphics program
Microsoft Visual Studio 2005	Text editor
MiKTeX 2.7	LaTeX source program (freeware)
WinEdt 5	LaTeX text editor (shareware)

The chapters in the report are numbered 1,2,3 etc. and the appendices are numbered A,B,C etc. The references in the thesis are made using the Harvard model, as follows; [*Surname(s) Year, page number(s)*].

In the equations throughout the report, the following notation has been used:

- Matrices are written in capital letters using a bold font as:	$\mathbf{M}$, $\mathbf{K}$
- Vectors are written in small letters using a bold font as:	$\mathbf{q}$, $\dot{\mathbf{q}}$
- Variables, constants, and coefficients are written in using italic font as:	u, C

The Master Thesis has been written under the supervision of Assistant Professor Jesper Winther Stærdahl from Aalborg University and Professor Niels N. Sørensen from Risø.

Contents

1 Introduction

During the recent years the heights of buildings being constructed have increased rapidly. The demand for tall and impressive buildings has grown and is still growing. An example of this is Burj Dubai (Tower of Dubai), which as of the 21st of July 2007 became the tallest building in the world with its 512.1 m. This is spectacular in it self but the tower is far from finished. The final height of the building is still a secret but figures released from a contractor suggest 818 m as the final height with an expected number of floors of 160. Fig. 1.1 shows Burj Dubai as it is supposed to look like when it is finished.

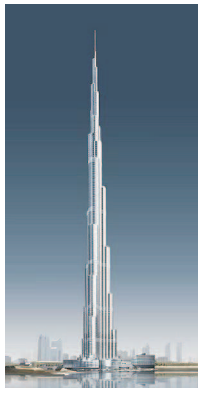


Figure 1.1: Burj Dubai, The Worlds tallest building

Impressive as these new skyscrapers are they introduce a number of problems during the design phase. Especially the determination of wind loads on the structure is a difficult task. The Eurocode covers buildings of up to 200 m only [CEN/TC250 2005], which leaves it up to the designers to determine the loads when building heights exceed this. Presently, wind loads and the structural response are determined from scale model tests in a wind tunnel. The determined loads from wind tunnel tests give very good predictions of the real loads but the downside of this method is the cost of these. Wind tunnel tests are very expensive and as all designers are looking to keep their cost at a minimum an alternative and cheaper method than wind tunnels tests is valuable, which is where using Computational Fluid Dynamics (CFD) enters the picture.

1.1 Project Description

The purpose of this project it to model the aeroelastic response of a high-rise building, and use this to asses the effect of said aeroelasticity. Once this is done, it will be examined if it is possible to model the loads on the structure using the following form:

$$\mathbf{A}\mathbf{q} + \mathbf{B}\dot{\mathbf{q}} = \mathbf{f} \tag{1.1}$$

where

$\mathbf{A}$, $\mathbf{B}$	are $n \times n$ coefficient matrices
$\mathbf{q}$	is the n first modal coordinates
$\mathbf{f}$	is the modal load vector

The wind loads are obtained using the Computational Fluid Dynamics (CFD) program Ansys CFX 11.0, and the structural model (a modal model), is set up using MatLab. Finally the two are linked together through a series of Fortran routines.

There exist a wide variety of commercial CFD programs such as 'Fluent' or 'Numeca', and the choice of using Ansys CFX is entirely based on that the University has student licenses for CFX already, and that the program has been shortly introduced on an earlier semester, which made it familiar to use.

In secs. 1.1.1 - 1.1.9, each chapter will be shortly described along with it's purpose.

1.1.1 Hyperbolic Mesh Generation

The purpose of the chapter is to determine the most suitable method for producing meshes that are well suited for large exterior flow simulations, like the flow around a tall building. For any CFD simulation to be possible using the finite volume method, like Ansys CFX does, a mesh must first be produced, dividing the domain into a finite number of volumes. The quality of the mesh has a large influence on the quality of simulations carried out using the mesh.

In the chapter a method for creating a 2D mesh of these qualities is presented, employing an algorithm using hyperbolic partial differential equations. Different means of controlling the mesh generation is presented, along with a description of their use. After describing the method for creating the 2D mesh, a method to convert the 2D mesh into a 3D mesh is derived.

1.1.2 Mesh Analyses

The structured meshes from the previous chapter are entered into an analysis where meshes of varying coarseness produced using the algorithm is tested against unstructured meshes of similar coarseness, created with Ansys CFX-Mesh. This is done to determine the advantage of using structured meshes compared to other types. Also, the effect of using prismatic inflation layers with unstructured meshes is tested. Finally, it is concluded which method is best on the basis of the computational time.

1.1.3 Analyses Moving Mesh in CFX

When conduction simulations where the structure is allowed to move it is important for the mesh to be able to obtain the deformations. In this chapter an analysis is conducted to determine how Ansys CFX handles moving meshes and what settings of the mesh stiffness are preferable. Both built in functions and user specified functions are analyzed.

1.1.4 Model Setup in CFX

The purpose of the chapter is to describe the setup of the model in CFX.

In the chapter, the choice of geometry is described. Afterwards a description of the setup in Ansys-CFX Pre is given, along with an explanation of the choices made for model setup in CFX. The forces on the structure are determined via the use of additional variables in CFX. The last part of this chapter deals with how these additional variables are defined.

1.1.5 Structural Model

The purpose of the chapter is to describe the setup of the structural model in terms of defining the mass- and stiffness matrices, and afterwards to convert it to a modal representation using a selected number of eigenmodes. Furthermore the method for converting loads on the structure to nodal loads will be presented.

1.1.6 Preliminary Analyses

If the simulations in this project were carried out using "ideal" conditions, the time and computer power needed would by far exceed what has been available. Thus, the purpose of the chapter is to describe the effects of the delimitations made, when using simulation settings that are not ideal. Ideally the simulation setup would be sufficient to capture the phenomenon of vortex shedding, but due to the necessary delimitations made in the project, this is not possible. Therefore simulations have subsequently been conducted to examine if the vortex shedding could be captured using a simulation setup with a higher resolution.

1.1.7 Effect of Modeling Aeroelastic

The purpose of the chapter is analyze if there's any notable difference between modeling aeroelastic response, and the more common approach where loads are obtained from a stationary simulation. This is done by comparing the response of two simulations; one where the structure is given a prescribed deformation through the first eigenmode and subsequently allowed free motion in a flow, and a stationary simulation where the loads on the structure are then recorded and used in conjunction with the Newmark-Algorithm to produce a response from the same initial deformation as in the first simulation.

1.1.8 Modal Response

After the effect of modeling has been determined, it is examined if it is possible to describe the loads on the building through use of the modal representation in eq. 1.1, using the first two eigenmodes of the building. This is done through a series of simulations where the structure is subject to harmonic motions in either of the first two eigenmodes using a range of frequencies, and with a prescribed amplitude. The data from these simulations (load trails) are then used to determine the coefficients of $\mathbf{A}$ and $\mathbf{B}$ in eq. (1.1).

1.1.9 Conclusion

In this chapter the main results and observations of the conducted analyses and methods used are summed up. It is attempted to conclude on the obtained results

and the validation of these.
In the last part some suggestions are made as to where and how the methods and simulations used in this project can be improved to yield better results.

2

Hyperbolic Mesh Generation

In order to conduct the CFD simulations a suitable mesh must first be generated. Many different types of meshes exist. In this project a hyperbolic mesh generation method have been used which generate good structured meshes for exterior flow simulations.

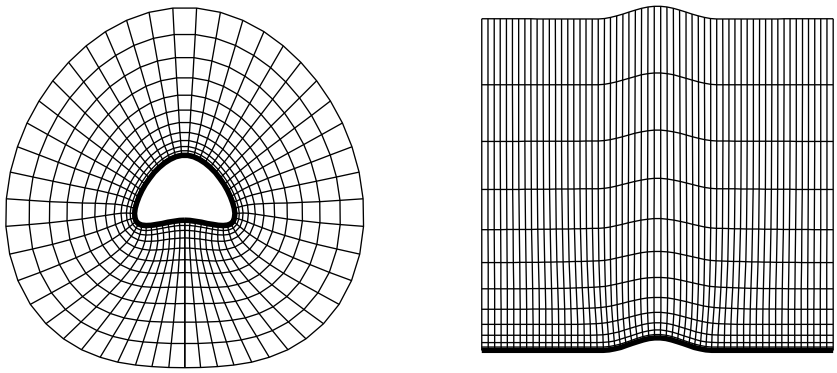
In the present chapter the MatLab program used to generate the meshes is described. The theory behind the hyperbolic mesh algorithm is explained in app. A and therefore this chapter deals with the implementation of the algorithm into a MatLab program and the methods used to control different aspects of the mesh generation.

Besides the hyperbolic mesh generation technique an interpolation method called "Transfinite Interpolation" has also been used in the process. This interpolation scheme is described in this chapter as well.

When computing flow problems using CFD it is generally accepted that CFD simulations with an orthogonal mesh runs faster than with a non-orthogonal mesh. In order to obtain a mesh with orthogonal cells around an arbitrary closed boundary, a possible approach is to generate the mesh using an algorithm governed by Hyperbolic Partial Differential Equations (PDE's).
In the following, the implementation of such an algorithm into a program will be described. The theory behind the hyperbolic mesh algorithm can be found in app. A and the MatLab files are included on the DVD in [*DVD:\Mesh_ Generation*]. The mesh generation program is used throughout the project. In each case a reference is made to a directory on the DVD containing one or more *.m* files. These files replace the *main.m* file in [*DVD:\Mesh_ Generation*] to generate the respective mesh.

2.1 Implementation of Algorithm into a MatLab Program

The algorithm is implemented into a MatLab program, with the purpose of generating a suitable mesh from specified initial data along the body surface. Two different modes have been included; a cyclic mode, fig. 2.1a, and a vertical boundary mode, fig. 2.1b.



a) Cyclic mesh *b)* Vertical boundary mesh

Figure 2.1: The two available mesh modes. The bold lines represent the body surface.

In fig. 2.2, an overview of the programs used to generate the mesh and subse-

quently write it to a format loadable by Ansys CFX is shown. In the following, each of the functions will be explained and examples of certain features and their effect will be given.

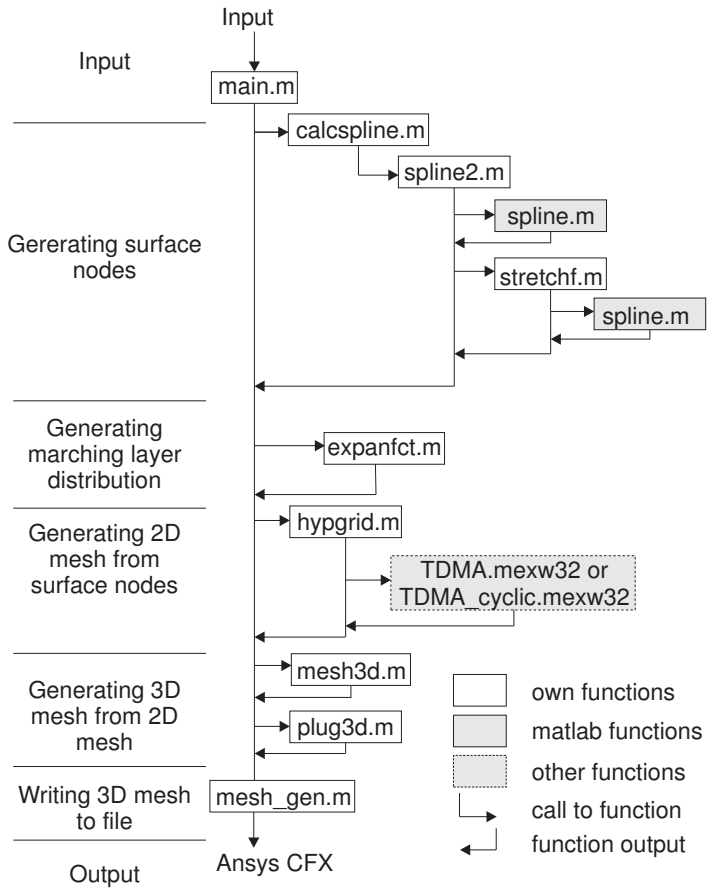


Figure 2.2: Structure of programs.

2.1.1 main.m

In `main.m` all input for the programs are entered, these include:

`mode`	determines whether the mesh is generated as a cyclic mesh (0) or a vertical boundary mesh (1).
`COORDS`	is a matrix containing coordinates from which to spline the geometry
`BREAKS`	is a row vector defining in between which coordinates the splines are to run. e.g.: `BREAKS` = [1 3 4] defines that the first spline runs over coordinates 1,2,3 and the second over 3,4.

`n`	is a row vector defining the number of desired nodes on each spline.
`a1, b1`	are matrices defining the start- and end slopes for each spline.
`x1, y1`	are cells defining the coordinates for the stretch functions for each spline. See sec. 2.1.5 for info on stretch functions.
`a2, b2`	are matrices defining the start- and end slopes of each stretch function.
`N`	defines the number of marching layers wanted.
`L`	defines the height of the innermost cell. (`L` was initially used as an approximate extent of the mesh from the body geometry.)
`x2 ,y2`	defines the coordinates for the stretch function determining the outwards marching layer distance.
`a3, b3`	defines the start- and end slopes for the stretch function determining the outwards marching layer distance.
`dis`	is a factor determining the amount of dissipation at the body surface.
`eps1`	is a factor determining the volume averaging for each layer
`eps2, eps3`	are factors determining the diminishing of dissipation for each layer.

After all the input are given, `main.m` calls the function `calcspline.m` (see sec. 2.1.2) in order to generate the body geometry at $\eta = 0$. Once the body geometry is returned from `calcspline.m`, a call is made to `expanfct.m` returning the stretch function of the outwards marching layers. After this, a call is made to `hypgrid.m` (see sec. 2.1.7) which calculate the marching layers for k=2 to k=N.

The 2D mesh returned from `hypgrid.m` is used when calling `mesh3D.m` which uses the 2D mesh as input and returns a generated 3D mesh on the basis of the specified number of layers in the z-direction and the rotation angle. The 3D mesh is returned to `main.m`which can then finally call `mesh_gen.m` (see sec. 2.1.11) in order to generate a mesh file to be loaded into ANSYS CFX.

2.1.2 calcspline.m

```
SYNTAX: [Xout Yout]=calcspine(BREAKS,COORDS,a1,a2,b1,b2,x1,y1,n).
```

In `calcspline.m` the coordinates defining the boundary surface are splined as given by the input in `main.m`, and a specified number of nodes are then assigned to each spline and distributed according to the respective stretch functions. The theory behind splines can be found in app. B. This produces coordinates to a

number of splines, which are then assembled into a complete body surface. In fig 2.3, it is shown how 7 defining coordinates are put together into a single geometry using 3 splines.

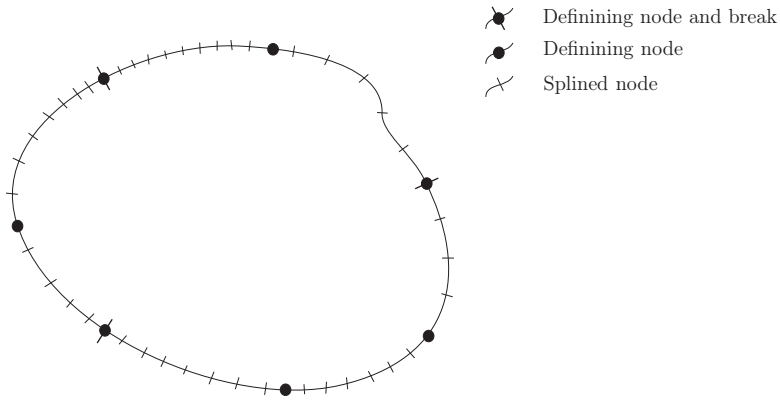


Figure 2.3: Definition sketch, combining splines into geometry.

The order of `calcspline.m` is as follows:

1. The coordinates provided from COORDS is divided into sets of coordinates for each spline, using info from BREAKS.
2. For each spline, the function `spline2.m` (see sec. 2.1.3) is called with the appropriate coordinates as well as input for the stretch functions of each.
3. The coordinates returned from each call of `spline2.m` is gathered into an X- and Y vector, defining the body surface.

Afterwards, the X- and Y vectors are returned to `main.m`.

2.1.3 spline2.m

`SYNTAX: [x y]=spline2(Y,a1,b1,xin,yin,n,a2,b2).`

In `spline2.m` a set of coordinates defining one spline is input along with input for the stretch function defining the distribution of nodes on the given spline. The function makes a linearized curve length estimate by use of:

$$s = \sum_{j=1}^{n} \Delta s = \sum_{j=1}^{n} \sqrt{\Delta x^2 + \Delta y^2} \tag{2.1}$$

The function uses this estimated curve length along with the coordinates as input into the MatLab function `spline.m`. The function can handle constrained start- and end slopes for the stretch function, as well as free slopes.

2.1.4 spline.m

`SYNTAX: PP = spline(X,Y)`

The function `spline.m` is an internal MatLab function. For further theory on this function, see app. B.

The output from spline is of the MatLab type "structure" holding coefficients for the proper cubic polynomials which can be used to evaluate the spline using the function `ppval.m`. If the input into `spline.m` is on the form $([\mathbf{X}], [\mathbf{Y}])$, the evaluation will return the y value for a given x input. If the input into `spline.m` are on the form $([\mathbf{S}], [\mathbf{X}; \mathbf{Y}])$, the evaluation will return the x, y value for a given s input, where s denotes a curve length.

2.1.5 stretchf.m

`SYNTAX: [Y]=stretchf(x,y,n,a,b).`

In `stretchf.m` a set of coordinates are entered, ranging from 0 to 1 for both x- and y values. These are used to spline a function ranging from $(x, y) = (0, 0)$ to $(x, y) = (1, 1)$. This spline is then used with a number of evenly distributed points on the X axis, to which the evaluated spline values of y are returned. Also, a constraint on the start and end slopes is possible, but not required.

The purpose of the stretch function is to define a distribution of nodes on a body surface, that are not evenly distributed - so that it is possible to have a high concentration of nodes near sharp corners or areas of particular interest to the flow calculations. It also ensures, that there are no strong deviation of node spacing from one cell to the next. As can be seen in fig. 2.4, the evaluated values of the stretch function on the y axis are unevenly spaced, but has a smooth transition from a higher node density to a lower node density, whereas the input is evenly spaced nodes on the x axis.

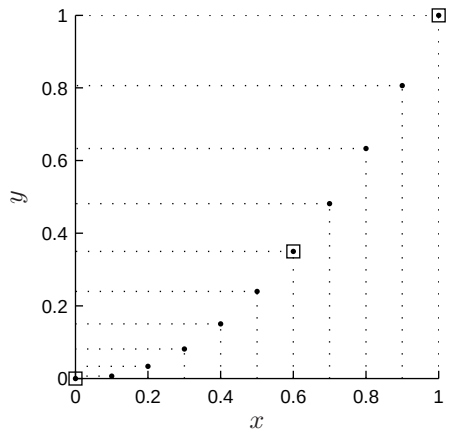


Figure 2.4: Plot of the stretch function, □ are input coordinates, • are splined values.

The vector defining the points on the y axis with values ranging from 0 to 1 is then used as normalized curve lengths when calling the spline function, or to determine the outwards marching distance between different layers of η.

2.1.6 expanfct.m

`SYNTAX: [p]=expanfct(n1,m,a1,a2).`

In `expanfct.m` a function for determining the marching layer distance is given. The function creates a layer distribution for `n1` layers, where the first `m` % layers have a constant expansion of `a1` % and the remaining layers go from `a1` % expansion in the inner part to `a2` % expansion in the outer part. To illustrate this, an example of three different functions is given in fig. 2.5. Here the dashed and dashed-dotted lines have a constant expansion of 10 and 20 % respectively, where the solid line have an expansion of 5 % for the inner 70 % layers, after which the expansion gradually changes to 35 %, giving the very steep rise in the end.

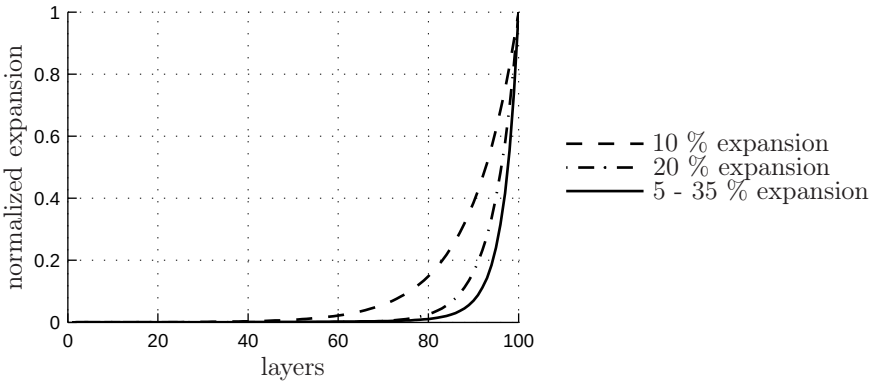
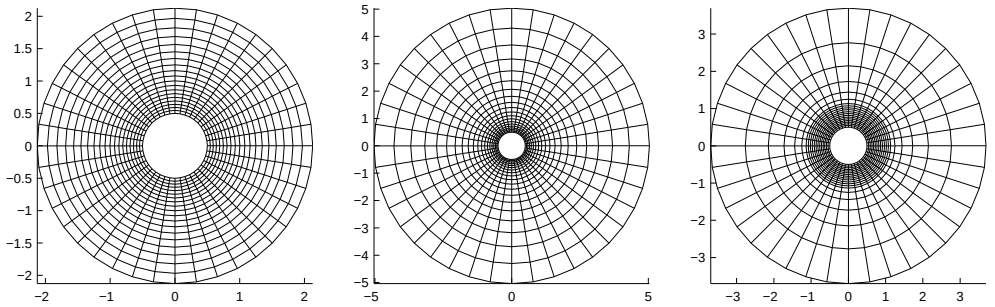


Figure 2.5: Examples of expansion function for marching layers.

To illustrate this effect, three meshes have been generated around a cylinder with a diameter of 1 m. There are 40 nodes on the cylinder geometry and 20 layers, where the innermost cells have a constrained height of 0.05 m. This is shown in fig. 2.6.



a) 10% *b)* 20% *c)* 5 – 35%

Figure 2.6: Visualization of expansion effect.

Here it can be seen that for figs. 2.6a and 2.6b the expansion rates are constant, but the mesh expands more rapidly for the latter. In fig. 2.6c the expansion is very low for the innermost layers, followed by a very rapid expansion in the outer layers.

2.1.7 hypgrid.m

`SYNTAX: [X Y]=hypgrid(mode,X,Y,N,P,L,dis,eps,eps2,eps3).`

In `hypgrid.m` the algorithm, given by eqs. (A.15) through (A.24), is used to calculate each new layer of η from the data given in the previous layer, by solving the system $\mathbf{Ux} = \mathbf{v}$, where $\mathbf{U}$ is a blocktridiagonal matrix, $\mathbf{x}$ is the solution to the coordinates of the current layer of η and $\mathbf{v}$ is the righthand vector. The righthand vector $\mathbf{v}$ consists of several parts; the known near solution state (x_0, y_0), the prescribed volume, and the dissipation. Depending on the mode of the mesh - the boundary conditions, given by $j = 1 \quad \wedge \quad j = n$, where n is the number of boundary nodes, are treated differently, as described in app. A.2.

The function runs as follows:

1. If the mode is cyclic, the last coordinate of the body surface is deleted as it is equal to the first.

2. A start guess for V^0 is made, using a central difference scheme to determine the base length of each cell, and a stretch function to determine the height h, see eq. (A.17).

3. An outer loop from k=2 to k=`N`, solving one outwards marching layer at the time.

4. An inner loop sets up the equations for constant k. The algorithm given by eq. (A.15) gives a number of equations equal to $2n$, where n is the number of surface nodes. By applying the correct boundary conditions, either as a cyclic mesh given by eq. (A.20), or a vertical boundary mesh given by eq. (A.21), the equations are set up on the form $\mathbf{Ux} = \mathbf{v}$ and solved using `TDMA.mexw32` or `TDMA_cyclic.mexw32`, giving the nodes of the new layer.

2.1.8 TDMA.mexw32 or TDMA_cyclic.mexw32

In order to solve the system of equations, a blocktridiagonal solver has been used. The advantage of such a solver is that the solution can be obtained in $O(n)$, the number of unknowns, rather than $O(n^3)$ as would be the case using Gauss elimination [Ferziger & Peric 2002, p. 95].
The solvers used, `TDMA.mexw32` and `TDMA_cyclic.mexw32` respectively, have been obtained from the project supervisors.

2.1.9 mesh3d.m

`SYNTAX:[Xout Yout Zout]=mesh3d(X,Y,N,xs,ys,as,bs,ntotal,H,nstart,rot)`

In `mesh3d.m` the 2D mesh generated in `hypgrid.m` is transformed into a 3D mesh. A stretch function determines the upwards layer distribution (in the z-direction). From a specified starting layer, the mesh is then gradually rotated towards a variable slope, α, at the top of the mesh. This is done by transforming the coordinates using simple geometric relations, as follows.

Let P denote a given node that is to be rotated α degrees, as shown in fig. 2.7. Furthermore, let (x_0, y_0, z_0) denote the coordinates of the corresponding node on the surface, (x, y, z) the coordinates of the transformed node, ds the distance from the node to the corresponding surface node, dx the difference in x between the node and the corresponding surface node, and dy the difference in y between the node and the corresponding surface node.

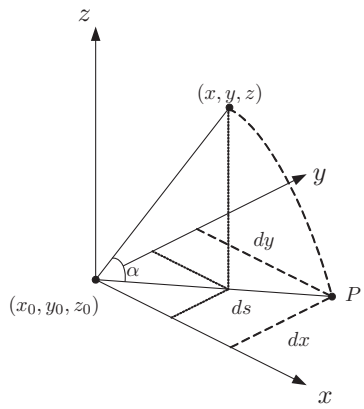


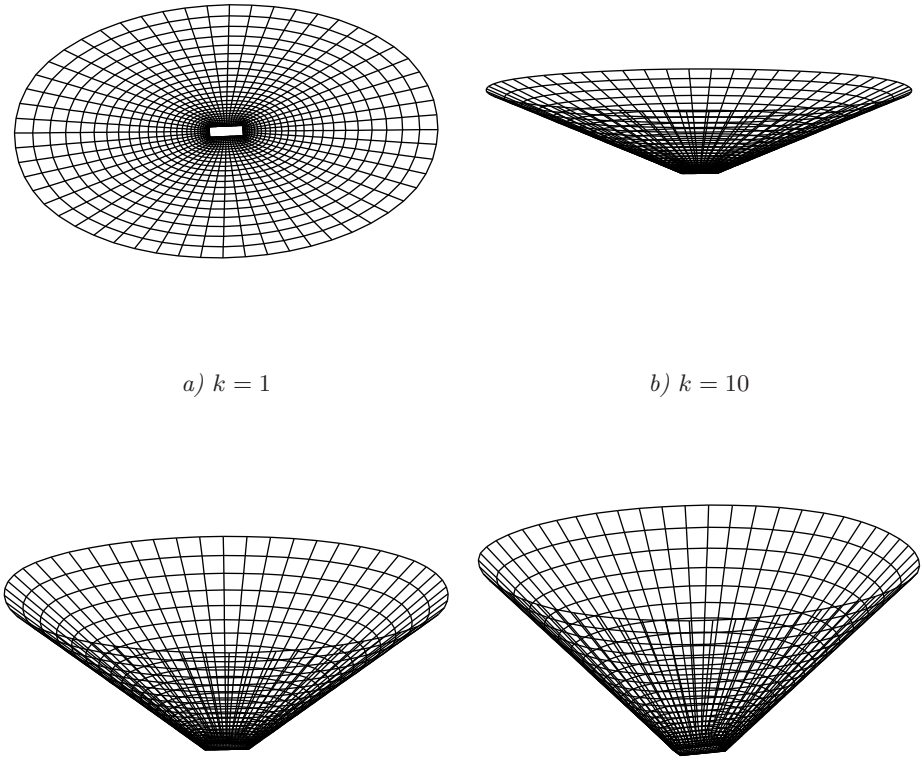
Figure 2.7: Displacement of node due to rotation - definition sketch.

It is then obvious to see, that the new coordinate (x, y, z) is given by:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_o \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \cos\alpha & 0 & 0 \\ 0 & \cos\alpha & 0 \\ 0 & 0 & \sin\alpha \end{bmatrix} \begin{bmatrix} dx \\ dy \\ ds \end{bmatrix} \tag{2.2}$$

Using eq. (2.2) the layers are rotated to an α degree angle. The stretch function defining the layer distribution in the z-direction is also used to control the gradual rotation of the layers - a larger layer spacing gives larger rotation. Finally, the coordinates are averaged gradually for each layer, so that the outer boundary for

the top layer will have z=constant. In fig. 2.8 a plot of four layers is shown, where the bottom layer shown on fig. 2.8a, has a 0 degree rotation, and the top layer shown on fig 2.8d has a 45 degree rotation.



a) $k = 1$ *b)* $k = 10$

c) $k = 20$ *d)* $k = 30$

Figure 2.8: Plot of mesh for four values of k: 1,10,20,30.

2.1.10 plug3d.m

`SYNTAX: [Xplug Yplug Zplug]=plug3d(X,Y,Z,n).`

In `plug3d.m` the top layer, rotated to an angle of α degrees is used to inter-

polate a mesh for the plug. In order to obtain $x-$ and $y-$ values for the nodes, a transfinite interpolation between the outer boundaries (given from `mesh3d.m`) is performed, assuming z=constant.
After the $(x_{i,j}, y_{i,j})$ coordinates are obtained for all layers in the plug, the $z_{i,j}$ coordinates are found for each layer by weighting between the corresponding nodes on the boundary; $z_{1,j}$, $z_{n,j}$, $z_{i,1}$, and $z_{i,m}$ as shown for one layer in fig. 2.9. Each of the four boundary nodes are weighted by the reciprocal of the distance.

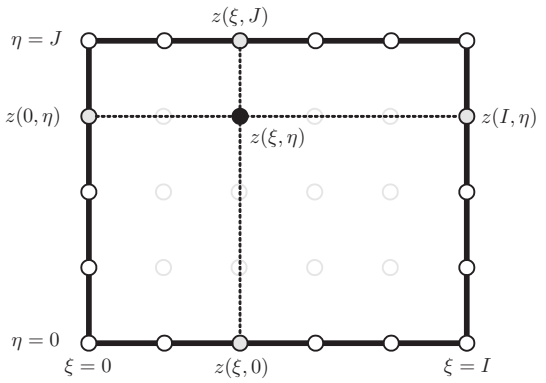


Figure 2.9: Definition sketch, reciprocal weighting

Transfinite Interpolation

Given a domain contained between four boundaries and a known distribution of nodes on the boundaries, for which it holds that two facing boundaries should have the same number of nodes. Let ξ denote the node numbering in one direction, going from 0 to I and η denote the node numbering in the other direction, going from 0 to J. Then the transfinite interpolation of the domain is given as follows:

$$\begin{aligned} X(\xi,\eta) &= X_1(\xi,\eta) + X_2(\xi,\eta) \\ Y(\xi,\eta) &= Y_1(\xi,\eta) + Y_2(\xi,\eta) \end{aligned} \tag{2.3}$$

where,

$$\begin{aligned} X_1(\xi,\eta) &= (1-\frac{\xi}{I})X(0,\eta) + \frac{\xi}{I}X(I,\eta) \\ Y_1(\xi,\eta) &= (1-\frac{\xi}{I})Y(0,\eta) + \frac{\xi}{I}Y(I,\eta) \\ X_2(\xi,\eta) &= (1-\frac{\eta}{J})(X(\xi,0) - X_1(\xi,0)) + \frac{\eta}{J}(X(\xi,J) - X_1(\xi,J)) \\ Y_2(\xi,\eta) &= (1-\frac{\eta}{J})(Y(\xi,0) - Y_1(\xi,0)) + \frac{\eta}{J}(Y(\xi,J) - Y_1(\xi,J)) \end{aligned} \tag{2.4}$$

[Thompson, Warsi & Mastin 1985]

In this case, $\frac{\xi}{I}$ and $\frac{\eta}{J}$ have been replaced by the node distribution in their respective directions, going from 0 to 1. An example of a plug mesh, corresponding to the geometry in fig. 2.8d is shown in fig. 2.10.

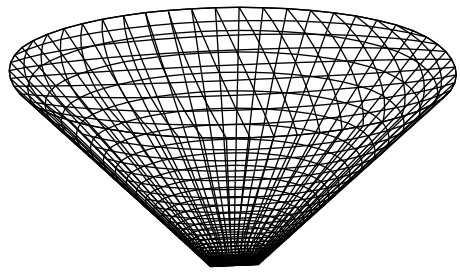


Figure 2.10: Plug mesh - only the outer boundaries are plotted.

2.1.11 mesh_gen.m

In `mesh_gen.m` the 3D mesh coordinates obtained in `mesh3d.m` or `plug3d.m` are written to a chosen `.msh` file, in a manner that allows ANSYS CFX to read it. For the cyclic mesh, the Mesh2plug, Bottom, and Bodysurface boundaries are defined, along with a variable number of exterior boundaries, Outer1,Outer2, etc. For the plug mesh 3 boundaries are defined; BodySurface, Top, and Plug2mesh.

The `.msh` format

The format to which the mesh is written, is called `.msh` and is one of the formats used by the mesh generation program Ansys ICEM CFD. In tab. 2.1, the structure of the `.msh` format is shown for a single 8 noded cell, with the dimensions 1.0 m x 2.0 m x 3.0 m.

Table 2.1: Example of .msh output (# indicates number)

Printed to `.msh`	**Explanation**
`1128683573`	standard info
`Version number:  5.6s`	standard info
`8 0 0 1 0 1 6`	#nodes 0 0 #elements 0 #domains #boundaries
`0.0000 0.0000 0.0000`	x_1 y_1 z_1 (1. node)
`1.0000 0.0000 0.0000`	x_2 y_2 z_2 (2. node)
`1.0000 2.0000 0.0000`	...
`0.0000 2.0000 0.0000`	...
`0.0000 0.0000 3.0000`	...
`1.0000 0.0000 3.0000`	...
`1.0000 2.0000 3.0000`	...
`0.0000 2.0000 3.0000`	x_n y_n z_n (n. node)
`1 2 3 4 5 6 7 8`	1. element (node numbering)
`1 Domain`	#elements in domain and domainname
`1`	element#
`1 West`	#elements on boundary and boundaryname
`1 5`	element# side#
`1 East`	...
`1 6`	...
`1 Bottom`	...
`1 4`	...
`1 Top`	...
`1 3`	...
`1 South`	...
`1 1`	...
`1 North`	...
`1 2`	...

The cell is shown in fig. 2.11.

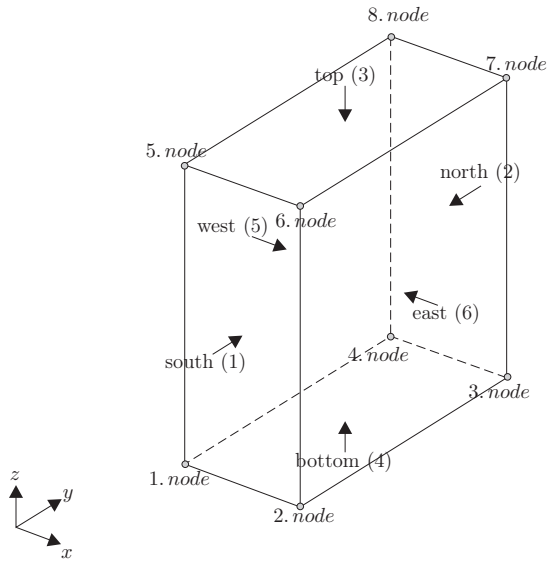


Figure 2.11: Definition sketch, 8 noded cell, numbers in ( ) refer to side number.

2.2 Examples

In this section, a number of examples will be given, showing the effects of certain input parameters in `main.m`:

1. Dissipation
2. Volume averaging
3. Surface node distribution

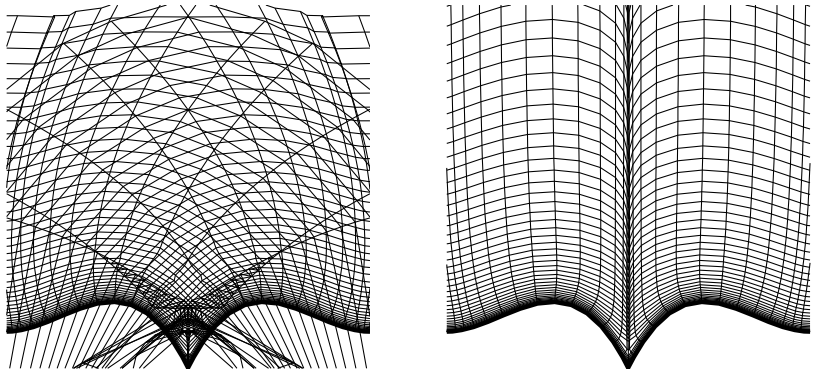
2.2.1 Dissipation

Dissipation is added to the equations in order to deal with a non-smooth geometry that would otherwise cause the mesh lines to coalesce or cross, usually due to a highly concave or convex geometry.
The amount of dissipation (given by eq. (A.18)) at the body surface is controlled by the input parameter `dis`, corresponding to ε in (A.15). A function going from 1 at the body surface and converging towards a constant value ε_2 in the far field is constructed;

$$\alpha_1(k) = \varepsilon((1-\varepsilon_2)(1-\varepsilon_1)^{k-2} + \varepsilon_2) \tag{2.5}$$

The function $\alpha_1(k)$ is used to control how fast the dissipation is turned off as the layers march outwards - but allow for a certain amount of the initially assigned dissipation to remain in the far field.

In fig. 2.12a it is obvious to see that the algorithm fails to produce a mesh that is one-to-one, when different sets of (ξ, η) can lead to the same x, y. The problem occurs when the outwards going lines coalesce and the mesh overlaps. In some cases such a problem can be solved by a higher mesh resolution. However it can be seen from fig. 2.12b that by adding dissipation into the equations, the mesh becomes smooth - and although some crowding of lines occurs in the middle, the mesh does not overlap it self and is one-to-one.



a) No dissipation added (`dis`=0.0). *b)* Dissipation added (`dis`=-0.05).

Figure 2.12: Mesh with and without dissipation.

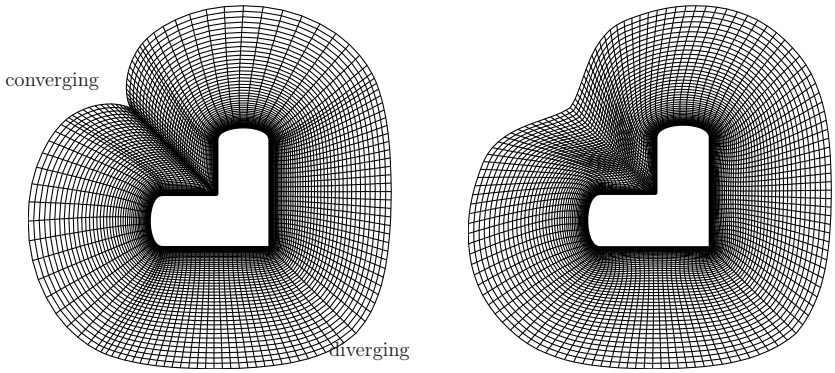
2.2.2 Volume averaging

In order to ensure an evenly sized mesh in the far field, volume averaging is introduced. This is mainly to remove the inherent converging of mesh lines from a concave body surface and the separation of lines from a convex body surface. These effects are illustrated in fig. 2.13a, where a boundary with convex and concave sharp corners is generated.
A function going from 1 at the surface and converging towards 0 in the far field to govern the volume averaging is constructed. The function is given by:

$$\alpha_2(k) = (1 - \varepsilon_3)^{k-2} \tag{2.6}$$

Here, ε_3 is the input parameter `eps`. The volume of a cell is then given by eq. (A.24), where α is replaced by α_2 from eq. (2.6).
By setting `eps`=0.3, it can be seen from fig. 2.13 that the tendency to converge and diverge is eliminated in the far field when the volume averaging is used.

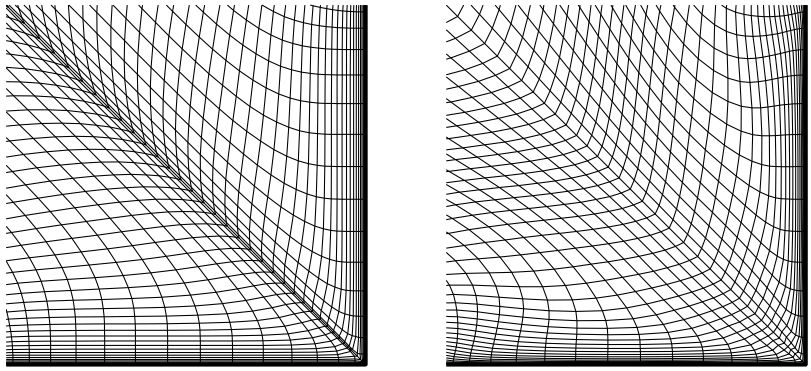
Furthermore it is seen, that with the given volume averaging, a faster convergence of the far field towards a circle is obtained.



a) No volume averaging (`eps`=0.0). *b)* Volume averaging (`eps`=0.3).

Figure 2.13: Mesh with and without volume averaging.

Further inspection of the concave corner, fig. 2.14b, shows how the lines quickly start diverging, whereas the crowding of lines is obvious in fig. 2.14a. Similarly, the lines converge towards a uniform distribution at the convex corner.



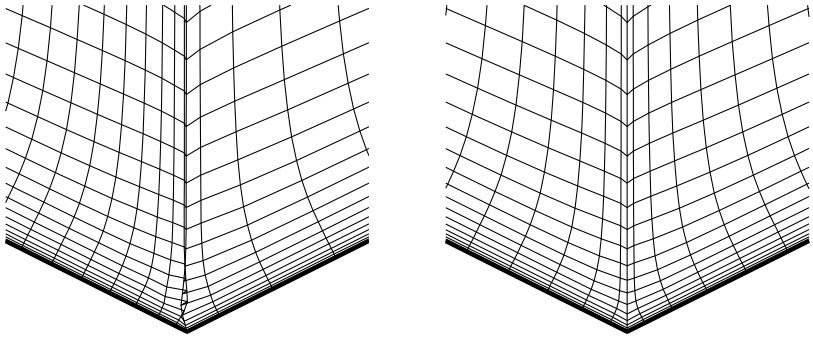
a) No volume averaging (`eps`=0.0). *b)* Volume averaging (`eps`=0.3).

Figure 2.14: Closeup of concave corner.

Depending on the layer spacing near the body surface, `eps` can be adjusted to slow down the volume averaging in the innermost layers. An optimal value of `eps` varies from one mesh to the other.

2.2.3 Surface node distribution

It can cause serious problems for the mesh generation if there's a sudden and strong deviation of node density on the body surface, as can be seen from fig. 2.15a. If they are smoothly distributed however a better mesh is generated, as can be seen from fig. 2.15b.



a) Unevenly spaced nodes *b)* Evenly spaced nodes

Figure 2.15: Effects of smoothly distributed nodes on the body surface.

In this example the smooth distribution of nodes is ensured by use of the stretch functions (generated with `stretchf.m`), allowing a certain side to have a higher or lower node density at the ends, to match that of the adjoining side.

2.3 Alternative Methods for 3D Mesh Generation

In this project, the first step to meshing was to create a 2D mesh well suited for modeling the exterior flow around an enclosed arbitrary boundary like the plane geometry of a high-rise building. To do this, the 2D hyperbolic mesh algorithm proposed by [Steger & Chaussee 1980] was chosen, and implemented into a MatLab program with various additional control-mechanisms to ensure a smooth, orthogonal mesh. After a successful generation of a 2D mesh that met our requirements, the approach was then to extrude this to a 3D mesh, and close it over the top of the building using an additional mesh, a so called "plug", with a one-to-one interface between the two meshes. This approach enforced some heavy constraints, as it was chosen to use a transfinite interpolation scheme to close the plug. Another approach which was considered, was to model the plug mesh using a 2D elliptic mesh algorithm, but this choice was discarded, as creating an additional MatLab program with a complexity similar to that of the first, would require too much time and effort.

The delimitations imposed by the choice of method for closing the 3D mesh are avoided in [Chan & Steger 1992], where a 3D hyperbolic mesh algorithm to produce meshes around highly complex geometries, a space shuttle and a telescope, have been used. The algorithm works in a much similar manner to that of [Steger & Chaussee 1980], merely incorporating a 3^{rd} dimension to the governing equations, and adding an additional equation governing the orthogonality. The meshes produced this way, see [Chan & Steger 1992, p. 192 and p. 203], retain the same orthogonality as the 2D algorithm, and further more incorporates a series of control mechanisms to ensure smooth gridding such as spatially varying smoothing coefficients, metric correction procedures and local treatment of severe convex corners.

The method of [Chan & Steger 1992] is of course more complex, but as can be seen from the results in the article, it allows 3D meshing of exterior flow boundaries with arbitrary highly complex geometries, the major shortcoming of the approach taken in this project.

3
Mesh analysis

This chapter contains an analysis of the behavior of Ansys-CFX when using different mesh types. Three different mesh types are entered into the analysis: unstructured mesh, unstructured mesh with inflation layers and fully structured mesh. The test case is chosen as laminar flow ($Re = 40$) past a circular cylinder with a diameter of $1\,m$.

The analysis shows that there is a significant difference in the computation time between the structured and unstructured meshes with the structured being considerably faster than the unstructured. Also when generating a mesh for use in Ansys-CFX an edge length ratio < 100 *and a volume ratio between two adjacent elements* < 5 *should be secured for the solution to converge normally (CFX-Post manual p. 108).*

In this chapter CFD simulations with both structured and unstructured mesh types on the same geometry are run in Ansys CFX and the results are compared to get an idea of how the mesh types perform during simulations and if there is a significant difference in using a structured mesh instead of an unstructured mesh. The structured meshes used are generated by the hyperbolic mesh program described in chap. 2 while the unstructured meshes are generated by use of Ansys Workbench - CFX-mesh. All CFX files for this analysis are included on the DVD in [*DVD:\Mesh_ Analysis*].

3.1 Test case

The test case for the simulations is the flow around a cylinder. The simulations are restricted to 2D to save computation time. The geometry for the simulations is shown in fig. 3.1.

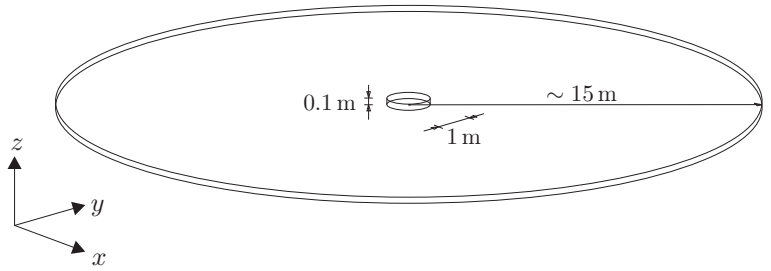


Figure 3.1: Test case with dimensions

The outer boundaries used when setting up the flow in Ansys CFX are shown in fig. 3.2. Problems in the solver can occur when connecting the inlet and outlet directly. To account for this opening boundaries are included between the inlet and outlet. This boundary type allows both flow in and out of the domain. The extent of the opening boundaries in fig. 3.2 should not be considered as precise indications of the extent but only as indicators of the position as the size of the openings are a bit different in especially the simulations with the structured mesh. The main thing is, that in all cases, the openings covers the "top" and "bottom" part of the boundary.

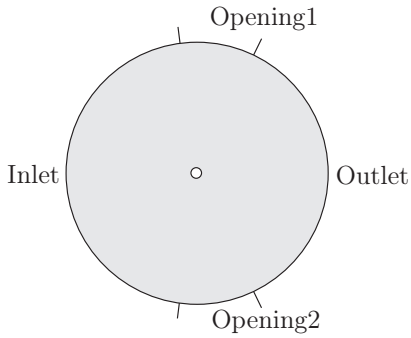


Figure 3.2: Definition of outer boundaries

The results of the conducted simulations can be compared to [Dennis & Chang 1970], where similar flows have been computed. The results in [Dennis & Chang 1970] are based on $Re = 40$. To get the same Reynolds number in the simulations the predefined parameters for air in Ansys CFX are altered. The values used are listed below.

$$\begin{aligned} u_\infty &= 1\ \tfrac{\mathrm{m}}{\mathrm{s}} \\ \rho &= 1\ \tfrac{\mathrm{kg}}{\mathrm{m}^3} \\ \mu &= 0.025\ \tfrac{\mathrm{kg}}{\mathrm{m\,s}} \\ Re &= \frac{\rho D u}{\mu} = 40 \end{aligned} \tag{3.1}$$

The parameter used to check the results is the dimensionless surface pressure, $P(\theta)$, determined by:

$$P(\theta) = \frac{P_0 - P_\infty}{\frac{1}{2}\rho u_\infty^2} \tag{3.2}$$

where

P_0	is the pressure on the cylinder
P_∞	is the pressure in the freestream far from the cylinder
ρ	is the fluid density (See eq. (3.1))
u_∞	Is the freestream velocity in the x-direction (See eq. (3.1))

The pressure in the free stream is equal to the reference pressure from table 3.5 but by default Ansys CFX determines the relative pressure deviation from the reference pressure unless the user defines otherwise. Therefore the value extracted from Ansys CFX is equal to the nominator in eq. (3.2).

It has to be noted that the results from [Dennis & Chang 1970] are extracted from a figure where it is relatively difficult to make an accurate reading which should be kept in mind when reading the following. The results from [Dennis & Chang 1970] are introduced when dealing with the results of the simulations.

3.2 Unstructured mesh

As mentioned earlier the unstructured mesh is generated by use of Ansys CFX-Mesh which is an application in Ansys Workbench. To make sure that the outer boundary of the domain has no influence on the flow around the cylinder, the boundary is made up of a circle with a diameter 30 times larger than the cylinder with the same center.

To generate the mesh the geometry is loaded into Ansys CFX-Mesh. The mesh is generated by specifying a maximum constant element size wanted and by overwriting this overall scale in regions where necessary. Fig. 3.3 shows a screen cap of the program interface.

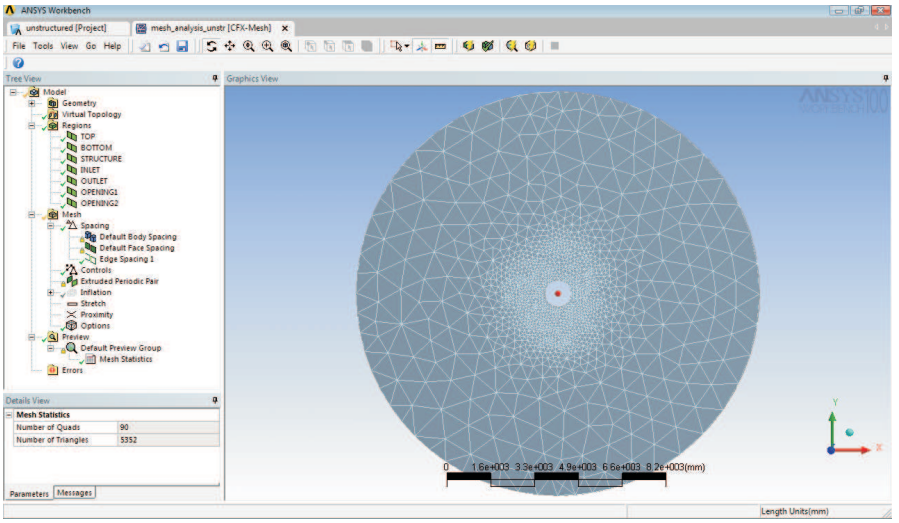


Figure 3.3: CFX-Mesh interface

The mesh specifications are defined under the point *mesh* in the structure tree. Under each of the elements different parameters can be set. An elaboration of the elements under the point *mesh* in the structure tree is shown in fig. 3.4. A specific value in the figure means that this value is used in all the unstructured meshes generated. Values which are changed during the generations are marked

with "variable". In the figure, *Inflation* is marked as this option is not used initially but taken into account later on.

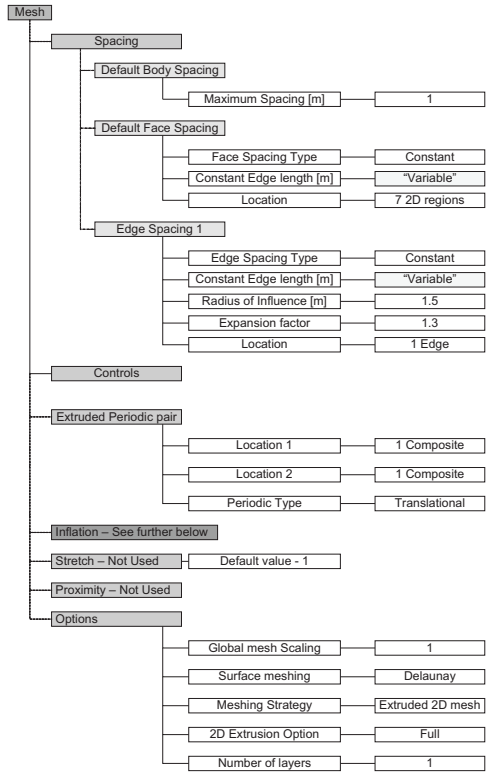


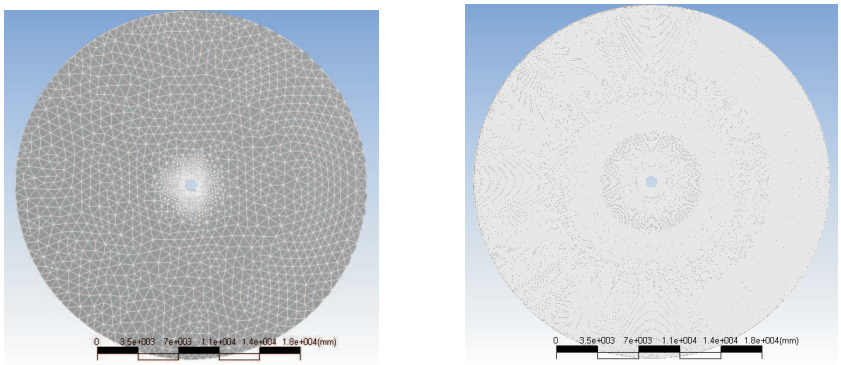
Figure 3.4: Structure tree

The variable parameters indicated in fig. 3.4 are listed in tab. 3.1 where the value used for the respective mesh given by the file-name is indicated. As seen the spacings are halved for each new mesh.

