ECN-C--03-079

DEVELOPMENT OF A WIND TURBINE AERODYNAMICS SIMULATION MODULE

A. van Garrel

August 2003

Abstract

The simulation of wind turbine aerodynamics can be improved with more physically realistic models than the ones currently in use in engineering practice. In this report the mathematical and numerical aspects and the practical use of a new wind turbine aerodynamics simulation module is described. The new simulation code is based on non-linear lifting line vortex wake theory which is a more accurate model for rotor wake physics; an important aspect in wind turbine load simulations. The new model is verified for some test cases with analytical solutions. Wake dynamics are shown to behave as anticipated. The new simulation module, designated AWSM, is expected to substantially reduce the number of uncertainties that accompany currently used blade-element-momentum methods. To assert the quality of AWSM results a comparison with experimental data is recommended. Numerical tests should be performed to investigate the influence of AWSM simulation parameters on the computed results. Further functionality extensions and algorithm optimizations are suggested.

Acknowledgement This report is part of the project “Aerodynamic Windturbine Simulation Module”, In this project a new windturbine aerodynamics simulation code will be developed.

The project is partly funded by NOVEM, the Netherlands Agency for Energy and the Environment, contract number 224.312-0001. Additional funding is provided by ECN, project number 7.4318.

CONTENTS

NOMENCLATURE

Roman symbols

b	[m]	wing span
C_d	[–]	drag coefficient
C_l	[–]	lift coefficient
C_m	[–]	pitching moment coefficient
c	[m]	chord length
L	[N]	lift force
l	[m]	line element length
Ma	[–]	Mach number
Re	[–]	Reynolds number
r	[m]	distance (length of relative position vector), radius
S	[m^2]	surface area
s	[m]	distance
t	[s]	time
U	[$\mathrm{m\,s}^{-1}$]	velocity
V	[m^3]	volume

Greek symbols

α	[rad]	angle of attack
δ	[–]	cut-off radius
η	[–]	dimensionless radial or spanwise coordinate
Γ	[$\mathrm{m}^2\,\mathrm{s}^{-1}$]	vortex-line strength
λ	[–]	tip speed ratio
Ω	[$\mathrm{rad\,s}^{-1}$]	Rotational speed
ρ	[$\mathrm{kg\,m}^{-3}$]	mass density
σ	[s^{-1}]	source strength
θ	[–]	relaxation factor, rotation angle

Tensors, matrices and vectors

$\vec{F}$	[N]	force vector
$\vec{L}$	[N]	lift force vector
$\vec{l}$	[m]	line element
$\vec{n}$	[–]	unit surface normal vector
$\vec{r}$	[m]	relative position vector
$\vec{u}$	[$\mathrm{m\,s}^{-1}$]	velocity vector (also written as (u, v, w))
$\vec{x}$	[m]	position vector (also written as (x, y, z))
$\vec{\Gamma}$	[$\mathrm{m}^2\,\mathrm{s}^{-1}$]	vortex-line vector
$\vec{\nabla}$	[m^{-1}]	derivative operator pseudo-vector $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$
$\vec{\omega}$	[s^{-1}]	vorticity vector

Subscripts, superscripts and accents

$()_\infty$	value at infinity
$()_{cp}$	value at control point
$()_\alpha$	value dependent on angle of attack
$()_{\vec{\omega}}$	value from vorticity distribution
$()_\Gamma$	value from vortex-line
$()_\sigma$	value from source distribution

Acronyms

AWSM	Aerodynamic Windturbine Simulation Module
TE	Trailing Edge
2D	Two Dimensional
3D	Three Dimensional

1 INTRODUCTION

In this report a new wind turbine aerodynamics simulation module developed in the AWSM project is described.

This time-accurate aerodynamics module is based on non-linear vortex-line theory in which the shape and strength of the wake of the blades will develop in time. In this approach the aerodynamic lift-, drag-, and pitching-moment characteristics of the blade cross-sections are assumed to be known and corrected for the effects of blade rotation. In comparison to the currently used blade-element-momentum theory codes, more accurate predictions are expected in situations where local aerodynamic characteristics strongly vary with time and where dynamic wake effects play a significant role. A broad outline of the system's functionality can be found in ref. [3]

This report describes the most important mathematical and computational issues that constitute the core of the AWSM simulation module and are essential to the understanding of the actual Fortran90 code. This is supplemented by a description of the concepts and the information needed to perform a succesful AWSM run.

The mathematical theory underlying the aerodynamics and computational geometrics used in the AWSM simulation code is given in Chapter 2. A non-oscillatory spline interpolation algorithm for the robust interpolation of cross-sectional aerodynamic characteristics is described and demonstrated in this chapter also.

In Chapter 3 the geometric elements, the most important conceptual elements, and the functionality of the software system's Fortran90 programming units are described along with their interdependencies. In this chapter the general philosophy and actual layout of the input and output files can be found also.

Some basic verification tests are reported in Chapter 4. These test computations are performed for problems for which the exact solutions are known.

User guidelines and some example program runs are given in Chapter 5.

In Chapter 6 some concluding remarks and recommendations regarding future developments are made.

2 MATHEMATICAL DESCRIPTION

This chapter describes the mathematics behind the most important components of the wind turbine aerodynamics simulation module. In the first part of this chapter the mathematics for the aerodynamics components is given.

In the second part the theory that concerns the computational geometry aspects of the AWSM simulation code is given. A newly developed non-oscillatory spline interpolation algorithm for the robust interpolation of cross-sectional aerodynamic data is described and demonstrated in this part also.

2.1 Aerodynamics

2.1.1 Introduction

The flowfield around an arbitrary 3D body can be described in terms of velocity vectors $\vec{u}$ or through the distribution of sources σ and vortices $\vec{\omega}$ in the flow domain (see figure 1). Both descriptions are used in the present work. For clarity most of the pictures are shown as 2D representations of a 3D problem.

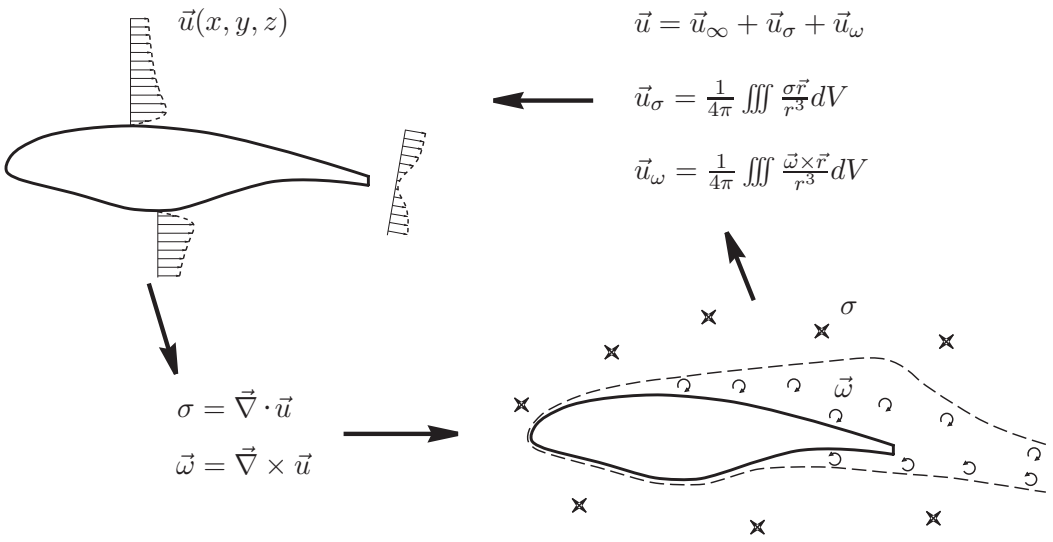

Figure 1: Flowfield representations

The two representations of the flowfield are completely equivalent so there is no cause and effect relationship between them. It is therefore somewhat misfortunate that "*induced* velocity" is the current term used for the velocity field associated with the field distributions of sources and vortices. For a clear discussion on this topic see ref. [1].

As a simplifying measure, the vorticity in the volume around the configuration (see figure 1) can be lumped into vortex lines. In figure 2, adopted from ref. [1], this process of dimension reduction from volume integral to surface integral to line integral is illustrated. Notice that this process causes the velocity distribution to change from character in the vicinity of the vorticity layer. For the source volume distribution a similar simplification can be constructed. Because of their singular behavior, sources and vortices are also referred to as "singularities".

To illustrate the use of these simplifying source and vortex distributions three example flowfield

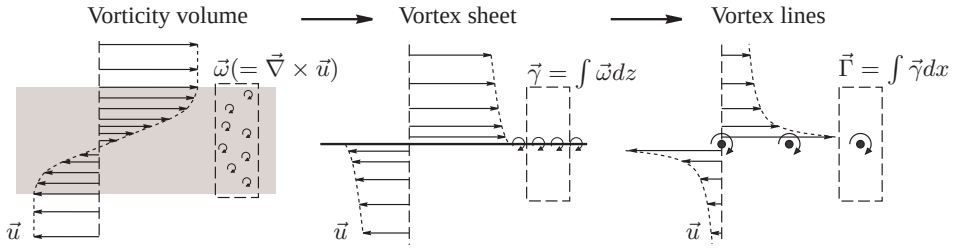

Figure 2: Dimension reduction

approximations for lifting, incompressible flow are shown in figure 3. For the panel method all vorticity and source singularities are distributed on the configuration surface and in the wake. In the lifting surface method no thickness effects are modeled and all surface vorticity is transferred to the mean line of the configuration. Lumping this mean line vorticity to a single point at quarter chord results in the lifting line method. This is the model used for the AWSM simulation module.

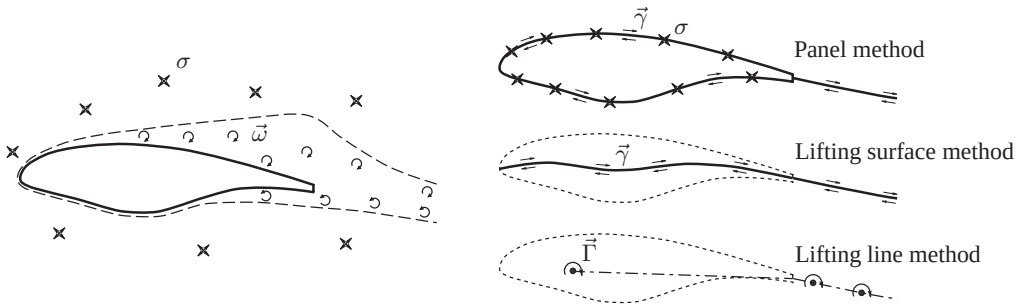

Figure 3: Flowfield approximations

For more information on these integral representations the interested reader is referred to refs. [4],[5],[6] and [8].

2.1.2 Vortex-Line Model

The AWSM flow simulation module for wind turbine rotor aerodynamics is based on generalised lifting line theory. A major principal assumption in this theory is that for each cross-section of a lifting surface the generated lift acts at the quarter chord location and is determined by a local onset flow direction (figure 4) assumed to be aligned with the plane of the cross-section. This restricts the flow simulation to slender and planar or slightly curved blade geometries that do not show strong radial flow interactions.

The effects of viscosity are taken into account through the user-supplied nonlinear relationship between local flow direction and local lift (figure 4), drag and pitching moment coefficients.

Locally occurring onset flow velocities are supposed to be much smaller than the speed of sound and therefore incompressible flow can be assumed. In addition to this thickness or displacement effects are not included in the model. This leads to $\sigma = (\vec{\nabla}\cdot\vec{u}) = 0$ everywhere in the flow domain so that only vorticity effects have to be modeled.

In figure 5 the resulting flow model is sketched. It should be noted that the vortex lines shown

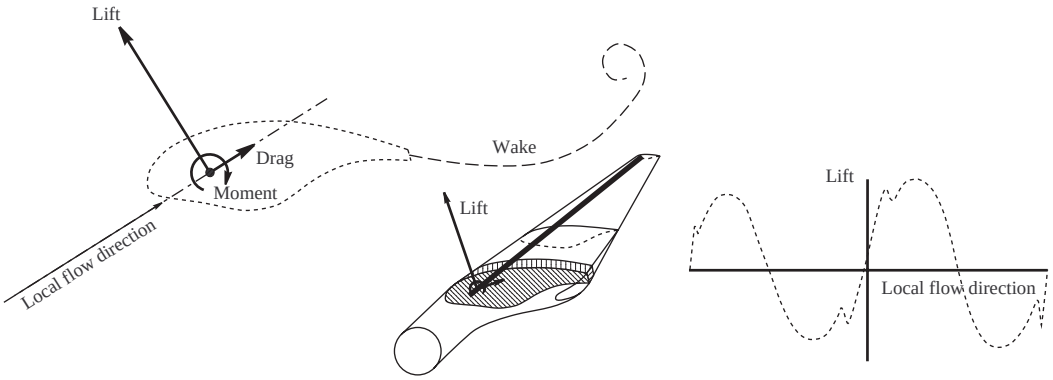

Figure 4: Lifting line representation

are all part of a closed vortex ring. This is a result from Kelvin's circulation theorem that implicitly states that vortex tubes cannot have free ends and have to be closed. A discussion on this topic can be found in ref. [8].

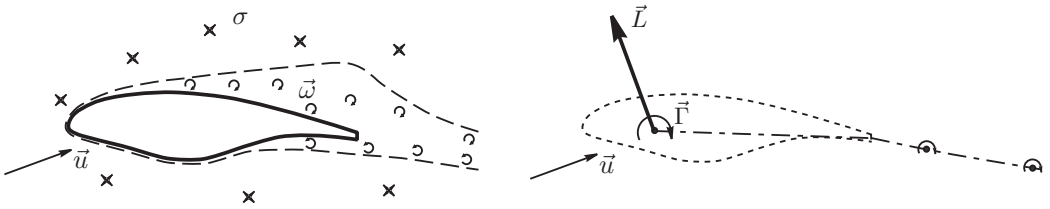

Figure 5: Flowfield model

For a total external force $\vec{F}$ exerted on a body by the fluid we have (see [8])

$$\vec{F} = \iiint \rho(\vec{u} \times \vec{\omega}) dV \tag{1}$$

where the fluid vorticity $\vec{\omega}$ is defined by

$$\vec{\omega} = \vec{\nabla} \times \vec{u} \tag{2}$$

For a vortex line element $d\vec{l}$ this transforms into

$$d\vec{L} = \rho(\vec{u} \times \vec{\Gamma}) dl = \rho\Gamma(\vec{u} \times d\vec{l}) \tag{3}$$

where the vortex line element direction is defined positive in the direction of the lumped volume vorticity distribution.

For the velocity field associated with volume distributed vorticity we have (see [8])

$$\vec{u}_\omega(\vec{x}_p) = \frac{1}{4\pi} \iiint \frac{\vec{\omega} \times \vec{r}}{r^3} dV \tag{4}$$

where $\vec{x}_p$ is the evaluation point and

$$\vec{r} = \vec{x}_p - \vec{x} \tag{5}$$

$$r = \sqrt{\vec{r} \cdot \vec{r}} = |\vec{r}| \tag{6}$$

The equivalent formula for the velocity "induced" by a volume of vorticity lumped into a vortex line element is known as the Biot-Savart law and reads

$$\vec{u}_\Gamma(\vec{x}_p) = \frac{-1}{4\pi}\int \Gamma \frac{\vec{r}\times d\vec{l}}{r^3} \tag{7}$$

2.1.3 Vortex-Line Induced Velocity

For a straight line element with constant vortex strength Γ the integral for its velocity field (7) can be expressed analytically as (see [7])

$$\vec{u}_\Gamma(\vec{x}_p) = \frac{\Gamma}{4\pi}\frac{(r_1+r_2)(\vec{r}_1\times\vec{r}_2)}{r_1 r_2(r_1 r_2+\vec{r}_1\cdot\vec{r}_2)} \tag{8}$$

In this expression $\vec{r}_1$ and $\vec{r}_2$ are the position vectors from the vortex line start and end positions $\vec{x}_1$ and $\vec{x}_2$ to the evaluation point $\vec{x}_p$ respectively (see figure 6). The r_1 and r_2 terms are the position vector lengths.

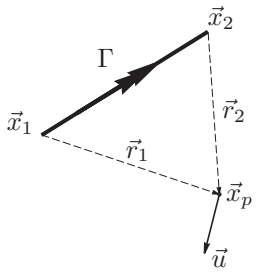

Figure 6: Vortex line geometry

For an evaluation point approaching the vortex line itself equation (8) behaves singularly. This behavior is numerically undesirable and is circumvented with the introduction of "cut-off radius" parameter δ. Equation (8) now becomes

$$\vec{u}_\Gamma(\vec{x}_p) = \frac{\Gamma}{4\pi}\frac{(r_1+r_2)(\vec{r}_1\times\vec{r}_2)}{r_1 r_2(r_1 r_2+\vec{r}_1\cdot\vec{r}_2)+(\delta l_0)^2} \tag{9}$$

In this expression l_0 is the length of the vortex line element.

As the evaluation point approaches the vortex line element the velocity now smoothly goes to zero. In figure 7 the influence of cut-off radius parameter δ on the vortex line velocity is shown for an evaluation line starting at the vortex line midpoint in a direction perpendicular to it. As can be seen the influence of cut-off radius δ on the velocity is strongly felt in the immediate vicinity of the vortex line itself but is hardly noticeable at a distance of half the vortex line length.

A linear cut-off radius function is also implemented in AWSM. Each vortex element is enclosed by a cylinder and two spherical caps. Within this cut-off radius region the velocity decreases linearly towards the vortex line as is shown in figure 8.

It should be noted that with the removal of the singularity, the vortex line velocity field now more resembles a smooth physical vortex. However, we also lost the equivalence between the two flowfield representations shown in figure 1.

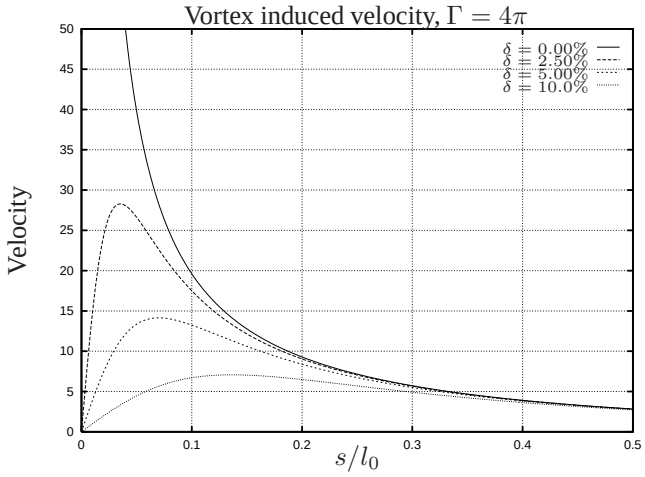

Figure 7: Vortex line velocity, smooth cut-off

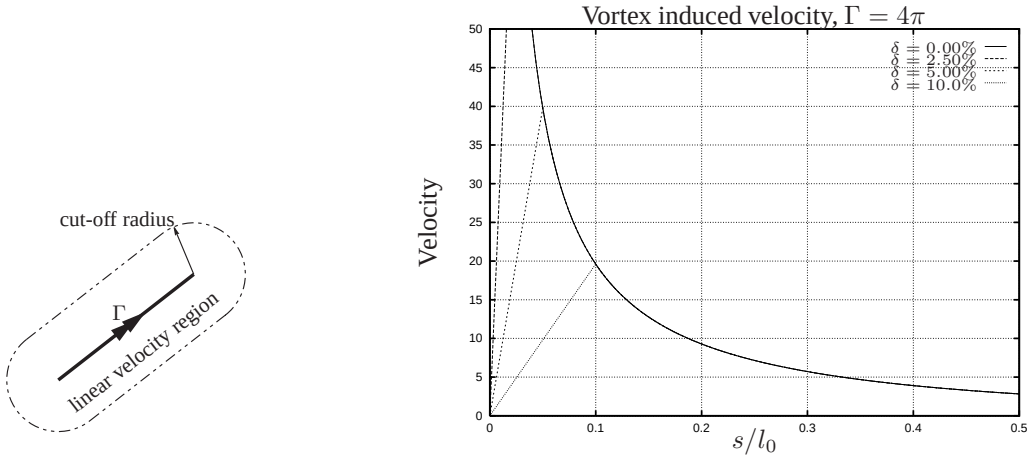

Figure 8: Vortex line velocity, linear cut-off

2.1.4 Vortex Wake

As in the continuous flowfield representation (see figure 1), the vorticity is shed from the trailing edge of the configuration surface and convected downstream in the AWSM flow model as time advances. The blade geometry consists of one or more strips that carry a vortex ring whose bound vortices are located at the quarter chord position and at the trailing edge.

The vortex strengths Γ of these vortex rings are to be determined. Each timestep Δt new vortex rings with these strengths are shed from the trailing edge and joined with the older vortex rings. These vortex rings together will form a vortex lattice. A sketch of the wake geometry for three strips after four timesteps is shown in figure 9.

The position of the first shed free spanwise vortex behind the trailing edge (TE) lies at some fraction between the current TE position and the wind-convected TE position from the previous timestep. In accordance with vortex-lattice practice this fraction is chosen to be 25%. Upstream of this position the vortex rings have a strength equal to the corresponding vortex ring at the configuration.

The position of the downstream part of the wake is determined each timestep by convection of the wake vortex-lattice nodes. This convection is applied in 2 separate steps; convection by the onset wind velocity and convection by the "induced velocity" of all bound- and trailing

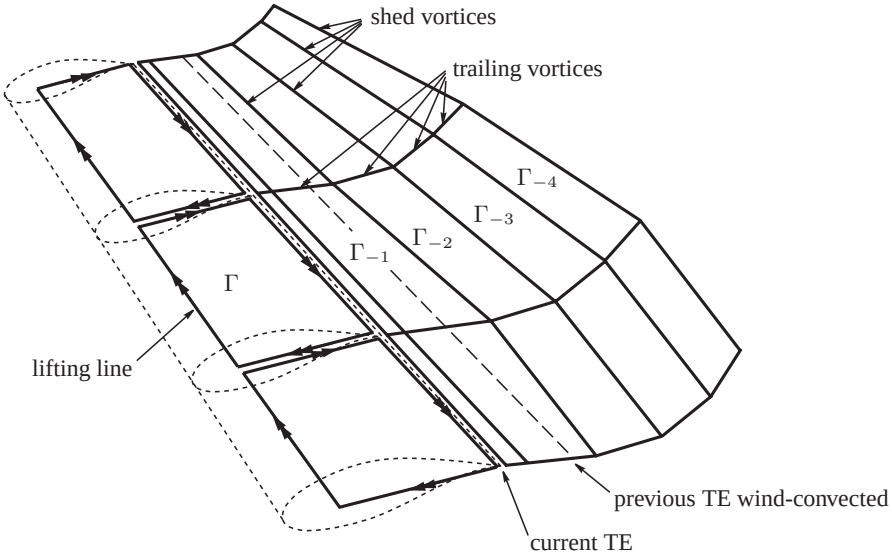

Figure 9: Wake geometry

vortices of all wakes and lifting line systems

$$\Delta\vec{x} = \vec{u}_{wind}\Delta t \quad (10)$$

$$\Delta\vec{x} = \vec{u}_{\Gamma}\Delta t \quad (11)$$

Notice that the wake shed vortices are formed by the adjoining sides of two vortex rings from successive timesteps. This means that they cancel each other in case the vortex ring strengths are identical. For steady flows therefore only the trailing vortices are active.

2.1.5 Non-linear Vortex-Line Strength Computation

The unknown vortex line strengths Γ of the blade strips can be determined by matching the lift force from equation (3) with the lift force that is associated with the local flow direction. The latter is obtained from the user supplied aerodynamic data that relates local angle of attack α to a liftcoefficient C_l. The corresponding lift force is obtained through

$$dL = C_l(\alpha)\frac{1}{2}\rho U^2 dA \quad (12)$$

where U is the strip onset velocity magnitude and dA is the strip area.

The non-linear nature of the vortex-line problem is now easily explained. The lift of each strip depends upon the local flow velocity direction and magnitude as stated by equation (12). This lift in turn fixes the strip vortex strength through equation (3). The strip vortex ring with this strength however acts upon the complete flowfield according equation (9) and therefore influences the lift of each strip we started with.

Matching the strip lift forces from equations (3) and (12) is performed in the cross-section plane defined by the unit vectors in chordwise and normal direction $\vec{a}_1$ and $\vec{a}_3$ (see figure 10).

For the quarter chord lifting line vortex strength related force in the cross-section plane we have from equation (3)

$$dL_{\Gamma} = \rho\Gamma\sqrt{\left((\vec{u}_{cp}\times d\vec{l})\cdot\vec{a}_1\right)^2 + \left((\vec{u}_{cp}\times d\vec{l})\cdot\vec{a}_3\right)^2} \quad (13)$$

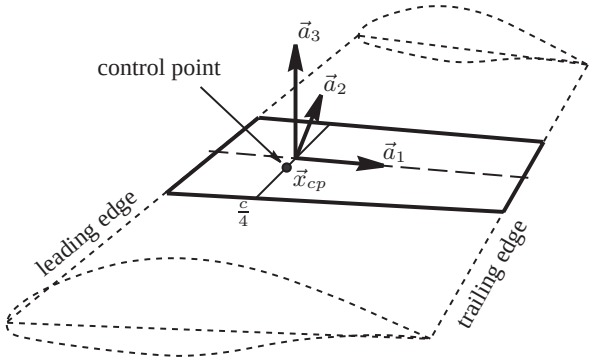

Figure 10: Blade strip geometry definitions

where $\vec{u}_{cp}$ is the total onset velocity at control point location $\vec{x}_{cp}$ composed of wind, motion and the vorticity contributions of all lifting line and wake elements

$$\vec{u}_{cp} = \vec{u}_{wind} + \vec{u}_{motion} + \vec{u}_{\Gamma} \tag{14}$$

It should be noted that the vortex cut-off radius δ used for the vortex induced velocity $\vec{u}_{\Gamma}$ in equation (14) is not necessarily equal to the radius used for the calculation of $\vec{u}_{\Gamma}$ in equation (11) for wake rollup.

The wind onset velocity $\vec{u}_{wind}$ is assumed to be a function of time known in advance and the motion related onset velocity is computed from current and previous control point positions with a backward difference formula

$$\vec{u}_{motion} = -(\vec{x}_{cp} - \vec{x}_{cp}^{old})/\Delta t \tag{15}$$

This formulation for the motion related onset velocity greatly facilitates the future implementation of effects caused by geometry deformations. This topic is discussed further in appendix F at page 101.

From the aerodynamic data table we can write for the force in the cross-sectional plane

$$dL_{\alpha} = C_l(\alpha_{cp})\frac{1}{2}\rho\left((\vec{u}_{cp}\cdot\vec{a}_1)^2 + (\vec{u}_{cp}\cdot\vec{a}_3)^2\right)dA \tag{16}$$

where the angle of attack α_{cp} at the control point is defined by

$$\alpha_{cp} = \arctan\left(\frac{\vec{u}_{cp}\cdot\vec{a}_3}{\vec{u}_{cp}\cdot\vec{a}_1}\right) \tag{17}$$

Setting the vorticity related lift force from equation (13) equal to the liftcoefficient related force from equation (16), we get an expression for the vortex strength as a function of onset velocity:

$$\Gamma_{cl} = C_l(\alpha_{cp})\frac{\frac{1}{2}\left((\vec{u}_{cp}\cdot\vec{a}_1)^2 + (\vec{u}_{cp}\cdot\vec{a}_3)^2\right)dA}{\sqrt{\left((\vec{u}_{cp}\times d\vec{l})\cdot\vec{a}_1\right)^2 + \left((\vec{u}_{cp}\times d\vec{l})\cdot\vec{a}_3\right)^2}} \tag{18}$$

The algorithm to ensure that all strip vortex line strengths satisfy equation (18) for each timestep is as follows:

1. Guess a distribution of blade strip vortex strengths Γ_j
2. Obtain for each blade strip $\vec{u}_{cp}$, the onset velocity at its control point, through equations (9), (14) and (15)
3. Obtain for each blade strip control point the local angle of attack α from equation (17)
4. Determine for each blade strip the lifcoefficient at this angle of attack by interpolation in the corresponding aerodynamic data table
5. Compute for each blade strip a new guess Γ_{cl} for the vortex strength from this liftcoefficient with equation (18)
6. Determine the difference between the new guess and the current blade strip vortex strengths

$$\Delta\Gamma = \Gamma_{cl} - \Gamma_j$$

7. Apply this difference vortex strength for each blade strip with underrelaxation factor θ:

$$\Gamma_j = \Gamma_j + \theta\Delta\Gamma \tag{19}$$

8. Go to start if somewhere the vortex strength difference, made dimensionless by the maximum occurring vortex strength, is larger than some convergence criterion Γ_{crit}:

$$\frac{|\Delta\Gamma|}{|\Gamma|_{max} + 1} > \Gamma_{crit} \tag{20}$$

As an aside, this simple algorithm could possibly be replaced by a more sophisticated Newton-Raphson algorithm to drive the difference between dL_Γ and dL_α to zero. As yet, no such algorithm has been implemented.
The factor 1 in the denominator of equation 20 was introduced to circumvent division by zero.

2.2 Geometry

2.2.1 Strip Reference Quantities

The chordwise and strip normal vectors $\vec{a}_1$ and $\vec{a}_3$ are determined from the four strip points $\vec{x}_6$, $\vec{x}_8$, $\vec{x}_9$ and $\vec{x}_{10}$ (see figure 11) by

$$\vec{a}_1 = \frac{(\vec{x}_6 - \vec{x}_8)}{|(\vec{x}_6 - \vec{x}_8)|} \tag{21}$$

$$\vec{a}_3 = \frac{(\vec{x}_6 - \vec{x}_8) \times (\vec{x}_{10} - \vec{x}_9)}{|(\vec{x}_6 - \vec{x}_8) \times (\vec{x}_{10} - \vec{x}_9)|} \tag{22}$$

where $\vec{x}_9$ and $\vec{x}_{10}$ are the quarter chord vortex end points. Points $\vec{x}_6$ and $\vec{x}_8$ are positioned at the strip leading and trailing edge midway between the strip endpoints. The spanwise unit vector $\vec{a}_2$ is defined by

$$\vec{a}_2 = \vec{a}_3 \times \vec{a}_1 \tag{23}$$

The strip endpoints $\vec{x}_1$, $\vec{x}_2$, $\vec{x}_3$ and $\vec{x}_4$ are extracted from the geometry input specification in which the geometry definition points are supplied as an array of three-component position vectors with the i- and j-direction as indicated in figure 11. Storage order in the array is such

that the i-index runs fastest. The strip normal vectors $\vec{a}_3$ point in a direction defined by the right hand rule vector product from the i-direction to the j-direction.

For the strip area dA we have with $\vec{x}_5$ and $\vec{x}_7$ as points midway between $\vec{x}_1$, $\vec{x}_2$ and $\vec{x}_3$, $\vec{x}_4$ (see figure 11)

$$dA = |(\vec{x}_6 - \vec{x}_8) \times (\vec{x}_7 - \vec{x}_5)| \tag{24}$$

and strip chord c is determined by

$$c = |\vec{x}_6 - \vec{x}_8| \tag{25}$$

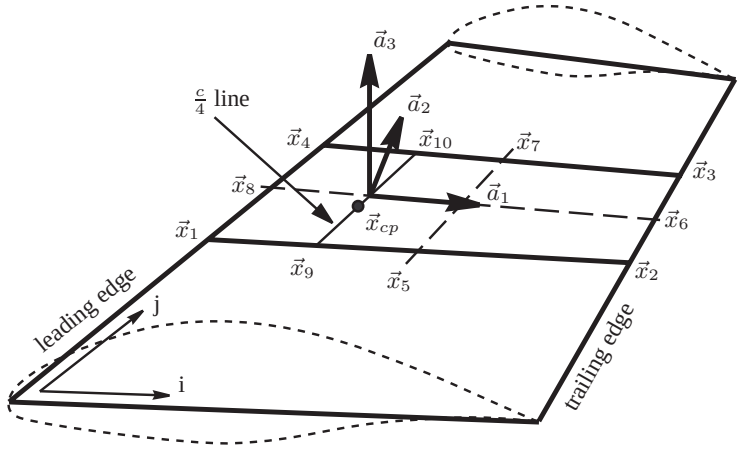

Figure 11: Strip geometry definitions

For the control points where the onset velocity is determined the use of a so-called full-cosine distribution in spanwise direction is known to enhance the accuracy. In the left side of figure 12 the full-cosine distribution is explained. The control points are, like the element ends, distributed at equi-angle increments.

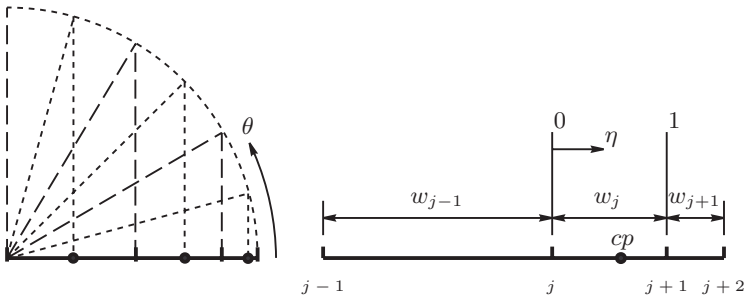

Figure 12: Strip control point location

In the AWSM simulation module this full-cosine control point location is approximated. From the user given geometry the control point location for internal strips is defined with the help of neighboring strip widths w_{j-1} and w_{j+1} computed for each strip j by (see figure 11)

$$w_j = |\vec{x}_{10} - \vec{x}_9| \tag{26}$$

through equation

$$\eta_{cp} = \frac{1}{4}\left(\frac{w_{j-1}}{(w_{j-1} + w_j)} + \frac{w_j}{(w_j + w_{j+1})} + 1\right) \tag{27}$$

The first element with width w_1 has a control point location defined by

$$\eta_{cp} = \frac{w_1}{w_1 + w_2} \tag{28}$$

and for the last element with width w_{n-1} this becomes

$$\eta_{cp} = \frac{w_{n-2}}{w_{n-2} + w_{n-1}} \tag{29}$$

Notice that for an equidistant distribution of the strip endpoints the full-cosine approximation will reproduce an equidistant control point distribution. In general, the control point locations will tend to be pulled towards the narrowest neigboring segment strip.

2.2.2 Transformation Matrices

In the AWSM simulation module solid body rotation and translation are performed through the use of transformation matrices expressed in a homogeneous coordinate system. Compound transformations combining body rotation and translation can then be represented as matrix products (see ref. [2]). These 4 by 4 matrices work on homogeneous coordinates which are composed of the 3 components of a Cartesian vector supplemented with a 4th component equal to 1. Thus the vector (x, y, z) is extended in homogeneous form to $(x, y, z, 1)$.

