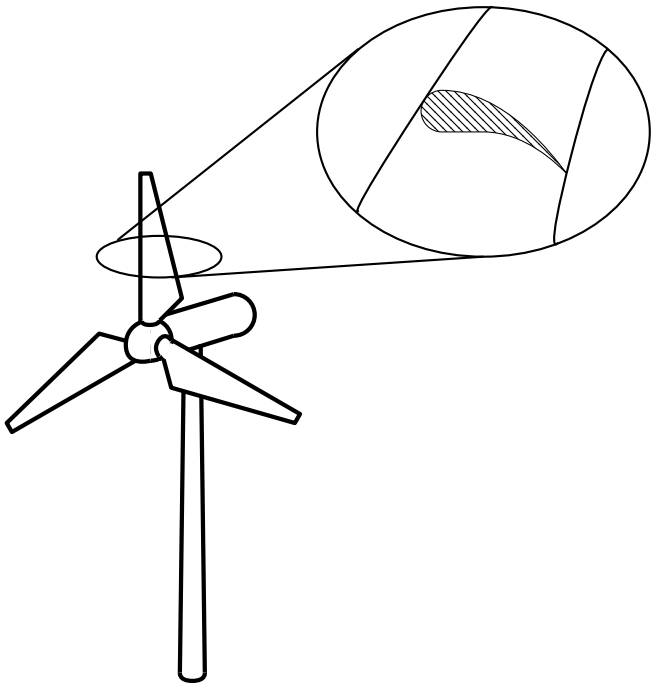
CHALMERS



Actuator turbine models and trailing edge flow: implementation in an in-house code

Master's Thesis in Applied Mechanics

JOHANNA MATSFELT

Department of Applied Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2015
Master's Thesis 2015:76

Actuator turbine models and trailing edge flow: implementation in an in-house code

Master's Thesis 2015:76

Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
SE-41296 Göteborg
Sweden
Tel. +46-(0)31 772 1000

Reproservice / Department of Applied Mechanics
Göteborg, Sweden 2015

Abstract

To be able to simulate a trailing edge flow in CALC some modifications needs to be done in the code. The main reason for this is that CALC only can handle a computational domain consisting of one block. Modifications are made both in the multigrid solver that solves the pressure field and the flow solver. Some of the modifications that are made in the multigrid solver can be recognized from the implementations in the flow solver but the multigrid solver uses 1D arrays compared to 3D arrays in the flow solver.

This implementation allows CALC to run a flat plate simulation and this was one of the validating cases run. One laminar flow and one turbulent flow using the Reynolds Averaged Navier Stokes (RANS) turbulent model and the results showed good agreement. One more validating RANS case was used here representing an airfoil instead of a flat plate. This simulation showed that the flow fulfilled the no slip condition at the surface of the airfoil which was the focus of the validation.

The second part of this master thesis consisted of implementing two Actuator turbine models named Actuator disk model (ADM) and Actuator line model (ALM). The implementation of ADM which is the less accurate model of the two was validated by an axisymmetric flow using the 5-MW National Renewable Energy Laboratory (NREL) wind turbine. The ALM model was validated using the same turbine but with a 3D flow due to its 3D behaviour. The results were acceptable when comparing to the NREL data and results in [1]. The results from the ADM simulation obtained by restricting the ε_i variable in the Gaussian function showed that more consistent predictions of the rotor thrust and power between different meshes could be obtained.

Acknowledgements

This master thesis started in the beginning of September 2014 and ended in the middle of August 2015. It is a 60 HE project carried out at the fluid dynamics division at Chalmers University of Technology in Gothenburg.

I would like to thank my supervisor and examiner Lars Davidson for believing in me and letting me carry out this one year master thesis. I would also like to thank the staff at the fluid dynamics division for their support.