Table 3.1: Variable parameters for unstructured mesh generation

Mesh	Face Spacing Const. Edge Length [m]	Edge Spacing Const. Edge Length [m]	Nodes in simulation
unstr1.gtm	0.80	0.080	3914
unstr2.gtm	0.40	0.040	11950
unstr3.gtm	0.20	0.020	33894
unstr4.gtm	0.10	0.010	167686
unstr5.gtm	0.05	0.005	728360

To illustrate the effect of the parameters the meshes *unstr1.gtm* and *unstr4.gtm* are shown in fig. 3.5.



a) Coarse mesh (unstr1.gtm) *b)* Dense mesh (unstr4.gtm)

Figure 3.5: Illustration of generated unstructured meshes

3.2.1 Inflation layers

When using inflation layers the settings shown in fig. 3.6 are used. As in fig 3.4 the values changed for each mesh are marked with *"Variable"*.

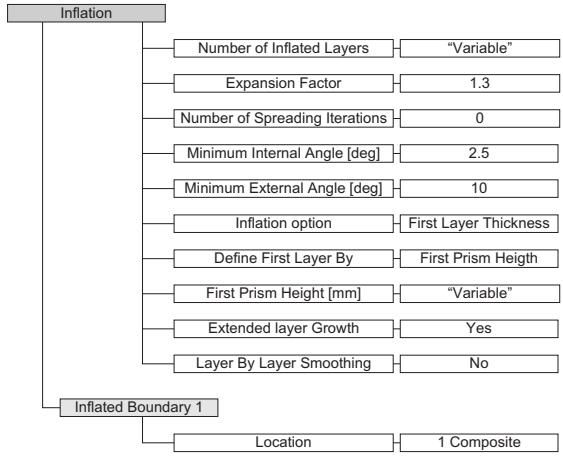


Figure 3.6: Variable parameters for inflation layers

The parameters for each set of inflation layers are listed in tab. 3.2. The parameters used to generate the unstructured mesh outside the inflation layers are

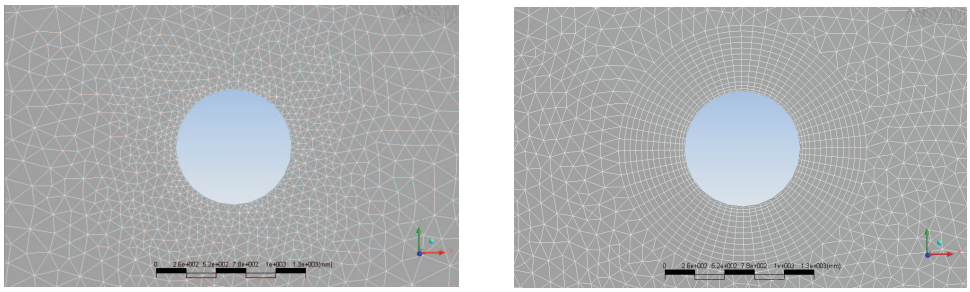
the same as listed in tab 3.1. The connection between the full unstructured mesh and the matching unstructured mesh with inflation layers can be seen by the file name where only *_infl* have been added.

Table 3.2: Variable parameters for inflation layers

Mesh	Number of Inflated layers	First Prism heigth [mm]
unstr1_infl.gtm	7	40
unstr2_infl.gtm	13	20
unstr3_infl.gtm	25	10
unstr4_infl.gtm	50	5

The number of inflation layers are determined by the fact that the maximum number of layers possible are 50. To construct the inflation layers it was intended to halve the first prism height and doubling the number of layers for each successive mesh. The limitation in the maximum number of layers caused the number of layers to have a non regular sequence. This resulted in the fact that the inflation layer did not have the same total height for all meshes.

The influence of using inflation layers can be seen in fig. 3.7 where a close up of the region around the cylinder is shown. The meshes shown are *unstr2.gtm* and *unstr2_infl.gtm*.



a) Without inflation layers (unstr2.gtm) *b)* With inflation layers (unstr2_infl.gtm)

Figure 3.7: Unstructured mesh with and without inflation layers

3.3 Structured mesh

The structured meshes are generated by the program described in chap. 2. From the most coarse mesh the number of nodes on the boundary and the number of marching layers are doubled. It is not possible to control the mesh to give a diameter of precisely 30 m as it is the case for the unstructured meshes. The

mesh is therefor generated as close to this as possible which is reach by specifying an approximate height of `L`= 26 m which give a diameter between $29 - 30$ m. It should here be noted that this method relates to the first version of the mesh generation program where an approximate extent of the mesh is given. The final version uses `L` as the height of the first cell. More about this later.

The *.m* files used to generate the meshes can be found on the attached DVD in [*DVD:\Mesh_ Analysis\Structured_ Mesh*]. The constant parameters used are listed in tab 3.3.

Table 3.3: Constant parameters for structured mesh generation

Parameter	Value	Parameter	Value
`mode`	0 (cyclic mesh)	`dis`	0
`BREAKS`	[1 41]	`eps`	0
`a1`	$[0\,;\,-1]^{\mathrm{T}}$	`eps2`	0
`b1`	$[0\,;\,-1]^{\mathrm{T}}$	`eps3`	0
`x1(1)`	[0 1]	`H`	0.1
`y1(1)`	[0 1]	`as`	2.0
`a2`	[1]	`bs`	0.2
`b2`	[1]	`ntotal`	2
`L`	26	`rot`	0
`a3`	Inf	`nstart`	1
`b3`	6.1		

The vectors defining the outwards layer distribution, `x2` and `y2`, are specified below:

$$\begin{aligned} \texttt{x2} &= [0\ 0.1\ 0.2\ 0.3\ 0.4\ 0.5\ 0.6\ 0.7\ 0.8\ 0.9\ 1.0] \\ \texttt{y2} &= [0\ 0.001\ 0.008\ 0.027\ 0.064\ 0.125\ 0.216\ 0.343\ 0.512\ 0.729\ 1.000] \end{aligned} \tag{3.3}$$

Later in this chapter problems with the initial structured meshes generated by the settings shown above will be explained. The outwards layer distribution will be changed and new structured meshes are generated. More about this in subsec. 3.5.3.

In tab. 3.4 the parameters that are variable are specified which as mentioned are the number of nodes on the boundary and the number of marching layers. The specific mesh is given by the file name.

Table 3.4: Variable parameters for structured mesh generation

Mesh	Nodes on boundary	Marching layers	Nodes in simulation
struc1.msh	40	40	3200
struc2.msh	80	80	12800
struc3.msh	160	160	51200
struc4.msh	320	320	204800

As an example of a structured mesh the mesh generated by *struc1.m*, which is the most coarse mesh, is shown in fig. 3.8.

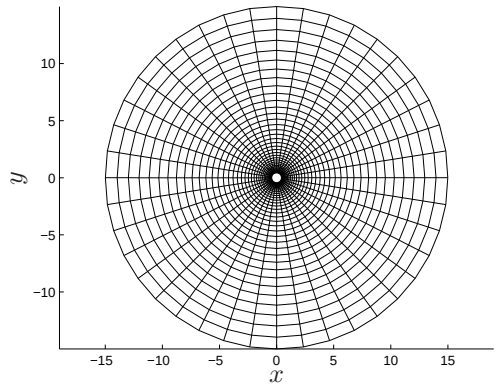


Figure 3.8: Example of structured mesh (struc1.msh)

3.4 CFD input and Definitions

In this section the flow parameters and boundary conditions for the simulations are presented. In this case the simulation type is a steady state laminar flow.

3.4.1 Domain Parameters and Global Initial Conditions

The parameters defining the flow in the domain and the global initial conditions used as input in Ansys CFX are listed in tab. 3.5. Also the parameters defined as solution criteria are specified.

Table 3.5: Domain parameters and global initial conditions

Domain Parameters			
General options		Fluid Models	
Basic Settings		**Heat transfer Model**	
- Location	Assembly	- Option	None
- Domain type	Fluid Domain		
- Fluid List	Air Custom		
- Coord frame	Coord 0		
Domain Models		**Turbulence Model**	
- Ref. Press	1 [atm]	- Option	None (Laminar)
Buoyancy			
- Option	Non Buoyant		
Domain motion			
- Option	Stationary		
Global Initial Conditions		**Solver Control**	
Initial Conditions		**Advection Scheme**	
- Velocity Type	Cartesian	- Option	High Resolution
Cart. Vel. Components		**Convergence Control**	
- u	1 [m s^-1]	- Timescale Control	Physical Timescale
- v	0 [m s^-1]	- Physical Timescale	1 [s]
- w	0 [m s^-1]	- Max Iterations	100
Static pressure		**Convergence Criteria**	
- Option	Automatic	- Residual Type	RMS
		- Residual Target	1.E − 4

3.4.2 Boundary Conditions

The boundary conditions used for the simulations are listed in tab. 3.6.

Table 3.6: Boundary conditions

Basic Settings		**Boundary Details**	
STRUCTURE:			
- Boundary type	Wall	**Wall Influence on Flow**	
- Location	Structure	- Option	No Slip
INLET:			
- Boundary Type	Inlet	**Flow Regime**	
- Location	INLET	- Option	Subsonic
		Mass and Momentum	
		- Option	Cart. Vel. Components
		- u	1 [m s^-1]
		- v	0 [m s^-1]
		- w	0 [m s^-1]
OUTLET:			
- Boundary Type	Outlet	**Flow Regime**	
- Location	OUTLET	- Option	Subsonic
		Mass and Momentum	
		- Option	Average Static Pressure
		- Relative Pres.	0 [Pa]
		Pressure Averaging	
		- Option	Average Over Whole Outlet
OPENING:			
- Boundary Type	Opening	**Flow Regime**	
- Location	OPENING1, OPENING2	- Option	Subsonic
		Mass and Momentum	
		- Option	Opening Pres. and Dirn
		- Relative Pres.	0 [Pa]
		Flow Direction	
		- Option	Normal to Boundary Condition
TOP/BOTTOM:			
- Boundary Type	Symmetry		
- Location	Bottom, Top		

3.5 Results

In this section the results of the simulations are presented. The mesh types are treated seperately and where possible the previous results are entered into a comparison.

Subsec. 3.5.3 which deals with the result with the structured mesh includes an analysis of some numerical error which occurred when running the simulations with the initially defined meshes from sec. 3.3.

3.5.1 Unstructured Mesh

All the simulations with the fully unstructured meshes showed the same characteristics. Convergence was reached relatively quickly with a maximum of 32 iterations. The number of iterations decreased with increasing number of nodes in the simulations. The computation time vs. the number of nodes in the computations is plotted in fig. 3.9. For comparative reasons the computation time have been normalized to the computation time for one node per. iteration. This is done because the simulations does not contain the same number of nodes when comparing to the other mesh types. The normalized time is determined by:

$$t_{node} = \frac{t_{10} - t_2}{8 \cdot n_{node}} \tag{3.4}$$

where

t_{10} is the starting time for iteration 10
t_2 is the starting time for iteration 2
8 is the number of iterations between t_{10} and t_2.
n_{node} is the number of nodes in the present simulation

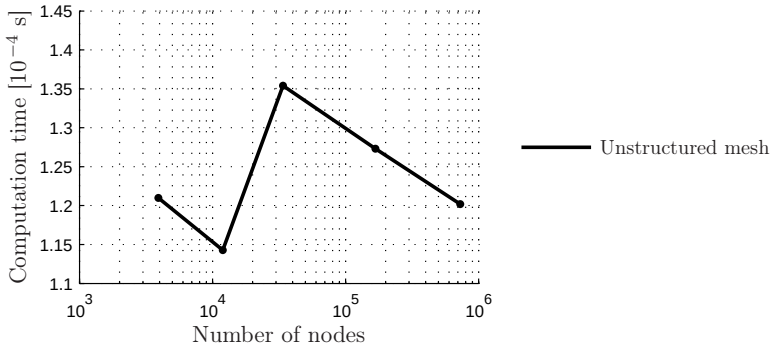
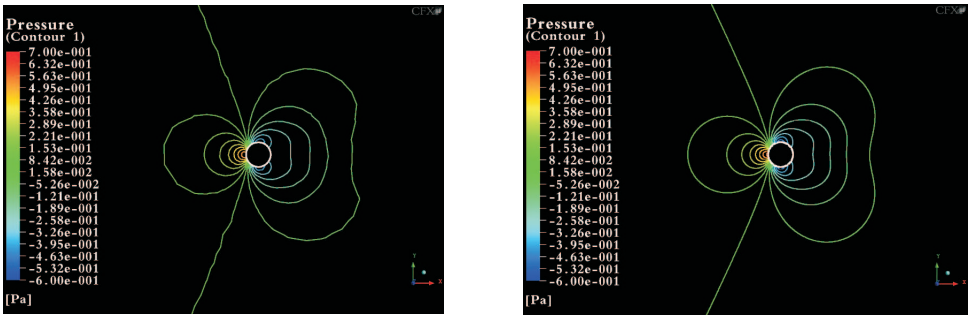


Figure 3.9: Number of nodes vs. computation time for one node per iteration, unstructured mesh.

The simulations were run on a cluster at Aalborg University. The time used is therefor the CPU-time on this cluster. It was found that the computation time on the cluster was shorter than on a laptop (1.80 GHz Dual Core with 2046 MB RAM). No further analysis of this difference have been made.

The plot was expected to be more smooth than the one shown in fig. 3.9. The reason for this has not been investigated further. This is also the case for the plot for the simulations with unstructured meshes with inflation layers as shown later on.

The pressure isobars in the area around the cylinder is shown for the simulations with the most coarse and most dense mesh in fig. 3.10. A close look at fig. 3.10a shows, that the pressure isobars for the most coarse mesh are not smooth. Other than this there is not much difference in the pressure distribution.



a) Coarse mesh (unstr1.gtm) *b)* Dense mesh (unstr5.gtm)

Figure 3.10: Pressure on cylinder for simulations with most coarse and most dense unstructured mesh

When increasing the number of nodes the simulations should converge towards the same result. The final value of the force in the x-direction on the cylinder for each simulation is listed in tab. 3.10.

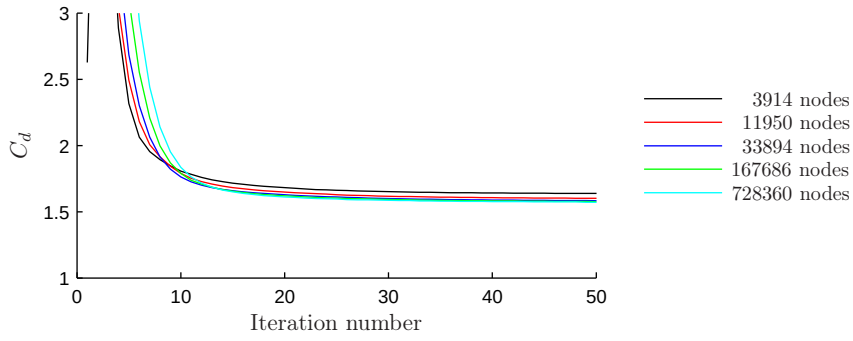
Table 3.7: Force on cylinder for simulations with unstructured mesh

Nodes in simulation	$F_{total}\left[\frac{\mathrm{N}}{\mathrm{m}}\right]$	$F_{press}\left[\frac{\mathrm{N}}{\mathrm{m}}\right]$	$F_{visc}\left[\frac{\mathrm{N}}{\mathrm{m}}\right]$
3914	0.8240	0.5286	0.2955
11950	0.8080	0.5137	0.2942
33894	0.8030	0.5174	0.2855
167686	0.7930	0.5121	0.2809
728360	0.7840	0.5111	0.2724

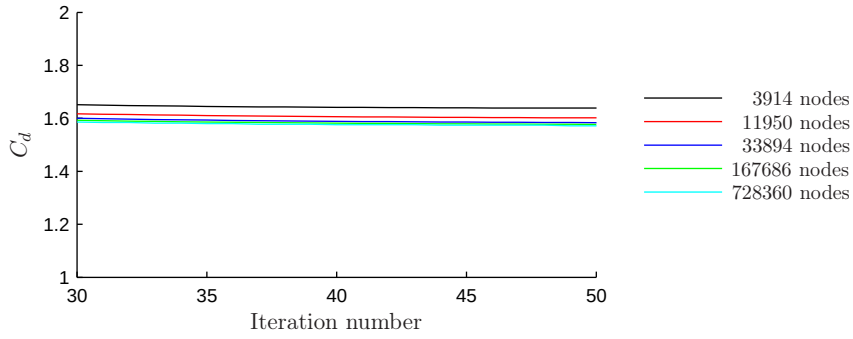
It is seen from tab. 3.10 that the simulations do not fully converge towards a constant value. The drag coefficient, C_d is calculated by:

$$C_d = \frac{F_{total}}{\frac{1}{2}\rho u^2} \tag{3.5}$$

The drag coefficient is determined for the entire simulation and the progress is illustrated in fig. 3.11a for all five simulations. Fig. 3.11b shows a closeup of the last 20 iterations. It is evident from fig. 3.11 that the simulations are converging but they do not fully converge.



a) Overall progress



b) Close-up of final 20 iteration steps

Figure 3.11: Progress of drag coefficient, unstructured mesh

To check and also verify the results the pressure distribution is compared to the values from [Dennis & Chang 1970] as earlier mentioned. The dimensionless surface pressure for all five simulations is determined by eq. (3.2) and plotted in fig. 3.12 along with the results from [Dennis & Chang 1970].

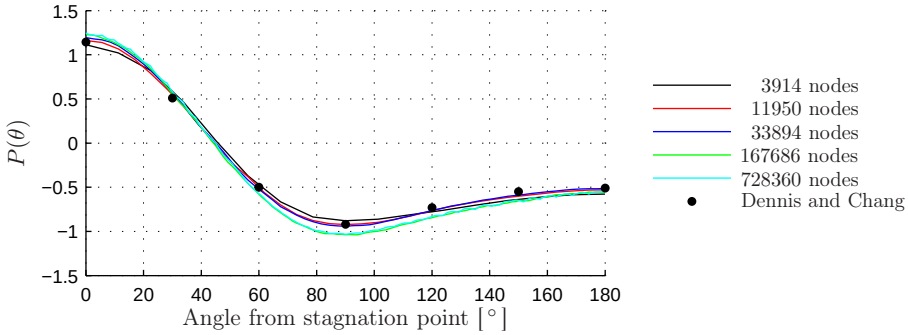


Figure 3.12: Dimensionless surface pressure, unstructured mesh

From fig. 3.12 it is seen that the simulations that fits the data from [Dennis & Chang 1970] closest are with the coarse meshes. This corresponds well with the fact that the data from [Dennis & Chang 1970] are obtained with a relatively coarse mesh as well. Fig. 3.12 also shows that the simulation which deviates the most is with the finest mesh. This behavior was not expected. More about this later.

3.5.2 Unstructured mesh with inflation layers

The simulations with unstructred meshes with inflation layers also converged relatively quickly with a maximum of 31 iterations. The computation time vs. the number of nodes in the computations is plotted in fig. 3.13. For comparative reasons the data for the simulations with the fully unstructured mesh (fig. 3.9) are included. The total computation time have been normalized to the computation time for one node per iteration.

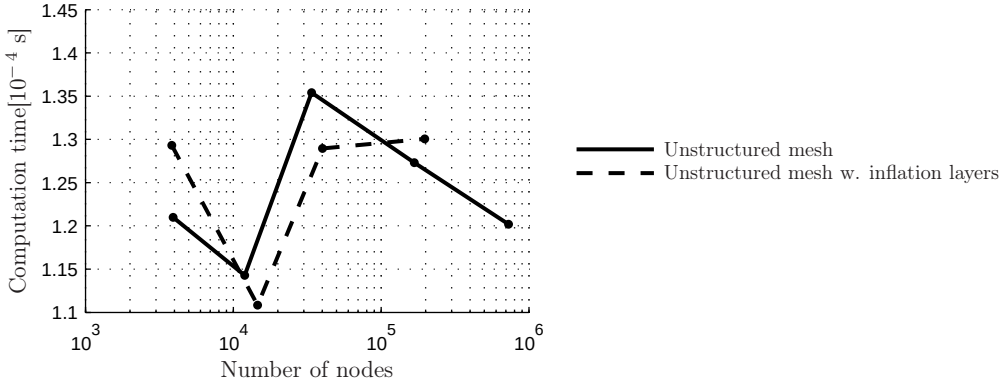


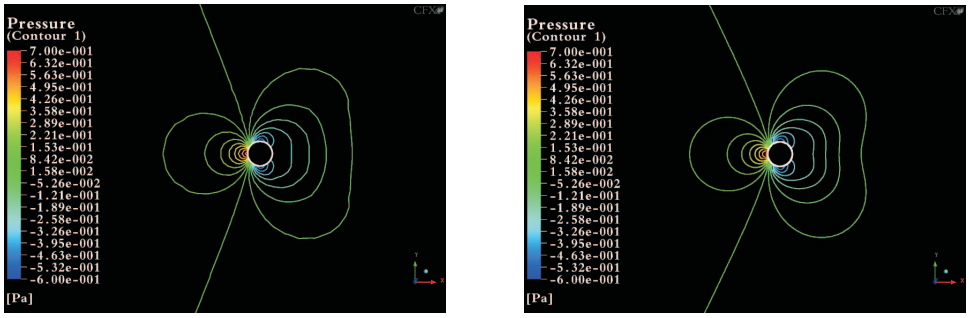
Figure 3.13: Number of nodes vs. computation time, unstructured mesh with and without inflation layers

The reason that only four and not five simulations with inflation layers have been run is because of the lack of computer power necessary to generate the mesh in CFX-Mesh. The program crashed numerous times when checking for intersecting inflation layers.

Inspection of fig. 3.13 shows some difference in the computation time but the relation between the two curves are unclear. This could indicate, that the inclusion of prismatic layers have no real influence on the computation time. The number of prismatic elements are less than 9 % of the total number of elements in all simulations and the amount of prismatic elements should be much larger to see any clear relation at least for these simulations. As it will be shown later, when dealing with the simulations with the fully structured meshes, the computation time will decrease significantly when using fully structured meshes with only prismatic elements.

The main reason for using inflation layers is to capture the large gradients in the boundary layer flow and still keep the number of elements as low as possi-

ble to save computation time. In fig. 3.14 pressure isobars for the simulations with the most coarse and most dense unstructured mesh with inflation layers are shown. These show no major difference except for the back side where the most dense mesh shows some effect on the pressure from the wake.



a) Coarse mesh (unstr1_infl.gtm) *b)* Dense mesh (unstr4_infl.gtm)

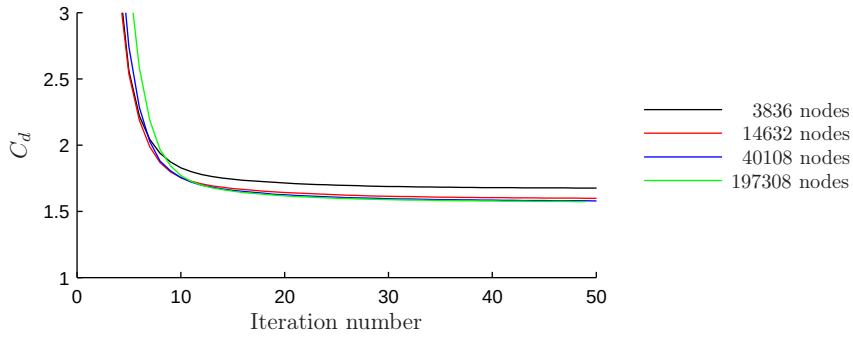
Figure 3.14: Pressure on cylinder for simulations with most coarse and most dense unstructured mesh with inflation layers

Similar to tab. 3.10 the forces on the cylinder at the end of the simulations are listed in tab. 3.8.

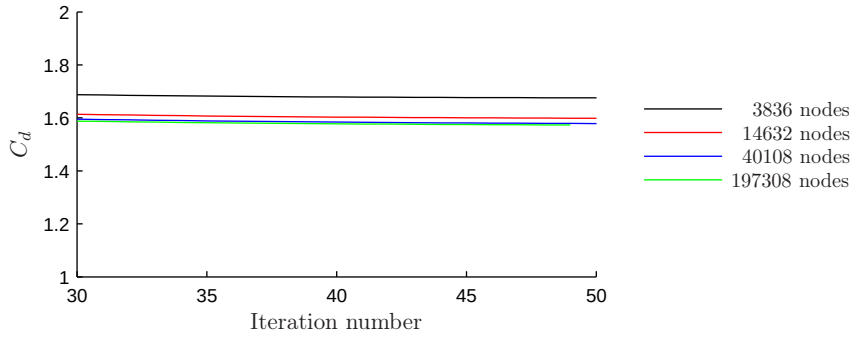
Table 3.8: Force on cylinder for simulations with unstructured mesh with inflation layers

Nodes in simulation	$F_{total}\left[\frac{\mathrm{N}}{\mathrm{m}}\right]$	$F_{press}\left[\frac{\mathrm{N}}{\mathrm{m}}\right]$	$F_{visc}\left[\frac{\mathrm{N}}{\mathrm{m}}\right]$
3836	0.8430	0.5467	0.2966
4632	0.8060	0.5215	0.2846
40108	0.8020	0.5226	0.2793
97308	0.7980	0.5181	0.2799

The drag coefficient is determined by eq. (3.2) and plotted in fig. 3.15. As was the case for the simulations with the fully unstructured meshes, the simulations are converging but not fully converged.



a) Overall progress



b) Close-up of final 20 iteration steps

Figure 3.15: Progress of drag coefficient, unstructured mesh with inflation layers

The surface pressure distribution is determined by eq. (3.2) for all four simulations and plotted in fig. 3.16 along with the results from [Dennis & Chang 1970].

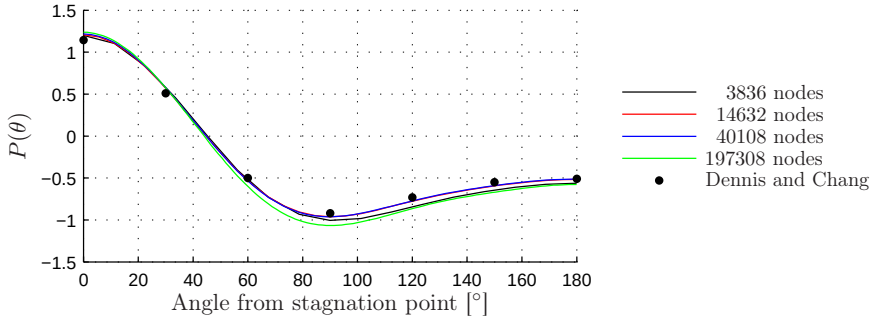


Figure 3.16: Dimensionless surface pressure, unstructured mesh with inflation layers

As is was the case for the simulations with unstructured mesh the best fit is with a relatively coarse mesh. Also in this case the finest mesh is the one which deviates most from the reference results which was not expected.

3.5.3 Structured Mesh

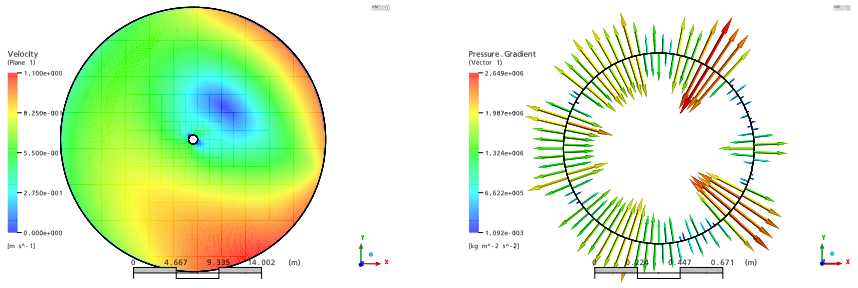
The simulations with the structured mesh caused some problems. The initially generated structured meshes with the settings presented in sec. 3.3 resulted in CFX runs which did not converge or terminated with error. In the following subsection the cause for this error and solution to this is sought for.

Numerical Error with Structured Meshes

The first CFX runs, as mentioned, did not converge after 100 iterations or terminated with the following error message:

```
ERROR # 004100018 has occurred in subroutine FINMES.
| Message:
| Fatal overflow in linear solver.
```

The error message above occurs when the solution is diverging instead of converging. The velocity in the domain and the pressure gradients on the cylinder from one of these simulations are shown in fig 3.17.



a) Velocity profile in domain *b)* Pressure gradients on cylinder

Figure 3.17: Screenshots of results from simulations with initial structured mesh (struc3.msh)

It is easily seen that the results of the simulations are not corresponding to the expected results.

Before choosing the cylinder as test case, the simulations were run on a geometry with only straigth faces. These simulations resultet in relatively fast convergence

and no errors. To see whether the mesh generation had difficulties dealing with curved faces, the geometry with straigth faces was changed to contain one curved face. The initial and altered geometry are shown in fig. 3.18

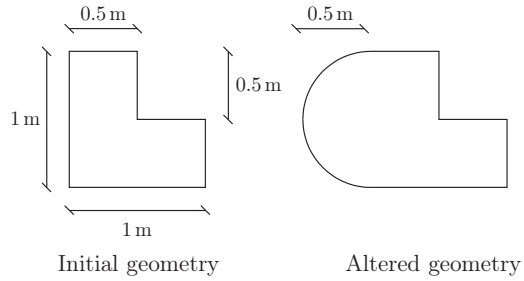


Figure 3.18: Initial and altered geometry

Contrary to the initial geometry, which as mentioned converged quickly, the simulation with the altered geometry resulted in a run which did not converge after 100 iterations. Due to this behavior the emphazise was now turned to the meshes generated. A closer inspection of the mesh *struc1.msh* showed, that the stretch function determining the outwards layer distribution, defined by the vectors in eq. (3.3), resulted in a very thin first layer at the cylinder surface. Fig. 3.19 shows a close-up of the region close to the cylinder surface where the thin layer is visible.

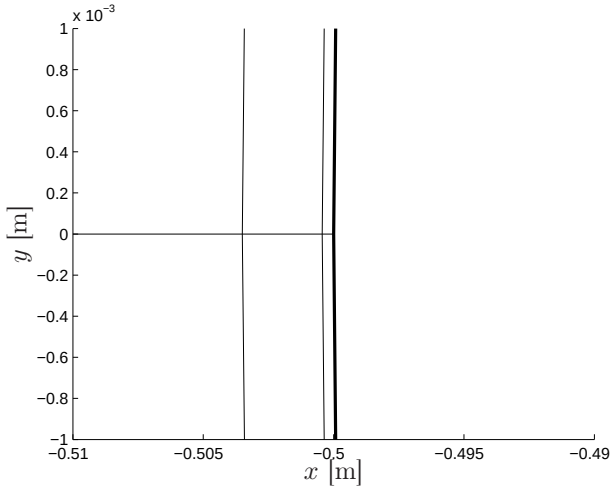


Figure 3.19: Close-up of mesh next to cylinder surface (struc1.msh)

Besides the thin first layer fig. 3.19 also reveals that the height ratio between the first and second layer is relatively large. In the CFX-Post manual a volume ratio between two adjacent elements larger than 5 can cause problems when running the solver. To avoid this the mesh generation program was changed so instead of specifying an approximate extent and a stretch function to define the outwards layer distribution, another method is used. The outwards layer distribution was now defined by specifying the element height of the first layer and a constant expansion factor and the outwards extent limitation was removed. A reference is made to subsec. 2.1.6 where this function is described in more detail.

With this new function it was initially intended to double the expansion factor every time the number of surface nodes and marching layers were halved. The expansion factor was found for the most dense mesh which gave an approximate extent of the mesh of 30 m with a starting height of $L = 10^{-4}$ m. Doubling this factor three times and halving the number of nodes also three times gave the most coarse mesh which only extended about two times the diameter of the cylinder.

Finally the generation of the structured mesh was done by generating the most dense mesh and hereafter deleting every second layer outwards and every second node around the boundary. The most dense mesh was first generated with a height of the first layer of 10^{-4} m. This resulted in a simulation with the most coarse mesh which did not converge after 100 iterations. In the CFX-Post manual it states that an edge length ratio of more than 100 can cause errors when running the solver. An estimate showed an edge length ratio of over 780 in the first layer when using 10^{-4} m. Instead 10^{-3} m was used and hereafter no errors occurred. The parameters used to generate the most dense mesh are listed in tab. 3.9.

Table 3.9: Parameters used to generate new structured meshes

Number of nodes on boundary `n`	Number of marching layers `N`	Expansion factor `expan_fac`
640	641	0.0088

An example of the new structured mesh is shown in fig. 3.20 which is the second most coarse mesh.

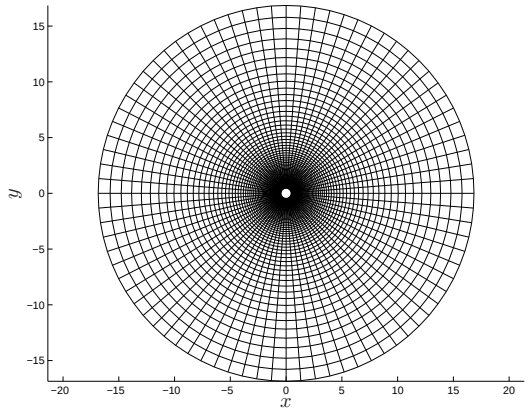


Figure 3.20: Example of final structured mesh

Results from Runs with new Structured Meshes

After generating the five new mesh files the simulations were run. The simulations converged after a maximum of 29 iterations. The computation time have been normalized the same way as in fig. 3.13 and plotted in fig. 3.21 along with the results from the two other mesh types. It is clear that the computation time is significantly smaller for the simulations with structured mesh. As mentioned when dealing with the unstructured mesh with inflation layers which did not show any real time difference compared to the fully unstructured mesh, there is a significant time to save if the amount of prismatic layers is sufficiently large.

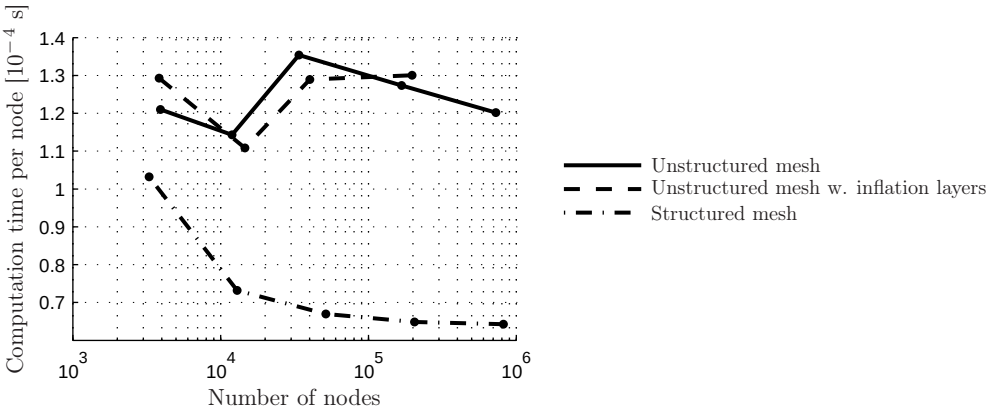
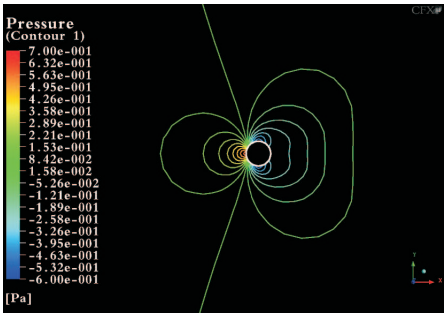
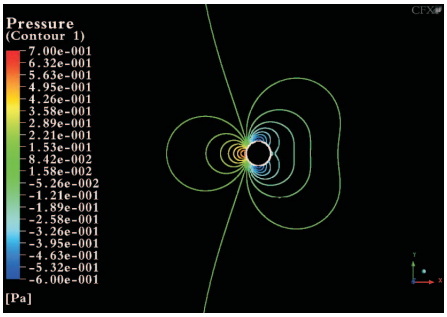


Figure 3.21: Number of nodes vs. computation time per node, all mesh types

The pressure isobars for the simulations with the most coarse and most dense structured mesh are shown in fig. 3.22. A closer look at the two figures shows a difference in the pressure on the back side of the cylinder. Also comparing fig. 3.10b and 3.14b with 3.22b shows a difference in the pressure on the back side. The structured mesh is apparently better at including the effect of the wake behind the cylinder.



a) Coarse mesh (struc1_new.msh)



b) Dense mesh (struc5_new.msh)

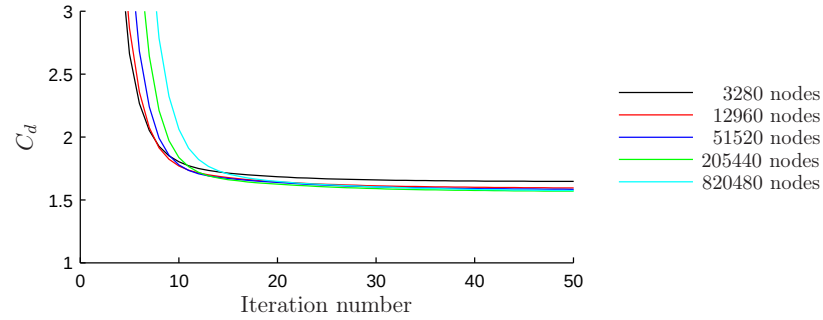
Figure 3.22: Pressure on cylinder for simulations with most coarse and most dense structured mesh

The forces on the cylinder at the end of the simulations are listed in tab. 3.10.

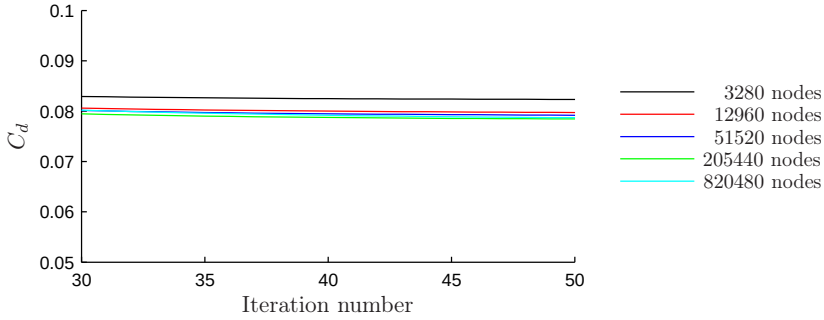
Table 3.10: Force on cylinder for simulations with unstructured mesh

Nodes in simulation	F_{total} [N/m]	F_{press} [N/m]	F_{visc} [N/m]
3280	0.8230	0.5372	0.2857
12960	0.7960	0.5200	0.2760
51520	0.7910	0.5179	0.2727
205440	0.7840	0.5141	0.2698
820840	0.7950	0.5214	0.2736

The drag coefficient is determined by eq. (3.2) and plotted in fig. 3.23. The same behavior as for the two previous mesh types is also present here with the simulations converging but not fully converged.



a) Overall progress



b) Close-up of final 20 iteration steps

Figure 3.23: Progress of drag coefficient, structured mesh

The dimensionless surface pressure is determined by eq. (3.2) for all five simulations and plotted in fig. 3.24. Again the best fit of the data from [Dennis & Chang 1970] are the results from the simulations with the most coarse meshes.

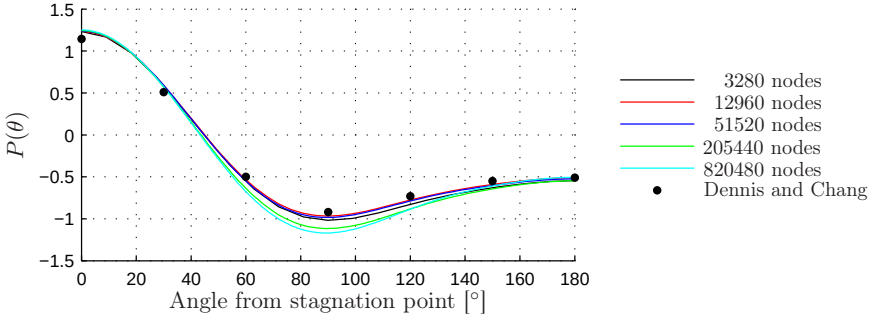


Figure 3.24: Dimensionless surface pressure, structured mesh

As it is the case for all mesh types, the pressure distribution for the one/two most dense meshes deviate from the rest of the simulations and from the reference results.

3.6 Observations

To sum up on the observations during the analysis, the main difference in using structured and unstructured mesh types in Ansys CFX is the computation time. There is a large difference in the necessary computation time when going from a fully unstructured mesh to a fully structured. The inclusion of inflation layers in the unstructured meshes did not show any real improvement of the results and the computation time. This could be due to the ratio of prismatic elements to the total number of elements which was below 9 % in all simulations. The much shorter computation time for the fully structured meshes indicate that there is a lot to gain by including these prismatic layers as long as the ratio between prismatic and unstructured elements become sufficiently high.

The structured meshes introduced some problems in the simulations. By analysis it was found, that the edge length ratio and volume ratio between two adjacent elements are of great importance. In CFX-Post the mentioned limits for these ratios are not listed as maximum ratios but only listed as values which *can* cause problems in the solver. This analysis showed that these ratios should be considered as maximum limits when generating meshes to use in simulations in Ansys CFX.

It was the case for all three mesh types that the simulations did not fully converge and on top of this, the simulations which deviated the most from the reference results was the ones with the finest mesh. This was not at all expected. Supervisor of this project, Niels N. Sørensen, Risø, have conducted simulations with the same settings but with a different solver developed at Risø. The pressure distribution from his simulation with the finest mesh is plotted in fig. 3.25 along with the results from [Dennis & Chang 1970]. Also the results from the simulation with the finest mesh of the three mesh types are included.

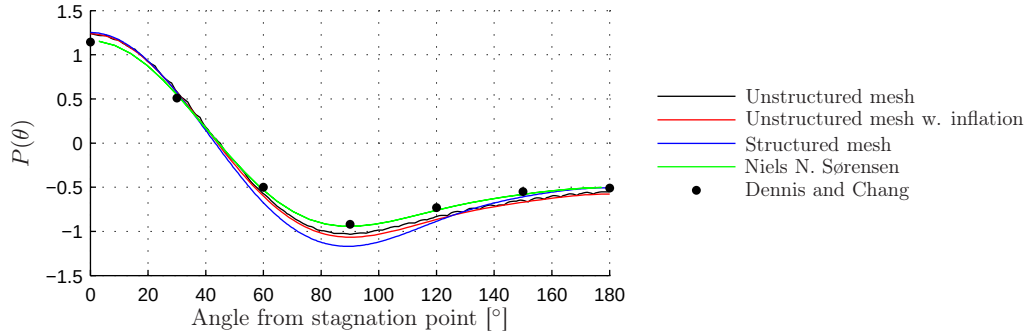


Figure 3.25: Comparison of results from the conducted simulations with results from supervisor Niels N. Sørensen

It is seen from fig. 3.25 that the results from Niels N. Sørensen fits the reference data very well compared to the results from the conducted simulations in this project. The reason for this behavior has not been searched for any further.

4

Analyses of Moving mesh in CFX

In this chapter a problem with moving mesh in Ansys CFX is described, where the mesh orthogonality would deteriorate through a transient run. An analysis of six different functions to govern mesh stiffness is performed. The goal of this analysis is to determine which functions are less likely to cause problems with deteriorating orthogonality in the mesh during transient runs with moving mesh.

The analysis shows that the best way to govern the mesh stiffness is by using either the reciprocal of wall distance as a stiffness parameter, or by using the predefined function "Increase near Boundaries". This conclusion holds only if there are no unforseen problems arising during 3D simulations, that were not visible in the 2D test case.

When conducting CFD simulations where the structure is allowed to deflect, the mesh in the simulation has to be able to obtain the deformation. In this chapter the ability to control the mesh stiffness in Ansys CFX is analyzed. Also it became apparent that a certain problem can occur in CFX when the mesh is moved, being that the mesh is slowly deforming, or folding. This also has to be taken care of so the solver does not terminate before reaching the end of the specified simulation time.

The mesh deformation is in Ansys CFX governed by a displacement diffusion model given by, cf. *ANSYS CFX-Solver Modeling Guide, p. 4*:

$$\nabla \cdot \left(\Gamma_{disp} \nabla \delta\right) = 0 \tag{4.1}$$

where

Γ_{disp}	is the mesh stiffness
δ	is the displacement relative to the previous mesh locations
∇	is the gradient operator

Eq. (4.1) expands to:

$$\frac{\partial}{\partial x}\Gamma_{disp}\frac{\partial \delta}{\partial x} + \frac{\partial}{\partial y}\Gamma_{disp}\frac{\partial \delta}{\partial y} + \frac{\partial}{\partial z}\Gamma_{disp}\frac{\partial \delta}{\partial z} = 0 \tag{4.2}$$

which in the case of $\Gamma_{disp} = constant$ becomes:

$$\frac{\partial^2 \delta}{\partial x} + \frac{\partial^2 \delta}{\partial y} + \frac{\partial^2 \delta}{\partial z} = 0 \tag{4.3}$$

4.1 Test case

The test case for the moving mesh is a cylinder with a diameter of 1 m in a circular domain, like the test case in sec. 3.1. The cylinder is translated in a circular motion and during these cycles of translation the mesh will deform away from the initial mesh. The cyclic motion is governed by the functions dx and dy, controlling the displacement of the cylinder. These functions are given as:

$$dx = 0.1 \cos(1\tfrac{\mathrm{m}}{\mathrm{s}}\, t) - 0.1 \tag{4.4}$$

$$dy = 0.1 \sin(1\tfrac{\mathrm{m}}{\mathrm{s}}\, t) \tag{4.5}$$

Fig. 4.1 shows a sketch of the circular motion of the cylinder, and fig. 4.2 shows an example of the mesh before and after permanent deformations have occurred even though the cylinder has returned to it's initial position.

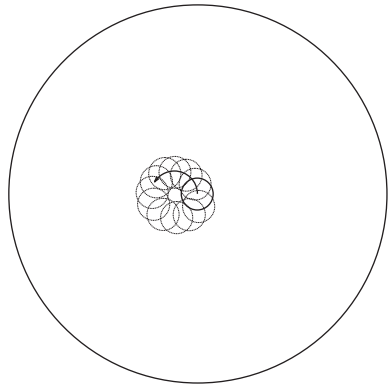
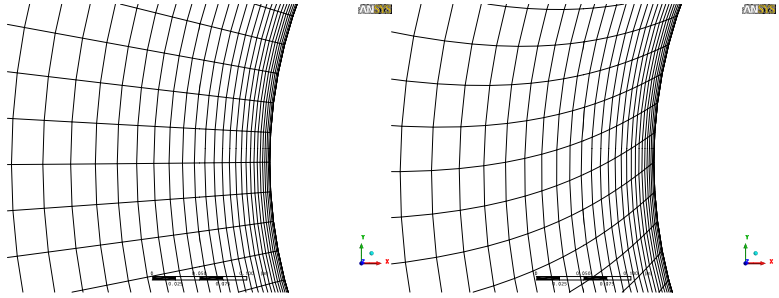


Figure 4.1: Sketch of test case



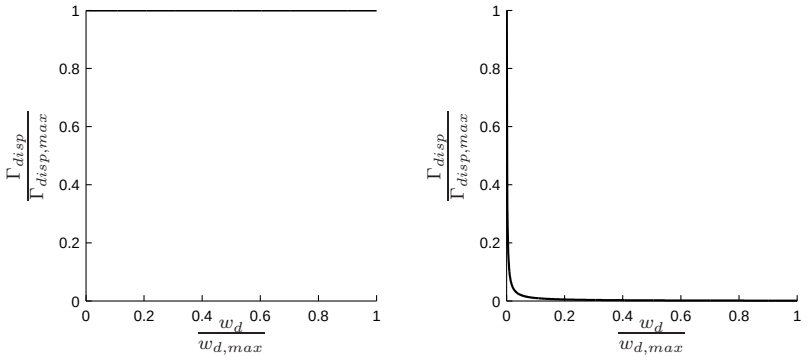
a) Mesh before deformation *b)* Mesh after deformation

Figure 4.2: Mesh before and after deformation

The problems were first encountered while running CFX 10.0 and were so extensive that the deformations would create negative volume elements and prematurely shut down the transient runs. Several different approaches to govern the mesh stiffness in the domain were tried, with varying degrees of success. The problem of the mesh deforming or folding became less severe if the mesh stiffness was controlled in a way, that would ensure a high stiffness in the very small elements close to the cylinder and a lower stiffness away from the cylinder where the elements are much larger.

After installing CFX 11.0 the problems were less severe. Functions for governing mesh stiffness that caused problems in CFX 10.0 worked much better. Still it is desired to compare different ways of governing the mesh stiffness, to determine the optimal method for doing so. Furthermore the effects of the time step size is investigated.

To do this, four different approaches for governing the mesh stiffness are attempted. The functions for these are illustrated in figs. 4.3a - 4.3d, where the mesh stiffness and wall distance have been normalized. Here it is important to remember that a mesh stiffness of 1 does not equal infinite stiffness, but merely the maximum stiffness in the domain. As such, deformations *can* occur in areas with a mesh stiffness of 1.



a) Constant

b) Reciprocal of wall distance

c) tanh function

d) cos function

Figure 4.3: Normalized mesh stiffness as function of normalized wall distance

The tanh and cos functions, S_{tanh} and S_{cos} respectively, are given by eqs. (4.6) and (4.7) where w_d denotes the normalized wall distance.

$$S_{tanh}(w_d) = \begin{cases} \tanh(30\,w_d - 18)^2 & \text{for } w_d < 0.6 \\ \tanh(30\,w_d - 24)^2 & \text{for } w_d > 0.83 \\ 0 & \text{else} \end{cases} \tag{4.6}$$

$$S_{cos}(w_d) = \begin{cases} 1 & \text{for } w_d < 0.2 \\ 0 & \text{for } w_d > 0.9 \\ 0.5\,\cos(\frac{\pi}{0.7}\,(w_d - 0.2)) & \text{else} \end{cases} \tag{4.7}$$

Also, two internal CFX 11.0 functions were tried; Increase near Boundaries and Increase near Small Volumes. Both were used with the standard settings.

As a parameter for the mesh quality, the minimum orthogonality factor has been measured in the surface layer. The mesh orthogonality is in *ANSYS CFX-Solver Modeling Guide p. 371* defined as:

> *"The most relevant measure of mesh orthogonality in ANSYS CFX-Solver is illustrated below. It involves the angle between the vector that joins two mesh (or control volume) nodes (s) and the normal vector for each integration point surface (n) associated with that edge. Significant orthogonality and non-orthogonality are illustrated at ip1 and ip2, respectively."*

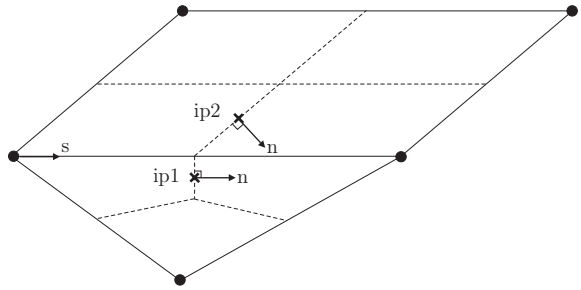


Figure 4.4: Definition sketch of mesh orthogonality

The minimum orthogonality factor is in *ANSYS CFX-Solver Modeling Guide p. 372* defined as:

> *"Minimum of all integration point surface scalar products of unit n and s vectors (i.e., n·s) associated with each control volume."*

An orthogonality factor of 1 defines a fully orthogonal mesh.
Each of the six functions have been tested with a case of 50 cyclic motions. One test using 20 steps for a cycle and one using 40, so that any influence the step size may have will be shown.

In tab. 4.1 a list of the runs made is shown, along with their path on the DVD. For each type of test, there is a 20 step cycle run and a 40 step cycle run.

Table 4.1: Moving mesh test runs

Moving mesh test runs Name	Path to .res and .out files
Constant	DVD\Moving_Mesh\constant\
Reciprocal of wall distance	DVD\Moving_Mesh\one_over_WD\
tanh function	DVD\Moving_Mesh\tanh_function\
cos function	DVD\Moving_Mesh\cos_function\
Increase near Boundaries	DVD\Moving_Mesh\inc_near_bound\
Increase near Small Volumes	DVD\Moving_Mesh\inc_near_small_vol\

For each run there is a result file (.res), and an out file (.out). The transient result files have not been included on the DVD due to excessive usage of disk space (>4 GB per run).

4.1.1 Constant Stiffness

A constant stiffness parameter throughout the domain has proven a very poor choice, as can easily be seen on fig. 4.5.

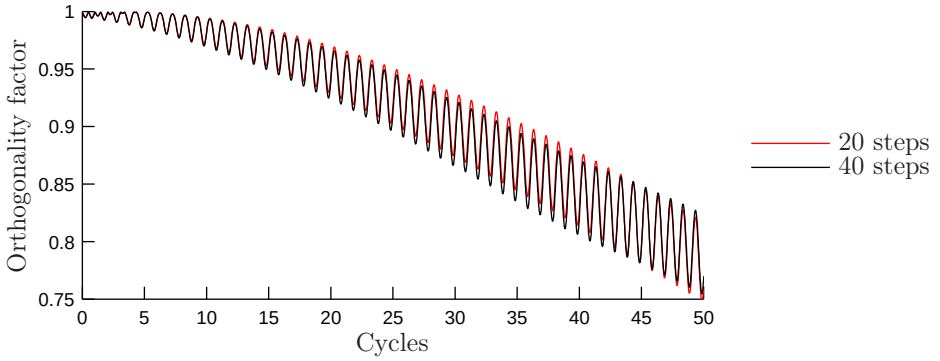


Figure 4.5: Deformation of innermost elements; constant stiffness

The large amplitudes of each cycle shows there are large deformations of the elements in each cycle, and the deteriorating orthogonality factor shows that the permanent deformations are rapidly growing.

There is practically no difference between using 20 step cycles and 40 step cycles.

4.1.2 Reciprocal of Wall Distance

Using the reciprocal of the wall distance as a stiffness parameter shows that there are no deformation of the innermost elements, as seen on fig. 4.6. This is contradictory to the initial tests in CFX 10.0 where this approach gave large deformations.