The rotation matrix $\mathbf{R}$ for rotation of a (position) vector about a general axis through the origin with direction $\vec{w} = (w_1, w_2, w_3)$ through an angle θ, interpreted in a right-hand sense, reads

$$\mathbf{R} = \begin{pmatrix} w_1w_1(1-c)+c & w_1w_2(1-c)-w_3s & w_1w_3(1-c)+w_2s & 0 \\ w_2w_1(1-c)+w_3s & w_2w_2(1-c)+c & w_2w_3(1-c)-w_1s & 0 \\ w_3w_1(1-c)-w_2s & w_3w_2(1-c)+w_1s & w_3w_3(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{30}$$

where

$$\begin{aligned} c &= \cos\theta \\ s &= \sin\theta \end{aligned} \tag{31}$$

and

$$\vec{w} \cdot \vec{w} = 1 \tag{32}$$

The translation of a position vector $\vec{x}$ can be performed through multiplication with the translation matrix $\mathbf{T}$ as in $\vec{x}^{new} = \mathbf{T}\vec{x}$. The translation matrix reads

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{33}$$

where (t_1, t_2, t_3) are the Cartesian components of the translation.

Notice that the transformation matrix for a translation in the reverse direction $(-t_1, -t_2, -t_3)$ is the inverse of matrix $\mathbf{T}$. Rotation about a general axis *not* through the origin can now be performed by a compound transformation matrix $\mathbf{A}$, which is composed of a translation of the axis to the origin by $\mathbf{T}$, a rotation $\mathbf{R}$ and a translation back to the original position with $\mathbf{T}^{-1}$.

$$\vec{x}^{new} = \mathbf{T}^{-1}\mathbf{R}\mathbf{T}\vec{x} = \mathbf{A}\vec{x}$$

2.2.3 Spline Interpolation

The user supplied aerodynamic lift-, drag- and pitching moment coefficients are given for a finite number of angle of attack values. For use in equation (16), some kind of interpolation has to be performed. In this section a new non-oscillatory spline algorithm is given. That is, locally convex data will be splined with a convex interpolant. The data is assumed to be a single valued function.

The 2nd order derivative of the spline is not required to be continuous across elements. As a result, the interpolation algorithm does not set up a system of equations that have to be solved for.

As is illustrated in figure 13, the 1st order derivatives for internal points are determined in advance from a line through the neighboring data points.

$$y'_k = \frac{(y_{k+1} - y_{k-1})}{(x_{k+1} - x_{k-1})} \tag{34}$$

For the first and last data points at x_1 and x_n the first order derivatives are set by

$$y'_1 = 2\frac{(y_2 - y_1)}{(x_2 - x_1)} - y'_2 \tag{35}$$

$$y'_n = 2\frac{(y_n - y_{n-1})}{(x_n - x_{n-1})} - y'_{n-1} \tag{36}$$

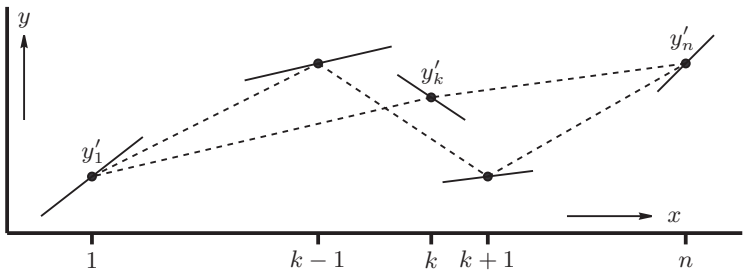

Figure 13: Rational Hermite spline 1st order derivative determination

For the spline function $f_k(x)$ in each interval $[x_k, x_{k+1}]$ we choose from ref. [9]

$$f_k(x) = A_k u + B_k t + C_k \frac{u^3}{p_k t + 1} + D_k \frac{t^3}{q_k u + 1} \tag{37}$$

where

$$t = \frac{x - x_k}{x_{k+1} - x_k}, \quad u = 1 - t \quad \text{and} \quad -1 < p_k,\, q_k < \infty \tag{38}$$

The p_k and q_k terms are so-called spline weights. For $p_k, q_k \to \infty$ the function will transform to a linear interpolant. This property and the particular choice for the 1st order derivatives guarantees the existence of a set of p_k and q_k values that interpolates convex data with a convex interpolant and concave data with a concave interpolant.

For a given set of spline weights p_k, q_k and 1st order derivatives y'_k we can determine the spline constants A_k, B_k, C_k and D_k through (see ref. [9])

$$C_k = \frac{(3 + q_k)\Delta y_k - (2 + q_k)\Delta x_k y'_k - \Delta x_k y'_{k+1}}{(2 + p_k)(2 + q_k) - 1} \tag{39}$$

$$D_k = \frac{-(3+p_k)\Delta y_k + \Delta x_k y'_k + (2+p_k)\Delta x_k y'_{k+1}}{(2+p_k)(2+q_k)-1} \quad (40)$$

$$A_k = y_k - C_k \quad (41)$$

$$B_k = y_{k+1} - D_k \quad (42)$$

where

$$\Delta x_k = x_{k+1} - x_k \quad (43)$$

$$\Delta y_k = y_{k+1} - y_k \quad (44)$$

The 2nd order derivatives of the spline function $f_k(x)$ in interval $[x_k, x_{k+1}]$ at the end points are related to the spline constants by (ref. [9])

$$y''_k(\Delta x_k)^2 = 2(p_k^2 + 3p_k + 3)C_k \quad (45)$$

$$y''_{k+1}(\Delta x_k)^2 = 2(q_k^2 + 3q_k + 3)D_k \quad (46)$$

The sign of the 2nd order derivatives is therefore directly dependent upon the sign of the C_k and D_k values. For the 2nd order derivatives y''_k and y''_{k+1} at the interval end points we can construct finite difference type of estimations

$$\begin{aligned} y''_k &\approx \Delta^2 y_k = 2\frac{\frac{\Delta y_k}{\Delta x_k} - y'_k}{\Delta x_k} \\ y''_{k+1} &\approx \Delta^2 y_{k+1} = 2\frac{y'_{k+1} - \frac{\Delta y_k}{\Delta x_k}}{\Delta x_k} \end{aligned} \quad (47)$$

Unwanted oscillatory behavior of the spline function for the given set of spline constants is present when $\Delta^2 y_k\, C_k < 0$ or $\Delta^2 y_{k+1}\, D_k < 0$. When this happens the spline weights p_k or q_k are respectively set at a value that results in $C_k = 0$ or $D_k = 0$ and therefore zero 2nd order derivatives.

Setting the numerator in equation (39) to zero when $\Delta^2 y_k\, C_k < 0$ results in

$$q_k = \frac{-3\Delta y_k + 2\Delta x_k y'_k + \Delta x_k y'_{k+1}}{\Delta y_k - \Delta x_k y'_k} \quad (48)$$

For the case that $\Delta^2 y_{k+1}\, D_k < 0$ we can find from equation (40)

$$p_k = \frac{-3\Delta y_k + 2\Delta x_k y'_{k+1} + \Delta x_k y'_k}{\Delta y_k - \Delta x_k y'_{k+1}} \quad (49)$$

With the new p_k or q_k weights the spline constants A_k, B_k, C_k and D_k are recalculated. This process is repeated until the spline weights p_k and q_k do not change anymore. Some interpolated example datasets are shown in figures 14, 15 and 16.

In figure 14 the data is constructed from two linear functions with discontinuous 1st order derivatives. As can be seen in the blowup of the kink the spline's 1st order derivative is continuous.

The second example dataset is constructed from a linear piece abutted to a quarter circle of radius 0.5. In this case the 2nd order derivatives of the two pieces are discontinuous. As can be seen in figure 15 there are no extraneous inflection points in the interpolant.

As a final test a dataset resembling the liftcurve of an airfoil with a sudden change in function value is interpolated. Cubic spline algorithms introduce large overshoots in such cases. The results from the new rational Hermite spline algorithm are shown in figure 16.

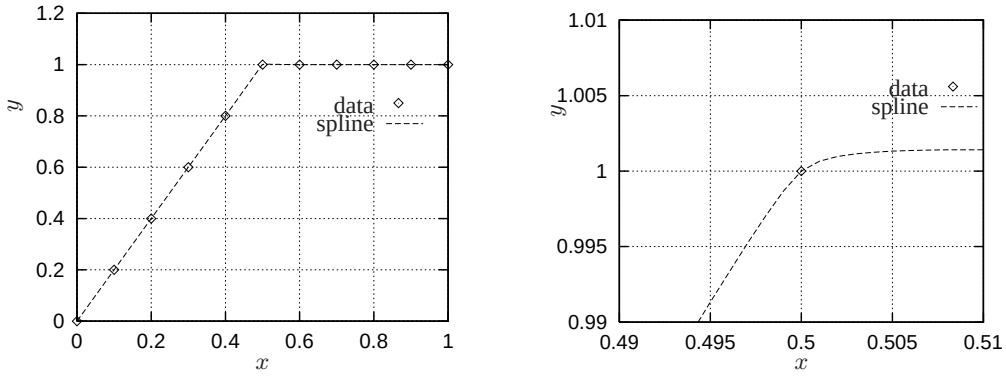

Figure 14: Rational Hermite spline, test 1

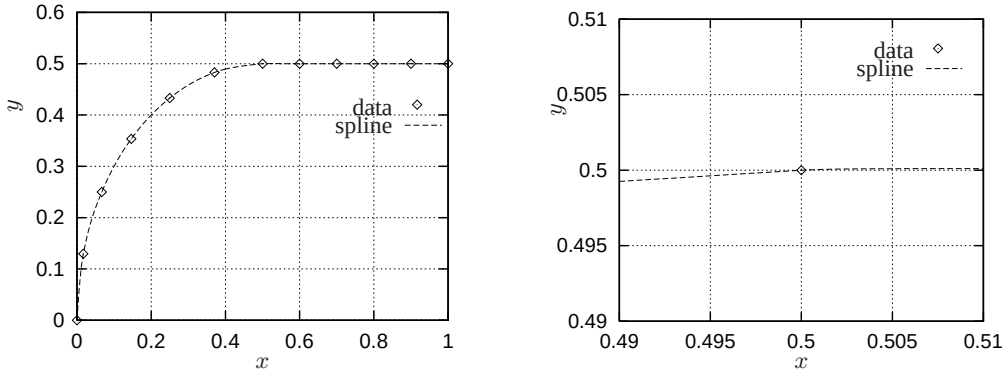

Figure 15: Rational Hermite spline, test 2

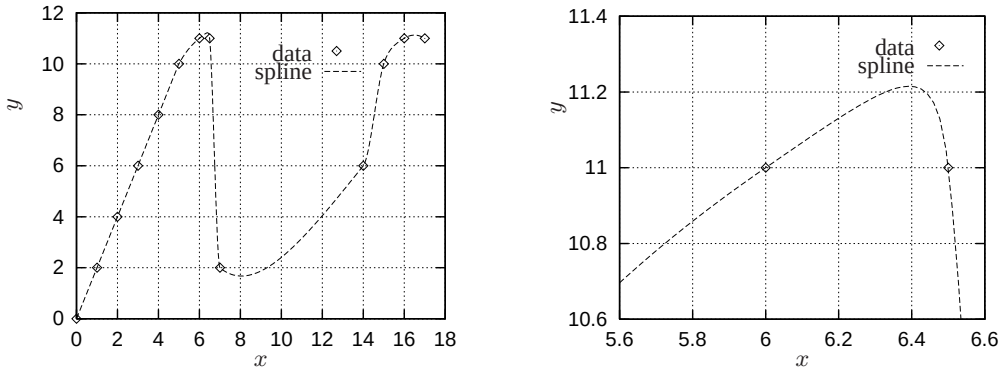

Figure 16: Rational Hermite spline, test 3

3 SOFTWARE DESCRIPTION

In this chapter a broad outline of the AWSM software system is given. The software components into which the system is subdivided are described and their interdependence is clarified. In addition, the philosophy behind the layout of the input files is given in a separate section. It should be noted that no attempt is made to be complete in the description of the software system. The final say in this matter is given to the actual code itself.

3.1 Programming

3.1.1 Fortran90

The AWSM software components are written in standard Fortran90. No use is made of any operating system or compiler specific functions. Deprecated and obsolete features, present in Fortran90 for backward compatibility reasons, are discarded altogether. Among these are `COMMON` blocks, `INCLUDE`, `BLOCK DATA`, `DO WHILE` and fixed source form.

The choice for Fortran90 as programming language was mainly suggested by its long history in technical environments and the maturity of optimizing compilers. Another factor in favor of Fortran90 were features like dynamic memory allocation, pointers and the inclusion of derived data types.

However, Fortran90 lacks features that modern Object Oriented languages like C++ and Java offer of which inheritance is the most important one. This precludes subtyping and most instances of polymorphism. A more subtle shortcoming is the inability to have arrays of pointers in Fortran90.

In addition to this there is no enforced cohesion in Fortran90 between derived data types and the operations that can be performed on them. The resulting functional programming style in Fortran90 makes it more cumbersome to write well-organised, easy maintainable programs.

3.1.2 Structure and Documentation

Modules For each self-contained group of related functionality, defined by types, subroutines, parameters, functions and interfaces, a new Fortran90 module is created.

Initially, in AWSM each derived data type was placed in its own Fortran90 module together with the subroutines and functions that operated on them or made use of them. However, this Object Oriented programming style imitation was quickly abandoned because of the unavoidable, but impossible, circular references. Now, all derived data types are placed in one module named `VXTypes_m`. A listing can be found in appendix A at page 68.

All AWSM modules are named like `MyModule_m` and are placed in a file with the same name and file extension `.f90`.

Naming Conventions Although Fortran90 is case-insensitive, the naming conventions followed in the AWSM project take advantage of the clarifying use of character case.
However, the compiler makes no distinction between for example the readable variable name `myInputVariable` and the more obscure `MYINPUTVARIABLE`. In table 1 the naming conventions used are summarised.

Table 1: AWSM Fortran90 naming conventions

Subject		Example name
keywords	:	lowercase
modules	:	CapitalizedWithInternalWordsAlsoCapitalized_m
constants	:	UPPER_CASE_WITH_UNDERSCORES
variables	:	firstWordLowerCaseButInternalWordsCapitalized
functions	:	firstWordLowerCaseButInternalWordsCapitalized()
subroutines	:	firstWordLowerCaseButInternalWordsCapitalized()

Consistency To increase code readability and maintainability a consistent set of choices for coding and code layout is used in the AWSM project:

1. Use lower case words as default
2. Coding clarity is of highest priority.
3. Code indentation with two spaces.
4. Use `implicit none` in each module.
5. Use of `intent(in), intent(out)` or `intent(inout)` for dummy subroutine parameter declarations.
6. Use inline comments generously and keep them short and simple.
7. Keep global constants together.
8. Use meaningful variable names, if possible unabbreviated.
9. Use empty lines generously to separate logical program blocks from each other.
10. Use variable names starting with `i,j,k,l,m` or `n` for integer variables.

All AWSM modules start with a comment block explaining purpose and functionality of the code elements. A template comment block is listed here:

```
!******************************************************************************
!* FILE :  The current filename (ModuleName_m.f90)
!*
!* PROGRAM UNIT : The current module name (module ModuleName_m)
!*
!* PURPOSE : In one or two sentences the module's purpose
!*
!* MODULES USED : All modules imported with FORTRAN90 USE statements
!*
!* UPDATE HISTORY :
!* VSN                DATE                AUTHOR          SUMMARY OF CHANGES
!* module version     date of change      name            description of changes
!*
!******************************************************************************
!* SHORT DESCRIPTION :
!*   Short description of the overall functionality
!*
!* REMARKS :
!*   Warnings, peculiarities etc.
!*
!* USAGE:
```

```
!*   Description of how to use the most important components of this module
!*
!*****************************************************************************
module ModuleName_m
  use Types_m
  implicit none
  ...
contains
  ...
end module ModuleName_m
```

3.1.3 AWSM Fortran90 modules

In this section a short description of all AWSM Fortran90 modules is given. These modules are listed together with their mutual dependencies in table 2. Moreover, for each module a reference to the page in appendix A is given where all module comment blocks are included. A listing of all Fortran90 interfaces, subroutines and functions for the relevant AWSM modules is given in appendix B.

The AWSM main program itself is, in slightly shortened form, listed in appendix C. From the source code listing the overall program logic should be easily understood.

An extensive description of the AWSM specific geometric and conceptual elements used in this chapter is given in appendix D at page 89.

It is mentioned here again that all derived data types are placed in one module named `VXTypes_m` of which a listing can be found in appendix A on page 68.

Table 2: AWSM Fortran90 module dependencies

Nr	Page	Filename	Uses	Used by
1	85	AWSM.f90	3,4,5,6,7,9,10,13,14,15,16,17,19,20,21	none
2	51	AeroData_m.f90	10,13,14,15,17	3,9,19
3	51	AeroLink_m.f90	2,9,10,13,14,17	1
4	52	Ask_m.f90	14,15	1,5
5	53	DataInput_m.f90	4,13,14,16	1,19,20,21
6	54	DataOutput_m.f90	13,14,15,16,17,18,22	1
7	55	GeomScenario_m.f90	8,9,10,11,12,13,14,15,16,17,18	1
8	56	GeomSnapshot_m.f90	9,10,11,13,14,15,16,17,18	7
9	57	Geom_m.f90	2,10,11,13,14,15,16,17,18,19,22	1,3,7,8
10	58	InputFile_m.f90	13,14,15,16,17,18	1,2,3,7,8,9,20,21
11	61	Mat4_m.f90	13,14,15,16,17,18	7,8,9,21
12	62	Spline_m.f90	13,14,15	7,19,20
13	63	Trace_m.f90	14,16	1,2,3,5,6,7,8,9,10,11,12,15,18,20,21,22
14	65	Types_m.f90	none	all
15	65	Utils_m.f90	13,14,16,17	1,2,4,6,7,8,9,10,11,12,20,21
16	66	VXParam_m.f90	14	1,5,6,7,8,9,10,11,13,15,17,18,19,20,21
17	68	VXTypes_m.f90	14,16	1,2,3,6,7,8,9,10,11,15,18,19,20,21,22
18	72	Vec34_m.f90	13,14,16,17	6,7,8,9,10,11,19,20,21,22
19	73	VortexLattice_m.f90	2,5,12,14,16,17,18,22	1,9
20	75	WindScenario_m.f90	5,10,12,13,14,15,16,17,18,21	1
21	76	WindSnapshot_m.f90	5,10,11,13,14,15,16,17,18	1,20
22	77	Wind_m.f90	13,14,17,18,21	6,9,19

AWSM The main program is AWSM, a wind turbine aerodynamics simulator based on generalized non-linear lifting line vortex wake theory. The main program itself (see listing at page 85) can be considered the top-level view of the time-stepping procedure.

AeroData_m The AeroData_m module handles the creation of an AeroData object from a given aerodynamic database file that contains aerodynamic coefficients like lift, drag

and pitching moment for one or more airfoils.

AeroLink_m The AeroLink_m module handles input and use of data that establish the relation between aerodynamic coefficients and airfoil sections in the geometry.

Ask_m Ask_m is a general purpose module to ask the user to enter an answer to a question on standard output (i.e. screen). An answer can be in one of the Fortan90 types `INTEGER`, `REAL`, `DOUBLE`, `LOGICAL` or `CHARACHTER(*)`. An extra feature is the ability to incorporate specific behavior through the use of a special first character in the answer. For AWSM this is defined to be the forward slash "/" character.

DataInput_m In the DataInput_m module user command line input is stored as global data. See section 5 for a discussion on the specific command line input questions.

DataOutput_m This module's purpose is to write results from the computation to output files. It is expected that this will be the most frequently adjusted module to comply with user specific demands.

GeomScenario_m The GeomScenario_m module handles the user supplied prescribed geometry motion scenario. It also includes the option to make a snapshot of the GeomScenario at the current instance in time.

GeomSnapshot_m The GeomSnapshot_m module enables geometry snapshot handling; the location and orientation of Markers in space at a specific instance in time. With the movement of the Markers the geometry of all attached child Marker - Part combinations are transformed.

Geom_m The Geom_m module enables basic geometry input handling. It includes the option to link two Marker - Part combinations together and the activation of the current GeomSnapshot object. The convection of the old trailing edge location with the wind velocity (see section 2.1.4) is located here also.

InputFile_m The InputFile_m module is a module that enables interpretation of input files. It contains generic subroutines and functions for the iteration over lines and words.

Mat4_m This module contains the definitions and functions associated with solid body transformation matrices. The option to determine the transformation matrix that matches two different Markers together is located here also.

Spline_m This module contains the subroutine for rational Hermite spline interpolation. This enables the non-oscillatory interpolation of a dataset. The spline is used to interpolate the aerodynamic airfoil data and optionally the WindScenario and GeomScenario objects at the current instance in time.

Trace_m The Trace_m module provides logging of messages, generated during the execution of the program, to the enabled output streams. It can be used as a debugging tool during development and as a troubleshooting tool once the system has been deployed in a production environment. There are five log levels. Once the log level has been set by the user, only messages in the execution of the program of equal or higher severity will be logged to standard output (i.e. screen) or to a specified log file. In table 3 the interpretation of the log levels are given in order of decreasing severity

Types_m The Types_m module contains general use Fortran90 `KIND` types, mathematical constants and tolerance factors (see page 65).

Table 3: AWSM logging facility

Log level		Description
MESSAGE	:	Informational message; always written to the enabled output streams.
FATAL	:	Something bad has happened, the system is of little use to anyone.
ERROR	:	A specific task may have failed, but the system will keep running.
WARNING	:	Something went wrong but the system will try to recover from it.
INFO	:	Informational messages about what's happening.
DEBUG	:	Hints used during the initial coding and debugging process.

Utils_m A small collection of utility program units (see appendix A page 65).

VXParam_m In module VXParam_m the definition of project AWSM specific constant parameters is given. It contains all input file specific keywords. Both the relative position of the lifting line and the first free "bound vortex" in the wake (see figure 9) are defined here also.

VXTypes_m In this module all project AWSM specific types are defined among which are the definitions for Parts, Segments, Markers and Links . A listing can be found in appendix A at page 68.

Vec34_m This general use module handles 3D vectors, their associated 4D homogeneous forms and mathematical operations for 3D vectors like addition, subtraction, negation, multiplication and division by a factor, the vector dot product and the vector cross product. Moreover, it contains the utility functions to determine the length of a vector, the angle between two vectors and some functions for conversion to and from regular three-element arrays and four-element homogeneous coordinates.

VortexLattice_m Module VortexLattice_m is the core of the aerodynamics part of AWSM and handles operations on the vortex system that is attached to a Segment's surface. Induced velocity computations, lifting line forces and wake rollup deformations are all performed by subroutines defined in this module.

WindScenario_m The WindScenario_m module handles the time sequence of onset wind velocity. It contains the subroutine to create a snapshot (through interpolation or extrapolation) from the wind field at a specific moment in time.

WindSnapshot_m The WindSnapshot_m module enables handling of a wind snapshot; the wind velocity components at a specific instance in time from the WindScenario.

Wind_m The Wind_m module's purpose is to provide functions that return wind velocity vectors at specified points in space at a specific instance in time.

3.2 Input Files

This section describes the general ideas and concepts that underlie the design of the input file layout. A formal and more detailed description of all required input files for an AWSM simulation is given in appendix E at page 93. In section 5.2 at page 38 the actual layout of the AWSM input files for an example computation is given.

AWSM input files are loosely based on the XML markup language layout. A big advantage with respect to the frequently used key-value based file organization is the possibility to define

hierarchical input structures. By this it is feasible to have a more Object Oriented input file style definition and to organize input data more resembling the internal Fortran90 `TYPE` definitions. For example, the key-value representation for a sphere would be:

```
Sphere_Purpose="soccer"
Sphere_Radius=0.15
Sphere_Color="red"
Sphere_Weight=500
```

This would pose ambiguity problems in case there are multiple spheres defined in the input file or if other objects use spheres as components. The AWSM style input would look like

```
! AWSM style definition of a sphere
<Sphere>
  <Purpose> "soccer" </Purpose>
  <Radius> 0.15 </Radius>
  <Color> "red" </Color>
  <Weight> 500 </Weight>
</Sphere>
```

The use of an explicit begin `<Tag>` and end `</Tag>` for an object or object property simplifies the construction of compound objects. Note the comment line at the top of the "AWSM sphere" definition. The leading blanks in some of the input lines are ignored by AWSM but can be inserted to make the data more human readable. In fact, the whole definition of the sphere could be written on one line.

A list of general formatting rules for AWSM input files is given here:

1. Objects and properties begin and end identifiers are specified by `<Tag1>` and `</Tag1>` respectively with the proper keyword substituted for `Tag1`. There are no blanks allowed on either side of the keyword.

2. Words are separated from begin and end identifiers and from each other by at least 1 blank.

3. Lines starting with an exclamation mark `!` or a sharp # character are comment lines and are completely disregarded. Trailing comments are *not* supported.

4. Empty lines are considered comment lines.

5. Fortran90 `LOGICAL` input, i.e. yes/no type of data, can be specified in a multitude of ways. For example, in

   ```
   <LightIsOn> YES </LightIsOn>
   ```

   the word `YES` could be replaced by `Y, 1, ON, T` and `TRUE`. The negative data version would use `NO, N, 0, OFF, F` and `FALSE`. The data is interpreted case-insensitive.

6. The data in the specification of array elements can be partitioned freely. So is the following data for a two by five array properly specified:

   ```
   <Array2x5>
    1.0 2.0 3.0 4.0
    5.0 6.0 7.0
    8.0 9.0
    10.0
   </Array2x5>
   ```

7. Character string variables are identified by one of the special separation characters to allow for inclusion of blanks. Character strings are allowed to span multiple lines in the input file in which case each line must start and end with the same separation character. Character string separation characters are `"`, `^` and `|`. Examples of well-formed character string data are

```
<Name> "Arne van Garrel" </Name>
<Info>
  |AWSM a wind turbine aerodynamics simulation code |
  |that is based on generalised lifting line theory.|
</Info>
```

8. There is no ordering assumed for the specification of objects or object properties in the AWSM input files. For example, the earlier mentioned sphere could also be specified as

```
<Sphere>
  <Weight> 500 </Weight>
  <Color> "red" </Color>
  <Purpose> "soccer" </Purpose>
  <Radius> 0.15 </Radius>
</Sphere>
```

3.3 Output Files

The AWSM output files currently are all ASCII based text files. The files are created in Fortran90 `REPLACE` mode and will thus overwrite files that have the same name.

The aerodynamic and geometric output files are handled in the DataOutput_m module which definitely is the most dynamically changing part of the AWSM group of Fortran90 modules. Because of the vast amount of aerodynamic and geometric data that is generated during a simulation, it is not feasible to write everything to external files. The actual subset of possible output quantities is currently adjusted for each problem at hand. In a future AWSM version this practice will be replaced by one in which output quantities can be specified from the command line or through the use of an input file. The use of Fortran90 direct access files is then an obvious enhancement.

In an AWSM simulation run it is possible to have log data written to file and/or to screen. A description of the log data is given in appendix G at page 103.

3.3.1 Aerodynamics

As was already mentioned, the AWSM output files do not yet have a fixed layout or have a mechanism to dynamically select the quantities to be written to file. Currently the aerodynamics results are written to the AeroResults file through a call in the main program to writeAeroResults(theGeom,...). The clear construction of all AWSM defined types in the VXTypes_m module however (see appendix A, page 68) makes it easy to adapt the output to ones needs.

The core of subroutine writeAeroResults(aGeom,...), defined in the DataOutput_m module, looks like:

```
! Write the segment and wake aerodynamic data to file
subroutine writeAeroResults(aGeom,...)
  type(Geom), intent(in)   :: aGeom
  ...
```

```
  ! Loop over all Parts in the Geom object
  do ip=1,aGeom%nPrt

    ! Loop over all Segments in the Part object
    do is=1,aGeom%prt(ip)%nSeg

      ! Loop over all strips in the Segment
      do j=1,aGeom%prt(ip)%seg(is)%nj-1

        gamma = aGeom%prt(ip)%seg(is)%lline%gam(j)
        alpha = aGeom%prt(ip)%seg(is)%lline%alpha(j)
        cl    = aGeom%prt(ip)%seg(is)%lline%cl(j)
        array1 = toArray(aGeom%prt(ip)%seg(is)%lline%cpnt(j))
        array2 = toArray(aGeom%prt(ip)%seg(is)%lline%uvwNW12(j)) + &
                 toArray(aGeom%prt(ip)%seg(is)%lline%uvwNW3N(j)) + &
                 toArray(aGeom%prt(ip)%seg(is)%lline%uvwLLine(j))
        write(LU_AERO_RESULTS,'(7x,9(f14.7,2x))') array1, gamma, alpha, cl, array2
      end do

    end do ! Loop over all Segments

  end do !Loop over all Parts

end subroutine writeAeroResults
```

As can easily be deduced from the code, the data written to the AeroResults file are, for all strips in all Segments in all Parts in the Geom object, the control point coordinates, the vortex strength, the local angle of attack, the liftcoefficient and the assembled “induced” velocities at the control points

3.3.2 Geometry

The GeomResults file does, like the AeroResults file, not have a final layout or a mechanism to select specific output quantities. Geometric data is written to file through a call in the main program to writeGeomResults(theGeom,...). As was also the case with the subroutine to write aerodynamics results, the source code of writeGeomResults(aGeom,...) can be easily adapted.

The core of subroutine writeGeomResults(aGeom,...), defined in the DataOutput_m module, looks like:

```
! Write the segment and wake geometry data to file
subroutine writeGeomResults(aGeom, ...)
  type(Geom), intent(in)   :: aGeom
  ...

  ! Loop over all Parts in the Geom object
  do ip=1,aGeom%nPrt

    ! Loop over all Segments in the Part object
    do is=1,aGeom%prt(ip)%nSeg

      ! Loop over all spanwise and chordwise points in the Segment
      do j=1,aGeom%prt(ip)%seg(is)%nj
        do i=1,aGeom%prt(ip)%seg(is)%ni
          array1 = toArray(aGeom%prt(ip)%seg(is)%pnt(i,j))
          write(LU_GEOM_RESULTS,'(7x,3(f14.7,2x))') array1
        end do
      end do

      ! Loop over all spanwise and chordwise points in the Segment's wake
      do j=1,aGeom%prt(ip)%seg(is)%nj
        do i=1,aGeom%prt(ip)%seg(is)%nwake%ni
          array1 = toArray(aGeom%prt(ip)%seg(is)%nwake%pnt(i,j))
```

```
          write(LU_GEOM_RESULTS,'(7x,3(f14.7,2x))') array1
        end do
      end do

      ! Loop over all spanwise and chordwise vortex rings in the Segment's wake
      do j=1,aGeom%prt(ip)%seg(is)%nwake%nj-1
        do i=1,aGeom%prt(ip)%seg(is)%nwake%ni-1
          array1 = (toArray(aGeom%prt(ip)%seg(is)%nwake%pnt(i,j)) + &
                    toArray(aGeom%prt(ip)%seg(is)%nwake%pnt(i+1,j)) + &
                    toArray(aGeom%prt(ip)%seg(is)%nwake%pnt(i,j+1)) + &
                    toArray(aGeom%prt(ip)%seg(is)%nwake%pnt(i+1,j+1)))*0.25
          gamma = aGeom%prt(ip)%seg(is)%nwake%gam(i,j)
          write(LU_GEOM_RESULTS,'(7x,4(f14.7,2x))') array1,gamma
        end do
      end do

    end do ! Loop over all Segments

  end do !Loop over all Parts

end subroutine writeGeomResults
```

As can be concluded from the code, three blocks of data are written to the GeomResults file for all Segments in all Parts in the Geom object: the Segment's coordinates, the Segment's wake coordinates and a third block with the Segment's wake vortex ring midpoint coordinates and vortex strengths. Other possible output quantities can be found in the description of the VXTypes_m module in appendix A at page 68 where all AWSM-specific types are defined.

4 VERIFICATION TESTS

To verify that AWSM solves the lifting line equation correctly, some simulations are performed using configurations that have known analytical solutions. The first series of computations is performed for an elliptical planform wing with a low aspect ratio. The second test is on a rotating blade whose twist distribution is such that its shape is part of a helix. For onset flow conditions matching the helix advance ratio, the blade should have zero lift along the entire blade.

4.1 Elliptic Wing

An elliptic wing with a span of $b = 5.0\ \mathrm{m}$, a root chord of $c_{root} = 1.0\ \mathrm{m}$, the same airfoil along the span, and a straight quarter chord line is selected. The wing planform is plotted in figure 17.

The aspect ratio of this wing is

$$AR = \frac{b^2}{S} = \frac{b^2}{(\pi b c_{root}/4)} = 6.3662$$

For an elliptic wing the liftcoefficient along the span is constant and can be determined analytically by (see [6])

$$C_l = \frac{2\pi}{1 + \frac{2}{AR}}\alpha$$

For the current test an onset wind velocity vector of $\vec{u}_{wind} = (1.0, 0.0, 0.1)\ \mathrm{m\,s^{-1}}$ is specified in the WindScenario file. The angle of attack for the elliptic wing in the XY-plane is now $\alpha = 5.7106\ ^\circ$ and for the liftcoefficient we find $C_l = 0.4765$.

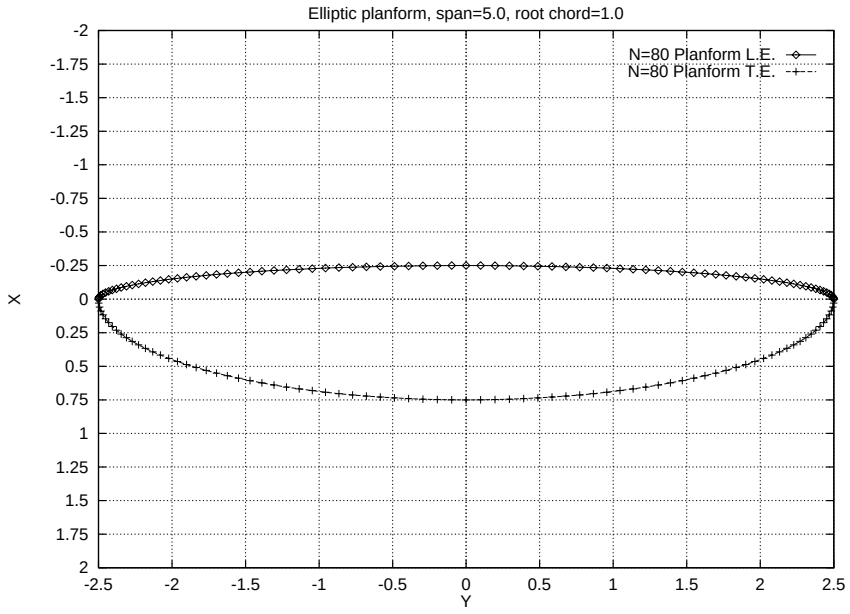

Figure 17: Elliptic wing planform

The analytical lift distribution for the elliptic wing is compared with three AWSM results in which the number of segment strips was increased from 20 to 40 to 80. The wing strips were

distributed along the span with a decreasing strip width towards the wingtips as shown in figure 17. The location of the strip control points was computed with the "full-cosine" approximation explained in section 2.2.1. The strip lift coefficients as function of local angle of attack were set in the AeroData file according to the inviscid 2D analytical value (i.e. $Cl = 2\pi\alpha$). The simulation was performed with the wake rollup feature switched off.