Johanna Matsfelt, Gothenburg, August 2015

Contents

Nomenclature

Symbols

Trailing edge flow

a_N	Northern coefficient of each cell
a_S	Southern coefficient of each cell
$arean$	Northern face area of each cell
$areanjafp1$	Southern face area of the cells on top of the airfoil
$anjafp1$	The x part of the southern face area of the cells on top of the airfoil
$aljafp1$	The y part of the southern face area of the cells on top of the airfoil
$ahjafp1$	The z part of the southern face area of the cells on top of the airfoil
$countMax$	Maximum number of guesses preformed for the x coordinate on the airfoil
δ	Changes between different multigrid levels or boundary layer thickness
δ_{99}	Boundary layer thickness defined at 99% of u_∞
ε	Dissipation
f_{yP}	Weight function through the northern face of the specified cell
f_{yS}	Weight function through the northern face of the cell south of the specified cell
H	Thickness of the airfoil along the y direction
iaf	Last cell along the x axis that is alongside the airfoil
$iafp$	iaf but traced through the multigrid levels
jaf	Cell below next to the airfoil along the y axis
$jafp$	jaf but traced through the multigrid levels
$jb2jtt$	Nodes part of interpolation for the synthetic inlet fluctuations
k	Turbulent kinetic energy
$movedivde$	Distance related variable to move the x coordinate guessed on the airfoil
ν	Kinematic viscosity
ϕ	Generic variable
ϕ_N	ϕ at the center of the cell north of the specified cell
ϕ_n	ϕ at the northern face of the specified cell
ϕ_P	ϕ at the center of the specified cell
ϕ_S	ϕ at the center of the cell south of the specified cell
ϕ_s	ϕ at the southern face of the specified cell
Re_H	Reynolds number based on the thickness of the flat sturt
Re_x	Reynolds number at position x

S_P	Source term
S_U	Source term
Θ	Momentum thickness
u	Velocity component along the x direction, $[m/s]$
u_∞	Velocity infinitely away from the airfoil, $[m/s]$
U_0	Local mean free-stream velocity component along the x direction, $[m/s]$
v	Velocity component along the y direction, $[m/s]$
w	Velocity component along the z direction, $[m/s]$

Actuator turbine models

$AngBlade$	Angle between the blades
B	Number of blades on the wind turbine
$Bladek$	Instantaneous k plane location of each of the blades
c	Local chord length of the section, $[m]$
C_D	Local drag coefficient
C_L	Local lift coefficient
dt	Time step, $[s]$
D_{ADM}	Drag force in ADM, $[N/m^2]$
D_{ALM}	Drag force in ALM, $[N/m]$
Δr	Local radial mesh size, $[m]$
$\Delta\theta$	Local tangential mesh size, $[m]$
Δx	Local axial mesh size, $[m]$
e_D	Unit vector in the direction of the drag
e_L	Unit vector in the direction of the lift
ε^{1D}	Variable controlling the width of the Gaussian distribution in ADM
ε^{3D}	Variable controlling the width of the Gaussian distribution in ALM
ε_i	Variable controlling the width of the Gaussian distribution on cell level
$\eta(p^{1D})_\varepsilon^{1D}$	1D Gaussian distribution
$\eta(p^{2D})_\varepsilon^{3D}$	2D Gaussian distribution
f_{ADM}	Forces in ADM, $[N/m^2]$
$imiddiff$	Number of cells streamwise before and after the wind turbine
L_{ADM}	Lift force in ADM, $[N/m^2]$
L_{ALM}	Lift force in ALM, $[N/m]$

$omega$	The Rotational speed to the blades, $[rad/s]$
p^{1D}	Axial distance from evaluated point to the wind turbine, $[m]$
p^{2D}	Distance from evaluated point to the node on the line with the same radii, $[m]$
r	Local radii of the section, $[m]$
R	Rotor radii, $[m]$
θ	Polar angle, $[rad]$
$Theata2k$	Maps angles turned by the blades to k planes
$tottime$	The total time of the simulation at this time step, $[s]$
u_{tang}	Tangential velocity used in ADM and ALM, $[m/s]$
u	Cartesian x-axis velocity component in CALC, $[m/s]$
v	Cartesian y-axis velocity component in CALC, $[m/s]$
V_{rel}	Local relative velocity, $[m/s]$
w	Cartesian z-axis velocity component in CALC, $[m/s]$

Glossary

ADM	Actuator Disk Model
ALM	Actuator Line Model
BEM	Blade Element Momentum Method
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
LES	Large Eddy Simulation
$NREL$	National Renewable Energy Laboratory
$PANS$	Partially Averaged Navier Stokes
$RANS$	Reynolds Averaged Navier Stokes
$TDMA$	Tri-Diagonal Matrix Algorithm
$URANS$	Unsteady Reynolds Averaged Navier Stokes

1

Trailing edge flow: implementation in CALC

1.1 Geometry

The CALC code can only handle a computational domain consisting of one block. Here the trailing edge of an airfoil will be part of the domain as seen in fig. 1.1. With this setup the choice would have been to use multiple blocks. When a one block mesh is used the location of the airfoil is needed to be traced. The variables *iaf* and *jaf* are used for this. *iaf* is the last cell along the x axis that is alongside the airfoil and *jaf* is the cell below the airfoil along the y axis.