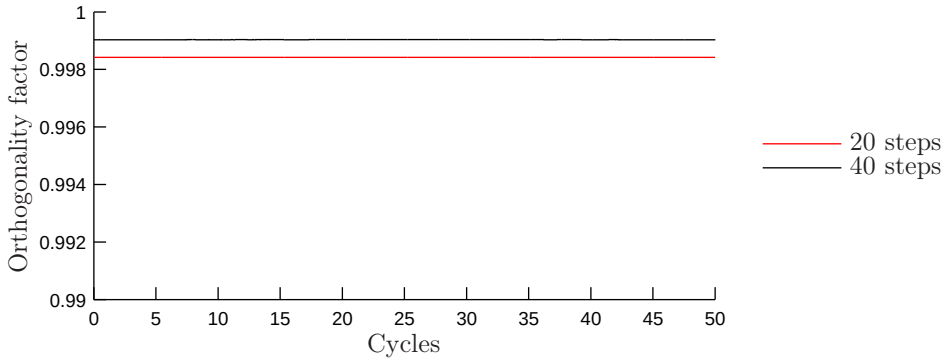


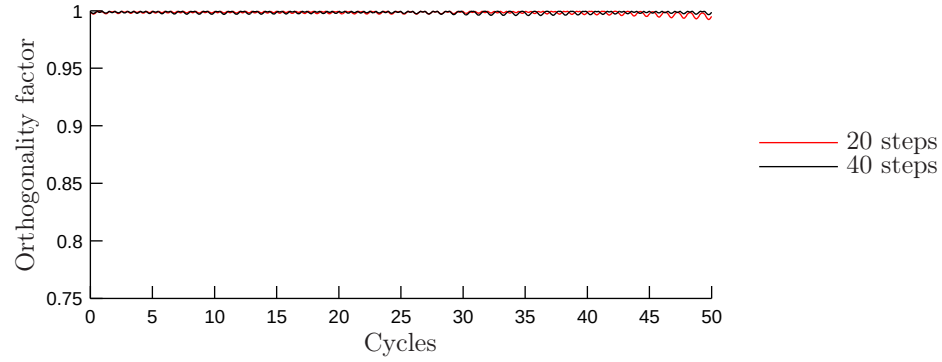
Figure 4.6: Deformation of innermost elements; reciprocal of wall distance

That there is no deformation in the innermost elements suggests that the stiffness here goes towards infinite.

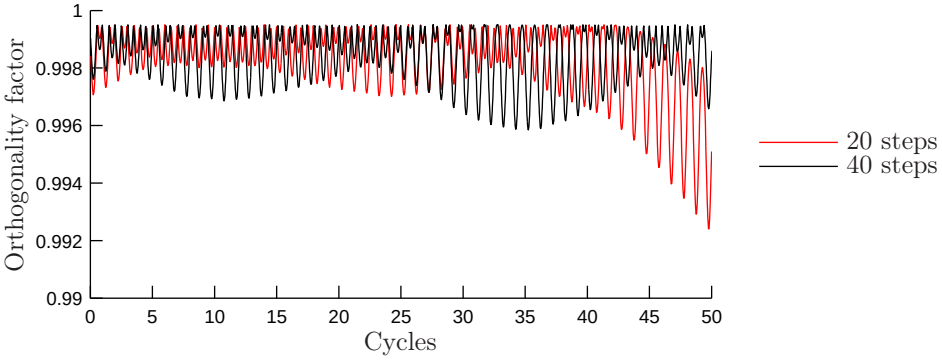
The difference between 20 step cycles and 40 step cycles cannot be explained in any other way than some numerical error, as it is obvious the two simulations start out with a different orthogonality factor, despite using the same mesh.

4.1.3 Tanh Function

Using the tanh function, it can be seen from figs. 4.7a and 4.7b that although there is a clear cyclic behavior of the deformations, they are quite small. It seems however, that there is a slowly deterioration in the orthogonality factor over time.



a) Overview



b) Closeup

Figure 4.7: Deformation of innermost elements; tanh function

There is an obvious difference between using 20 step cycles and 40 step cycles, where the smaller steps causes less deformation over time.

4.1.4 Cos Function

It can be seen from fig. 4.8 that the cos function deforms fast and is certainly not as good as the tanh or reciprocal of wall distance functions.

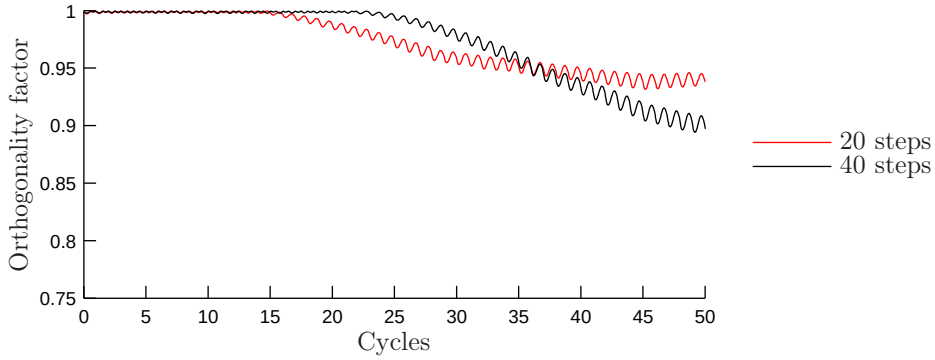


Figure 4.8: Deformation of innermost elements; cos function

There is also an obvious difference between using 20 step cycles and 40 step cycles here, where the 20 step cycles starts deteriorating sooner than using 40 steps, but then flattens out and actually causes less permanent deformation at the end of the run. This behavior is peculiar and probably has to do with some unforseen numerical effects.

4.1.5 Increase near Boundaries

This method is virtually identical to using the reciprocal of wall distance, as can be seen by comparing fig. 4.9 to fig. 4.6.

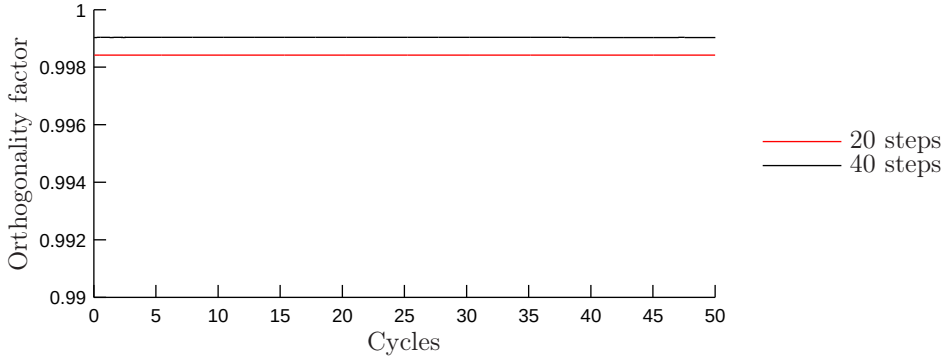
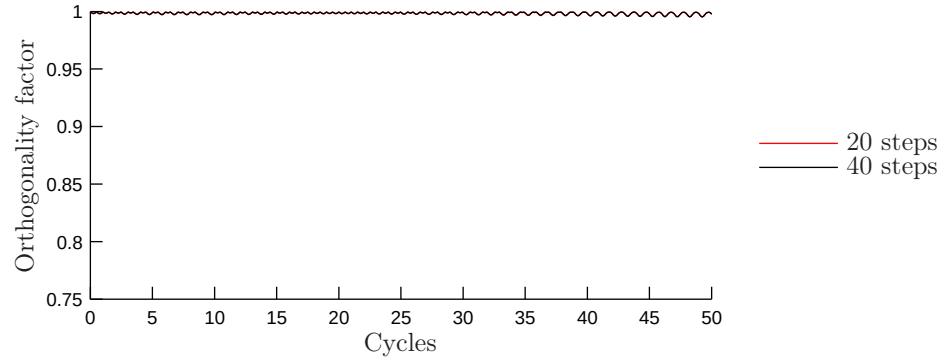


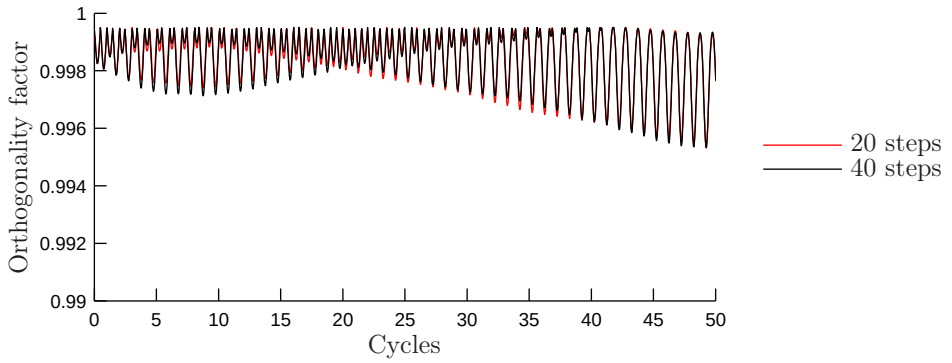
Figure 4.9: Deformation of innermost elements; increase near boundaries

4.1.6 Increase near Small Volumes

This method shows cyclic behavior similar to that of the tanh function. As can be seen from figs. 4.10a and 4.10b, the oscillations cause very little deterioration of the orthogonality over time.



a) Overview



b) Closeup

Figure 4.10: Deformation of innermost elements; increase near small volumes

There seem to be no clear difference between using 20 step cycles and 40 step cycles with this method.

4.1.7 Summary

Figs. 4.11 to 4.16 show the minimum orthogonality angle in the mesh for each of the six methods during one cycle at 0, 90, 180, 270 and 360 degrees rotation. Cycle 26 has been chosen to illustrate this, going from step 1000 to step 1040.

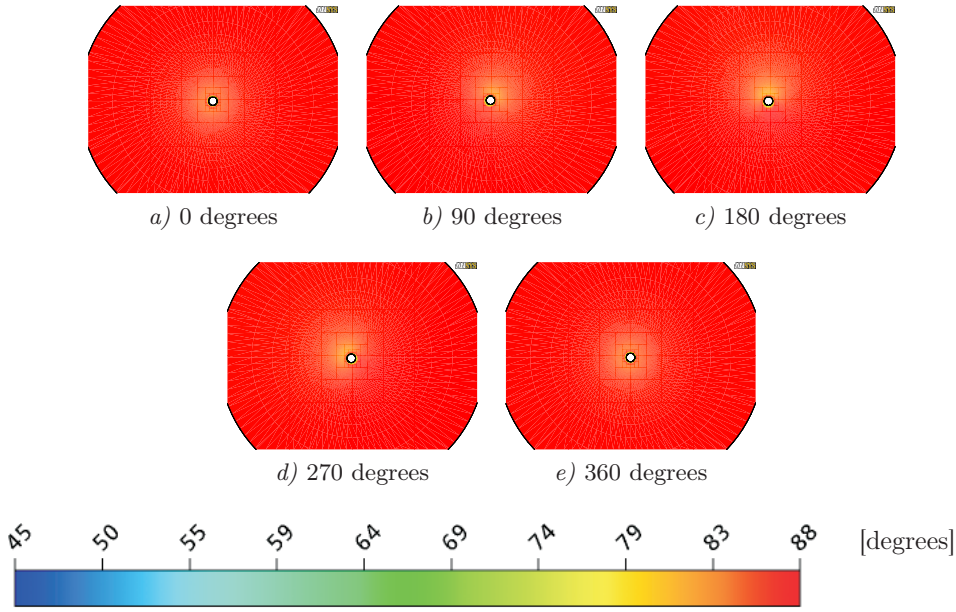


Figure 4.11: Minimum orthogonality angle in mesh, constant stiffness

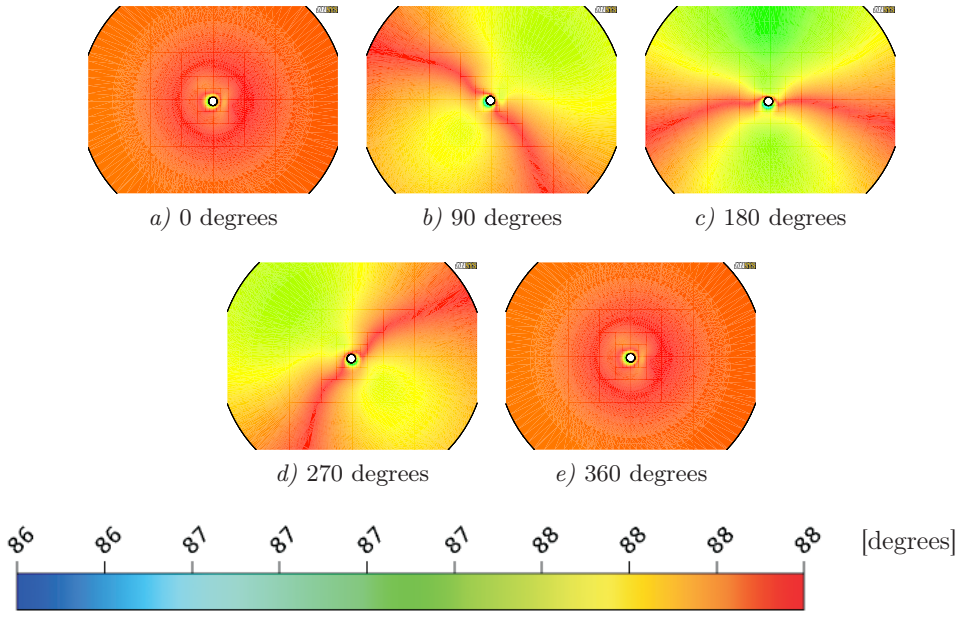


Figure 4.12: Minimum orthogonality angle in mesh, reciprocal of wall distance

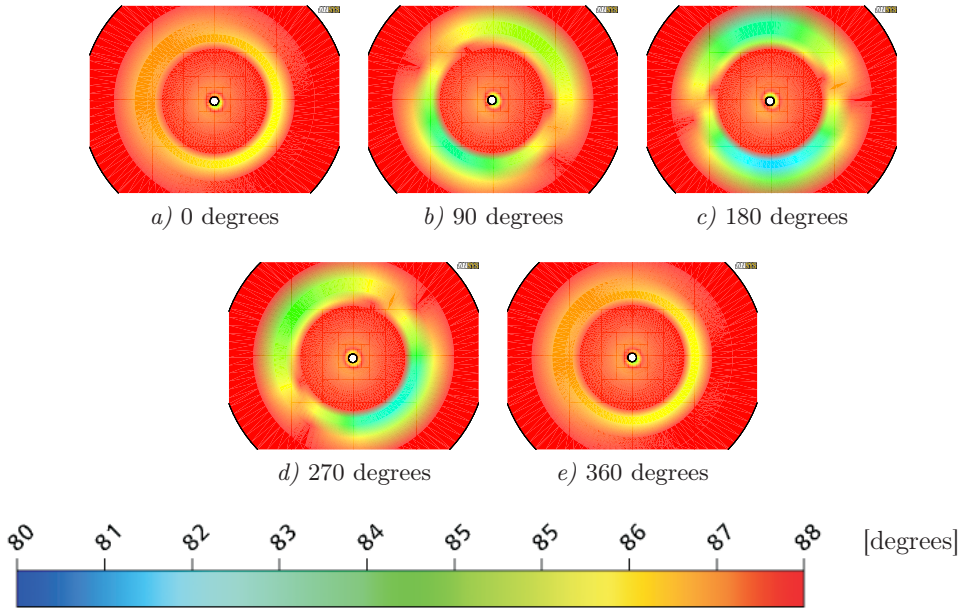


Figure 4.13: Minimum orthogonality angle in mesh, tanh function

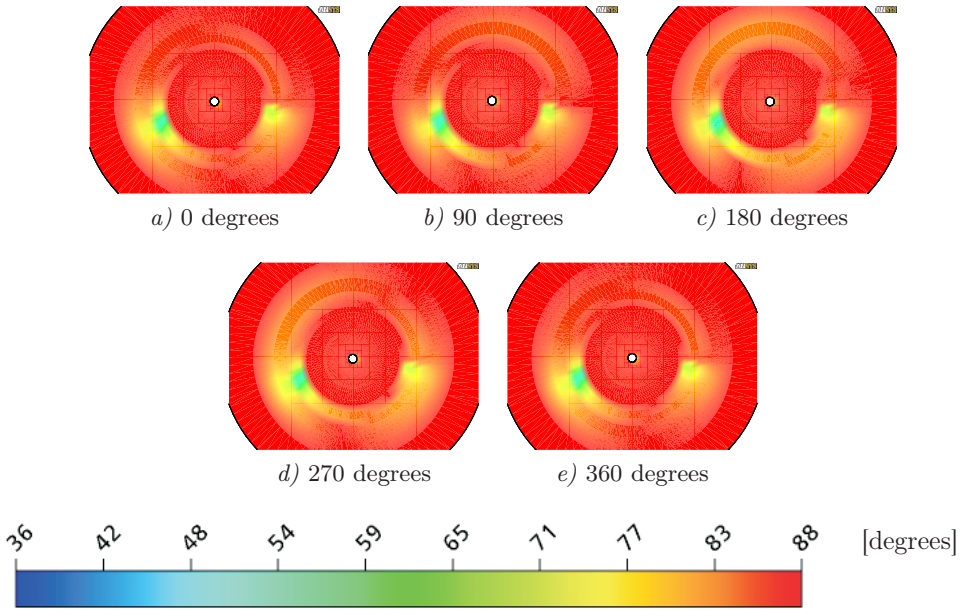


Figure 4.14: Minimum orthogonality angle in mesh, cos function

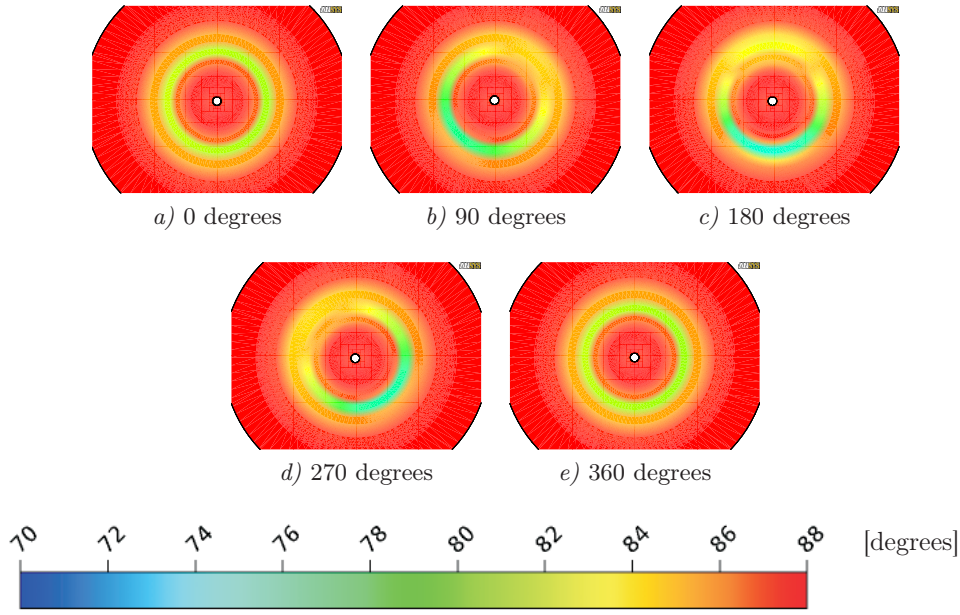


Figure 4.15: Minimum orthogonality angle in mesh, Increase near boundaries

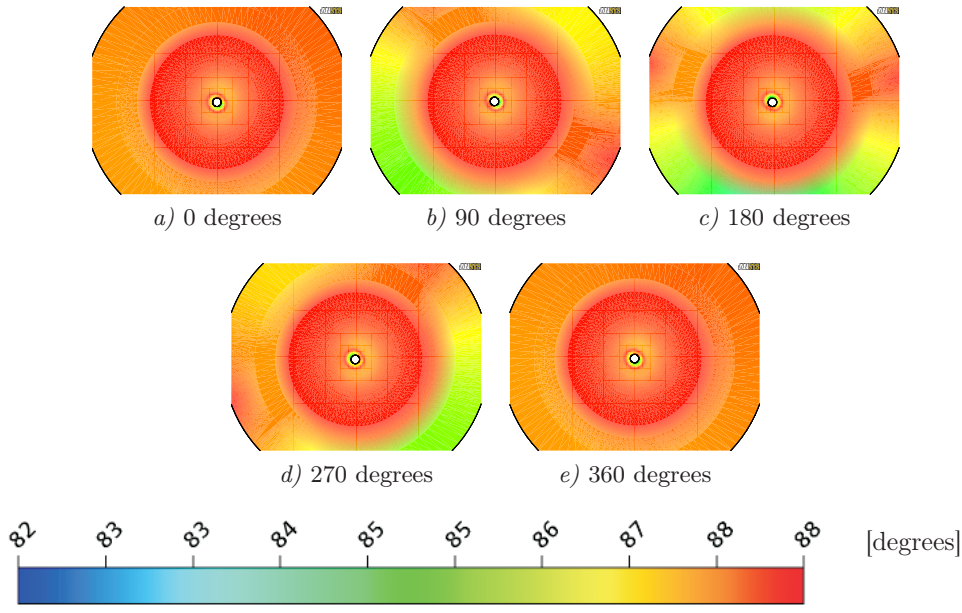


Figure 4.16: Minimum orthogonality angle in mesh, Increase near small volumes

It is evident that as the cylinder moves away from its point of origin, the orthogonality throughout mesh deteriorates, however most of this orthogonality is subsequently reestablished as the cylinder returns to the point of origin.

From figs. 4.5 - 4.10 it can be seen that the two functions which give the most orthogonal elements in the surface layer is either the reciprocal of the wall distance or the increase near boundaries. Both these functions show no deformation of the surface layer at all, and as such no deterioration over time either. This can also be seen from tabs. 4.2 and 4.3 where the minimum orthogonality factors are listed in descending orders.

Table 4.2: Moving mesh results, 20 step cycle

Moving mesh results, 20 step cycle Name	Minimum orthogonality factor
Increase near boundaries	0.9984
Reciprocal of wall distance	0.9984
Increase near small volumes	0.9953
Tanh function	0.9924
Cos function	0.9315
Constant	0.7482

Table 4.3: Moving mesh results, 40 step cycle

Moving mesh results, 40 step cycle Name	Minimum orthogonality factor
Increase near boundaries	0.9990
Reciprocal of wall distance	0.9990
Tanh function	0.9959
Increase near small volumes	0.9953
Cos function	0.8942
Constant	0.7546

When studying the entire mesh however, figs. 4.17 - 4.22, it can be seen that the reciprocal of the wall distance, fig. 4.18b, performs significantly better than increase near boundaries, 4.21b, where a small band shows a lot of permanent deformation. The strange behavior of the cos function with two areas of low orthogonality, fig. 4.20b, seems to be due to some numerically odd behaviour in the solver.

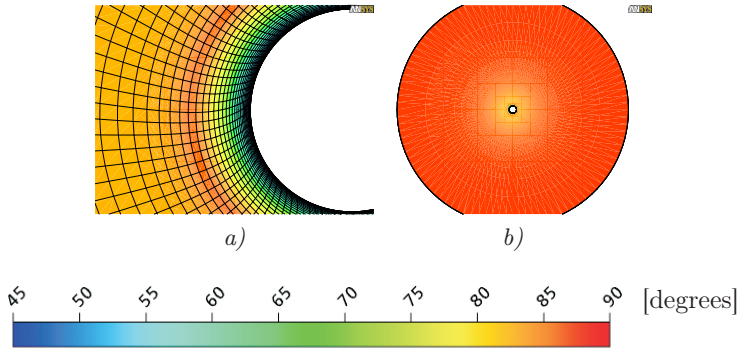


Figure 4.17: Screencaps after finished run, constant stiffness

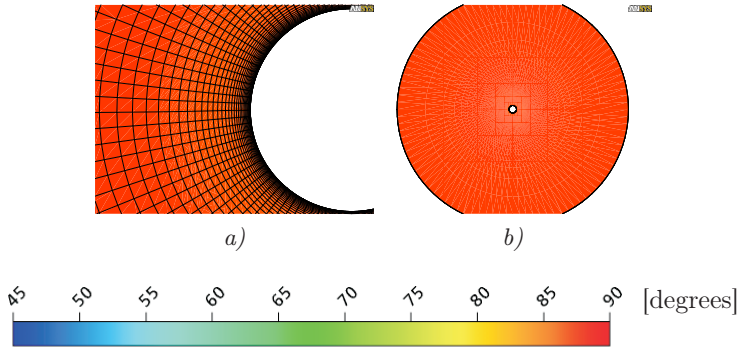


Figure 4.18: Screencaps after finished run, reciprocal of wall distance

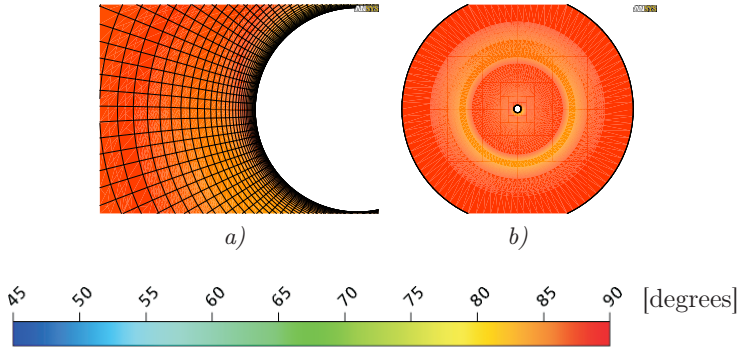


Figure 4.19: Screencaps after finished run, tanh function

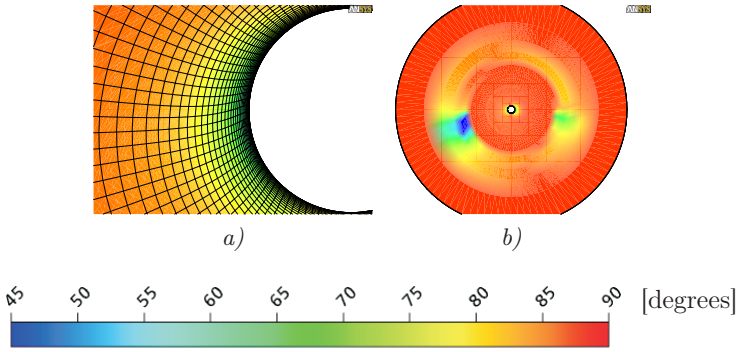


Figure 4.20: Screencaps after finished run, cos function

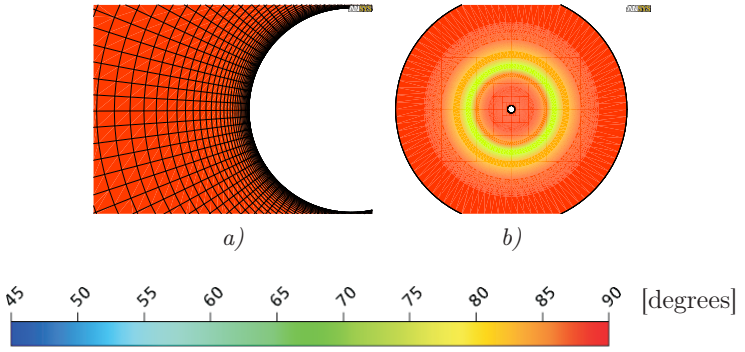


Figure 4.21: Screencaps after finished run, increase near boundaries

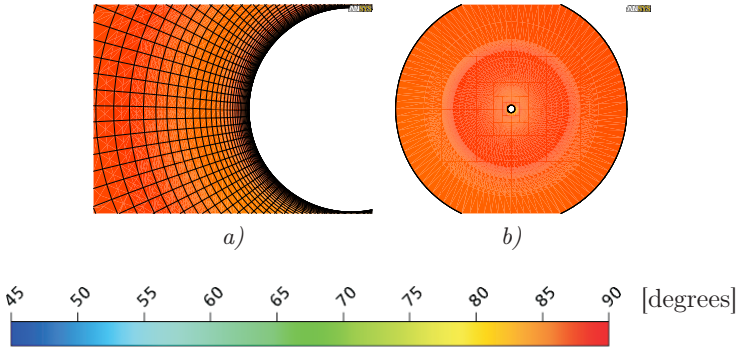


Figure 4.22: Screencaps after finished run, increase near small volumes

Again, it is obvious that the reciprocal of the wall distance or the increase near boundaries produce a more orthogonal mesh also further away from the boundary. On this note, it is concluded that unless unforseen problems occur when running 3D moving mesh simulations, these two methods are superior and either can be used.

5

Model Setup for Aeroelastic Analyses

In this chapter the model used for the simulations, used for aeroelastic analyses, is presented. The dimensions of the model are shown and the implementation of the model into Ansys CFX is explained. This includes the setup of the boundary conditions as well as the flow properties used in the simulations.

As many of the simulations conducted deals with a moving structure the method for controlling the structure movement by external Fortran routines is presented. This includes defining user routines and functions in Ansys CFX linking to the relevant Fortran routines.

Lastly the method used for dividing the structure into a defined number of sections to obtain the loads in the Fortran routines is described. The programming language called PERL have been used in this process. Not much emphasis is put on describing this programming language and a reference is made to app. G where this is described in further detail.

In this chapter the model for the simulations, used for aeroelastic analyses, and the setup in CFX is presented. The structure is chosen as a square cylindrical structure with dimensions as shown in fig. 5.1.

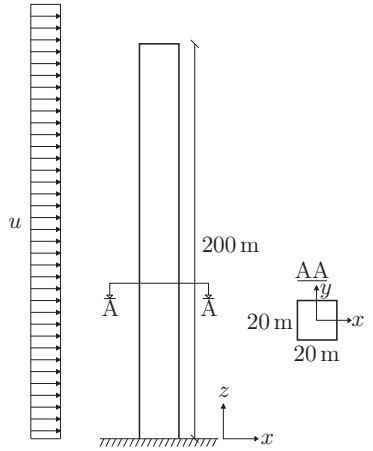
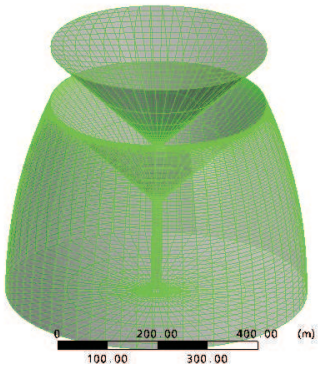


Figure 5.1: Dimensions of model structure

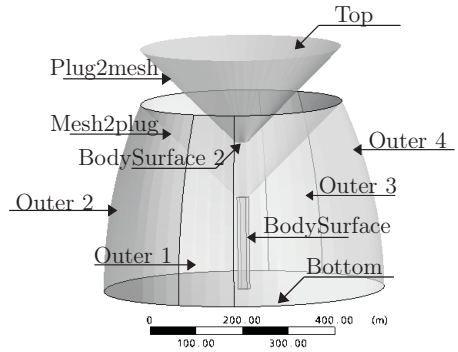
The structure is exposed to a uniform wind in the x-direction. The wind speed is selected as $u = 20\frac{\mathrm{m}}{\mathrm{s}}$. The simulations are transient and the definition of total time and time step will be presented when dealing with the respective simulation and will not be explained further in this section.

5.1 Simulation Setup

In this section the input for the simulations are presented. This includes the flow parameters and the boundary conditions used. The full domain is made up of two mesh files. This requires the definition of a domain interface between the two meshes for the solver to know how the flow is affected by this. Fig. 5.2a shows an illustration of the two meshes making up the entire domain and fig. 5.2b shows the names of the boundaries. The latter will be helpful in subsec. 5.1.3 dealing with the boundary conditions.



a) Illustration of the two meshes making up the entire domain



b) Definition of boundary names

Figure 5.2: Illustration of domain and boundaries

The input for the domain interface are listed in tab. 5.1.

Table 5.1: Definition of domain interface for simulations

Basic Settings	
- Interface Type	Fluid Fluid
Interface Side 1	
- Domain (Filter)	Domain 1
- Region List	Mesh2plug
Interface Side 2	
- Domain (Filter)	Domain 2
- Region List	Plug2mesh
Interface Models	
- Option	General Connection
- **Frame Change/Mixing Model**	
- Option	None
- **Pitch Change**	
- Option	None
Mesh Connection Method	
- Option	Automatic

5.1.1 Domain Parameters and Global Initial Conditions

The flow parameters and initial conditions for the simulations are listed in tab. 5.2.

Table 5.2: Domain parameters and global initial conditions for simulations

Domain Parameters			
General options		Fluid Models	
Basic Settings		**Heat transfer Model**	
- Location	DOMAIN1, DOMAIN2	- Option	None
- Domain type	Fluid Domain		
- Fluid List	Air at 25 C		
- Coord frame	Coord 0		
Domain Models		**Turbulence Model**	
- Ref. Press	1 [atm]	- Option	Shear Stress Transport
		- Wall Function	Automatic
Buoyancy			
- Option	Non Buoyant		
Domain Motion			
- Option	Stationary		
Mesh Deformation			
- Option	*"Varies"*		
Global Initial Conditions		**Solver Control**	
Initial Conditions		**Advection Scheme**	
- Velocity Type	Cartesian	- Option	High Resolution
Cart. Vel. Components		**Convergence Control**	
- Option	Automatic with Value		
- u	20 [m sˆ-1]		
- v	0 [m sˆ-1]		
- w	0 [m sˆ-1]		
Static pressure		**Convergence Criteria**	
- Option	Automatic	- Residual Type	RMS
Turbulence Eddy Dissipation		- Residual Target	1.E-4
- Option	Automatic		

As can be seen from tab. 5.2, the fluid is chosen as *Air at 25 C*. In tab. 5.3 below, the case relevant data for this fluid are listed.

Table 5.3: Data for the fluid Air at 25 C

Density, ρ	1.185 $\left[\frac{\text{kg}}{\text{m}^3}\right]$
Dynamic Viscosity, μ	1.831e-005 $\left[\frac{\text{kg}}{\text{m s}}\right]$

In the case of specification of the mesh deformation, different options are used throughout this project. Therefore specification of this will be explained in the respective section when relevant.

It is seen in tab. 5.2 that the turbulence model used is the Shear Stress transport model (SST). The motivation for using this model is presented in subsec. 5.1.2.

5.1.2 Turbulence Model

As mentioned above, the turbulence model used is the Shear Stress Transport model (SST) which utilizes the strengths of both the $k-\epsilon$ and the $k-\omega$ model. The two latter will be discussed first to explain the motivation for using the SST

model. Only a discussion of the pros and cons of the two models will be conducted. For a derivation of the governing eqs. for the turbulence models, see app. C.

Both the $k-\epsilon$ and the $k-\omega$ model are two equation models which are also called *Complete* models as they do not require any prior knowledge of the turbulence structure in the flow. As the names indicate they both use the specific turbulence kinetic energy, k, in the model but uses different variables for determining the turbulence length scale. The turbulence kinetic energy is used to compute the energy in the turbulence while the computation of the turbulent length scale require knowledge of an additional variable. Here, the $k-\epsilon$ model uses the dissipation, ϵ, while the $k-\omega$ model uses the specific dissipation rate, ω.

$k-\epsilon$ model

The (*High Re*) $k-\epsilon$ model is the most widely used turbulence model. This model uses the dissipation, ϵ, to determine the turbulent length scales. Studies in [Wilcox 2002] show that the $k-\epsilon$ model gives inaccurate results in regions with adverse pressure gradients. The flow in this thesis is expected to separate and cause adverse pressure gradients and hereby contain a region, where the model does not give accurate results.
The $k-\epsilon$ model only yield an accurate solution for fully turbulent flows. In flows near walls, regions exist in which the local turbulent Reynolds number is so small that the viscous effects become more significant than the turbulent ones providing inaccurate results when using the $k-\epsilon$ model through the viscous sublayer.
When increasing the distance from wall bounded flows and into the free-stream in the farfield the $k-\epsilon$ model yield very accurate results as it is not very sensitive to the free-stream conditions.

$k-\omega$ model

As mentioned, the $k-\omega$ model uses the specific dissipation rate, ω, as the additional variable. Studies in [Wilcox 2002] show that the $k-\omega$ model yields good results in flows with adverse pressure gradients and is also capable of handling the viscous sublayer with low turbulent Reynolds number. However the ω term in the $k-\omega$ model shows a high sensitivity to the free-stream conditions, which is opposite the behavior of the $k-\epsilon$ model.

SST model

As the two models have their strengths and weaknesses in opposite regions of the flow, a model used to utilize the best features of both models and take the transport of shear stress into account has been developed. This model is called the Shear Stress Transport model (SST). The SST model uses a $k-\omega$ formulation in the near wall regions with adverse pressure gradients and low turbulent Reynolds number, where this formulation yield accurate results. An important aspect when dealing with Fluid-Structure interaction (FSI) is to determine the pressure on the body as accurate as possible. Flow separation and the displacement effects associated with this is very important to predict which the $k-\epsilon$ model is incapable of.
When moving away from the wall the model is gradually transformed into a $k-\epsilon$ formulation which, as mentioned, is less sensitive to the free-stream conditions. The transition is controlled by a blending function which can distinguish between the different zones using the appropriate turbulence model in the correct zone. Furthermore, to account for the effects of shear stress transport, a blending function is introduced in the SST model to govern the eddy viscosity, ν_T, which significantly improves the behavior in strong adverse pressure-gradient flows.

5.1.3 Boundary Conditions

In this subsection the boundary conditions used for the simulations are presented. The boundary conditions used are listed in tab. 5.4.

Table 5.4: Boundary Conditions

Basic Settings		**Boundary Details**	
STRUCTURE:			
- Boundary type	Wall	**Wall Influence on Flow**	
- Location	BodySurface, BodySurface 2	- Option	No Slip
		Mesh Motion	
		- Option	Specified Displacement
		- X Component	dispX(z)
		- Y Component	dispY(z)
		- Z Component	0 [m]
INLET:			
- Boundary Type	Inlet	**Flow Regime**	
- Location	Outer 2	- Option	Subsonic
		Mass and Momentum	
		- Option	Cart. Vel. Components
		- u	20 [m s^-1]
		- v	0 [m s^-1]
		- w	0 [m s^-1]
		Turbulence	k and Omega
		- Turb. Kinetic Energy	0.01 [m^2 s^-2]
		- Turb. Eddy Frequency	1e+006 [s^-1]
		Mesh motion	
		- Option	Stationary
OUTLET:			
- Boundary Type	Outlet	**Flow Regime**	
- Location	Outer 4	- Option	Subsonic
		Mass and Momentum	
		- Option	Average Static Pressure
		- Relative Pres.	1 [Pa]
		Pressure Averaging	
		- Option	Average Over Whole Outlet
		Mesh motion	
		- Option	Stationary
OPENING:			
- Boundary Type	Opening	**Flow Regime**	
- Location	Outer 1, Outer 3	- Option	Subsonic
		Mass and Momentum	
		- Option	Opening Pres. and Dirn
		- Relative Pres.	1 [Pa]
		Flow Direction	
		- Option	Normal to Boundary Condition
		Turbulence	Medium (Intensity = 5 %)
		Mesh motion	
		- Option	Stationary
BOTTOM:			
- Boundary Type	Wall	**Wall Influence on Flow**	
- Location	Bottom	- Option	No Slip
		Mesh motion	
		- Option	Stationary
TOP:			
- Boundary Type	Opening	**Flow Regime**	
- Location	Top	- Option	Subsonic
		Mass and Momentum	
		- Option	Opening Pres. and Dirn
		- Relative Pres.	1 [Pa]
		Flow Direction	
		- Option	Normal to Boundary Condition
		Turbulence	Medium (Intensity = 5 %)
		Mesh motion	
		- Option	Stationary

The inclusion of a domain interface results in two extra boundaries, where the boundary conditions used are listed in tab. 5.5

Table 5.5: Boundary Conditions for Domain Interface

Basic Settings		**Boundary Details**	
DOMAIN INTERFACE 1 SIDE 1:			
- Boundary Type	Interface	**Mass and momentum**	
- Location	Mesh2plug	- Option	Conservative Interface Flux
		Turbulence	
		- Option	Conservative Interface Flux
		Mesh motion	
		- Option	Unspecified
DOMAIN INTERFACE 1 SIDE 2:			
- Boundary Type	Interface	**Mass and momentum**	
- Location	Plug2mesh	- Option	Conservative Interface Flux
		Turbulence	
		- Option	Conservative Interface Flux
		Mesh motion	
		- Option	Unspecified

The specifications of the mesh motion under "Structure" in tab. 5.4 are valid for simulations where the structure is allowed to move. In simulations with a stationary structure mesh motion is not used an these settings are not available.

Furthermore, the names, `dispX(z)` and `dispY(z)`, are not CFX related names. The names relate to defined "User Functions" in CFX named by the user. The mesh deformation is governed by Fortran routines where the "User Functions" refer to specified "User Routines". The connection can be seen in fig. 5.3.

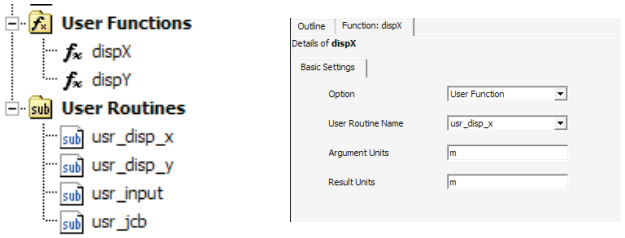


Figure 5.3: Definition of mesh motion in CFX using "User Functions" and "User Routines"

The turbulence settings under "Inlet" are specified due to the expected presence of vortex shedding. in order to capture these vortices, the eddy viscosity has to be sufficiently low. The eddy viscosity is given by:

$$\nu_t = \frac{k}{\omega} \tag{5.1}$$

where

k is the turbulent kinetic energy, $\left[\frac{\mathrm{m}^2}{\mathrm{s}^2}\right]$
ω is the specific dissipation rate , $\left[\frac{1}{\mathrm{s}}\right]$,
denoted *the turbulent eddy frequency* in CFX

Using the values for k and ω listed in tab. 5.4 the eddy viscosity at the inlet boundary becomes:

$$\nu_t = 10^{-8}\frac{\mathrm{m}^2}{\mathrm{s}} \sim \frac{\nu}{1500} \tag{5.2}$$

Opposite the molecular viscosity, the eddy viscosity is not a fluid property. The eddy viscosity is purely an imaginary concept. The eddy viscosity is introduced when simplifying the Reynolds stresses. The simplification states that the Reynolds stresses are proportional to the gradient of the mean velocity. The proportionality coefficient is termed the eddy viscosity.

The reason that a low eddy viscosity is used is to try and keep the Reynolds number sufficiently high, and thereby to ensure a turbulent flow. The Reynolds number is proportional to the reciprocal of the sum of the molecular and the eddy viscosity. As the molecular viscosity is a fluid property the eddy viscosity is kept at a minimum. In Ansys CFX it is possible to overwrite the predefined molecular viscosity, but that function has not been used.

5.1.4 User Routines and User Functions

In order to simulate FSI several "User Routines" and "User Functions" have been specified. In this subsection the purpose of the individual routines and functions are described. For information on the setup of User Routines and User Functions in CFX, see app. F.

usr_input

This User Routine (Junction Box Routine) links to the Fortran routine `usr_input_XX.F`, in which all the structure model related info, and some additional simulation related info is loaded into CFX and saved for use in the other routines. The `XX` in the Fortran file name are not to be considered to be the actual name. `XX` merely indicates that different names are used after the terms `usr_input`. The routine is called only once during a simulation, at the Junction box location "User Input". The following info is loaded in `usr_input_XX.F`:

- The eigenmodes of the structure are loaded from `P.dat`
- The eigenvalues of the structure are loaded from `O.dat`
- The transformation matrix is loaded from `T.dat`
- The number of beams in the structural model, the length of the beams, the number of sections each beam is divided into, the eigenmodes included in the modal model, and the position of `P.dat`, `O.dat`, `T.dat` and `Output.dat` are loaded from `input.dat`

The data loaded is subsequently stored on the Ansys CFX Memory Management System (MMS), for more info on this, see app. E.

usr_jcb

This User Routine (Junction Box Routine) links to the Fortran routine `usr_jcb_XX.F` where `XX` again only means that this is not the full file name. The routine calculates loads on the structure, displacements of the structural model, and is called at the end of each time step. The procedure of the routine is as follows:

- The wind loads on the building are calculated section-wise, sections being determined from the z coordinate. How the sections are specified can be found in sec. 5.2.
- These section loads are then distributed onto the nodes in the structural model as nodal loads, using the shape functions for the beams to do so
- A modal model of the structure is setup, using a limited number of eigenmodes
- The Newmark-algorithm is employed to calculate the resulting displacements of the nodes in the modal model
- The displacements are transformed back to nodal displacements of the structural model
- The nodal displacements are saved on the MMS, for use in the routines `usr_disp_x` and `usr_disp_y`

usr_disp_x

This User Routine (User CEL Function) links to the Fortran routine `usr_disp_x_XX.F`. The routine calculates the x-displacement of the nodes on the structure boundary, and is called when CFX calculates mesh displacements. The procedure of the routine is as follows:

- The z coordinate of a node is used to determine which beam the placement of the node corresponds to
- The displacements for the given beam , calculated in `usr_jcb_XX.F`, are loaded from the MMS
- The shapefunctions for the beam are evaluated for the z value
- The x-displacement of the node is found by multiplying the shapefunctions with the displacements, and sent to CFX

usr_disp_y

This User Routine (User CEL Function) links to the Fortran routine `usr_disp_y_XX.F`. The routine calculates the y-displacement of the nodes on the structure boundary, and is called when CFX calculates mesh displacements. The procedure of the routine is as follows:

- The z coordinate of a node is used to determine which beam the placement of the node corresponds to
- The displacements for the given beam , calculated in `usr_jcb_XX.F`, are loaded from the MMS
- The shapefunctions for the beam are evaluated for the z value
- The y-displacement of the node is found by multiplying the shapefunctions with the displacements, and sent to CFX

dispX

This User Function links to the User Routine (User CEL Function) `usr_disp_x` and is used when specifying x-displacements of the structure, see tab. 5.4, as follows: `dispX(z)`. This sends the z values to the User Routine, in which they are processed and used to calculate the x-displacements, as described for `usr_disp_x`.

dispY

This User Function is links to the User Routine (User CEL Function) `usr_disp_y` and is used when specifying y-displacements of the structure, see tab. 5.4, as follows: `dispY(z)`. This sends the z values to the User Routine, in which they are processed and used to calculate the y-displacements, as described for `usr_disp_y`.

5.2 Pressure Variables

As mentioned above, the forces on the structure are determined in sections. In order to get these forces in the Fortran routine a macro is written which adds variables to the CFX environment. The variables are pressure variables. The definition of each variable is, that it contains the nodal values of the pressure in a defined section of the domain and 0 in the rest of the domain. This retains the possibility of using the built in `areaInt` function in CFX. The `areaInt` function calculates an integral of a given variable on a given surface (area). By creating variables where the pressure is set to 0 outside the section of interest, the function can then perform an area integral over the entire structure surface and obtain only the forces on the section specified.

The macro used, (*add_ variables.ccl*), is included on the DVD in [*DVD:\CFX\Macros*]. The structure of the macro is based on the introduction in app. G and will not be elaborated further here. Only the structure of the loops and expressions included will be introduced. The individual terms and expressions are presented the same way as they are written in the macro.

As mentioned each additional variable contains pressure data for a specific region and set to 0 in the rest of the domain. To separate the section data from the full set a built in step function in CFX is used. The structure of this function is given below:

$$\text{step}(x) = \begin{cases} 1 & \text{for} \quad x > 0 \\ 0 & \text{for} \quad x < 0 \\ 0.5 & \text{for} \quad x = 0 \end{cases} \tag{5.3}$$

where

x is the argument for the step function

The force on the structure is made up of two contributions: from the pressure on the structure and from the wall shear. The two contributions are first added in the Fortran routine and two sets of additional variables are made. Also different expressions have been used, one for the first section and one for the rest of the sections. The expressions for the first section are listed below:

```
p1 = p * step((dz-z)/1[m])              Pressure term
s1 = wall shear * step((dz-z)/1[m])     Wall shear term
```

where

`dz` is the height of the section [m]
`z` is the vertical coordinate [m]

The height of the section, `dz`, is determined by dividing the total height of the structure with the wanted number of sections:

```
dz=maxZ/$nsection
```

where

```
maxZ = max(z)Structure
```

The reason that the argument in the step function is divided by the unit of the nominator is, that the step function only works for dimensionless arguments. it is easily seen from the step functions that the argument becomes negative as `z` increases above `dz`.

The additional variables, called `pSect001` and `sSect001`, are programmed to attain the value of `p1` and `s1` respectively by the method described in app G.

As mentioned the expressions for the remaining sections are a bit different than the ones used for the first section. This is due to the structure of the step function. When dealing with the section `$i` the step function used above would cause the variables to contain pressure values for all nodes below `z=$i*dz` and not only within the height `dz` of section `$i` as wanted. To correct this another step function is introduced which is multiplied onto the existing step function. An illustration of these step functions are shown in fig. 5.4 where it should be noted that `$nhelp=$i-1`.

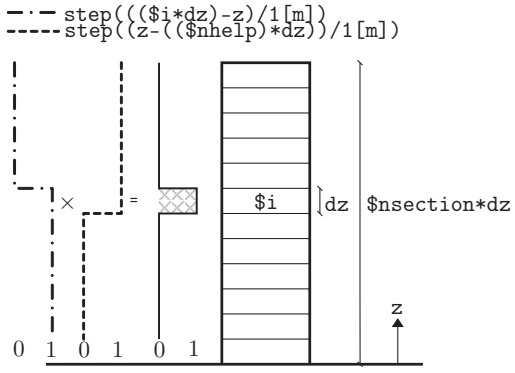


Figure 5.4: Illustration of step functions

The expressions for the remaining sections are therefore given by:

```
p$i = p * step((($i*dz)-z)/1[m])*step((z-(($nhelp)*dz))/1[m])
s$i = wall shear * step((($i*dz)-z)/1[m])*step((z-(($nhelp)*dz))/1[m])
```

The use of `$i` and `$nhelp` is because the expressions are determined within a loop `for $i (2 .. $nsection)`.

Progress in the project showed that due to program problems the use of the `areaInt` function on the wall shear variables were not possible in Ansys CFX 11. According to Ansys Support this should be fixed in the coming version to be released later in 2008. Therefore the wall shear variables are not actually used and the presentation of the variables in this section should be regarded as a presentation of how it would have been done if possible.

6
Dynamic Model

In this chapter the structural model used to govern the structure movement is described. The structure is divided into a number of beam segments and the mass and stiffness properties are assembled into global matrices. These matrices form the basis for the derivation of a modal representation of the structure which is then used throughout the rest of the project.

Besides the structural model this chapter also deals with the method used for determining the loads on the structure. The loads are extracted from Ansys CFX in specified sections of the structure during transient runs and these loads are processed in Fortran routines to generate nodal loads in the beam model. The way this transition works is described. Also the method used for determining the structure movement based on these nodal loads is presented.

In order to simulate the aeroelastic response of the structure, a way to govern the structure displacements from the loads in the flow has to be set up. In this chapter the dynamic model used in this project is presented. Also, the method for determining nodal loads in the chosen FEM beam model is described. Lastly the method used for transforming these loads into structure displacements is presented.

6.1 Structural Model

First step towards modeling the FSI, is to set up a structural model. In the present case, a FEM model using 3D beam elements has been chosen. The purpose of the FEM model is to supply a stiffness matrix, **K**, and a mass matrix, **M**, for the entire structural model, which is shown on fig. 6.1.

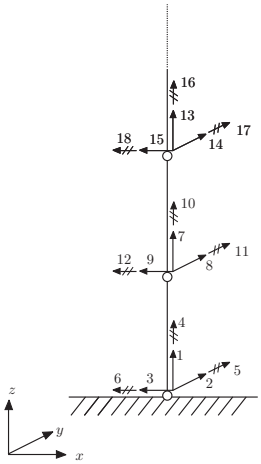


Figure 6.1: Illustration of undeformed structural model composed of 3D beam elements. The coordinate system shown is the global coordinate system, used in CFX, and the degrees of freedom are shown in the local coordinates of the beams, used in the structural model.

In this section, the theory used to obtain the global stiffness and mass matrices, **K** and **M** respectively, will be explained.

The MatLab program used to assemble the mass and stiffness matrices is a multi purpose FEM program created by Lars Andersen, Jesper W. Stærdahl and Johan Clausen at Aalborg University. The input needed to calculate the mass and stiffness matrices are:

- Material data
- Section data
- Node coordinates
- Beam topology

Under Material data, the mass per length, μ, needs to be specified. This is estimated from the assumption that the load bearing structure looks like the sketch in fig. 6.2, with one 0.25 m thick floor structure every 3.50 m, and that the entire load bearing structure is made of reinforced concrete with an average density of $2500\frac{\text{kg}}{\text{m}^3}$. The cross section of the core is assumed quadratic, $c_1 = c_2$. Similarly, under section data the moments of inertia about the x- and y-axes needs to be given along with the area. These are calculated based upon the same square solid core, as mentioned above. The MatLab program consists of several MatLab files, where only the ones used with the FEM beam model have been used. These are included on the DVD under [*DVD:\Beam_ Model*].

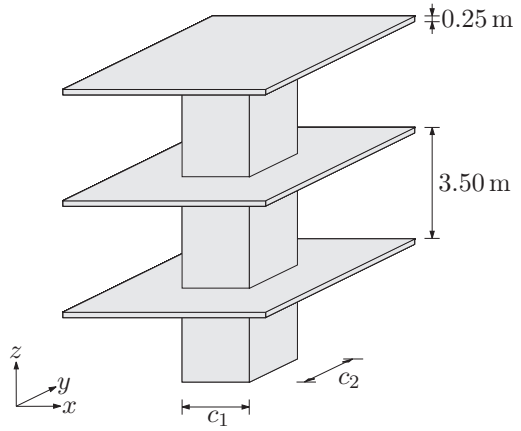


Figure 6.2: Concept of the load bearing structure.

6.1.1 Stiffness and Mass Matrices for a Single 3D Beam Element

A single 3D beam has 12 degrees of freedom (DOF), 6 for each node as shown in fig. 6.1, and it's shape functions are given in the matrix **N** as:

$$\mathbf{N}^T = \begin{bmatrix} N_1 & 0 & 0 & 0 \\ 0 & N_3 & 0 & 0 \\ 0 & 0 & N_3 & 0 \\ 0 & 0 & 0 & N_1 \\ 0 & 0 & -N_4 & 0 \\ 0 & N_4 & 0 & 0 \\ N_2 & 0 & 0 & 0 \\ 0 & N_5 & 0 & 0 \\ 0 & 0 & N_5 & 0 \\ 0 & 0 & 0 & N_2 \\ 0 & 0 & -N_6 & 0 \\ 0 & N_6 & 0 & 0 \end{bmatrix} \tag{6.1}$$

where the shape functions N_1-N_6 are functions of $\xi = \frac{x}{L}$, given as:

$$\begin{aligned} N_1 &= 1 - \xi \\ N_2 &= \xi \\ N_3 &= 1 - 3\xi^2 + 2\xi^3 \\ N_4 &= L\xi(-1 + 2\xi - \xi^2) \\ N_5 &= \xi^2(3 - 2\xi) \\ N_6 &= L\xi^2(1 - \xi) \end{aligned} \tag{6.2}$$

[Nielsen 2004, p. 152]

The shape functions are shown in fig. 6.3. N_1 and N_2 govern axial torsion and axial displacement, N_3 and N_5 govern cross stream displacement, and lastly N_4 and N_6 govern bending.