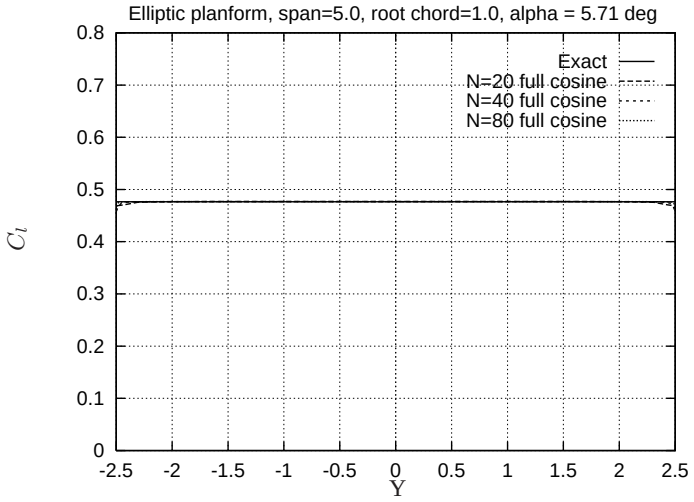

Figure 18: Elliptic wing lift distribution

In figure 18 the three computed results are shown together with the analytical solution. At this scale for the liftcoefficient, the differences between the four lines are hardly discernible.

The same results are shown in figure 19 where the range of liftcoefficients is now from 0.47 to 0.48. As can be seen, the computed results tend to the analytical solution as the number of strips is increased. The largest differences occur near the wing tips where vortex strength rapidly changes. A possible further explanation can be found in differences in wake geometry. Whereas the analytical solution assumes a planar wake in the plane of the wing, the AWSM wake is wind-convected. Especially in the tip region, the resulting kink between wing and trailing vortex lines could have a significant effect.

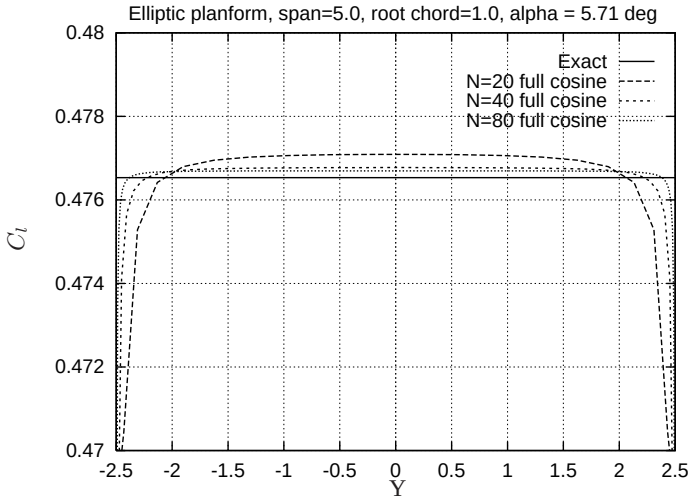

Figure 19: Elliptic wing lift distribution

As a test for the wake rollup algorithm, a 10 second simulation was performed with the wake allowed to deform through its self-induced velocity field. In figure 20 the elliptic wing and its rolled up wake are plotted. Both the expected starting vortex and the tip vortices show up unmistakably.

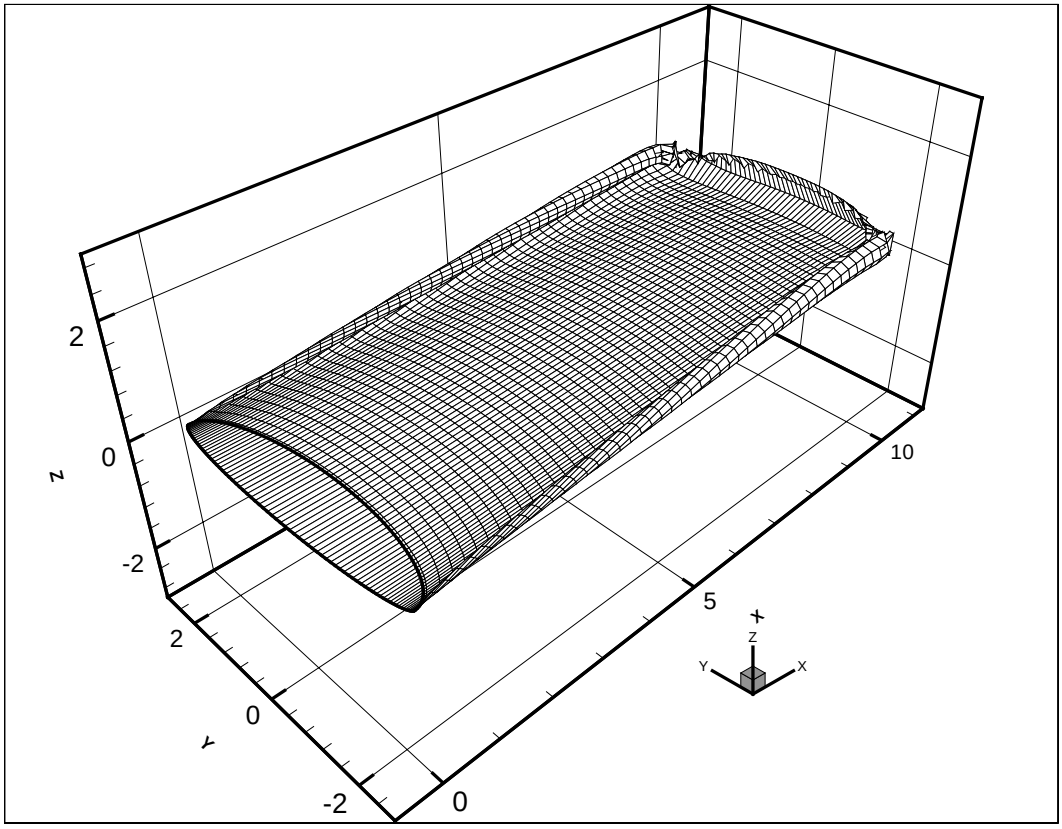

Figure 20: Elliptic wing wake geometry

4.2 Helical Rotor

For a rotor blade shaped as part of a helix, the blade sections are force free for flow conditions that match the advance ratio of the helix. This test was performed for a blade designed for a tip speed ratio of $\lambda = 10$. The blade has a constant chord of $c = 1.0\,\mathrm{m}$ and its root and tip sections are located at $r_{root} = 2.0\,\mathrm{m}$ and $r_{tip} = 10.0\,\mathrm{m}$ from the rotor axis respectively.

The rotational speed Ω was obtained from the definition of the tip speed ratio

$$\Omega = \lambda \frac{|\vec{u}_{wind}|}{r_{tip}}\,\mathrm{rad\,s^{-1}}$$

For the current test a wind velocity of $\vec{u}_{wind} = (10.0, 0.0, 0.0)\mathrm{ms}^{-1}$ was selected and specified in the WindScenario file. The corresponding rotational speed of $\Omega = 572.9578^{\circ}\,\mathrm{s}^{-1}$ was used in the GeomScenario file. The strip lift coefficients were set in the AeroData file according the inviscid 2D analytical value of $C_l = 2\pi\alpha$.

For the same rotational speed two additional AWSM runs were performed; one for a 10 percent higher and one for a 10 percent lower wind velocity. In the first case the rotor blade should act

as a wind turbine blade and in the second case the blade should operate in “propeller mode”. In both cases wake rollup was switched off.

The three results are shown in figure 21. As can be seen the force free rotor blade case is reproduced correctly. For the two other test cases no analytical results exist but the expected behavior is displayed.

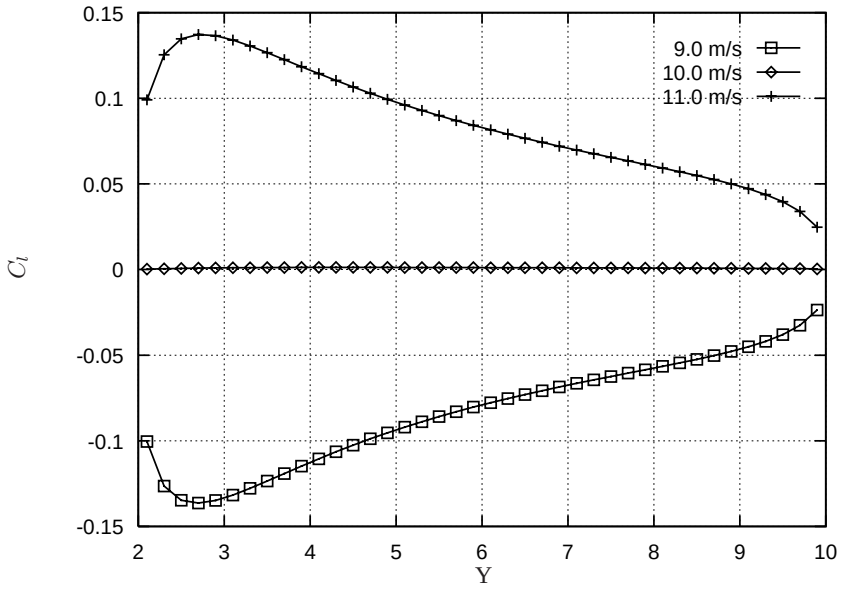

Figure 21: Helical rotor, blade lift distribution

The generated wake after a 10.0 second simulation is shown in figure 22 and a closeup view is shown in figure 23.

For the 10 percent higher onset wind velocity, i.e. the “wind turbine mode”, the simulation was repeated with wake rollup allowed during the 2.0 seconds simulation time. The resulting wake geometry is shown in figure 24. The influence of the starting vortex and the blade root and tip vortices on the wake geometry can be noticed.

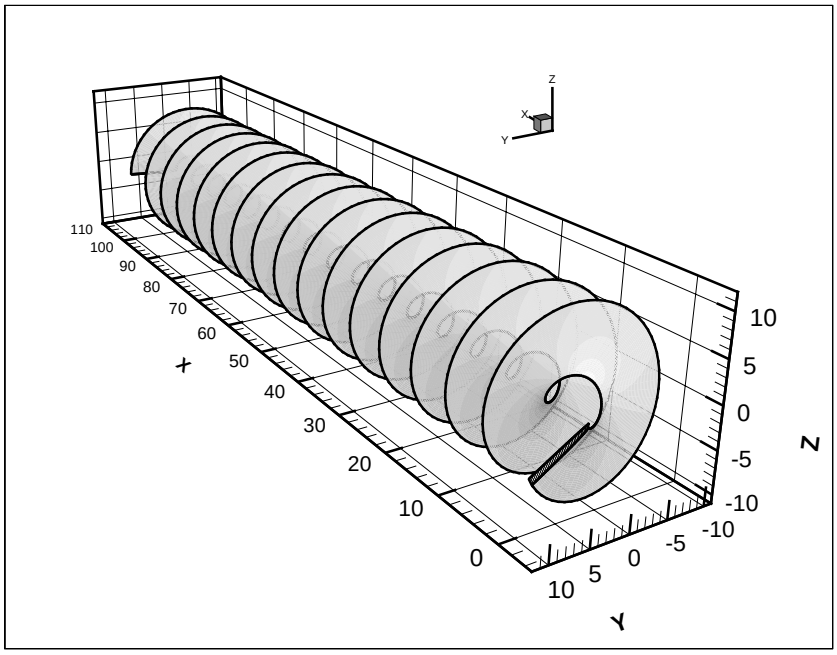

Figure 22: Helical rotor, wake geometry

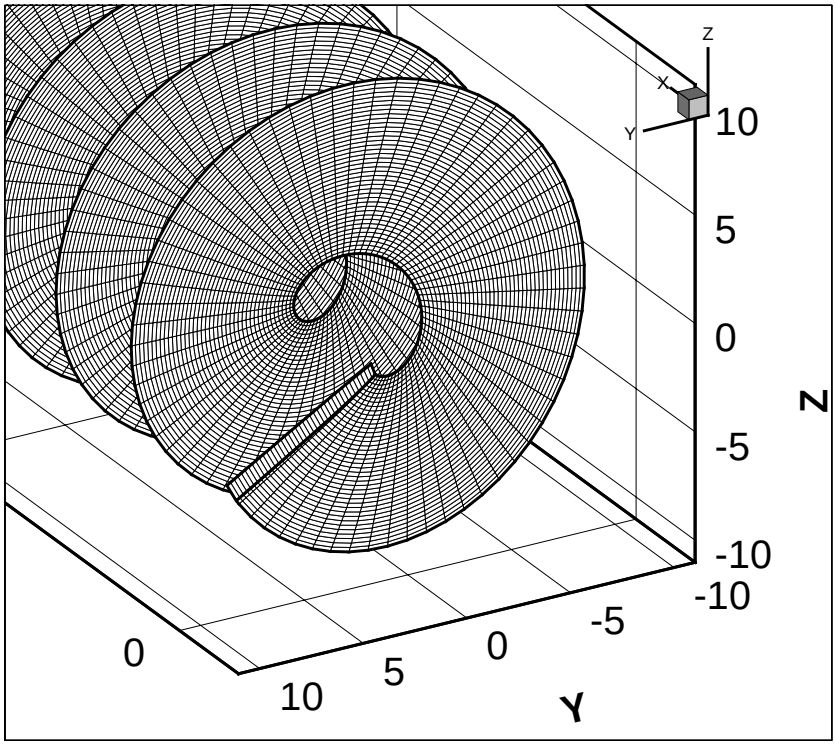

Figure 23: Helical rotor, wake geometry detail

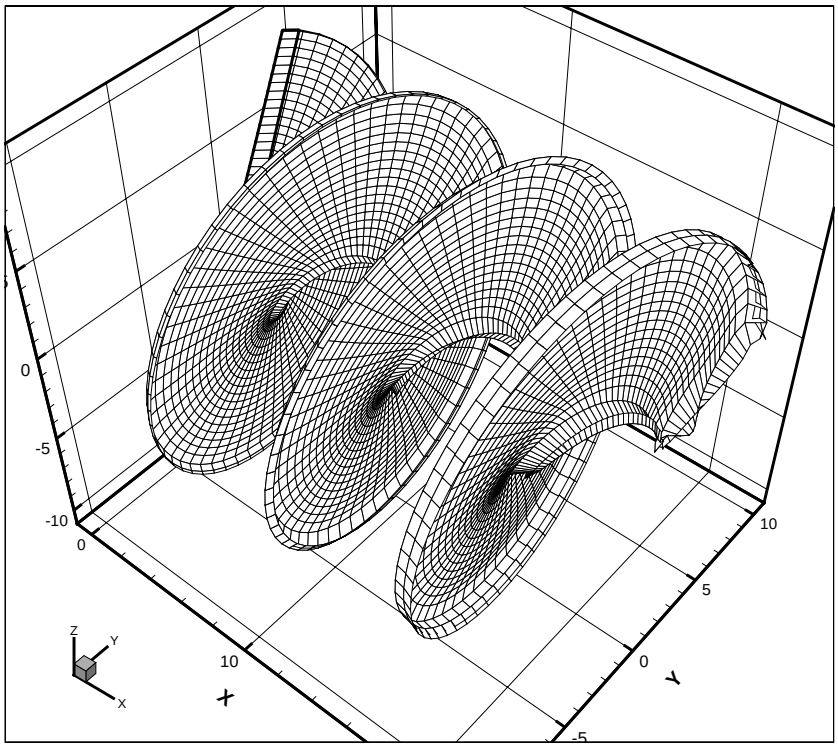

Figure 24: Helical rotor, wake rollup aft view

5 PROGRAM USAGE

In this chapter running a simulation of a flow problem with the AWSM computer program is explained. An AWSM run starts with a series of questions at the command line and the responses given by the user set some parameters that control the flow of the simulation.

In the first section the special command line features are discussed and in the following section one by one the question blocks that will be encountered will be explained.

The next section shows a simple, valid set of input files and in a subsequent section results are shown for a realistic configuration.

5.1 Command Line Input

5.1.1 Introduction

At the start of an AWSM run a number of questions are posed. With the help of an actual AWSM question the format used is explained:

```
Give Trace level [0:5] {/bfk} <CR>=2 :
```

The command line prompt is right after the ":" prompt chararacter. Between the actual question "Give Trace level" and the prompt character some hints are given.

Within rectangular braces `[ ]` the range of valid answers is given for questions that ask for a numeric value. In this example the user can enter only integer values between 0 and 5. If there is no minimum or maximum, the first or second value is left out respectively. The version for the input of a floating point number would be `[0.000000:5.000000]`.

Between curly braces { } the allowed special "slash handling" characters are given. The forward slash character followed by one of the subsequent characters will result in special code behavior. The answer is interpreted case sensitive. In table 4 the AWSM behavior is shown.

Table 4: AWSM slash handling behavior

/b	:	go **b**ack to the previous question
/f	:	go **f**orward to the end of the questions
/k	:	**k**ill this run

This means that giving multiple "`/b`" answers can take the user back to the start of the questions. A "`/f`" will skip all questions and assume default answers. Entering "`/k`" will kill the AWSM run.

The value after "`<CR>=`" is the default answer that will be assumed if an empty "carriage return" is given. In the example question a trace level of 2 is assumed on an empty line return.

In case the entered input cannot be interpreted, an error message is generated and the question is repeated. Answering the previous example question with "`3.14159`", "`8`" or "`in.dat`" for example will respectively give the error messages

```
*** Error, not an integer number ***
*** Error, value out of range ***
*** Error, not an integer number ***
```

and entering "/d" or "/F" will result in the error message

```
*** Error, invalid flag character ***
*** Valid flag characters are: /bfk ***
```

For wrongfully entered data on a floating point number request the error message is:

```
*** Error, not a floating point number ***
```

and a wrong answer to a "yes/no" or "on/off" type of question gives

```
*** Error, answer Y(ES)/T(RUE)/ON/1 or N(O)/F(ALSE)/OFF/0 ***
```

5.1.2 AWSM Command Line Questions

The AWSM command line questions are grouped into logically related question blocks. These blocks will be listed successively and discussed in more detail with the help of an actual session:

```
Give run input identification {/fk} <CR>= :01
Give run output identification {/bfk} <CR>= :_tst
```

The run input identification string that is entered, in this case "01", will be inserted in the default input file names just before the file extension. The resulting default filenames will be "geom01.dat", "geomscenario01.dat" etc.

The run output identification string will be inserted in the output file names like "log_tst.dat" and "aeroresults_tst.dat". The default run input and output identification strings are empty.

```
Give Geom filename {/bfk} <CR>=geom01.dat :<return>
Give AeroLink filename {/bfk} <CR>=aerolink01.dat :<return>
Give GeomScenario filename {/bfk} <CR>=geomscenario01.dat :<return>
Give WindScenario filename {/bfk} <CR>=windscenario01.dat :<return>

Give Log filename {/bfk} <CR>=log_tst.dat :<return>
Give GeomSnapshot filename {/bfk} <CR>=geomsnapshot_tst.dat :<return>
Give WindSnapshot filename {/bfk} <CR>=windsnapshot_tst.dat :<return>
Give GeomResults filename {/bfk} <CR>=geomresults_tst.dat :<return>
Give AeroResults filename {/bfk} <CR>=aeroresults_tst.dat :<return>
```

In this block of questions the names for the AWSM input and output files are requested. In the example session the default filenames were selected by giving empty line returns.

```
Give Trace level [0:5] {/bfk} <CR>=2 :<return>
Trace to stdout? {/bfk} <CR>=YES :<return>
Trace to logfile? {/bfk} <CR>=YES :<return>
```

The default trace level 2 (i.e. WARNING, see table 9 at page 103) is selected and data will be logged to the screen (i.e. standard output) and to the log file specified earlier ("log_tst.dat").

```
Give simulation start time {/bfk} <CR>=0.000000 :<return>
Give simulation end time [0.000000:] {/bfk} <CR>=1.000000 :10.0
Give nr. of timesteps [1:] {/bfk} <CR>=1 :1000
```

The start and end time of the simulation are set to 0.0 and 10.0 seconds respectively. This 10 second simulation is subdivided into 1000 timesteps, so each timestep will take 0.01 second. The number of timesteps should be given a value that will guarantee an adequate representation of the continuous problem.

```
Give aero output start timestep nr. {/bfk} <CR>=1 :100
Give aero output end timestep nr. {/bfk} <CR>=1000 :<return>
Give aero output timestep increment [1:] {/bfk} <CR>=900 :100
```

The first and last timestep number at which the program will write aerodynamics results to the file specified earlier in the session (i.e. “aeroresults_tst.dat”) are set to 100 and 1000. The timestep interval at which data is written in between is given also and in this example data will be written at timesteps 100, 200, 300, ... and 1000 which corresponds to simulation times at 1.0, 2.0, 3.0,... and 10.0 seconds.

```
Give geom output start timestep nr. {/bfk} <CR>=1000 :<return>
Give geom output end timestep nr. {/bfk} <CR>=1000 :<return>
Give geom output timestep increment [1:] {/bfk} <CR>=1 :<return>
```

Like the previous block these questions determine at what timesteps the geometric data will be written to file (in this case “geomresults_tst.dat”). Only for the final timestep the (wake-) geometry will be stored in this case. The geometric information contains a vast amount of data so it is advised to write geometric output only sporadically.

```
Centered control points? {/bfk} <CR>=NO :<return>
```

The empty return answer given (i.e. “NO”) will pull control points towards the narrowest neigboring segment strip as was discussed in section 2.2.1. Answering “YES” would have set them at the center of the vortex lines.

```
Give maximum nr. of streamwise wakepoints [2:1001] {/bfk} <CR>=1001 :<return>
Give nr. of free streamwise wakepoints [0:1001] {/bfk} <CR>=1001 :0
Linear (1) or smooth (2) vortex cut-off velocities [1:2] {/bfk} <CR>=2 :<return>
Give wake rollup vortex cut-off radius [0.000000:] {/bfk} <CR>=0.5000000-01 :0.001
```

The maximum number of streamwise wakepoints determines how many shed vortices behind each strip will take effect in the influence of the wake. In this case, wake panels older than 1000 timesteps are discarded altogether. The value given puts some maximum on the required CPU time per timestep.

The number of free streamwise wakepoints determines how many of the most recently shed vortices are allowed to roll up through wake self-influence. Because the value 0 was entered, the wake will only be deformed by wind velocities in this case. It should be noted that the value given has quadratic influence on computing time per timestep.

Linear and smooth “induced velocity” functions for the wake vortex lines were discussed in section 2.1.3. The default (i.e. “smooth”) is recommended. A sensible cut-off radius value is anything between 0.0 and 0.1. A larger wake rollup cut-off radius will result in a smoother velocity field “induced” by the wake and the lifting line vortex rings. This will result in a smoother wake geometry in case the wake is allowed to roll up through self-influence.

```
Give lift force vortex cut-off radius [0.000000:] {/bfk} <CR>=0.1000000E-03 :<return>
Give circulation convergence criterion [0.000000:] {/bfk} <CR>=0.5000000E-02 :<return>
```

The lift force vortex cut-off radius is used in the computation of the velocity vectors, “induced” by the wake and the lifting line vortex rings, at the segment strip control points. The cut-off radius value for the lifting line “induced velocities” can be set at a smaller value than that for the wake. The default value of 0.0001 is recommended.

The circulation convergence criterion Γ_{crit}, discussed in section 2.1.5 at page 10, is set to the default answer of 0.005 by simply hitting the <return> key.

```
Give relaxation factor for circulation increment [0.01:1.0] {/bfk} <CR>=0.8 :<return>
Give extra relaxation factor for segment outer 10% [0.01:1.0] {/bfk} <CR>=0.4 :<return>
Give relaxation factor scaling [0.5000000:1.000000] {/bfk} <CR>=0.9500000 :<return>
Give maximum change in liftcoefficient [0.1000000E-01:] {/bfk} <CR>=0.05 :<return>
Give minimum relaxation factor [0.1000000E-01:] {/bfk} <CR>=0.2000000 :<return>
Give maximum relaxation factor [0.2000000:1.000000] {/bfk} <CR>=1.000000 :<return>
Give max nr. of iterations for structural equilibrium [1:] {/bfk} <CR>=10 :<return>
Give max nr. of iterations for aerodynamic convergence [1:] {/bfk} <CR>=400 :<return>
```

This last block of (slightly edited) questions basically restricts the allowed change in vortex strength during the iterative determination of the lifting line vorticity values discussed in section 2.1.5, page 10. In the current example all default answers were selected.

In this example, the relaxation factor θ for circulation increment (see equation (19), page 12) is given the default value of $\theta = 0.8$. This value is used for all segment strips.

In addition, the segment outer 10% is given an extra under-relaxation. The relaxation factor specified in the previous question (i.e. 0.8) is scaled down by a factor decreasing linearly from 1.0 to the given value (i.e. 0.4) at the ends of the segment. In this case, for the end strips the relaxation factor $\theta = 0.32$ will be used (i.e. 0.4 times 0.8).

The relaxation factor scaling, here given the default value of 0.95, is applied each new iteration to the old θ relaxation value: $\theta_{new} = 0.95\theta_{old}$. However, the relaxation factor is restricted to the interval given by the minimum and maximum relaxation factor values, in this example respectively set to 0.2 and 1.0.

The maximum number of iterations for structural equilibrium is currently not used. It is included for the situation that a structural dynamics simulation module is coupled to AWSM.

The maximum number of iterations for aerodynamic convergence limits the number of cycles in the non-linear algorithm explained in section 2.1.5.

Because trace data to standard output was activated, the given input values are echoed back immediately after the last answer. The trace info for this example was already listed in section G.

5.2 Example Input Files

The example input files in this section should give a more practical description of what input files actually look like than the description given in appendix E. The input files are taken from the helical rotor testcase described in section 4.2. A run input and output identification of “01” will be set for the simulation so all files will be named “xxxx01.dat”. The input files themselves contain comment lines clarifying possible unclear issues.

5.2.1 Geom File

The definition of the geometry for the helical rotor blade is located in an example file named “geom01.dat”. This rotor is designed to rotate about the X-axis.

```
! File : geom01.dat
```

```
! Helical rotor definition
! Root radius =      2.0000000
! Root chord  =      1.0000000
! Tip radius  =     10.0000000
! Tip chord   =      1.0000000
! Tip-Speed-Ratio  = 10.00
! Number of strips = 40
! Reference area =      8.0000000
! Aspect ratio   =      8.0000000

<Geom>
  <Id> "Helical rotor" </Id>

  <Part>
    <Id> "Helical rotor blade" </Id>
    <IdNr> 1 </IdNr>
    <Segment>
      <Id> "Blade segment 40 strips" </Id>
      <IdNr> 1 </IdNr>
      <Coordinates>
      ! Only leading edge and trailing edge points are specified here
    -0.1118034     2.0000000     0.2236068     0.3354102     2.0000000    -0.6708204
    -0.1034507     2.2000000     0.2275916     0.3103522     2.2000000    -0.6827749
    -0.0961538     2.4000000     0.2307692     0.2884615     2.4000000    -0.6923077
    -0.0897448     2.6000000     0.2333364     0.2692343     2.6000000    -0.7000092
    -0.0840841     2.8000000     0.2354355     0.2522523     2.8000000    -0.7063064
    -0.0790569     3.0000000     0.2371708     0.2371708     3.0000000    -0.7115125
    -0.0745687     3.2000000     0.2386200     0.2237062     3.2000000    -0.7158600
    -0.0705416     3.4000000     0.2398414     0.2116247     3.4000000    -0.7195241
    -0.0669110     3.6000000     0.2408795     0.2007329     3.6000000    -0.7226384
    -0.0636233     3.8000000     0.2417686     0.1908700     3.8000000    -0.7253059
    -0.0606339     4.0000000     0.2425356     0.1819017     4.0000000    -0.7276069
    -0.0579051     4.2000000     0.2432016     0.1737154     4.2000000    -0.7296047
    -0.0554053     4.4000000     0.2437832     0.1662158     4.4000000    -0.7313496
    -0.0531074     4.6000000     0.2442941     0.1593222     4.6000000    -0.7328823
    -0.0509886     4.8000000     0.2447451     0.1529657     4.8000000    -0.7342353
    -0.0490290     5.0000000     0.2451452     0.1470871     5.0000000    -0.7354355
    -0.0472118     5.2000000     0.2455016     0.1416355     5.2000000    -0.7365048
    -0.0455223     5.4000000     0.2458205     0.1365669     5.4000000    -0.7374615
    -0.0439477     5.6000000     0.2461069     0.1318430     5.6000000    -0.7383207
    -0.0424767     5.8000000     0.2463650     0.1274302     5.8000000    -0.7390951
    -0.0410997     6.0000000     0.2465985     0.1232992     6.0000000    -0.7397954
    -0.0398081     6.2000000     0.2468103     0.1194243     6.2000000    -0.7404308
    -0.0385942     6.4000000     0.2470030     0.1157827     6.4000000    -0.7410090
    -0.0374513     6.6000000     0.2471789     0.1123540     6.6000000    -0.7415366
    -0.0363735     6.8000000     0.2473398     0.1091205     6.8000000    -0.7420194
    -0.0353553     7.0000000     0.2474874     0.1060660     7.0000000    -0.7424621
    -0.0343921     7.2000000     0.2476231     0.1031763     7.2000000    -0.7428692
    -0.0334795     7.4000000     0.2477481     0.1004384     7.4000000    -0.7432443
    -0.0326136     7.6000000     0.2478636     0.0978409     7.6000000    -0.7435907
    -0.0317911     7.8000000     0.2479704     0.0953732     7.8000000    -0.7439112
    -0.0310087     8.0000000     0.2480695     0.0930261     8.0000000    -0.7442084
    -0.0302636     8.2000000     0.2481615     0.0907908     8.2000000    -0.7444844
    -0.0295532     8.4000000     0.2482471     0.0886597     8.4000000    -0.7447412
    -0.0288752     8.6000000     0.2483268     0.0866256     8.6000000    -0.7449805
    -0.0282274     8.8000000     0.2484013     0.0846823     8.8000000    -0.7452039
    -0.0276079     9.0000000     0.2484709     0.0828236     9.0000000    -0.7454128
    -0.0270148     9.2000000     0.2485361     0.0810444     9.2000000    -0.7456083
    -0.0264465     9.4000000     0.2485972     0.0793395     9.4000000    -0.7457917
    -0.0259015     9.6000000     0.2486546     0.0777046     9.6000000    -0.7459638
    -0.0253784     9.8000000     0.2487085     0.0761353     9.8000000    -0.7461256
    -0.0248759    10.0000000     0.2487593     0.0746278    10.0000000    -0.7462779
      </Coordinates>
      <Ni> 2 </Ni>
      <Nj> 41 </Nj>
    </Segment>

    <Marker>
      <Id> "Blade root marker" </Id>
      <IdNr> 1 </IdNr>
      <Location> 0.0  0.0  0.0 </Location>
```

```
      <Vector1> 1.0  0.0  0.0 </Vector1>
      <Vector2> 0.0  1.0  0.0 </Vector2>
    </Marker>
  </Part>

  ! Part only containing Marker that will rotate
  <Part>
    <Id> "Hub" </Id>
    <IdNr> 2 </IdNr>
    <Marker>
      <Id> "Blade attachment marker" </Id>
      <IdNr> 1 </IdNr>
      <Location> 0.0  0.0  0.0 </Location>
      <Vector1> 1.0 0.0  0.0 </Vector1>
      <Vector2> 0.0 1.0  0.0 </Vector2>
    </Marker>
  </Part>

  ! Rotation of the ParentMarker will propagate to ChildPart's geometry
  <Link>
    <Id> "Blade - Hub connection" </Id>
    <IdNr> 1 </IdNr>
    <ParentPart> 2 </ParentPart>
    <ParentMarker> 1 </ParentMarker>
    <ChildPart> 1 </ChildPart>
    <ChildMarker> 1 </ChildMarker>
  </Link>

</Geom>
```

The 1-bladed rotor geometry is specified as an array of leading edge and trailing edge points for blade sections (at constant Y-values) running from root to tip. The resulting surface normal in this case points in positive Z-direction.

In the Geom file two Parts (resembling the rotor and the hub) are defined, each containing a Marker. The two Markers already are positioned at the same point in space and have their directional vectors point in the same direction. The Markers are linked to each other and the "hub" Marker will act as parent.

In the GeomScenario file the "hub" Marker will be subjected to a specified rotation. This will enforce the (child) "rotor blade" Marker, and thereby the rotor geometry, to be rotated and/or translated to match it's parent Marker.

5.2.2 AeroData File

The example AeroData file "aerodata01.dat" contains only one Airfoil object that specifies the 2D inviscid liftcurve $C_l = 2\pi\alpha$.

```
! File : aerodata01.dat
! File containing aerodynamic coefficients

<AeroData>
  <Id> "This is an example dataset" </Id>

  <Airfoil>
    <Id> "Symmetric Airfoil" </Id>
    <IdNr> 1 </IdNr>
    <Polar>
      <Id> "Inviscid" </Id>
      <IdNr> 1 </IdNr>
      <Re> 2100000 </Re>
      <Ma> 0.1 </Ma>

      ! Cl = 2*pi*alpha
      ! Angle of attack (in degrees) versus liftcoefficient table
```

```
      <Alpha-Cl>
       -180.000   -19.739209
        180.000    19.739209
      </Alpha-Cl>

      ! Angle of attack (in degrees) versus dragcoefficient table
      <Alpha-Cd>
       -180.000    0.0
        180.000    0.0
      </Alpha-Cd>

      ! Angle of attack (in degrees) versus pitching moment coefficient table
      <Alpha-Cm>
       -180.000    0.0
        180.000    0.0
      </Alpha-Cm>
    </Polar>
  </Airfoil>

</AeroData>
```

5.2.3 AeroLink File

The AeroLink example file “aerolink01.dat” links all Segment strips from “rotor blade” Part 1 in file “geom01.dat” to the same Polar in the “aerodata01.dat” file. As is also mentioned in the listing itself, the StripLink objects for strip numbers 2 to 39 are left out because they are, except for the strip number, the same as those for strip numbers 1 and 40.

```
! File : aerolink01.dat
! File containing the links between of Segment strips and
! their accompanying aerodynamic coefficients.

<AeroLink>
  <Id> "This is an example <AeroLink> dataset" </Id>

  ! Link ALL strips in a Segment to an Airfoil Polar
  <StripSet>
    <PartIdNr> 1 </PartIdNr>
    <SegmentIdNr> 1 </SegmentIdNr>

    <StripLink>
      <StripNr> 1 </StripNr>
      <AeroDataFile> "aerodata01.dat" </AeroDataFile>
      <AirfoilIdNr> 1 </AirfoilIdNr>
      <PolarIdNr> 1 </PolarIdNr>
    </StripLink>

!  StripLink objects for StripNr 2 - 39 left out
!  These are, except for the StripNr, the same as those for StripNr's 1 and 40

    <StripLink>
      <StripNr> 40 </StripNr>
      <AeroDataFile> "aerodata01.dat" </AeroDataFile>
      <AirfoilIdNr> 1 </AirfoilIdNr>
      <PolarIdNr> 1 </PolarIdNr>
    </StripLink>
  </StripSet>

</AeroLink>
```

5.2.4 GeomScenario File

The GeomScenario example file “geomscenario01.dat” specifies the rotation of Marker 1 from “hub” Part 2 in file “geom01.dat” about the X-axis.

```
! File: geomscenario01.dat
! File containing rotations and/or translations of specified Markers at
! discrete instances of time.
! This scenario is for a helical rotor blade with tip-speed-ratio TSR=10
! Wind velocity is selected to be 10 m/s

! Rotational velocity:
!  Omega = TSR*Uref/Rtip = 10*10/10 = 10 rad/s = 10*180/pi deg/s = 572.9578 deg/s
! Time for 1 full rotation:
!      T = 2*pi/Omega = 2*pi/10 = 0.6283185 s
! Rotations per minute:
!      n = 60/T = 60*Omega/(2*pi) = 95.492966 rpm

! Timesteps of 0.01 s give a blade rotation of 5.729578 deg
! After 10 seconds (= 1000 timesteps) there are 10/T = 15.915494 rotations made

<GeomScenario>
! Rotation about X-axis of the Hub part's Marker (i.e. Marker 1 of Part 2)

  <Id> "Rotation about X-axis" </Id>

  <MarkerTimeSeries>
    <PartIdNr> 2 </PartIdNr>
    <MarkerIdNr> 1 </MarkerIdNr>
    <InterpolationType> "LINEAR" </InterpolationType>
    <Extrapolation> YES </Extrapolation>

    <MarkerTimeInstance>
      <Time> 0.0 </Time>
      <Translation> 0.0  0.0  0.0 </Translation>
      <RotationAxis> 1.0 0.0 0.0 </RotationAxis>
      <RotationAngle> 0.0 </RotationAngle>
    </MarkerTimeInstance>

    <MarkerTimeInstance>
      <Time> 100.0 </Time>
      <Translation> 0.0  0.0  0.0 </Translation>
      <RotationAxis> 1.0 0.0 0.0 </RotationAxis>
      <RotationAngle> 57295.78 </RotationAngle>
    </MarkerTimeInstance>
  </MarkerTimeSeries>
</GeomScenario>
```

5.2.5 WindScenario File

The WindScenario example file "windscenario01.dat" specifies a constant wind speed of $10\mathrm{ms}^{-1}$ in x-direction.

```
! File: windscenario01.dat
! Constant wind velocity vector of (10, 0, 0) m/s

<WindScenario>
  <Id> "Constant wind" </Id>
  <InterpolationType> "LINEAR" </InterpolationType>
  <Extrapolation> YES </Extrapolation>

  <WindTimeInstance>
    <Time> 0.0 </Time>
    <WindUVW> 10.0  0.0  0.0 </WindUVW>
  </WindTimeInstance>

  <WindTimeInstance>
    <Time> 100.0 </Time>
    <WindUVW> 10.0  0.0  0.0 </WindUVW>
  </WindTimeInstance>
</WindScenario>
```

5.3 Example Program Run

For the example run the 3-bladed rotor windtunnel model from the MEXICO project was taken. The rotor design conditions are shown in table 5 together with the AWSM simulation input parameters that differ from the default values. Per timestep the angular displacement of the rotor is 12.732° and at the end of the simulation, after 200 timesteps, over 7 full blade rotations are made. At that time the wake will extend approximately $14.7\ \mathrm{m}$ downstream.