The variables are inserted in their locations with respect to the airfoil in fig. 1.2. The one block configuration will by default treat the *jaf* cells as within the airfoil if the x cell counter is less or equal to *iaf*. The physical interpretation of this can be seen in fig. 1.3.

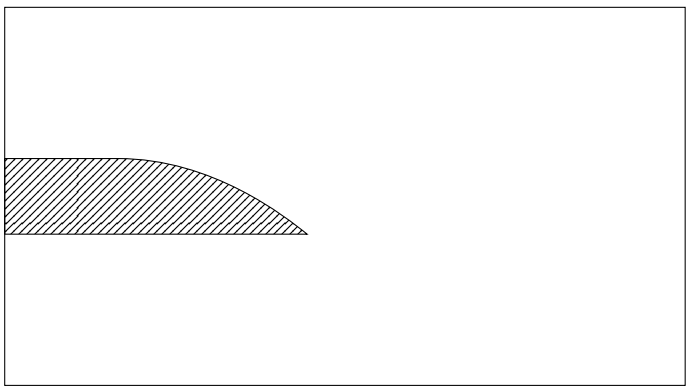


Figure 1.1: Case setup including the trailing edge of the airfoil, not drawn to scale.

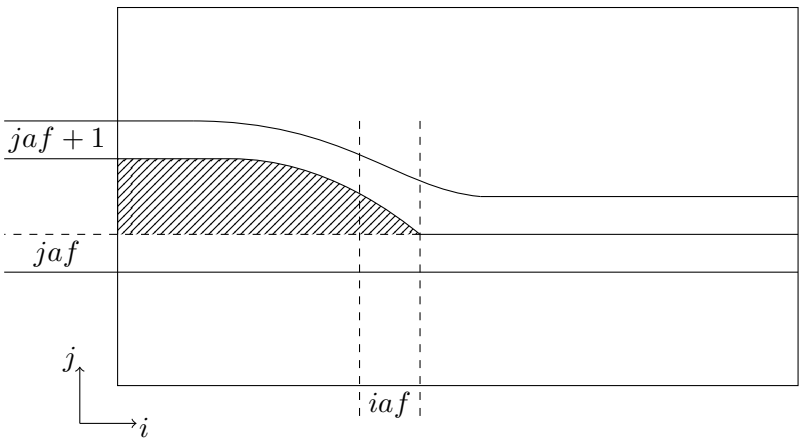


Figure 1.2: Variables used to track the airfoil location in the computational domain.

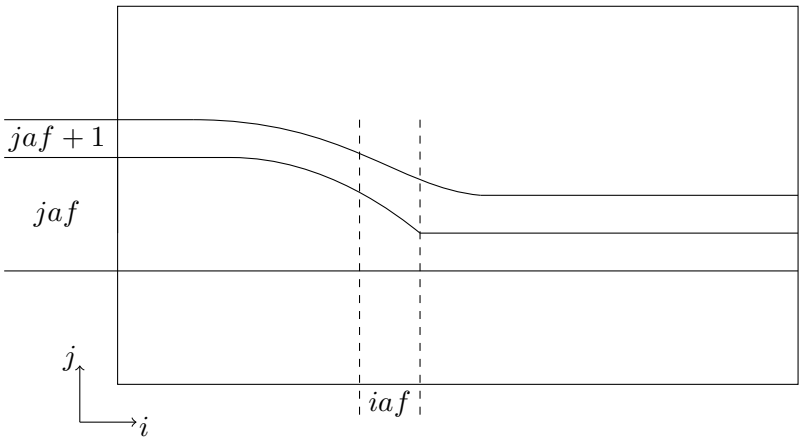


Figure 1.3: Variables used to track the airfoil location in the computational domain seen in default setup.

To avoid this problem the calculation of the center locations uses the y coordinate of the corner of the cell with the x counter *iaf* and the y counter *jaf* i.e. the last cell along the x axis below the airfoil. The same strategy is used when calculating the areas and volumes of the cells below the airfoil.