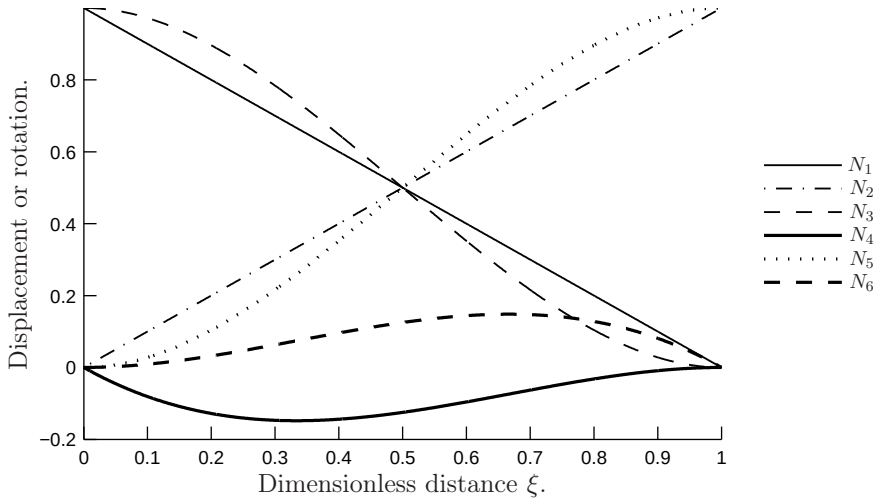


Figure 6.3: Illustration of shape functions N_1-N_6.

The mass and stiffness matrices of the beam, $\mathbf{K_b}$ and $\mathbf{M_b}$, are then given as:

$$\begin{aligned} \mathbf{K_b} &= \int \mathbf{B}^T \mathbf{E} \mathbf{B}\, dV \\ \mathbf{M_b} &= \int \rho \mathbf{N}^T \mathbf{N}\, dV \end{aligned} \tag{6.3}$$

[Cook, Markus, Plesha & Witt 2002, p. 89 & p. 376]

where ρ is the density of the beam, $\mathbf{E}$ is given by eq. (6.4), and $\mathbf{B}$ is given by eq. (6.5).

$$\mathbf{E} = \begin{bmatrix} EA & 0 & 0 & 0 \\ 0 & EI_y & 0 & 0 \\ 0 & 0 & EI_z & 0 \\ 0 & 0 & 0 & GA \end{bmatrix} \tag{6.4}$$

$$\mathbf{B}^T = \begin{bmatrix} \frac{d}{dx}N_1 & 0 & 0 & 0 \\ 0 & \frac{d^2}{dx^2}N_3 & 0 & 0 \\ 0 & 0 & \frac{d^2}{dx^2}N_3 & 0 \\ 0 & 0 & 0 & \frac{d}{dx}N_1 \\ 0 & 0 & -\frac{d^2}{dx^2}N_4 & 0 \\ 0 & \frac{d^2}{dx^2}N_4 & 0 & 0 \\ \frac{d}{dx}N_2 & 0 & 0 & 0 \\ 0 & \frac{d^2}{dx^2}N_5 & 0 & 0 \\ 0 & 0 & \frac{d^2}{dx^2}N_5 & 0 \\ 0 & 0 & 0 & \frac{d}{dx}N_2 \\ 0 & 0 & -\frac{d^2}{dx^2}N_6 & 0 \\ 0 & \frac{d^2}{dx^2}N_6 & 0 & 0 \end{bmatrix} \tag{6.5}$$

Using eq. (6.3) the 3D beam stiffness matrix, $\mathbf{K_b}$, is given as:

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 & 0 & -A & 0 & 0 & 0 & 0 & 0 \\ 0 & B_z & 0 & 0 & 0 & C_z & 0 & -B_z & 0 & 0 & 0 & C_z \\ 0 & 0 & B_y & 0 & -C_y & 0 & 0 & 0 & -B_y & 0 & -C_y & 0 \\ 0 & 0 & 0 & F & 0 & 0 & 0 & 0 & 0 & -F & 0 & 0 \\ 0 & 0 & -C_y & 0 & D_y & 0 & 0 & 0 & C_y & 0 & E_y & 0 \\ 0 & C_z & 0 & 0 & 0 & D_z & 0 & -C_z & 0 & 0 & 0 & E_z \\ -A & 0 & 0 & 0 & 0 & 0 & A & 0 & 0 & 0 & 0 & 0 \\ 0 & -B_z & 0 & 0 & 0 & -C_z & 0 & B_z & 0 & 0 & 0 & -C_z \\ 0 & 0 & -B_y & 0 & C_y & 0 & 0 & 0 & B_y & 0 & C_y & 0 \\ 0 & 0 & 0 & -F & 0 & 0 & 0 & 0 & 0 & F & 0 & 0 \\ 0 & 0 & -C_y & 0 & E_y & 0 & 0 & 0 & C_y & 0 & D_y & 0 \\ 0 & C_z & 0 & 0 & 0 & E_z & 0 & -C_z & 0 & 0 & 0 & D_z \end{bmatrix}$$

where indices y and z are substituted with index i as

$$\begin{aligned} A &= \frac{AE}{L} & B_i &= \frac{12EI_i}{L^3} & C_i &= \frac{6EI_i}{L^2} \\ D_i &= \frac{4EI_i}{L} & E_i &= \frac{2EI_i}{L} & F &= \frac{GI_x}{L} \end{aligned} \tag{6.6}$$

Using eq. (6.3) the 3D beam mass matrix, $\mathbf{M_b}$, is given as:

$$\begin{bmatrix}
M_1 & 0 & 0 & 0 & 0 & 0 & M_5 & 0 & 0 & 0 & 0 & 0 \\
0 & M_2 & 0 & 0 & 0 & M_3 & 0 & M_4 & 0 & 0 & 0 & -M_6 \\
0 & 0 & M_2 & 0 & -M_3 & 0 & 0 & 0 & M_4 & 0 & M_6 & 0 \\
0 & 0 & 0 & M_9 & 0 & 0 & 0 & 0 & 0 & M_0 & 0 & 0 \\
0 & 0 & -M_3 & 0 & M_7 & 0 & 0 & 0 & -M_6 & 0 & -M_8 & 0 \\
0 & M_3 & 0 & 0 & 0 & M_7 & 0 & M_6 & 0 & 0 & 0 & -M_8 \\
M_5 & 0 & 0 & 0 & 0 & 0 & M_1 & 0 & 0 & 0 & 0 & 0 \\
0 & M_4 & 0 & 0 & 0 & M_6 & 0 & M_2 & 0 & 0 & 0 & -M_3 \\
0 & 0 & M_4 & 0 & -M_6 & 0 & 0 & 0 & M_2 & 0 & M_3 & 0 \\
0 & 0 & 0 & M_0 & 0 & 0 & 0 & 0 & 0 & M_9 & 0 & 0 \\
0 & 0 & M_6 & 0 & -M_8 & 0 & 0 & 0 & M_3 & 0 & M_7 & 0 \\
0 & -M_6 & 0 & 0 & 0 & -M_8 & 0 & -M_3 & 0 & 0 & 0 & M_7
\end{bmatrix}$$

where

$$\begin{aligned}
M_1 &= \frac{140m}{420} \qquad M_2 = \frac{156m}{420} \qquad M_3 = \frac{22Lm}{420} \\
M_4 &= \frac{54m}{420} \qquad M_5 = \frac{70m}{420} \qquad M_6 = \frac{13Lm}{420} \\
M_7 &= \frac{4L^2m}{420} \qquad M_8 = \frac{3L^2m}{420} \qquad M_9 = \frac{140JL}{420} \\
M_0 &= \frac{70JL}{420}
\end{aligned} \tag{6.7}$$

The stiffness and mass matrices used in the various simulations in the project are obtained from the following input in MatLab:

```
matdat{1} =  [2 30e9 0.3 151850 1e6 0 0];
section{1} = [4 400 1000000 100 100 1];
```

The input in `matdat{1}` are:

`[`m_t `-` E `-` ν `-` μ `-` J `-` a_m `-` a_k`]`

where

m_t	is the material type (2=homogeneus linear elastic)
E	is Youngs Modulus, [Pa]
ν	is Poissons Ratio, [-]
μ	is the mass per length of the beam, $\left[\frac{\text{kg}}{\text{m}}\right]$

J	is the polar moment of inertia, [m^4]
a_m,a_k	are the constants for the Rayleigh damping matrix, [-] (not used directly)

The input in `section{1}` are:

`[`s_t `-` A `-` I_x `-` I_y `-` I_z`]`

where

s_t	is the section type (4=3D beam)
A	is the area, [m^2]
I_x	is the moment of inertia about the x-axis (local coordinates), [m^4]
I_y	is the moment of inertia about the y-axis (local coordinates), [m^4]
I_z	is the moment of inertia about the z-axis (local coordinates), [m^4]

6.1.2 Assembling Global Stiffness and Mass Matrices

Once the local stiffness and mass matrices, $\mathbf{K_b}$ and $\mathbf{M_b}$, have been constructed, they need to be transformed into global coordinates, as they are given in local coordinates, with the x-axis running along the beam. This is done using a 12×12 transformation matrix, $\mathbf{T}$, as follows:

$$\begin{aligned} \mathbf{K_{b},}_{global} &= \mathbf{T}\mathbf{K_b}\mathbf{T}^T \\ \mathbf{M_{b},}_{global} &= \mathbf{T}\mathbf{M_b}\mathbf{T}^T \end{aligned} \tag{6.8}$$

Then the matrices are assembled according to the topology of the model, so matrices from adjoining beams overlap in the global matrices, $\mathbf{K}$ and $\mathbf{M}$.

The eigenmodes and eigenvalues are found, according to app. D.1, and normalized to $\mathbf{M}$ so that $\mathbf{P}^T\mathbf{M}\mathbf{P} = \mathbf{I}$, where $\mathbf{P}$ is the modal matrix $\mathbf{P} = [\mathbf{\Phi}^{(1)}\mathbf{\Phi}^{(2)} \dots \mathbf{\Phi}^{(n)}]$ and $\mathbf{\Phi}^{(i)}$ is the i'th eigenmode.

The normalization is done as follows:

$$\begin{aligned} \overline{\mathbf{M}} &= \overline{\mathbf{P}}^T \mathbf{M} \overline{\mathbf{P}} \\ \mathbf{\Phi}^{(i)} &= \frac{1}{\sqrt{M_i}} \overline{\mathbf{\Phi}}^{(i)} \end{aligned} \tag{6.9}$$

[Nielsen 2005, p. 23]

where a bar denotes unnormalized, and $\overline{\mathbf{M}}$ is given below:

$$\overline{\mathbf{M}} = \begin{bmatrix} M_1 & 0 & \cdots & 0 \\ 0 & M_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & M_n \end{bmatrix} \tag{6.10}$$

In this project, only the first four eigenmodes are used. These are shown in fig. 6.4. The corresponding undamped circular eigenfrequencies are:

$$\begin{aligned} \omega_1 &= 0.3907 \tfrac{\text{rad}}{\text{s}} \Rightarrow f_1 = 0.0622\,\text{Hz} \\ \omega_2 &= 0.3907 \tfrac{\text{rad}}{\text{s}} \Rightarrow f_2 = 0.0622\,\text{Hz} \\ \omega_3 &= 2.4486 \tfrac{\text{rad}}{\text{s}} \Rightarrow f_3 = 0.3897\,\text{Hz} \\ \omega_4 &= 2.4486 \tfrac{\text{rad}}{\text{s}} \Rightarrow f_4 = 0.3897\,\text{Hz} \end{aligned} \tag{6.11}$$

As can be seen, the undamped circular eigenfrequencies are paired two and two. This is due to the circumstance that the structure is symmetric about the global x- and y-axes both structurally and geometrically. In this project it is chosen not to include axial deformation and torsion in the dynamic model. This is mainly based on an expectation, that axial deformation and torsion will be much smaller than the other parts.

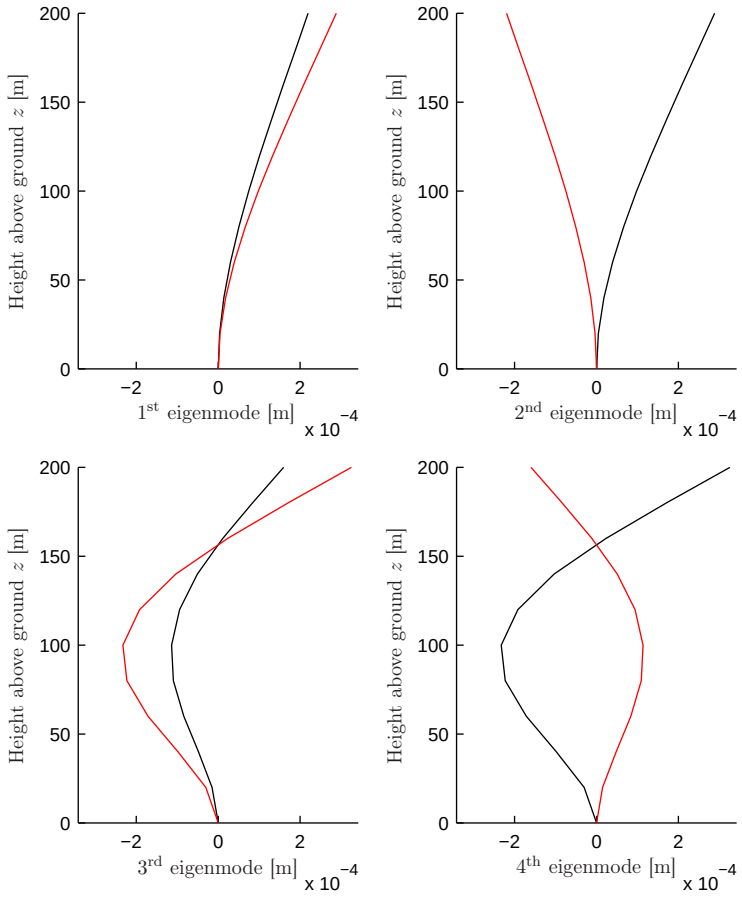
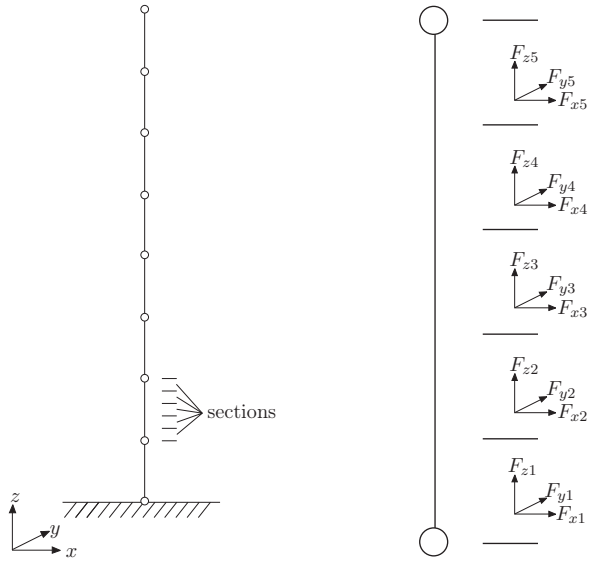


Figure 6.4: Plot of the first four eigenmodes. — are displacements in the x-direction and — are displacements in the y-direction.

6.2 Obtaining Loads

During a transient run in CFX, the loads need to be extracted for each time step. The loads are obtained by integrating the pressure section-wise over the surface of the structure, using additional variables as described in sec. 5.2. Due to a bug in the routine `USER_GET_GVAR` the shear forces cannot be obtained and are thus not used. This is a bug in CFX 11.0 which should be fixed in CFX 12.0.
The section forces are illustrated in fig. 6.5.



a) Structural model

b) Closeup of beam with section forces

Figure 6.5: Illustration of structural model with a closeup of one beam and the section forces belonging to it. Coordinates are global, as used in CFX.

The integrated pressure on a section of the building, in the x, y and z directions respectively, is obtained using the call `USER_GET_GVAR`:

```
      CALL USER_GET_GVAR(
&   'Varname','Location','areaInt_x',CRESLT,FX,CZ,DZ,IZ,LZ,RZ)
      CALL USER_GET_GVAR(
&   'Varname','Location','areaInt_y',CRESLT,FY,CZ,DZ,IZ,LZ,RZ)
      CALL USER_GET_GVAR(
&   'Varname','Location','areaInt_z',CRESLT,FZ,CZ,DZ,IZ,LZ,RZ)
```

where `Varname` is the name of the additional variable, `Location` is the name of the structure boundary, `FX`, `FY`, `FZ` are the variables in which the results are stored, and `areaInt_x`, `areaInt_y`, `areaInt_z` defines the integration to be carried out. `CRESLT`, `CZ`, `DZ`, `IZ`, `LZ`, and `RZ` are explained in app. E.

6.2.1 Obtaining Nodal Loads

Once the section forces are calculated, they need to be transformed into nodal loads, so a load vector can be assembled. First step is to transform the section forces into local coordinates for the beam (where the x axis runs along the beam) and furthermore change the section forces into evenly distributed loads over the

section length. The nodal loads for a single beam, $\mathbf{f}_{nodal}$, are then obtained in the following way:

$$\mathbf{f}_{nodal} = \sum_{i=1}^{n} \left(\int_{(i-1)\frac{L}{n}}^{i\frac{L}{n}} \mathbf{N}^T(x)\mathbf{f}_i \, dx \right) \tag{6.12}$$

where

L	is the length of the beam, [m]
n	is the number of sections on the beam
$\mathbf{N}^T$	is the shape functions where the 4^{th} row governing axial torsion has been omitted
$\mathbf{f}_i$	is the evenly distributed loads in the x, y, and z directions for section i

It has been deemed that the influence of axial torsion is quite small. As such, the axial torsion has been omitted as the benefits of including it would be overshadowed by the added complexity in the programming.

Finally, the nodal loads for each beam are transformed back to global coordinates and assembled into the load vector.

6.3 Modal Model

The modal model is introduced using the equations given in app. D.2, where the four first eigenmodes are used, corresponding to 1^{st} and 2^{nd} eigenmode about the principal axes. The equations of motion are then given as:

$$\ddot{q}_i + 2\zeta_i\omega_i\dot{q}_i + \omega_i^2 q_i = \mathbf{\Phi}^{(i)T}\mathbf{f} \tag{6.13}$$

where

$\mathbf{x} = \sum_1^\infty \mathbf{\Phi_i} q_i$	is the coupling between cartesian coordinates ($\mathbf{x}$) and modal coordinates ($\mathbf{q}$)
ζ_i	is the i'th structural damping coefficient
ω_i	is the i'th undamped circular eigenfrequency
$\mathbf{\Phi}^{(i)}$	is the i'th eigenmode
$\mathbf{f}$	is the load vector

For use with the Newmark algorithm, eq. (6.13) can be rewritten as:

$$\mathbf{M}_m\ddot{\mathbf{q}} + \mathbf{C}_m\dot{\mathbf{q}} + \mathbf{K}_m\mathbf{q} = \mathbf{f}_m \tag{6.14}$$

where

$\mathbf{M}_m = \mathbf{I}$ is the modal mass matrix
$\mathbf{C}_m$ is the modal damping matrix which is a diagonal matrix with $\mathbf{C}_m(i,i) = \zeta_i \omega_i$
$\mathbf{K}_m$ is the modal stiffness matrix which is a diagonal matrix with $\mathbf{K}_m(i,i) = \omega_i^2$
$\mathbf{f}_m = \mathbf{\Phi}^{(i)T}\mathbf{f}$ is the modal load vector

6.3.1 Modal Damping Matrix

The modal damping ratios, ζ_i, are used in the modal damping matrix $\mathbf{C}_m$. In the actual structural model, the damping matrix is given as:

$$\mathbf{C} = a_m \mathbf{M} + a_k \mathbf{K} \tag{6.15}$$

also known as Rayleigh's damping model. In this, ζ_1 and ζ_2 are known, and the coefficients a_m and a_k are given as:

$$\begin{bmatrix} a_m \\ a_k \end{bmatrix} = \frac{2\omega_1\omega_2}{\omega_2^2 - \omega_1^2} \begin{bmatrix} \omega_2 & -\omega_1 \\ -\frac{1}{\omega_2} & \frac{1}{\omega_1} \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \end{bmatrix} \tag{6.16}$$

The remaining modal damping ratios, ζ_i, can then be found from:

$$\zeta_i = \frac{1}{2\omega_i}\left(a_m + a_k \omega_i^2\right) = \frac{a_m}{2\omega_i} + \frac{a_k}{2}\omega_i \tag{6.17}$$

[Nielsen 2004, p. 100]

In case of a structure where $\omega_1 = \omega_2$, eq. (6.16) can be used where $\omega_2 = \omega_3$ and $\zeta_2 = \zeta_3$. Then eq. (6.17) can be used to determine the remaining modal damping ratios as normal.

6.4 Obtaining Nodal Displacements

The Newmark algorithm, described in app. D.4, is used with eq. (6.14) to find displacements in modal coordinates with $\beta = 0.25$, $\gamma = 0.50$, corresponding to a constant acceleration in the time interval, and the initial conditions $\mathbf{q}_0 = \dot{\mathbf{q}}_0 = \ddot{\mathbf{q}}_0 = 0$.
For each time step $\ddot{\mathbf{q}}$, $\dot{\mathbf{q}}$, and $\mathbf{q}$ are stored in MMS (see app. E), and then loaded in the following time step as "initial conditions" to calculate new values.

The nodal displacements are given as $\mathbf{x} = \mathbf{P}\mathbf{q}$, and stored at each time step as well.

6.5 Obtaining Displacement of Structure

The displacement of the structure boundary in CFX is calculated one node at the time in two separate routines `usr_disp_x.F` and `usr_disp_y.F`. The method for calculating the displacements is:

1. The z value of the node is found in global coordinates (used in the fluid model).

2. The z value is used to determine which beam the node belongs to, e.g. for beams with a length of 10 m, a z value of 21.7 m would mean it belongs to the 3$^{\text{rd}}$ beam.

3. From the full nodal displacement vector $\mathbf{x}$ (in global coordinates), a partial displacement vector $\mathbf{f}_p$ corresponding to the position of the node is extracted.

4. The partial displacement vector $\mathbf{x}_p$ (12×1) is transformed to local coordinates (used in the structural model).

5. The displacements of the node in local coordinates are found by $\mathbf{N}(x)\mathbf{x}_p$, where x is the position of the node in the local coordinate system of the beam, where x runs along the length of the beam.

6. The displacements of the node is transformed back to global coordinates and sent to CFX.

The Fortran routines `usr_disp_x.F` and `usr_disp_y.F` used to control the displacements are included on the DVD under [*DVD:\CFX\Functions_Routines*]. These routines are accessed as User CEL Functions from within CFX, with z as input argument for both (see app. F.1.2 for info on how to set up user functions and user routines in CFX).

6.6 Areas of Concern

During the programming of the Fortran code that is used to extract loads from the CFX simulation and calculate displacements, certain note worthy experiences have been made, which if taken into consideration may drastically reduce potential difficulties in the programming process. It should here be noted that all the programs are written to work with a Fortran77 compiler, and some of the issues discussed below would be of no concern if operating with a Fortran90 compiler.

6.6.1 Fortran77 vs Fortran90

When the compilers used are Fortran77 compilers, it is important to make sure the code is compatible with Fortran77. Most things which are possible in Fortran90 can also be done in Fortran77, albeit sometimes with a bit extra work involved.

6.6.2 Allocation of Space for Variables

When allocating space for variables on either of the stacks `CZ`, `DZ`, `IZ`, `LZ`, or `RZ` (see. app. E for info on these), it is of **paramount importance** that enough space is allocated, as to avoid overwriting other areas when writing to the variable. It is preferable to allocate the correct amount of space, but if at the time of allocation that number is unknown, make sure to allocate plenty.

Allocating Space for Variables of unknown size

When allocating space for a variable of unknown size, it is possible in Fortran90 to allocate a space of "yet to be defined" size using the variable definition:
`REAL, ALLOCATABLE, DIMENSION(:,:) ::`.
Allocating space like that is **not** possible in Fortran77, but can be circumvented using MMS, see app. E.

6.6.3 Initialization of Variables

Depending on the compiler used, Fortran may or may not initialize variables (scalars and arrays alike) once space has been allocated, that is - deleting any previous data that may exist in the designated area for the variable data. As a precaution, it is therefor advised always to initialize data areas. This can be done manually, or the CFX routine `SET_A_0` can be used in either of two following ways:

```
CALL SET_A_0 (VARNAME ,SIZE)
CALL SET_A_0 (STACK(pPOINT) ,SIZE)
```

where `VARNAME` is the name of the variable, `SIZE` is the size of the variable e.g. 12 for a 2×6 array, `STACK` is the appropriate stack for the type of variable(`CZ`, `DZ`, `IZ`, `LZ`, and `RZ`), and `pPOINT` is the stack pointer to the variable.

6.6.4 Writing to and reading from the stacks

As the stacks are one dimensional arrays, when operating with two- or higher dimension arrays, it is important to read the variables the same way they were written e.g. if the array is written one row at the time, it should be read one row at the time. Writing to and reading from the stacks is further explained in app. E.

7
Preliminary Analyses

In this chapter an analysis is conducted to find out if and how the different parameters in a CFD simulation influences on the results. The analysis is limited to deal with four parameters: mesh fineness, number of coefficient loops, time step size and use of initial conditions from another simulation.

The analyses show that all parameters have a significant influence in the results, especially when looking at cross-wind loads. These loads are very dependent on vortex shedding and to capture this flow phenomenon, requirements for the mesh fineness and time step size are apparent. The streamwise loads show only little dependency on the different parameters in question.

The analyses are not used to determine the actual size of the errors made when using coarse settings but merely to find out if the particular parameter has an effect and and what order the errors are.

In order to get the most information in the least amount of computation time a number of preliminary analyses can be carried out in the aid of determining the choice of mesh quality and fineness, time duration of simulation, time step size etc. In this project it has turned out that the simulations, taking all aspects into account, will be very time consuming. An analysis using ideal settings would therefore take a considerable amount of time taking the limited computer power available into account. Therefore it has been decided to turn the attention towards the method of modeling FSI with CFD and accept that the simulations may not give the best possible result. Instead of using the preliminary analysis to set up the best possible simulation, the analyses are conducted to find out how big an influence it has on the result when changing different parameters to reduce the computation time. The analysis are limited to four parameters listed below:

- Mesh fineness
- Coefficient loops
- Time step
- Time duration

From a previous to the next analysis of one of the parameters, the most coarse settings from the previous analysis have been used for the following. This is done well aware that an alteration to one parameter may have an influence on the others. The analyses are conducted with simulations with a moving structure determined on the basis of a structural modal model. This will not be dealt with more here and a reference is made to chap. 6 where a description of the model used can be found. The setup of the simulations in CFX is described in chap. 5.

7.1 Mesh fineness

The mesh quality is one of the most important factors when trying to reduce the computation time. In a normal simulation the mesh should be fine enough to resolve the flow into sufficiently small volumes for the numerical scheme to converge and to capture the flow phenomena important for the specific analysis. In this case the presence of vortex shedding have a big impact on the mesh fineness needed. Vortex shedding is important in FSI as the cross stream dynamic response of the structure is closely related to this phenomenon. When dealing with a 3D mesh, a refinement in all directions leads to a significant increase of the number of nodes in the computation. Therefore running the simulations with a very coarse mesh could reduce the computation time significantly. By doing

this it might not be possible for the mesh to resolve the flow properly which leads to results deviating from the results that could be obtained by a mesh optimized for the specific flow.

In order to investigate the influence of using a very coarse mesh, three simulations are run and finally compared. The comparison of the results are based on the load series on the structure extracted via Fortran routines. The loads extracted during the CFX runs are used to determine the response of the structure and a large deviation from the "correct" loads will also cause a response deviation.

The CFX- and Fortran files for this analysis can be found on the DVD in [*DVD:\Preliminary_ Analyses\CFX_ files\Mesh_ Analysis*].

7.1.1 Mesh

The three meshes are generated by the hyperbolic mesh program described in chap. 2. The MatLab files used can be found in [*DVD:\Preliminary_ Analyses\ Mesh_ files*] and are called *Pre_ Sim_ Mesh_ X.m* where $X = 1, 2, 3$. The input parameters for the mesh generation are listed in tab. 7.1 and 7.2. In the latter a row containing the number of nodes in the mesh has been included.

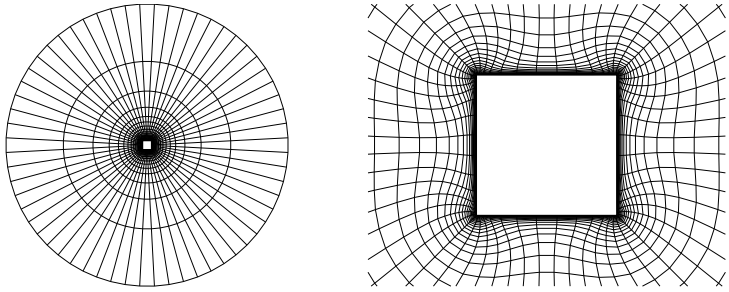
Table 7.1: Constant parameters for all three meshes

Parameter	Value	Parameter	Value
mode	0 (cyclic mesh)	dis	0.001
BREAKS	[1 2 3 4 5]	eps	0.2
a1	[0 -1 0 1 -1 0 1 0];	eps2	0.1
b1	[0 -1 0 1 [-1 0 1 0];	eps3	0
x1{1:4}	[0 0.35 0.65 1];	rot	45
y1{1:4}	[0 0.2 0.8 1];	nstart	1
a2	[0.1 0.1 0.1 0.1];		
b2	[0.1 0.1 0.1 0.1];		
xs	[0 0.4 0.8 1];		
ys	[0 0.6 0.95 1];		
as	2		
bs	0.1		

Table 7.2: Variable parameters for the three meshes

Parameter	*Pre_Sim_Mesh_1.m*	*Pre_Sim_Mesh_2.m*	*Pre_Sim_Mesh_3.m*
`n`	`[15 15 15 15]+1;`	`[20 20 20 20]+1;`	`[25 25 25 25]+1;`
`L`	`0.004;`	`0.004;`	`0.004;`
`N`	`35;`	`40;`	`45;`
`expan_fac`	`[0.34 0.9 0.75];`	`[0.285 0.9 0.70];`	`[0.24 0.9 0.65];`
`ntotal`	`70;`	`75;`	`80;`
Number of nodes	155960	257640	390420

An example of the generated meshes is shown in fig. 7.1a which is the most coarse mesh. Fig. 7.1b shows a close up of the mesh around the structure where the element distribution is visible.



a) Full mesh

b) Close up of mesh around structure

Figure 7.1: Example of generated mesh

7.1.2 Time step and duration

Determination of time duration and time step is based on model size, shedding frequency and eigenfrequencies of the structure. Requirements for each of the elements are determined and the lowest time step is adapted. The time duration also have to be long enough to ensure that any possible effect of the initial conditions are dissipated away.

Time duration

The simulation time due to the structure eigenfrequencies are based on the lowest eigenfrequency and a wanted number of minimum displacements through the corresponding eigenmode. In eq. 6.11 the lowest circular eigenfrequency is listed as $\omega = 0.39\,\frac{\text{rad}}{\text{s}}$. The period of motion, T, is determined by:

$$T = \frac{2\,\pi}{\omega} = 16.11\,\text{s} \tag{7.1}$$

The period of motion during the simulations are expected to be equal to or lower than the period in eq. (7.6) and using this as the basis for the time duration of the simulations is therefore on the safe side to ensure a sufficient number of motions.

As the computations are very time consuming the time duration of the simulations are kept relatively short. This is chosen well aware, that all affects over time may not be modeled.

The time duration of the simulations is chosen to $t = 60\,\mathrm{s}$. This is a conservative choice based on the highest period of motion of the structure determined in eq. (7.6). This duration allows a minimum of three motion periods while leaving time for the effects of the initial conditions to dissipate away. The simulations will show if this simulation period is in fact sufficient to dissipate the initial conditions in time to also ensure enough information on the structure motion.

Time step

The time step selection is based on requirements for the time step due to model size and vortex shedding.

The required time step due to model size is based on the chosen dimensions of the structure and the wind speed. In fig. 5.1 a streamwise side length of 20 m and a wind velocity of $20\,\frac{\mathrm{m}}{\mathrm{s}}$ is specified. The time an air particle use to pass the entire structure side have to be divided into a sufficient amount of time steps. It is assumed, that 50 time steps is sufficient. This results in a required time step size of:

$$\Delta t_{model} = \frac{\left(\frac{20\,\mathrm{m}}{20\,\frac{\mathrm{m}}{\mathrm{s}}}\right)}{50} = 0.02\,\mathrm{s} \tag{7.2}$$

In order to capture the vortex shedding on the lee side of the structure the time period of the shedding have to be divided into a sufficient number of time steps. The dimensionless frequency of the vortex shedding is known as the Strouhal number. For a square cylinder the Strouhal number is given by:

$$St = \frac{f\,H}{u_\infty} \tag{7.3}$$

where

f is the shedding frequency of the vortices, [Hz]

H is the side length of the square cylinder, [m]
u_∞ is the free stream velocity in the x-direction, $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$

According to [CEN/TC250 2005, pp. 116-117] the Strouhal number for a square cylinder is 0.12. With a cylinder side length of 20 m and a free stream velocity of 20 $\frac{\mathrm{m}}{\mathrm{s}}$ the shedding frequency, f, is determined by eq. (7.3) to:

$$f = \frac{St\, u_\infty}{H} = 0.12\,\mathrm{Hz} \tag{7.4}$$

The shedding frequency in eq. (7.4) corresponds to a shedding period of 8.33 s. A number of 50 time steps per shedding period is assumed sufficient corresponding to a time step size of:

$$\Delta t_{vortex} = \frac{8.33\,\mathrm{s}}{50} = 0.17\,\mathrm{s} \tag{7.5}$$

Based on eq. (7.2) and (7.5) the time step is chosen to $\Delta t = 0.02\,\mathrm{s}$ for the simulations with the different meshes.

CFL condition

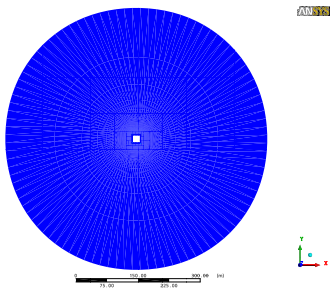
The CFL (Courant-Friedrichs-Lewy) condition is a mathematical condition for certain algorithms, solving partial differential conditions, to be convergent. The condition have no impact on the algorithm stability. In CFD the condition is related to the Courant number, C_r, given by:

$$C_r = \frac{u\,\Delta t}{\Delta x} \tag{7.6}$$

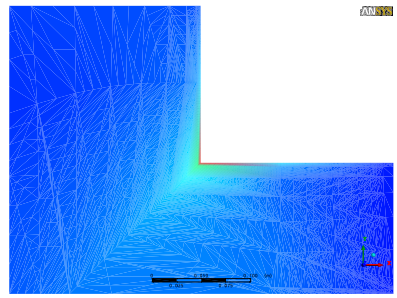
where

u is the fluid velocity, $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$
Δt is the time step size, [s]
Δx is the mesh size, [m]

A condition for the courant number can be used to determine the time step size. If the condition is, that a fluid particle must be captured at least once within every control volume it passes through, this leads to the condition that $C_r \leq 1$. The grid size and velocity are not constant and therefore the Courant number varies throughout the domain. In order to check the Courant number with a time step size of 0.02 s as specified above fig. 7.2a shows the Courant number in a plane in the domain and 7.2b a close up of one of the corners on the structure, which is representative for the three other corners.



a) Courant number in whole plane



b) Courant number at one corner

c) Legend, Courant number

Figure 7.2: Example of Courant number in simulation with $\Delta t = 0.02\,s$

It is seen from fig. 7.2 that $C_r < 1$ in the entire domain except close to the corners where the value increases significantly. In order to keep the Courant number below 1 in the entire domain the time step size should therefore be reduced by a factor corresponding to the highest C_r value in the domain. In this case the time step size is approximately 100 times too large for this to be fulfilled. The time step size should therefore have been reduced to 0.0002 s which with the relatively limited computer power would make the simulations be extremely time consuming (hundreds of days).

Looking at fig. 7.2 it is seen that the Courant number rapidly decreases when moving away from the corner of the structure. Therefore to save computation time, it is decided not to incorporate the CFL condition in this analysis keeping in mind that if the algorithm solving the flow equations seems to be non convergent, the reason might be the exclusion of the CFL criterion.

7.1.3 Results

The results of the simulations are presented in the form of the total simulation time and load series. The simulation time is taken into account as these analyses are conducted to investigate the errors which must be accepted to reduce the computation time. To see which effect the individual parameter has on the computation time the total time is presented.

The simulations with the finest mesh (*Mesh 3*) with the setting *1[m^3 s^-1]/(Wall Distance)* for the mesh stiffness, resulted in an error where the solver terminated

due to the wall distance somewhere in the domain becoming 0. The mesh stiffness was therefore changed to the built in CFX setting *Increase near Small Volumes* but this also resulted in an error, this time the creation of a negative volume element. Finally the setting was changed to *Increase near Boundaries* which did not return an error.

For the simulations with different mesh fineness the simulation time is listed in tab. 7.3 along with the number of nodes in the simulation. The specific simulation is given by the CFX file-name.

Table 7.3: Number of nodes and total simulation time for different mesh fineness

Simulation	Number of nodes	Simulation time [hhh:mm:ss]
Pre_Analysis_Mesh_1.cfx	155960	231:25:42
Pre_Analysis_Mesh_2.cfx	257640	283:53:20
Pre_Analysis_Mesh_3C.cfx	390420	*Simulation time n/a*

The inclusion of the *C* in the last file-name relates to the fact that it was the 3rd setting of the mesh stiffness as mentioned. The simulations with the 3rd choice of mesh stiffness also terminated before running the total number of time steps. The reason for this solver termination could not be found as the *.out* file did not contain anything (file size - 0 KB) which was also the case for the *.res* file. The reason for the file size being 0 is probably due to lack of storage space on the cluster where the simulations were run and therefore no space was available to write the files. A couple of times during the project period this behavior was encountered. Even though the solver failed the simulation time before the failure was above 50 s. It was decided that this amount of results was enough for this comparison. This decision was mainly based on the long computation time it would require to get results for the last 10 s. It will later be shown, that the loads have almost stabilized within the first 50 s and no real deviation would be obtained by running a new simulation.

The loads chosen for the comparison are the streamwise and cross stream loads at the nodes indicated in fig. 7.3.

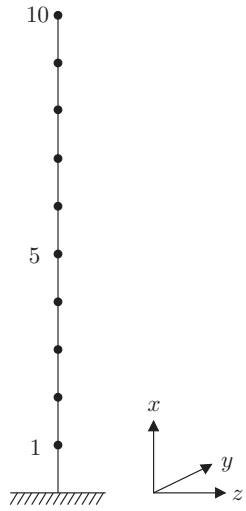


Figure 7.3: Indication of chosen nodes for load comparison, local coordinates

Regarding the loads in the streamwise direction it is expected, that the mesh fineness will not have a significant impact as the streamwise loads does not depend as much on the expected vortex shedding as the cross-wind loads.. The streamwise loads at the indicated nodes are shown in fig. 7.4.

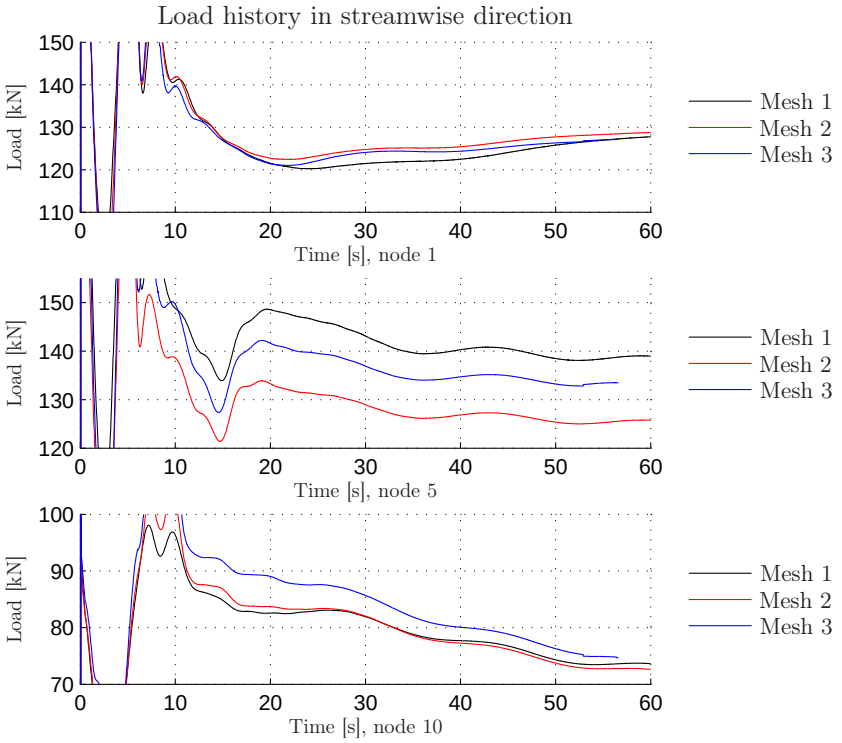


Figure 7.4: Streamwise load history

It is seen in fig. 7.4 that the loads show only little dependence on the mesh fineness as expected. The load series is almost identical for all three meshes. After about 20 s the loads seem to stabilize a bit and close inspection of fig. 7.4, especially the one for node 5, show that the structure movement causes an almost periodic variation in the load. This is said keeping the relative short simulated time in mind. This behavior is not as apparent for node 1 and 10 respectively but some load dependency on the structure movement can be seen.

The loads in the cross stream direction are very dependent on the vortex shedding expected to occur on the lee side of the structure. The mesh fineness is very important in order to fully capture the vortices. By making the mesh more coarse the loads are expected to deviate more and more from the ones obtained by the finest mesh. The load series of the cross stream loads at nodes 1, 5 and 10 are shown in fig. 7.5

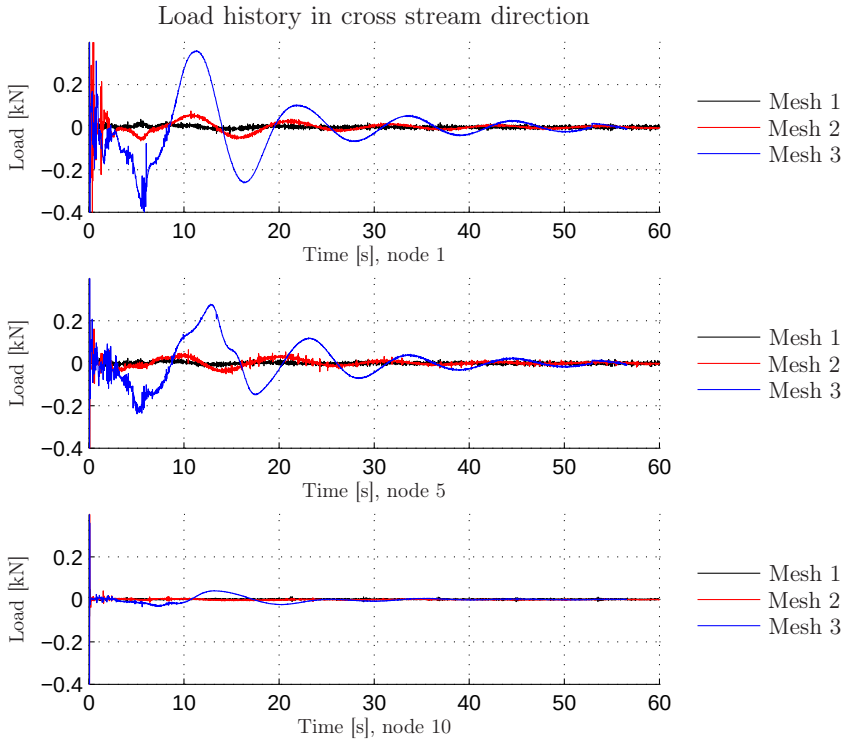


Figure 7.5: Cross stream load history

As expected the cross stream loads show a large deviation as the mesh coarseness increases. The progress of the loads obtained with the finest mesh (*Mesh 3*) could indicate, that vortex shedding is present. The size of the loads indicate otherwise as these are relatively small. Also, looking at the progress of the loads with the most coarse mesh (*Mesh 1*) indicate that this mesh is inable to resolve the flow properly to capture the vortices. The cross stream load seems to fluctuate around 0 during the entire simulation.

7.2 Coefficient loops

In the simulations presented in sec. 7.1 the maximum number of coefficient loops was set to 10. In the simulation with the most coarse mesh, (*Pre_Analysis_Mesh_1.cfx*), the maximum number of coefficient loops was used for every time step. This is due to the inability of the coarse mesh to resolve the flow sufficiently. The use of 10 coefficient loops for all time steps have a large impact on the computation time. Therefore in this section the effect of lowering the

maximum number of coefficient loops to 5 and 2 respectively is investigated. The simulations are run with the most coarse mesh and finally the results are compared to the simulation with the most coarse mesh from the previous section.

The CFX- and Fortran files for this analysis can be found on the DVD in [*DVD:\Preliminary_Analyses\CFX_files\Coefficient_Loops_Analysis*].

The simulation time is listed in tab. 7.4. The number of nodes are the same as the one with the coarse mesh in tab. 7.3. The results for this simulation are presented here also.

Table 7.4: Number of nodes and total simulation time for different maximum number of coefficient loops

Simulation	Max. Coeff. Loops	Simulation time [hhh:mm:ss]
Pre_Analysis_Mesh_1.cfx	10	231:25:42
Pre_Analysis_Coeff_5.cfx	5	*Simulation time n/a*
Pre_Analysis_Coeff_2.cfx	2	70:51:01

It is seen in tab. 7.4 that the computation time have been reduced by approximately 70 % by reducing the number of coefficient loops from 10 to 2. As it was the case with the simulation with the finest mesh in sec. 7.1 the solver failed when running the simulation with a maximum of 5 coefficient loops. The two simulations were run at the same time but it has not been investigated if this had anything to do with the solver failure.

A new simulation with a maximum of 5 coefficient loops were not run with the same argumentation as used for the simulation with the finest mesh in the previous section.

The streamwise and cross stream load series are shown in fig. 7.6 and 7.7 where the node numbers refer to the nodes shown in fig. 7.3.

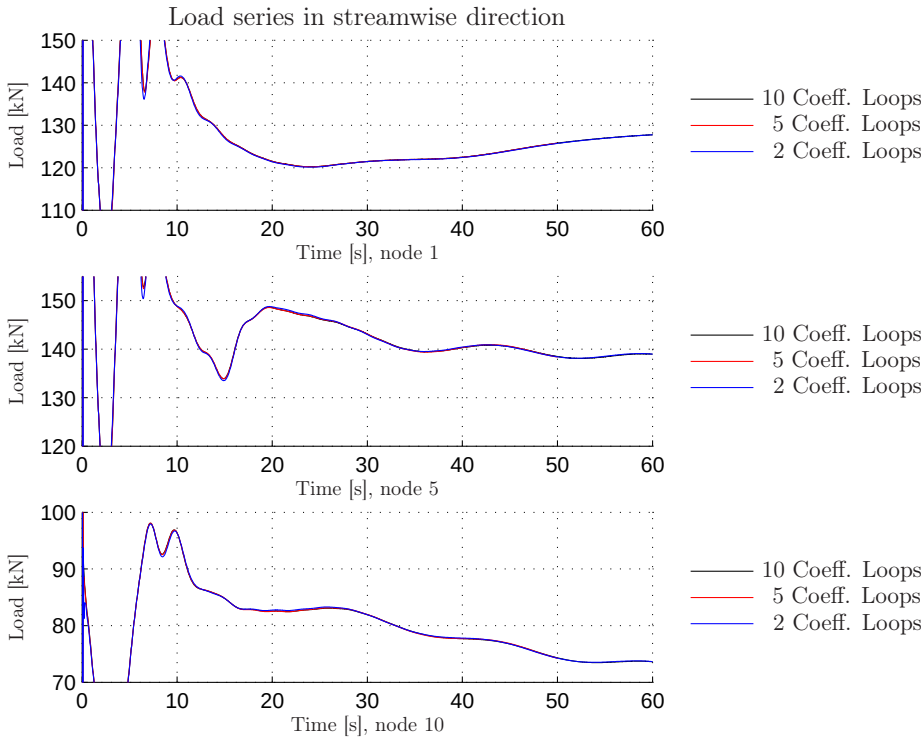


Figure 7.6: Streamwise load history

As it was the case with the simulations with different mesh fineness the loads in the streamwise direction show negligible dependency on the number of coefficient loops wherefore only the same small dependency on the structure movement is visible. The effect in the cross stream direction is on the other hand evident.

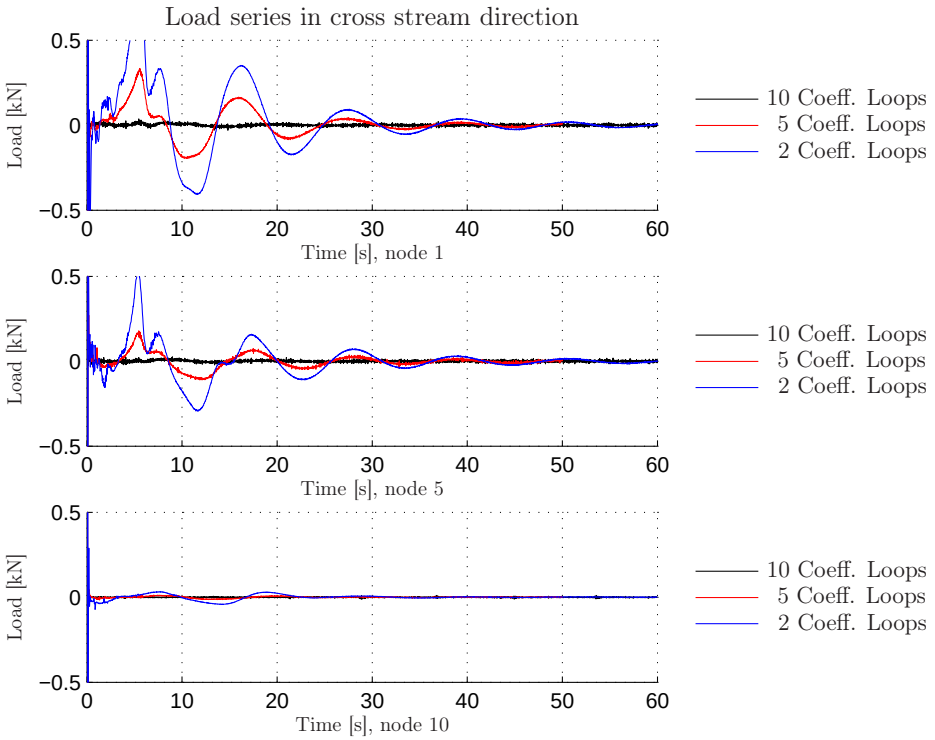


Figure 7.7: Cross stream load history

It is seen that there is a significant difference in the cross-wind loads at the beginning of the simulations which over time gets closer to the ones determined by the simulation with 10 coefficient loops.

At the instant the simulation is started the structure is exposed to wind at $20\,\frac{\mathrm{m}}{\mathrm{s}}$. The instantaneous large gradients which arise to move the wind around the structure seems to cause large peaks in the cross stream loads which takes a considerable amount of time to dissipate away and the flow to stabilize. To further illustrate this behavior the cross stream displacement of the top node (node 10) is shown in fig. 7.8 for both simulations.

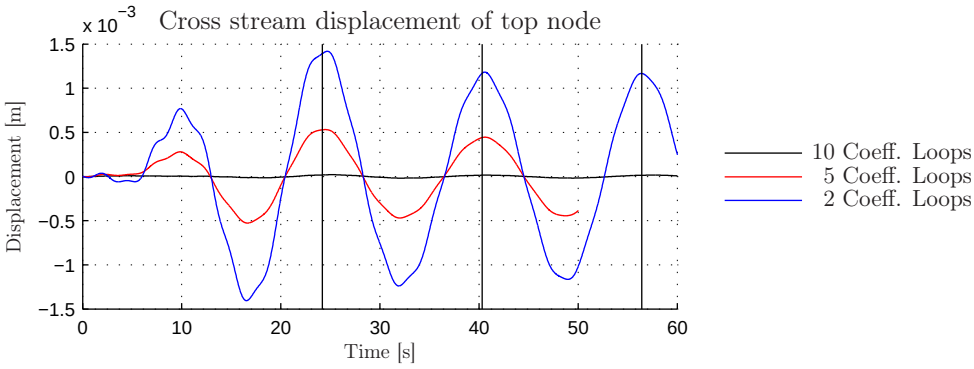


Figure 7.8: Cross stream displacement of top node

It is seen in fig. 7.8 that the cross-wind displacements also deviate considerably. Contrary to the load the displacement does not show the same decrease as the load over time. The lowest eigenfrequency is in eq. 6.11 listed as $0.39\frac{\text{rad}}{\text{s}}$ corresponding to a period of 16.11 s determined in eq. (7.6). The vertical lines included in fig 7.8 are placed with one period in distance. it is seen that the structure top moves very close to the vibration period of the 1^{st} eigenmode. Even though the load decreases over time the displacement stays almost at the same level.

Instead of introducing a relatively diffuse error by setting a maximum number of coefficient loops it is possible to reduce the computation time and have a better idea of the error in the simulations. This could have been done by increasing the residual target from the 10^{-4}, which have been used, and to a higher value. By doing this the number of coefficient loops necessary at each time step might decrease but still leave the simulation to determine the number necessary. Controlling the error this way have not been used in this project.

7.3 Time Step

The time step size is an important parameter in CFD simulations. When dealing with turbulent flows the time step have to be low enough to resolve the flow into small steps to make the numerical scheme converge. Areas of high turbulence means that the flow properties can change very rapidly in one point over time. Modeling this requires a sufficiently low time step.