Table 5: MEXICO simulation run parameters

Description	Value
Rotor diameter	$D = 4.50\ \mathrm{m}$
Tip speed ratio	$\lambda = 6.80$
Tip velocity	$U_{tip} = 100.00\ \mathrm{m\,s^{-1}}$
Wind velocity	$U_{wind} = 14.70\ \mathrm{m\,s^{-1}}$
Rotational speed	$\Omega = 44.44\ \mathrm{rad\,s^{-1}}$
Simulation start time	$t_{start} = 0.0\ \mathrm{s}$
Simulation end time	$t_{end} = 1.0\ \mathrm{s}$
Number of timesteps	$N_t = 200$
Timestep	$\Delta t = 0.005\ \mathrm{s}$
Wake rollup cut-off radius	$\delta = 0.01$
Number of free wake points	$N_{free} = 100$

In figure 25 the airfoil lift curves used in the simulation are plotted. These liftcurves were linked in the AeroLink input file to the blade strips in accordance with the distribution in table 6.

Table 6: MEXICO rotor blade airfoil locations

Radius [m]	$\eta = r/r_{tip}$	Strip nrs.	Airfoil
0.450 - 1.125	0.2 - 0.5	1 - 9	DU91-W2-250
1.125 - 1.575	0.5 - 0.7	10 - 15	RISOE A221
1.575 - 2.250	0.7 - 1.0	16 - 24	NACA 64418

The locally occurring angle of attack α at a blade's quarter chord line at time $t = 1.0\ \mathrm{s}$ is plotted in figure 26. The kink in local angle of attack at $\eta = 0.70$ is due to a jump in blade twist. This jump was introduced in the design of the geometry to compensate for the difference in liftcurves for airfoils "RISOE A221" and "NACA 64418" around $\alpha = 0^\circ$ (see figure 25).

In figures 27 and 28 the liftcoefficient C_l and vortex strength Γ distributions over the blade span are shown. As expected a sharp drop in function value occurs in the blade root and tip regions.

The wake was allowed to rollup during the simulation. The wake's geometry at the end of the AWSM run is shown in front view in figure 29 and a rear view is given in figure 30. For clarity the wake geometry of only one blade is shown. The vortex strength Γ of the wake is visualised through the use of colorbands.

The starting vortex that was created in the first timestep can be seen in figure 30 to be heavily distorted.

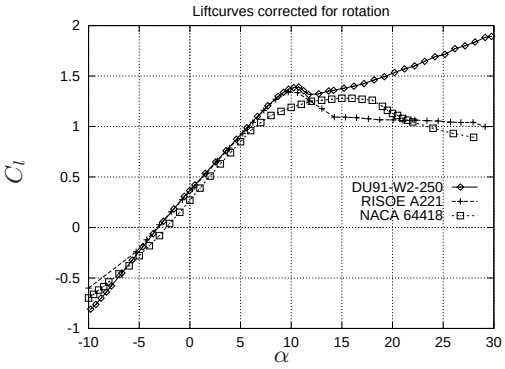

Figure 25: Lift curves

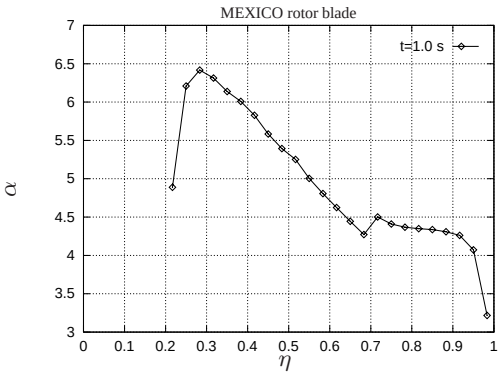

Figure 26: Angle of attack distribution

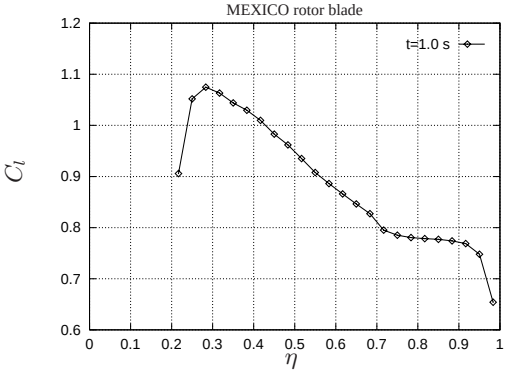

Figure 27: Lift distribution

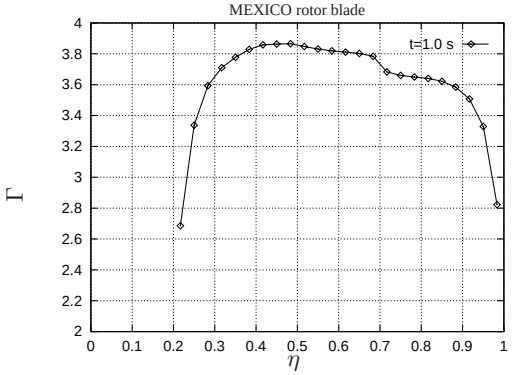

Figure 28: Vortex strength distribution

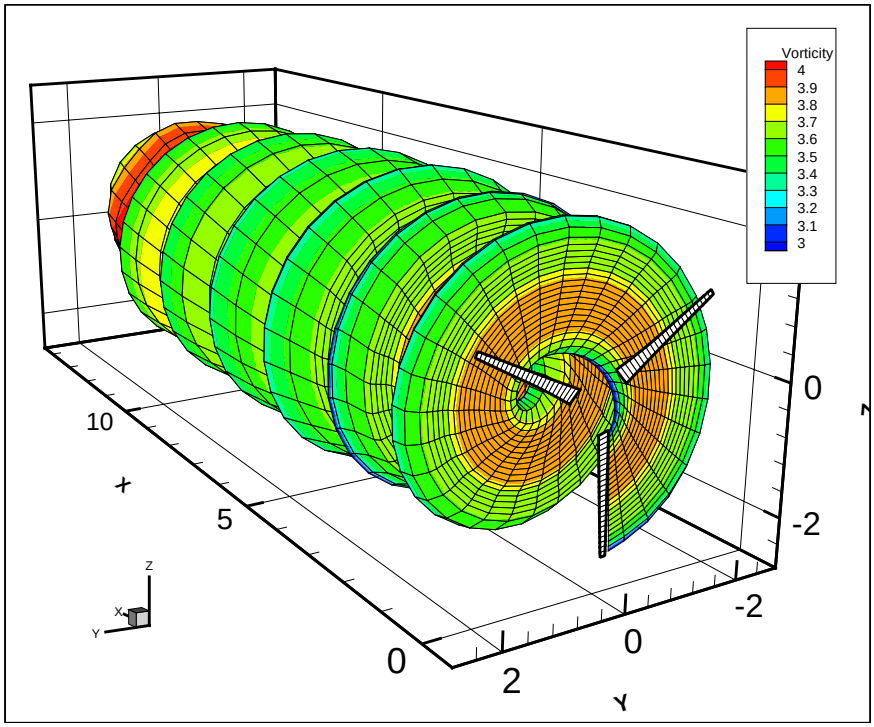

Figure 29: Blade 3 wake geometry front view

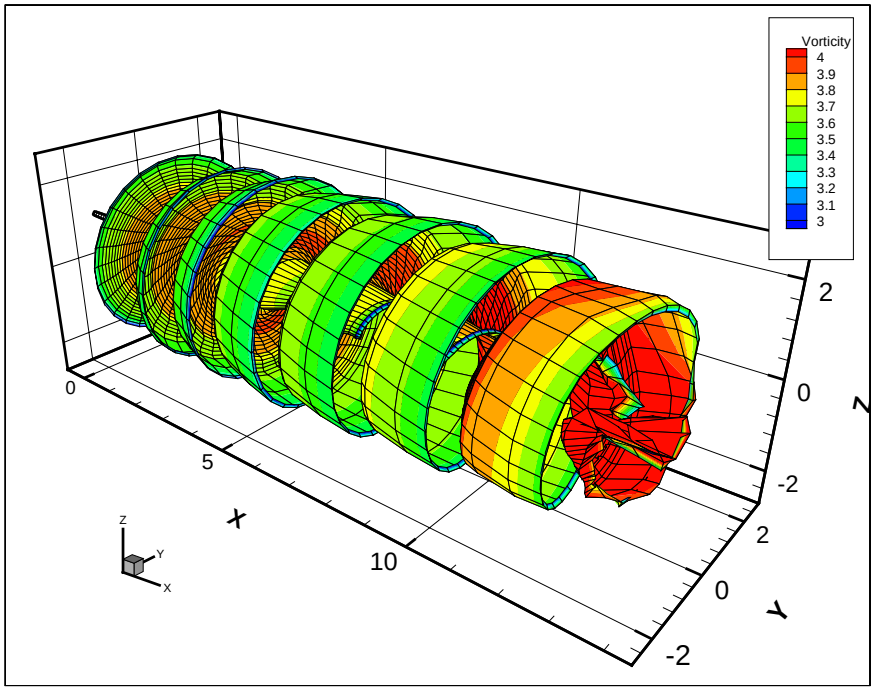

Figure 30: Blade 3 wake geometry aft view

6 CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

A new Fortran90 wind turbine aerodynamics simulation code, designated AWSM, is developed. The model is based on non-linear generalised lifting-line theory in which volume vorticity is lumped into vortex lines. Rotor wake rollup and the non-linear aerodynamic characteristics of the blade sections are incorporated in this model.

The mathematical and software related fundamentals that form the backbone of the AWSM computer code are described. This should greatly facilitate further enhancements and code extensions.

Comparison of simulation runs with analytically known test cases verified the correctness of the AWSM code. Wake rollup deformation tests showed the expected behavior.

The practical application of the AWSM simulation code was demonstrated with the help of an actual test run.

6.2 Recommendations

The following list is a collection of recommendations collected during the course of the AWSM code development:

1. Assert the quality of AWSM results through a comparison with experimental data. It is suggested to use the forthcoming results from the MEXICO project for this.
2. Investigation of the influence of AWSM simulation parameters on the computed results:
    - Investigation of the influence of wake rollup through its self-induced flowfield and how it compares to the influence of wind convection. This should also include cases with yaw misalignment.
    - Investigation of the minimum wake length in order to ensure an adequate computation of wake effects. What is the minimum number of streamwise wake panels that can be chosen without serious result deterioration.
    - Investigation of flow conditions with partially stalled blades. This includes the possible occurrence of lift-hysteresis.
    - Investigation if the influence of the parameter that controls the length of the first wake strip attached to the trailing edge.
3. Implementation of functionality enhancements or extensions:
    - Implement the capability to specify output quantities from the command line or through specification in an input file. The use of Fortran90 direct access files instead of the currently used ASCII based output files is then an obvious next step.
    - Implement the effects of more physically correct wind fields.
    - Implement calculation of bending moments, torque, etc.
    - Additional to specifying rotation angles in the GeomScenario and GeomSnapshot files: allow specification of rotational angle and speed $\theta = \theta_0 + \frac{d\theta}{dt}$

- Incorporation of the effects of geometry deformations and deformation velocities as sketched in appendix F at page 101.
- Include the option to take into account the effects of non-homogeneous onset wind calculated by an advanced wind field simulator.

4. Implemention of algorithm improvements that can speed up AWSM run times:

   - Use of the more sophisticated Newton-Raphson method to drive the difference between the lift force computed from the user given aerodynamic tables and the lift force associated with the lifting-line vortex strength to zero.
   - The algorithm to calculate the required update in vortex strengths iterates over all strips in a segments. At the moment no use is made of already computed new vortex strengths guesses of neighboring strips.
   - Use of Fortran90 direct access files for aerodynamic and geometric output quantities.

REFERENCES

[1] M. Drela, *Assorted Views on Teaching of Aerodynamics*, AIAA-98-2792, 1998

[2] I.D. Faux, M.J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd, 1979.

[3] A. van Garrel, *Requirements for a Wind Turbine Aerodynamics Simulation Module*, ECN-C–01-099, 1999.

[4] H.W.M. Hoeijmakers, *Panel Methods for Aerodynamic Analysis and Design*, In: AGARD-FDP/VKI Special Course on Engineering Methods in Aerodynamic Analysis and Design of Aircraft, AGARD Report R-783, NLR TP 91404 L, 1991.

[5] B. Hunt, *The Panel Method for Subsonic Aerodynamic Flows: A Survey of Mathematical Formulations and Numerical Models and an Outline of the New British Aerospace Scheme*, VKI Lecture Series 1978-4, 1978.

[6] J. Katz, A. Plotkin, *Low-Speed Aerodynamics: From Wing Theory to Panel Methods*, McGraw-Hill, 1991.

[7] W.F. Phillips, D.O. Snyder, *Modern Adaptation of Prandtl's Classic Lifting-Line Theory*, Journal of Aircraft, Vol.37, No.4, pp.662-670, 2000

[8] P.G. Saffman, *Vortex Dynamics*, Cambridge University Press, 1992

[9] H. Späth, *Spline-Algorithmen zur Konstruktion glatter Kurven und Flächen*, R. Oldenbourg Verlag, 1973

A FORTRAN90 MODULES

AeroData_m

```
!******************************************************************************
!* FILE : AeroData_m.f90
!*
!* PROGRAM UNIT : module AeroData_m
!*
!* PURPOSE : The AeroData_m module handles the creation of an "AeroData" object
!*           that contains airfoil aerodynamic coefficients like lift, drag
!*           and pitching moment.
!*
!* MODULES USED : Types_m, VXTypes_m, InputFile_m, Trace_m, Utils_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2002   A. van Garrel   Initial coding
!*
!******************************************************************************
!* SHORT DESCRIPTION :
!*
!*   Public subroutines are:
!*
!*     ! Create an "AeroData" object from the given "InputFile"
!*     subroutine nextAeroData(anIF, anAD, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(AeroData), intent(inout)  :: anAD
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Check the "AeroData" object for internal consistency
!*     subroutine checkAeroData(anAD, iostat)
!*       type(AeroData), intent(inout) :: anAD
!*       integer(i4b), intent(out)     :: iostat
!*
!*     ! Deallocate all memory that was allocated for the "Polar" object
!*     subroutine deallocatePolarArrays(aPolar)
!*       type(Polar), intent(inout) :: aPolar
!*
!*     ! Copy the specified "Polar" object in "AeroData" to "targetPolar"
!*     subroutine copyPolar(anAD, airIdNr, polIdNr, targetPolar, iostat)
!*       type(AeroData), intent(in) :: anAD
!*       integer(i4b), intent(in)   :: airIdNr
!*       integer(i4b), intent(in)   :: polIdNr
!*       type(Polar), intent(inout) :: targetPolar
!*       integer(i4b), intent(out)  :: iostat
!*
!* REMARKS :
!*   Example of an "AeroData" datafile :
!*
!*   <AeroData>
!*     <Id> "This is an example <AeroData> dataset" </Id>
!*
!*     <Airfoil>
!*       <Id> "NACA 4412" </Id>
!*       <IdNr> 1 </IdNr>
!*       <Polar>
!*         <Id> "NACA 4412 Clean" </Id>
!*         <IdNr> 1 </IdNr>
!*         <Re> 2100000 </Re>
!*         <Ma> 0.1 </Ma>
!*         <Alpha-Cl>
!*           -4.000   0.0270
!*            4.000   0.9240
!*            8.000   1.3139
!*           12.000   1.6047
!*         </Alpha-Cl>
!*         <Alpha-Cd>
!*           -1.000   0.00634
!*            0.000   0.00631
!*            6.000   0.00755
!*         </Alpha-Cd>
!*         <Alpha-Cm>
!*            0.000   -0.1040
!*            8.000   -0.0955
!*         </Alpha-Cm>
!*       </Polar>
!*     </Airfoil>
!*   </AeroData>
!*
!* USAGE:
!*   No description.
!*
!******************************************************************************
```

AeroLink_m

```
!******************************************************************************
```

```
!* FILE : AeroLink_m.f90
!*
!* PROGRAM UNIT : module AeroLink_m
!*
!* PURPOSE : The AeroLink_m module handles input and use of data that relate
!*           aerodynamic coefficients with airfoil sections in the geometry
!*
!* MODULES USED : Types_m, VXTypes_m, InputFile_m, Trace_m, Geom_m, AeroData_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2002   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   The three public routines nextAeroLink(), checkAeroLink() and
!*   applyAeroLink()
!*   - handle the creation of an "AeroLink" object
!*   - check if this "AeroLink" object could be a valid one
!*   - transfer the aerodynamic data to the airfoil sections
!*
!*   Public subroutines are:
!*
!*     ! Create an "AeroLink" object from the given "InputFile"
!*     subroutine nextAeroLink(anIF, anALnk, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(AeroLink), intent(inout)  :: anALnk
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Check the "AeroLink" object for internal consistency
!*     subroutine checkAeroLink(anALnk, iostat)
!*       type(AeroLink), intent(inout) :: anALnk
!*       integer(i4b), intent(out)     :: iostat
!*
!*     ! Update the Geom object data given an AeroLink object.
!*     ! All strips of all segments are extended with aerodynamic
!*     ! polars containing coefficients as function of angle of attack
!*     subroutine applyAeroLink(aGeom, anALnk, iostat)
!*       type(Geom), intent(inout)  :: aGeom
!*       type(AeroLink), intent(in) :: anALnk
!*       integer(i4b), intent(out)  :: iostat
!*
!* REMARKS :
!*   Example of the "AeroLink" datafile :
!*
!*   <AeroLink>
!*     <Id> "This is an example <AeroLink> dataset" </Id>
!*
!*     ! Linking ALL strips in a Segment to particular Airfoil Polars
!*     <StripSet>
!*       <PartIdNr> 1 </PartIdNr>
!*       <SegmentIdNr> 1 </SegmentIdNr>
!*
!*       <StripLink>
!*         <StripNr> 1 </StripNr>
!*         <AeroDataFile> "aerodata_xmpl.dat" </AeroDataFile>
!*         <AirfoilIdNr> 1 <AirfoilIdNr>
!*         <PolarIdNr> 1 </PolarIdNr>
!*       </StripLink>
!*
!*       <StripLink>
!*         <StripNr> 2 </StripNr>
!*         <AeroDataFile> "aerodata_xmpl.dat" </AeroDataFile>
!*         <AirfoilIdNr> 1 <AirfoilIdNr>
!*         <PolarIdNr> 1 </PolarIdNr>
!*       </StripLink>
!*     </StripSet>
!*   <AeroLink>
!*
!* USAGE:
!*   call openInputFile(LU_AERO_LINK, "aerolink_xmpl", myIF)
!*   call nextAeroLink(myIF, theAL, iosAL)
!*   call closeInputFile(myIF)
!*
!*   call checkAeroLink(theAL, iosCheckAL)
!*   call applyAeroLink(theGeom, theAL, iosApplyAL)
!*
!*   if ((iosAL/=0).or.(iosCheckAL/=0).or.(iosApplyAL/=0)) then
!*     call trace(FATAL,"AeroLink_test :")
!*     call trace(FATAL,"  Error while interpreting input file")
!*     stop
!*   endif
!*
!*****************************************************************************
```

Ask_m

```
!*****************************************************************************
!* FILE : Ask_m.f90
```

```
!*
!* PROGRAM UNIT : module Ask_m
!*
!* PURPOSE : Enabling generic prompting for INTEGER, REAL, DOUBLE, LOGICAL and
!*           CHARACHTER(*) data.
!*
!* MODULES USED : Types_m, Utils_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    8-APR-2001   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   This module enables reading a variable VAR from standard input and checks
!*   for errors. The question is printed on standard output. If an empty line is
!*   entered the variable VAR stays untouched.
!*   In addition there are a few [optional] parameters to the routines like
!*   setting limits and handling flag-type input.
!*
!*  Prompting for INTEGERs, single and double precision REALs:
!*    subroutine ask(text, var, [min], [max], [flagChars], [flagOut])
!*      text       : character string with text.
!*      var        : integer_i4b, real_sp or real_dp variable.
!*      min, max   : integer_i4b, real_sp or real_dp minimum and maximum values.
!*      flagChars  : character string of flag with allowed characters.
!*      flagOut    : character(len=1) with one of the allowed characters, empty
!*                   if no flag-type answer was given.
!*
!*  Prompting for a CHARACTER string:
!*    subroutine ask(text, var, [flagChars], [flagOut])
!*      var        : character(len=*) variable
!*
!*  Prompting for LOGICAL type of answer:
!*    subroutine ask(text, var, [flagChars], [flagOut])
!*      var        : logical variable
!*
!* REMARKS :
!*   The first character in string FLAG_CHARS distinguishes between normal input
!*   and flag-type input. All other charachters are allowed to follow this first
!*   flag-type character in input. So, specifying flagChars='/bdf' allows input
!*   to be of the form /b, /d or /f
!*
!*   Recommended usage of flag-type input handling is:
!*
!*   /b  : go Back to the previous question
!*   /d  : give a Default answer
!*   /f  : go Forward to the next question, leave VAR untouched
!*   /k  : Kill all questions
!*   /s  : go back to the Start of the series of questions
!*
!*   If MIN>MAX and no flag characters are allowed an infinite loop results.
!*
!*
!* USAGE:
!*
!*   call ask('Enter an INTEGER value', i, min=-20, &
!*            max=30, flagChars='/dfk', flagOut=outflag )
!*   if (outflag == 'd') i=1024_i4b
!*   if (outflag == 'k') exit
!*   write (*,*)'INTEGER number = ', i
!*
!*   call ask('Enter a REAL value', r, max=2048.0, &
!*            flagChars='#ab', flagOut=outflag )
!*   if (outflag == 'a') r=3.14159_sp
!*   if (outflag == 'b') r=6.28318_sp
!*   write (*,*)'REAL number = ', r
!*
!*   call ask('Enter a DOUBLE value', d, flagChars='#34', &
!*            flagOut=outflag )
!*   if (outflag == '1') d=0.0_dp
!*   if (outflag == '2') d=4.0e30_dp
!*   write (*,*)'DOUBLE number = ', d
!*
!*   s = 'c:\temp\myfile.dat'
!*   call ask('Give a filename' , s, flagChars='/d', flagOut=outflag)
!*   write(*,*)' CHARACTER string = ', s
!*
!*   b=.true.
!*   call ask('Do you want to continue?' , b, &
!*            flagChars='/k', flagOut=outflag)
!*   if ((outflag == 'k').or.(.not.b)) exit
!*   write (*,*)'LOGICAL variable = ', b
!*
!*****************************************************************************
```

DataInput_m

```
!*****************************************************************************
```

```
!* FILE : DataInput_m.f90
!*
!* PROGRAM UNIT : module DataInput_m
!*
!* PURPOSE : In the DataInput_m module user command line input is stored as
!*           global data.
!*
!* MODULES USED : Types_m, VXParam_m, Trace_m, Ask_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    7-NOV-2002    A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   The global data defined in this module is initialized by the
!*   (private) subroutine setDefaults(). These values can be changed through
!*   a call to userInput() that asks for command line input.
!*
!*   ! Command line input routine
!*   subroutine userInput()
!*
!*   ! Write commandline input to the enabled log output files
!*   subroutine traceUserInput()
!*
!* REMARKS :
!*   Questions in userInput() can be skipped by entering '/f'.
!*   Typing '/b' returns the previous question
!*
!* USAGE:
!*   call userInput()
!*   call traceUserInput()
!*
!*****************************************************************************
```

DataOutput_m

```
!*****************************************************************************
!* FILE : DataOutput_m.f90
!*
!* PROGRAM UNIT : DataOutput_m
!*
!* PURPOSE : To write results from the computation to output files
!*
!* MODULES USED :  Types_m, Trace_m, VXTypes_m, VXParam_m, Utils_m, Vec34_m,
!*                 Wind_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2002   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*
!*  ! Log some data from the current time(step)
!*  subroutine dataOutput(aGeom, iTime, prevTime, curTime)
!*    type(Geom), intent(in)   :: aGeom
!*    integer(i4b), intent(in) :: iTime
!*    real(dp), intent(in)     :: prevTime, curTime
!*
!*  ! Write the segment and wake geometry data to file
!*  subroutine writeGeomResults(aGeom, iTime, curTime, iTimeStart, iTimeEnd, iTimeStep)
!*    type(Geom), intent(in)   :: aGeom
!*    integer(i4b), intent(in) :: iTime, iTimeStart, iTimeEnd, iTimeStep
!*    real(dp), intent(in)     :: curTime
!*
!*  ! Write the segment and wake aerodynamic data to file
!*  subroutine writeAeroResults(aGeom, iTime, curTime, iTimeStart, iTimeEnd, iTimeStep)
!*    type(Geom), intent(in)   :: aGeom
!*    integer(i4b), intent(in) :: iTime, iTimeStart, iTimeEnd, iTimeStep
!*    real(dp), intent(in)     :: curTime
!*
!*  ! Open a 'sequential formatted' file to write results to
!*  subroutine openResultsFile(lu, filename)
!*    integer(i4b), intent(in) :: lu
!*    character(*), intent(in) :: filename
!*
!*  ! Close the 'sequential formatted' output file
!*  subroutine closeResultsFile(lu)
!*    integer(i4b), intent(in) :: lu
!*
!* REMARKS :
!*  None.
!*
!* USAGE:
!*  No description.
!*
!*****************************************************************************
```

GeomScenario_m

```
!*****************************************************************************
!* FILE : GeomScenario_m.f90
!*
!* PROGRAM UNIT : module GeomScenario_m
!*
!* PURPOSE : The GeomScenario_m module supplies geometry motion subroutines
!*
!* MODULES USED : Types_m, VXTypes_m, Utils_m, Trace_m, InputFile_m, Vec34_m,
!*                Mat4_m, VXParam_m, Geom_m, GeomSnapshot_m, Spline_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    15-MAR-2001  A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   This module supplies basic geometry motion subroutines
!*
!*   Public subroutines:
!*
!*     ! Create a new GeomScenario object from a given InputFile object that is
!*     ! wrapped around a geometry scenario file
!*     subroutine nextGeomScenario(anIF, aGSc, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(GeomScenario), intent(inout)  :: aGSc
!*       integer(i4b), intent(out)          :: iostat
!*
!*     !Check if Geom and GeomScenario objects are consistent
!*     subroutine checkGeomScenario(aGSc, aGeom, iostat)
!*       type(GeomScenario), intent(in) :: aGSc
!*       type(Geom), intent(in)         :: aGeom
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Create a new GeomSnapshot object at a specific Time from a given
!*     ! GeomScenario object
!*     subroutine makeGeomSnapshot(aGSc, aTime, theGSn)
!*       type(GeomScenario), intent(in) :: aGSc
!*       real(dp), intent(in)           :: aTime
!*       type(GeomSnapshot), intent(out):: theGSn
!*
!* REMARKS :
!*   An example geometry scenario file is:
!*
!*   ! File containing rotations and/or translations of specified Markers at
!*   ! discrete instances of time. Together they form a scenario for the
!*   ! geometric transformations. Values at a specified moment in time can be
!*   ! obtained through interpolation.
!*   ! In a future version Segment deformations can be added to this file.
!*
!*   <GeomScenario>
!*     <Id> "A GeomScenario identification string"  </Id>
!*
!*     <MarkerTimeSeries>
!*       <PartIdNr> 1 </PartIdNr>
!*       <MarkerIdNr> 90210 </MarkerIdNr>
!*       <InterpolationType> "SPLINE" </InterpolationType>
!*       <Extrapolation> NO </Extrapolation>
!*
!*       <MarkerTimeInstance>
!*         <Time> 0.0 </Time>
!*         <Translation> 100.0  100.0  100.0 </Translation>
!*         <RotationAxis> 1.0 1.0 1.0 </RotationAxis>
!*         <RotationAngle> 0.0 </RotationAngle>
!*       </MarkerTimeInstance>
!*
!*       <MarkerTimeInstance>
!*         <Time> 600.0 </Time>
!*         <Translation> 100.0  100.0  100.0 </Translation>
!*         <RotationAxis> 1.0 1.0 1.0 </RotationAxis>
!*         <RotationAngle> 3600.0 </RotationAngle>
!*       </MarkerTimeInstance>
!*     </MarkerTimeSeries>
!*
!*     <MarkerTimeSeries>
!*       <PartIdNr> 2 </PartIdNr>
!*       <MarkerIdNr> 2001 </MarkerIdNr>
!*       <InterpolationType> "LINEAR" </InterpolationType>
!*       <Extrapolation> NO </Extrapolation>
!*
!*       <MarkerTimeInstance>
!*         <Time> 0.0 </Time>
!*         <Translation> 0.0  0.0  0.0 </Translation>
!*         <RotationAxis> 0.0 0.0 1.0 </RotationAxis>
!*         <RotationAngle> 0.0 </RotationAngle>
!*       </MarkerTimeInstance>
!*
!*       <MarkerTimeInstance>
!*         <Time> 60.0 </Time>
!*         <Translation> 0.0  0.0  0.0 </Translation>
!*         <RotationAxis> 0.0 0.0 1.0 </RotationAxis>
!*         <RotationAngle> 60.0 </RotationAngle>
!*       </MarkerTimeInstance>
!*     </MarkerTimeSeries>
!*
```

```
!*   !  Segment deformations are implemented in a future version !
!*   !  <SegmentTimeSeries>
!*   !    <PartIdNr> 1 </PartIdNr>
!*   !    <SegmentIdNr> 11 </SegmentIdNr>
!*   !    <InterpolationType> "LINEAR" </InterpolationType>
!*   !    <Extrapolation> NO </Extrapolation>
!*   !
!*   !    <SegmentTimeInstance>
!*   !      <Time> 0.0 </Time>
!*   !      <Deformation>
!*   !        0.00 0.005 0.0     0.01  0.01  0.0    0.0    0.0    0.0
!*   !        0.00 0.002 0.0     0.0   0.0   0.0    0.02   0.0001 0.0
!*   !      </Deformation>
!*   !    </SegmentTimeInstance>
!*   !
!*   !    <SegmentTimeInstance>
!*   !      <Time> 100.0 </Time>
!*   !      <Deformation>
!*   !        0.01 0.005 0.0    -0.001 0.001 0.0    0.0    0.0    0.0
!*   !        0.02 0.002 0.0     0.0   0.0   0.0    0.0012 0.0    0.0
!*   !      </Deformation>
!*   !    </SegmentTimeInstance>
!*   !  </SegmentTimeSeries>
!*   </GeomScenario>
!*
!* USAGE:
!*   No description.
!*
!*******************************************************************************
```

GeomSnapshot_m

```
!*******************************************************************************
!* FILE : GeomSnapshot_m.f90
!*
!* PROGRAM UNIT : module GeomSnapshot_m
!*
!* PURPOSE : The GeomSnapshot_m module enables geometry snapshot handling; the
!*           orientation of Markers in space at a specific instance in time.
!*
!* MODULES USED : Types_m, VXTypes_m, Utils_m, Trace_m, InputFile_m
!*                Vec34_m, Mat4_m, VXParam_m, Geom_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    12-JUN-2001  A. van Garrel   Initial coding
!*
!*******************************************************************************
!* SHORT DESCRIPTION :
!*   This module supplies geometry snapshot handling subroutines
!*
!*   Public subroutines:
!*
!*     ! Create a new MTimeInst object from a given InputFile object
!*     subroutine nextMTimeInst(anIF, aMTI, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(MTimeInst), intent(inout)     :: aMTI
!*       integer(i4b), intent(out)          :: iostat
!*
!*     ! Create a new GeomSnapshot object from a given InputFile object
!*     ! that is wrapped around a geometry snapshot file
!*     subroutine nextGeomSnapshot(anIF, theGSn, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(GeomSnapshot), intent(inout)  :: theGSn
!*       integer(i4b), intent(out)          :: iostat
!*
!*     ! Write a given GeomSnapshot object to file
!*     subroutine writeGeomSnapshot(theGSn, iostat)
!*       type(GeomSnapshot), intent(in) :: theGSn
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Deallocate all arrays that were allocated in the "GeomSnapshot" object
!*     subroutine deallocateGeomSnapshotArrays(aGSn)
!*       type(GeomSnapshot), intent(inout) :: aGSn
!*       integer(i4b) :: status
!*
!* REMARKS :
!*
!*   An example geometry snapshot file is:
!*
!*   ! File containing rotations and translations of 'Markers' at a specific
!*   ! moment in time (defined in their Part's own coordinate system).
!*   ! In a future version Segment deformations can be added to this file.
!*   <GeomSnapshot>
!*     <MarkerTransformation>
!*       <PartIdNr> 1 </PartIdNr>
!*       <MarkerIdNr> 90210 </MarkerIdNr>
!*       <MarkerTimeInstance>
!*         <Time> 12.340 </Time>
```

```
!*         <Translation> 100.0  100.0  100.0 </Translation>
!*         <RotationAxis> 0.577  0.577  0.577 </RotationAxis>
!*         <RotationAngle> 0.000 </RotationAngle>
!*       </MarkerTimeInstance>
!*     </MarkerTransformation>
!*
!*     <MarkerTransformation>
!*       <PartIdNr> 2 </PartIdNr>
!*       <MarkerIdNr> 2001 </MarkerIdNr>
!*       <MarkerTimeInstance>
!*         <Time> 12.340 </Time>
!*         <Translation> 100.0  100.0  100.0 </Translation>
!*         <RotationAxis> 1.0 1.0 1.0 </RotationAxis>
!*         <RotationAngle> 0.0 </RotationAngle>
!*       </MarkerTimeInstance>
!*     </MarkerTransformation>
!*
!*    ! Segment deformations are implemented in a future version !
!*    ! <SegmentTransformation>
!*    !   <PartIdNr> 1 </PartIdNr>
!*    !   <ComponentType> "SEGMENT" </ComponentType>
!*    !   <ComponentIdNr> 11 </ComponentIdNr>
!*    !   <SegmentTimeInstance>
!*    !     <Time> 12.340 </Time>
!*    !     <Deformation>
!*    !       0.01 0.005 0.0    -0.001 0.001 0.0    0.0    0.0    0.0
!*    !       0.02 0.002 0.0     0.0   0.0   0.0    0.0012 0.0001 0.0
!*    !     </Deformation>
!*    !   </SegmentTimeInstance>
!*    ! </SegmentTransformation>
!*
!*   </GeomSnapshot>
!*
!*
!* USAGE:
!*   No description.
!*
!******************************************************************************
```