Due to the presence of the airfoil and the one block configuration a problem occurs with the areas located between the cells *jaf* and *jaf*+1. This is the areas that now should represent the top and bottom of the airfoil. So they are no longer representing the same areas between the cells which is the normal case when a one block configuration is used. The northern areas of the *jaf* cells should represent the bottom of the airfoil while the southern areas of the *jaf*+1 cells should represent the top of the airfoil. This problem was solved by introducing a new variable. The northern areas of the *jaf* cells have the standard variable name ***arean*** while the southern areas of the *jaf*+1 cells will

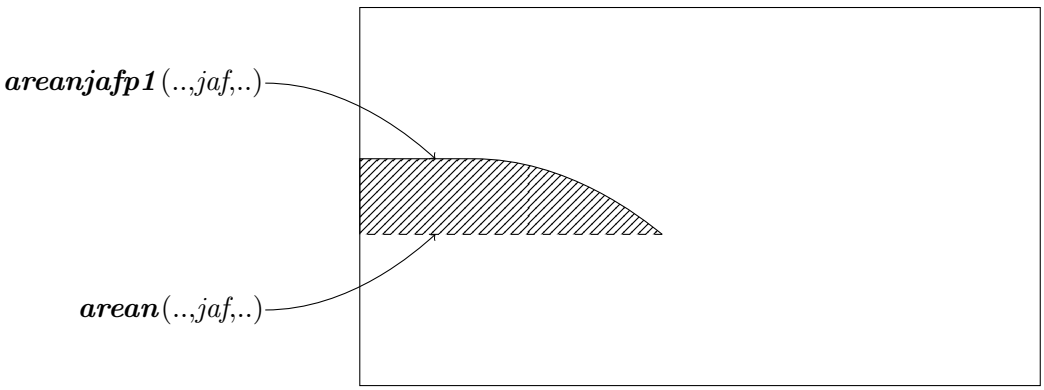


Figure 1.4: Varibale names of the different areas on top of and below the airfoil.

be identified by the variable ***areanjafp1***.

The use of the different area variables can be seen in fig. 1.4. The primary reason for implementing it like this is that the southern areas of the $jaf+1$ cells are used at fewer locations in the code than the northern areas of the jaf cells. The second contributing factor is that the bottom of the airfoil is flat which makes it straightforward to find the correct coordinates of these nodes. If the line in the mesh would have continued along the bottom instead of the top of the airfoil, the bottom corners of the top cells would have been needed to be provided to the code separately and needed to be changed for each mesh to avoid approximations. The volumes of the cells on top of the airfoil are separately calculated using the corner coordinates specified by the mesh and using the south areas from the variable ***areanjafp1***. In the default setup the volumes on top of the airfoil would have been calculated correctly but due to the modifications of ***arean*** now only representing the cells below the airfoil an error is introduced if they are not calculated separately. The calculations described above are in CALC performed in the file init.f seen in appendix A.

1.2 Finite Volume Coefficients

If there is a body in the flow, the flow will react differently compared to if the body wasn't there. In the one block configuration with the trailing edge of an airfoil in the flow and the modifications above the body has no surface for the flow to feel. The physics of the body need to be introduced as modifications of how the flow is handled by the code. One modification needed is if the airfoil is not in the flow all the cells should send information to each other. But when the airfoil is present the cells on top of and below the airfoil should not have any connection to each other. One step is to put the coefficient a_N to zero for the cells below the airfoil [2]. For the cells on the top of the airfoil coefficient a_S is put to zero. The cells are now isolated from each other for all the ϕ variables e.g. u, v, w. In CALC this is done in the file mod.f.

Another modification is that the convective fluxes from the bottom of the airfoil are

set to zero to represent that no flow is allowed to flow through the airfoil. This is made in the mod.f file in CALC. The convective fluxes in CALC consists of a velocity-pressure coupling, for this the Rhie-Chow interpolation is used. The Rhie-Chow interpolation is used rather than linear interpolation to make sure that no unphysical oscillations will appear in the velocity and pressure fields [3]. In the interpolation values at four nodes are used. To have no connection along the y axis between the top and bottom of the airfoil the interpolation for the nodes close to the airfoil cannot use values on the other side of the airfoil. This modification can be seen in the conv.f file, the modified section of conv.f can be seen in appendix B.