On the other hand the time step size also has a great impact on the computation time in the simulations. In sec. 7.1.2 the time step for the present simulations

was determined to 0.02 s when taking model size and vortex shedding into consideration. This means each second of simulation time is modeled by 50 time steps and for each time step a number of coefficient loops are run. In sec. 7.2 the effect of lowering the number of coefficient loops was investigated. In this section the effect of the time step size is presented. It is chosen to run two more simulations equal to the one in the previous section with a maximum of two coefficient loops to reduce the computation time. The time step size is set at 0.05 s and 0.1 s respectively.

The CFX- and Fortran files for this analysis can be found on the DVD in [*DVD:\Preliminary_Analyses\CFX_files\Time_Step_Analysis*].

Table 7.5: Number of nodes and total simulation time for different time step size

Simulation	Time step size	Simulation time [hh:mm:ss]
Pre_Analysis_Coeff_2.cfx	0.02 s	70:51:01
Pre_Analysis_tstep_0_05.cfx	0.05 s	29:00:39
Pre_Analysis_tstep_0_1.cfx	0.10 s	14:38:26

It is seen from tab. 7.5 that the simulation time varies almost linearly with the time step size. In this case this is due to the fact, that all simulations are set to use a maximum of two coefficient loops per iteration which they all do. Therefore doubling the time step size and hereby halving the total number of time steps in the simulations will halve the simulation time. If the simulations were run with a finer mesh and a larger number of coefficient loops the situation could have been different. The simulation with the lowest time step might have converged using a smaller number number of coefficient loops while the large time steps would result in the use of several more loops. This could have the effect, that the computation time saved by increasing the time step size would not have been as significant as in this case. This matter has not been investigated further in this project.

As with the previous tests the streamwise and cross stream load series are plotted in fig. 7.9 and 7.10.

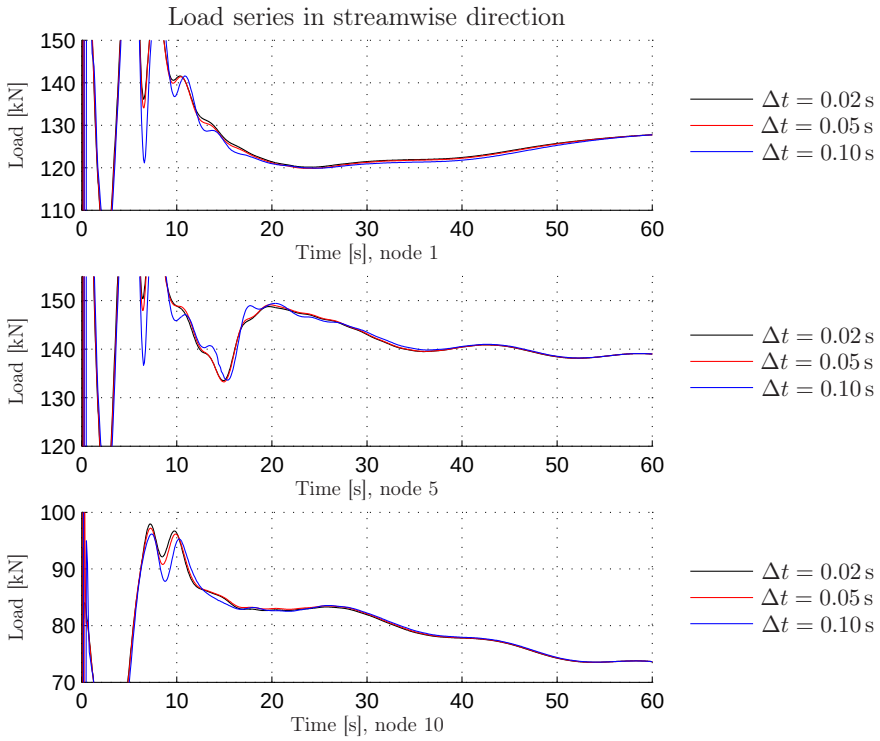


Figure 7.9: Streamwise load history

The streamwise load show only little difference when changing the time step size and the progress all follow the same path. Again it is the load progress in the cross stream direction which show a significant deviation.

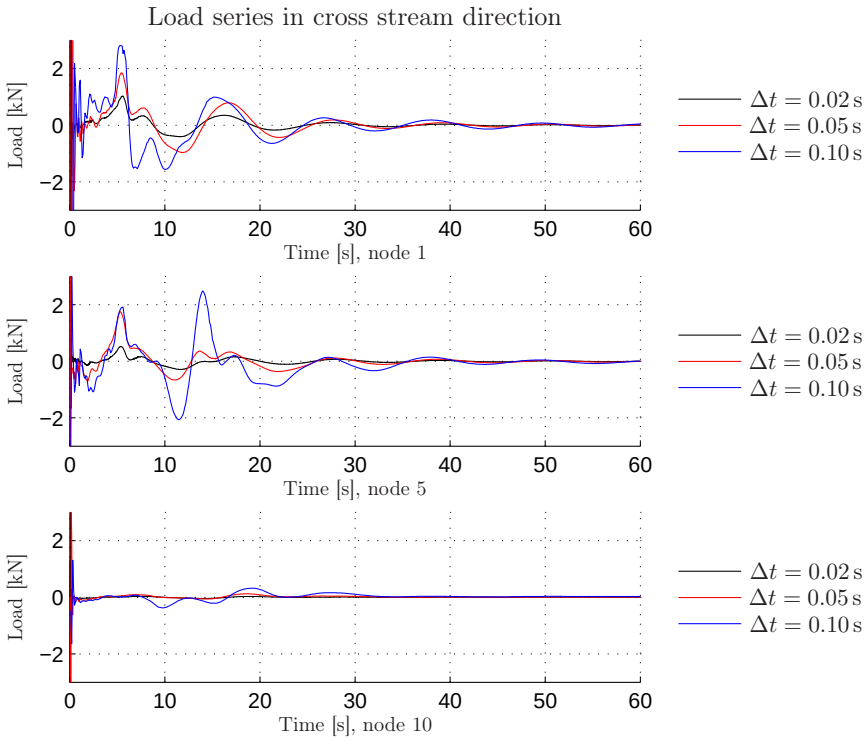


Figure 7.10: Cross stream load history

The cross stream loads are very irregular in time over the first $15-20$ s whereafter it becomes more or less regular. It is seen in fig. 7.10 that the load for all time step sizes approach the same value at the end of the simulation but the larger the time step size the longer it takes for the simulation to stabilize. The simulation with $\Delta t = 0.1$ s also shows some large peak values in the beginning. As mentioned earlier, the structure is exposed to wind instantaneously when the simulation is started and in connection with the low number of coefficient loops and coarse mesh the large time step size causes the large gradient areas to be very poorly resolved.

7.4 Initial Conditions

In this section results from simulations with time duration shorter and longer than the 60 s used above was intended to be presented. By inspection of fig. 7.4 - 7.5, 7.6 - 7.7 and 7.9 - 7.10 the loads seem to have almost stabilized in all cases. It is therefore chosen not to run simulations with different time durations and

use 60 s as the duration for the future simulations in this project.

Instead it is chosen to run two simulations with the use of initial conditions from a steady state simulation to see whether or not this has an effect on the time it takes for the effects of the initial conditions to dissipate away. In sec. 7.2, regarding the number of coefficient loops, it was found that lowering the number had a big effect on the cross stream loads due to the large gradient flow. This was also the case in sec. 7.3 regarding the time step size. The initial conditions used could also have been results from a transient run but in this case steady state simulations have been used as mentioned.

The initial conditions for these simulations are steady state simulations run with the exact same settings as the transient runs. The maximum number of iterations is set to 100. Two different steady state simulations are run, one with the most coarse mesh, *Mesh 1*, and one with the finest mesh, *Mesh 3*. The simulation files can be found in [*DVD:\Preliminary_Analyses\CFX_files\Initial_Conditions*]. The transient simulations, *Pre_Analysis_Mesh1_Ini_Cond.cfx* and *Pre_Analysis_Mesh1_Ini_CondB.cfx*, used for the comparison are simulations set to a maximum of 2 coefficient loops and a time step size of $\Delta t = 0.05\,\mathrm{s}$ corresponding to the red curve in fig. 7.10. Regarding the computation time needed the three simulations (one without and two with initial conditions) showed no real difference with a deviation of only a few minutes.

The results of the simulations are shown in fig. 7.11 - 7.12 for the streamwise and cross stream loads respectively.

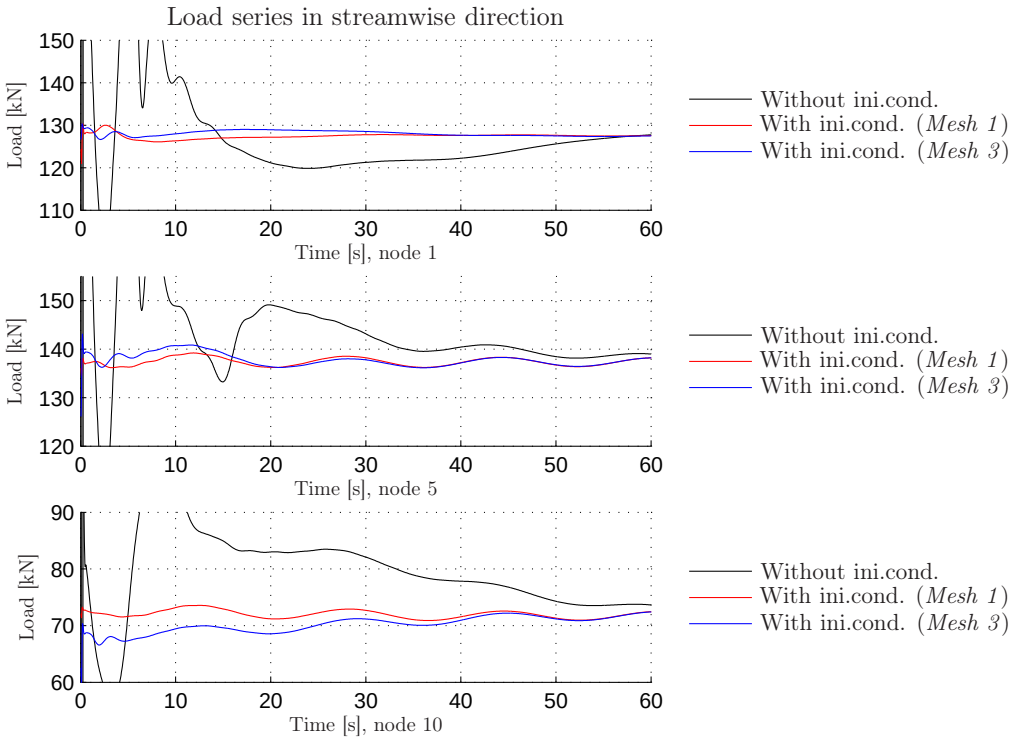


Figure 7.11: Streamwise load history

For the streamwise direction the load at the beginning of the simulation is almost at the same level as the end when using the initial conditions from the steady state simulations. The load is periodic over almost the entire simulated period with the load converging towards a constant amplitude. The dissipation of the effects from the initial conditions in the original transient simulation can be completely removed by using a steady state simulation as initial conditions no matter which mesh chosen for the steady state simulation.

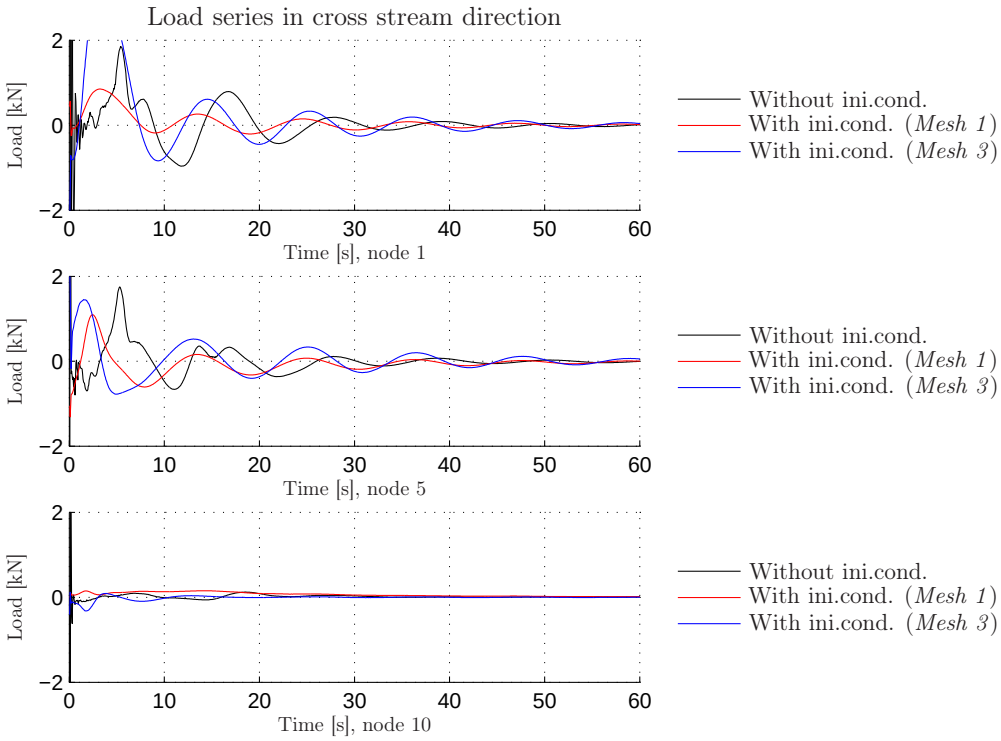


Figure 7.12: Cross stream load history

In the cross-wind direction it is seen in fig. 7.12 that the use of a steady state simulation does not seem to stabilize the simulation faster. Only the size of the loads in the beginning of the simulations seems to be affected by the initial conditions. Is was expected, that the simulation with the initial conditions determined with the finest mesh would produce better results but inspection of fig. 7.12 shows otherwise. The reason for this could be, that the condition at the end of the steady state with the finer mesh is further away from what the coarse mesh can actually represent.

7.5 Observations

The preliminary analyses above have been conducted to determine which parameters have the largest effect on the results achieved. This is done to find out how to save computation time without losing to much information from the simulations.

It is found in the analyses that the loads in the streamwise direction show almost no deviation no matter which parameter is altered. The loads attain nearly the same value and follow the same load path during the respective simulations. A lot of computation time can therefore be saved if the loads of interest only include the streamwise loads.

Regarding the cross stream loads, these show a considerable dependency on all parameters which have been included in this analysis. It is difficult to say which parameter have the largest effect. To be able to make this determination the simulations should have been run with exactly the same settings and only one parameter changed for each simulation. These simulations could then have been compared and the parameter with the largest effect might have been clear. The problem with these analyses is that all simulations would have been very time consuming and the limited computer power available would have caused the analyses to take a very long time to conduct.

The way the analyses are conducted in this case results in the fact, that it is only possible to determine if the parameter has an impact in the results but not actually the size of this impact.

The analysis in sec. 7.4 shows, that when using a steady state simulation as initial conditions it is important that the two simulations are able to represent the same state, i.e. both simulations must be able to capture the same flow phenomena. As it can be seen, a steady state simulation able to capture more flow properties than the transient simulation where it is used as initial condition causes the simulation to use longer time for the effects of the initial conditions to dissipate away and the flow to stabilize. Again this condition applies mainly to the loads in the cross-wind direction.

7.6 Capturing Vortex Shedding

From the analyses in secs. 7.1 - 7.4 it has been determined that the loads in the cross stream direction show a significant dependency on the parameters mesh fineness, number of coefficient loops, time step size, and the use of initial conditions. However in these tests the vortex shedding has not been captured. To examine what it would take to capture this phenomenon, four additional simulations have been run. These simulations are carried out on 2D meshes rather than 3D to save computation time. In the 2D simulations two different meshes are used, both with a time step size of 0.02 s and 0.01 s, respectively. The coarsest

of the two meshes corresponds to first layer in the z-direction of the finest mesh from the analyses in sec. 7.1.1 (*Mesh 3*), extruded to generate a 2D mesh of one cell thickness. The second and finer mesh is generated to have twice the number of boundary nodes, and twice the number of marching layers.

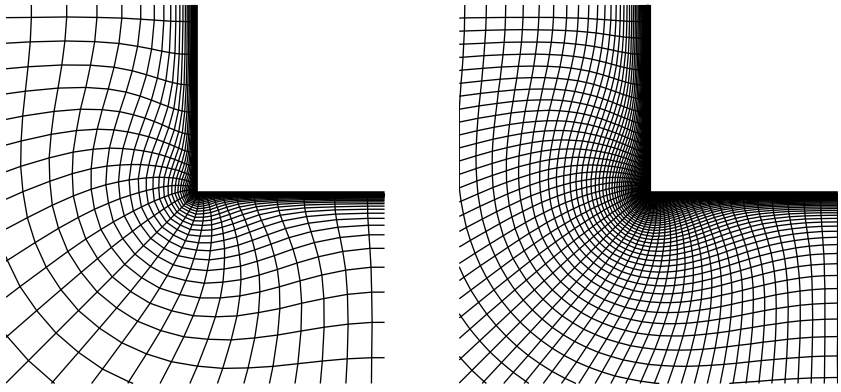
The two meshes are generated by the same hyperbolic mesh program as the 3D meshes in sec. 7.1.1.
The MatLab files used can be found in [*DVD:\Preliminary_ Analyses\Mesh_ files*] and are called *Pre_ Sim_ 2D_ norm.m* and *Pre_ Sim_ 2D_ fine.m*.

Most of the parameters for the two 2D meshes are given in tab. 7.1 with the difference that `rot` is set to 0 and `ntotal` is set to 2. The remaining input parameters that vary between the two meshes are given in tab. 7.6.

Table 7.6: Varying parameters for the two 2D meshes

Parameter	*Pre_ Sim_ 2D_ norm.m*	*Pre_ Sim_ 2D_ fine.m*
`n`	`[25 25 25 25]+1;`	`[50 50 50 50]+1;`
`L`	`0.004;`	`0.001;`
`N`	`45;`	`90;`
`expan_fac`	`[0.24 0.9 0.65];`	`[0.12 0.7 0.2];`
Number of nodes	9000	36000

To show the difference in mesh fineness, a closeup of a corner of the structure is shown in fig. 7.13 for each of the two meshes. It should here be noted that the coarse mesh in fig. 7.13a corresponds to the first layer in the z-direction of the finest mesh in sec. 7.1.1.



a) Coarse 2D mesh *b)* Fine 2D mesh

Figure 7.13: Example of mesh fineness for the two 2D simulations

All four simulations are set to run for 300 s, but due to problems with memory allocation in CFX, none of the analyses ran the full 300 s. The results of the analyses is however still conclusive, by inspection of figs. 7.14 and 7.15, as it shows a significant difference in the loads for the cross stream direction when using a finer mesh than in the 3D analyses.

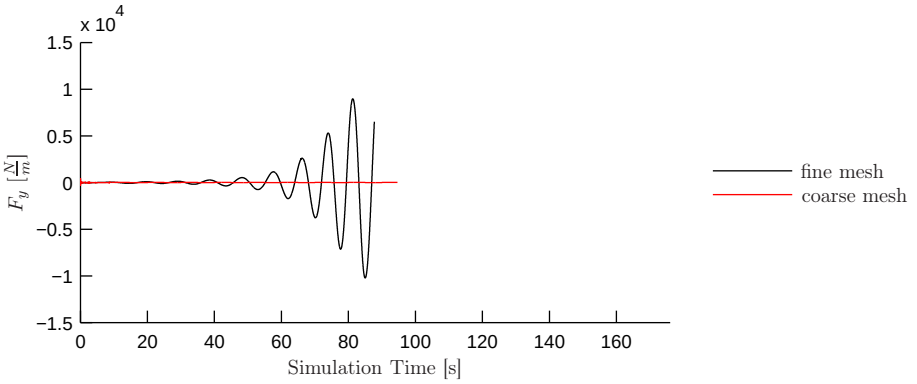


Figure 7.14: Cross stream load history for time steps of 0.01 *s*

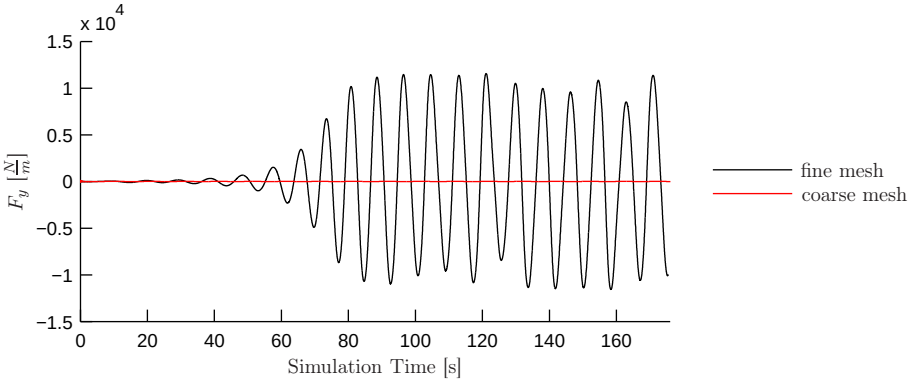
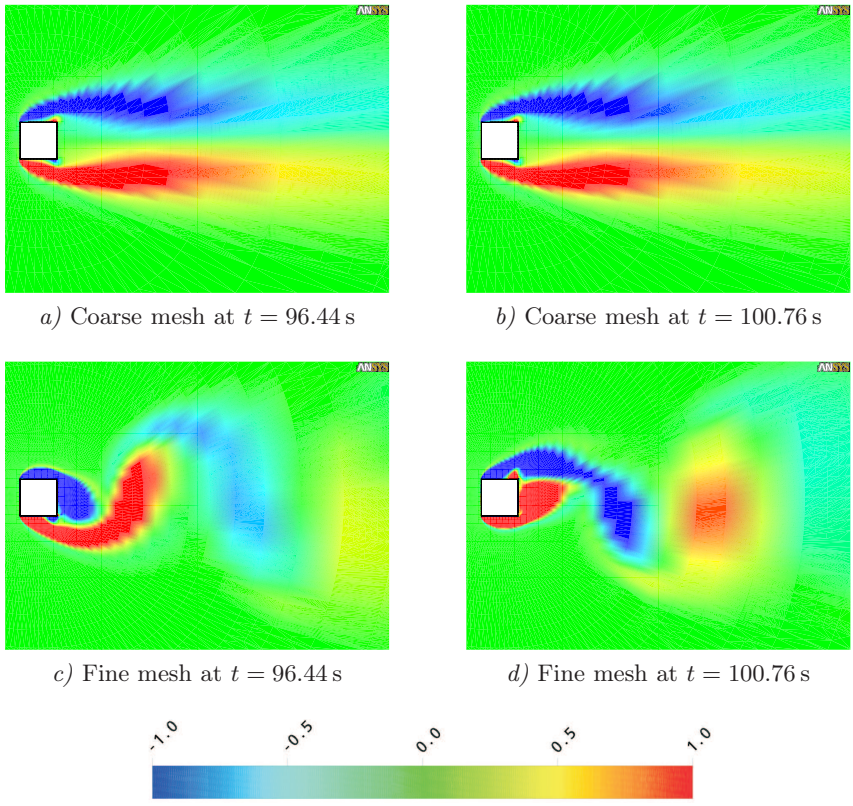


Figure 7.15: Cross stream load history for time steps of 0.02 *s*

It can be seen from the two figures that whereas halving the time step size from 0.02 s to 0.01 s makes little difference, the doubling of nodes on the structure boundary and in the marching direction shows a huge deviation, and a very clear behavior corresponding to that of vortex shedding is observed. To further demonstrate the appearance of vortex shedding, figs. 7.16a to 7.16d show the vorticity in the domain at two separate time steps, one corresponding to where

F_y has a maximum, 96.44 s on fig. 7.15, and the other corresponding to the following minimum, 100.76 s on fig. 7.15.



a) Coarse mesh at $t = 96.44\,\mathrm{s}$

b) Coarse mesh at $t = 100.76\,\mathrm{s}$

c) Fine mesh at $t = 96.44\,\mathrm{s}$

d) Fine mesh at $t = 100.76\,\mathrm{s}$

Figure 7.16: Vorticity $[s^{-1}]$ for each of the two meshes at $F_{y,max}$ and $F_{y,min}$ respectively

Figs. 7.16c and 7.16d clearly shows the vortex shedding. The shedding frequency can be estimated as

$$f = \frac{1}{2\,(100.76 - 96.44)\,\mathrm{s}} = 0.1157\,\mathrm{s}^{-1} \tag{7.7}$$

which corresponds well with the shedding frequency of $0.12\,\mathrm{s}^{-1}$ given in (7.4).

To conclude this section it is seen that the ratio of number of nodes between the coarsest mesh and the finest mesh in sec. 7.1.1 is ≈ 2.5, and the ratio of number of nodes between the finest and the coarsest mesh in this section is 4. Thus, assuming that the computation time scales linearly with the number of nodes in the domain, this suggests that simulations carried out on a 3D mesh,

with a resolution similar to the finest of the 2D meshes, would take approximately 10 times longer than on the coarse mesh in sec. 7.1.1, which has been used in the further analyses of the project. Also taking into account the time step size, number of coefficient loops this means that for a 60 s 3D a simulation to be able to capture vortex shedding, it would require a simulation time of 300-1000 hours, as opposed to the $\approx$ 30 hours the simulations run in this project takes.

8

Effect of Modeling Aeroelasticity

In this chapter the effect of modeling the aeroelastic response of a structure is analyzed. This is done by comparing two different simulations, one conducted on a stationary structure and one where aeroelasticity has been included. The two simulations are hereafter compared by considering the modal coordinates.

It is found, that modeling the aeroelasticity have some effect on the structure movement. In this case the simulations returned results opposite of what was expected. It was not possible to determine any notable positive or negative aeroelactic damping on the basis of the conducted simulations.

It is common procedure, when calculating wind loads on structures, not to include the aeroelastic response of the building. This means that the wind loads are calculated on a stationary structure, and subsequently used to find the response of the structure. The purpose of this analysis is to show, whether modeling the aeroelasticity has any notable effect on a simulation or not. To test this, two different simulations are performed, a stationary simulation and an aeroelastic simulation. The concept is as follows:

1. A prescribed deformation of the structure is enforced using the first eigenmode.
2. The structure is then released from the confined displacement, allowing free movement governed by the equation of motion.
3. The displacement history is compared.

Both simulations are run on the coarsest mesh used in sec. 7.1 with a setup as described in chap. 5, using the simulation type, solver control, and domain settings in tabs. 8.1, 8.2, and 8.3, respectively.

Table 8.1: Simulation Type settings

Details of Simulation Type Basic Settings	
Time Duration	
- Option	Total Time
- Total Time	60 [s]
Time Steps	
- Option	Timesteps
- Timesteps	0.05 [s]

Table 8.2: Solver Control settings

Details of Solver Control Basic Settings	
Convergence Control	
- Max. Coeff. Loops	2

Table 8.3: Domain settings

Details of Domain 1 General Options	
Mesh Deformation	
- Option	Regions of Motion Specified
Mesh Motion Model	
- Option	Displacement Diffusion
Mesh Stiffness	
- Option	Increase near Boundaries
- Model Exponent	10

The following two sections will describe the two models, after which they will be compared.

8.1 Stationary Simulation

In the stationary simulation, there are no displacements of the geometry in CFX, the load history is merely stored. The load history from the CFX simulation is then used in conjunction with the Newmark-algorithm in MatLab to produce a displacement history. For initial conditions, the Newmark algorithm is given the same displacements as in the aeroelastic simulation, see sec. 8.2.

In fig. 8.1 the resulting displacements of the structure are plotted, given as the time series of the first two modal coordinates, q_1 and q_2 respectively.

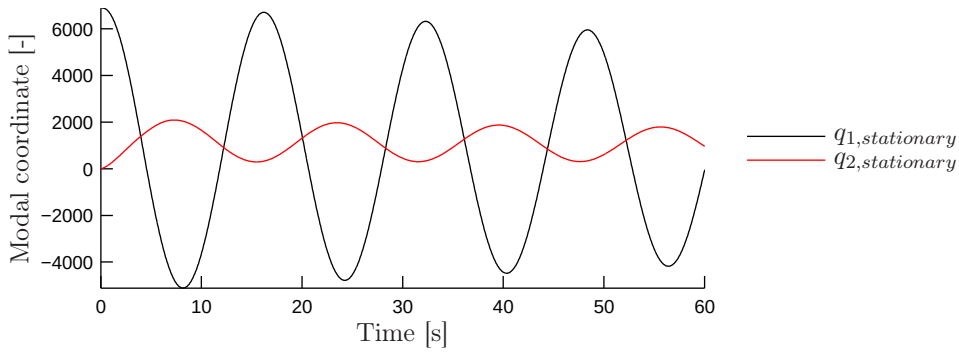


Figure 8.1: Time series of the modal coordinates q_1 and q_2 in the stationary analysis.

It can be seen from fig. 8.1 that there is a slight damping of the motion over time for both q_1 and q_2. The MatLab files used to generate the response of the structure can be found in [*DVD:\Aeroelastic_ Analyses\Effect_ of_ Aeroelasticity*]

8.2 Aeroelastic Simulation

In the aeroelastic simulation, the structure is moved to a prescribed displacement, so that the top node is deflected 2.5 m in the first eigenmode. The simulation runs as follows:

1. The structure is moved to a prescribed displacement, so that the top node is deflected 2.5 m. This displacement is done over 5 seconds.

2. Once the structure is displaced, it is held stationary in that position for 1 second, allowing the flow to somewhat adjust.

3. Then, the structure is released from the confined displacement, letting the equation of motion govern the displacement of the structure, and letting the flow forces effect this displacement.

The toggle between prescribed displacement and utilizing the Newmark-algorithm is done as follows

$$\mathbf{f}_{disp} = \begin{cases} \Delta q_1 \, \mathbf{\Phi}_1 & \text{for} \quad 0 \leq t < t_1 \\ q_{1,max} \mathbf{\Phi}_1 & \text{for} \quad t_1 \leq t < (t_1 + t_2) \\ \text{Newmark} & \text{for} \quad (t_1 + t_2) \leq t \end{cases} \tag{8.1}$$

where

t_1	is the prescribed time for the displacement, [s]
t_2	is the prescribed time for the flow to adjust, [s]
q_1	is the first modal coordinate, [-]
$q_{1,max}$	is the maximum of q_1, [-]
Δq_1	is the modal step size given as $\frac{t}{t_1} q_{1,max}$, [-]
Δt	is the time step size, [s]
$\mathbf{\Phi}_1$	is the first eigenmode, [m]

This toggle between prescribed displacement and the Newmark-algorithm is carried out in the Fortran routine `usr_spec_jcb.F`, which is found on the DVD under [*DVD:\Aeroelastic_ Analyses\Effect_ of_ Aeroelasticity*].

In fig. 8.2 the resulting displacements of the structure are plotted, given as the time series of the first two modal coordinates.

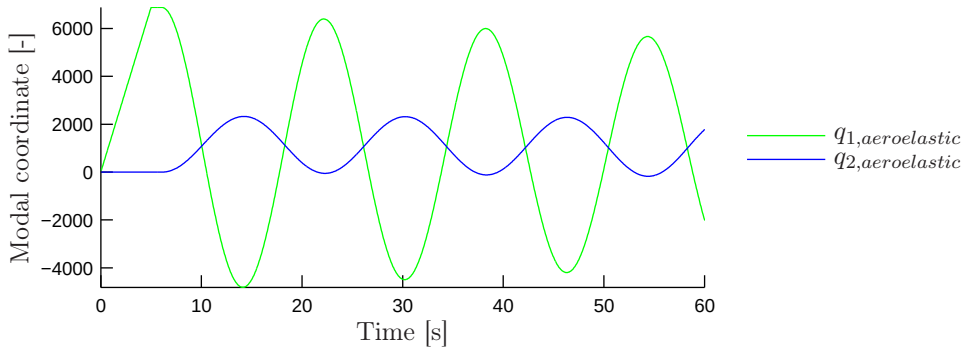


Figure 8.2: Time series of the modal coordinates q_1 and q_2 in the aeroelastic analysis.

It can be seen from fig. 8.2 that there is a slight damping of the motion over time for both q_1 and q_2, similar to that in the stationary analysis.

8.3 Comparison of Methods

To compare the two methods of modeling the modal coordinates, q_1 and q_2, from the two respective methods have been plotted together in fig. 8.3.

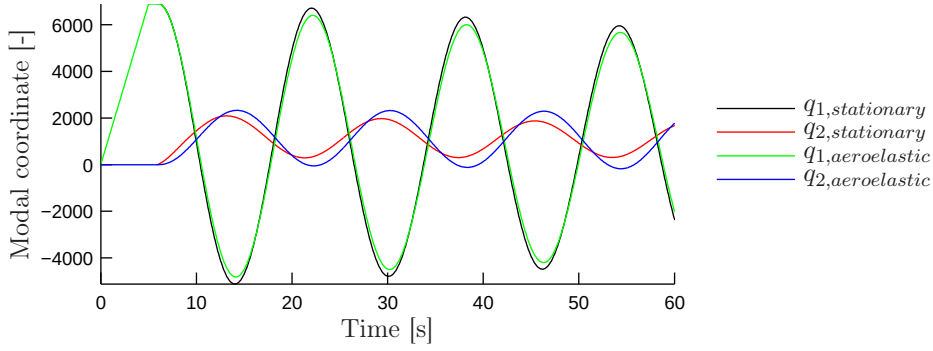


Figure 8.3: Comparison of figs. 8.1 and 8.2.

It seems difficult to say anything conclusive about the differences between the two methods from inspection of fig. 8.3, to which extent two additional simulations have been run with the same setup as described in the two previous sections. The only difference is that these simulations are set to run for 300 s, rather than 60 s. The modal loads, f_1 and f_2, for the two methods are shown in fig. 8.4, and the

response in terms of modal coordinates q_1 and q_2 are shown for the two methods in fig. 8.5.

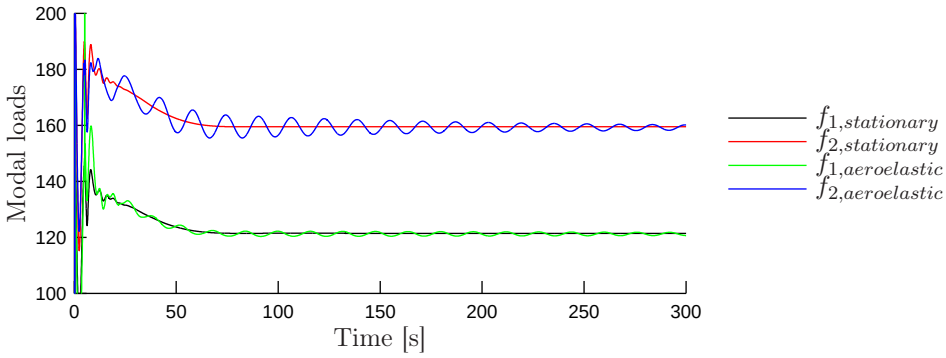


Figure 8.4: Comparison of modal loads from the two methods from a 300 s simulation.

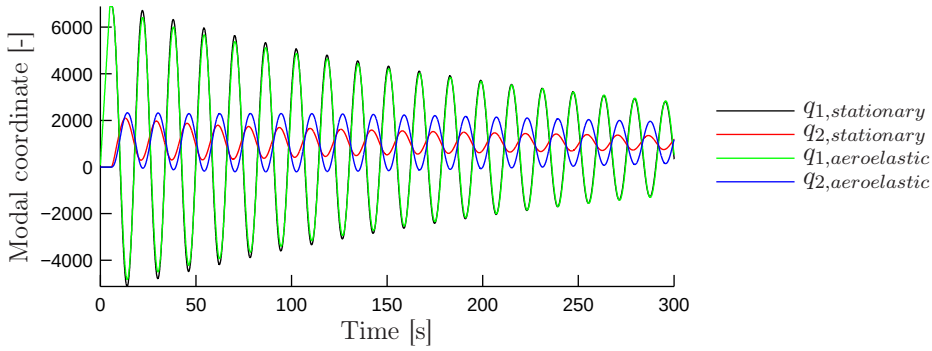


Figure 8.5: Comparison of modal coordinates from the two methods from a 300 s simulation.

In fig. 8.4 it can be seen how the load signal from the two methods differ, and it is clear to see that the loads in the aeroelastic analysis fluctuates around the loads from the stationary method. From fig. 8.5 it is seen that the time series of the first modal coordinate shows no distinctive difference between the two methods. This is a somewhat curious behavior, considering the obvious difference in the load signals. The time series of the second modal coordinate however, shows a distinct difference between the two methods. For the stationary method the amplitude of the second modal coordinate shows a clear damping behavior over

time, whereas the amplitude for the aeroelastic method seems to stay at a steady level.

It may not be obvious what causes the difference between the two methods, but it can however be concluded that there **is** a clear difference between the two methods. It was however expected to see a behavior opposite of the one shown by fig. 8.5, that is it was expected that the aeroelastic simulation would display a faster damping of the motions than the stationary, this due to the forces from the fluid working in opposite direction of the structure motion. In sec. 7.6 the necessary mesh resolution for capturing vortex shedding has been determined. Assuming a simulation was run on such a mesh, sufficiently fine to resolve the vortex shedding, this would lead to additional forces on the structure from these vortices. Depending on the frequency and phase shift, the vortex shedding could then result in an even stronger damping, or if shedding in phase with the structure, cause an increase in the amplitudes of the structure movement.

9 Modal Response

In this chapter a method to determine modal loads, f_1 and f_2, on a structure using the modal coordinates, q_1 and q_2, along with modal velocities, $\dot{q}_1$ and $\dot{q}_2$, is derived by determining the coefficients of matrices $\mathbf{A}$ *and* $\mathbf{B}$ *from a series of CFX simulations, so that the loads can be described by* $\mathbf{Aq} + \mathbf{B\dot{q}} = \mathbf{f}$.

Eight simulations are used to determine the coefficients of $\mathbf{A}$ *and* $\mathbf{B}$. *Four simulations are run where the building is excited harmonically in the first eigenmode with a constant amplitude and four different circular frequencies, and another four where the building is excited harmonically in the second eigenmode, with the same amplitude and circular frequencies as the former four simulations.*

From a test simulation it is determined that the method described above shows good correlation with the load signal directly extracted from the simulation, and on that basis it is concluded that the method can be used to give a good estimate of the modal loads through use of the modal coordinates and modal velocities.

In this chapter it is attempted to derive a method to describe the loads on the structure from a modal point of view. In what follows it is assumed, that the modal loads, $\mathbf{f}$, on the structure can be written as:

$$\mathbf{f} = \mathbf{A}\mathbf{q} + \mathbf{B}\dot{\mathbf{q}} \tag{9.1}$$

where

$\mathbf{A}, \mathbf{B}$	are modal coefficient matrices
$\mathbf{q}$	is the modal coordinate vector
$\dot{\mathbf{q}}$	is the modal velocity vector

The entries of the coefficient matrices, $\mathbf{A}$ and $\mathbf{B}$, are dependent on the number of eigenmodes retained in the analysis. In this project only the first two eigenmodes are taken into account as the project focusses more on the method than the actual results. The coefficient matrices are therefore both 2×2 matrices.

The representation in eq. (9.1) is a very simplified expression. It could be discussed whether an acceleration term, a constant term and a term dependent on the vortex shedding frequency should be included. Of course the more terms the expressions include the possibility of the expression to yield accurate results increase. But in this project only loads due to displacement and velocity are considered.

9.1 Method for Determining Coefficient Matrices

In this section the method for determining the coefficient matrices is presented. The basis for the determination are CFD simulations where the structure is forced to move in a specific eigenmode by a specified frequency and amplitude. The harmonic motion of the structure during the simulations is governed by the following:

$$q_j(t) = \tilde{q}_j \cos(\omega t + \Psi) \tag{9.2}$$

where

$\tilde{q}_j$	is the amplitude of the harmonic motion, [-]
ω	is the circular frequency of the harmonic motion, $\left[\frac{\text{rad}}{\text{s}}\right]$
t	is the time, [s]
Ψ	is the phase of the motion, [rad]

[Nielsen 2004]

The forced harmonic motion of the structure is determined by specifying a maximum value of the displacement, q_j, and inserting this into eq. (9.2) as the amplitude of the motion, $\tilde{q}_j(t)$. The specified frequency and eigenmode determines the motion of the structure on the basis of the selected maximum amplitude, $\tilde{q}_j$. It should be noted that the specified displacement is given as the maximum displacement of the top node and hereafter the eigenmode determines the motion of the rest of the structure.

The structure is initially considered at rest in a vertical position, $t = 0$, and by inspection of eq. (9.2) the phase, Ψ, must therefore be selected as $\Psi = \frac{\pi}{2}$.

The amplitude of the harmonic motion in the simulations is chosen to 2 m. The structure vibration frequency is expected to be somewhere in the region of the first eigenfrequency listed as 0.39 $\left[\frac{\text{rad}}{\text{s}}\right]$ in eq. 6.11. It is therefore chosen to run four sets of simulations, each set containing two simulations, one for each of the first two eigenmodes. Each set of simulations have their own specific frequency. The four sets of simulations are listed in tab. 9.1 where the period of motion, T, have been determined by $T = \frac{2\pi}{\omega}$.

Table 9.1: Definition of simulations

Simulation	Frequency ω, $\left[\frac{\text{rad}}{\text{s}}\right]$	Period of motion T, [s]
Sim 1	0.3	20.9
Sim 2	0.4	15.7
Sim 3	0.5	12.6
Sim 4	0.6	10.5

The harmonic motion of the structure top governed by the frequencies listed in tab. 9.1 are shown in fig.

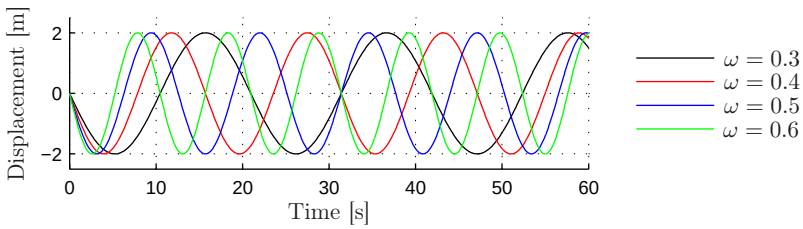


Figure 9.1: Harmonic motion of structure top. Frequencies in $\frac{rad}{s}$

The harmonic motion given in eq. (9.2) also works for the modal coordinate, q. Utilizing Euler's formula, $e^{it} = \cos(t) + i\sin(t)$, the harmonic motion can be written in the following complex notation where $\tilde{q}_j \cos(\omega t + \Psi)$ corresponds to the real part of the complex notation:

$$\mathbf{q} = \tilde{\mathbf{q}} e^{i(\omega t + \Psi)} \tag{9.3}$$

In the following it is assumed that the load on the structure is also harmonic with the same frequency as the structure movement but with some phase shift, Ψ. The load can therefore be written as:

$$\mathbf{f} = \tilde{\mathbf{f}} e^{i(\omega t + \Psi)} = \tilde{\mathbf{f}} e^{i\omega t} e^{i\Psi} \tag{9.4}$$

As the method is based on a modal representation the modal loads must be used. The modal loads are determined by:

$$\mathbf{f}_{\text{modal}} = \mathbf{P}\,\mathbf{f} \tag{9.5}$$

where

$\mathbf{P}$	is the modal matrix, $\mathbf{P} = \begin{bmatrix} \Phi^{(1)} & \cdots & \Phi^{(n)} \end{bmatrix}$
$\Phi^{(j)}$	is the j^{th} eigenvector
$\mathbf{f}$	is the nodal forces and moments
n	is the number of eigenmodes taken into account

The harmonic representation of the load in eq. (9.4) works just as well for the modal load. Inserting eqs. (9.3) and (9.4) in (9.1) gives:

$$\begin{aligned} &\mathbf{A}\,\tilde{\mathbf{q}}\, e^{i\omega t} + i\omega \mathbf{B}\,\tilde{\mathbf{q}}\, e^{i\omega t} = \tilde{\mathbf{f}}_{\text{modal}}\, e^{i\omega t}\, e^{i\Psi} \\ &\mathbf{A}\,\tilde{\mathbf{q}} + i\omega\,\mathbf{B}\,\tilde{\mathbf{q}} = \tilde{\mathbf{f}}_{\text{modal}}\, e^{i\Psi} \end{aligned} \tag{9.6}$$

Eq. (9.6) can be written in the following way using index notation:

$$A_{jk}\,\tilde{q}_k + i\omega\, B_{jk}\,\tilde{q}_k = \tilde{f}_j\, e^{i\Psi} \tag{9.7}$$

where

$\tilde{f}_j$	is the load components of the load vector $\mathbf{f}_{\text{modal}}$

From eq. (9.7) it is seen, that the term containing A_{jk} corresponds to the real part of the right hand side while the term with B_{jk} corresponds to the imaginary

part. This fact is utilized to determine the coefficients in **A** and **B**. This yields the following two equations to determine the coefficients:

$$\begin{aligned} A_{jk} &= \mathrm{Re}\left(\frac{\tilde{f}_j\, \mathrm{e}^{i\,\Psi}}{\tilde{q}_k}\right) \\ B_{jk} &= \mathrm{Im}\left(\frac{\tilde{f}_j\, \mathrm{e}^{i\,\Psi}}{\tilde{q}_k\, \omega}\right) \end{aligned} \tag{9.8}$$

The procedure for determining the coefficients is summarized below.

1. The load vector, **f**, determined from the CFD simulations is loaded into a MatLab program
2. The modal load, $\mathbf{f}_{\mathrm{modal}}$, is determined by eq. (9.5)
3. By use of the specified frequency, ω, the motion amplitude, $\tilde{q_j}$, and the time step size, Δt, a time series of the modal coordinate, q_j, is generated by use of eq. (9.2)
4. The modal load and modal coordinate are plotted and the phase, Ψ, and load amplitude, $\tilde{f}$, is determined
5. The phase and amplitude is inserted into eq. (9.8) and the coefficients in **A** and **B** are determined. Which simulation relates to the individual coefficients in **A** and **B** can be seen in fig. 9.2
6. The procedure is repeated for simulations with different frequency

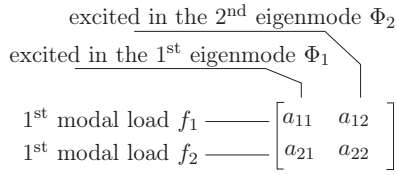


Figure 9.2: Relation between coefficients and simulations

9.2 Simulation Setup and Observations

The setup of the simulations in CFX are basically the same is listed in sec. 5.1. The settings for *mesh deformation* was initially set to *Increase near Boundaries* with a model exponent of the default value 10. This resulted in simulations which relatively quickly failed due to the creation of a negative volume element.

The *Increase near Boundaries* model uses the following relationship for the mesh stiffness, cf. *ANSYS CFX-Solver Modeling Guide, p. 5*:

$$\Gamma_{\text{disp}} = \left(\frac{1}{d}\right)^{C_{\text{stiff}}} \tag{9.9}$$

where

d	is the distance from the nearest boundary
C_{stiff}	is the model exponent

Eq.(9.9) yields an exponential increase in the mesh stiffness as the distance to the nearest boundary, d, decreases. The model exponent, C_{stiff}, determines how quickly this increase occurs. The model exponent was hereafter increased to 100 to get the stiffness to increase more rapidly. The simulations did hereafter finish without error. It should be noted that increasing the stiffness parameter does not move the area where deformation of the mesh must be absorbed that much. If the simulations keep crashing after increasing the stiffness parameter another mesh deformation model must be used. Either the user can try one of the other built in models in CFX or make one in Fortran and incorporate this into CFX. The effect of both methods have been analyzed in chap. 4.

The mesh used for the simulations is the one called *Mesh 1* in subsec. 7.1.1. This mesh does not capture all flow related phenomena according to the analyses in chap. 7. Since the method presented in this chapter is a simplified model intended to describe the load variation from the structure displacements only, the need for including e.g. vortex shedding in the simulations is unnecessary as this will have no impact on the coefficients in **A** and **B**.

In the analyses in chap. 7 it was found, that the simulations had almost stabilized after 60 s of simulated time. To be sure to get a usable harmonic signal the simulated time is doubled to 120 s. As mentioned earlier a total of 8 simulations have been run in four sets of two simulations. The simulation names are listed below and can be found on the attached DVD in [*DVD:\Aeroelastic_Analysis\Modal_Response\CFX_Files*]:

- $\omega = 0.3\,\frac{\text{rad}}{\text{s}}$: $\begin{cases} \textit{Sim\_omega\_0\_3\_V\_20\_Phi1\_120.cfx} \\ \textit{Sim\_omega\_0\_3\_V\_20\_Phi2\_120.cfx} \end{cases}$
- $\omega = 0.4\,\frac{\text{rad}}{\text{s}}$: $\begin{cases} \textit{Sim\_omega\_0\_4\_V\_20\_Phi1\_120.cfx} \\ \textit{Sim\_omega\_0\_4\_V\_20\_Phi2\_120.cfx} \end{cases}$

- $\omega = 0.5\,\frac{\text{rad}}{\text{s}}$: $\begin{cases} Sim\_omega\_0\_5\_V\_20\_Phi1\_120.cfx \\ Sim\_omega\_0\_5\_V\_20\_Phi2\_120.cfx \end{cases}$
- $\omega = 0.6\,\frac{\text{rad}}{\text{s}}$: $\begin{cases} Sim\_omega\_0\_6\_V\_20\_Phi1\_120.cfx \\ Sim\_omega\_0\_6\_V\_20\_Phi2\_120.cfx \end{cases}$

9.3 Results

In this section the results of the analyses are presented. The first part deals with the determination and analysis of the coefficients in **A** and **B**. Hereafter the model is tested on a simulation where the structure is allowed to move freely during the simulation.

9.3.1 Determination of Coefficients

The determination of the coefficient matrices is done on the basis of the harmonic motion specified for the structure movement and the resulting load history from the simulation. The main part deals with treatment of the load history to determine the phase shift and load amplitude of the load variation which is assumed harmonically varying with the same frequency as the structure movement with some phase shift.

The harmonic part of the load history does not begin at once. The effects of the initial conditions must be dissipated away before the data treatment starts. This is done by specifying a specific place in the load history from where the data treatment starts. The time series of the modal loads from all eight simulations are shown in fig. 9.3. The specified cut-off limit is also shown for each simulation.

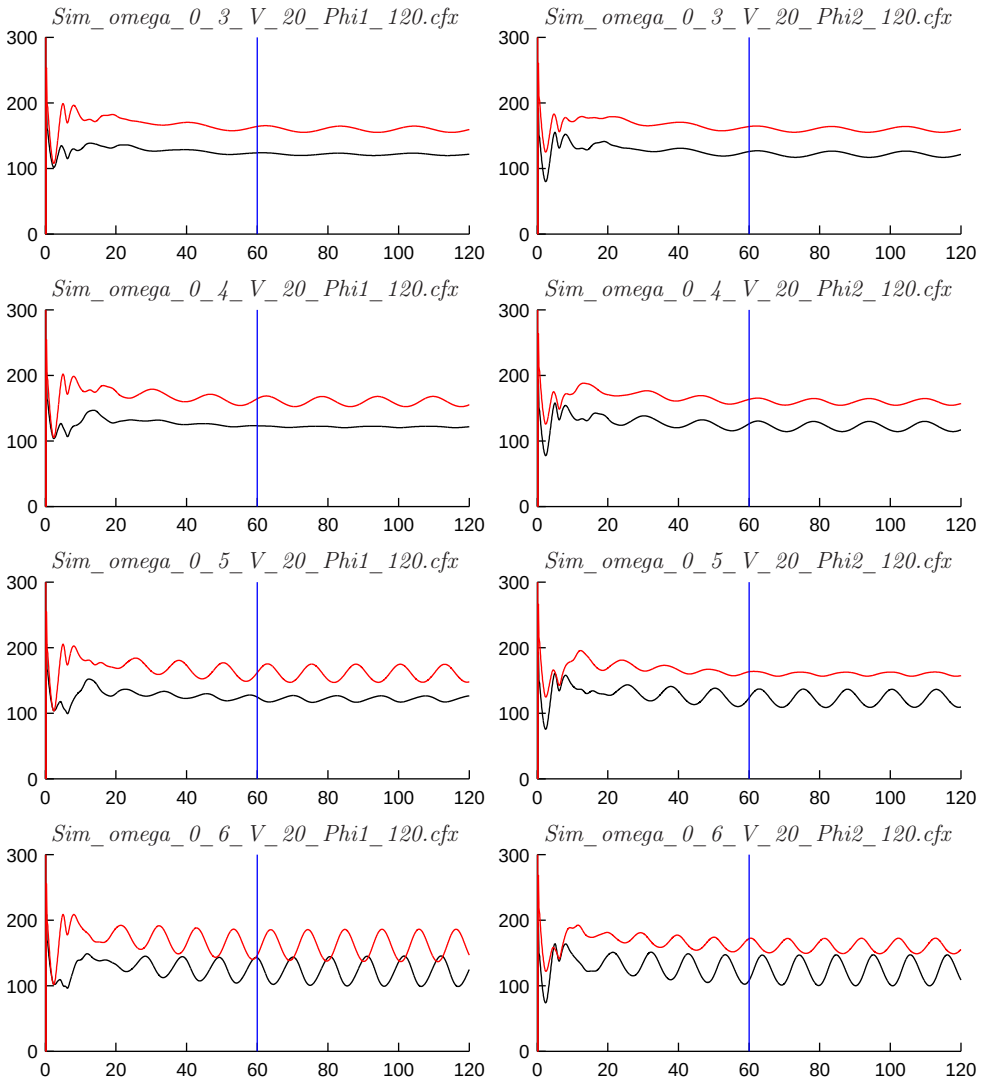


Figure 9.3: Modal loads for all simulations. — is load vector multiplied by Φ_1, — is load vector multiplied by Φ_2 and — specifies the cut-off limit. x-axis is time and y-axis is modal load

To start with the top points on the two curves are localized to the right of the cut-off limit and the time to which these maxima occur is stored. Each top point on the displacement curve is connected with the top point on the load curve to the left of it. Each phase between the connected points is determined and the final phase, Ψ, is determined as a mean value of these.

The load amplitude is determined by localizing the minima and maxima on the load curve. The amplitude, $\tilde{f}$, is then calculated as the difference between the mean of the maxima and the mean of the minima, divided by two.

In fig. 9.6 the 16 fits are shown, corresponding to two for each of the 8 simulations run. The subfigures are named e.g. '$\omega = 0.3$, Φ_1, f_2', meaning that it is the simulation where the structure has been subjected to a harmonic motion in the first eigenmode at a circular frequency of 0.3, and that it is a plot of the 2nd modal load.

The coefficients of **A** and **B** are plotted in figs. 9.4 and 9.5 respectively.