Geom_m

```
!******************************************************************************
!* FILE : Geom_m.f90
!*
!* PROGRAM UNIT : module Geom_m
!*
!* PURPOSE : The Geom_m module enables basic geometry handling.
!*           Aerodynamic data are linked to the geometry also.
!*
!* MODULES USED : Types_m, VXTypes_m, Utils_m, Trace_m, InputFile_m, Vec34_m,
!*                Mat4_m, VortexLattice_m, AeroData_m, VXParam_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    23-NOV-2001   A. van Garrel   Initial coding
!*
!******************************************************************************
!* SHORT DESCRIPTION :
!*   This module supplies basic geometry handling subroutines
!*
!*   Public subroutines:
!*
!*     ! Create a new Geom object from a given InputFile object that is
!*     ! wrapped around a geometry file
!*     subroutine nextGeom(anIF, aGeom, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(Geom), intent(inout)      :: aGeom
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Check if all of Geom's IdNr's are unique within their encompassing
!*     ! data structure
!*     subroutine checkGeomIdNrs(aGeom, iostat)
!*       type(Geom), intent(inout) :: aGeom
!*       integer(i4b), intent(out) :: iostat
!*
!*     ! Set references to parent and child Parts for all Links in the Geom
!*     ! object
!*     subroutine activateGeomLinks(aGeom, iostat)
!*       type(Geom), intent(inout)   :: aGeom
!*       integer(i4b), intent(out)   :: iostat
!*
!*     ! Search matrix indices for specific Part and Marker IdNrs
!*     subroutine searchPMIndex(aGeom, pIdNr, mIdNr, iPart, iMarker, iostat)
!*       type(Geom), intent(in)   :: aGeom
!*       integer(i4b), intent(in)    :: pIdNr   ! Part IdNr
!*       integer(i4b), intent(in)    :: mIdNr   ! Marker IdNr
!*       integer(i4b), intent(out)   :: iPart   ! Part index
!*       integer(i4b), intent(out)   :: iMarker ! Marker index
!*       integer(i4b), intent(out)   :: iostat
```

```
!*
!*     ! Save current actual trailing edge points as "old" values
!*     subroutine saveTrailingEdge(aGeom)
!*       type(Geom), intent(inout) :: aGeom
!*
!*     ! Update the Geom object data given a GeomSnapshot object
!*     ! In addition to an update of the Marker data, the Link Matrices
!*     ! that represent the relations between child and parent Parts are
!*     ! updated also. On exit the actual Segment coordinates are updated.
!*     ! The "old" Segment t.e. coordinates are left unaltered.
!*     subroutine applyGeomSnapshot(aGSn, theGeom, iostat)
!*       type(GeomSnapshot), intent(in) :: aGSn
!*       type(Geom), intent(inout)      :: theGeom
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Convect the "old" trailing edge position with wind velocity
!*     subroutine convectOldTrailingEdge(aGeom, prevTime, curTime, iostat)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: prevTime
!*       real(dp), intent(in)      :: curTime
!*       integer(i4b), intent(out) :: iostat
!*
!*
!* REMARKS :
!*   An example geometry file is :
!*
!*   ! File containing the initial definition of geometric entities like
!*   ! Parts, Segments and Markers and Links.
!*   <Geom>
!*     <Part>
!*       <Id> "Blade" </Id>
!*       <IdNr> 1 </IdNr>
!*       <Segment>
!*         <Id> "Segment 1" </Id>
!*         <IdNr> 1 </IdNr>
!*         <Coordinates>
!*           1.0 1.0 0.0  2.0 1.0 0.0
!*           1.0 2.0 0.0  2.0 2.0 0.0
!*           1.0 3.0 0.0  2.0 3.0 0.0
!*         </Coordinates>
!*         <Ni> 2 </Ni>
!*         <Nj> 3 </Nj>
!*       </Segment>
!*
!*       <Marker>
!*         <Id> "Root Marker" </Id>
!*         <IdNr> 1 </IdNr>
!*         <Location> 0.0  1.0  1.0 </Location>
!*         <Vector1> 1.0  0.0  0.0 </Vector1>
!*         <Vector2> 0.0  1.0  0.0 </Vector2>
!*       </Marker>
!*     </Part>
!*
!*     <Part>
!*       <Id> "Rotor axis" </Id>
!*       <IdNr> 2 </IdNr>
!*       <Marker>
!*         <IdNr> 11 </IdNr>
!*         <Id> "Blade attachment Marker" </Id>
!*         <Location> 0.0  0.0  0.0 </Location>
!*         <Vector1> 0.0 -1.0  0.0 </Vector1>
!*         <Vector2> 1.0  0.0  0.0 </Vector2>
!*       </Marker>
!*     </Part>
!*
!*     <Link>
!*       <Id> "Blade-Rotor axis connection" </Id>
!*       <IdNr> 11 </IdNr>
!*       <ParentPart> 2 </ParentPart>
!*       <ParentMarker> 11 </ParentMarker>
!*       <ChildPart> 1 </ChildPart>
!*       <ChildMarker> 1 </ChildMarker>
!*     </Link>
!*   </Geom>
!*
!*
!*    The relation between Link, Part, Segment and Marker objects
!*    in a Geom object can be represented graphically as:
!*
!*    PART <------> LINK <------> PART
!*      |                         |
!*    Marker(s)                 Marker(s)
!*    Segment(s)                Segment(s)
!*
!*
!* USAGE:
!*   No description.
!*
!*******************************************************************************
```

InputFile_m

```
!*******************************************************************************
!* FILE : InputFile_m.f90
!*
!* PROGRAM UNIT : module InputFile_m
!*
!* PURPOSE : The InputFile_m module enables interpretation of input files
!*           by iterating over the words.
!*
!* MODULES USED : Types_m, VXTypes_m, Utils_m, Trace_m, Vec34_m, VXParam_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    15-NOV-2001  A. van Garrel   Initial coding
!*
!*******************************************************************************
!* SHORT DESCRIPTION :
!*   This module enables reading a file and iterating over its words so they
!*   subsequently be interpreted. XML-style open and close tags can be tested
!*   for.
!*
!*   Public subroutines:
!*
!*     ! Open underlying file of the InputFile variable.
!*     subroutine openInputFile(lu, filename, anIF)
!*       lu        : Unit nr. of type integer(i4b) associated with the InputFile
!*       filename  : Filename of type character* associated with the InputFile
!*       anIF      : InputFile variable
!*
!*     ! Close underlying file of the InputFile variable.
!*     subroutine closeInputFile(anIF)
!*       anIF      : InputFile variable
!*
!*     ! Return TRUE if current line contents is non-empty or if one of the lines
!*     ! following the current one is a non-empty or non-comment line. The current
!*     ! line is set to the non-empty or non-comment line.
!*     function hasNext(anIF)
!*       anIF      : InputFile variable
!*
!*     ! Remove and return the next word in the current line as character string
!*     ! variable. Blanks are used as word separation identifiers.
!*     subroutine nextWord(anIF, aWord)
!*       anIF      : InputFile variable
!*       aWord     : Character string variable containing word on return
!*
!*     ! Return TRUE if next word in the current line equals a specific open tag.
!*     ! The open tag is constructed by concatenating OPENTAG1//aTag//OPENTAG2.
!*     function atOpenTag(anIF, aTag)
!*       anIF      : InputFile variable
!*       aTag      : A specific open tag character string
!*
!*     ! Return TRUE if next word in the current line equals a specific close tag.
!*     ! The close tag is constructed by concatenating CLOSETAG1//aTag//CLOSETAG2.
!*     function atCloseTag(anIF, aTag)
!*       anIF      : InputFile variable
!*       aTag      : A specific open tag character string
!*
!*     ! Read a single integer that is enclosed by tags: <Tag> 7 </Tag>
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getOneInteger(anIF, theInt, aTag, traceString, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       integer(i4b), intent(out)      :: theInt
!*       integer(i4b), intent(inout)    :: iostat
!*       character(len=*), intent(in)   :: aTag, traceString
!*
!*     ! Read a single double that is enclosed by tags: <Tag> 3.14159 </Tag>
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getOneDouble(anIF, theDouble, aTag, traceString, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       real(dp), intent(out)          :: theDouble
!*       integer(i4b), intent(inout)    :: iostat
!*       character(len=*), intent(in)   :: aTag, traceString
!*
!*     ! Read a single string that is enclosed by tags: <Tag> "Hello" </Tag>
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getOneString(anIF, theVS, aTag, traceString, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(VarString), intent(out)   :: theVS
!*       integer(i4b), intent(inout)    :: iostat
!*       character(len=*), intent(in)   :: aTag, traceString
!*
!*     ! Read a single logical that is enclosed by tags: <Tag> TRUE </Tag>
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getOneLogical(anIF, theLog, aTag, traceString, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       logical, intent(out)           :: theLog
!*       integer(i4b), intent(inout)    :: iostat
!*       character(len=*), intent(in)   :: aTag, traceString
!*
!*     ! Read a single 3-component vector that is enclosed by tags:
!*     ! <Tag> 3.14  2.71  9.81 </Tag>
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getOneVec3(anIF, theVec3, aTag, traceString, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(Vec3), intent(out)        :: theVec3
!*       integer(i4b), intent(inout)    :: iostat
!*       character(len=*), intent(in)   :: aTag, traceString
```

```
!*
!*     ! Read a number of 3-component vectors enclosed by tags:
!*     ! <Tag>
!*     !   1.0  0.0  0.0
!*     !   0.0  1.0  0.0
!*     !   0.0  0.0  1.0
!*     ! </Tag>
!*     ! The Vec3 components are stored in a rank-1 buffer.
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getVec3Buffer(anIF, buf, nVec, aTag, traceString, iostat)
!*       type(InputFile), intent(inout)    :: anIF
!*       type(Vec3), dimension(:), pointer :: buf
!*       integer(i4b), intent(out)         :: nVec
!*       character(len=*), intent(in)      :: aTag, traceString
!*       integer(i4b), intent(out)         :: iostat
!*
!*     ! Read a number of DOUBLE values enclosed by tags:
!*     ! <Tag>
!*     !   1.0  2.0  3.0
!*     !   4.0  5.0
!*     ! </Tag>
!*     ! On exit the InputFile has stepped over the closing tag
!*     subroutine getDoubleBuffer(anIF, buf, nDbl, aTag, traceString, iostat)
!*       type(InputFile), intent(inout)  :: anIF
!*       real(dp), dimension(:), pointer :: buf
!*       integer(i4b), intent(out)       :: nDbl
!*       character(len=*), intent(in)    :: aTag, traceString
!*       integer(i4b), intent(out)       :: iostat
!*
!*     ! Strip all leading and trailing blanks from a string
!*     function strip(aStr)
!*       character(len=*), intent(in)  :: aStr
!*       character(len=len_trim(adjustl(aStr))) :: strip
!*
!*     ! Check if the "TextLine" currently stands at "aText"
!*     function atText(theTL, aText)
!*       type(TextLine), intent(in)   :: theTL
!*       character(len=*), intent(in) :: aText
!*       logical :: atText
!*
!* REMARKS :
!*   Comment lines and empty lines in the input file will not be interpreted.
!*   A comment line is any line that starts with a CHARS_COMMENT character.
!*   A typical inputfile would, at primitive data level, look like:
!*
!*     # Words must be separated by blanks
!*     <Integer> 987 </Integer>
!*     <Logical> YES </Logical>  <Logical> Off </Logical>
!*     <Double> 1.234E6  2.234  0.00987 </Double>
!*     <String> "A string that is "
!*              "distributed over 2 lines!" </String>
!*     <String>
!*       ^This is a valid String^
!*     </String>
!*     ! This is a comment line
!*
!*   Composite data can be defined by a hierarchical use of (tagged) primitive
!*   data types. A well-formed 'composite' data example is:
!*
!*     <Square>
!*       <Name> "Square 1" </Name>
!*       <Length> 4.6 </Length>
!*       <Position> 2.3 0.0 </Position>
!*       <IdNr> 9 </IdNr>
!*     <Square>
!*
!*   The next example is incorrect formatted data:
!*
!*     <Circle>
!*     # Specifying name, radius and position:
!*       "Wheel"  1.1
!*       <Pos> 2.3 0.0 </Pos>
!*     <Circle>
!*
!*
!* USAGE:
!*   type(InputFile) :: myII
!*   call openInputFile(12, 'tst.dat', myII)
!*   do
!*     if (hasNext(myII) then
!*       if (atOpenTag(myII,"Data")) then
!*         if (atOpenTag(myII,"Integer")) then
!*           call getOneInteger(myII, theInt, "Integer", "Error 1", iostat)
!*         if (atOpenTag(myII,"Vector")) then
!*           call getOneVec3(myII, theVec3, "Vector", "Error 2", iostat)
!*         else if (atOpenTag(myII,"String")) then
!*           call getOneString(myII, theStr, "String", "Error 3", iostat)
!*         else if (atCloseTag(myII,"Data"))
!*           exit
!*         else
!*           write(*,*) 'Current word not recognized'
!*           call nextWord(myII, aVS)  ! Step over word
!*         endif
!*       endif
!*     else
!*       exit
```

```
!*     endif
!*   end do
!*   call closeInputFile(myII)
!*
!*******************************************************************************
```

Mat4_m

```
!*******************************************************************************
!* FILE : Mat4_m.f90
!*
!* PROGRAM UNIT : module Mat4_m
!*
!* PURPOSE : This module contains the definitions and functions associated with
!*           solid body transformation matrices.
!*
!* MODULES USED : Types_m, VXTypes_m, Vec34_m, Utils_m, Trace_m, VXParam_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    19-DEC-2001  A. van Garrel   Initial coding
!*
!*******************************************************************************
!* SHORT DESCRIPTION :
!*
!*  Interface definition for Mat4-Mat4 and Mat4-Vec4 multiplication
!*    interface mat4Mul
!*      module procedure mat4Mul_m4m4, mat4Mul_m4v4
!*    end interface mat4Mul
!*    public :: mat4Mul
!*    private :: mat4Mul_m4m4, mat4Mul_m4v4
!*
!*  ! Return a Mat4 translation matrix representing the given shift vector
!*  function tMat4(shift)
!*    type(Mat4) :: tMat4
!*    type(Vec3), intent(in) :: shift
!*
!*  ! Return a Mat4 rotation matrix representing the rotation about the given
!*  ! axis vector and rotation angle in radians. The axis direction defines the
!*  ! direction of rotation and is NOT required to be of unit length.
!*  function rMat4(axis, angle)
!*    type(Mat4) :: rMat4
!*    type(Vec3), intent(in) :: axis
!*    real(dp), intent(in)   :: angle
!*
!*  ! Combined Translation-Rotation-Translation transformation matrices. Vector
!*  ! shift1 is applied first followed by a rotation and the shift2 translation.
!*  function trtMat4(shift1, axis, angle, shift2)
!*    type(Mat4) :: trtMat4
!*    type(Vec3), intent(in) :: shift1, shift2
!*    type(Vec3), intent(in) :: axis
!*    real(dp), intent(in)   :: angle
!*
!*  ! Transformation matrix that matches two source vectors and a source point
!*  ! with two target vectors and a target point.
!*  ! Actually srcVec1 and tarVec1 will be aligned and the planes defined by the
!*  ! source and target vector pairs and points will be matched. (This deals
!*  ! with the situation that the included angles of the vector pairs are not
!*  ! EXACTLY the same and thus cannot be matched by a solid body transformation)
!*  function matchMat4(srcVec1, srcVec2, srcPoint, tarVec1, tarVec2, tarPoint)
!*    type(Mat4) :: matchMat4
!*    type(Vec3), intent(in) :: srcVec1, srcVec2    ! Source vectors
!*    type(Vec3), intent(in) :: tarVec1, tarVec2    ! Target vectors
!*    type(Vec3), intent(in) :: srcPoint, tarPoint  ! Source and target points
!*
!*  ! Mat4-Mat4 multiplication (private function)
!*  function mat4Mul_m4m4(aMat4, bMat4)
!*    type(Mat4) :: mat4Mul_m4m4
!*    type(Mat4), intent(in) :: aMat4, bMat4
!*
!*  ! Mat4-Vec4 multiplication (private function)
!*  function mat4Mul_m4v4(aMat4, aVec4)
!*    type(Vec4) :: mat4Mul_m4v4
!*    type(Mat4), intent(in) :: aMat4
!*    type(Vec4), intent(in) :: aVec4
!*
!*  ! Conversion from Mat4 to a regular 4x4 array
!*  function toArray_m4(aMat4)
!*    real(dp), dimension(4,4) :: toArray_m4
!*    type(Mat4), intent(in) :: aMat4
!*
!*  ! Conversion from regular 4x4 array to Mat4
!*  function toMat4(anArray)
!*    type(Mat4) :: toMat4
!*    real(dp), dimension(4,4), intent(in) :: anArray
!*
!*  ! Mat4 transpose
!*  function mat4Transpose(aMat4)
!*    type(Mat4) :: mat4Transpose
```

```
!*    type(Mat4), intent(in) :: aMat4
!*
!*  ! Check if a Mat4 is the identity matrix. A difference of eps is allowed for
!*  ! each matrix element.
!*  function isIdentity(aMat4, eps)
!*    logical :: isIdentity
!*    type(Mat4), intent(in) :: aMat4
!*    real(dp), intent(in) :: eps
!*
!*  ! Check if a Mat4 is an orthogonal matrix. The matrix and its transpose are
!*  ! multiplied and checked to be the identity matrix. A difference of eps with
!*  ! the identity matrix is allowed for each element.
!*  function isOrthogonal(aMat4, eps)
!*    logical :: isOrthogonal
!*    type(Mat4), intent(in) :: aMat4
!*    real(dp), intent(in) :: eps
!*
!*  ! Return the identity matrix
!*  function identityMat4()
!*    type(Mat4) :: identityMat4
!*
!* REMARKS :
!*   The 4x4 transformation matrices have the following form
!*
!*     Translation :          Rotation :
!*
!*     |1  0  0  t1|          |a  d  g  0|
!*     |0  1  0  t2|          |b  e  h  0|
!*     |0  0  1  t3|          |c  f  i  0|
!*     |0  0  0  1 |          |0  0  0  1|
!*
!*   These matrices work on homogeneous coordinates which are composed of the
!*   3 components of a Cartesian vector supplemented with a 4th component equal
!*   to 1.0 : (x1, x2, x3, 1.0)
!*   The use of homogeneous coordinates enables the combination of translation
!*   and rotation into 1 transformation matrix.
!*
!*   A rotation matrix is an orthogonal matrix: its inverse is equal to its
!*   transpose.
!*
!* USAGE:
!*
!*   ! Match source vector/point pairs with target vector/point pairs
!*   aMat4 = matchMat4(sv1, sv2, spoint, tv1, tv2, tpoint)
!*   ! Compute the matrix for back-transformation
!*   bMat4 = matchMat4(tv1, tv2, tpoint, sv1, sv2, spoint)
!*   ! Multiply the two matrices
!*   cMat4 = mat4Mul(aMat4,bMat4)
!*   ! This should be the identity matrix
!*   call trace(DEBUG," Identity matrix? : ", isIdentity(cMat4, EPS8))
!*
!*   ! Define the z-axis as rotation axis and a rotation angle of 90 degrees
!*   rotAxis = toVec3(0.0_dp, 0.0_dp, 100.0_dp)
!*   rotAngle = DEG2RAD*90.0_dp
!*   ! Determine the rotation matrix
!*   dMat4 = rMat4(rotAxis,rotAngle)
!*   ! This should be an orthogonal matrix
!*   call trace(DEBUG," Orthogonal matrix? : ", isOrthogonal(dMat4, EPS8))
!*
!*   ! Transform point (1.0, 1.0, 0.0) with this rotation matrix
!*   aVec4 = mat4Mult(dMat4,toVec4(1.0_dp, 1.0_dp, 0.0_dp, 1.0_dp))
!*
!*****************************************************************************
```

Spline_m

```
!*****************************************************************************
!* FILE : Spline_m.f90
!*
!* PROGRAM UNIT : module Spline_m
!*
!* PURPOSE : Spline interpolation of dataset at a given abscissa value
!*
!* MODULES USED : Types_m, Trace_m, Utils_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2001   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   Spline interpolation/extrapolation and data scaling.
!*   This module performs linear spline and rational Hermite spline
!*   interpolation in a dataset with monotonically increasing abscissa values.
!*   The function derivative in each internal point is determined by the line
!*   through the neighboring points. In this way the larger interval will
!*   contribute more to the curve's tangent. The weight of the rational part
!*   effectively straightens the spline in case of non-smooth derivative
!*   variation and will prevent unwanted oscillations. The utility scaling
```

```
!*   routines SCALING(), NORMALIZE() and DENORMALIZE() can be used to obtain
!*   similar sized function and abscissa values.
!*
!*   Interpolate/extrapolate data at given abscissa value with linear spline:
!*     subroutine linear(x, y, xVal, yVal)
!*       x    : real_dp vector of abscissa values
!*       y    : real_dp vector of function values
!*       xVal : real_dp abscissa value to be interpolated/extrapolated
!*       yVal : real_dp interpolated function value
!*
!*   Interpolate/extrapolate data at given abscissa value with a rational
!*   Hermite spline:
!*     subroutine ratHermite(x, y, xVal, yVal)
!*       x    : real_dp vector of abscissa values
!*       y    : real_dp vector of function values
!*       xVal : real_dp abscissa value to be interpolated/extrapolated
!*       yVal : real_dp interpolated function value
!*
!*   Scale data; mapping of range [x1from,x2from] to [x1to,x2to]
!*     subroutine scaling(x, x1from, x2from, x1to, x2to)
!*       x      : real_dp vector or single variable of data
!*       x1from : real_dp value 1 of original data range
!*       x2from : real_dp value 2 of original data range
!*       x1to   : real_dp value 1 of target data range
!*       x2to   : real_dp value 2 of target data range
!*
!*   Normalize data; scale data range from [xmin,xmax] to [0,1]
!*     subroutine normalize(x, xmin, xmax)
!*       x    : real_dp vector or single variable of data
!*       xmin : real_dp minimum value of data range
!*       xmax : real_dp maximum value of data range
!*
!*   Normalize data; scale data range from [0,1] back to [xmin,xmax]
!*     subroutine denormalize(x, xmin, xmax)
!*       x    : real_dp vector or single variable of data
!*       xmin : real_dp minimum value of data range
!*       xmax : real_dp maximum value of data range
!*
!* REMARKS :
!*   The abscissa values are assumed to be monotonically increasing.
!*   For abscissa values outside the data interval function values are
!*   obtained by extrapolation.
!*   Discontinuous functions are identified by duplicate abscissa values
!*   and are treated as separate spline functions. Triple (or higher) valued
!*   internal points and double (or higher) valued endpoints will result in a
!*   FATAL error.
!*   Continuity of the 2nd derivative is NOT enforced.
!*   The definition of the rational spline was taken from:
!*
!*   H. Spath, "Spline-Algorithmen zur Konstruktion glatter Kurven und
!*   Flaechen", 1973
!*
!*   The 1D array with function values should match the array with
!*   corresponding abscissa values. A minimum of 2 data points is allowed.
!*
!*   The NORMALIZE() and DENORMALIZE() subroutines scale the range of the
!*   supplied dataset from [MIN,MAX] to [0,1] and back. The more general
!*   version of linear scaling is handled by subroutine SCALING() where the
!*   two points X1FROM and X2FROM are mapped to X1TO and X2TO respectively.
!*   A FATAL error occurs when the range that the data will be scaled to is
!*   of zero length.
!*
!* USAGE:
!*   ! Scale vectors x and y from [min,max] to [0,1]
!*   call normalize(x, xmin, xmax)
!*   call normalize(y, ymin, ymax)
!*
!*   do j=1,n
!*     xVal = xmin + (xmax-xmin) * (j-1) / (n-1)
!*     call normalize(xVal, xmin, xmax)
!*     call ratHermite(x, y, xVal, yVal)
!*     call linear(x, y, xVal2, yVal2)
!*
!*     ! Scale xVal and yVal back from [0,1] to [min,max]
!*     call denormalize(xVal, xmin, xmax)
!*     call denormalize(yVal, ymin, ymax)
!*   end do
!*
!*******************************************************************************
```

Trace_m

```
!*******************************************************************************
!* FILE : Trace_m.f90
!*
!* PROGRAM UNIT : module Trace_m
!*
!* PURPOSE : The Trace_m module enables logging of messages generated during
!*           the flow of the program. It can be used as a debugging tool during
```

```
!*           development, and as a troubleshooting tool once the system has
!*           been deployed in a production environment.
!*
!*
!* MODULES USED : Types_m, VXParam_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    11-APR-2001  A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   The Trace_m module enables messages and variable values to be written to
!*   standard output and/or user-specified files. The type of messages written
!*   can be controlled by setting a message severity level. Messages of the same
!*   and higher priority will be processed, while lower severity messages are
!*   discarded.
!*
!*   Setting for the Trace_m module can be set through calls to
!*     subroutine traceInit([lvl], [stdout], [file], [filename])
!*       lvl      : Message severity level of value: DEBUG, INFO, WARNING,
!*                  ERROR, FATAL or MESSAGE. Default: level=WARNING
!*       stdout   : logical; Is .TRUE. if standard output enabled
!*       file     : logical; Is .TRUE. if output to file enabled
!*       filename : character(*); File to write to, if file=.TRUE.
!*
!*     Default initialization settings are: lvl=MESSAGE,
!*                                          stdout=.TRUE.
!*                                          file=.FALSE.
!*                                          filename=''
!*
!*     Trace subroutines are of the form:
!*
!*       ! Trace output character string
!*       subroutine trace(lvl,txt)
!*         integer(i4b), intent(in)               :: lvl
!*         character(len=*), intent(in)           :: txt
!*  or
!*       ! Trace output character string and variable value
!*       subroutine trace(lvl, txt, var, form)
!*         implicit none
!*         integer(i4b), intent(in)               :: lvl
!*         character(len=*), intent(in)           :: txt
!*         real(dp), dimension(:), intent(in)     :: var
!*         integer(i4b), intent(in)               :: n
!*         character(len=*), optional, intent(in) :: form
!*  or
!*       ! Trace output character string and double precision real array
!*       subroutine trace(lvl, txt, var, n, form)
!*         implicit none
!*         integer(i4b), intent(in)               :: lvl
!*         character(len=*), intent(in)           :: txt
!*         real(dp), dimension(:), intent(in)     :: var
!*         integer(i4b), intent(in)               :: n
!*         character(len=*), optional, intent(in) :: form
!*  where
!*
!*         lvl  : Severity level of the trace statement
!*         txt  : character(*) text to be written to active output streams
!*         var  : integer_i4b, real_sp, real_dp or logical variable
!*         n    : integer_i4b number of entries in "var"
!*         form : character(*) format for "var" variable
!*
!* REMARKS :
!*   Logging messages each have an associated severity level that show the
!*   importance of each message. The trace statements with level 'MESSAGE'
!*   will always write to the enabled output streams. The other severity levels
!*   should be interpreted as follows:
!*
!*   FATAL   : Something bad has happened, the system is of little use to anyone.
!*   ERROR   : A specific task may have failed, but the system will keep running.
!*   WARNING : Something went wrong but the system will try to recover from it.
!*   INFO    : Informational messages about what's happening.
!*   DEBUG   : Hints used during the initial coding and debugging process.
!*
!* When no output streams are active, trace statements should have minimal
!* impact on CPU-time. In module 'VXParam_m' the logical parameter 'TRACE_ON'
!* and the integer(i4b) parameter 'LU_TRACE' are defined. The former can set
!* the entire 'Trace_m' module inactive while the latter sets the logical unit
!* number for the logfile.
!*
!* USAGE:
!*   call trace(MESSAGE,'******* Copyright A. van Garrel, 2001 *******')
!*   call trace(FATAL,'Denominator is zero; exit program')
!*   call trace(MESSAGE,'')
!*
!*   call traceInit(lvl=DEBUG, stdout=.TRUE., file=.TRUE., filename='log.txt')
!*
!*   call trace(DEBUG,'Variable i=',i, form='(i3)')
!*   call trace(INFO,'=== Now within subroutine do_something() ===')
!*   call trace(WARNING,'Variable d=',d)
!*
!*****************************************************************************
```

Types_m

```
!*****************************************************************************
!* FILE : Types_m.f90
!*
!* PROGRAM UNIT : module Types_m
!*
!* PURPOSE : Collection of general use kind types, mathematical constants and
!*           tolerance factors.
!*
!* MODULES USED : None
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2001   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   In this module some genral use types are defined together with some
!*   mathematical constants and tolerance factors:
!*
!*   ! Symbolic names for kind types of 4-,2-,and 1-byte integers:
!*   integer, parameter :: i4b = selected_int_kind(9)
!*   integer, parameter :: i2b = selected_int_kind(4)
!*   integer, parameter :: i1b = selected_int_kind(2)
!*
!*   ! Symbolic names for kind types of single-and double-precision reals:
!*   integer, parameter :: sp = kind(1.0)
!*   integer, parameter :: dp = kind(1.0d0)
!*
!*   ! Symbolic names for kind types of single-and double-precision complex:
!*   integer, parameter :: spc = kind((1.0,1.0))
!*   integer, parameter :: dpc = kind((1.0d0,1.0d0))
!*
!*   ! Mathematical constants:
!*   real(sp), parameter :: PI=3.14159265358979323846264338327950288419 7_sp
!*   real(sp), parameter :: PIO2=1.57079632679489661923132169163975144209858_sp
!*   real(sp), parameter :: TWOPI=6.28318530717958647692528676655900576839 4_sp
!*   real(sp), parameter :: SQRT2=1.41421356237309504880168872420969807856967_sp
!*   real(sp), parameter :: EULER=0.57721566490153286060651209008240243104 22_sp
!*   real(dp), parameter :: PI_D=3.14159265358979323846264338327950288419 7_dp
!*   real(dp), parameter :: PIO2_D=1.57079632679489661923132169163975144209858_dp
!*   real(dp), parameter :: TWOPI_D=6.28318530717958647692528676655900576839 4_dp
!*
!*   ! Tolerance factors:
!*   real(dp), parameter :: EPS2=1.0e-2_dp
!*   real(dp), parameter :: EPS3=1.0e-3_dp
!*   real(dp), parameter :: EPS4=1.0e-4_dp
!*   real(dp), parameter :: EPS5=1.0e-5_dp
!*   real(dp), parameter :: EPS6=1.0e-6_dp
!*   real(dp), parameter :: EPS7=1.0e-7_dp
!*   real(dp), parameter :: EPS8=1.0e-8_dp
!*   real(dp), parameter :: EPS16=1.0e-16_dp
!*   real(dp), parameter :: EPS32=1.0e-32_dp
!*
!* REMARKS:
!*   None.
!*
!* USAGE:
!*   real(dp)     :: aDouble = 18.021965_dp
!*   integer(i4b) :: aLongInt = 1
!*
!*****************************************************************************
```

Utils_m

```
!*****************************************************************************
!* FILE : Utils_m.f90
!*
!* PROGRAM UNIT : module Utils_m
!*
!* PURPOSE : Collection of utility program units.
!*
!* MODULES USED : Types_m, VXTypes_m, VXParam_m, Trace_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    8-APR-2001   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   In this module a collection of non-related program units is defined:
!*
!*   ! Convert all letters in a string to uppercase
!*   subroutine to_uppercase(string)
!*     character(len=*), intent(inout) :: string
!*
!*   ! Convert all letters in a string to lowercase:
!*   subroutine to_lowercase(string)
```

```
!*     character(len=*), intent(inout) :: string
!*
!*   ! Convert a value from Degrees to Radians
!*   function deg2rad(d) result(r)
!*     real(dp), intent(in) :: d
!*     real(dp) :: r
!*
!*   ! Convert a value from Radians to Degrees
!*   function rad2deg(r) result(d)
!*     real(dp), intent(in) :: r
!*     real(dp) :: d
!*
!*   ! Determine the order of the entries in a rank-1 array
!*   function orderType(x, n)
!*     real(dp), dimension(:), intent(in) :: x
!*     integer(i4b), intent(in)  :: n
!*     integer(i4b) :: orderType
!*     ! -1, no discernable order
!*     !  0, all entries are equal
!*     !  1, ascending order
!*     !  2, strictly ascending order
!*     !  3, descending order
!*     !  4, strictly descending order
!*
!*   ! Reverse the order of a rank-1 array
!*   subroutine reverseOrder(x, n)
!*     real(dp), dimension(:), intent(inout) :: x
!*     integer(i4b), intent(in)  :: n
!*     ! x(1) --> x(n)
!*     ! x(2) --> x(n-1)
!*     !  :        :
!*     ! x(n) --> x(1)
!*
!*     ! Write a single open tag "Tag" to file with unit number "lu": <Tag>
!*     ! The output string is preceeded by "nspaces" nr. of blanks
!*
!*   subroutine writeOpenTag(lu, nspaces, tag)
!*     integer(i4b), intent(in)     :: lu
!*     integer(i4b), intent(in)     :: nspaces
!*     character(len=*), intent(in) :: tag
!*
!*   ! Write a single close tag "Tag" to file with unit number "lu": </Tag>
!*   ! The output string is preceeded by "nspaces" nr. of blanks
!*   subroutine writeCloseTag(lu, nspaces, tag)
!*     integer(i4b), intent(in)     :: lu
!*     integer(i4b), intent(in)     :: nspaces
!*     character(len=*), intent(in) :: tag
!*
!*   ! Write a single integer that is enclosed by tags to file with unit
!*   ! number "lu":   <Tag> 2002 </Tag>
!*   ! The output string is preceeded by "nspaces" nr. of blanks
!*   subroutine writeOneInteger(lu, nspaces, tag, theInt)
!*     integer(i4b), intent(in)     :: lu
!*     integer(i4b), intent(in)     :: nspaces
!*     character(len=*), intent(in) :: tag
!*     integer(i4b), intent(in)     :: theInt
!*
!*   ! Write a single double that is enclosed by tags to file with unit
!*   ! number "lu":   <Tag> 3.14159 </Tag>
!*   ! The output string is preceeded by "nspaces" nr. of blanks
!*   subroutine writeOneDouble(lu, nspaces, tag, theDouble)
!*     integer(i4b), intent(in)     :: lu
!*     integer(i4b), intent(in)     :: nspaces
!*     character(len=*), intent(in) :: tag
!*     real(dp), intent(in)         :: theDouble
!*
!*   ! Write a 3-component vector that is enclosed by tags to file with unit
!*   ! number "lu":    <Tag> 1.0  2.0  3.0 </Tag>
!*   ! The output string is preceeded by "nspaces" nr. of blanks
!*   subroutine writeOneVec3(lu, nspaces, tag, theVec3)
!*     integer(i4b), intent(in)     :: lu
!*     integer(i4b), intent(in)     :: nspaces
!*     character(len=*), intent(in) :: tag
!*     type(Vec3), intent(in)       :: theVec3
!*
!*   ! Write AWSM identification data to the enabled log output files
!*   subroutine traceAWSMVersion()
!*
!* REMARKS:
!*   None.
!*
!* USAGE:
!*   call to_lowercase(string)
!*   angle = deg2rad(180.0_dp)
!*   iOrder = orderType(x, n)
!*   if ((iOrder==3).or.(iOrder==4)) then
!*     call reverseOrder(x, n)   ! Ordering from decreasing to increasing
!*   endif
!*
!*******************************************************************************
```