1.3 No slip condition via source terms

The one block configuration with no surface for the flow to feel also introduces a problem because the no slip condition needs to be fulfilled at the surface of the airfoil. Here it is introduced as a boundary condition between the cells on top of and below the airfoil. The boundary condition is implemented by the use of source terms. The connection between the cells on top of and below the airfoil is cut. This is done by putting the coefficient a_N to zero for the cells below the airfoil and a_S to zero for the cells on top of the airfoil [2]. This modification is identical to what is done to cut the connection between the cells above and below the airfoil. So no further modification of the code is needed to fulfill this criterion.

The source term implementation uses the original values of the coefficients a_N and a_S and implement them via the source terms S_U and S_P. For the cells below the airfoil the source terms will be

$$S_P = -a_N, S_U = a_N \phi_{airfoil}$$

The no slip condition only applies to the turbulent kinetic energy k and the velocity components thereby ϕ refers to k, u, v or w. Due to the no slip condition at the surface of the airfoil, all of the quantities represented by ϕ are equal to zero at the surface. This results in a cancelling of the contribution to the source term S_U. The unmodified coefficients a_N and a_S in CALC contain contributions from both convection and diffusion. This implementation is only for the cells closest to the airfoil so the contribution due to convection is zero resulting in

$$S_P = -a_{N_{diffusion}}, S_U = 0$$

The source terms are modified in the same way for the top of the airfoil only switching from the north coefficient a_N to the south coefficient a_S

$$S_P = -a_{S_{diffusion}}, S_U = 0$$

1.4 Derivatives of pressure close to the airfoil

Next to a surface the pressure change along the normal direction is zero. In the case of the airfoil the pressure change along the y direction next to the airfoil should be zero due to the boundary condition $d\phi/dn = 0$. In CALC the pressure gradient over a cell is calculated using the values at the faces. To interpolate ϕ here representing the pressure from the node center to the faces linear interpolation with weight functions f is used in CALC [3]. This can be seen in the files dpdxf.f, dpdyf.f, dpdzf.f and in the dpdxfo.f, dpdyfo.f, dpdzfo.f. For the northern face of a cell the interpolation is made using formula

$$\phi_n = f_{y_P}\phi_N + (1 - f_{y_P})\phi_P \tag{1.1}$$

When interpolating the values of the northern faces for the cells below the airfoil, the cells on top of the airfoil should not have any connection to it. This is obtained by choosing the value of zero for the weight function f_{y_P} in eq. 1.1 for these cells and this is done in the file mod.f. The northern face value are thereby set to the same values as the center of the cell and a zero $d\phi/dn$ between the cell center and the airfoil is the result as required.

In the same way as for the cells below the airfoil, the value at the faces of the cells on top of the airfoil should not have any connection to the value of the cells below the airfoil. In the same way as above the airfoil a zero pressure change along the y direction next to the surface below the airfoil should be obtained. To interpolate ϕ from the center to the southern face the following formula is used

$$\phi_s = f_{y_S}\phi_P + (1 - f_{y_S})\phi_S \tag{1.2}$$

No contact between the cells on top of and below the airfoil occurs if f_{y_S} is set to one for the cells on top of the airfoil. It should be noted that the weight function f_{y_S} in eq. 1.2 for the cells on top of the airfoil is the same weight function as f_{y_P} in eq. 1.1 for the cells below the airfoil. This weight function has from above already been set to zero in favour of the cells below the airfoil; but for the cells on top of the airfoil a value of one is required. This is solved by an if statement to override the south face value after it has been calculated using the center value of the cell above the airfoil. This results in a zero $d\phi/dn$ between the cell center and the airfoil. The special area variable ***areanjafp1*** is used in the calculation to represent the south area of the cell at the top of the airfoil. The implementation in all the files are made in the same way and as an example the implementation in the dpdyf.f file can be seen in appendix C.

1.5 Multigrid

To obtain the pressure field in CALC a multigrid solver is used. The multigrid solver is prefered compared to the tri-diagonal matrix algorithm (TDMA) because of its speed up of the convergence rates. Further information about the theory of the multigrid solver and its default implementation in CALC can be found in [4].