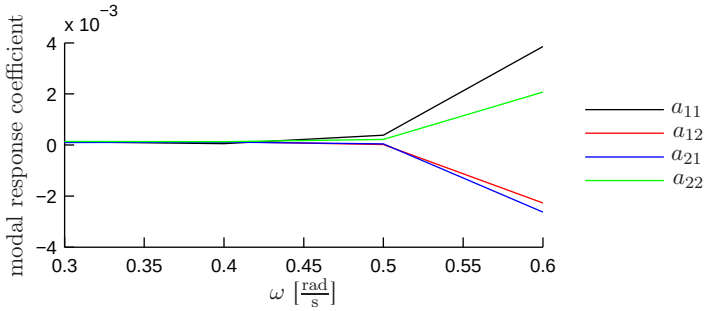


Figure 9.4: Coefficients of **A** *as a function of the circular frequency* ω

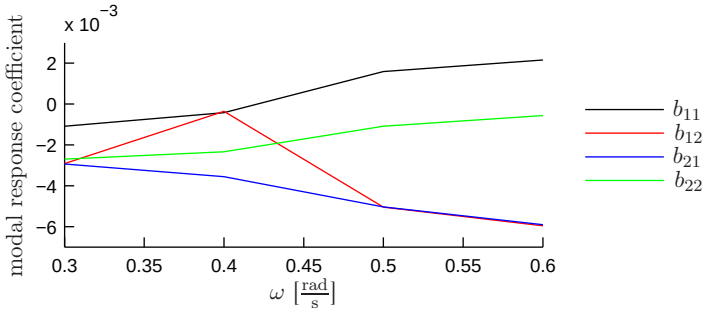


Figure 9.5: Coefficients of **B** *as a function of the circular frequency* ω

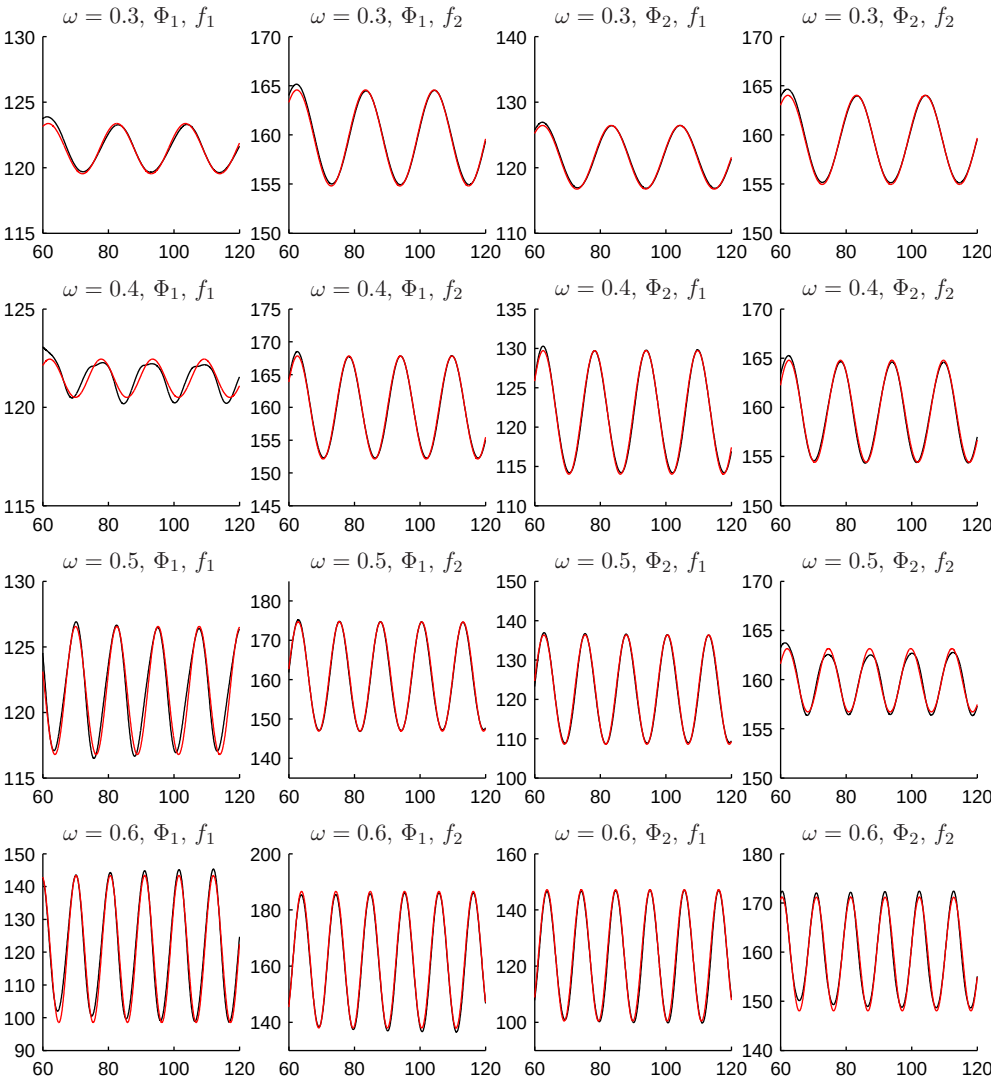


Figure 9.6: Modal loads for all simulations, compared with those generated by use of eq. (9.1). — *are the modal load vectors from the simulations, and* — *are the modal load vectors generated by use of eq.* (9.1). *x-axis is time and y-axis is modal load*

As can be seen in fig 9.6, the harmonic signals produced match the load signals from the simulations quite well. The few notable deviations that are seen are all due to the load signal from the simulations not quite being harmonic yet. It should here be noted that the output load signal from the MatLab program fluctuates around zero, as the mean value is not considered. So, in fig. 9.6 the

mean value of the incoming load signal (after the cut-off limit) have been added to these fluctuations. The MatLab program, *matrix_coeff.m*, can be found on the DVD in: [*DVD:\Aeroelastic_Analyses\Modal_Response*].

The coefficients obtained this way, as given in figs. 9.4 and 9.5, are only valid for this specific setup. If a more general approach had to be taken, the dependency on the circular frequency ω should be replaced by the *reduced frequency* k, which is given as $k = \frac{\omega D}{u}$, where D is the side length of the structure and u is the wind velocity. Furthermore the entries in $\mathbf{A}$ and $\mathbf{B}$ would need to be normalized which is not directly possible in the modal presentation, because the eigenmodes consists of both translation and rotation. To make the coefficient matrices dimensionless, this would have to be done while the equations are still on cartesian form, $\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\dot{\mathbf{x}}$, where the entries containing translation/forces would be divided by $\left(\frac{1}{2}\,\rho\,u^2\,D\right)$ and the entries containing rotation/moments would be divided by $\left(\frac{1}{2}\,\rho\,u^2\,D^2\right)$

9.3.2 Test of Model

To test the model with the coefficients determined in subsec. 9.3.1 a simulation have been run, where the structure is allowed to move freely in the first two eigenmodes corresponding to the ones used to set up the model. The time series of the load is extracted as usual, but this time also the time series of the modal coordinates, q_1 and q_2, and modal velocities, $\dot{q}_1$ and $\dot{q}_2$, are extracted. The modal coordinates are used to determine frequencies for each modal coordinate respectively. These frequencies are used to find values in $\mathbf{A}$ and $\mathbf{B}$. This is done by interpolation between the determined coefficients for the four known frequency values used to set up the model. The coefficient matrices determined by this interpolation is inserted into eq. (9.1) and a loop in MatLab is run over the number of time steps in the simulation each time updating eq. (9.1) with the matching modal coordinates and velocities. In this way a time history of the modal loads is determined.

The modal coordinates from the simulation are shown in fig. 9.7 where it is clear that the modal coordinates have the same phase and circular frequency, but have different amplitudes.

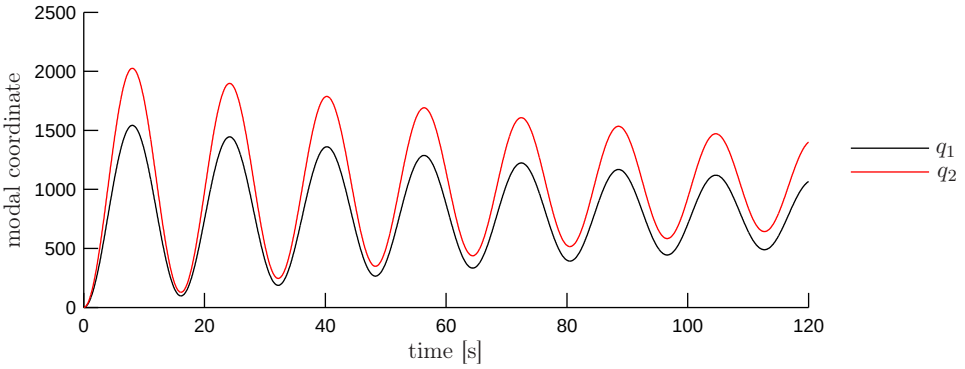


Figure 9.7: Modal coordinates extracted from the test simulation.

To conduct the interpolation a linear variation of the coefficients between the frequencies are assumed. Instead of using a mean value of the frequency over the entire time series for the interpolation the time series are divided into sections, each section as the distance between two adjacent top points on the curve for the relevant modal coordinate. The time, T, covered by one section is used to determine the corresponding frequency by:

$$\omega_i = \frac{2\,\pi}{T} \tag{9.10}$$

whereafter the reduced frequency is determined by eq. (**??**). The frequency determined for the respective section is assigned to the midpoint of this section and a linear variation is assumed between the frequency values. For each step in the loop over the number of time steps a frequency for this time step is found using this linear variation. The method is illustrated in fig. 9.8.

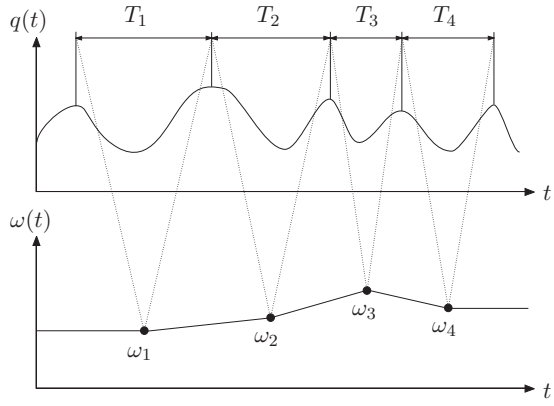


Figure 9.8: Illustration of determination of ω

The frequency to each time step is hereafter used to interpolate in **A** and **B** to determine the coefficients for the relevant frequency. As mentioned earlier, eq. (9.1) is hereafter used to generate a load signal. By the method presented the load signal generated are shown in figs. 9.9 and 9.10 against the fluctuations of the load signals extracted from a simulation in CFX where the structure is allowed to move freely in the two first eigenmodes. In fig. 9.11 the full load signals extracted from the CFX simulation is shown.

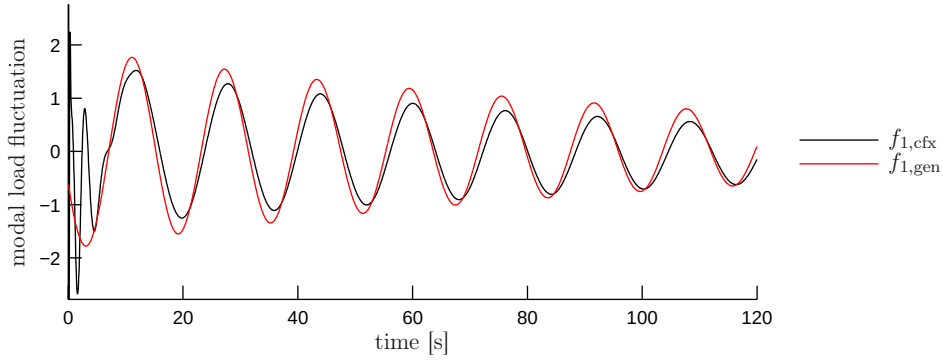


Figure 9.9: Comparison of the load fluctuations in f_1 from the generated and extracted loads.

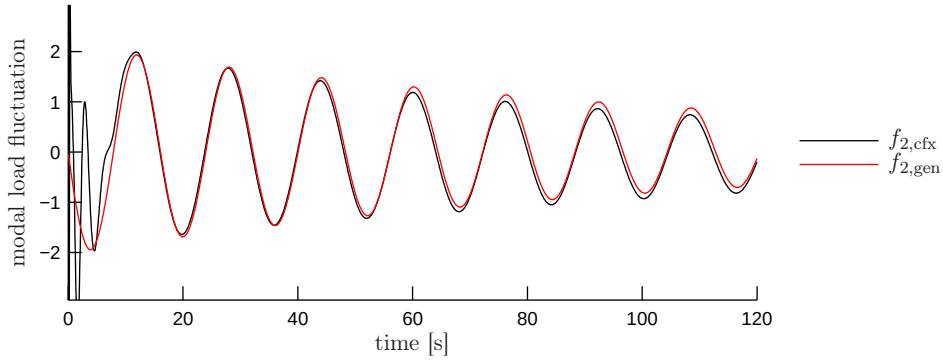


Figure 9.10: Comparison of the load fluctuations in f_2 from the generated and extracted loads.

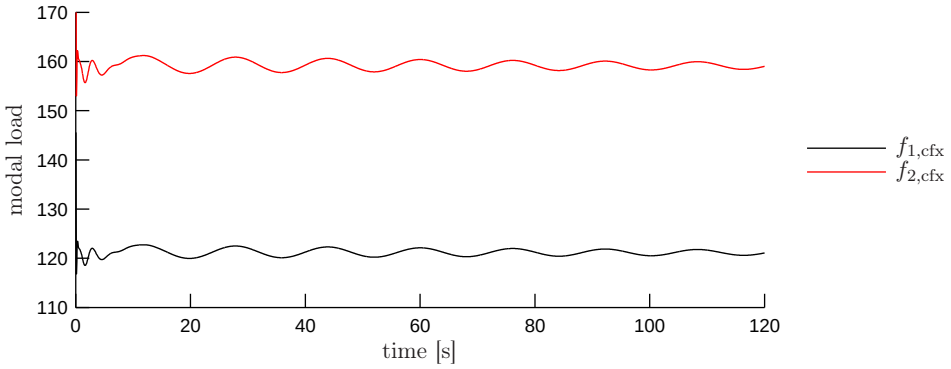


Figure 9.11: Extracted modal loads $f_{1,cfx}$ and $f_{2,cfx}$.

By inspection of figs. 9.9 and 9.10 it can be seen that the fluctuations of loads generated through use of eq. (9.1) fit quite nicely, albeit with a slight deviation of amplitudes in f_1. As mentioned earlier, the mean value is not considered in this analysis, as it is more dependent on the flow, than the deformation of the structure. If this was to be included, additional simulations would have to be run, as a minimum including a variation of the free stream velocity, u_∞. As this is not done here, the behavior of fluctuations is the main focus, and it can be concluded that the method of eq. (9.1) can in fact produce reasonable modal loads, corresponding to those measured in a CFX simulation.

9.4 Comparison of Results

The use of modal response coefficients, or flutter derivatives, is a well known method within dynamic bridge design. In this project it is attempted to verify if such a method could be used to model the aeroelastic response of a high-rise building, and yield qualitative good results. It was determined that the method, where the flutter derivatives are determined as a function of the reduced frequency, K, yielded results in terms of the modal load vector, which were well correlated with results directly extracted from a CFD simulation.

A similar conclusion has been reached in [Stærdahl, Sørensen & Nielsen 2007], where flutter derivatives have been used to determine the aeroelastic stability (flutter stability) of the Great Belt Bridge of Denmark. [Stærdahl et al. 2007] determines that the mesh does not need to be fine enough to resolve the vortex shedding, which works in favor of the delimitations made in the present project, as long as the deformation frequency can be observed in the load signal. Finally, it is concluded in [Stærdahl et al. 2007] that using CFD to compute the flut-

ter derivatives can yield qualitative and quantitatively good results, compared to the conventional approach of determining the flutter derivatives from wind tunnel tests. In this project there are no wind tunnel tests to compare against, but it has similarly been concluded that the method can produce good results when using the modal coordinates and flutter derivatives to determine the modal loads.

10 Conclusion

In this final chapter the main results and observations are summed up for the different analyses and methods. Due to the simulations conducted being run with coarse meshes and settings some small methods of improvement for the analyses in this project are presented.

In the last part of this chapter different areas to which attention could be turned for the analyses and methods in this project to be improved are presented. To each area different proposals are made on how the specific area can be improved compared to the present project. The last part should be considered as inspiration for future work within this field.

The purpose of this project was to model and analyze aeroelasticity in connection with high-rise buildings. The analyses have been conducted by use of CFD simulations of the wind flow around a square structure.

One of the main flow phenomena which has an impact on the wind induced forces on the structure is vortex shedding. In the process of conducting the analyses it became apparent, that to capture vortex shedding during the simulations the mesh has to be sufficiently fine and the time step size so low that the computation time on one CPU would be extremely long. Instead the focus of the project was turned towards the method of simulating aeroelastic response by use of CFD and then accept, that the results obtained might not be physically correct.

The first part of the project deals with generation of a suitable mesh for the simulations. A mesh generation algorithm was programmed in MatLab based on hyperbolic partial differential equations. By use of this algorithm structured meshes for the exterior flow can be generated. The advantage of using a structured mesh was analyzed by comparing simulations with these to simulations run with fully unstructured meshes (chap. 3). It was clear, that the main advantage is in the computation time needed. The time used for the simulations with the structured mesh was much lower than for the simulations with unstructured meshes (see. fig. 3.21). Therefore it is recommended to use a structured mesh if the generation of this does not take considerable longer than an unstructured mesh. This difference is evened out as the amount of simulations to be conducted increase.

As the simulations used to model the aeroelastic response was conducted with a coarse mesh and a relatively high time step size, analyses was carried out to find out, if one parameter in the setup has a larger impact than others. Due to the long computation time needed the analyses in chap. 7 only showed, that all parameters analyzed have some influence on the result but not which one had the largest impact. The influence on the results was most visible in the cross-wind direction whereas the loads in the streamwise direction showed only little dependency on the different parameters. In order to find out the magnitude of influence of the different parameters a reference simulation should have been used and then simulations where only one parameters at the time have been altered should have been run. The latter analysis is left out due to the limited computer power and the needed computation time needed for the analysis.

In chap. 8 an analysis of the effect of modeling the aeroelastic response is conducted. The analysis did however not show the expected result. It was expected,

that the presence of aeroelastic damping would reduce the structure movement faster than for the simulation with the stationary structure. Instead, the simulations showed some presence of damping on the second modal coordinate for the simulation with a stationary structure whereas the simulation with a moving structure did not show the expected result. The displacement of the structure including aeroelasticity maintains approximately the same magnitude during the entire run. This could indicate some negative aeroelastic damping acting on the structure causing the displacements to maintain their amplitude and not dampen away over time.

In chap. 9 a method is derived for determining the load on a structure based on the movement of this. The method is based on a modal representation of the structure. Different simulations with enforced harmonic motions in known frequencies have been conducted and based on these a model for determining the load response have been set up. The model is then tested on a simulation with the structure allowed to move freely and the load response from this simulation is compared to the one generated by the model on the basis of the structure movement.
It is found that it is possible to model the load response quite well. The disadvantage of the present model is, that the mean of the load is not contained in the model and only the load fluctuations around the mean value are determined by the model. In order to incorporate the mean of the load into the model a number of simulations have to be run with different settings of the wind velocity to determine the mean load dependency on this parameter. If the model is to be made generally applicable, also the structure size and shape have to be incorporated into the mean load determination.

As mentioned the simulations for the analyses were conducted using coarse settings and mesh and hereby not all flow phenomena were captured. In the following section suggestions are made to where and how improvements to the conducted analyses can be made. This should be thought of as inspiration for any future work to be done within this field.

10.1 Suggestions for Improvements

In this chapter some areas as to where the analyses in this project can be improved will be presented. In this project the major limitation was the relatively limited computer power available. This condition was the main reason for the simplifications of the simulations. Therefore the main emphasis of this chapter will be used to present areas where the simulations from this project should be

improved assuming that the necessary computer power was available.

10.1.1 CFD Simulation Setup

The first area where the simulations should be improved is the use of a suitable mesh with a resolution fine enough to resolve all relevant flow phenomena, in this case especially vortex shedding. In this project some preliminary analyses have been conducted (chap. 7) where it has been determined that the use of coarse settings for mesh fineness, time step etc. has a significant impact in the results. Instead preliminary analyses can be used to determine the necessary mesh fineness, time step size and other relevant settings for the simulations to perform the best possible way.

If the results are to be used in some comparison with results from other methods the emphasis must be put into this area as all the results depend on simulations capable of representing nature in the best possible way. The CFD settings does not necessarily need to be optimized as long as the settings are finer than actually needed. This might save some time in the short term if only a small number of simulations need to be run. If several simulations are to be run it is advised that the user takes the time to conduct relevant analyses to optimize the simulation setup and hereby save computation time. It takes longer in the short run but it is the experience that the extra time used is better in the long run.

10.1.2 Load Determination

Due to program deficiencies in Ansys CFX 11 the total loads presented and used in this project only contain contributions from the pressure on the structure. This is due to the method used to extract the loads during a transient simulation. This method does not work with the wall shear in Ansys CFX 11. Therefore the presented loads are not to be considered as representative of nature. The deficiency in CFX 11 should be fixed in the coming version 12 and should hereafter of course be included in the total loads.

In this project no analysis have been conducted to check whether the use of 50 sections (5 on each beam) for determining the loads is sufficient, or whether the number of sections can be decreased or needs to be increased. However, the emphasis of this project is turned towards the method of using modal analysis in modeling fluid-structure interaction rather than obtaining accurate simulation results.

If the method in the future is to be used to get reliable results an analysis of

the necessary sections for load determination could be an area of interest. The loads are in this project determined after each time step in the transient runs but the experience is, that the time consumption of this area in the simulations is negligible compared to the fluid solving part. The analysis of needed sections might therefore be left out and the number of sections set to a sufficiently high number without increasing the total computation time much.

10.1.3 Mesh Generation

In this project, the 3D mesh has been generated from the basis of a 2D mesh. The 2D mesh is generated from an algorithm based on hyperbolic PDE's. As long as the algorithm is used purely for 2D meshing, it has shown to be quite good. However, when extruding to a 3D mesh, the mesh program has some very prominent deficiencies with regards to closing the mesh above the structure, leading to the very limiting constraint, that the mesh has to be created from four boundaries, and that the number of nodes on these boundaries must be equal for the pairwise opposing sides.

In the interest of performing analyses similar to the analyses performed in this project, but on a more complex geometry, the need for a different mesh algorithm is irrefutable. Such algorithms exist, e.g. by means of three dimensional hyperbolic PDE's. Using such a method to generate the 3D grid would give the user far more flexibility in choosing a 3D geometry which more adequately represents a real structure.

Another aspect of the mesh generation which could be improved, for a more user friendly approach, is the use of adaptive meshing. This could be utilized when specifying e.g. the dissipation, so that the dissipation would be determined from local conditions in the mesh, rather than a set dissipation for each marching layer, as has been done in this project. The problem with a constant dissipation factor for one marching layer is that the optimal factor varies depending on whether the mesh is near a corner, or near a concave or convex boundary. The adaptive dissipation would thus ensure that an "optimal" (relative to a constant factor for each marching layer) dissipation factor is used for every cell.

10.1.4 Inlet Conditions

The inlet conditions used in the project are constant both in time and in the global z-coordinate. For a more precise representation of nature, it could be argued that the inlet conditions should model the atmospheric boundary layer,

corresponding to a no-slip condition imposed on the ground surface boundary.

Furthermore it is somewhat naive to model an inlet where the wind direction and velocities do not vary even a fraction over time. To this end, some random fluctuations of the inlet velocities could be added to the inlet conditions, controlled by a few user specified parameters.

Bibliography

CEN/TC250, T. C. [2005]. *Eurocode 1, Actions on structures - Part 1-4: General actions - Wind actions*, BSI.

Chan, W. M. & Steger, J. L. [1992]. Enhancements of a three-dimensional hyperbolic grid generation scheme, *Applied Mathematics and Computation* **Vol. 51**: pp. 181–205.

Cook, R. D., Markus, D. S., Plesha, M. E. & Witt, R. J. [2002]. *Concepts and Applications of Finite Element Analysis*, 4th edn, John Wiley and Sons Inc.

Dennis, S. & Chang, G.-Z. [1970]. Numerical solutions for steady flow past a circular cylinder at reynolds numbers up to 100, *Journal of Fluid Mechanics* **Vol. 42, part 3**: pp. 471–489.

Ferziger, J. H. & Peric, M. [2002]. *Computational Methods for Fluid Dynamic*, 3. edition edn, Springer-Verlag.

Nielsen, S. R. K. [2004]. *Vibration Theory - Linear Vibration Theory*, Vol. 1, Aalborg tekniske Universitetsforlag.

Nielsen, S. R. K. [2005]. *Structural Dynamics - Computational Dynamics*, Vol. 9, Aalborg tekniske Universitetsforlag.

Steger, J. L. & Chaussee, D. S. [1980]. Generation of body-fitted coordinates using hyperbolic partial differential equations, *SIAM Journal SCI. STAT. COMPUT.* **Vol. 1**(No. 4): pp. 431–437.

Stærdahl, J. W., Sørensen, N. & Nielsen, S. R. K. [2007]. Aeroelastic stability of suspension bridges using cfd.

Thompson, J. F., Warsi, Z. & Mastin, C. W. [1985]. *Numerical Grid Generation*, Elsevier Science Publishing Co., Inc.

Wilcox, D. C. [2002]. *Turbulence Modeling for CFD*, 2nd edn, D C W Industries.

Appendix

Contents

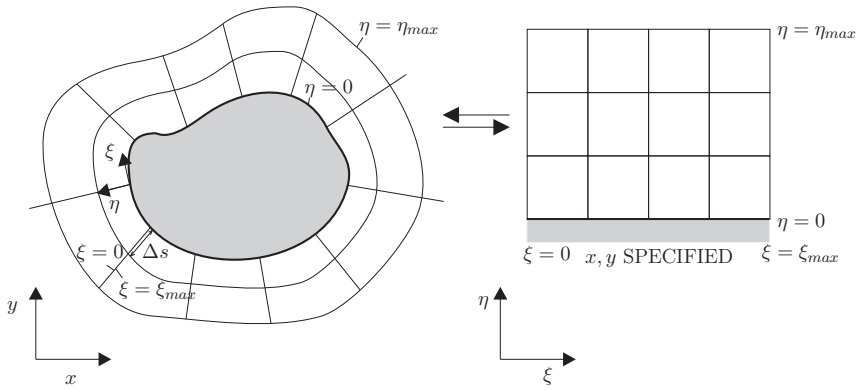
A

Hyperbolic Mesh Generation, Theory

When conducting CFD analyses an important feature is the mesh used. Many different methods for mesh generation exist. In this project a hyperbolic mesh generation method have been used. In this appendix, the theory behind the algorithm will be presented. Unless anything else is mentioned the following is based on [Steger & Chaussee 1980].

A.1 Obtaining the PDE

First, the generation of an exterior mesh around an arbitrary closed boundary like the one in fig. A.1 is considered. On fig. A.1a the physical plane is shown, in coordinates x, y - on fig A.1b the computational plane is shown in coordinates ξ, η. It can be seen how ξ, η is denoted in the physical plane, as radial lines and encircling lines respectively.
The inner boundary is prescribed by a set of coordinates, whereas the outer boundary is not; it only needs to be sufficiently far from the inner boundary.



a) Physical plane x, y. *b*) Computational plane ξ, η.

Figure A.1: Illustration of grid mapping procedure.

The PDE's should produce smoothly distributed grid lines so that lines of the same family (η and ξ respectively) do not cross or coalesce. If this is fulfilled, a one-to-one mapping will exist between the grid in the computational plane and that in the physical plane.

Consider an arc length scheme, imposed by a constraint of orthogonality, given by:

$$x_\xi x_\eta + y_\xi y_\eta = 0 \tag{A.1}$$

where lower index indicates differentiation, e.g. $x_\xi = \frac{\partial x}{\partial \xi}$.

The distance, ds, between levels of η = constant lines is constrained by:

$$\begin{aligned}(ds)^2 &= (dx)^2 + (dy)^2 \\ &= (x_\xi d\xi + x_\eta d\eta)^2 + (y_\xi d\xi + y_\eta d\eta)^2\end{aligned} \tag{A.2}$$

A uniform grid is specified in the computational plane, see fig. A.1b. For convenience, $d\eta/d\xi = 1$ is chosen. Expanding eq. (A.2) and using eq. (A.1) gives:

$$\begin{aligned}ds^2 &= x_\xi^2 d\xi^2 + x_\eta^2 d\eta^2 + y_\xi^2 d\xi^2 + y_\eta^2 d\eta^2 + 2(x_\xi d\xi x_\eta d\eta + y_\xi d\xi y_\eta d\eta) \\ (\Delta s)^2 &\equiv \left(\frac{ds}{d\eta}\right)^2 = x_\xi^2 + y_\xi^2 + x_\eta^2 + y_\eta^2\end{aligned} \tag{A.3}$$

Eqs. (A.1) and (A.3) forms a system of nonlinear PDE's with initial data in

$\eta = 0$. Local linearization of this system of equations followed by analysis shows that the equations are hyperbolic. This mean the equations can be marched in η from initial data along the body, $\eta = 0$, as shown in fig. A.1a.

Let $\tilde{x} = x - x^0$ and $\tilde{y} = y - y^0$, where x^0 and y^0 denote a known near solution state, i.e. the coordinates of the previous η line. Term by term linearization of eqs. (A.1) and (A.3) yields expressions on the following form, where $\tilde{x}$ is assumed small compared to x and x_0:

$$\begin{aligned} x_\xi x_\eta &= (x^0 + \tilde{x})_\xi (x^0 + \tilde{x})_\eta \\ &= (x^0_\xi x^0_\eta + x^0_\xi \tilde{x}_\eta + x^0_\eta \tilde{x}_\xi) + O(\tilde{x}^2) \\ &\simeq x^0_\xi x^0_\eta + x^0_\xi (x - x^0)_\eta + x^0_\eta (x - x^0)_\xi \\ &= x^0_\xi x_\eta + x^0_\eta x_\xi - x^0_\xi x^0_\eta \end{aligned} \tag{A.4}$$

In analogy to eq. (A.4), the remaining terms can be obtained as:

$$\begin{aligned} y_\xi y_\eta &= y^0_\xi y_\eta + y^0_\eta y_\xi - y^0_\xi y^0_\eta \\ x_\xi y_\eta &= x^0_\xi y_\eta + y^0_\eta x_\xi - x^0_\xi y^0_\eta \\ y_\xi x_\eta &= y^0_\xi x_\eta + x^0_\eta y_\xi - y^0_\xi x^0_\eta \end{aligned} \tag{A.5}$$

The resulting system of equations corresponding to linearization of eqs. (A.1) and (A.3) is then given as:

$$\begin{bmatrix} x^0_\eta & y^0_\eta \\ x^0_\xi & y^0_\xi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_\xi + \begin{bmatrix} x^0_\xi & y^0_\xi \\ x^0_\eta & y^0_\eta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_\eta = \begin{bmatrix} x^0_\xi x^0_\eta + y^0_\xi y^0_\eta \\ \frac{1}{2}((\Delta s)^2 + (x^0_\xi)^2 + (x^0_\eta)^2 + (y^0_\xi)^2 + (y^0_\eta)^2) \end{bmatrix} \tag{A.6}$$

or:

$$\mathbf{A}\mathbf{r}_\xi + \mathbf{B}\mathbf{r}_\eta = \mathbf{f} \tag{A.7}$$

If the transformation Jacobian, $x^0_\xi y^0_\eta - x^0_\eta y^0_\xi \neq 0$, then $\mathbf{B}^{-1}$ exist. Furthermore if $\mathbf{B}^{-1}\mathbf{A}$ has real distinct eigenvalues, the system is hyperbolic and therefore well posed for initial data in η [Steger & Chaussee 1980, p. 433]. The characteristic equation for $\mathbf{B}^{-1}\mathbf{A}$ is:

$$\sigma^2 - (x^0_\xi y^0_\eta - x^0_\eta y^0_\xi)^2 = 0 \tag{A.8}$$

which has real distinct roots, σ, as $x^0_\xi y^0_\eta - x^0_\eta y^0_\xi \neq 0$. If however $x^0_\xi y^0_\eta - x^0_\eta y^0_\xi = 0$, the mapping from x, y to ξ, η is no longer one-to-one.

In order to ensure a non-singular mapping from x, y to ξ, η, an alternative set of equations are formulated in a cell volume scheme. Again, the constraint of orthogonality is imposed. If a finite grid cell volume is specified, the transformation Jacobian should be non-singular [Steger & Chaussee 1980, p. 433]:

$$dxdy = (x_\xi y_\eta - x_\eta y_\xi) d\xi d\eta = J\, d\xi d\eta \tag{A.9}$$

In the numerical implementation $\Delta\xi = \Delta\eta = 1$, so the Jacobian determinant is approximately equal to the physical cell volume $\Delta x \Delta y$. A set of grid generation equations are then given by:

$$x_\xi x_\eta + y_\xi y_\eta = 0 \tag{A.10}$$

$$x_\xi y_\eta - x_\eta y_\xi = J \equiv V\,(Area) \tag{A.11}$$

Here eq. (A.10) is the constraint of orthogonality and in eq. (A.11) V represents the grid cell volume, which is user specified. Linearization of the two equations yields:

$$\begin{bmatrix} x_\eta^0 & y_\eta^0 \\ y_\eta^0 & -x_\eta^0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_\xi + \begin{bmatrix} x_\xi^0 & y_\xi^0 \\ -y_\xi^0 & x_\xi^0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_\eta = \begin{bmatrix} 0 \\ V + V^0 \end{bmatrix} \tag{A.12}$$

which again can be put on the same form as eq. (A.7).

If $(x_\xi^0)^2 + (y_\xi^0)^2 \neq 0$ then $\mathbf{B}^{-1}$ exists. Furthermore this means that $\mathbf{B}^{-1}\mathbf{A}$ is a symmetric matrix which has real eigenvalues:

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} x_\eta^0 & y_\eta^0 \\ y_\eta^0 & -x_\eta^0 \end{bmatrix} \\ \mathbf{B}^{-1} &= \frac{1}{(x_\xi^0)^2 + (y_\xi^0)^2} \begin{bmatrix} x_\xi^0 & -y_\eta^0 \\ y_\xi^0 & x_\xi^0 \end{bmatrix} \\ \mathbf{f} &= \begin{bmatrix} 0 \\ V + V^0 \end{bmatrix} \\ \mathbf{B}^{-1}\mathbf{A} &= \frac{1}{(x_\xi^0)^2 + (y_\xi^0)^2} \begin{bmatrix} x_\xi^0 x_\eta^0 - y_\xi^0 y_\eta^0 & x_\xi^0 y_\eta^0 + x_\eta^0 y_\xi^0 \\ x_\xi^0 y_\eta^0 + x_\eta^0 y_\xi^0 & y_\xi^0 y_\eta^0 - x_\xi^0 x_\eta^0 \end{bmatrix} \end{aligned} \tag{A.13}$$

So, again the system is hyperbolic and suitable for marching in η.

A.2 Obtaining the Algorithm

A mesh generation algorithm is developed using eqs. (A.10) and (A.11) by choosing V and subsequently solving numerically with specified initial data along the body surface. A non iterative implicit finite difference scheme is used which is centrally differenced in ξ and first order accurate in the marching direction η as follows.

$$\begin{aligned} \mathbf{r}_\eta &= \mathbf{r}_{j,k+1} - \mathbf{r}_{j,k} \\ \mathbf{r}_\xi &= \frac{\mathbf{r}_{j+1,k+1} - \mathbf{r}_{j-1,k+1}}{2} \end{aligned} \tag{A.14}$$

Let $\Delta\xi = \Delta\eta = 1$ so that $\eta = k - 1$ and $\xi = j - 1$. Eq. (A.12) can then replace eqs. (A.10) and (A.11) to second order numerical accuracy, and be differenced as:

$$\mathbf{r}_{j,k+1} - \mathbf{r}_{j,k} + \mathbf{B}^{-1}\mathbf{A}\frac{\mathbf{r}_{j+1,k+1} - \mathbf{r}_{j-1,k+1}}{2} = \mathbf{B}^{-1}\mathbf{f}_{j,k+1} + \varepsilon(\nabla_j\Delta_j)^2\mathbf{r}_{j,k} \tag{A.15}$$

where lower index refers to node numbering, $\mathbf{A}$, $\mathbf{B}^{-1}$, and $\mathbf{f}$ are given by eq. (A.13), and $x^0_\xi, y^0_\xi, x^0_\eta, y^0_\eta, V^0$ are evaluated at the previous level k as

$$\begin{aligned} x^0_\xi &= \frac{x_{j+1,k} - x_{j-1,k}}{2} \\ y^0_\xi &= \frac{y_{j+1,k} - y_{j-1,k}}{2} \\ x^0_\eta &= -\frac{y^0_\xi V^0}{(x^0_\xi)^2 + (y^0_\xi)^2} \\ y^0_\eta &= \frac{x^0_\xi V^0}{(x^0_\xi)^2 + (y^0_\xi)^2} \end{aligned} \tag{A.16}$$

and V is given as

$$V = h\sqrt{(x^0_\xi)^2 + (y^0_\xi)^2} \tag{A.17}$$

where

h is a user specified height

The term $(\nabla_j\Delta_j)^2$ is an added, fourth order numerical dissipation term, given by:

$$(\Delta_j\nabla_j)^2\mathbf{r}_{j,k} = \mathbf{r}_{j+2,k} - 4\mathbf{r}_{j+1,k} + 6\mathbf{r}_{j,k} - 4\mathbf{r}_{j-1,k} + \mathbf{r}_{j-2,k} \tag{A.18}$$

[Pulliam 1986, p. 1933]

The resulting equations for solving for each new $k+1$ will be on the form $\mathbf{Ux} = \mathbf{v}$ where $\mathbf{Ux}$ and $\mathbf{v}$ are given as:

$$\begin{aligned} \mathbf{Ux} &= \mathbf{r}_{j,k+1} + \mathbf{B}^{-1}\mathbf{A}\frac{\mathbf{r}_{j+1,k+1} - \mathbf{r}_{j-1,k+1}}{2} \\ \mathbf{v} &= \mathbf{B}^{-1}\mathbf{f}_{j,k+1} + \varepsilon(\nabla_j \Delta_j)^2 \mathbf{r}_{j,k} + \mathbf{r}_{j,k} \end{aligned} \tag{A.19}$$

A.2.1 Boundary Conditions

As with all PDE's, special care has to be taken when considering the boundaries, given by $j = 1$ and $j = n$, where n is the number of surface nodes. Two different boundary conditions have been considered here; a cyclic mesh enclosing a body surface, and a vertical boundary mesh growing from a "horizontal" body surface.

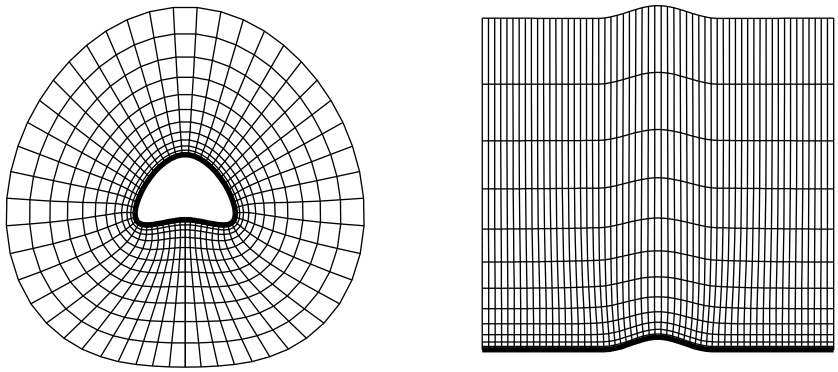
If the mode is set to cyclic mesh, the boundary conditions are given so that the nodes on each boundary for the same value of η are treated as being coinciding. This gives the following alterations:

$$\begin{aligned} \mathbf{r}_{j-1,k} &= \mathbf{r}_{n,k} \\ \mathbf{r}_{j-1,k+1} &= \mathbf{r}_{n,k+1} \\ \mathbf{r}_{n+1,k} &= \mathbf{r}_{1,k} \\ \mathbf{r}_{n+1,k+1} &= \mathbf{r}_{1,k+1} \end{aligned} \tag{A.20}$$

If the mode is set to vertical boundary mesh, the boundary conditions are given so the boundaries are vertical. This gives the following alterations:

$$\begin{aligned} x_{1,k} &= x_{1,k-1} \\ y_{1,k} &= y_{2,k} \\ x_{n,k} &= x_{n,k-1} \\ y_{n,k} &= y_{n-1,k} \end{aligned} \tag{A.21}$$

Depending on the boundary conditions chosen, the mesh chosen will either be cyclic fig. A.2a, or with a vertical boundary fig. A.2b.



a) Cyclic mesh *b)* Vertical boundary mesh

Figure A.2: The two available mesh modes. The bold lines represent the body surface.

A.2.2 Dissipation

The dissipation given by eq. (A.18) is controlled by the user defined parameter ε. At the boundaries special considerations has to be made for the dissipation as it is of fourth order.

If the mode is set to cyclic mesh, the alterations are given as eq. (A.20), plus an additional two:

$$\begin{aligned} \mathbf{r}_{j-2,k} &= \mathbf{r}_{n-1,k} \\ \mathbf{r}_{n+2,k} &= \mathbf{r}_{2,k} \end{aligned} \tag{A.22}$$

If the mode is set to vertical boundary mesh, the dissipation is merely turned off for the outer 2 nodes in both ends:

$$\begin{aligned} (\Delta_j \nabla_j)^2 \mathbf{r}_{1,k} &= 0 \\ (\Delta_j \nabla_j)^2 \mathbf{r}_{2,k} &= 0 \\ (\Delta_j \nabla_j)^2 \mathbf{r}_{n-1,k} &= 0 \\ (\Delta_j \nabla_j)^2 \mathbf{r}_{n,k} &= 0 \end{aligned} \tag{A.23}$$

An alternative approach would be to add a second order dissipation at $j = 2$ and $j = n - 1$, however the body surface at the boundaries should be sufficiently smooth, so dissipation is not needed there.

A.2.3 Volume Averaging

In order to obtain an evenly spaced mesh in the far field, removing converging or diverging tendencies originating from the geometry of the body surface, volume averaging is used. The volume averaging is given as:

$$V^{\star}_{j,k} = \alpha V_{j,k} + (1-\alpha)\frac{1}{n}\sum_{i=1}^{n} V_{i,k} \tag{A.24}$$

where

$V^{\star}_{j,k}$	is the averaged volume
α	is a parameter between 0 and 1
$V_{j,k}$	is the volume given by eq. (A.17)

B
Spline Theory

In the process of generating a hyperbolic mesh, the MatLab function *spline* is used. In this appendix the theory behind this function is described. Unless anything else is mentioned the following is based on [Mathews & Fink 2004, pp. 280-290].

Generally the spline is used to fit a curve through data points. The property of the spline is, that the spline goes through all the points and does not only generate a curve making the best fit of the available data. This is done by considering the distance between two adjoining points and determine a polynomial which include the considered points and have continuity in both the first and second derivative across node points.

The spline function in MatLab makes use of a so-called *Cubic Spline*. It is called cubic because between each two points a third degree polynomial is fitted. The final spline, $S(x)$, consists of N polynomials where $N+1$ is the number of points to be fitted i.e $\{(x_k, y_k)\}_{k=0}^{N}$. This can be written as:

$$\begin{aligned} S(x) = S_k(x) =& s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3 \\ & \text{for} \quad x \in [x_k, x_{k+1}] \quad \text{and} \quad k = 0, 1, \ldots, N-1 \end{aligned} \tag{B.1}$$

where

$S_k(x)$	is a third degree polynomial (Cubic polynomial)	
$s_{k,i}$	are coefficients in the polynomial	$i = 0, 1, \ldots, 3$

Fig. B.1 shows an example of the numbering of points and splines when four points exist.

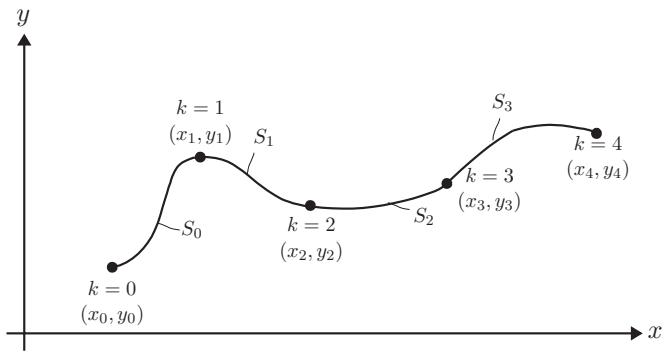


Figure B.1: Example of a spline and definition of index

The task is to determine the coefficients, $s_{k,i}$, in the spline in terms of known values.

For the final spline to fit the data in a satisfying way it has to fulfill 4 criteria which are:

$$\begin{aligned}
\text{I}: &\ S(x_k) = y_k && \text{for} && k = 0, 1, \ldots, N \\
\text{II}: &\ S_{k-1}(x_k) = S_k(x_k) && \text{for} && k = 1, \ldots, N-1 \\
\text{III}: &\ S'_{k-1}(x_k) = S'_k(x_k) && \text{for} && k = 1, \ldots, N-1 \\
\text{IV}: &\ S''_{k-1}(x_k) = S''_k(x_k) && \text{for} && k = 1, \ldots, N-1
\end{aligned}$$

Statement I indicates that the piecewise cubic polynomials interpolate all data points. By inserting in eq. (B.1) this yields:

$$\begin{aligned}
S_k(x_k) =& s_{k,0} + s_{k,1}(x_k - x_k) + s_{k,2}(x_k - x_k)^2 + s_{k,3}(x_k - x_k)^3 \\
S_k(x_k) =& s_{k,0} = y_k
\end{aligned} \tag{B.2}$$

Statements III and IV indicate that the cubic polynomials make up a smooth curve which is continuous in both the first and second derivative. These statements form the basis for the derivation of the polynomial coefficients.

Each cubic polynomial contains four unknowns, the coefficients $s_{k,0}$, $s_{k,1}$, $s_{k,2}$, $s_{k,3}$, and as there are N polynomials this gives $4N$ coefficients to be determined. The data points $\{(x_k, y_k)\}_{k=0}^{N}$ gives $N+1$ conditions and statement II-IV gives $N-1$ conditions each. Together this supplies $N+1+3(N-1) = 4N-2$ conditions leaving two unknown coefficients to be determined. This will be dealt with later by considering end conditions of either the first, $S'(x)$, or second derivative, $S''(x)$.

In the following an equation system is set up which can be used to determine the coefficients, $s_{k,i}$, in eq. (B.1). The starting point for this derivation is the continuity of the second derivative.

As a consequence of the polynomials being cubic the second derivative is piecewise linear on $[x_0, x_N]$. Due to this linearity a linear Lagrange interpolation can be used to determine the value of the second derivative between two adjacent points. The linear Lagrange interpolation formula is given by:

$$S_k''(x) = S''(x_k)\frac{x - x_{k+1}}{x_k - x_{k+1}} + S''(x_{k+1})\frac{x - x_k}{x_{k+1} - x_k} \tag{B.3}$$

By defining $m_k = S''(x_k)$ and $h_k = x_{k+1} - x_k$ eq. (B.3) can be written as:

$$\begin{aligned} S_k''(x) =& \frac{m_k}{h_k}(x_{k+1} - x) + \frac{m_{k+1}}{h_k}(x - x_k) \\ \text{for} \quad & x_k \le x \le x_{k+1} \quad \wedge \quad k = 0, 1, \ldots, N-1 \end{aligned} \tag{B.4}$$

A solution to eq. (B.4) is obtained by integrating twice.:

$$\begin{aligned} S'(x) =& \int S''(x)\, dx \\ =& \frac{m_k}{h_k}x_{k+1}x - \frac{m_k}{2h_k}x^2 + \frac{m_{k+1}}{2h_k}x^2 - \frac{m_{k+1}}{h_k}x_k x + C_1 \end{aligned} \tag{B.5}$$

$$\begin{aligned} S(x) =& \int S'(x)\, dx \\ =& \frac{m_k}{2h_k}x_{k+1}x^2 - \frac{m_k}{6h_k}x^3 + \frac{m_{k+1}}{6h_k}x^3 - \frac{m_{k+1}}{2h_k}x_k x^2 + C_1 x + C_2 \end{aligned} \tag{B.6}$$

where

C_1, C_2 are integration constants to be determined

The constants, C_1 and C_2, can after some rewriting be written as:

$$\begin{aligned} C_1 &= p_k x_{k+1} - q_k x_k + \frac{m_k}{6h_k} x_{k+1}^3 - \frac{m_{k+1}}{6h_k} x_k^3 + C_2 \\ C_2 &= q_k - p_k - \frac{m_k}{2h_k} x_{k+1}^2 + \frac{m_{k+1}}{2h_k} x_k^2 \end{aligned} \tag{B.7}$$

where

p_k, q_k are two new constants to be determined

By inserting the constants in eq. (B.7), eq. (B.6) can be written as:

$$S_k(x) = \frac{m_k}{6h_k}(x_{k+1} - x)^3 + \frac{m_{k+1}}{6h_k}(x - x_k)^3 + p_k(x_{k+1} - x) + q_k(x - x_k) \tag{B.8}$$

Using eq. (B.2) for k and $k+1$ in eq. (B.8) results in the following two equations containing p_k and q_k respectively:

$$y_k = \frac{m_k}{6} h_k^2 + p_k h_k \quad \text{and} \quad y_{k+1} = \frac{m_{k+1}}{6} h_k^2 + q_k h_k \tag{B.9}$$

Solving eq. (B.9) for p_k and q_k respectively and inserting in eq. (B.8) yields the following equation for the cubic function:

$$\begin{aligned} S_k(x) = &- \frac{m_k}{6h_k}(x_{k+1} - x)^3 + \frac{m_{k+1}}{6h_k}(x - x_k)^3 \\ &+ \left(\frac{y_k}{h_k} - \frac{m_k h_k}{6}\right)(x_{k+1} - x) + \left(\frac{y_{k+1}}{h_k} - \frac{m_{k+1} h_k}{6}\right)(x - x_k) \end{aligned} \tag{B.10}$$

In eq. (B.10) the only unknowns are the second derivatives represented by m. To calculate these values the derivative of (B.10) is used:

$$\begin{aligned} S_k'(x) = &- \frac{m_k}{2h_k}(x_{k+1} - x)^2 + \frac{m_{k+1}}{2h_k}(x - x_k)^2 \\ &- \left(\frac{y_k}{h_k} - \frac{m_k h_k}{6}\right) + \frac{y_{k+1}}{h_k} - \frac{m_{k+1} h_k}{h_k} \end{aligned} \tag{B.11}$$

Eq. (B.11) is evaluated at x_k. Secondly k in eq. (B.11) is replaced by $k-1$ and again evaluated at x_k. This yields the following two equations:

$$S_k'(x_k) = -\frac{m_k}{3} h_k - \frac{m_{k+1}}{6} h_k + d_k \quad \text{where} \quad d_k = \frac{y_{k+1} - y_k}{h_k} \tag{B.12}$$

$$S_{k-1}'(x_k) = \frac{m_k}{3} h_{k-1} + \frac{m_{k-1}}{6} h_{k-1} + d_{k-1} \tag{B.13}$$

Eqs. (B.12) and (B.13) represent the right- and left hand side of statement III respectively. Insertion yields:

$$h_{k-1}m_{k-1} + 2(h_{k-1} + h_k)m_k + h_k m_{k+1} = u_k \qquad \text{(B.14)}$$
$$\text{where} \quad u_k = 6(d_k - d_{k-1}) \quad \text{for} \quad k = 1, 2, \ldots, N-1$$

Eq. (B.14) expresses a useful relation between m_{k-1}, m_k and m_{k+1}. The only unknowns are the values of m_{k-1}, m_k and m_{k+1} while all other terms are constants dependent on only the known data points. This leaves an under determined equation with $N+1$ unknowns and $N-1$ linear equations.

It has earlier been mentioned that the two additional equations required to solve the equation system is obtained by specifying end conditions of the derivatives of $S_k(x)$. These equations are used to eliminate m_0 from the first equation and m_N from equation $N-1$.

The end point constraints are usually specified in one of the five different ways listed below:

1. Clamped cubic spline: Specify $S'(x_0)$ and $S'(x_N)$
2. Natural cubic spline: $m_0 = 0$ and $m_N = 0$
3. Extrapolate $S''(x)$ to the end points: m_0 a function of m_1 and m_2, m_N a function of m_{N-1} and m_{N-2}
4. $S''(x)$ is constant near the end points: $m_0 = m_1$, $m_N = m_{N-1}$
5. Specify $S''(x)$ at each end point: $m_0 = S''(x_0)$, $m_N = S''(x_N)$

By default the *spline* function in MatLab uses a *not-a-knot* condition at the end points. This interpolation scheme makes use of the condition that the third derivative of $S_k(x)$ is continuous at x_1 and x_{N-1}. Another option is to specify the first derivative which gives a clamped spline as in point 1 in the list above. This is the spline used in this project as a tool for generating the hyperbolic mesh and is therefore the only strategy considered in the following. A reference is made to [*Mathews and Fink, 2004, pp. 280-290*] for more information about the other four end conditions.

Regardless of the end conditions chosen, eq. (B.14) can be rewritten and thereby obtain a tridiagonal linear equation system. For $k = 1$ and $k = N-1$ eq. (B.14) attain a different form than for $k = 2, 3, \ldots, N-2$ which have been taken into

account setting up the equation system. The linear system can be written as:

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_1 & b_2 & c_2 & & & & \\ & a_2 & & & & & \\ & & \ddots & & & & \\ & & & & c_{N-3} & & \\ & & & a_{N-3} & b_{N-2} & c_{N-2} & \\ & & & & a_{N-2} & b_{N-1} & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \\ \vdots \\ \\ m_{N-2} \\ m_{N-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \\ \vdots \\ \\ v_{N-2} \\ v_{N-1} \end{bmatrix} \tag{B.15}$$

In matrix notation eq. (B.15) can be written as:

$$\mathbf{Am} = \mathbf{v} \tag{B.16}$$

The entries in **A** and **v** are different dependent on the type of spline wanted and are not further elaborated here. Later the entries in **A** and **v** for a clamped spline are introduced.

Eq. (B.15) is solved for **m** and the results for m_1 and m_{N-1} are used to compute the values of m_0 and m_N which have been eliminated from the equation system.