VXParam_m

```
!******************************************************************************
!* FILE : VXParam_m.f90
!*
!* PROGRAM UNIT : module VXParam_m
!*
!* PURPOSE : Definition of project AWSM specific constant parameters.
!*
!* MODULES USED : Types_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2001   A. van Garrel   Initial coding
!*
!******************************************************************************
!* SHORT DESCRIPTION :
!*
!*   ! Enable/disable error tracing
!*   logical, parameter, public       :: TRACE_ON = .TRUE.
!*
!*   ! Set the maximum input line length
!*   integer(i4b), parameter, public  :: MAX_LL   = 256
!*
!*   ! Set the logical unit number for the logfile
!*   integer(i4b), parameter, public  :: LU_TRACE         = 99
!*   ! Set the logical unit number for the geometry input file
!*   integer(i4b), parameter, public  :: LU_GEOM          = 98
!*   ! Set the logical unit number for the geometry scenario input file
!*   integer(i4b), parameter, public  :: LU_GEOM_SCENARIO = 97
!*   ! Set the logical unit number for the geometry snapshot file
!*   integer(i4b), parameter, public  :: LU_GEOM_SNAPSHOT = 96
!*   ! Set the logical unit number for the AeroData file
!*   integer(i4b), parameter, public  :: LU_AERO_DATA     = 95
!*   ! Set the logical unit number for the AeroLink file
!*   integer(i4b), parameter, public  :: LU_AERO_LINK     = 94
!*   ! Set the logical unit number for the wind scenario input file
!*   integer(i4b), parameter, public  :: LU_WIND_SCENARIO = 93
!*   ! Set the logical unit number for the wind snapshot file
!*   integer(i4b), parameter, public  :: LU_WIND_SNAPSHOT = 92
!*
!*   ! Set the logical unit number for the geometry snapshot file
!*   integer(i4b), parameter, public  :: LU_GEOM_RESULTS  = 91
!*   ! Set the logical unit number for the AeroData file
!*   integer(i4b), parameter, public  :: LU_AERO_RESULTS  = 90
!*
!*   ! Set the relative chordwise position of the lifting line
!*   real(dp), parameter, public :: CHORD_POSITION_LIFT = 0.25_dp
!*
!*   ! Set the relative streamwise position of the shed vortex
!*   ! Katz&Plotkin suggest a value between 0.2 and 0.3
!*   ! NTUA uses a value of 1.0 in their GENUVP code
!*   real(dp), parameter, public :: POSITION_SHED_VORTEX = 0.25_dp
!*
!*   ! Vortex line 'cut-off' lengths specifying relative distance where vortex
!*   ! induced velocities decrease lineary towards zero
!*   real(dp), parameter :: CUTOFF_LINEAR_MIN=EPS8
!*   real(dp), parameter :: CUTOFF_LINEAR_MAX=0.2_DP
!*
!*   ! Vortex line 'cut-off' lengths squared for desingularized vortex induced velocities
!*   real(dp), parameter :: CUTOFF_SMOOTH_MIN2=EPS16
!*   real(dp), parameter :: CUTOFF_SMOOTH_MAX2=0.2_DP
!*
!*   ! AWSM version identification data
!*   character(len=*), parameter, public :: AWSM_VERSION = "1.0.0"
!*   character(len=*), parameter, public :: AWSM_DATE    = "1-AUG-2003"
!*
!*   ! Set input file character constants
!*   character(*),parameter :: CHARS_COMMENT = '#!'  ! Comment identifier characters
!*   character(*),parameter :: CHARS_STRSEP = '|^"'  ! String identifier characters
!*   character(*),parameter :: OPENTAG1 = '<'        ! Open tag begin identifier
!*   character(*),parameter :: OPENTAG2 = '>'        ! Open tag end identifier
!*   character(*),parameter :: CLOSETAG1 = '</'      ! Close tag begin identifier
!*   character(*),parameter :: CLOSETAG2 = '>'       ! Close tag end identifier
!*
!*   ! Character strings used as tags in an "AeroData"-file
!*   character(len=*), parameter, public :: AERO_DATA_TAG      = "AeroData"
!*   character(len=*), parameter, public :: AERO_DATA_ID_TAG   =   "Id"
!*
!*   character(len=*), parameter, public :: AIRFOIL_TAG        =    "Airfoil"
!*   character(len=*), parameter, public :: AIRFOIL_ID_TAG     =      "Id"
!*   character(len=*), parameter, public :: AIRFOIL_IDNR_TAG   =      "IdNr"
!*
!*   character(len=*), parameter, public :: POLAR_TAG          =      "Polar"
!*   character(len=*), parameter, public :: POLAR_ID_TAG       =        "Id"
!*   character(len=*), parameter, public :: POLAR_IDNR_TAG     =        "IdNr"
!*   character(len=*), parameter, public :: POLAR_RE_TAG       =        "Re"
!*   character(len=*), parameter, public :: POLAR_MA_TAG       =        "Ma"
!*   character(len=*), parameter, public :: ALPHA_CL_TAG       =        "Alpha-Cl"
!*   character(len=*), parameter, public :: ALPHA_CD_TAG       =        "Alpha-Cd"
!*   character(len=*), parameter, public :: ALPHA_CM_TAG       =        "Alpha-Cm"
!*
!*
!*   ! Character strings used as tags in an "AeroLink"-file
!*   character(len=*), parameter, public :: AERO_LINK_TAG      = "AeroLink"
!*   character(len=*), parameter, public :: AERO_LINK_ID_TAG   =   "Id"
```

```
!*
!*   character(len=*), parameter, public :: STRIP_SET_TAG      =   "StripSet"
!*   character(len=*), parameter, public :: SSET_PARTIDNR_TAG  =     "PartIdNr"
!*   character(len=*), parameter, public :: SSET_SEGIDNR_TAG   =     "SegmentIdNr"
!*
!*   character(len=*), parameter, public :: STRIP_LINK_TAG     =      "StripLink"
!*   character(len=*), parameter, public :: STRIP_NR_TAG       =        "StripNr"
!*   character(len=*), parameter, public :: AERO_DATA_FILE_TAG =        "AeroDataFile"
!*   character(len=*), parameter, public :: SLINK_AIRIDNR_TAG  =        "AirfoilIdNr"
!*   character(len=*), parameter, public :: SLINK_POLIDNR_TAG  =        "PolarIdNr"
!*
!*
!*   ! Character strings used as tags in a "Geom"-file
!*   character(len=*), parameter, public :: GEOM_TAG           = "Geom"
!*   character(len=*), parameter, public :: GEOM_ID_TAG        =   "Id"
!*
!*   character(len=*), parameter, public :: PART_TAG           =   "Part"
!*   character(len=*), parameter, public :: PART_ID_TAG        =     "Id"
!*   character(len=*), parameter, public :: PART_IDNR_TAG      =     "IdNr"
!*
!*   character(len=*), parameter, public :: MARKER_TAG         =     "Marker"
!*   character(len=*), parameter, public :: MARKER_ID_TAG      =       "Id"
!*   character(len=*), parameter, public :: MARKER_IDNR_TAG    =       "IdNr"
!*   character(len=*), parameter, public :: MARKER_LOC_TAG     =       "Location"
!*   character(len=*), parameter, public :: MARKER_VEC1_TAG    =       "Vector1"
!*   character(len=*), parameter, public :: MARKER_VEC2_TAG    =       "Vector2"
!*
!*   character(len=*), parameter, public :: SEGMENT_TAG        =     "Segment"
!*   character(len=*), parameter, public :: SEGMENT_ID_TAG     =       "Id"
!*   character(len=*), parameter, public :: SEGMENT_IDNR_TAG   =       "IdNr"
!*   character(len=*), parameter, public :: SEGMENT_NI_TAG     =       "Ni"
!*   character(len=*), parameter, public :: SEGMENT_NJ_TAG     =       "Nj"
!*   character(len=*), parameter, public :: SEGMENT_PNT_TAG    =       "Coordinates"
!*
!*   character(len=*), parameter, public :: LINK_TAG           =   "Link"
!*   character(len=*), parameter, public :: LINK_ID_TAG        =     "Id"
!*   character(len=*), parameter, public :: LINK_IDNR_TAG      =     "IdNr"
!*   character(len=*), parameter, public :: LINK_PPRTIDNR_TAG  =     "ParentPart"
!*   character(len=*), parameter, public :: LINK_PMRKIDNR_TAG  =     "ParentMarker"
!*   character(len=*), parameter, public :: LINK_CPRTIDNR_TAG  =     "ChildPart"
!*   character(len=*), parameter, public :: LINK_CMRKIDNR_TAG  =     "ChildMarker"
!*
!*
!*   ! Character strings used as tags in a "GeomScenario"-file
!*   character(len=*), parameter, public :: GEOM_SCENARIO_TAG         = "GeomScenario"
!*   character(len=*), parameter, public :: GEOM_SCENARIO_ID_TAG      =   "Id"
!*   character(len=*), parameter, public :: MTIME_SERIES_TAG          =   "MarkerTimeSeries"
!*   character(len=*), parameter, public :: INTERPOLATION_TYPE_TAG    =     "InterpolationType"
!*   character(len=*), parameter, public :: INTERPOLATION_TYPE_LINEAR =       "LINEAR"
!*   character(len=*), parameter, public :: INTERPOLATION_TYPE_SPLINE =       "SPLINE"
!*   character(len=*), parameter, public :: EXTRAPOLATION_TAG         =     "Extrapolation"
!*
!*
!*   ! Character strings used as tags in a "GeomSnapshot"-file
!*   character(len=*), parameter, public :: GEOM_SNAPSHOT_TAG   = "GeomSnapshot"
!*   character(len=*), parameter, public :: MTRANSFORM_TAG      =   "MarkerTransformation"
!*   character(len=*), parameter, public :: MTRANS_PARTIDNR_TAG =     "PartIdNr"
!*   character(len=*), parameter, public :: MTRANS_MARKIDNR_TAG =     "MarkerIdNr"
!*   character(len=*), parameter, public :: MTIME_INSTANCE_TAG  =     "MarkerTimeInstance"
!*   character(len=*), parameter, public :: TIME_TAG            =       "Time"
!*   character(len=*), parameter, public :: TRANSLATION_TAG     =       "Translation"
!*   character(len=*), parameter, public :: ROTATION_AXIS_TAG   =       "RotationAxis"
!*   character(len=*), parameter, public :: ROTATION_ANGLE_TAG  =       "RotationAngle"
!*
!*   ! Character strings used as tags in a "WindScenario"-file
!*   character(len=*), parameter, public :: WIND_SCENARIO_TAG         = "WindScenario"
!*   character(len=*), parameter, public :: WIND_SCENARIO_ID_TAG      =   "Id"
!* ! character(len=*), parameter, public :: INTERPOLATION_TYPE_TAG    =   "InterpolationType"
!* ! character(len=*), parameter, public :: INTERPOLATION_TYPE_LINEAR =   "LINEAR"
!* ! character(len=*), parameter, public :: INTERPOLATION_TYPE_SPLINE =   "SPLINE"
!* ! character(len=*), parameter, public :: EXTRAPOLATION_TAG         =   "Extrapolation"
!*
!*   ! Character strings used as tags in a "WindSnapshot"-file
!*   character(len=*), parameter, public :: WIND_SNAPSHOT_TAG   = "WindSnapshot"
!*   character(len=*), parameter, public :: WTIME_INSTANCE_TAG  =   "WindTimeInstance"
!*   character(len=*), parameter, public :: WIND_UVW_TAG        =     "WindUVW"
!* ! character(len=*), parameter, public :: TIME_TAG            =     "Time"
!*
!*
!* REMARKS :
!*   None.
!*
!* USAGE:
!*   No description.
!*
!*******************************************************************************
```

VXTypes_m

```
!*******************************************************************************
```

```
!* FILE : VXTypes_m.f90
!*
!* PROGRAM UNIT : module VXTypes_m
!*
!* PURPOSE : In this module project AWSM specific types are defined
!*
!* MODULES USED : Types_m, VXParam_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2002   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*
!*   ! Type definition for 4x4 matrix in column-major form used for
!*   ! transformation matrices for homogeneous coordinates
!*   type Mat4
!*     real(dp), dimension(4,4) :: a = 0.0_dp
!*   end type Mat4
!*
!*
!*   ! Vector used for 3D coordinates
!*   type Vec3
!*     real(dp), dimension(3) :: x
!*   end type Vec3
!*
!*
!*   ! Vector used for 3D homogeneous coordinates
!*   type Vec4
!*     real(dp), dimension(4) :: x
!*   end type Vec4
!*
!*
!*   ! Generic type for function values
!*   type DataSet
!*     integer(i4b) :: n = 0   ! Nr. of data points
!*     real(dp), dimension(:), pointer :: x ! Abscissa values size(n)
!*     real(dp), dimension(:), pointer :: y ! Function values size(n)
!*   end type DataSet
!*
!*
!*   ! Container for airfoil aerodynamic coefficients as function of angle of attack
!*   type Polar
!*     character(len=MAX_LL) :: id  ! Identification string
!*     integer(i4b)  :: idNr        ! Identification number
!*     real(dp)      :: Re = 0.0_dp ! Reynolds number
!*     real(dp)      :: Ma = 0.0_dp ! Mach number
!*     type(DataSet) :: aCl         ! Alpha - Cl data
!*     type(DataSet) :: aCd         ! Alpha - Cd data
!*     type(DataSet) :: aCm         ! Alpha - Cm data
!*   end type Polar
!*
!*
!*   ! Container for aerodynamic "Polar"s for a specific "Airfoil"
!*   type Airfoil
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)    :: idNr     ! Identification number
!*     integer(i4b)    :: nPol     ! Nr. of Polars > 0
!*     type(Polar), dimension(:), pointer :: polar ! Polars size(nPol)
!*   end type Airfoil
!*
!*
!*   ! Container for "Airfoils"
!*   type AeroData
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)    :: nAir     ! Nr. of Airfoils > 0
!*     type(Airfoil), dimension(:), pointer :: airfoil ! Airfoils size(nAir)
!*   end type AeroData
!*
!*
!*   ! Type linking "Airfoil"-"Polar"s with a specific strip in a "Segment"
!*   type StripLink
!*     integer(i4b) :: sNr = 0       ! Strip number
!*     character(len=MAX_LL) :: adFn ! AeroData filename
!*     integer(i4b) :: airIdNr       ! Airfoil identification number
!*     integer(i4b) :: polIdNr       ! Polar identification number
!*   end type StripLink
!*
!*
!*   ! Container of "StripLink"s for ALL strips in a "Segment"
!*   type StripSet
!*     integer(i4b)  :: pIdNr     ! Part identification number
!*     integer(i4b)  :: sIdNr     ! Segment identification number
!*     integer(i4b)  :: nSLnk = 0 ! Nr. of strip links
!*     type(StripLink), dimension(:), pointer  :: stripLnk ! Links for ALL strips
!*   end type StripSet
!*
!*
!*   ! Container of "StripSets" for ALL "Segment"s
!*   type AeroLink
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)  :: nSSet = 0  ! Nr. of strip sets
!*     type(StripSet), dimension(:), pointer  :: stripSet ! Strip sets for ALL segments
!*   end type AeroLink
!*
```

```
!*
!*   ! Vortex line definition
!*   type LiftingLine
!*     integer(i4b) :: nj   ! Nr. of points in J direction
!*     type(Vec3), dimension(:,:), pointer :: pnt       ! Coordinates size(2,nj)
!*     type(Vec3), dimension(:), pointer   :: cpnt      ! Control points size(nj-1)
!*     type(Vec3), dimension(:), pointer   :: cpnt_old  ! Old control points size(nj-1)
!*     real(dp), dimension(:), pointer     :: aref      ! Reference areas size(nj-1)
!*     real(dp), dimension(:), pointer     :: lref      ! Reference lengths size(nj-1)
!*     type(Vec3), dimension(:,:), pointer :: axis      ! Coordinate systems size(3,nj-1)
!*     type(Polar), dimension(:), pointer  :: polar     ! Airfoil polar data size(nj-1)
!*     real(dp), dimension(:), pointer     :: gam       ! Vortex strength size(nj-1)
!*     real(dp), dimension(:), pointer     :: gam_old   ! Vortex strength size(nj-1)
!*     real(dp), dimension(:), pointer     :: wrk1      ! Work array size(nj-1)
!*     real(dp), dimension(:), pointer     :: wrk2      ! Work array size(nj-1)
!*     real(dp), dimension(:), pointer     :: alpha     ! Angle of attack size(nj-1)
!*     real(dp), dimension(:), pointer     :: cl        ! Lift coefficient size(nj-1)
!*     real(dp), dimension(:), pointer     :: cd        ! Lift coefficient size(nj-1)
!*     real(dp), dimension(:), pointer     :: cm        ! Lift coefficient size(nj-1)
!*     type(Vec3), dimension(:), pointer   :: df_inert  ! Force vector at control point
!*                                                        in inertial coordinate system
!*     type(Vec3), dimension(:), pointer   :: dm_inert  ! Pitching moment vector at control
!*                                                        point in inertial coordinate system
!*     type(Vec3), dimension(:), pointer   :: df_part   ! Force vector at control point
!*                                                        in Part's coord. system
!*     type(Vec3), dimension(:), pointer   :: dm_part   ! Pitching moment vector at control
!*                                                        point in Part's coord. system
!*     type(Vec3), dimension(:), pointer   :: uvwWind   ! Wind velocity at control point
!*     type(Vec3), dimension(:), pointer   :: uvwMotion ! Motion related velocity at
!*                                                        control point
!*     type(Vec3), dimension(:), pointer   :: uvwNW3N   ! Near wake strips 3 to N velocity
!*                                                        at control point
!*     type(Vec3), dimension(:), pointer   :: uvwNW12   ! Near wake strips 1 and 2 velocity
!*                                                        at control point
!*     type(Vec3), dimension(:), pointer   :: uvwLLine  ! LiftingLine induced velocity at
!*                                                        control point
!*   end type LiftingLine
!*
!*
!*   ! A vortex-lattice representation of the near wake of a "Segment"
!*   type NearWake
!*     integer(i4b) :: max_ni ! Maximum nr. of points in streamwise direction
!*     integer(i4b) :: ni ! Nr. of points in streamwise direction
!*     integer(i4b) :: nj ! Nr. of points in spanwise direction
!*     type(Vec3), dimension(:,:), pointer :: pnt ! Geometry size(max_ni,nj)
!*     type(Vec3), dimension(:,:), pointer :: uvw ! Velocity size(max_ni,nj)
!*     real(dp), dimension(:,:), pointer :: gam   ! Vortex strength size(max_ni-1,nj-1)
!*   end type NearWake
!*
!*
!*   ! Discrete representation of an aerodynamic force carrying surface
!*   type Segment
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)      :: idNr   ! Identification number
!*     integer(i4b)      :: ni, nj ! Nr. of points in I, J directions
!*     type(LiftingLine) :: lline  ! Lifting line data
!*     type(NearWake)    :: nwake  ! Near wake data
!*     type(Vec3), dimension(:), pointer   :: te_old   ! Old trailing points size(nj)
!*     type(Vec3), dimension(:,:), pointer :: pnt_init ! Initial points size(ni,nj)
!*     type(Vec3), dimension(:,:), pointer :: pnt      ! Actual points size(ni,nj)
!*   end type Segment
!*
!*
!*   ! Position and orientation containing node
!*   type Marker
!*     character(len=MAX_LL) :: id  ! Identification string
!*     integer(i4b)    :: idNr      ! Identification number
!*     type(Vec3)      :: loc_init  ! Initial Marker location
!*     type(Vec3)      :: vec1_init ! Initial Marker Vector1
!*     type(Vec3)      :: vec2_init ! Initial Marker Vector2
!*     type(Vec3)      :: loc       ! Actual Marker location
!*     type(Vec3)      :: vec1      ! Actual Marker Vector1
!*     type(Vec3)      :: vec2      ! Actual Marker Vector2
!*   end type Marker
!*
!*
!*   ! Container for "Segment"s and/or "Marker"s
!*   type Part
!*     character(len=MAX_LL) :: id  ! Identification string
!*     integer(i4b)    :: idNr      ! Identification number
!*     integer(i4b)    :: nSeg = 0  ! Nr. of Segments >=0
!*     integer(i4b)    :: nMrk = 0  ! Nr. of Markers  > 0
!*     integer(i4b)    :: npLnk = 0 ! Nr. of parent Links  0,1
!*     integer(i4b)    :: ncLnk = 0 ! Nr. of child Links   0,1
!*     integer(i4b)    :: pLnkIdx   ! Parent Link index
!*     integer(i4b)    :: cLnkIdx   ! Child Link index
!*     type(Segment), dimension(:), pointer :: seg ! Segments size(nSeg)
!*     type(Marker),  dimension(:), pointer :: mrk ! Markers size(nMrk)
!*   end type Part
!*
!*
!*   ! Geometric relation between two "Marker"s
!*   type Link
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)    :: idNr     ! Identification number
!*
```

```
!*     integer(i4b)    :: pPrtIdNr ! parent Part IdNr
!*     integer(i4b)    :: pMrkIdNr ! parent Part Marker IdNr
!*     integer(i4b)    :: pPrtIdx  ! parent Part
!*     integer(i4b)    :: pMrkIdx  ! parent Part Marker
!*
!*     integer(i4b)    :: cPrtIdNr ! child Part IdNr
!*     integer(i4b)    :: cMrkIdNr ! child Part Marker IdNr
!*     integer(i4b)    :: cPrtIdx  ! child Part
!*     integer(i4b)    :: cMrkIdx  ! child Part Marker
!*
!*     type(Mat4) :: c2p   ! Child to parent transformation matrix
!*     type(Mat4) :: p2c   ! Parent to child transformation matrix
!*   end type Link
!*
!*
!*   ! Container for "Part"s and "Link"s
!*   type Geom
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)    :: idNr     ! Identification number
!*     integer(i4b)    :: nPrt     ! Nr. of Parts >0
!*     integer(i4b)    :: nLnk     ! Nr. of Links <=nPrt-1
!*     type(Part), dimension(:), pointer :: prt ! Parts size(nPrt)
!*     type(Link), dimension(:), pointer :: lnk ! Links size(nLnk)
!*   end type Geom
!*
!*
!*   ! Generic character string
!*   type TextLine
!*     character(len=MAX_LL) :: text = ''
!*     integer(i4b) :: tail = 0  ! Last character position
!*   end type TextLine
!*
!*
!*   ! File wrapper
!*   type InputFile
!*     integer(i4b)          :: lu   ! File unit number
!*     character(len=MAX_LL) :: fn   ! Filename
!*     type(TextLine)        :: line ! Current line contents
!*     integer(i4b)          :: lnr  ! Current line number
!*   end type InputFile
!*
!*
!*   ! Marker transformation data at a specific instance in time
!*   type MTimeInst
!*     real(dp)    :: time
!*     type(Vec3)  :: shift
!*     type(Vec3)  :: rotAxis
!*     real(dp)    :: rotAngle   ! Angle in degrees
!*   end type MTimeInst
!*
!*
!*   ! A temporal sequence of "MTimeInst"s and the type of inter/extrapolation.
!*   type MTimeSeries
!*     integer(i4b)    :: partIdNr    ! Part identification number
!*     integer(i4b)    :: markIdNr    ! Marker identification number
!*     logical         :: extrapol    ! Marker data extrapolation flag
!*     character(len=MAX_LL) :: interpolTyp ! Type of interpolation
!*     integer(i4b)    :: nMTI = 0    ! Nr. of MTimeInst > 1
!*     type(MTimeInst), dimension(:), pointer :: mTI
!*   end type MTimeSeries
!*
!*
!*   ! A container of "MTimeSeries"
!*   type GeomScenario
!*     character(len=MAX_LL) :: id ! Identification string
!*     integer(i4b)    :: nMTS = 0  ! Nr. of MTimeSeries >0
!*     type(MTimeSeries), dimension(:), pointer :: mTS
!*   end type GeomScenario
!*
!*
!*   ! A "MTimeInst" for a specific "Marker"
!*   type MTransform
!*     integer(i4b)    :: partIdNr
!*     integer(i4b)    :: markIdNr
!*     type(MTimeInst) :: mTI
!*   end type MTransform
!*
!*
!*   ! A container for "MTransform"s
!*   type GeomSnapshot
!*     integer(i4b)    :: nMTrans ! >=0
!*     type(MTransform), dimension(:), pointer :: mTrans
!*   end type GeomSnapshot
!*
!*
!*   ! Wind velocity data at a specific instance in time
!*   type WTimeInst
!*     real(dp)    :: time
!*     type(Vec3)  :: uvwWind
!*   end type WTimeInst
!*
!*   ! A temporal sequence of "WTimeInst"s and the type of inter/extrapolation.
!*   type WindScenario
!*     character(len=MAX_LL) :: id          ! Identification string
!*     character(len=MAX_LL) :: interpolTyp ! Type of interpolation
!*     logical :: extrapol                  ! Wind data extrapolation flag
```

```
!*     integer(i4b) :: nWTI = 0             ! Nr. of WTimeInst > 1
!*     type(WTimeInst), dimension(:), pointer :: wTI
!*   end type WindScenario
!*
!*   ! A container for one "WTimeInst"
!*   type WindSnapshot
!*     type(WTimeInst) :: wTI ! A WTimeInst at the current time
!*   end type WindSnapshot
!*
!* REMARKS :
!*   None.
!*
!* USAGE:
!*   No description.
!*
!*****************************************************************************
```

Vec34_m

```
!*****************************************************************************
!* FILE : Vec34_m.f90
!*
!* PROGRAM UNIT : module Vec34_m
!*
!* PURPOSE : This module defines 3D vectors, their associated 4D homogeneous
!*           forms and mathematical operations for the 3D vectors
!*
!* MODULES USED : Types_m, VXTypes_m, Trace_m, VXParam_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    5-DEC-2001   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*
!* Interface definitions
!*
!*   Vec3 type creation/conversion functions
!*   interface toVec3
!*     module procedure toVec3_x123, toVec3_v4, toVec3_a3
!*   end interface toVec3
!*   public :: toVec3
!*   private :: toVec3_x123, toVec3_v4, toVec3_a3!*
!*
!*   Vec4 type creation/conversion functions
!*   interface toVec4
!*      module procedure toVec4_x1234, toVec4_v3, toVec4_a4
!*   end interface toVec4
!*   public :: toVec4
!*   private :: toVec4_x1234, toVec4_v3, toVec4_a4
!*
!*   Conversion of Vec3 and Vec4 types to regular arrays
!*   interface toArray
!*      module procedure toArray_v3, toArray_v4
!*   end interface toArray
!*   public :: toArray
!*   private :: toArray_v3, toArray_v4
!*
!* Operators that are defined for Vec3 variables are :
!*   =      : assignment operator
!*   +      : addition operator
!*   -      : negation/subtraction operator
!*   *      : multiplication operator (with scalar)
!*   /      : division operator (by scalar)
!*  .dot.   : vector dot product
!*  .cross. : vector cross product
!*
!* Return the angle between two vectors in radians
!*   function angle(aVec1, aVec2)
!*     type(Vec3), intent(in) :: aVec1, aVec2
!*     real(dp) :: angle
!*
!* Return the normalized vector
!*   function unit(aVec3)
!*     type(Vec3) :: unit
!*     type(Vec3),intent(in) ::aVec3
!*
!* Return the length of a 3D vector
!*   function length(aVec3)
!*     type(Vec3), intent(in) :: aVec3
!*     real(dp) :: length
!*
!* Return a 3D vector with the specified components
!*   function toVec3_x123(x1, x2, x3)
!*     type(Vec3) :: toVec3_x123
!*     real(dp), optional, intent(in) :: x1,x2,x3
!*
!* Convert a Vec4 to a Vec3 variable
```

```
!*   function toVec3_v4(aVec4)
!*     type(Vec3) :: toVec3_v4
!*     type(Vec4), intent(in) :: aVec4
!*
!* Convert a regular array to a Vec3 variable
!*   function toVec3_a3(anArray)
!*     type(Vec3) :: toVec3_a3
!*     real(dp), dimension(3), intent(in) :: anArray
!*
!*   function toVec4_v3(aVec3)
!*     type(Vec4) :: toVec4_v3
!*     type(Vec3), intent(in) :: aVec3
!*
!* Convert a regular array to a Vec4 variable
!*   function toVec4_a4(anArray)
!*     type(Vec4) :: toVec4_a4
!*     real(dp), dimension(4), intent(in) :: anArray
!*
!* Create a Vec4 variable from its components
!*   function toVec4_x1234(x1, x2, x3, x4)
!*     type(Vec4) :: toVec4_x1234
!*     real(dp), optional, intent(in) :: x1,x2,x3,x4
!*
!* Convert a Vec3 variable to a regular array
!*   function toArray_v3(aVec3)
!*     real(dp), dimension(3) :: toArray_v3
!*     type(Vec3) , intent(in) :: aVec3
!*
!* Convert a Vec4 variable to a regular array
!*   function toArray_v4(aVec4)
!*     real(dp), dimension(4) :: toArray_v4
!*     type(Vec4) , intent(in) :: aVec4
!*
!* REMARKS:
!*   None.
!*
!* USAGE:
!*   aVec4 = toVec4(1.0_dp, 1.2_dp, 101.1_dp, 1.0_dp)
!*   ! Do a Vec3 negation, division, addition and multiplication
!*   cVec3 = (-aVec3/16.2_dp + 3.4_dp*bVec3)
!*   ! Determine the cross product of two Vec3 vectors
!*   dVec3 = aVec3.cross.bVec3
!*   ! Compute the length from the dot product of a Vec3 vector with itself
!*   r = sqrt(dVec3.dot.dVec3)
!*   ! Normalize the vector
!*   dVec3 = unit(dVec3)
!*   ! Compute the length with a function call
!*   r = length(dVec3)
!*
!*******************************************************************************
```