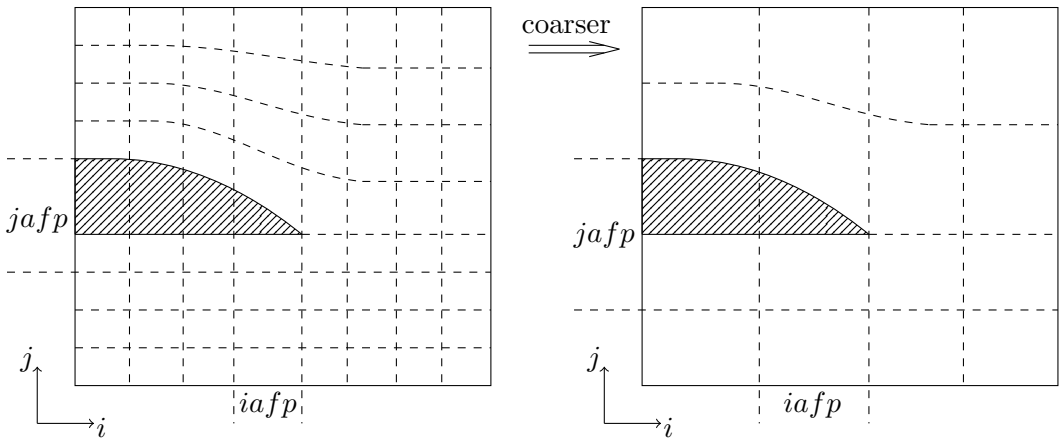


Figure 1.5: *iafp* and *jafp* through different multigrid levels.

Many modifications in the multigrid solver for the pressure can be recognized from earlier sections for the velocity components. The major differences are that the multigrid solver uses 1D arrays while the flow solver uses 3D arrays. Another difference is that in the multigrid solver the variables has to be traced through the different multigrid levels.

1.5.1 Geometry

The location of the last cell along the x axis below the airfoil is named *iaf* from the implementation in the flow solver. The cell that would by default be treated as inside the airfoil along the y axis is named *jaf*. These variables are now needed to be traced through the different multigrid levels, which is why new variable names are introduced. The variable names were changed from *iaf* and *jaf* to *iafp* and *jafp*. The original variables are integers but the new ones are 1D arrays containing integers. The size of the arrays corresponds to the number of multigrid levels used. For this implementation to work the original choice of the *iaf* and *jaf* has to be in such a way so that they through each multigrid level still corresponds to the same positions with respect to the airfoil. An example when this is applied between two multigrid levels can be seen in fig. 1.5. In the same way as before the center locations, areas and volumes of the cells below the airfoil needs to be modified. The modifications are made in the same way as before but the coding could at first appear as different because of the 1D arrays instead of 3D arrays and that the operations are made at each multigrid level. The southern areas of the cells on top of the airfoil have a different variable name compared to the original ***areanjafp1*** now changed to *anjafp1*, *aljafp1* and *ahjafp1* for the x, y and z part of the area. The modifications explained above can be seen in the file peter_init.f in appendix D.

1.5.2 Finite Volume Coefficients

The coefficients a_N and a_S are treated in the same way as for the velocity. a_N is set to zero for the cells below the airfoil and for the cells on top of the airfoil the coefficients a_S

are set to zero and they are traced through the different multigrid levels. This is done at the end of the file peter_init.f seen in appendix D.

1.5.3 Prolongation

When going from a multigrid level of a coarse grid to one with a finer grid prolongation interpolation is used. The type of interpolation for the prolongation is in CALC bilinear for the two dimensional part of the multigrid solver and trilinear for the three dimensional part [4]. The three dimensional prolongation used in CALC follow the formula

$$\begin{aligned}\delta^2_{i,j,k} =& \frac{1}{64}(27\delta^1_{I,J,K} + 9\delta^1_{I+1,J,K} + 9\delta^1_{I,J+1,K} + 9\delta^1_{I,J,K+1} \\ &+ 3\delta^1_{I+1,J+1,K} + 3\delta^1_{I+1,J,K+1} + 3\delta^1_{I,J+1,K+1} \\ &+ \delta^1_{I+1,J+1,K+1})\end{aligned} \tag{1.3}$$