The coefficients, $s_{k,i}$, in eq. (B.1) are hereafter determined by:

$$\begin{aligned} s_{k,0} &= y_k \\ s_{k,1} &= d_k - \frac{h_k(2m_k + m_{k+1})}{6} \\ s_{k,2} &= \frac{m_k}{2} \\ s_{k,3} &= \frac{m_{k+1} - m_k}{6h_k} \end{aligned} \tag{B.17}$$

The coefficients determined by eq. (B.17) are inserted in eq. (B.1) and the final spline is defined over the entire interval $x_0 \leq x \leq x_N$.

B.1 Clamped Cubic Spline

In this appendix the procedure for fitting a spline with end conditions for the slope of the spline is presented and afterwards examples of clamped splines are shown.

For the clamped cubic spline the elimination of m_0 and m_N in the equation system is done by considering eqs. (B.12) and (B.13) and solving these for m_k which yields the following two equations:

$$m_k = \frac{3}{h_k}(d_k - S'_k(x_k)) - \frac{m_{k+1}}{2} \tag{B.18}$$

$$m_k = \frac{3}{h_{k-1}}(S'_k(x_k) - d_{k-1}) - \frac{m_{k-1}}{2} \tag{B.19}$$

Inserting $k = 0$ in eq. (B.18) and $k = N$ in (B.19) yields:

$$m_0 = \frac{3}{h_0}(d_0 - S'_0(x_0)) - \frac{m_1}{2} \tag{B.20}$$

$$m_N = \frac{3}{h_{N-1}}(S'_N(x_N) - d_{N-1}) - \frac{m_{N-1}}{2} \tag{B.21}$$

Eqs. (B.20) and (B.21) are used when the equation system for $k = 1, 2, \ldots, N-1$ have been solved. In eqs. (B.20) and (B.21) $S'_0(x_0)$ and $S'_N(x_N)$ are the prescribed end conditions of the spline.

As stated earlier the equation system is dependent on the type of spline. For the clamped spline the linear system is made up of the following equations:

$$\begin{aligned}
&\left(\frac{3}{2}h_0 + 2h_1\right)m_1 + h_1 m_2 = u_1 - 3(d_0 - S'(x_0)) \qquad \text{for} \qquad k = 1 \\
&h_{k-1}m_{k-1} + 2(h_{k-1} + h_k)m_k + h_k m_{k+1} = u_k \qquad \text{for} \qquad k = 2, 3, \ldots, N-2 \\
&h_{N-2}m_{N-2} + 2\left(2h_{N-2} + \frac{3}{2}h_{N-1}\right)m_{N-1} = u_{N-1} - 3(S'(x_N) - d_{N-1}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for} \qquad k = N-1
\end{aligned} \tag{B.22}$$

By comparing eqs. (B.15) and (B.22) the equation system for clamped splines can be written as:

$$\begin{bmatrix} \frac{3}{2}h_0 + 2h_1 & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & & \ddots & & & \\ & & & h_{N-3} & 2(h_{N-3}+h_{N-2}) & h_{N-2} \\ & & & & h_{N-2} & 2h_{N-2} + \frac{3}{2}h_{N-1} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{N-2} \\ m_{N-1} \end{bmatrix} = \begin{bmatrix} u_1 - 3(d_0 - S'(x_0)) \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} - 3(S'(x_N) - d_{N-1}) \end{bmatrix} \tag{B.23}$$

B.1.1 Examples

The first example illustrates the procedure for two different data sets listed below and shown in fig. B.2. Also $\mathbf{A}$ and $\mathbf{v}$ are shown where the entries are determined as indicated in eq. (B.23) where $h_k = x_{k+1} - x_k$, $d_k = \frac{y_{k+1}-y_k}{h_k}$ and $u_k = 6(d_k - d_{k-1})$.

$$\begin{aligned} X_1 &= [0; 1; 2; 3] \\ Y_1 &= [0; 0.5; 2; 1.5] \\ S'(0) &= 0.2 \\ S'(3) &= -1 \\ \mathbf{A} &= \begin{bmatrix} 3.5 & 1.0 \\ 1.0 & 3.5 \end{bmatrix} \\ \mathbf{v} &= \begin{bmatrix} 5.1 \\ -10.5 \end{bmatrix} \end{aligned} \tag{B.24}$$

$$
\begin{aligned}
X_2 &= [1;2;3;4] \\
Y_2 &= [2.5;0;3;0.5] \\
S'(0) &= 2 \\
S'(4) &= 1 \\
\mathbf{A} &= \begin{bmatrix} 3.5 & 1.0 \\ 1.0 & 3.5 \end{bmatrix} \\
\mathbf{v} &= \begin{bmatrix} 46.5 \\ -43.5 \end{bmatrix}
\end{aligned}
\tag{B.25}
$$

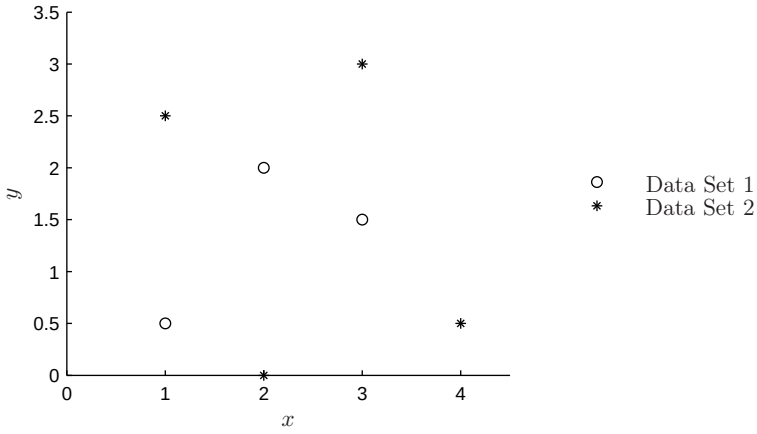


Figure B.2: Illustration of data points

The equation system given by eq. (B.16) is solved and afterwards m_0 and m_N are determined by eq. (B.20) and (B.21). Finally the coefficients, $s_{k,i}$ are determined by eq. (B.17). The results are listed in tab. B.1 for the data in eq. (B.24) and tab. B.2 for the data in eq. (B.25).

Table B.1: Coefficients for cubic spline for data set 1.

	$s_{k,0}$	$s_{k,1}$	$s_{k,2}$	$s_{k,3}$
$k=0$	0	0,2	−0,18	0,48
$k=1$	0,5	1,28	1,26	−1,04
$k=2$	2	0,68	−1,86	0,68
	m_0	m_1	m_2	m_3
	−0.36	2.52	−3.72	0.36

Table B.2: Coefficients for cubic spline for data set 2.

	$s_{k,0}$	$s_{k,1}$	$s_{k,2}$	$s_{k,3}$
$k=0$	2,5	2	−11,33	−6,83
$k=1$	0	−0,17	9,17	−6
$k=2$	3	0,17	−8,83	6,17
	m_0	m_1	m_2	m_3
	−22.67	18.33	−17.67	19.33

Fig. B.3 shows the cubic splines determined above. In addition the results from a call of the MatLab function *spline* with the same data are shown and it is clear that they coincide with the manually determined splines.

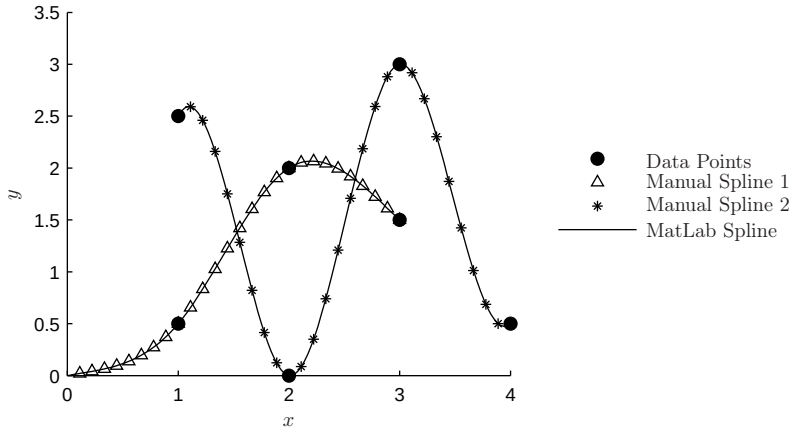


Figure B.3: Cubic splines for data set 1 and 2

To illustrate the effect of the end conditions two cubic splines fitting the data set

in eq. (B.24) but with different end conditions have been made. The gradients used for end conditions are the same as in eq. (B.24) and (B.25) but both on the first data set as mentioned. A plot of the splines are shown in fig B.4 along with the results from a call of the *spline* function in MatLab.

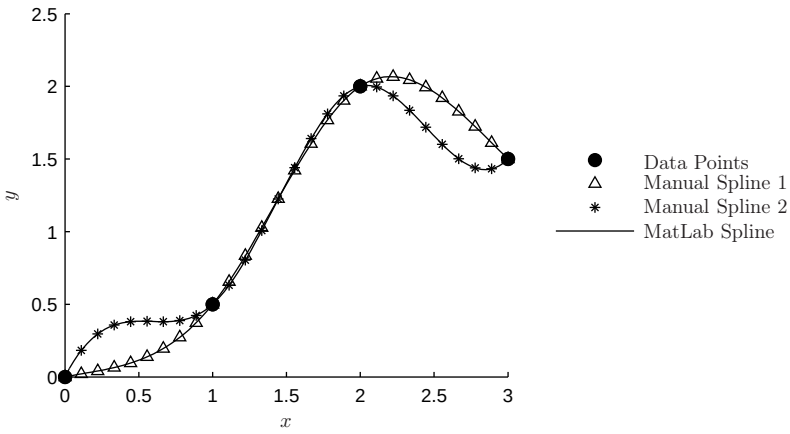


Figure B.4: Illustration of effect of end conditions

As it is seen from fig. B.4 the splines are different even though they go through the same four points.

B.2 Limitations

The theory presented in the previous appendix has the limitation, that the considered data points have to be lined up in increasing order of x. When the data points have x-values that not necessarily increase from point to point, the problem arises, that one x value may very well have several y values. This means that for an evaluation of x the function cannot give an unambiguous output in the form of a single y. The figure below shows an example of this, where two or more y values exist for a given x. Despite splines being used for several purposes in the process of generating the hyperbolic mesh, the only time at which the above mentioned problem can occur is when defining the body surface using splines. In all other cases the spline is used to generate values between 0 and 1 defining some type of distribution and the value used for this is always increasing.

When splining the body surface, the spline function is called twice. To do this, the curve length between the adjacent points in the spline is needed. The curve length calculated is the cumulative linear distance between the points as indi-

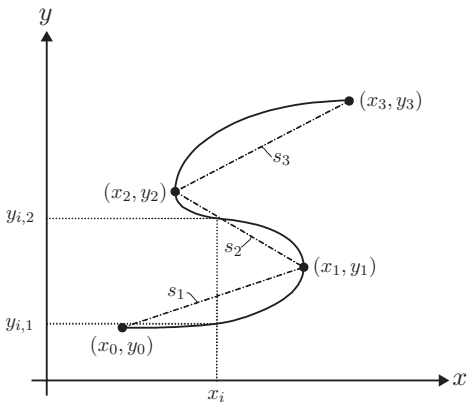
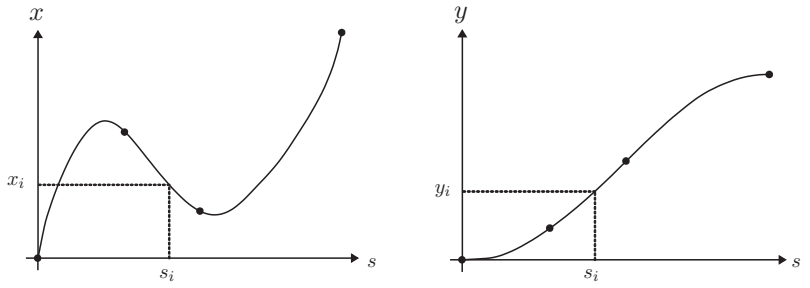


Figure B.5: x and y not one-to-one

cated by s_k in fig. B.5.

Two splines are then created, one using the input s, x and one using the input s, y, where s is calculated as defined above. This creates two splines, shown in figs. B.6a and B.6b for s, x and s, y respectively. For a given curve length s, the corresponding x- and y-values can now be found, which if combined will produce the spline shown in fig. B.5.



a) s, x spline *b) s, y* spline

Figure B.6: Illustration of method for determining points for splines with non-increasing x-values

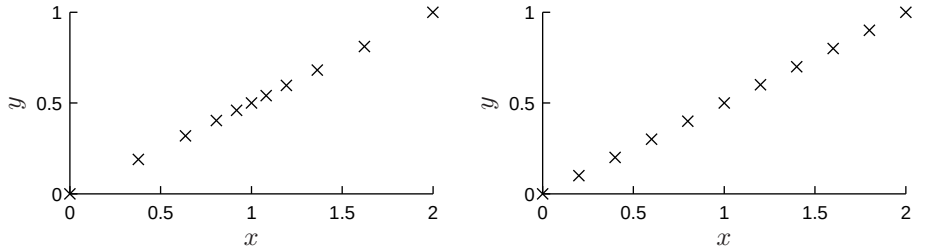
The node distribution on the geometry is determined by a stretch function which indicate the normalized curve length from the starting point to the respective point. This curve length is the one used to determine the x- and y-values of the node.

note: Special care has to be taken when utilizing the clamped spine, that is when constraining the end slopes of the spline. In the following an example is given with a linear spline going from x, y=(0,0) to x, y=(2,1). 11 equidistant points are used as curve lengths to evaluate the spline, going from 0 to $\sqrt{5}$, the length of the line.

Let a_1 and a_2 define the end slope at one end and let b_1 and b_2 define the end slope at the other end, so that a_1 is the change of x for $\Delta s = 1$, a_2 is the change of y for $\Delta s = 1$. Analogous for the other end, b_1 and b_2 are defined.

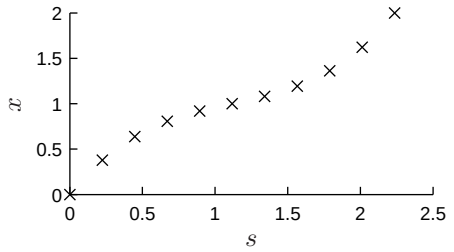
Firstly, we define the end slopes a_1=b_1=2 and a_2=b_2=1, corresponding to the slope of the line $y = 0.5x$. Evaluating this spline for the 11 equidistant points yields a straight line going from x, y=(0,0) to x, y=(2,1) as shown in fig B.7a. However, it can be seen that the points are no longer equidistant. The reason for this can be found by examining the s, x and s, y plots, shown in figs. B.7c and B.7e. Here it is obvious that the relation between the curve length s and the corresponding values for x and y is no longer linear but combining the x values from fig. B.7c and the y values from fig. B.7e does produce a straight line. This is due to the fact that a_1 and a_2 or b_1 and b_2 are *not* to be considered as vectors defining the "direction" of the slope on the x, y spline, but rather as the end slopes on the s, x and s, y splines respectively.

If we instead of using a_1=b_1=2 and a_2=b_2=1 try to normalize these values, to the actual values corresponding to $\Delta s = 1$ we get a_1=b_1=$2/\sqrt{5}$ and a_2=b_2=$1/\sqrt{5}$. This too, produces an even line as can be seen in fig. B.7b, but unlike with fig. B.7a the points are equidistant as desired. Inspecting figs. B.7d and B.7f it can be seen that the relations between s and x or y are all linear as they should be.

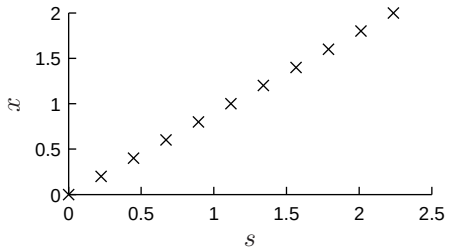


a) x, y - regular end slopes

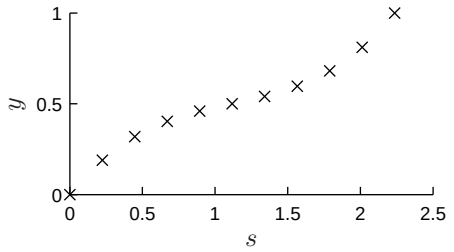
b) x, y - normalized end slopes



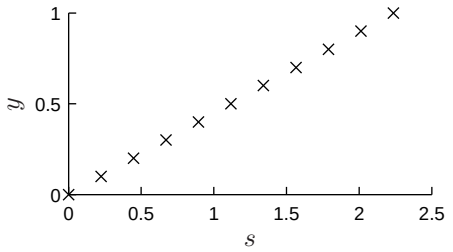
c) s, x - regular end slopes



d) s, x - normalized end slopes



e) s, y - regular end slopes



f) s, y - normalized end slopes

Figure B.7: Examples of splines with "regular" and "normalized" end slopes

In conclusion to this, the end slopes should be "normalized", in order not to interfere with the node distribution given by the input.

C

Computational Fluid Dynamics, Theory

In this appendix, the basic theory and ideas behind Computational Fluid Dynamics are explained. Unless otherwise noted, this appendix is based on [Wilcox 2002]. First the governing equations for fluid flow will be described, then a select few of the two-equation turbulence models will be described, and finally the finite volume method will be described.

C.1 Governing Equations for Fluid Flow

To be able to model a turbulent flow, one must first realize that turbulence can be described as random variations in the various properties which describe the flow. The constant fluctuations of the flow properties make them difficult to describe, to which end Reynolds time averaging is used. In what follows, index notation has been used where roman letters attain the value 1, 2, or 3.

C.1.1 Reynolds Time Averaging

The concept of Reynolds time averaging is, that for an instantaneous flow variable such as $u_i(\mathbf{x}, t)$, it's time-average, $U_i(\mathbf{x})$, is defined as:

$$U_i(\mathbf{x}) = \lim_{T\to\infty} \frac{1}{T} \int_t^{t+T} u_i(\mathbf{x}, t)\, dt \tag{C.1}$$

Using Reynolds time averaging to describe the instantaneous property of e.g. the flow velocity, $u_i(\mathbf{x}, t)$, this is given as the sum of a mean and a fluctuating part:

$$u_i(\mathbf{x}, t) = U_i(\mathbf{x}) + u_i'(\mathbf{x}, t) \tag{C.2}$$

where

$U_i(\mathbf{x})$ is the mean
$u_i'(\mathbf{x}, t)$ is the fluctuating part

By use of eqs. (C.1) and (C.2), it is assumed that U_i is not time dependent. For many flows however, the mean value varies over time. To incorporate this, eqs. (C.1) and (C.2) are replaced with eqs. (C.3) and (C.4):

$$u_i(\mathbf{x}, t) = U_i(\mathbf{x}, t) + u_i'(\mathbf{x}, t) \tag{C.3}$$

$$U_i(\mathbf{x}, t) = \frac{1}{T}\int_t^{t+T} u_i(\mathbf{x}, t)\, dt, \qquad T_1 \ll T \ll T_2 \tag{C.4}$$

where

T_1 is the maximum period of the fluctuations
T_2 is the time scale characteristic of the slow variations in the flow, that are not regarded as turbulence

It is here assumed that T_1 and T_2 differ by several orders of magnitude or else the equations cannot be used.

The Reynolds-averaged products of two and three properties are given in eqs. (C.5) and (C.6):

$$\overline{\phi\psi} = \overline{(\Phi + \phi')(\Psi + \psi')} = \overline{\Phi\Psi + \Phi\psi' + \Psi\phi' + \phi'\psi'} = \Phi\Psi + \overline{\phi'\psi'} \tag{C.5}$$

$$\overline{\phi\psi\xi} = \Phi\Psi\Xi + \overline{\phi'\psi'}\Xi + \overline{\psi'\xi'}\Phi + \overline{\phi'\xi'}\Phi + \overline{\phi'\psi'\xi'} \tag{C.6}$$

where it has been used that the mean of a fluctuation quantity is 0, $\overline{\phi'} = \overline{\psi'} = \overline{\xi'} = 0$. Hence, all terms with only one fluctuating quantity vanish, but terms with two or more fluctuating quantities do not.

C.1.2 Reynolds Averaged Equations

Assuming incompressible, constant-property flow, the Navier-Stokes equations for conservation of mass and momentum are given in eqs. (C.7) and (C.8):

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{C.7}$$

$$\rho \frac{\partial u_i}{\partial t} + \rho u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial t_{ji}}{\partial x_j} \tag{C.8}$$

where

u_i	is the velocities
x_i	is the position
t	is the time
p	is the pressure
ρ	is the density
t_{ij}	is the viscous stress tensor defined in eq. (C.9)

The viscous stress tensor is given by:

$$t_{ij} = 2\mu s_{ij} \tag{C.9}$$

where

μ	is the molecular viscosity
s_{ij}	is the strain-rate tensor defined in eq. (C.10)

The strain-rate tensor is given by:

$$s_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{C.10}$$

It should be noted that for simple viscous fluids the strain-rate tensor is symmetric, $s_{ij} = s_{ji}$, which means that the viscous stress tensor is also symmetric, $t_{ij} = t_{ji}$. To simplify the time averaging process, the 2$^{\text{nd}}$ (convective) term in eq. (C.8) is rewritten:

$$\rho u_j \frac{\partial u_i}{\partial x_j} = \rho \frac{\partial}{\partial x_j}(u_i u_j) - \rho u_i \frac{\partial u_j}{\partial x_j} = \rho \frac{\partial}{\partial x_j}(u_i u_j) \tag{C.11}$$

where eq. (C.7) is used to drop the term $\rho u_i \frac{\partial u_j}{\partial x_j}$. Combining eqs. (C.8) - (C.11) gives the Navier-Stokes equation in conservation form:

$$\rho \frac{\partial u_i}{\partial t} + \rho \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}(2\mu s_{ji}) \tag{C.12}$$

Finally, time-averaging of eqs. (C.7) and (C.12) yields the *Reynolds averaged equations of motion in conservation form*:

$$\frac{\partial U_i}{\partial x_i} = 0 \tag{C.13}$$

$$\rho\frac{\partial U_i}{\partial t} + \rho\frac{\partial}{\partial x_j}\left(U_j U_i + \overline{u_j' u_i'}\right) = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}(2\mu S_{ji}) \tag{C.14}$$

Eq. (C.14) can be rewritten to it's most common form, yielding the *Reynolds-averaged Navier-Stokes equations (RANS)*:

$$\rho\frac{\partial U_i}{\partial t} + \rho U_j\frac{\partial U_i}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}(2\mu S_{ji} - \rho\overline{u_j' u_i'}) \tag{C.15}$$

where

$\rho\overline{u_j' u_i'}$ are denoted Reynolds stresses

Apart from replacing instantaneous variables with mean values, the only difference between the instantaneous momentum equations, eq. (C.12), and the RANS equations, eq. (C.14), is the term $\rho\overline{u_i' u_j'}$, the Reynolds-stresses. This is the fundamental problem of turbulence: In order to compute all mean-flow properties of the turbulent flow, a method for calculating the six components of $\overline{u_i' u_j'}$ is needed.

This leads to the subject of turbulence modeling, in which methods are devised to approximate the six unknown correlation terms, $\overline{u_i' u_j'}$, in terms of flow properties that are known. Once this is done, a closed set of equations exists.

C.2 Turbulence Models

A wide range of turbulence models exists, ranging from very simple to very complex models. Depending on the problem at hand, some models may work better than others, and there is no model universally better than the others.

In this appendix, three of the most common turbulence models will be described; the $k-\omega$ model, the $k-\epsilon$ model, and the $k-\omega$ SST model. These models belong to a class of models known as *two-equation models*.

C.2.1 The Turbulence Energy Equation

The kinetic energy (per unit mass) of the turbulent fluctuations, k, is given as:

$$k = \frac{1}{2}\overline{u_i' u_i'} = \frac{1}{2}\left(\overline{u'^2 + v'^2 + w'^2}\right) \tag{C.16}$$

It can be seen that the kinetic energy of the turbulent fluctuations has the dimension $\frac{length^2}{time^2}$. The kinematic eddy viscosity has the dimension $\frac{length^2}{time}$ and Thus, dimensional arguments dictate that the kinematic eddy viscosity in terms of the turbulence length scale, l, and k is given as:

$$\nu_T = c \cdot k^{1/2} l \tag{C.17}$$

where

c is a constant

To determine k, the trace of the Reynolds stress tensor, τ_{ij}, is taken:

$$\tau_{ii} = -\overline{u_i' u_i'} = -2k \tag{C.18}$$

As can be seen, the trace of the Reynolds stress tensor is proportional to the kinetic energy of the turbulent fluctuations. The Reynolds Stress equation is given by eq. (C.19):

$$\frac{\partial \tau_{ij}}{\partial t} + U_k \frac{\partial \tau_{ij}}{\partial x_k} = -\tau_{ik}\frac{\partial U_j}{\partial x_k} - \tau_{jk}\frac{\partial U_i}{\partial x_k} + \epsilon_{ij} - \Pi_{ij} + \frac{\partial}{\partial x_k}\left[\nu \frac{\partial \tau_{ij}}{\partial x_k} + C_{ijk}\right] \tag{C.19}$$

where

$$\Pi_{ij} = \overline{\frac{p'}{\rho}\left(\frac{\partial u_i'}{\partial x_j} + \frac{\partial u_j'}{\partial x_i}\right)} \tag{C.20}$$

$$\epsilon_{ij} = 2\nu \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_j'}{\partial x_k}} \tag{C.21}$$

$$\rho C_{ijk} = \rho \overline{u_i' u_j' u_k'} + \overline{p' u_i'}\delta_{jk} + \overline{p' u_j'}\delta_{ik} \tag{C.22}$$

By taking the trace of the Reynolds stress equation, eq. (C.19), and noting that the trace of the tensor Π_{ij} vanishes for incompressible flow, this leads to the transport equation for the turbulence kinetic energy, given in eq. (C.23).

$$\begin{aligned} \underbrace{\frac{\partial k}{\partial t}}_{\text{unsteady}} + \underbrace{U_j \frac{\partial k}{\partial x_j}}_{\text{convection}} = \underbrace{\tau_{ij}\frac{\partial U_i}{\partial x_j}}_{\text{production}} - \underbrace{\epsilon}_{\text{dissipation}} + \underbrace{\frac{\partial}{\partial x_j}\left(\nu \frac{\partial k}{\partial x_j}\right)}_{\text{molecular diffusion}} \\ - \underbrace{\frac{\partial}{\partial x_j}\left(\frac{1}{2}\overline{u_i' u_i' u_k'}\right)}_{\text{turbulent transport}} - \underbrace{\frac{\partial}{\partial x_j}\left(\frac{1}{\rho}\overline{p' u_j'}\right)}_{\text{pressure diffusion}} \end{aligned} \tag{C.23}$$

where the dissipation per unit mass, ϵ, is defined as:

$$\epsilon = \nu \overline{\frac{\partial u_i'}{\partial x_k} \frac{\partial u_i'}{\partial x_k}} \tag{C.24}$$

The terms of eq. (C.23) represent physical processes occurring as the turbulence moves about in a given flow:

- The sum of the *unsteady term* and the *convection* is the substantial derivative of k which gives the rate of change of k following a fluid particle.
- The *production* represents the rate at which kinetic energy is transferred from the mean flow to the turbulence.
- The *dissipation* is the rate at which turbulence kinetic energy is converted into thermal energy.
- The *molecular diffusion* represents the diffusion of turbulence energy caused by the natural molecular transport process of the fluid.
- The *turbulent transport* is regarded as the rate at which turbulence energy is transported through the fluid by means of turbulent fluctuations.
- The *pressure diffusion* is regarded as another form of transport resulting from correlation of pressure and velocity fluctuations.

The dissipation per unit mass, ϵ, differs from the classical definition, as given in the text above. The true dissipation, ϵ_{true}, is proportional to the square of the fluctuating strain-rate tensor, s_{ik}', and given as:

$$\epsilon_{true} = 2\nu \overline{s_{ik}' s_{ik}'} \qquad s_{ik}' = \frac{1}{2}\left(\frac{\partial u_i'}{\partial x_k} + \frac{\partial u_k'}{\partial x_i}\right) \tag{C.25}$$

Thus, ϵ is given by:

$$\epsilon = \epsilon_{true} - \frac{\partial}{\partial x_k}\left(\nu \overline{u_i' \frac{\partial u_k'}{\partial x_i}}\right) \tag{C.26}$$

In practice there is little difference between ϵ and ϵ_{true}, and the difference should only be expected significant in areas of strong gradients, e.g. in shock waves or the viscous wall region. In the latter case, cf. [Wilcox 2002, p. 105], it has been shown by Bradshaw and Perot (1993) that the maximum difference is only 2%, and can thus be ignored.

The unsteady, molecular diffusion, and convection terms are exact, while the production, dissipation, turbulence transport, and pressure diffusion terms involve unknown correlations. To close this equation , the Reynolds-stress tensor, dissipation, turbulence transport, and pressure diffusion must be specified.

The Reynolds-Stress Tensor

The Boussinesq approximation (which states that the Reynolds-stress tensor, τ_{ij} is proportional to the mean-strain-rate tensor, S_{ij}, and that the coefficient of proportionality between the two is the eddy viscosity, ν_T) is assumed valid, and the specific Reynolds-stress tensor is thus given by:

$$\tau_{ij} = 2\nu_T S_{ij} - \frac{2}{3}k\delta_{ij} \tag{C.27}$$

where

S_{ij} is the mean strain-rate tensor

Note that the trace of τ_{ij} is $-2k$, as prescribed in eq. (C.18), because $S_{ii} = 0$ for incompressible flows.

Turbulent Transport and Pressure Diffusion

It is assumed that the sum of the pressure diffusion and the turbulence transport behave as a gradient-transport process and, cf. [Wilcox 2002, p. 106], Direct Numerical Simulation (DNS) results indicate that the term is quite small for simple flows. Thus it is assumed that the terms can be written as:

$$\frac{1}{2}\overline{u_i' u_i' u_j'} + \frac{1}{\rho}\overline{p' u_k'} = -\frac{\nu_T}{\sigma_k}\frac{\partial k}{\partial x_j} \tag{C.28}$$

where

σ_k is a closure coefficient

Dissipation

Two unknown parameters still exist, the turbulence length scale, l, and the dissipation ϵ. If both properties are assumed strictly to be functions of the turbulence, independent of natural fluid properties, purely dimensional arguments show that:

$$\epsilon \sim \frac{k^{3/2}}{l} \tag{C.29}$$

Hence, a length scale is needed to close the system of equations. This task is up to the individual turbulence models. Finally, by combining eqs. (C.23) and

(C.26), the model of the turbulence kinetic energy equation used by virtually all turbulence energy equation models is given:

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = \tau_{ij} \frac{\partial U_i}{\partial x_j} - \epsilon + \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_T}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] \tag{C.30}$$

where τ_{ij} is given by eq. (C.27).

C.2.2 The $k - \omega$ Model

The reasoning for an ω equation can, on the grounds of dimensional analysis, be given as follows:

- Since k already appears in the constitutive relation, eq. (C.27), it is plausible that ν_T is proportional to k.
- The dimensions of ν_T are $\frac{length^2}{time}$, and the dimensions of k are $\frac{length^2}{time^2}$.
- This means that $\frac{\nu_T}{k}$ has the dimensions $time$.
- Turbulence dissipation, ϵ, has the dimensions $\frac{length^2}{time^3}$.
- This means that $\frac{\epsilon}{k}$ has the dimensions $\frac{1}{time}$.
- Eqs. (C.27) and (C.30) can be closed by introducing a variable (ω) with dimensions $time$ or $\frac{1}{time}$.

Through physical reasoning and dimensional analysis, the following equation for ω is proposed, cf. [Wilcox 2002, p. 120]:

$$\frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} = -\beta \omega^2 + \frac{\partial}{\partial x_j} \left[\sigma \nu_T \frac{\partial \omega}{\partial x_j} \right] \tag{C.31}$$

where

β	is a closure coefficient
σ	is a closure coefficient

There have been different interpretations of ω in the years, but ω can be interpreted simply as the ratio of ϵ to k. The remaining equations, closure coefficients, and auxiliary relations for the $k - \omega$ model are given below.

Kinematic eddy viscosity:

$$\nu_T = \frac{k}{\omega} \tag{C.32}$$

Turbulence kinetic energy:

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = \tau_{ij} \frac{\partial U_i}{\partial x_j} - \beta^* k\omega + \frac{\partial}{\partial x_j}\left[\left(\nu + \sigma^* \nu_T\right) \frac{\partial k}{\partial x_j} \right] \tag{C.33}$$

Specific dissipation rate:

$$\frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} = \alpha \frac{\omega}{k} \tau_{ij} \frac{\partial U_i}{\partial x_j} - \beta \omega^2 + \frac{\partial}{\partial x_j}\left[\left(\nu + \sigma \nu_T\right) \frac{\partial \omega}{\partial x_j} \right] \tag{C.34}$$

Closure coefficients and auxiliary relations:

$$\begin{aligned}
&\alpha = \frac{13}{25}, \qquad \beta = \beta_o f_\beta, \qquad \beta^* = \beta_o^* f_{\beta^*}, \qquad \sigma = \frac{1}{2}, \qquad \sigma^* = \frac{1}{2} \\
&\beta_o = \frac{9}{125}, \qquad f_\beta = \frac{1+70\chi_\omega}{1+80\chi_\omega}, \qquad \chi_\omega \equiv \left| \frac{\Omega_{ij}\Omega_{jk}S_{ki}}{\left(\beta_o^* \omega\right)^3} \right| \\
&f_{\beta^*} = \begin{cases} 1, & \text{for } \chi_k \leq 0 \\ \frac{1+680\chi_k^2}{1+400\chi_k^2}, & \text{for } \chi_k > 0 \end{cases}, \qquad \chi_k \equiv \frac{1}{\omega^3} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} \\
&\beta_o^* = \frac{9}{100}, \qquad \epsilon = \beta^* \omega k, \qquad l = \frac{k^{1/2}}{\omega}
\end{aligned} \tag{C.35}$$

where the mean-rotation and mean-strain-rate tensors, Ω_{ij} and S_{ij} respectively, are defined by:

$$\Omega_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i} \right), \qquad S_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right) \tag{C.36}$$

C.2.3 The $k - \epsilon$ Model

To formulate the $k - \epsilon$ model, the approach is to derive an exact equation for ϵ and finding suitable closure approximations for exact equations that govern it's behavior. The definition of ϵ is given in eq. (C.24), and the exact equation for ϵ is derived by taking the following moment of the Navier-Stokes equation:

$$2\nu \overline{\frac{\partial u_i'}{\partial x_j} \frac{\partial}{x_j} \left[\mathcal{N}\left(u_i\right) \right]} = 0 \tag{C.37}$$

where $\mathcal{N}$ is the Navier-Stokes operator defined by:

$$\mathcal{N}(u_i) = \rho \frac{\partial u_i}{\partial t} + \rho u_k \frac{\partial u_i}{\partial x_k} + \frac{\partial p}{\partial x_i} - \mu \frac{\partial^2 u_i}{\partial x_k \partial x_k} \tag{C.38}$$

This leads to the following equation for ϵ, cf. [Wilcox 2002, p. 123]:

$$\begin{aligned}\frac{\partial \epsilon}{\partial t} + U_j \frac{\partial \epsilon}{\partial x_j} = &\underbrace{-2\nu \left[\overline{u'_{i,k} u'_{j,k}} + \overline{u'_{k,i} u'_{k,j}}\right] \frac{\partial U_i}{\partial x_j} - 2\nu \overline{u'_k u'_{i,j}} \frac{\partial^2 U_i}{\partial x_k \partial x_j}}_{\text{production of dissipation}} \\ &\underbrace{- 2\nu \overline{u'_{i,k} u'_{i,m} u'_{k,m}} - 2\nu^2 \overline{u'_{i,km} u'_{i,km}}}_{\text{dissipation of dissipation}} \\ &\underbrace{+ \frac{\partial}{\partial x_j} \left[\nu \frac{\partial \epsilon}{\partial x_j} - \nu \overline{u'_j u'_{i,m} u'_{i,m}} - 2\frac{\nu}{\rho} \overline{p'_{,m} u'_{j,m}}\right]}_{\text{turbulent transport of dissipation}}\end{aligned} \tag{C.39}$$

which is far more complicated than the turbulence kinetic energy equation, eq. (C.30), and involves many unknown double and triple correlations of fluctuating velocity, pressure and velocity gradients. The terms on the three lines on the right-hand side are generally regarded as *production of dissipation*, *dissipation of dissipation*, and *turbulent transport of dissipation* respectively, cf. [Wilcox 2002, p. 123].

The $k-\epsilon$ model (also referred to as the *standard* $k-\epsilon$ *model*) is then given below.

Kinematic eddy viscosity:

$$\nu_T = \frac{C_\mu k^2}{\epsilon} \tag{C.40}$$

Turbulence kinetic energy:

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = \tau_{ij} \frac{\partial U_i}{\partial x_j} - \epsilon + \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_T}{\sigma_k}\right) \frac{\partial k}{\partial x_j}\right] \tag{C.41}$$

Dissipation rate:

$$\frac{\partial \epsilon}{\partial t} + U_j \frac{\partial \epsilon}{\partial x_j} = C_{\epsilon 1} \frac{\epsilon}{k} \tau_{ij} \frac{\partial U_i}{\partial x_j} - C_{\epsilon 2} \frac{\epsilon^2}{k} + \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_T}{\sigma_\epsilon}\right) \frac{\partial \epsilon}{\partial x_j}\right] \tag{C.42}$$

Closure coefficients and auxiliary relations:

$$\begin{aligned} &C_{\epsilon 1} = 1.44, \qquad C_{\epsilon 2} = 1.92, \qquad C_\mu = 0.09, \qquad \sigma_k = 1.0, \qquad \sigma_\epsilon = 1.3 \\ &\omega = \frac{\epsilon}{C_\mu k}, \qquad l = \frac{C_\mu k^{3/2}}{\epsilon} \end{aligned} \tag{C.43}$$

C.2.4 The SST Model

The appendix about the Shear-Stress Transport (SST) model is based on [Menter 1993]. The SST model is a modification of the new Baseline model (BSL), and before explaining the SST model, the BSL model will be described.

The BSL Model

The concept of the BSL model is to counteract some of the shortcomings of the $k-\epsilon$ and the $k-\omega$ models, by introducing a blending function, F_1, which ensures the use of the $k-\omega$ model in the inner half of the boundary layer ($\frac{\delta}{2}$) and then gradually changes to the $k-\epsilon$ model in the outer wake region. The reason for this blending is further described in the main report.

In order to utilize the blending function, both models are written in a similar $k-\omega$ formulation below. In the formulations the substantial derivative, $\frac{D}{Dt} = \frac{\partial}{\partial t} + u_i \frac{\partial}{\partial x_i}$, is used.

The original $k-\omega$ model:

$$\frac{D\rho k}{Dt} = \tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta^*\rho\omega k + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_{k1}\mu_T)\frac{\partial k}{\partial x_j}\right] \tag{C.44}$$

$$\frac{D\rho\omega}{Dt} = \frac{\gamma_1}{\nu_T}\tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta_1\rho\omega^2 + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_{\omega 1}\mu_T)\frac{\partial \omega}{\partial x_j}\right] \tag{C.45}$$

The transformed $k-\epsilon$ model:

$$\frac{D\rho k}{Dt} = \tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta^*\rho\omega k + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_{k2}\mu_T)\frac{\partial k}{\partial x_j}\right] \tag{C.46}$$

$$\frac{D\rho\omega}{Dt} = \frac{\gamma_2}{\nu_T}\tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta_2\rho\omega^2 + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_{\omega 2}\mu_T)\frac{\partial \omega}{\partial x_j}\right] + 2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \tag{C.47}$$

Eqs. (C.44) and (C.45) are then multiplied by F_1 while eqs. (C.46) and (C.47) are multiplied by $(1-F_1)$:

$$\frac{D\rho k}{Dt} = \tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta^*\rho\omega k + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_k\mu_T)\frac{\partial k}{\partial x_j}\right] \tag{C.48}$$

$$\begin{aligned}\frac{D\rho\omega}{Dt} = \frac{\gamma}{\nu_T}\tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta\rho\omega^2 + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_\omega\mu_T)\frac{\partial \omega}{\partial x_j}\right] \\ +2\rho(1-F_1)\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}\end{aligned} \tag{C.49}$$

Here ϕ_1 represents any constant in the original $k-\omega$ model and ϕ_2 represents any constant in the transformed $k-\epsilon$ model so that the constants used in eqs. (C.48) and (C.49) are given by $\phi = F_1\phi_1 + (1-F_1)\phi_2$. The constants used, are given below.

Set 1 (ϕ_1) (*Wilcox's $k-\omega$ model*):

$$\begin{aligned} &\sigma_{k1} = 0.5, \qquad \sigma_{\omega 1} = 0.5, \qquad \beta_1 = 0.0750 \\ &\beta^* = 0.09, \qquad \kappa = 0.41, \qquad \gamma_1 = \frac{\beta_1}{\beta^*} - \frac{\sigma_{\omega 1}\kappa^2}{\sqrt{\beta^*}} \end{aligned} \tag{C.50}$$

Set 2 (ϕ_2) (*Standard $k-\epsilon$*):

$$\begin{aligned} &\sigma_{k2} = 1.0, \qquad \sigma_{\omega 2} = 0.856, \qquad \beta_2 = 0.0828 \\ &\beta^* = 0.09, \qquad \kappa = 0.41, \qquad \gamma_1 = \frac{\beta_2}{\beta^*} - \frac{\sigma_{\omega 2}\kappa^2}{\sqrt{\beta^*}} \end{aligned} \tag{C.51}$$

Set 1 corresponds to the original $k-\omega$ model and will be used in the near wall region only. Set 2 corresponds to the transformation of the standard $k-\epsilon$ model, with $C_{1\epsilon} = 1.44$ and $C_{2\epsilon} = 1.92$, cf. eq. (C.43), and will be used mainly in the free shear layers.

The model has to be supplemented by a definition of the kinematic eddy viscosity:

$$\nu_T = \frac{\mu_T}{\rho} = \frac{k}{\omega} \tag{C.52}$$

The turbulent stress tensor is then given as:

$$\tau_{ij} = \mu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij} \right) - \frac{2}{3}\rho k \delta_{ij} \tag{C.53}$$

Finally, in order to complete the model, F_1 needs to be defined. The function will be defined in terms of the variable arg_1

$$arg_1 = \min\left(\max\left(\frac{\sqrt{k}}{0.09\omega y}; \frac{500}{y^2\omega} \right); \frac{4\rho\sigma_{\omega 2}k}{CD_{k\omega}y^2} \right) \tag{C.54}$$

as follows:

$$F_1 = \tanh(arg_1^4) \tag{C.55}$$

where $CD_{k\omega}$ is the cross-diffusion term of eq. (C.47):

$$C\,D_{k\omega} = \max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}; 10^{-20}\right) \tag{C.56}$$

The first argument in eq. (C.54) is the turbulent length scale $L_T = \frac{\sqrt{k}}{0.09\omega} = \frac{k^{3/2}}{\epsilon}$ divided by the wall distance, y. The ratio $\frac{L_T}{y}$ is equal to 2.5 in the logarithmic region of the boundary layer and goes to zero towards the boundary layer edge. The second argument in eq. (C.54) ensures that F_1 does not go to zero in the viscous sublayer. The third argument in eq. (C.54) is an additional safeguard against the "degenerate" solution of the original $k-\omega$ model with small free stream values.

The SST Model

One of the major differences between eddy-viscosity and full Reynolds-Stress models is that the latter accounts for the important effects of the transport of the principal turbulent shear-stress $\tau = -\rho\overline{u'v'}$ by inclusion of the term:

$$\frac{D\tau}{Dt} = \frac{\partial \tau}{\partial t} + \mu_k \frac{\partial \tau}{\partial x_k} \tag{C.57}$$

In two equation models, the turbulent shear-stress, τ, is computed from:

$$\tau = \mu_T \Omega \tag{C.58}$$

with $\Omega = \frac{\partial u}{\partial y}$, whereas based on Bradshaws assumption, it is proportional to the turbulence kinetic energy, k:

$$\tau = \rho a_1 k \tag{C.59}$$

For conventional two-equation models this relation can be written as:

$$\tau = \rho\sqrt{\frac{\text{production of } k}{\text{dissipation of } k}}\, a_1 k \tag{C.60}$$

In adverse pressure gradient flows the ratio between production and dissipation of turbulence kinetic energy can be significantly larger than one, which leads to an over prediction of τ, using eq. (C.60). In order to satisfy eq. (C.59) in a two-equation model, the kinematic eddy-viscosity would have to be redefined as:

$$\nu_T = \frac{a_1 k}{\Omega} \tag{C.61}$$

The reason for this modification is as follows: In conventional two-equation models, the turbulent shear stress, τ, responds instantly to changes in the shear-strain rate, Ω, whereas eq. (C.61) ensures that τ does not change any faster than $\rho a_1 k$. Eq. (C.61) is however not desirable for the complete flow field, as it would lead to infinitely high eddy viscosities at points where Ω goes towards zero. In most cases with adverse pressure gradient flows, the production is larger than dissipation for the largest part of the boundary layer ($\Omega > a_1\omega$). The following equation ensures the use of eq. (C.61) for the most of the adverse pressure gradient regions, and eq. (C.52) is used for rest of the boundary layer.

In order to recover the original formulation of the eddy viscosity for free shear-layers (where Bradshaws assumption, eq. (C.59), may not hold), the modification to the SST model must be limited to wall bounded flows. This is done by introducing a blending function F_2:

$$\nu_T = \frac{a_1 k}{\max\left(a_1\omega; \Omega F_2\right)}] \tag{C.62}$$

where F_2 is defined as:

$$\begin{aligned} arg_2 &= \max\left(2\frac{\sqrt{k}}{0.09 a_1 \omega}; \frac{500\nu}{y^2\omega}\right) \\ F_2 &= \tanh(arg_2^2) \end{aligned} \tag{C.63}$$

Since the modification to the eddy viscosity has the largest impact in the wake region of the boundary layer, it is imperative that F_2 extends further out into the boundary layer than F_1. Similarly, F_1 must reach zero within the boundary layer in order to prevent the freestream dependence inherent to the $k-\omega$ model.

The modifications of the eddy viscosity in the SST model, given in eqs. (C.62) and (C.63), are used in conjunction with the BSL model, where the constants of set 1, eq. (C.50), are slightly adjusted in order to recover the correct behavior for a flat plate boundary layer:

Set 1 (SST - inner):

$$\begin{aligned} &\sigma_{k1} = 0.85, \qquad \sigma_{\omega 1} = 0.5, \qquad \beta_1 = 0.0750, \qquad a_1 = 0.31 \\ &\beta^* = 0.09, \qquad \kappa = 0.41, \qquad \gamma_1 = \frac{\beta_1}{\beta^*} - \frac{\sigma_{\omega 1}\kappa^2}{\sqrt{\beta^*}} \end{aligned} \tag{C.64}$$

Set 2 remains the same as in the BSL model. Furthermore, Ω is taken to be the absolute value of the vorticity for general flows.

C.3 Finite Volume Method

The sappendix about the finite volume method is based on [Versteeg & Malalasekera 1995, pp. 85-102]. The finite volume method for 3D problems can be derived by considering simple diffusion in steady state. The governing equation for the property ϕ becomes:

$$\frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(\Gamma\frac{\partial\phi}{\partial y}\right)+\frac{\partial}{\partial z}\left(\Gamma\frac{\partial\phi}{\partial z}\right)+S=0 \tag{C.65}$$

where

S is a source term
Γ is a diffusion coefficient

Eq. (C.65) is derived from the general transport equation, where the transient and convective terms are omitted. The general transport equation is given as:

$$\frac{\partial(\rho\phi)}{\partial t}+\mathrm{div}\,(\rho\phi\mathbf{u})=\mathrm{div}\,(\Gamma\mathrm{grad}\phi)+S_\phi \tag{C.66}$$

A three-dimensional grid is used to subdivide the domain into control volumes. A typical control volume (CV) is shown in fig. C.1.

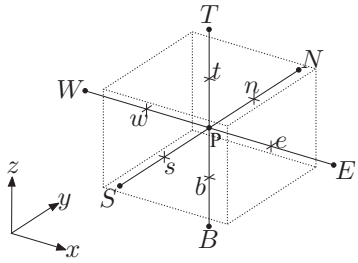


Figure C.1: Definition sketch of a typical control volume in three dimensions with neighboring nodes.

A control volume containing the node P has six neighboring nodes; north, south, east, west, top, and bottom (N, S, E, W, T, and B). The cell faces are named after the adjoining nodes as n, s, e, w, t, and b. Integration of eq. (C.65) over

the control volume shown in fig. C.1 yields:

$$\begin{aligned}
&\left[\Gamma_e A_e \left(\frac{\partial\phi}{\partial x}\right)_e - \Gamma_w A_w \left(\frac{\partial\phi}{\partial x}\right)_w\right] \\
&+\left[\Gamma_n A_n \left(\frac{\partial\phi}{\partial y}\right)_n - \Gamma_s A_s \left(\frac{\partial\phi}{\partial y}\right)_s\right] \\
&+\left[\Gamma_t A_t \left(\frac{\partial\phi}{\partial z}\right)_t - \Gamma_b A_b \left(\frac{\partial\phi}{\partial z}\right)_b\right] \\
&+\overline{S}\Delta V = 0
\end{aligned} \tag{C.67}$$

Here the *Divergence Theorem (Gauss's Theorem)*, eq. (C.68), has been used to change the volume integral into an area integral.

$$\int_V \mathrm{div}(\mathbf{F}) dV = \int_A \mathbf{n}\cdot\mathbf{F} dA \tag{C.68}$$

Furthermore it is assumed that all boundaries have normals in the global coordinate systems directions, so that the flux $\frac{\partial\phi}{\partial x_i}$ of the property ϕ over a boundary can be described by one of three components: $\frac{\partial\phi}{\partial x}$, $\frac{\partial\phi}{\partial y}$, or $\frac{\partial\phi}{\partial z}$. In the case that the boundary normals are not aligned with the directions of the global coordinate system, additional terms will appear in eq. (C.67) taking into account the decomposition of the flow over the boundaries into the three general directions.

Using a central differencing scheme, and approximating the source term by the linear form $\overline{S}\Delta V = S_u + S_P\phi_P$, the discretised form of eq. (C.67) yields:

$$\begin{aligned}
&\left[\Gamma_e \frac{(\phi_E - \phi_P) A_e}{\delta x_{PE}} - \Gamma_w \frac{(\phi_P - \phi_W) A_w}{\delta x_{WP}}\right] \\
&+\left[\Gamma_n \frac{(\phi_N - \phi_P) A_n}{\delta y_{PN}} - \Gamma_s \frac{(\phi_P - \phi_S) A_s}{\delta y_{SP}}\right] \\
&+\left[\Gamma_t \frac{(\phi_T - \phi_P) A_t}{\delta z_{PT}} - \Gamma_b \frac{(\phi_P - \phi_B) A_b}{\delta z_{BP}}\right] \\
&+(S_u + S_P\phi_P) = 0
\end{aligned} \tag{C.69}$$

Eq. (C.69) can be rearranged to give the discretised equation for interior nodes:

$$a_P\phi_P = a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + a_T\phi_T + a_B\phi_B + S_u \tag{C.70}$$

where the coefficients are given as:

$$\begin{aligned}
a_E &= \frac{\Gamma_e A_e}{\delta x_{PE}}, \qquad a_W = \frac{\Gamma_w A_w}{\delta x_{WP}}, \qquad a_N = \frac{\Gamma_n A_n}{\delta y_{PN}} \\
a_S &= \frac{\Gamma_s A_s}{\delta y_{SP}}, \qquad a_T = \frac{\Gamma_t A_t}{\delta z_{PT}}, \qquad a_B = \frac{\Gamma_b A_b}{\delta z_{TP}} \\
a_P &= a_E + a_W + a_N + a_S + a_T + a_B - S_P
\end{aligned} \tag{C.71}$$

Boundary conditions can be incorporated by cutting links with the appropriate faces (setting the coefficient to the appropriate face equal to zero), and introducing the boundary side flux, either exact or as a linear approximation through additional source terms.

The theory for the finite volume method for diffusion problems can be extended to cover more complex problems, following the approach given in this appendix. If more complex problems are being solved it will result in additional terms in the equations, but the principal setup will remain the same.

D

Structural Dynamics, Theory

In dynamic analysis the computation time is often considerably longer than for a pure static analysis of the same problem. Therefore it can be helpful to reduce the number of DOF and hereby the matrices being manipulated. Reduction of the system can be done in several ways. In this project a modal representation is used for the reduced system. In this appendix the modal method is described. Unless otherwise noted, the theory in this appendix is based on [Nielsen 2004].

Modal methods use an alternative set of DOF and solving for these DOF as a function of time. Last the reduced DOF is transformed back into the original DOF.

D.1 General Dynamic Equations

The equation of motion for a MDOF system is generally given by:

$$\begin{aligned} &\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f} \quad , \quad t > t_0 \\ &x(t_0) = x_0 \quad , \quad \dot{x}(t_0) = \dot{x}_0 \end{aligned} \tag{D.1}$$

Eq. (D.1) states an initial value problem for the motion of the MDOF system. The undamped eigenmodes for the system are determined by solving the following generalized eigenvalue problem:

$$\left(\mathbf{K} - \omega^2\mathbf{M}\right)\mathbf{\Phi} = \mathbf{0} \tag{D.2}$$

where

ω is the undamped circular eigenfrequency
$\mathbf{\Phi}$ is the undamped eigenmode

Non-trivial solutions, $\mathbf{\Phi} \neq \mathbf{0}$, to eq. (D.2) is obtained by requiring:

$$\det\left(\mathbf{K} - \omega^2\mathbf{M}\right) = \mathbf{0} \tag{D.3}$$

Solutions to eq. (D.3) yield n eigenvalues which by insertion in eq. (D.2) gives n eigenmodes. If the eigenmodes, $\mathbf{\Phi}^{(i)}$ and $\mathbf{\Phi}^{(j)}$, are associated with different eigenvalues, ω_i^2 and ω_j^2, where $\omega_i^2 \neq \omega_j^2$, the eigenmodes fulfil the orthogonality condition:

$$\mathbf{\Phi}^{(i)T}\mathbf{M}\mathbf{\Phi}^{(j)} = \begin{cases} 0 & \text{for} \quad i \neq j \\ M_i & \text{for} \quad i = j \end{cases} \tag{D.4}$$

$$\mathbf{\Phi}^{(i)T}\mathbf{K}\mathbf{\Phi}^{(j)} = \begin{cases} 0 & \text{for} \quad i \neq j \\ \omega_i^2 M_i & \text{for} \quad i = j \end{cases} \tag{D.5}$$

The parameter M_i is denoted the undamped modal mass and is given by:

$$M_i = \mathbf{\Phi}^{(i)T}\mathbf{M}\mathbf{\Phi}^{(i)} \quad \text{for} \quad i = 1, \ldots, n \tag{D.6}$$

In most cases, the eigenmodes are normalized to the mass matrix which means that $M_i = 1$ for $i = 1, \ldots, n$.