VortexLattice_m

```
!*******************************************************************************
!* FILE : VortexLattice_m.f90
!*
!* PROGRAM UNIT : module VortexLattice_m
!*
!* PURPOSE : Module VortexLattice_m handles operations on the vortex system
!*           that is attached to a "Segment"s surface
!*
!* MODULES USED : Types_m, VXTypes_m, VXParam_m, AeroData_m, Vec34_m, Wind_m,
!*                Spline_m, DataInput_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    1-JAN-2002   A. van Garrel   Initial coding
!*
!*******************************************************************************
!* SHORT DESCRIPTION :
!*
!*   Public subroutines are:
!*
!*     ! Set lifting line from actual geometry and old t.e. location
!*     subroutine setLiftingLineGeom(aVar, iostat)
!*       type(Segment / Geom), intent(inout) :: aVar
!*       integer(i4b), intent(inout) :: iostat
!*
!*     ! Save current actual control points to the old control points data array
!*     subroutine saveControlPoints(aVar)
!*       type(Segment / Geom), intent(inout) :: aVar
!*
!*     ! Deallocate all arrays that were allocated in the "LiftingLine" object
!*     subroutine deallocateLiftingLineArrays(aLLine)
!*       type(LiftingLine), intent(inout) :: aLLine
!*       integer(i4b) :: status, iPol
!*
!*     ! Allocate storage for a LiftingLine object
!*     subroutine allocateLiftingLineArrays(aLLine, nj)
```

```
!*       type(LiftingLine), intent(inout) :: aLLine
!*       integer(i4b), intent(in) :: nj
!*
!*     ! Compute the induced velocity associated with the given vortex line element.
!*     ! The cut-off type determines whether to use a linear or smooth function
!*     function uvwVortexLine(vtxStart, vtxEnd, gamma, cut-off, cutoffType, point) result(uvw)
!*       type(Vec3), intent(in)   :: vtxStart ! Startpoint of vortex line
!*       type(Vec3), intent(in)   :: vtxEnd   ! Endpoint of vortex line
!*       real(dp), intent(in)     :: gamma    ! Vortex strength
!*       real(dp), intent(in)     :: cut-off   ! Cut-off radius fraction 0.0<cut-off<0.05
!*       integer(i4b), intent(in) :: cutoffType  ! Linear (1) or smoothed (2) velocities
!*       type(Vec3), intent(in)   :: point    ! Evaluation point
!*       type(Vec3) :: uvw ! "Induced" velocity by vortex line at "point"
!*
!*     ! Deallocate all arrays that were allocated in the "NearWake" object
!*     subroutine deallocateNearWakeArrays(aNWake)
!*       type(NearWake), intent(inout) :: aNWake
!*       integer(i4b) :: status
!*
!*     ! Allocate storage for all NearWake objects
!*     subroutine allocateNearWakeArrays(aGeom, max_ni)
!*       type(Geom), intent(inout) :: aGeom
!*       integer(i4b), intent(in)  :: max_ni
!*
!*     ! Set lifting line geometry from actual geometry
!*     subroutine setLiftingLineGeom_seg(aSeg, cpnt_mid, iostat)
!*       type(Segment), intent(inout) :: aSeg
!*       logical, intent(in)          :: cpnt_mid
!*       integer(i4b), intent(inout)  :: iostat
!*
!*     ! Set lifting line geometry from actual geometry
!*     subroutine setLiftingLineGeom_geom(aGeom, cpnt_mid, iostat)
!*       type(Geom), intent(inout)   :: aGeom
!*       logical, intent(in)         :: cpnt_mid
!*       integer(i4b), intent(inout) :: iostat
!*       integer(i2b) :: ip, is
!*
!*     ! Set actual vortex strengths equal to "old" vortex strengths
!*     subroutine setLiftingLineGamma(aGeom)
!*       type(Geom), intent(inout)   :: aGeom
!*       integer(i2b) :: ip, is
!*
!*     ! Save "LiftingLine"s current vortex strengths into "old" data array
!*     subroutine saveLiftingLineGamma(aGeom)
!*       type(Geom), intent(inout)   :: aGeom
!*       integer(i2b) :: ip, is
!*
!*     ! Save current actual control points to the old control points data array
!*     ! NOTE: It is assumed that the control points exist, i.e. setLiftingLineGeom()
!*     ! should have been called at least once
!*     subroutine saveControlPoints_seg(aSeg)
!*       type(Segment), intent(inout) :: aSeg
!*
!*     ! Save current actual control points to the old control points data array
!*     ! NOTE: It is assumed that the control points exist, i.e. setLiftingLineGeom()
!*     ! should have been called at least once
!*     subroutine saveControlPoints_geom(aGeom)
!*       type(Geom), intent(inout) :: aGeom
!*     ! Set the 1st "NearWake" strip geometry (i.e. the strip connected to the t.e.)
!*     ! from actual and "old" t.e.
!*     subroutine setNearWakeStrip1Geom(aGeom)
!*       type(Geom), intent(inout)   :: aGeom
!*
!*     ! Set the 1st "NearWake" strip vortex strengths equal to actual vortex
!*     ! strengths of "LiftingLine" object
!*     subroutine setNearWakeStrip1Gamma(aGeom)
!*       type(Geom), intent(inout)   :: aGeom
!*
!*     ! Convect free near wake vortex lines with wind velocity and possibly
!*     ! transfer oldest vortex rings to the far wake
!*     ! Point indices in streamwise direction are "convected" also
!*     subroutine convectNearWake(aGeom, prevTime, curTime, iostat)
!*       type(Geom), intent(inout)   :: aGeom
!*       real(dp), intent(in)        :: prevTime
!*       real(dp), intent(in)        :: curTime
!*       integer(i4b), intent(inout) :: iostat
!*
!*     ! Compute wind velocities at control points
!*     subroutine uvwCPointWind(aGeom, prevTime, curTime)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: prevTime
!*       real(dp), intent(in)      :: curTime
!*
!*     ! Compute motion related velocities felt at control points
!*     subroutine uvwCPointMotion(aGeom, prevTime, curTime, iostat)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: prevTime
!*       real(dp), intent(in)      :: curTime
!*       integer(i4b), intent(out) :: iostat
!*
!*     ! Compute "induced" velocities of near wake (minus 2 newest strips) at control points
!*     subroutine uvwCPointNearWakeStrips3N(aGeom, cut-off, cutoffType)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: cut-off     ! Cut-off radius fraction 0.0<cut-off<0.05
!*       integer(i4b), intent(in)  :: cutoffType ! Linear (1) or smoothed (2) velocities
!*
```

```
!*     ! Compute "induced" velocities of near wake's 2 newest strips on control points
!*     subroutine uvwCPointNearWakeStrips12(aGeom, cut-off, cutoffType)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: cut-off     ! Cut-off radius fraction 0.0<cut-off<0.05
!*       integer(i4b), intent(in)  :: cutoffType ! Linear (1) or smoothed (2) velocities
!*
!*     ! Compute "LiftingLine" induced velocities at control points
!*     subroutine uvwCPointLiftingLine(aGeom, cut-off, cutoffType)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: cut-off     ! Cut-off radius fraction 0.0<cut-off<0.05
!*       integer(i4b), intent(in)  :: cutoffType ! Linear (1) or smoothed (2) velocities
!*
!*
!*     ! Determine "LiftingLine" vortex strengths from local velocities and airfoil "Polar"s
!*     subroutine updateLiftingLineGamma(aGeom, relax, relax_end, converged)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(inout)   :: relax     ! Relaxation factor for circulation update
!*       real(dp), intent(in)      :: relax_end ! Extra relaxation at segment outer 10%
!*       logical, intent(out)      :: converged
!*
!*     ! Set all near wake velocities to zero
!*     subroutine initUVWNearWake(aGeom)
!*       type(Geom), intent(inout) :: aGeom
!*
!*     ! Compute and add "LiftingLine" induced velocities at near wake points
!*     subroutine uvwNWLiftingLine(aGeom, cut-off, nFree, cutoffType)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: cut-off     ! Cut-off radius fraction 0.0<cut-off<0.05
!*       integer(i4b), intent(in)  :: nFree      ! Number of 'free' streamwise wakepoints
!*       integer(i4b), intent(in)  :: cutoffType ! Linear (1) or smoothed (2) velocities
!*
!*     ! Compute "induced" velocities of near wake at near wake points
!*     subroutine uvwNWNearWake(aGeom, cut-off, nFree, cutoffType)
!*       type(Geom), intent(inout) :: aGeom
!*       real(dp), intent(in)      :: cut-off     ! Cut-off radius fraction 0.0<cut-off<0.05
!*       integer(i4b), intent(in)  :: nFree      ! Number of 'free' streamwise wakepoints
!*       integer(i4b), intent(in)  :: cutoffType ! Linear (1) or smoothed (2) velocities
!*
!*     ! Rollup near wake points with local velocity
!*     subroutine rollupNearWake(aGeom, prevTime, curTime, nFree, iostat)
!*       type(Geom), intent(inout)   :: aGeom
!*       real(dp), intent(in)        :: prevTime
!*       real(dp), intent(in)        :: curTime
!*       integer(i4b), intent(in)    :: nFree     ! Number of 'free' streamwise wakepoints
!*       integer(i4b), intent(inout) :: iostat
!*
!*     ! Compute the forces and moments at each lifting line control point in global
!*     ! and Part's local coordinate systems from current velocities and vortex strength
!*     subroutine computeForces(aGeom)
!*       type(Geom), intent(inout) :: aGeom
!*
!* REMARKS :
!*   None.
!*
!* USAGE:
!*   No description.
!*
!*****************************************************************************
```

WindScenario_m

```
!*****************************************************************************
!* FILE : WindScenario_m.f90
!*
!* PROGRAM UNIT : module WindScenario_m
!*
!* PURPOSE : The WindScenario_m module supplies wind velocity subroutines
!*
!* MODULES USED : Types_m, VXTypes_m, Utils_m, Trace_m, InputFile_m, Vec34_m,
!*                VXParam_m, Spline_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    9-NOV-2002   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   This module supplies basic wind velocity subroutines
!*
!*   Public subroutines:
!*
!*     ! Create a new WindScenario object from a given InputFile object that is
!*     ! wrapped around a wind scenario file
!*     subroutine nextWindScenario(anIF, aWSc, iostat)
!*       type(InputFile), intent(inout)    :: anIF
!*       type(WindScenario), intent(inout) :: aWSc
!*       integer(i4b), intent(out)         :: iostat
!*
!*     ! Create a new WindSnapshot object at a specific time from a given
```

```
!*     ! WindScenario object
!*     subroutine makeWindSnapshot(aWSc, aTime, theWSn)
!*       type(WindScenario), intent(in) :: aWSc
!*       real(dp), intent(in)           :: aTime
!*       type(WindSnapshot), intent(out):: theWSn
!*
!* REMARKS :
!*   An example wind scenario file is:
!*
!*   ! File: windscenario.dat
!*   ! File containing wind velocity vectors at discrete instances of time.
!*   ! This velocity is assumed to be valid for the whole domain.
!*   ! Values at a specified moment in time are obtained through interpolation.
!*
!*   <WindScenario>
!*     <Id> "Linear increasing wind" </Id>
!*     <InterpolationType> "LINEAR" </InterpolationType>
!*     <Extrapolation> YES </Extrapolation>
!*
!*     <WindTimeInstance>
!*       <Time> 0.0 </Time>
!*       <WindUVW> 0.0  0.0  0.0 </WindUVW>
!*     </WindTimeInstance>
!*
!*     <WindTimeInstance>
!*       <Time> 50.0 </Time>
!*       <WindUVW> 1.0  0.0  0.1 </WindUVW>
!*     </WindTimeInstance>
!*
!*     <WindTimeInstance>
!*       <Time> 100.0 </Time>
!*       <WindUVW> 1.0  0.0  0.1 </WindUVW>
!*     </WindTimeInstance>
!*   </WindScenario>
!*
!*
!* USAGE:
!*   No description.
!*
!*******************************************************************************
```

WindSnapshot_m

```
!*******************************************************************************
!* FILE : WindSnapshot_m.f90
!*
!* PROGRAM UNIT : module WindSnapshot_m
!*
!* PURPOSE : The WindSnapshot_m module enables wind snapshot handling; the
!*           wind velocity components at a specific instance in time.
!*
!* MODULES USED : Types_m, VXTypes_m, Utils_m, Trace_m, InputFile_m
!*                Vec34_m, Mat4_m, VXParam_m, DataInput_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    9-NOV-2002    A. van Garrel   Initial coding
!*
!*******************************************************************************
!* SHORT DESCRIPTION :
!*   This module supplies wind snapshot handling subroutines
!*
!*   Public subroutines:
!*
!*     ! Create a new WTimeInst object from a given InputFile object
!*     subroutine nextWTimeInst(anIF, aWTI, iostat)
!*       type(InputFile), intent(inout) :: anIF
!*       type(WTimeInst), intent(inout) :: aWTI
!*       integer(i4b), intent(out)      :: iostat
!*
!*     ! Create a new WindSnapshot object from a given InputFile object
!*     ! that is wrapped around a wind snapshot file
!*     subroutine nextWindSnapshot(anIF, theWSn, iostat)
!*       type(InputFile), intent(inout)     :: anIF
!*       type(WindSnapshot), intent(inout)  :: theWSn
!*       integer(i4b), intent(out)          :: iostat
!*
!*     ! Write a given WindSnapshot object to file
!*     subroutine writeWindSnapshot(theWSn, iostat)
!*       type(WindSnapshot), intent(in) :: theWSn
!*       integer(i4b), intent(out)      :: iostat
!*
!* REMARKS :
!*   An example wind snapshot file is:
!*
!*     <WindSnapshot>
!*       <WindTimeInstance>
!*         <Time> 10.0 </Time>
!*         <WindUVW> 0.2  0.0  0.02 </WindUVW>
```

```
!*       </WindTimeInstance>
!*     </WindSnapshot>
!*
!* USAGE:
!*   No description.
!*
!*****************************************************************************
```

Wind_m

```
!*****************************************************************************
!* FILE : Wind_m.f90
!*
!* PROGRAM UNIT : Wind_m
!*
!* PURPOSE : The Wind_m module's purpose is to provide functions that return
!*           wind velocity vectors at specified points at a specific instance
!*           in time
!*
!* MODULES USED : Types_m, VXTypes_m, Vec34_m, Trace_m, WindSnapshot_m
!*
!* UPDATE HISTORY :
!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
!* 1    8-MAR-2002   A. van Garrel   Initial coding
!*
!*****************************************************************************
!* SHORT DESCRIPTION :
!*   Given a WindSnapshot instance store the wind velocity vector module
!*   private variable.
!*     subroutine setWindUVW(aWSn)
!*       type(WindSnapshot) :: aWSn
!*
!*   Return the wind velocity vector at the specified position and time as a
!*   Vec3 type variable
!*     function uvwWind(position, time) result(uvw)
!*       type(Vec3), intent(in) :: position
!*       real(dp), intent(in)   :: time
!*       type(Vec3) :: uvw
!*
!* REMARKS :
!*   No description
!*
!* USAGE:
!*   No description
!*
!*****************************************************************************
```

B FORTRAN90 SUBROUTINES

In this appendix all Fortran90 interfaces, subroutines and functions from the AWSM project are listed under their containing module name. Their names should give a good indication of their functionality.

Note: Fortran90 `PRIVATE` methods are marked by a tilde `~`.

AeroData_m

```
   subroutine nextAeroData(anIF, anAD, iostat)
   subroutine checkAeroData(anAD, iostat)
   subroutine deallocatePolarArrays(aPolar)
   subroutine copyPolar(anAD, airIdNr, polIdNr, targetPolar, iostat)
~  subroutine nextPolar(anIF, aPolar, iostat)
~  subroutine nextAirfoil(anIF, anAirfoil, iostat)
~  subroutine deallocateAirfoilArrays(anAirfoil)
~  subroutine deallocateAeroDataArrays(anAD)
~  subroutine addPolar(theAirfoil, aPolar)
~  subroutine addAirfoil(theAD, anAirfoil)
```

AeroLink_m

```
   subroutine nextAeroLink(anIF, anALnk, iostat)
   subroutine checkAeroLink(anALnk, iostat)
   subroutine applyAeroLink(aGeom, anALnk, iostat)
~  subroutine nextStripLink(anIF, aStripLnk, iostat)
~  subroutine nextStripSet(anIF, aStripSet, iostat)
~  subroutine deallocateStripSetArrays(aStripSet)
~  subroutine deallocateAeroLinkArrays(anALnk)
~  subroutine addStripLink(theStripSet, aStripLink)
~  subroutine addStripSet(theAL, aStripSet)
```

Ask_m

```
   interface ask
   ~ recursive subroutine ask_i(text, var, min, max, flagChars, flagOut)
   ~ recursive subroutine ask_r(text, var, min, max, flagChars, flagOut)
   ~ recursive subroutine ask_d(text, var, min, max, flagChars, flagOut)
   ~ recursive subroutine ask_l(text, var, flagChars, flagOut)
   ~ recursive subroutine ask_c(text, var, flagChars, flagOut)
```

DataInput_m

```
   subroutine userInput()
   subroutine traceUserInput()
~  subroutine setDefaults()
~  subroutine echoDefaults()
```

DataOutput_m

```
   subroutine dataOutput(aGeom, iTime, prevTime, curTime)
   subroutine writeGeomResults(aGeom, iTime, curTime, iTimeStart, iTimeEnd, iTimeStep)
   subroutine writeAeroResults(aGeom, iTime, curTime, iTimeStart, iTimeEnd, iTimeStep)
   subroutine openResultsFile(lu, filename)
   subroutine closeResultsFile(lu)
```

GeomScenario_m

```
   subroutine nextGeomScenario(anIF, aGSc, iostat)
   subroutine checkGeomScenario(aGSc, aGeom, iostat)
   subroutine makeGeomSnapshot(aGSc, aTime, theGSn)
 ~ subroutine nextMTimeSeries(anIF, aMTS, iostat)
 ~ subroutine deallocateMTimeSeriesArrays(aMTS)
 ~ subroutine deallocateGeomScenarioArrays(aGSc)
 ~ subroutine addMTimeInst(theMTS, aMTI)
 ~ subroutine addMTimeSeries(theGSc, aMTS)
```

GeomSnapshot_m

```
   subroutine nextMTimeInst(anIF, aMTI, iostat)
   subroutine nextGeomSnapshot(anIF, theGSn, iostat)
   subroutine writeGeomSnapshot(theGSn, iostat)
   subroutine deallocateGeomSnapshotArrays(aGSn)
 ~ subroutine nextMTransform(anIF, theMTrans, iostat)
 ~ subroutine addMTransform(theGSn, aMTrans)
```

Geom_m

```
   subroutine nextGeom(anIF, aGeom, iostat)
   subroutine checkGeomIdNrs(aGeom, iostat)
   subroutine activateGeomLinks(aGeom, iostat)
   subroutine searchPMIndex(aGeom, pIdNr, mIdNr, iPart, iMarker, iostat)
   interface saveTrailingEdge
   ~ subroutine saveTrailingEdge_seg(aSegment)
   ~ subroutine saveTrailingEdge_geom(aGeom)
   subroutine applyGeomSnapshot(aGSn, theGeom, iostat)
   subroutine convectOldTrailingEdge(aGeom, prevTime, curTime, iostat)
 ~ subroutine nextMarker(anIF, aMarker, iostat)
 ~ subroutine nextSegment(anIF, aSegment, iostat)
 ~ subroutine nextPart(anIF, aPart, iostat)
 ~ subroutine nextLink(anIF, aLink, iostat)
 ~ subroutine updateGeomLinkMatrices(aGeom)
 ~ subroutine resetMarker(aMarker)
 ~ subroutine resetSegment(aSegment)
 ~ subroutine addMarker(thePart, aMarker)
 ~ subroutine addSegment(thePart, aSeg)
 ~ subroutine addPart(theGeom, aPart)
 ~ subroutine addLink(theGeom, aLink)
 ~ subroutine deallocateGeomArrays(theGeom)
 ~ subroutine deallocatePartArrays(thePart)
 ~ subroutine deallocateSegmentArrays(theSeg)
```

InputFile_m

```
   subroutine openInputFile(lu, filename, anIF)
   subroutine closeInputFile(anIF)
   interface nextWord
   ~ subroutine nextWord_if(anIF, theWord)
   ~ subroutine nextWord_tl(theTL, theWord)
   interface getOneInteger
   ~ subroutine getOneInteger_i4b(anIF, theInt, aTag, traceString, iostat)
   ~ subroutine getOneInteger_i2b(anIF, theInt, aTag, traceString, iostat)
   subroutine getOneDouble(anIF, theDP, aTag, traceString, iostat)
   subroutine getOneLogical(anIF, theLog, aTag, traceString, iostat)
   subroutine getOneString(anIF, theStr, aTag, traceString, iostat)
   subroutine getOneVec3(anIF, theVec3, aTag, traceString, iostat)
   subroutine getVec3Buffer(anIF, buf, nVec, aTag, traceString, iostat)
```

```
   subroutine getDoubleBuffer(anIF, buf, nDbl,  aTag, traceString, iostat)
   interface hasNext
   ~ function hasNextWord_tl(theTL)
   ~ function hasNextWord_if(anIF)
   function atText(theTL, aText)
   function strip(aStr)
   function atOpenTag(anIF, aTag)
   function atCloseTag(anIF, aTag)
 ~ subroutine splitTextLine(theTL, theWord, set, sep)
 ~ subroutine nextInputRecord(anIF, iostat)
 ~ interface toValue
   ~ subroutine toValue_i4b(aWord, theInt, iostat)
   ~ subroutine toValue_i2b(aWord, theInt, iostat)
   ~ subroutine toValue_dp(aWord, theDP, iostat)
   ~ subroutine toValue_sp(aWord, theSP, iostat)
   ~ subroutine toValue_l(aWord, aLogical, iostat)
 ~ function setTextLine(chars) result(theTL)
 ~ function atCommentLine(anIF)
```

Mat4_m

```
   function identityMat4()
   function tMat4(shift)
   function rMat4(axis, angle)
   function trtMat4(shift1, axis, angle, shift2)
   function matchMat4(srcVec1, srcVec2, srcPoint, tarVec1, tarVec2, tarPoint)
   function mat4Transpose(aMat4)
   function isIdentity(aMat4, eps)
   function isOrthogonal(aMat4, eps)
   interface mat4Mul
   ~ function mat4Mul_m4m4(aMat4, bMat4)
   ~ function mat4Mul_m4v4(aMat4, aVec4)
   interface toArray
   ~ function toArray_m4(aMat4)
   function toMat4(anArray)
```

Spline_m

```
   subroutine linear(x, y, xVal, yVal)
   subroutine ratHermite(x, y, xVal, yVal)
   interface scaling
   ~ subroutine scaling_v(x, x1from, x2from, x1to, x2to)
   ~ subroutine scaling_dp(x, x1from, x2from, x1to, x2to)
   interface normalize
   ~ subroutine normalize_v(x, xmin, xmax)
   ~ subroutine normalize_dp(x, xmin, xmax)
   interface denormalize
   ~ subroutine denormalize_v(x, xmin, xmax)
   ~ subroutine denormalize_dp(x, xmin, xmax)
```

Trace_m

```
   subroutine traceInit(lvl, stdout, file, filename)
   interface trace
   ~ subroutine trace_i4b(lvl, txt, i, form)
   ~ subroutine trace_i2b(lvl, txt, i, form)
   ~ subroutine trace_r(lvl, txt, r, form)
   ~ subroutine trace_d(lvl, txt, d, form)
   ~ subroutine trace_nd(lvl, txt, d, n, form)
   ~ subroutine trace_l(lvl, txt, b, form)
   ~ recursive subroutine trace_c(lvl, txt)
 ~ subroutine openTracefile()
 ~ subroutine closeTracefile()
```

Utils_m

```
subroutine to_uppercase(string)
subroutine to_lowercase(string)
subroutine reverseOrder(x, n)
subroutine writeOpenTag(lu, nspaces, tag)
subroutine writeCloseTag(lu, nspaces, tag)
subroutine writeOneInteger(lu, nspaces, tag, theInt)
subroutine writeOneDouble(lu, nspaces, tag, theDouble)
subroutine writeOneVec3(lu, nspaces, tag, theVec3)
function deg2rad(d) result(r)
function rad2deg(r) result(d)
function orderType(x, n)
```

Vec34_m

```
interface assignment(=)
~ subroutine vec3_eq_vec3(a,b)
~ subroutine vec3_eq_d(a,b)
~ subroutine vec3_eq_v3(a,d)
~ subroutine v3_eq_vec3(d,a)
interface operator(+)
~ function add_vec3(a,b)
interface operator(-)
~ function sub_vec3(a,b)
~ function neg_vec3(a)
interface operator(.dot.)
~ function vec3_dot_vec3(a,b)
interface operator(*)
~ function vec3_mult_d(a,d)
~ function d_mult_vec3(d,a)
interface operator(/)
~ function vec3_divide_d(a,d)
interface operator(.cross.)
~ function vec3_cross_vec3(a,b)
function length(aVec3)
function unit(aVec3)
function angle(aVec1, aVec2)
interface toVec3
~ function toVec3_x123(x1, x2, x3)
~ function toVec3_v4(aVec4)
~ function toVec3_a3(anArray)
interface toVec4
~ function toVec4_v3(aVec3)
~ function toVec4_a4(anArray)
~ function toVec4_x1234(x1, x2, x3, x4)
interface toArray
~ function toArray_v3(aVec3)
~ function toArray_v4(aVec4)
```

VortexLattice_m

```
subroutine deallocateLiftingLineArrays(aLLine)
subroutine allocateLiftingLineArrays(aLLine, nj)
subroutine deallocateNearWakeArrays(aNWake)
subroutine allocateNearWakeArrays(aGeom, max_ni)
interface setLiftingLineGeom
~ subroutine setLiftingLineGeom_seg(aSeg, cpnt_mid, iostat)
~ subroutine setLiftingLineGeom_geom(aGeom, cpnt_mid, iostat)
subroutine setLiftingLineGamma(aGeom)
subroutine saveLiftingLineGamma(aGeom)
interface saveControlPoints
~ subroutine saveControlPoints_seg(aSeg)
~ subroutine saveControlPoints_geom(aGeom)
```

```
   subroutine setNearWakeStrip1Geom(aGeom)
   subroutine setNearWakeStrip1Gamma(aGeom)
   subroutine convectNearWake(aGeom, prevTime, curTime, iostat)
   subroutine uvwCPointWind(aGeom, prevTime, curTime)
   subroutine uvwCPointMotion(aGeom, prevTime, curTime, iostat)
   subroutine uvwCPointNearWakeStrips3N(aGeom, cutoff, cutoffType)
   subroutine uvwCPointNearWakeStrips12(aGeom, cutoff, cutoffType)
   subroutine uvwCPointLiftingLine(aGeom, cutoff, cutoffType)
   subroutine updateLiftingLineGamma(aGeom, relax, relax_end, converged)
   subroutine initUVWNearWake(aGeom)
   subroutine uvwNWLiftingLine(aGeom, cutoff, nFree, cutoffType)
   subroutine uvwNWNearWake(aGeom, cutoff, nFree, cutoffType)
   subroutine rollupNearWake(aGeom, prevTime, curTime, nFree, iostat)
   subroutine computeForces(aGeom)
   function uvwVortexLine(vtxStart, vtxEnd, gamma, cutoff, cutoffType, point) result(uvw)
 ~ function uvwVortexLineLinear(vtxStart, vtxEnd, gamma, cutoff, point) result(uvw)
 ~ function uvwVortexLine_nocheck(vtxStart, vtxEnd, gamma, point) result(uvw)
 ~ function uvwVortexLineSmooth(vtxStart, vtxEnd, gamma, cutoff, point) result(uvw)
```

WindScenario_m

```
   subroutine nextWindScenario(anIF, aWSc, iostat)
   subroutine makeWindSnapshot(aWSc, aTime, theWSn)
 ~ subroutine addWTimeInst(theWSc, aWTI)
```

WindSnapshot_m

```
   subroutine nextWTimeInst(anIF, aWTI, iostat)
   subroutine nextWindSnapshot(anIF, theWSn, iostat)
   subroutine writeWindSnapshot(theWSn, iostat)
 ~ subroutine addWTimeInst(theWSn, aWTI)
```

Wind_m

```
   subroutine setWindUVW(aWSn)
   function uvwWind(position, time) result(uvw)
```

C FORTRAN90 MAIN PROGRAM

In this section the (slightly edited) AWSM main program is listed.

```
 1:!******************************************************************************
 2:!* FILE : AWSM.f90
 3:!*
 4:!* PROGRAM UNIT : program AWSM
 5:!*
 6:!* PURPOSE : Program AWSM is a wind turbine aerodynamics simulation program.
 7:!*           It is based on generalized non-linear lifting line vortex wake
 8:!*           theory.
 9:!*
10:!* MODULES USED : Types_m, VXTypes_m, VXParam_m, Trace_m, Ask_m, InputFile_m,
11:!*                Geom_m, GeomScenario_m, VortexLattice_m, AeroLink_m, Wind_m,
12:!*                WindScenario_m, WindSnapshot_m, DataInput_m, DataOutput_m,
13:!*                Utils_m
14:!*
15:!* UPDATE HISTORY :
16:!* VSN  DATE         AUTHOR          SUMMARY OF CHANGES
17:!* 1    1-JAN-2002   A. van Garrel   Initial coding
18:!*
19:!******************************************************************************
20:!* SHORT DESCRIPTION :
21:!*   No description.
22:!*
23:!* REMARKS :
24:!*   No remarks.
25:!*
26:!* USAGE:
27:!*   No description.
28:!******************************************************************************
29:program AWSM
30:  ... import of modules
31:  implicit none
32:  ... variable declarations
33:  ... variable initialisations
34:
35:  ! Initialize total nr. of iterations for aerodynamic convergence
36:  nitConvTot = 0
37:
38:  ! Initialize flag for equilibrium between aerodynamic and structural forces
39:  equilibrium = .TRUE.
40:
41:  ! Initialize flag for convergence of angle of attack and liftcoefficient
42:  converged = .FALSE.
43:
44:  ! Collect user input
45:  call userInput()
46:
47:  ! Set logging facilities
48:  call traceInit(lvl=diTraceLvl, stdout=diTraceStdOut, file=diTraceFileOn, &
49:                                                        filename=diTraceFn)
50:
51:  ! Echo AWSM version data to trace output streams
52:  call traceAWSMVersion()
53:
54:  ! Echo user input data to trace output streams
55:  call traceUserInput()
56:
57:  ! Open files to write (ASCII) results to
58:  call openResultsFile(LU_GEOM_RESULTS, diGeomResultsFn)
59:  call openResultsFile(LU_AERO_RESULTS, diAeroResultsFn)
60:
61:  call trace(MESSAGE,"Start of AWSM simulation...")
62:
63:  ! Read and check geometry and activate geometric links
64:  call trace(MESSAGE," Handle geometry...")
65:  call openInputFile(LU_GEOM, diGeomFn, myIF)
66:  call nextGeom(myIF, theGeom, iosGeom)
67:  call closeInputFile(myIF)
68:  call checkGeomIdNrs(theGeom, iosIdNrs)
69:  call activateGeomLinks(theGeom, iosActLnk)
70:
71:  ! Stop if an error occurred in geometry preparation
72:  if ((iosGeom/=0).or.(iosIdNrs/=0)) then
73:    call trace(FATAL," Error while preparing geometry... ")
74:    stop
75:  endif
76:
77:  ! Allocate storage for all NearWake objects
78:  call allocateNearWakeArrays(theGeom, diMaxNiNearwake)
79:
80:  ! Read and check aerodynamic coefficients and link them to the "Segment" strips
81:  call trace(MESSAGE," Handle aerodynamic coefficients...")
82:  call openInputFile(LU_AERO_LINK, diAeroLinkFn, myIF)
83:  call nextAeroLink(myIF, theAL, iosAL)
84:  call closeInputFile(myIF)
85:  call checkAeroLink(theAL, iosCheckAL)
86:  call applyAeroLink(theGeom, theAL, iosApplyAL)
87:
88:  ! Read and check the prescribed sequence of movements of the "Marker"s
89:  call trace(MESSAGE," Handle geometry scenario...")
```

```
 90:  call openInputFile(LU_GEOM_SCENARIO, diGeomScenarioFn, myIF)
 91:  call nextGeomScenario(myIF, theGSc, iosGS)
 92:  call closeInputFile(myIF)
 93:  call checkGeomScenario(theGSc, theGeom, iosCheckGSc)
 94:
 95:  ! Create and write the location and orientation of the "Marker"s
 96:  curTime = diStartTime
 97:  call makeGeomSnapshot(theGSc, curTime, theGSn)
 98:  call writeGeomSnapShot(theGSn, iosWriteGSn)
 99:
100:  ! Set the geometry at its starting position
101:  call applyGeomSnapshot(theGSn, theGeom, iosApplyGSn)
102:
103:  ! Set lifting line geometry from actual geometry
104:  call trace(MESSAGE," Set LiftingLine geometry...")
105:  call setLiftingLineGeom(theGeom, diCPointsMid, iosSetLLine)
106:
107:  ! Read the prescribed sequence of wind velocities
108:  call trace(MESSAGE," Handle wind scenario...")
109:  call openInputFile(LU_WIND_SCENARIO, diWindScenarioFn, myIF)
110:  call nextWindScenario(myIF, theWSc, iosWS)
111:  call closeInputFile(myIF)
112:
113:  ! Stop if an error occurred in setup for time stepping
114:  if ((iosAL/=0).or.(iosCheckAL/=0).or.(iosApplyAL/=0).or. &
115:      (iosGS/=0).or.(iosCheckGSc/=0).or.(iosWriteGSn/=0).or. &
116:      (iosApplyGSn/=0).or.(iosSetLLine/=0).or.(iosWS/=0)) then
117:    call trace(FATAL," Error while preparing for time stepping... ")
118:    stop
119:  endif
120:
121:  call trace(MESSAGE," Start time stepping...")
122:
123:  ! Loop over all timesteps
124:  do iTime=1,diNTimeStep
125:
126:    ! Set for this timestep current time, previous time and mid time
127:    curTime = diStartTime + iTime*(diEndTime-diStartTime)/diNTimeStep
128:    prevTime = diStartTime + (iTime-1)*(diEndTime-diStartTime)/diNTimeStep
129:    midTime = 0.5_dp*(prevTime + curTime)
130:
131:    ! Assume that local angle of attack and lifting line strength don't match
132:    converged = .FALSE.
133:
134:    ! Determine and write the wind velocities at 'midtime'
135:    call makeWindSnapshot(theWSc, midTime, theWSn)
136:    call writeWindSnapShot(theWSn, iosWriteGSn)
137:
138:    ! Reading and recreating the "WindSnapshot" object previously written
139:    ! Not neccesary when there is a "WindSnapshot" object already available
140:    !   call openInputFile(LU_WIND_SNAPSHOT, diWindSnapshotFn, myIF)
141:    !   call nextWindSnapshot(myIF, theWSn, iosNextWSn)
142:    !   call closeInputFile(myIF)
143:
144:    ! Set wind velocity
145:    call setWindUVW(theWSn)
146:
147:    ! Save all actual control points to the old control points data array
148:    call saveControlPoints(theGeom)
149:
150:    ! Save all actual "Segment" trailing edge points as "old" values
151:    call saveTrailingEdge(theGeom)
152:
153:    ! Convect the old trailing edge with wind velocity
154:    call convectOldTrailingEdge(theGeom, prevTime, curTime, iosConvectOldTE)
155:
156:    ! Convect free near wake vortex lines with wind velocity
157:    call convectNearWake(theGeom, prevTime, curTime, iosConvectNWake)
158:
159:    ! Create and write the location and orientation of the "Marker"s
160:    call makeGeomSnapshot(theGSc, curTime, theGSn)
161:    call writeGeomSnapShot(theGSn, iosWriteGSn)
162:
163:    ! Iterate until equilibrium between aerodynamic forces and geometry deformations
164:    ! Not to be used in case of prescribed geometry motion and wind velocities
165:    do itEq=1,diNitEqMax
166:
167:      ! Reading and recreating the "GeomSnapshot" object previously written
168:      ! Not neccesary when there is a "GeomSnapshot" object already available
169:      !   call openInputFile(LU_GEOM_SNAPSHOT, diGeomSnapshotFn, myIF)
170:      !   call nextGeomSnapshot(myIF, theGSn, iosNextGSn)
171:      !   call closeInputFile(myIF)
172:
173:      ! Update the actual "Geom" object data given a "GeomSnapshot" object
174:      ! On exit the actual "Segment" coordinates are updated
175:      ! The "old" Segment t.e. coordinates are left unaltered.
176:      call applyGeomSnapshot(theGSn, theGeom, iosApplyGSn)
177:
178:      ! Set lifting line geometry from actual geometry
179:      call setLiftingLineGeom(theGeom, diCPointsMid, iosSetLLine)
180:
181:      ! Set actual vortex strengths equal to "old" vortex strengths
182:      call setLiftingLineGamma(theGeom)
183:
184:      ! Set the 1st "NearWake" strip geometry from actual and (convected) "old" t.e.
185:      call setNearWakeStrip1Geom(theGeom)
```

```
186:
187:      ! Set the 1st "NearWake" strip vortex strengths equal to actual vortex
188:      ! strengths of "LiftingLine" object
189:      call setNearWakeStrip1Gamma(theGeom)
190:
191:      ! Compute wind velocities at control points
192:      call uvwCPointWind(theGeom, prevTime, curTime)
193:
194:      ! Compute motion related velocities at control points
195:      call uvwCPointMotion(theGeom, prevTime, curTime, iosUvwMotion)
196:
197:      ! Compute "induced" velocities of near wake (minus 2 newest strips)
198:      ! on control points JUST ONCE
199:      call uvwCPointNearWakeStrips3N(theGeom, diCutoff, diCutoffType)
200:
201:      ! Iterate until matching local angle of attack and lift (== vortex strength)
202:      do itConv=1,diNitConvMax
203:
204:         ! Update total nr. of iterations for matching a.o.a and vortex strength
205:         nitConvTot = nitConvTot + 1
206:
207:        ! Compute "induced" velocities of "LiftingLine" on control points
208:        call uvwCPointLiftingLine(theGeom, diCutoff, diCutoffType)
209:
210:        ! Compute "induced" velocities of near wake's 2 newest strips on control points
211:        call uvwCPointNearWakeStrips12(theGeom, diCutoff, diCutoffType)
212:
213:        ! Set the 1st "NearWake" strip vortex strengths equal to actual vortex
214:        ! strengths of "LiftingLine" object
215:        call setNearWakeStrip1Gamma(theGeom)
216:
217:        ! Determine "LiftingLine" vortex strengths from local velocities and airfoil "Polar"s
218:        call updateLiftingLineGamma(theGeom, diRelaxGamma, diRelaxGamma_end, converged)
219:
220:        ! Exit when local angle of attack and lifting line strength match
221:        if (converged) exit
222:
223:        ! Decrease relaxation factor for 'gamma' change
224:        diRelaxGamma = diRelaxGamma*diRelaxGammaScaling
225:
226:      end do ! End iteration until matching local a.o.a. and lift
227:
228:      ! Compute forces and moments
229:      call computeForces(theGeom)
230:
231:      if (equilibrium) then
232:        ! Save "LiftingLine"s current vortex strengths into "old" data array
233:        call saveLiftingLineGamma(theGeom)
234:        exit
235:      endif
236:    end do ! End iteration for equilibrium between structural and aerodynamic forces
237:
238:    ! Initialize velocities at near wake points to zero
239:    call initUVWNearWake(theGeom)
240:
241:    ! Compute velocities "induced" by "LiftingLine" and "NearWake" at
242:    ! 'free' wake geometry and apply wake rollup
243:    call uvwNWLiftingLine(theGeom, diCutoffNW, diNFree, diCutoffType)
244:    call uvwNWNearWake(theGeom, diCutoffNW, diNFree, diCutoffType)
245:    call rollupNearWake(theGeom, prevTime, curTime, diNFree, iosRollupNWake)
246:
247:    call writeGeomResults(theGeom, iTime, curTime, diITimeStartGeom, &
248:                                                   diITimeEndGeom, diITimeStepGeom)
249:    call writeAeroResults(theGeom, iTime, curTime, diITimeStartAero, &
250:                                                   diITimeEndAero, diITimeStepAero)
251:
252:  end do ! Next timestep
253:
254:  if (iosConvectNWake==1) then
255:    call trace(DEBUG," ")
256:    call trace(DEBUG,"  Oldest NearWake strips were discarded")
257:    call trace(DEBUG," ")
258:  endif
259:
260:  call trace(MESSAGE," End time stepping")
261:  call trace(MESSAGE," ")
262:  call trace(MESSAGE," Total nr. of aerodynamic iterations : ", nitConvTot, form='(I8)')
263:  call trace(MESSAGE," Mean nr. of aerodynamic iterations  : ", nitConvTot/diNTimeStep), form='(I8)')
264:
265:  ! Close output files
266:  call closeResultsFile(LU_GEOM_RESULTS)
267:  call closeResultsFile(LU_AERO_RESULTS)
268:
269:  call trace(MESSAGE,"End of AWSM simulation")
270:
271:  ! Echo AWSM version data to trace output streams
272:  call traceAWSMVersion()
273:
274:end program AWSM
```

D SOFTWARE ELEMENTS

In this section a short description of the geometric elements used in the AWSM simulation code is given, followed by an explanation of the conceptual elements that are used as objects in the software and in the input files.