D.2 Modal Equations

The orthogonality condition in eq. (D.4) can be written on matrix form in the following way:

$$\begin{bmatrix} \Phi^{(1)T} \\ \vdots \\ \Phi^{(n)T} \end{bmatrix} \mathbf{M} \begin{bmatrix} \Phi^{(1)} & \cdots & \Phi^{(n)} \end{bmatrix} = \begin{bmatrix} M_1 & 0 & \cdots & 0 \\ 0 & M_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & M_n \end{bmatrix} \tag{D.7}$$

$$\mathbf{P}^T\mathbf{M}\mathbf{P} = \mathbf{m}$$

where

$\mathbf{P}$ is the modal matrix
$\mathbf{m}$ is the modal mass matrix

A similar relation as eq. (D.7) holds for the stiffness matrix:

$$\mathbf{P}^T\mathbf{K}\mathbf{P} = \mathbf{k} \quad , \quad \mathbf{k} = \begin{bmatrix} M_1\omega_1^2 & 0 & \cdots & 0 \\ 0 & M_2\omega_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & M_n\omega_n^2 \end{bmatrix} \tag{D.8}$$

where

$\mathbf{k}$ is the modal stiffness matrix

As the eigenvectors are linearly independent, an arbitrary displacement vector, $\mathbf{x}(t)$, can be expressed as a linear combination of the eigenvectors which can be written as:

$$\begin{aligned} \mathbf{x}(t) &= q_1(t)\mathbf{\Phi}^{(1)} + q_2(t)\mathbf{\Phi}^{(2)} + \cdots + q_n(t)\mathbf{\Phi}^{(n)} \\ &= \mathbf{P}\mathbf{q}(t) \end{aligned} \tag{D.9}$$

where

$q_n(t)$ is the undamped modal coordinate

Eq. (D.9) express the transition between the cartesian and the modal coordinate system. Also eq. (D.9) is valid for the velocity and acceleration vectors. Including these yield:

$$\mathbf{x}(t) = \mathbf{P}\mathbf{q}(t) \quad , \quad \dot{\mathbf{x}}(t) = \mathbf{P}\dot{\mathbf{q}}(t) \quad , \quad \ddot{\mathbf{x}}(t) = \mathbf{P}\ddot{\mathbf{q}}(t) \tag{D.10}$$

Inserting eq. (D.9) into (D.1) and multiplying by $\mathbf{\Phi}^{(i)T}$ yields:

$$\sum_{j=1}^{n} \left(\mathbf{\Phi}^{(i)T}\mathbf{M}\mathbf{\Phi}^{(j)}\ddot{q}_j + \mathbf{\Phi}^{(i)T}\mathbf{C}\mathbf{\Phi}^{(j)}\dot{q}_j + \mathbf{\Phi}^{(i)T}\mathbf{K}\mathbf{\Phi}^{(j)}q_j \right) = \mathbf{\Phi}^{(i)T}\mathbf{f}(t) \tag{D.11}$$

Using the orthogonality conditions in eqs. (D.4) and (D.5) and the relation in (D.6), eq. (D.11) can be written as.

$$\begin{aligned} &\ddot{q}_i + \frac{1}{M_i}\sum_{j=1}^{n} \left(\mathbf{\Phi}^{(i)T}\mathbf{C}\mathbf{\Phi}^{(j)}\dot{q}_j \right) + \omega_i^2 q_i = \frac{1}{M_i}\mathbf{\Phi}^{(i)T}\mathbf{f}(t) \\ &\text{for} \quad i = 1, \ldots, n \quad , \quad t > 0 \end{aligned} \tag{D.12}$$

Introducing modal damping given by:

$$\mathbf{\Phi}^{(i)T}\mathbf{C}\mathbf{\Phi}^{(j)} = \begin{cases} 0 & \text{for} \quad i \neq j \\ 2\zeta_i\omega_i M_i & \text{for} \quad i = j \end{cases}, \tag{D.13}$$

Eq. (D.12) can be written as:

$$\ddot{q}_i + 2\zeta_i\omega_i\dot{q}_i + \omega_i^2 q_i = \frac{1}{M_i}\mathbf{\Phi}^{(i)T}\mathbf{f}(t) \quad , \quad i = 1, \ldots, n \quad , \quad t > 0 \tag{D.14}$$

where

ζ_i is the modal damping ratio

In many cases the eigenmodes for the structure are normalized to unit modal mass i.e $\mathbf{m} = \mathbf{I}$. In this case $M_i = 1$ which makes the term $\frac{1}{M_i}$ on the right-hand side of eq. (D.14) disappear.

Eq. (D.14) is solved for the n modal coordinates, q_i and eq. (D.9) is used to make the transition to the cartesian coordinate system.

D.3 Reduction of DOF

If all n eigenmodes are used in the calculations eq. (D.14) is a mathematically exact representation of eq. (D.1). Modal analysis on the other hand benefits from the fact that in many practical problems only the lowest few eigenfrequencies need to be included. This means that the number of DOF in modal analysis is significantly lower. Instead of being $n \times n$, the modal matrix, $\mathbf{P}$, becomes an $n \times m$ matrix, where $m \ll n$ in many cases. m corresponds to the number of eigenmodes retained in the calculations.

The reduced set of eigenmodes must include all lower modes up to a chosen frequency. There is no distinct definition of how this frequency must be chosen. The more abrupt in time and irregular in space the loading becomes, the more modes are needed. A guideline for the necessary frequency is double the highest important frequency contained in the loading [Cook, Markus, Plesha & Witt 2002, p. 397].

D.4 Newmark Algorithm

A widely used algorithm for solving the initial value problem in eq. (D.1) is the Newmark Algorithm. In this appendix the theory behind this algorithm will be presented. The theory is based on [Nielsen 2005, pp. 32-34].

The Newmark Algorithm is based on the following equations, which is a description of eq. (D.1) at time step t_{j+1}:

$$\mathbf{M}\ddot{\mathbf{x}}_{j+1} + \mathbf{C}\dot{\mathbf{x}}_{j+1} + \mathbf{K}\mathbf{x}_{j+1} = \mathbf{f}_{j+1} \quad , \quad j = 1, 2, \ldots, n \tag{D.15}$$

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \dot{\mathbf{x}}_j \Delta t + \left(\left(\frac{1}{2} - \beta \right) \ddot{\mathbf{x}}_j + \beta \ddot{\mathbf{x}}_{j+1} \right) \Delta t^2 \tag{D.16}$$

$$\dot{\mathbf{x}}_{j+1} = \dot{\mathbf{x}}_j + \left((1-\gamma)\ddot{\mathbf{x}}_j + \gamma \ddot{\mathbf{x}}_{j+1} \right) \Delta t \tag{D.17}$$

where

$\beta,\ \gamma$ are constants in the algorithm, more about these later

As mentioned, eq. (D.15) represents the differential equation at time t_{j+1} which is required to be fulfilled for the solution consisting of $\ddot{\mathbf{x}}_{j+1}$, $\dot{\mathbf{x}}_{j+1}$ and $\mathbf{x}_{j+1}$. By using integration theory the displacement and velocity at time t_{j+1} can be written as:

$$\begin{aligned} \mathbf{x}(t_{j+1}) &= \mathbf{x}(t_j) + \int_{t_j}^{t_{j+1}} \dot{\mathbf{x}}(\tau)\, d\tau \\ \dot{\mathbf{x}}_(tj+1) &= \dot{\mathbf{x}}(t_j) + \int_{t_j}^{t_{j+1}} \ddot{\mathbf{x}}(\tau)\, d\tau \end{aligned} \tag{D.18}$$

Considering only the first equation in eq. (D.18), integration by parts yields:

$$\mathbf{x}(t_{j+1}) = \mathbf{x}(t_j) - \Big[(t_{j+1} - \tau)\dot{\mathbf{x}}(\tau) \Big]_{t_j}^{t_{j+1}} + \int_{t_j}^{t_{j+1}} (t_{j+1} - \tau)\ddot{\mathbf{x}}(\tau)\, d\tau \quad \Rightarrow$$

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta t \dot{\mathbf{x}}_j + \int_{t_j}^{t_{j+1}} (t_{j+1} - \tau)\ddot{\mathbf{x}}(\tau)\, d\tau \tag{D.19}$$

The representation in eq. (D.19) can be interpreted as a truncated Taylor expansion where the integral represent a remainder in the expansion. In a similar manner the second equation of (D.18) can be written as:

$$\dot{\mathbf{x}}_{j+1} = \dot{\mathbf{x}}_j + \int_{t_j}^{t_{j+1}} \ddot{\mathbf{x}}(\tau)\, d\tau \tag{D.20}$$

The integrals in eq. (D.19) and (D.20) can be represented by the following linear combinations of the acceleration vector at the end of the time interval:

$$\begin{aligned}\int_{t_j}^{t_{j+1}} (t_{j+1}-\tau)\ddot{\mathbf{x}}(\tau)\,d\tau &\simeq \left(\frac{1}{2}-\beta\right)\Delta t^2\ddot{\mathbf{x}}_j + \beta\Delta t^2\ddot{\mathbf{x}}_{j+1} \\ \int_{t_j}^{t_{j+1}} \ddot{\mathbf{x}}(\tau)\,d\tau &\simeq (1-\gamma)\Delta t\ddot{\mathbf{x}}_j + \gamma\Delta t\ddot{\mathbf{x}}_{j+1}\end{aligned} \tag{D.21}$$

It is seen from eq. (D.21) that the result becomes correct if and only if the acceleration during the time interval is constant, i.e. $\ddot{\mathbf{x}}(\tau) \equiv \ddot{\mathbf{x}}_j = \ddot{\mathbf{x}}_{j+1}$. This means that the parameters β and γ express information about the actual variation of the acceleration during the time interval. Assuming $\ddot{\mathbf{x}}(\tau)$ constant and equal to the mean value of the end-point values, this yields $(\beta,\gamma) = \left(\frac{1}{4},\frac{1}{2}\right)$ while assuming a linear variation in the interval yields $(\beta,\gamma) = \left(\frac{1}{6},\frac{1}{2}\right)$. β and γ determines the numerical stability and accuracy of the algorithm.

Insertion of the integral representation in eq. (D.21) into eqs. (D.19) and (D.20) respectively results in the two equations postulated in eqs. (D.16) and (D.17) in the beginning of this appendix.

Eqs. (D.15)-(D.17) states the full family of Newmark Algorithms whereto several special cases exist. The most useful is a single step value implementation. This method starts with a calculation of *predictors* of the displacement and velocity at the next time step. The predictors are determined by:

$$\begin{aligned}\bar{\mathbf{x}}_{j+1} &= \mathbf{x}_j + \dot{\mathbf{x}}_j\Delta t + \left(\frac{1}{2}-\beta\right)\Delta t^2\ddot{\mathbf{x}}_j \\ \dot{\bar{\mathbf{x}}}_{j+1} &= \dot{\mathbf{x}}_j + (1-\gamma)\,\Delta t\ddot{\mathbf{x}}_j\end{aligned} \tag{D.22}$$

In words, eq. (D.22) yields a preliminary prediction of the displacement and velocity at time step t_{j+1} based on the acceleration, velocity and displacement at the previous time step, t_j, and the assumed variation of the acceleration represented by β and γ.

The idea of the algorithm is to insert eqs. (D.16) and (D.17), in this special case eq. (D.22), into (D.15) as it is required to fulfill the solution at time t_{j+1}. The new acceleration vector is given by the following based on information from the previous time step (represented by the predictors) and the load vector at the present time step, $\mathbf{f}_{j+1}$:

$$\left(\mathbf{M} + \gamma\Delta t\mathbf{C} + \beta\Delta t^2\mathbf{K}\right)\ddot{\mathbf{x}}_{j+1} = \mathbf{f}_{j+1} - \mathbf{C}\dot{\bar{\mathbf{x}}}_{j+1} - \mathbf{K}\bar{\mathbf{x}}_{j+1} \tag{D.23}$$

Eq. (D.23) is solved for the new acceleration, $\ddot{\mathbf{x}}_{j+1}$ and *corrected* displacement and velocity vectors are determined by:

$$\begin{aligned} \mathbf{x}_{j+1} &= \bar{\mathbf{x}}_{j+1} + \beta \Delta t^2 \ddot{\mathbf{x}}_{j+1} \\ \dot{\mathbf{x}}_{j+1} &= \dot{\bar{\mathbf{x}}}_{j+1} + \gamma \Delta t \ddot{\mathbf{x}}_{j+1} \end{aligned} \tag{D.24}$$

To start the algorithm the initial acceleration must be determined on the basis of the supplied initial conditions for the displacement, velocity and load. The acceleration at $t = 0$ is determined by the equation of motion:

$$\mathbf{M}\ddot{\mathbf{x}}_0 = \mathbf{f}_0 - \mathbf{C}\dot{\mathbf{x}}_0 - \mathbf{K}\mathbf{x}_0 \tag{D.25}$$

The single step single value implementation method of the Newmark family can summarized as follows:

1. Define the initial conditions for the displacement $\mathbf{x}_0$, velocity, $\dot{\mathbf{x}}_0$, and load, $\mathbf{f}_0$ at time $t = 0$. Calculate the initial acceleration vector, $\ddot{\mathbf{x}}_0$ by eq. (D.25)

2. Calculate predictors of the new displacement, $\bar{\mathbf{x}}_{j+1}$, and velocity, $\dot{\bar{\mathbf{x}}}_{j+1}$, by eq. (D.22)

3. Calculate the new acceleration vector, $\ddot{\mathbf{x}}_{j+1}$, by eq. (D.23)

4. Calculate new corrected displacement, $\mathbf{x}_{j+1}$, and velocity, $\dot{\mathbf{x}}_{j+1}$, vectors by eq. (D.24).

5. Repeat step 2-4 for $j = 0, 2, \ldots, n$, where n is the number of time steps wanted.

The Newmark Algorithm described above can just as well be used in the modal coordinate system. This is because eq. (D.9) express a one-to-one transformation between the physical and the modal coordinates. In this project the Newmark Algorithm have been used on the modal coordinates.

E

Ansys CFX Memory Management System

When using Fortran routines during transient runs, the need for accessing and storing data from one time step to the next may arise. To that purpose, CFX incorporates a Memory Management System (MMS) to use with Fortran. This appendix will describe some of the most useful routines and methods when using MMS. For further information about the MMS and specific routines, see *ANSYS CFX-Solver Modelling Guide pp. 478-495.*

The basic concept of the MMS is, that all data is stored in 5 different stacks, listed in tab E.1. These stacks are internal in CFX and their contents can be changed, but they cannot be renamed.

Table E.1: Stacks in MMS

Stack	Type
`CZ`	character
`DZ`	double precision
`IZ`	integer
`LZ`	logical
`RZ`	real

Each stack holds all data of the given type, and can only store values of the given type. The stacks are all one dimensional arrays, and are accessed using *stack*

pointers. A stack pointer is essentially an integer, giving the position of a given data area on a stack. The suggested naming convention is, that a stack pointer to a variable VARNAME is named pVARNAME. The stack pointers are declared of the type __stack_point__ along with the normal variable definitions.
As an example, if a 3×3 array A of the type REAL is stored on the stack RZ at position pA, then RZ(pA:pA+8) will contain the array A stored one row at the time as follows: RZ(pA:pA+2)=A(1,1:3), RZ(pA+3:pA+5)=A(2,1:3), and RZ(pA+6:pA+8)=A(3,1:3).

E.1 Allocating Space on the Stacks

To allocate space on one of the stacks, a call to MAKDAT is used:

```
CALL MAKDAT( CDANAM, CDTYPE, CERACT , ISIZE , pPOINT , CRESLT)
```

Here CDANAM is the name of the data area e.g 'NBEAM', CDTYPE is the type of the data e.g. 'INTR', CERACT is the action to be taken if the call produces errors e.g. 'STOP', ISIZE is the size of the data area e.g. 9 for a 3×3 array, pPOINT is the stack pointer (integer) as explained above e.g. 2178462, and CRESLT is a text string showing the result of the call e.g. 'GOOD'.

E.2 Writing Data onto Allocated Space on the Stacks

In order to write data to the allocated space on the stacks, the stack pointer is needed. Below is an example where a 3×2 matrix A of the type INTR is loaded onto the stack IZ using the stack pointer pA:

```
DO I=1,3
  DO J=1,2
    IZ(pA+(I-1)*2+J-1) = A(I,J)
  END DO
END DO
```

It is important to notice here, that it is possible to write more data to the stacks than space has been allocated for. This will result in writing data to an area which might be allocated for a different purpose. Thus, it is of **paramount importance** to make sure that the correct space has been allocated.

Special POKE routines exist to set values at a single address on a stack, POKER for real values, POKEI for integer values etc.

Below an example is given, if a REAL entry is to be altered:

```
CALL POKER (CDANAM, JADRES, RVALUE, CERACT, CRESLT, RZ)
```

Here CDANAM is the name of the data area (the name used when the space was allocated), JADRES is the position on the data area e.g. on a 9 byte data area JADRES can be 1-9, RVALUE is the value to be written to the specified address, CERACT is the action to be taken if the call produces an error, CRESLT is a text string showing the result of the call and RZ is the appropriate stack to be written on.

E.3 Reading Data from Allocated Space on the Stacks

In order to write data to the allocated space on the stacks, the stack pointer is needed. The data is read in a similar manner as it is written. Below is an example where a 2×5 matrix B of the type REAL is read from the stack RZ using the stack pointer pB:

```
DO I=1,2
  DO J=1,5
    B(I,J) = RZ(pB+(I-1)*5+J-1)
  END DO
END DO
```

Special PEEK routines exist to read values at a single address on a stack. Below an example is given, if an INTR entry is to be read:

```
CALL PEEKI (CDANAM, JADRES, IVALUE, CERACT, CRESLT, IZ)
```

Here CDANAM is the name of the data area (the name used when the space was allocated), JADRES is the position on the data area e.g. on a 9 byte data area JADRES can be 1-9, IVALUE is the variable to which the data is written, CERACT is the action to be taken if the call produces an error, CRESLT is a text string showing the result of the call and IZ is the appropriate stack to be read from.

E.4 Locating Data on the Stacks

Data on the stacks can be accessed from other routines or time steps than the ones in which they were created. To do this, the data needs to be located using LOCDAT. Below is an example:

```
CALL LOCDAT(CDANAM, CDTYPE, CERACT, ISIZE, JADRES, CRESLT)
```

Here CDANAM is the name of the data area (the name used when the space was allocated) e.g. 'NBEAM' or '/USER_DATA/NBEAM', CDTYPE returns the type of the area e.g. 'REAL' or 'CHAR', CERACT is the action to be taken if the call produces

an error, `ISIZE` returns the size of the data area e.g. 6 for a 2×3 array, `JADRES` returns an integer telling the position of the data area on the stack (essentially a stack pointer), and `CRESLT` is a text string showing the result of the call.

F

Moving Mesh in CFX

In order to model the aeroelastic response of a wind exposed high rise building, the ability to handle Fluid-Structure Interaction (FSI) in the CFD simulations is important. To facilitate this interaction, it must be possible to deform the structure geometry in a physically meaningful way, as a response to the wind loads.

In this appendix, a method for moving the mesh in Ansys CFX will be presented.

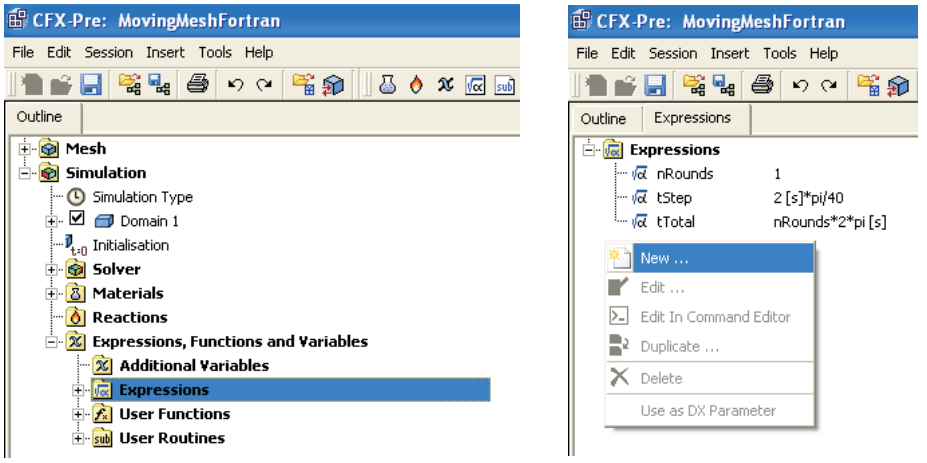
F.1 Setup

For the reason of simplicity, all the fluid related options will be neglected here and emphasis will be on the options related to mesh deformation only. In tab. F.1 the input for the simulation type are given.

The Total Time and Timesteps are defined as user expressions to allow easy access to these parameters in other expressions and functions. The expressions are set up by double clicking Expressions, as shown in fig. F.1a and right clicking, choosing New as shown in fig. F.1b. Once named, the expression can be anything from a scalar to a function of several variables (either user predefined expressions or internal variables).

*Table F.1: Simulation Type for mesh motion. * user expression*

Simulation Type	
Basic Settings	
External Solver Coupling	
- Option	None
Simulation Type	
- Option	Transient
Time Duration	
- Option	Total Time
- Total Time	tTotal*
Time Steps	
- Option	Timesteps
- Timesteps	tStep*
Initial Time	
- Option	Automatic with value
- Time	0 [s]



a) Expressions tab *b)* New expression

Figure F.1: Setting up user expression

In order to obtain data for each time step in the transient run, this needs to be defined. This is done by expanding Solver and entering the settings in tab. F.2 in Output Control, where the user defined expression tStep is used as time interval for logging data.

Table F.2: Output Control

Output Control Trn Results	
Transient Results	
- Option	New (input name)
Transient Results 1	
- Option	Selected Variables
- File Compression	Default
- Output Variables List	Mesh Displacement, Orthogonality Angle Minimum
- Include Mesh	Select
Output Frequency	
- Option	Time Interval
- Time Interval	tStep

Global Initialization is set up as normally.

F.1.1 Domain and Boundary Settings

To enable the mesh to move, double click Domain 1 (see fig. F.1a) and enter the input given in tab. F.3.

Table F.3: Domain Settings

Details of Domain 1 General Options	
Mesh Deformation	
- Option	Regions of Motion Specified

The mesh should be stationary on all external boundaries (inlets, outlets and openings). This is done by expanding Domain 1 (see fig. F.1a) and applying the following settings to each of these.

Table F.4: Inlet, Outlet and Opening Settings

Details of 'Boundary' in Domain 1 Boundary Details	
Mesh Motion	
- Option	Stationary

In a 2D simulation, the boundaries facing in and out of the plane should be set to symmetry plane due to computational reasons in CFX. These symmetry boundaries should not interfere with the mesh motion specified by other boundaries. This is ensured by expanding Domain 1 (see fig. F.1a) and applying the following settings to the symmetry boundaries.

Table F.5: Settings for Symmetry boundaries

Details of 'Boundary' in Domain 1 Boundary Details	
Mesh Motion	
- Option	Unspecified

Finally, the structure boundary should be able to move freely, according to input routines. This is done by entering the input given in tab. F.6. Note that dx, dy and dz are merely representative names of functions or expressions that govern mesh movement or deformation in that respective direction.

Table F.6: Structure Settings

Details of Structure in Domain 1 Boundary Details	
Wall Influence On Flow	
- Option	No Slip
- Wall Velocity Relative To	Select
- Wall Vel. Rel. To	Mesh Motion
Mesh Motion	
- Option	Specified Displacement
- X Component	dx(*input variables*)
- Y Component	dy(*input variables*)
- Z Component	dz(*input variables*) (0 [m] if 2D)

F.1.2 Setting up Fortran Routines

There are three steps to implementing a Fortran routine; Creating a User Routine in CFX that links to the Fortran routine, Creating a User Function in CFX that links to the User Routine, and finally creating and compiling the Fortran routine itself.

Firstly, a user routine has to be created - this is done by right clicking User Routines (see fig. F.1a) and choosing Insert → User Routine. Fig. F.2 shows the tab that opens and the input is given as shown in the figure. Note that calling name is the name of the Fortran routine without the file extension. The library name should be the same as the calling name. Also, it is a good idea to make these in all lower case letters to avoid problems. This is not a must however. The library path should direct to the path of the definition file and can be input either as a full path, or as a relative path.

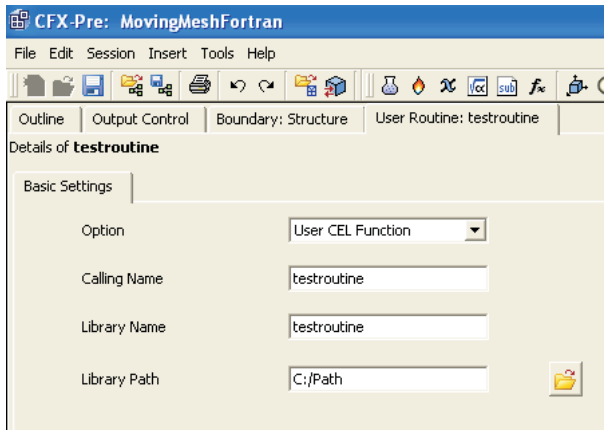


Figure F.2: Setting up user routine

After the user routine is set up, a corresponding user function should be created. In a similar manner to above, this is done by right clicking User Functions (see fig. F.1a) and choosing Insert → User Function. Fig. F.3 shows the input entered, where User Routine Name should point to the corresponding routine. Argument Units and Result Units should contain the units in which the input and output for the Fortran routine is given.

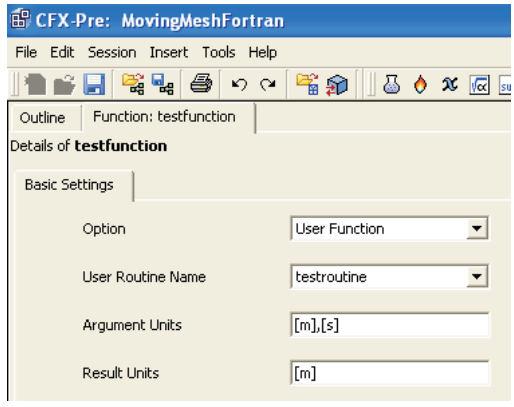


Figure F.3: Setting up user function

When creating a Fortran routine (or CEL function), CFX has a template ucf_template.F in the examples folder which is very useful, since it has the basic structure of a CEL function. The basic structure of a CEL function can be seen in the following:

```
#include "cfx5ext.h"
dllexport(<callingname>)
      SUBROUTINE <callingname>(
     &  NLOC, NRET, NARG, RET, ARGS, CRESLT, CZ,DZ,IZ,LZ,RZ )
C
      INTEGER NLOC,NARG,NRET
      CHARACTER CRESLT*(*)
      REAL ARGS(NLOC,NARG), RET(NLOC,NRET)
C
      INTEGER IZ(*)
      CHARACTER CZ(*)*(1)
      DOUBLE PRECISION DZ(*)
      LOGICAL LZ(*)
      REAL RZ(*)
C
      Executable statements...
C
      CRESLT = 'GOOD'
C
      END
```

The template makes use of mainly 5 variables; `NLOC`, `NRET`, `NARG`, `ARGS`, and `RET`. The input variables (defined in the call to the routine, see tab. F.6) are loaded into the matrix `ARGS(NLOC,NARG)`, and the output variables are saved to the matrix `RET(NLOC,NRET)`.
Here `NLOC` is the number of locales, i.e. when governing the movement of a boundary, `NLOC` will be the number of nodes on that boundary. `NARG` is the number of input arguments to the function, so if the function is dependant on time and x displacement, `NARG` would be two. `NRET` is the number of output variables, so if the output would be the x displacement, `NRET` would be one.

For instance, when governing the mesh stiffness in the domain, the input variable could be the wall distance for each element in the mesh. In that way `ARGS` would be a $n \times 1$ matrix, where n is the number of elements in the mesh, and each entry holds the wall distance for each element. Similarly `RET` would be a $n \times 1$ matrix where each entry holds the mesh stiffness for each element.

G
Macros in CFX

The use of macros in CFX can become useful when the user would like to make relatively quick changes in simulations where the normal way of doing the changes would take a considerable amount of time and work. This can be the case in both CFX-Pre when defining the simulation or in CFX-Post when processing the results. Macros can be used in both cases and an introduction to using macros in CFX-Pre and -Post are given in apps. G.1 - G.3.

Macros in CFX makes use of the programming language PERL. PERL was originally developed for text manipulation and is widely used in internet programming. PERL borrows features from many other programming languages including C. The macros can be written in any text editor. In this project Notepad has been used as editor for the macros.

G.1 General Features in CFX Macros

The beginning of the macro for both CFX-Pre and -Post contains a definition of the Graphical User Interface (GUI). This is done by writing the following lines at the beginning of the file:

```
# Macro GUI begin
#
# macro name =
# macro subroutine =
```

```
#
# Macro GUI end
```

What is to be put in between `Macro GUI begin` and `Macro GUI end` will be explained in apps. G.2 and G.3 as this differs in the two types of macros.

G.2 CFX-Pre Macro

The syntax of the macro for CFX-Pre is illustrated on the basis of a written macro which was intended for use in this project. Progress in the project showed, that only a portion of the macro was actually needed. The full initial macro, *CELexpressions.ccl* (note the file type), is included on the DVD in [*DVD:\CFX\Macros*] for learning purposes only. It is recommended that the reader opens the file to get the full context of the macro, as the macro is divided into pieces in this introduction.

The macro for CFX-Pre was programmed to automatically generate equivalent CEL expressions (CFX Expression Language) for a given number of sections of the structure used in the project. The GUI for this macro is as shown below:

```
# Macro GUI begin
#
# macro name = CELexpressions
# macro subroutine = CELexpressions
#
#
# Macro GUI end
```

The macro name is the one which is visible when the macro is opened in the macro calculator in CFX-Pre. This is shown in fig. G.1 which is a screen cap from the program.

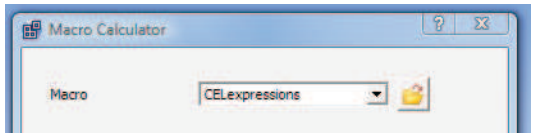


Figure G.1: Macro opened in macro calculator in CFX-Pre

The name of the subroutine that the macro calls is also defined in the GUI. The subroutine can be run directly from the command editor in CFX-Pre by entering

the subroutine name.

The next step in the macro is the subroutine mentioned. The subroutine is in this case written in the same file as the macro because only one subroutine is used. The subroutine is initiated and ended by the following command:

```
!  sub CELexpressions {
#
#
!  }
```

The routine is written in between the curly brackets. All arguments to a loop or statement must be enclosed in curly brackets. More about this later. It is seen that the statement where the subroutine is initiated and ended is started with a `!`. This is the way PERL knows that the line is a statement in the routine and all statements must be initiated with a `!`. Commenting lines in PERL are initiated by `#`. The function of this sign corresponds to `%` in MatLab and `C` in Fortran 77 (`!` in Fortran 90).

G.2.1 Scalar Values

Definition of scalar values in PERL must be started with a $. An example from the *.ccl* file is shown below. It is seen that the value of the scalar is not restricted to numbers but can also be character values.

```
!  $nsection = 2 ;
!  $bound = STRUCTURE;
```

G.2.2 Arrays

In PERL arrays must be initiated by `@`. An example of a one- and two-dimensional array is given below:

```
!  @vector = (1, 2, "test");
!  @matrix =( [1, 2, "test1"],[4, "test2", 6]);
```

As seen from the structure of the arrays above, the entries are not restricted to number but can contain all different types of scalar values.

To access entries in the array it should be noted, that the numbering of the entries starts with 0 for the first entry. For the `@vector` array above this means that the entries have to be called with either 0, 1 or 2. An example is given below:

```
!  $vector[0] = 1;
!  $vector[2] = test;

!  $matrix[0][1] = 2;
!  $matrix[1][1] = test2;
```

It should here be noted that the initial sign before `vector, matrix` have been changed from `@` to `$`. This is due to that fact, that all entries in the array are scalar values and when accessing one value this is also a scalar.

To alternate the arrays different commands listed in tab. G.1 can be used.

Table G.1: Array manipulation commands in PERL

Command	Example	Description
`push()`	`push(@vector, 3)`	Adds an element to the end of the array `@vector`. In this case the result is `@vector = (1, 2, "test", 3);`
`pop()`	`$last = pop(@vector)`	`pop()` removes the last entry of the array and return it into the scalar `$last`, in this case `$last = 3`. This is after the use of `push()` above, `@vector = (1, 2, "test");`
`shift()`	`$first=shift(@vector)`	`shift()` removes the first entry of the array and returns it into the scalar `$first`, in this case `$first = 1`, `@vector = (2, "test")`
`unshift()`	`unshift(@vector, 4)`	Adds an element to the beginning of the array `@vector`, In this case the result is `@vector = (4, 2, "test");`

An empty array can be created by `@vector = ();`. The commands in tab. G.1 can then be used to enter values into the array. This can be done directly or by including the commands into a loop. Definition of space for a two-dimensional array is a bit more difficult. Say the dimensions of the array is given by `nRows` and `nCols`. The first step is to make space for a single row with `nCols` entries:

```
for($col = 0; $col < $nCols; $col++)
push @rowMatrix, "0";
```

The next step is to make a two-dimensional array by adding more of these one-dimensional arrays into the same array. This can be done by:

```
for($row = 0; $row < $nRows; $row++)
push @matrix, [ @rowMatrix ];
```

The commands above result in a 0-matrix of dimension `nRows x nCols`.

G.2.3 Loops and Logical Statements

As in MatLab and Fortran it is possible to make loops in PERL. The syntax differs a bit from the other two. Below is an example of a simple `for` loop.

```
!  for $i (1 ..  $nsection) {
f$i = $i
!  }
```

The result of the loop would be:

```
f1 = 1
f2 = 2
```

It is seen that the counter can be used as both a value and as an index on the variables or functions. Also note that after definition of the counter, the loop is enclosed in curly brackets as earlier mentioned.

In the final *.ccl* file used in this project (*add_variables.ccl*) an `if` and `elsif` statement is used. The `if` and `elsif` statements are illustrated below.

```
!  $n = $i ;
!  if ($i < 10) { $n = "00$i" ; }
!  elsif ($i < 100) { $n = "0$i" ; }
```

Note the spelling of the `elsif` statement. The use of paragraphs is necessary when the return value of the statement should be considered as a character which is necessary when the first number is 0. The statement above would return the following:

```
$n = 001     for     $i = 1
$n = 010     for     $i = 10
$n = 100     for     $i = 100
```

The `if` and `elsif` statements are used to name additional variables defined in CFX. The reason that the variables are named this way is, that when calling for these variables from a Fortran routine it is necessary to define the character length and by naming the additional variables this way it is only necessary to define the character length by one value.

G.2.4 CFX Environments

In CFX all model definitions and settings are entered into the model library. For this reason when wanting to define something in CFX from a macro, this should be written in a library environment. In the macro this is done by writing:

```
LIBRARY:
   "Executable statements/definitions"
END
```

Note that when writing things in the macro which are to be entered into CFX the line should NOT be initiated by !. This is because the statements within the LIBRARY environment are not to be considered PERL statements but only text strings which can be read by CFX.

The different parameters in the structure tree in CFX-Pre can all be accessed from the macro. In this project the most useful thing to use the macro for, was to generate expression and additional variables automatically from a defined number of wanted quantities. These two cases are the only ones demonstrated in this introduction. Other parameters are written in the macro and therefore a reference is made to *CELexpressions.ccl*. The structure of these parameters should be obvious from the structure of the expression and additional variables definitions.

Expressions in CFX-Pre should be entered into the `CEL` environment (CFX Expression Language). The expressions in the macro are to be written in the same way as it would have been done when working directly in CFX-Pre. The expressions can be entered within a loop and the index and variables entered into the expression in the way shown above. Below is an example of how the expression structure should be in the macro:

```
   loop start
LIBRARY:
    CEL:
      EXPRESSIONS:
      fx$i = areaInt_x(pZoneX$i)@$bound+areaInt_x(sZoneX$i)@$bound
      px$i = p * step((($i*dz)-z)/1[m])*step((z-(($nhelp)*dz))/1[m])
      sx$i = wall shear x * step((($i*dz)-z)/1[m])*step((z-(($nhelp)*dz))/1[m])
      END
   END
END
   loop end
```

The content of the expressions are not explained further here. A reference is made to the main report where an explanation can be found. The expressions above are

entered into a loop in the *.ccl* file where `! for $i (2 .. $nsection)`. The expressions for `$i = 1` are a bit different. For `$nsection = 3` the macro results in the expressions in CFX-Pre shown in fig. G.2 which is a screen cap from the program.

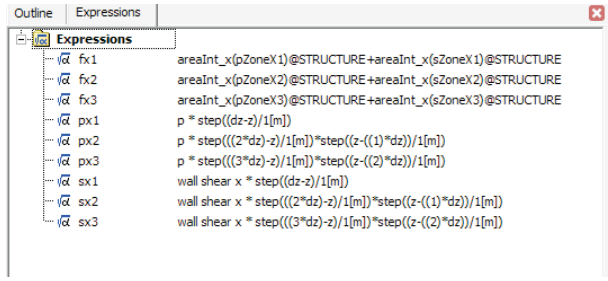


Figure G.2: Expression from macro

Adding an additional variable to the domain in CFX-Pre via a macro routine is relatively easy. When the user has defined a single additional variable the usual way in CFX-Pre the structure of the macro is evident. Below is an extract from *CELexpressions.ccl* where an additional variable is defined.

```
LIBRARY
   ADDITIONAL VARIABLE: pZoneX1
   Boundary Only Field = T
   Option = Definition
   Tensor Type = SCALAR
   Units = [N m^-2]
   Variable Type = Unspecified
   END
END
```

In fig. G.3 a screen cap from CFX-Pre is shown to link the macro language above to the program when defining an additional variable. The line `Boundary Only Field = T` cannot be set directly in CFX-Pre. It only means that the variable only exist on a boundary and not in the entire domain (`T` = True).

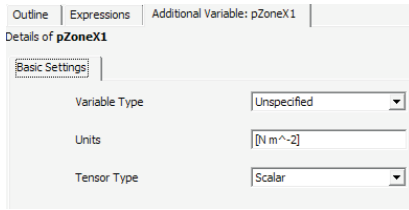


Figure G.3: Definition of additional variable in CFX-Pre

Besides defining the additional variable it has to be assigned a value. This is usually done in "fluid models" under the domain settings. In the macro this is done by entering the following. Note that this is no longer within the `LIBRARY` environment but now this is related to the flow in the simulation. Therefore the value of the additional variable is set in the `FLOW` environment.

```
FLOW:
   DOMAIN: Domain
      FLUID MODELS:
         ADDITIONAL VARIABLE: pZoneX1
         Additional Variable Value = px1
         Option = Algebraic Equation
         END
      END
   END
END
```

To relate the list above to the program a screen cap is shown in fig. G.4.

The illustration of the use of macros in CFX-Pre should not be considered a complete description as it only contains a small amount of the possibilities in using macros. When installing Ansys CFX two files exist within the installation which can be useful when trying to learn macros. In <install dir>\v110\CFX\etc\ a file called RULES and a file called VARIABLES can be found. These files can be opened in Notepad and contain a lot of information about how to write the commands in a macro and which variables are possible. Further in CFX-Pre it is possible to right-click on a lot of the parameters and chose "Edit in Command Editor". When choosing this the text shown in the command editor can easily be copied into the macro as this is the way CFX read the information from the macro. This can also be a useful thing to use when writing macros for the first time.

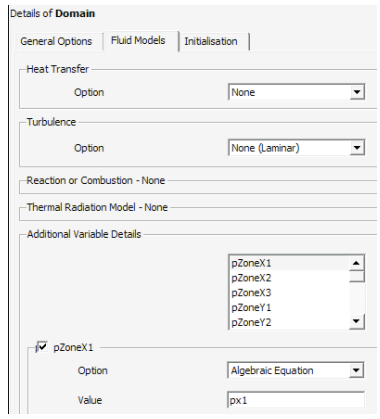


Figure G.4: Definition of value of additional variable in CFX-Pre

G.3 CFX-Post Macro

The basic structure of the macro in CFX-Post is the same as for the macro in CFX-Pre: A definition of the user interface and definition of a subroutine. The file type differs from the CFX-Pre macro. The macro for CFX-Post is a *.cse* file. This type can also be edited in Notepad.

A big difference is the amount of information which can be put into the user interface (GUI). In CFX-Pre only the macro name is visible. In CFX-Post it is possible to include elements for the user to chose the input for the macro. In this project a macro was developed for generating a contour plot on the structure and user surfaces on the basis of this contour plot. Progress in the project showed that the macro was not necessary anyway but is included in this introduction for learning purposes only. The macro (*User_Surface.cse*) is included on the DVD in [*DVD:\CFX\Macros*] and it is recommended that this file be opened when reading this introduction for the same reason as mentioned in app. G.2 for CFX-Pre.

For the use in CFX-Post some predefined macros exist included in the installation. These can be found in <install dir>\v110\CFX\etc\ and can be very useful as inspiration when programming macros for the first time. Especially when dealing with the GUI these are a big help as they all include a user interface which by comparison with CFX-Post when the macro is loaded is quite easy to understand.

The GUI for the macro written in this project is shown below:

```
# Macro GUI begin
#
# macro name = User Surfaces on Structure
# macro subroutine = Surfaces
#
# macro parameter = comment
# type = Comment
# default = This macro generates a contour plot on the
# specified boundary with the specified number of levels.
# Then generates user surfaces on the basis of the contour
# plot.
#
# macro parameter = Contour variable
# type = variable
# default = Z
#
# macro parameter = Location
# type = location
# location type = Boundary
# default = StructureBody
#
# macro parameter = Contour levels
# type = Int
# range = 1, 1000
# default = 1
#
# Macro GUI end
```

Instead of making a long explanation of the entries in the GUI a screen cap from CFX-Post is shown in fig. G.5. In this figure most of the entries can be identified and the result of the GUI should be obvious.

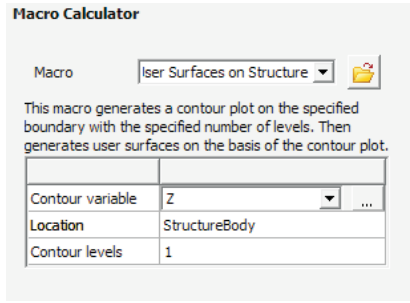


Figure G.5: Graphical User Interface in CFX-Post

It can be seen in fig. G.5 that a drop down list exist for the "Contour variable". In this list all variables available in CFX-Post can be accessed. In this case the vertical coordinate Z was the one used every time and therefor chosen as default. A drop down list also exist for choosing the "Location". The number of contour levels are selected either by entering the number manually or by using a slider which occurs under the data field when the cursor is placed in this. The range of this slider is evident from the GUI above.

The subroutine is written in the same file as the GUI. It is initiated and ended in the same way as the macro for CFX-Pre. In the beginning of the subroutine the input given in the GUI have to be defined as input for the macro. In this case this is done by writing the following line:

```
!  my ($contourvar, $contourLoc, $nlevels) = @_;
```

The term `my` used in the statement means, that the variables only belong to this subroutine. It can be excluded from the statement if the user is working with more subroutines and want to have the variables available in more than one routine. The number of variables in the brackets is dependent on the number of macro parameters entered into the GUI. The GUI used in this case specifies 3 input parameters as indicated in fig. G.5. The value of the variables is given by the row order of the macro parameters as the value is assigned in this order. The last term, `@_`, tells the subroutine that this is a list of incoming parameters to the routine.

As earlier mentioned the macro generates a contour plot on the specified boundary and a given number of User Surfaces equal to the number of contour levels minus 1. As in CFX-Pre a lot of the parameters in CFX-Post can be edited in the command editor by right-clicking and choosing this option. It can therefore

be useful to generate a contour plot manually and then choose to edit this in the command editor. The text in the editor can be copied into the macro directly. The user can then identify important parameters in the list and make use of PERL programming to make loops etc. By using the method of copying the text from the command editor a lot of default values are entered into the macro making a very long list. It can be a bit difficult to overview this list. Experience show however that this is the easiest approach when writing macros in the beginning.

The necessary parameters to generate the contour plot is extracted from the macro and shown below. The rest of the parameters in the list in the macro are default values and can be left out.

```
Domain List = All Domains
Location List = $contourLoc
Colour Variable = $contourvar
Number of Contours = $nlevels
```

The variables in the list is determined by the input in the GUI. In fig. G.6 a screen cap is shown with the default values given in the macro related to the list above.

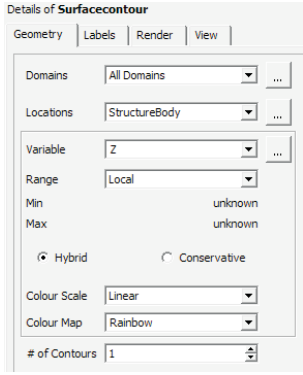
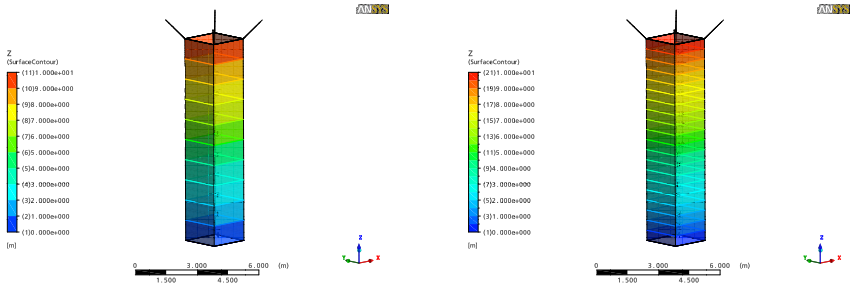


Figure G.6: Default input for generation of contour plot

To get an idea of how the contour looks for different number of levels, two plots are shown in fig. G.7 where the number of levels of Z is 11 and 21 respectively.



a) 11 contour levels *b)* 21 contour levels

Figure G.7: Illustration of contour plots

The User Surfaces are generated on the basis of the number of contour levels. For each level one User Surface is created. Therefore a loop is made in the following way:

```
!  for $nsurf (1 ..  $nlevels-1) {
!  $nhelp = $nsurf+1;
```

The code for the User Surfaces was written by generating a single surface manually and copying the code from the command editor into the macro. The contour level was located and the number was replaced by the value `$nhelp`. The reason that this variable is introduced is that `$nsurf` is used to name the User Surface by an index and that the contour level cannot be an expression e.g `$nsurf+1` but has to be a scalar. The first contour level that can be used is 2 because the surface is generated from the previous contour and up to the next. Using 1 would not result in any surface.

The most important lines in the code for the User Surfaces are extracted and listed below.

```
USER SURFACE: Surface$nsurf
Contour Level = $nhelp
Contour Name = SurfaceContour
Domain List = All Domains
Option = From Contour
```

In fig. G.8 a screen cap is shown relating the list above to CFX-Post.

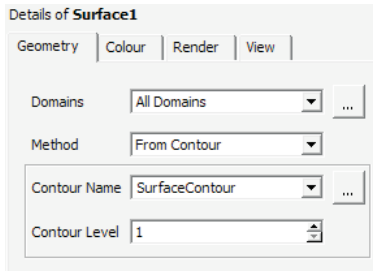


Figure G.8: Most important input for generation of User Surface

It is also possible in CFX-Post to set up expressions. These can also be included in the macro. In *User_Surface.cse* expressions are included which calculate the integrated forces on each of the defined User Surfaces. The expressions in the macro is given by:

```
!  $forceX = force("Surface$nsurf","X");
!  $forceY = force("Surface$nsurf","Y");
!  $forceZ = force("Surface$nsurf","Z");
```

The expressions corresponds to using the `force_<x,y,z>()@"Location"` function. The expressions are in this case not entered into CFX-Post. The results are instead written to a *.dat* file. To write to a file it must be "opened". This is done by the following command. Writing to a file is much like writing to files in html.

```
!  open(OUT,">Forces.dat");
```

The name `OUT` is used every time something is printed to the output file `Forces.dat`. To write to the output file the following command is used which writes a header line in the output file:

```
!  printf(OUT "%10s %13s %16s %19s \n","Surface","Force X","Force Y","Force Z")
```

The numbers in the line specify where the text strings should be placed. The `\n` specifies that a new line is started. By including the line above within the loop over contour levels gives the forces on each User Surface in the output file. After the loop the output file must be closed by the following command:

```
!  close(OUT);
```

For 11 contour levels corresponding to 10 User Surfaces the output files contain the following data listed below:

Table G.2: Data in and structure of `Forces.dat`

Surface	Force X [N]	Force Y [N]	Force Z [N]
1	114.135	0.105888	0.631613
2	128.633	0.0460073	0.393827
3	137.218	-0.0371464	0.206649
4	142.493	-0.126807	0.0783013
5	145.785	-0.0612079	-0.0102736
6	147.817	0.052662	-0.0828953
7	148.78	0.0870237	-0.122044
8	148.801	0.046391	-0.175223
9	146.96	-0.0117619	-0.277649
10	128.84	-0.0224208	-0.203655

The `Forces.dat` file have been included on the DVD in [*DVD:\CFX\Macros*]. It should be noted that the file does not contain data which have been used in any way in the project.

In the Ansys CFX-Post User's Guide more about the use of macros in CFX can be found in the sections regarding "Power Syntax in CFX" on p. 199.

H

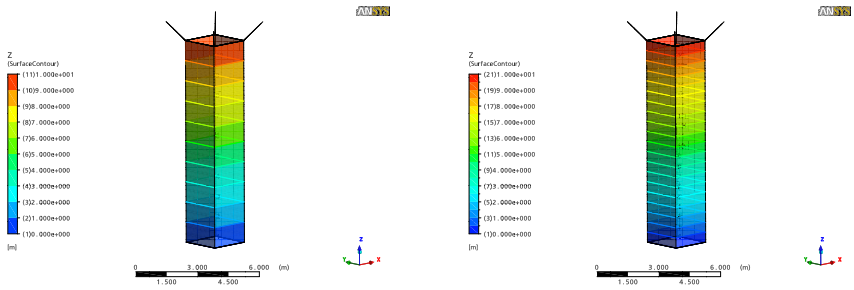
Load Extraction in Transient Runs in CFX, Trials and Error

To extract the loads on the structure in Ansys CFX different approaches were attempted until the final method was chosen. The final method deals with additional variables and calling integrated quantities in the Fortran routine. This method is described further in the main report.

The first approach was to get the loads as nodal values at the grid points on the structure. To get these nodal loads, the pressure at the grid points have to be integrated through a specific area connected to that specific point. A lot of time was used to try and obtain these nodal values. Firstly it was investigated whether it was possible to make a Fortran call to CFX to get these results directly which turned out to be impossible. Secondly if the pressure at the grid points could be extracted along with the specific area connected to each point, the load could be determined by a numeric integration scheme in Fortran. Different calls to different areas were attempted but the correct areas were never found.

During the trial and errors regarding the first approach it was found, that calling the forces as integrated pressure over a specific area, was relatively easy in Fortran. The second approach therefore concerned this fact. it was investigated whether is was possible to use features from CFX-Post during the transient runs. The approach was to generate "User Surfaces" in CFX-Post and use these areas to get the resulting forces on each "User Surface". The procedure was to gen-

erate a "Contour Plot" on the structure surface using the vertical coordinate, z, as variable. The "user Surfaces" could hereafter be generated on the basis of the contour levels, which can be set by the user. In this way it was possible to control the size of the "User Surfaces". Fig. H.1 shows examples of "User Surfaces" created by this procedure.



a) 11 contour levels *b)* 21 contour levels

Figure H.1: Illustration of contour plots

Unfortunately a solution to call this function in CFX-Post during a transient run was not found. The process was firstly used in Ansys CFX 10 where a call to make CFX-Post integrate the pressure over a specified "User Surface" did not show any failure. But when doing the same thing in Ansys CFX 11, CFX-Post started to "hang" and not return a value. A conference with Ansys Support showed, that this was a known problem which can be corrected with a custom fix.

In the process a macro for use in CFX-Post was written but eventually never used for the final approach. A description of the use of macros with Ansys CFX can be found in app. G, where both macros for CFX-Pre and -Post are described.

The third and final approach which finally showed useful in this project, was using additional variables in CFX-Pre as mentioned at the beginning of this appendix. The pressure is divided into n variables each containing the pressure over a specific region. More about this can be found in the main report. Initially also the integration procedure of the variables was included in CFX-Pre but this was later left out as it turned out to be easier to call this function directly from a Fortran routine. To assist in the process of generating n similar variables and CEL expressions a macro for use with CFX-Pre was written. The full macro was not used due to the exclusion of the integration expressions but a description of the macro can be found in app. G.

Bibliography

Cook, R. D., Markus, D. S., Plesha, M. E. & Witt, R. J. [2002]. *Concepts and Applications of Finite Element Analysis*, 4th edn, John Wiley and Sons Inc.

Mathews, J. H. & Fink, K. K. [2004]. *Numerical Methods Using MatLab*, 4th edn, Prentice-Hall Inc.

Menter, F. R. [1993]. Zonal two equation $k-\omega$ turbulence models for aerodynamic flows, *AIAA-93-2906* .

Nielsen, S. R. K. [2004]. *Vibration Theory - Linear Vibration Theory*, Vol. 1, Aalborg tekniske Universitetsforlag.

Nielsen, S. R. K. [2005]. *Structural Dynamics - Computational Dynamics*, Vol. 9, Aalborg tekniske Universitetsforlag.

Pulliam, T. H. [1986]. Artificial dissipation models for the euler equations, *AIAA Journal* **Vol. 24**(No. 12): pp. 1931–1940.

Steger, J. L. & Chaussee, D. S. [1980]. Generation of body-fitted coordinates using hyperbolic partial differential equations, *SIAM Journal SCI. STAT. COMPUT.* **Vol. 1**(No. 4): pp. 431–437.

Versteeg, H. K. & Malalasekera, W. [1995]. *An introduction to computational fluid dynamics: The finite volume method*, 1st edn, Longman Pub Group.

Wilcox, D. C. [2002]. *Turbulence Modeling for CFD*, 2nd edn, D C W Industries.