D.1 Geometric Elements

In the AWSM simulation module use is made of 5 specific geometry related elements which are shortly explained in order of increasing specialization with the help of figures 31 and 32.

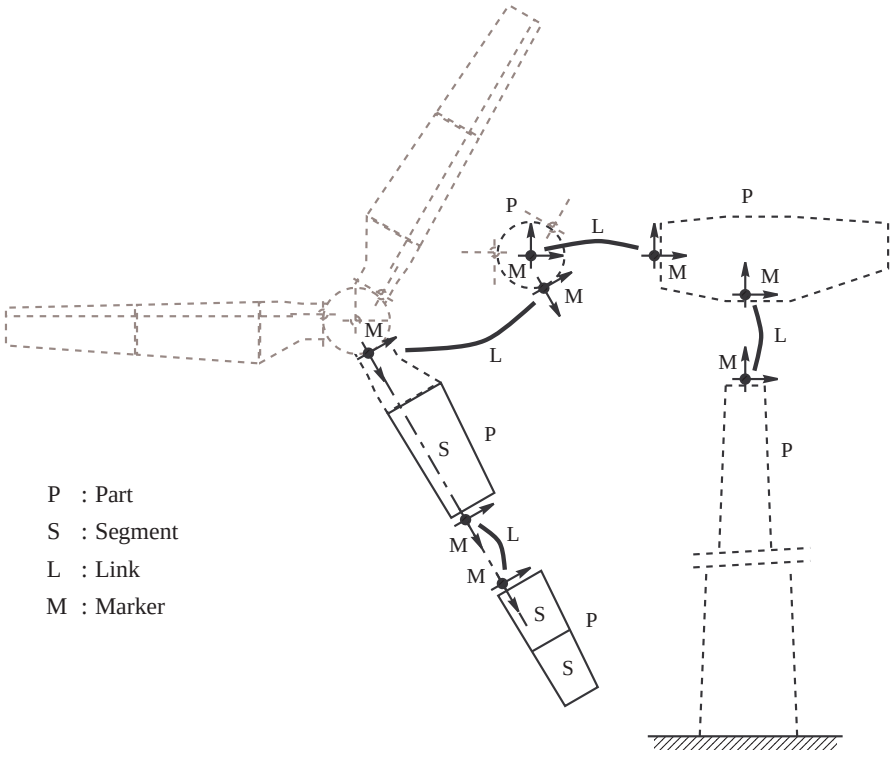

Figure 31: Geometric elements

Geom Geom acts as a container for Parts and Links and thus represents the collection of geometrical components and their mutual relations.

Part A Part is anything that can be considered a single solid body. For example a wind turbine tower with a non-yawing nacelle attached. The case of a tower and a free yawing nacelle should consists of at least two Parts. A Part is also some kind of container object that includes zero or more Segments and Markers.

Segment A Segment is the geometric representation used for the actual aerodynamic computations. It is the discrete representation of an aerodynamic force-carrying surface. A Segment is subdivided into strips that each have a vortex ring from the quarter chord line to the trailing edge (see figure 32). The Segment geometry coordinates are specified by the user as a rank-2 array of position vectors and from these coordinates the vortex-ring line elements are constructed.

Marker A Marker consist of two directional vectors and a position vector. As such a Marker serves as a local reference point and reference orientation. This facilitates the specification of geometric elements in their own local coordinate system. Parts are coupled by linking their Markers to each other. A transformation matrix is determined that translates and rotates the "child" marker so that it matches the position and orientation of the "parent" marker it is linked to. This transformation matrix is recursively applied to the geometry of all "children". Solid body motion of complete Parts is achieved by specifying the motion of Markers. After all transformations are recursively applied, the system is assumed to be defined in the global coordinate system.

Link A Link represents the coupling of two Markers belonging to two different Parts. One side of the Link acts as a parent, the other side as the child. Specifying motion of a parent Part's Marker is automatically transferred to the children. The specified motion of a child Part's Marker however is *not* transferred to its parent.

Because solid body motions, deformations and deformation rates of a wind turbine rotor typically vary in time and from blade to blade no mechanism is incorporated to specify and take advantage of geometric periodicity.

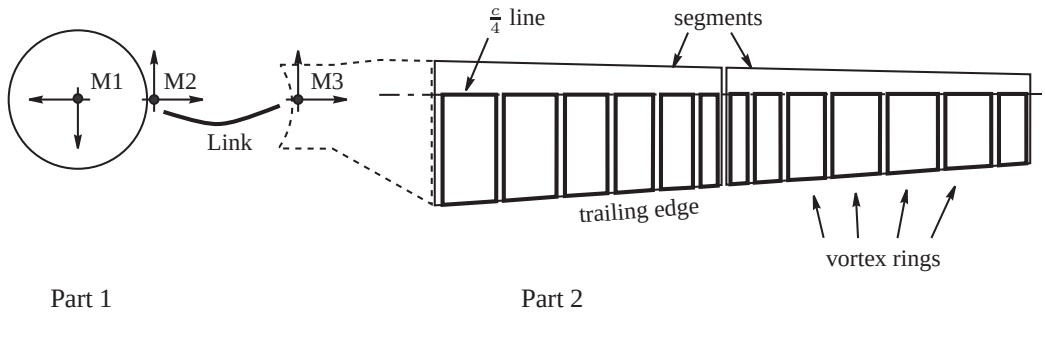

Figure 32: Rotor blade composition

D.2 Conceptual Elements

In this section a short introduction is given to the top level conceptual entities in the AWSM system that cannot be associated with real world objects and mostly are composed of sub-elements or collections of sub-elements. The sub-elements are handled in the description of the input files in section 3.2.

Polar Polar is a container for airfoil aerodynamic coefficients like lift, drag and pitching moment as function of angle of attack, all defined in separate aerodynamic tables.

Airfoil An Airfoil entity acts as a container for aerodynamic Polars for a specific airfoil. Each Airfoil can have a multitude of Polars.

AeroData AeroData is a storage element for Airfoils and acts as an aerodynamic database.

StripLink StripLink links a specific Airfoil's Polar to a specified strip in a Segment. Each StripLink therefore associates a specific blade cross-section with its own aerodynamic polar.

StripSet StripSet is a container of StripLinks for *all* strips in a Segment.

AeroLink AeroLink is a collection of StripSets for *all* Segments.

MarkerTimeInstance A MarkerTimeInstance contains Marker transformation data at a specific instance in time. It has no information of what Marker it is associated with.

MarkerTimeSeries MarkerTimeSeries is a temporal sequence of MarkerTimeInstances for a specific Marker - Part combination. It also contains the information for the type of inter- and extrapolation in time.

MarkerTransformation A MarkerTransformation is a MarkerTimeInstance at the current instance in time for a specific Marker - Part combination.

GeomScenario A GeomScenario contains the scenario for geometry movements. It is a container of MarkerTimeSeries each of which describes the rotation and translation of a Marker in a specific Marker - Part combination.

GeomSnapshot GeomSnapshot acts as a container of MarkerTransformations for one or more Markers at the same instance in time. As its name indicates, it is a snapshot taken at the current time from the GeomScenario data.

WindTimeInstance WindTimeInstance is the wind velocity data at a specific instance in time.

WindScenario WindScenario is a temporal sequence of WindTimeInstances and also contains the type of inter- and extrapolation used to make take a WindSnapshot at the current time instance.

WindSnapshot WindSnapshot is a snapshot taken at the current time from the WindScenario. It acts as a container for a WindTimeInstance at the current instance in time.

It should be noted that above elements are used in the input files and all have their software type counterpart. Some of the denominations however have been shortened for convenience. So has “MarkerTimeInstance” the Fortran90 type “`MTimeInst`” as counterpart. A listing of all the AWSM defined Fortran90 types can be found in module `VXTypes_m` on page 68. Their corresponding input file designations, if applicable, can be found in module `VXParam_m` on page 66.

E INPUT FILE DESCRIPTION

In this section a formal specification of the AWSM input files is given.

The required input files for an AWSM program run are listed in table 7 with a short description of their purpose. A description the objects referred to in the input files can be found in sections D.1 and D.2.

The AeroData file is a database containing a number of experimentally and/or numerically obtained 2D airfoil aerodynamics coefficients. It should be noted that, in the current AWSM version, the coefficients are supposed to be corrected for the stall-delaying influence of blade rotation.

The last two files in the table are intermediate files that currently are generated by the program itself. They are merely introduced to provide a point to interface with external programs in a future AWSM version.

The orientation and deformation of the geometry can then be supplied, through the GeomSnapshot file, by an external structural dynamics simulation program. This interface is described in appendix F at page 101. The wind velocity field could be supplied in the WindSnapshot file by an advanced wind field simulator that models non-homogeneous onset wind.

Table 7: AWSM input files

File		Purpose
Geom	:	geometry specification
AeroData	:	aerodynamic coefficients database
AeroLink	:	linking geometry to aerodynamic tables
GeomScenario	:	prescribing motion of geometry
WindScenario	:	prescribing wind velocities
GeomSnapshot	:	current geometry orientation
WindSnapshot	:	current wind velocities

AWSM will process all input files and will, in case of improper formatted data, try to log as much error conditions as it can to the active output streams. This circumvents the situation that for each single error in the input files a new program run has to be initiated. Note however that one error can cause a cascade of errors.

In the following paragraphs the actual layout of the AWSM input files will be described in more detail. The simplest form of each file will be shown together with a list of additional restrictions and possible options. The general restrictions and characteristics that apply for all AWSM input files are:

1. Unless otherwise stated, only one element of a specific type is allowed within the containing object.

2. Unless otherwise stated, all objects are mandatory components.

3. All `Id` object identification character strings are optional.

4. All `IdNr` object identification numbers for the same type of object are unique within their containing object.

In table 8 the shorthand notations used for the various object properties in the input file specifications is listed.

Table 8: Input variable shorthand notation

Symbol		Description
AFP	:	2-dimensional array of floating point numbers
AV3	:	2-dimensional array of vectors of 3 floating point numbers
CS	:	character string
FP	:	floating point number
I	:	integer number
L	:	logical variable
V3	:	vector of 3 floating point numbers

Geom The specification and composition of the various geometric parts in a Geom file for a specific simulation is described in this paragraph. The simplest form of a Geom file that includes the layout of `Parts`, `Segments`, `Markers` and `Links` looks like:

```
<Geom>
  <Id> CS </Id>

  <Part>
    <Id> CS </Id>
    <IdNr> I </IdNr>

    <Segment>
      <Id> I </Id>
      <IdNr> I </IdNr>
      <Ni> I </Ni>
      <Nj> I </Nj>
      <Coordinates>
        AV3
      </Coordinates>
    </Segment>

    <Marker>
      <Id> CS </Id>
      <IdNr> I </IdNr>
      <Location> V3 </Location>
      <Vector1> V3 </Vector1>
      <Vector2> V3 </Vector2>
    </Marker>

  </Part>

  <Part>
    <Id> CS </Id>
    <IdNr> I </IdNr>

    <Marker>
      <Id> CS </Id>
      <IdNr> I </IdNr>
      <Location> V3 </Location>
      <Vector1> V3 </Vector1>
      <Vector2> V3 </Vector2>
    </Marker>
  </Part>

  <Link>
    <Id> CS </Id>
```

```
    <IdNr> I </IdNr>
    <ParentPart> I </ParentPart>
    <ParentMarker> I </ParentMarker>
    <ChildPart> I </ChildPart>
    <ChildMarker> I </ChildMarker>
  </Link>

</Geom>
```

The Geom file contents is subject to the following number of restrictions and options:

1. There should be exactly one `Geom` object
2. A `Geom` object must contain at least one `Part` object.
3. A `Geom` object can contain zero or more `Link` objects.
4. A `Link` object must reference two `Marker` objects belonging to two different `Part` objects.
5. At least one `Part` object should contain one or more `Segment` objects.
6. The number of three-component vectors in the `Coordinates` property of a `Segment` object must exactly match the number of coordinates in the i-direction times the number in j-direction specified by the `Ni` and `Nj` objects respectively. Ordering is assumed to be such that the i-index runs fastest. This index is associated with the chordwise direction (see figure 11).
7. The number of coordinates in i- and j-direction, specified by `Ni` and `Nj`, must both be at least two.

AeroData The AeroData file layout is described in this paragraph. It acts as an aerodynamic database containing 2D aerodynamic lift-, drag- and pitching moment coefficients. Multiple AeroData files can be referenced in the AeroLink file. The simplest form of an AeroData file with only one aerodynamic polar for one airfoil looks like:

```
<AeroData>
  <Id> CS </Id>

  <Airfoil>
    <Id> CS </Id>
    <IdNr> I </IdNr>

    <Polar>
      <Id> CS </Id>
      <IdNr> I </IdNr>
      <Re> FP </Re>
      <Ma> FP </Ma>
      <Alpha-Cl>
         AFP
      </Alpha-Cl>
      <Alpha-Cd>
         AFP
      </Alpha-Cd>
      <Alpha-Cm>
         AFP
      </Alpha-Cm>
    </Polar>

  </Airfoil>
```

```
</AeroData>
```

The AeroData file contents is subject to the following number of restrictions and options:

1. There should be exactly one `AeroData` object
2. An `AeroData` object must contain at least one `Airfoil` object.
3. An `Airfoil` object must contain at least one `Polar` object.
4. The `Alpha-Cl`, `Alpha-Cd` and `Alpha-Cm` objects are lift-, drag- and pitching moment coefficients as a function of angle of attack. The tables must consist of two or more floating point pairs. Of each pair the first value is assumed to be the angle of attack and the second the specific aerodynamic coefficient.
5. The angle of attack is assumed to be specified in degrees and must be ordered monotonously increasing or decreasing. The allowed range of angle of attack values is from -180 degrees to $+180$ degrees.
6. It is assumed that the angle of attack range covers all flow-directions that occur in the simulation. If not, aerodynamic data will be spline extrapolated without any notification.
7. The `Re` and `Ma` objects are the Reynolds number and Mach number for the containing `Polar`.

AeroLink The AeroLink file contains all information necessary to associate aerodynamic polars in AeroData files to all strips of the `Segments` specified in the Geom file. The simplest form of an AeroLink file for a `Segment` with just one strip looks like:

```
<AeroLink>
  <Id> CS </Id>

  <StripSet>
    <PartIdNr> I </PartIdNr>
    <SegmentIdNr> I </SegmentIdNr>

    <StripLink>
      <StripNr> I </StripNr>
      <AeroDataFile> CS </AeroDataFile>
      <AirfoilIdNr> I </AirfoilIdNr>
      <PolarIdNr> 1 </PolarIdNr>
    </StripLink>

  </StripSet>

</AeroLink>
```

The AeroLink file contents is subject to the following number of restrictions and options:

1. There should be exactly one `AeroLink` object
2. The `StripSet` object's identity is defined by its `PartIdNr` and `SegmentIdNr` identification numbers. Each `StripSet` must be unique.
3. For all `Segments` in the Geom file a `StripSet` must be specified.

4. The PartIdNr and SegmentIdNr identification numbers in the StripLink must reference existing Parts and Segments in the Geom file.
5. Each StripSet object must contain StripLinks for all Nj strips in the specific Segment it points to.
6. A StripLink's identity is defined by its StripNr and must be unique within the containing StripSet object.
7. Each StripLink object must reference an existing AeroData file in its AeroDataFile object.
8. The AirfoilIdNr and PolarIdNr objects in a StripLink must point to existing Airfoils and Polars in the specified AeroData file.

GeomScenario The GeomScenario file contains all information for the prescribed motion of the geometry. It basically consists of rotation and/or translation data for specific Markers as a (discrete) function of time. The simplest form of an GeomScenario file for a single Marker-Part combination looks like:

```
<GeomScenario>
  <Id> CS </Id>

  <MarkerTimeSeries>
    <PartIdNr> I </PartIdNr>
    <MarkerIdNr> I </MarkerIdNr>
    <InterpolationType> CS </InterpolationType>
    <Extrapolation> L </Extrapolation>

    <MarkerTimeInstance>
      <Time> FP </Time>
      <Translation> V3 </Translation>
      <RotationAxis> V3 </RotationAxis>
      <RotationAngle> FP </RotationAngle>
    </MarkerTimeInstance>

    <MarkerTimeInstance>
      <Time> FP </Time>
      <Translation> V3 </Translation>
      <RotationAxis> V3 </RotationAxis>
      <RotationAngle> FP </RotationAngle>
    </MarkerTimeInstance>

  </MarkerTimeSeries>

</GeomScenario>
```

The GeomScenario file contents is subject to the following number of restrictions and options:

1. There should be exactly one GeomScenario object
2. The MarkerTimeSeries object's identity is defined by its PartIdNr and MarkerIdNr identification numbers. Each MarkerTimeSeries must be unique.
3. The PartIdNr and MarkerIdNr identification numbers in the MarkerTimeSeries must reference an existing Part-Marker combination in the Geom file.
4. The InterpolationType property can have its value set to LINEAR or SPLINE. These character strings are interpreted case insensitive.

5. The `Extrapolation` flag specifies whether or not to extrapolate the `MarkerTimeSeries` beyond its start and end `Time` values. `Extrapolation` should be set to `YES` in case the simulation exceeds these time boundaries. It should be noted that extrapolation in case `InterpolationType` is set to `SPLINE` can give unexpected results.

6. All `MarkerTimeInstance` components will be interpolated and, if specified, extrapolated.

7. The `MarkerTimeSeries` object must contain at least two `MarkerTimeInstances` that are ordered in time by increasing `Time` properties.

8. The `Translation` and `RotationAxis` objects in a `MarkerTimeInstance` are specified in the `Part-Marker` combination's local coordinate system.

9. A `RotationAxis` object must be a three-component vector of non-zero length.

10. The `RotationAngle` property is assumed to be specified in degrees and acts in a right-handed sense in the direction of the specified rotation axis.

11. Specifying rotation and/or translation of a `Marker` in a `MarkerTimeSeries` object will only transform that specific `Marker` in that `Part`. `Segments` and other `Markers` will be unaffected.

12. It is trough `Links` with other `Parts` that a `Marker's` transformation effects are propagated to other objects in the geometry.

13. A geometry fixed in space can be simulated by specification of a zero translation vector and a zero rotation angle for each instance in time.

WindScenario The WindScenario file contains wind velocity vectors as a (discrete) function of time and is assumed to be valid in the whole domain. The simplest form of a WindScenario file looks like:

```
<WindScenario>
  <Id> CS </Id>
  <InterpolationType> CS </InterpolationType>
  <Extrapolation> L </Extrapolation>

  <WindTimeInstance>
    <Time> FP </Time>
    <WindUVW> V3 </WindUVW>
  </WindTimeInstance>

  <WindTimeInstance>
    <Time> FP </Time>
    <WindUVW> V3 </WindUVW>
  </WindTimeInstance>

</WindScenario>
```

The WindScenario file contents is subject to the following number of restrictions and options:

1. There should be exactly one `WindScenario` object

2. The `InterpolationType` property can have its value set to `LINEAR` or `SPLINE`. These character strings are interpreted case insensitive.

3. The `WindScenario` object must contain at least two `WindTimeInstances` that are ordered with increasing `Time` properties.
4. The `WindUVW` object in a `WindTimeInstance` is the wind velocity vector specified in the global coordinate system.
5. The `Extrapolation` flag specifies whether or not to extrapolate the `WindScenario` beyond its `WindTimeInstance` start and end `Time` values. `Extrapolation` should be set to `YES` in case the simulation exceeds these time boundaries. It should be noted that extrapolation in case `InterpolationType` is set to `SPLINE` can give unexpected results.

GeomSnapshot The GeomSnapshot file is an intermediate file generated each timestep by the AWSM program itself. It is a snapshot at the current time of a `GeomScenario` object. It contains the current rotation and translation data for all moving `Markers`. As was already mention at the start of this section, the GeomSnapshot file was introduced to provide a point to interface with other programs in a future AWSM version. The GeomSnapshot file can then be supplied by an external multibody system dynamics simulation program.

The simplest GeomSnapshot file for a single moving `Marker` of a `Part-Marker` combination looks like:

```
<GeomSnapshot>
  <MarkerTransformation>
    <PartIdNr> I </PartIdNr>
    <MarkerIdNr> I </MarkerIdNr>
    <MarkerTimeInstance>
      <Time> FP </Time>
      <Translation> V3 </Translation>
      <RotationAxis> V3 </RotationAxis>
      <RotationAngle> FP </RotationAngle>
    </MarkerTimeInstance>
  </MarkerTransformation>
</GeomSnapshot>
```

The GeomSnapshot file contents is subject to the following number of restrictions and options:

1. There should be exactly one `GeomSnapshot` object
2. The `MarkerTransformation` object's identity is defined by its `PartIdNr` and `MarkerIdNr` identification numbers. Each `MarkerTransformation` must be unique.
3. There exist a `MarkerTransformation` for each `MarkerTimeSeries` in the GeomScenario file.
4. The `PartIdNr` and `MarkerIdNr` identification numbers in the `MarkerTransformation` must reference an existing `Part-Marker` combination in the Geom file.
5. Each `MarkerTransformation` object contains exactly one `MarkerTimeInstance`.
6. The `Translation` and `RotationAxis` objects in the `MarkerTimeInstance` are specified in the `Part-Marker` combination's local coordinate system.
7. The `RotationAxis` object is a three-component vector of non-zero length.
8. The `RotationAngle` property is specified in degrees and acts in a right-handed sense in the direction of the specified rotation axis.

9. The `RotationAngle` property written by the AWSM program itself is translated to the interval between 0 and 360 degrees.

WindSnapshot The WindSnapshot file is, like the GeomSnapshot file, an intermediate file generated each timestep by the program itself. It is a snapshot at the current time of a `WindScenario` object. It contains the current time and wind velocity vector defined in the global coordinate system. For now the WindSnapshot file contains only one wind velocity vector that is valid for the complete domain.

The WindSnapshot file was introduced to be able to have the wind field in a future AWSM version provided by an advanced wind field simulator that can model non-homogeneous onset wind. The WindSnapshot file contents looks like:

```
<WindSnapshot>
  <WindTimeInstance>
    <Time> FP </Time>
    <WindUVW> V3 </WindUVW>
  </WindTimeInstance>
</WindSnapshot>
```

The WindSnapshot file contents is subject to the following number of restrictions and options:

1. There should be exactly one `WindSnapshot` object
2. The `WindTimeInstance` object contains the current time in the `Time` object and the current wind velocity in the three-component vector `WindUVW` object.

F INTERFACE

In this section the provisions made to interface the AWSM wind turbine aerodynamics simulation program with an external structural dynamics simulator is described.

As was already explained in appendix E, solid body motion of geometric components is implemented in AWSM through the use of GeomSnapshot objects that in effect apply transformation matrices (see section 2.2.2) to the geometry. The GeomSnapshot objects are currently created by the AWSM program itself. It is however also possible that an external structural dynamics simulation program specifies the GeomSnapshot object each simulation timestep. An extra iteration loop is already incorporated in the AWSM main progam (appendix C, page 85) in which aerodynamic and structural dynamics forces should converge to the same value. This extra iteration loop is executed each timestep as can be seen in the following simplified version of the AWSM main program algorithm:

```
 1:  program AWSM
 2:    Initialize variables
 3:    Collect user input
 4:    Read, check and activate geometric and aerodynamic input data
 5:    Set the geometry at its starting position
 6:
 7:    ! Loop over all timesteps
 8:    do iTime=1,diNTimeStep
 9:      Get wind field from "WindSnapshot" data
10:      Convect wake vortex lines with wind velocity
11:
12:      ! Iterate until equilibrium between structural and aerodynamic
13:      ! forces. Only 1 iteration in case of prescribed geometry motion
14:      ! and wind velocities
15:      do itEq=1,diNitEqMax
16:        Get the "GeomSnapshot" deformation specification
17:        Update the geometry according this "GeomSnapshot" object
18:        Set the "LiftingLine" vortex strengths at "old" values
19:        Compute wind velocities at control points
20:        Compute motion related velocities at control points
21:        Compute wake related "induced" velocities
22:
23:        ! Iterate until matching local angle-of-attack and lift
24:        do itConv=1,diNitConvMax
25:          Compute lifting line "induced" velocities at control points
26:          Assemble velocity contributions at the control points
27:          Update "LiftingLine" vortex strengths from airfoil "Polar"s
28:          Exit when local angle-of-attack and lifting line strength match
29:        end do
30:
31:        Compute "LiftingLine" aerodynamic forces and moments
32:
33:        if (equilibrium between structural and aerodynamic forces) then
34:          Save "LiftingLine"s current vortex strengths
35:          Exit loop for structural equilibrium
36:        endif
37:      end do
38:
39:      Compute "induced" velocities and apply wake rollup
40:      Write current geometric data
41:      Write current aerodynamic data
42:    end do
43:  end program AWSM
```

It should be noted that currently only solid body motions can be specified in the “GeomScenario” and “GeomSnapshot” input files. To include the effects of geometry deformations and deformation rates these two input files can be extended. A possible GeomScenario data file layout for this extension is sketched here:

```
<GeomScenario>
! Future Geomscenario extension with geometry deformation
 <Id> "Deformation of segment 11" </Id>

 ! Segment deformations are implemented in a future version !
 <SegmentTimeSeries>
  <PartIdNr> 1 </PartIdNr>
  <SegmentIdNr> 11 </SegmentIdNr>
  <InterpolationType> "LINEAR" </InterpolationType>
  <Extrapolation> NO </Extrapolation>
   <SegmentTimeInstance>
    <Time> 0.0 </Time>
    <Deformation>
      0.00 0.005 0.0     0.01  0.01  0.0    0.0    0.0    0.0
      0.00 0.002 0.0     0.0   0.0   0.0    0.02   0.0001 0.0
    </Deformation>
   </SegmentTimeInstance>

   <SegmentTimeInstance>
    <Time> 100.0 </Time>
    <Deformation>
      0.01 0.005 0.0    -0.001 0.001 0.0    0.0    0.0    0.0
      0.02 0.002 0.0     0.0   0.0   0.0    0.0012 0.0    0.0
    </Deformation>
   </SegmentTimeInstance>
 </SegmentTimeSeries>

</GeomScenario>
```

In this file the geometry deformations are specified at discrete instances in time in the Segment’s own coordinate system. In the same way the GeomSnapshot file should be extended. Because of the specific way in which geometry motion related onset velocities are computed (equation 15) the effects of geometry deformation rates are automatically incorporated .

G LOG DATA

In all AWSM Fortran90 subroutines calls to the log facility from the Trace_m module are present. Dependent on the type of condition encountered and the log-level specified by the user a message is written to the active output streams. Possible output streams are the standard output stream (i.e. command line window) and a logfile.

In table 9 all possible log-levels, the message type abbreviation written as start of each line in the logfile and a short description of the message type are given.

Table 9: Log levels

Level	Linestart	Description
5	[MSG]	Informational message; always written to the enabled output streams.
4	[FTL]	Something bad has happened, the system is of little use to anyone.
3	[ERR]	A specific task may have failed, but the system will keep running.
2	[WRN]	Something went wrong but the system will try to recover from it.
1	[INF]	Informational messages about what's happening.
0	[DBG]	Hints used during the initial coding and debugging process.

AWSM will try to process as much input files as it can and log all occurring errors to the enabled output streams. In case the input files are formatted correctly, the log information that is produced during a successful program run will look like this:

```
[MSG] ****************************************
[MSG]      AWSM version nr   : 1.0.0
[MSG]      AWSM version date : 1-AUG-2003
[MSG] ****************************************
[MSG] ****************************************
[MSG]  Geom filename         : geom01.dat
[MSG]  AeroLink filename     : aerolink01.dat
[MSG]  GeomScenario filename : geomscenario01.dat
[MSG]  WindScenario filename : windscenario01.dat
[MSG]
[MSG]  Log filename          : log01.dat
[MSG]  GeomSnapshot filename : geomsnapshot01.dat
[MSG]  WindSnapshot filename : windsnapshot01.dat
[MSG]  GeomResults filename  : geomresults01.dat
[MSG]  AeroResults filename  : aeroresults01.dat
[MSG]
[MSG]  Trace level      : 2
[MSG]  Trace to stdout  : T
[MSG]  Trace to logfile : T
[MSG]
[MSG]  Simulation start time      :      0.0000000
[MSG]  Simulation end time        :     10.0000000
[MSG]  Simulation nr of timesteps :   1000
[MSG]
[MSG]  Writing aero results start time(step) :    100
[MSG]  Writing aero results end time(step)   :   1000
[MSG]  Writing aero results timestep         :    100
[MSG]
[MSG]  Writing geom results start time(step) :   1000
[MSG]  Writing geom results end time(step)   :   1000
[MSG]  Writing geom results timestep         :      1
[MSG]
[MSG]  Control points in strip centers : F
[MSG]
[MSG]  Max nr. of streamwise wakepoints                 :   5001
[MSG]  Nr. of free streamwise wakepoints                :      0
```

```
[MSG]  Linear (1) or smooth (2) vortex cutoff velocities :  2
[MSG]  Wake rollup vortex cutoff radius                  :      0.0010000
[MSG]
[MSG]  Lift force vortex cutoff radius               :      0.0010000
[MSG]  Liftingline circulation convergence criterion :      0.0010000
[MSG]
[MSG]  Relaxation of circulation increment           :      0.8000000
[MSG]  Extra relaxation factor for segment outer 10% :      0.4000000
[MSG]  Relaxation factor scaling                     :      0.9500000
[MSG]  Maximum change in liftcoefficient             :      0.0500000
[MSG]  Minimum relaxation factor                     :      0.2000000
[MSG]  Maximum relaxation factor                     :      1.0000000
[MSG]
[MSG]  Max nr. of iterations for structural equilibrium  :  10
[MSG]  Max nr. of iterations for aerodynamic convergence : 400
[MSG]
[MSG] ****************************************
[MSG] Start of AWSM simulation...
[MSG]  Handle geometry...
[MSG]  Handle aerodynamic coefficients...
[MSG]  Handle geometry scenario...
[MSG]  Set LiftingLine geometry...
[MSG]  Handle wind scenario...
[MSG]  Start time stepping...
[MSG]  End time stepping
[MSG]
[MSG]  Total nr. of aerodynamic iterations :     1226
[MSG]  Mean nr. of aerodynamic iterations  :        1
[MSG] End of AWSM simulation
[MSG] ****************************************
[MSG]      AWSM version nr   : 1.0.0
[MSG]      AWSM version date : 1-AUG-2003
[MSG] ****************************************
```

Notice that all the lines in the log are of “message” type and start with `[MSG]`. At the top of the log data the current AWSM version number and the date associated with this version number are written. Next is a listing of the data input by the user at the command line. The flow of the actual AWSM simulation is logged in the next block of messages. After a successful simulation the AWSM version number and date are printed once again.
What happens if some kind of error is made in the specification of an input file can be seen in the next listing. After the user input log data, the logfile looks like:

```
[MSG] ****************************************
[MSG] Start of AWSM simulation...
[MSG]  Handle geometry...
[FTL] Geom_m, nextSegment() :
[FTL]   Number of points given < Ni*Nj
[FTL]     File name : geom.dat
[FTL]     Line nr.  :     14
[FTL] Geom_m, nextGeom() :
[FTL]   Nr. of Link's should be less than nr. of Part's
[FTL]     File name : geom.dat
[FTL]     Line nr.  :      5
[FTL] Geom_m, activateGeomLinks() :
[FTL]   Link IdNr :                1
[FTL]   Can't find ChildPart and/or ChildMarker
[FTL]  Error while preparing geometry...
```

As can be seen the log data now contains “fatal” messages (`[FTL]`). In this particular case the number of spanwise coordinates of a Segment was set at a wrong value. The log data shows the Fortran90 module (`Geom_m`) and the specific subroutine (`nextSegment()`) in which the error was encountered. The line (`14`) of the offending object and the file it occurred in (`geom.dat`) are logged also. All next “fatal” messages occur as a consequence of above

mentioned error. Trace level one ([INF] in table 9, page 103) is useful in case one wants to monitor the convergence history of a particular parameter setting. The optimum value of some input parameters for the fastest possible convergence can be found this way by trial and error. The actual parameter for convergence checking (see equation (20) is written to the enabled log streams each iteration. An example output fragment is shown next:

```
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.3141950
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.2139405
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.1238954
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.0468264
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.0250050
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.0138502
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.0106953
[INF]  VortexLattice_m, updateLiftingLineGamma(), dgam_max =      0.0099531
```

	Date: August 2003	**Number of report:** ECN-C--03-079	
Title	DEVELOPMENT OF A WIND TURBINE AERODYNAMICS SIMULATION MODULE		
Subtitle			
Author(s)	A. van Garrel		
Principal(s)	NOVEM BV.		
ECN project number	7.4318		
Principal(s)project number	224.312-0001		
Programme(s)	TWIN-2		

Abstract

The simulation of wind turbine aerodynamics can be improved with more physically realistic models than the ones currently in use in engineering practice. In this report the mathematical and numerical aspects and the practical use of a new wind turbine aerodynamics simulation module is described. The new simulation code is based on non-linear lifting line vortex wake theory which is a more accurate model for rotor wake physics; an important aspect in wind turbine load simulations. The new model is verified for some test cases with analytical solutions. Wake dynamics are shown to behave as anticipated. The new simulation module, designated AWSM, is expected to substantially reduce the number of uncertainties that accompany currently used blade-element-momentum methods. To assert the quality of AWSM results a comparison with experimental data is recommended. Numerical tests should be performed to investigate the influence of AWSM simulation parameters on the computed results. Further functionality extensions and algorithm optimizations are suggested.

Keywords	wind turbines, aerodynamics, lifting line

Authorization	**Name**	**Signature**	**Date**
Checked	**D. Winkelaar**		
Approved	**H. Snel**		
Authorized	**H.J.M. Beurskens**		