The indices written as uppercase letters together with the superscript 1 in eq. 1.3 and the following equations in this section indicate locations in the coarser multigrid level. The indices as lowercase letters together with the superscript 2 indicate locations in the finer multigrid level. These notations are consistent with what is used in [4]. The formula used in the two dimensional part of the multigrid solver is

$$\delta^2_{i,j} = \frac{1}{16}(9\delta^1_{I,J} + 3\delta^1_{I+1,J} + 3\delta^1_{I,J+1} + \delta^1_{I+1,J+1}) \tag{1.4}$$

In the one block setup the interpolation is not allowed to include nodes from both below and on top of the airfoil. From the equations above 1.3 and 1.4 it can be seen that they include δ with both index δ_J and δ_{J+1}. By default CALC performs the prolongation step and interpolation using nodes both below and on top of the airfoil. The solution for the nodes below the airfoil is to choose δ_{J+1} equal to δ_J and for the nodes on top of the airfoil to choose δ_J equal to δ_{J+1}. For the three dimensional eq. 1.3 the nodes below the airfoil will now use the following prolongation step.

$$\begin{aligned}\delta^2_{i,j,k} =& \frac{1}{64}(27\delta^1_{I,J,K} + 9\delta^1_{I+1,J,K} + 9\delta^1_{I,J,K} + 9\delta^1_{I,J,K+1} \\ &+ 3\delta^1_{I+1,J,K} + 3\delta^1_{I+1,J,K+1} + 3\delta^1_{I,J,K+1} \\ &+ \delta^1_{I+1,J,K+1}) \\ =& \frac{1}{64}(36\delta^1_{I,J,K} + 12\delta^1_{I+1,J,K} + 12\delta^1_{I,J,K+1} + 4\delta^1_{I+1,J,K+1}) \\ =& \frac{1}{16}(9\delta^1_{I,J,K} + 3\delta^1_{I+1,J,K} + 3\delta^1_{I,J,K+1} + \delta^1_{I+1,J,K+1})\end{aligned} \tag{1.5}$$

The connection to eq. 1.4 can be seen because the the resulting eq. 1.5 is a two dimensional prolongation formula. The difference is that eq. 1.4 is in the x-y plane and eq. 1.5

is in the x-z plane. The modification implemented in the CALC code can be seen in the peter_multi.f file in appendix E. In the two dimensional case eq. 1.4 will for the cells below the airfoil take the following shape

$$\begin{aligned}\delta^2_{i,j} =& \frac{1}{16}(9\delta^1_{I,J} + 3\delta^1_{I+1,J} + 3\delta^1_{I,J} + \delta^1_{I+1,J}) \\ =& \frac{1}{16}(12\delta^1_{I,J} + 4\delta^1_{I+1,J}) \\ =& \frac{1}{4}(3\delta^1_{I,J} + \delta^1_{I+1,J})\end{aligned} \tag{1.6}$$

Also in the two dimensional case the prolongation eq. 1.6 decreases one dimension from the original eq. 1.4. The two dimensional prolongation for the nodes closest to the airfoil will be linear along the x axis instead of two dimensional in the x-y plane. This can be seen in the mg_2d.f file in appendix F.

The mg_2d.f file is coded in a way to be able to handle all possible two dimensional combinations i.e. the planes x-y, x-z and y-z without the knowledge where the plane is located inside the computational domain. When the one block configuration is used containing the trailing edge of an airfoil the code needs to know if the plane cuts through the airfoil or not. For the x-y plane the variables *iafp* and *jafp* have been added to the parameter list in the file peter_relax.f seen in appendix G. The x-z plane will never cut through the airfoil so no further information is needed for the prolongation in this plane. The location of the y-z plane may or may not cut through the airfoil. The variable *taf* was introduced for this purpose. If *taf* is true the plane will cut through the airfoil and the prolongation should be according to eq. 1.6 otherwise according to eq. 1.4.

1.6 Modifications needed for some turbulence models

In some of the turbulence models further modifications are needed to make the implementation work properly.

1.6.1 Distance to the airfoil

The shortest distance to the airfoil for each cell center is used in some of the turbulence models available. One of them is the Reynolds Averaged Navier Stokes (RANS) model implemented in CALC. The distance is here used in a damping function for calculating the viscosity and can be seen in the file vist_pans.f. The coordinates of the airfoil is found from the mesh and linear interpolation is used in between the given values. As a first guess it is assumed that the coordinate on the airfoil has the same x coordinate as the center of the cell. The guessed x coordinate and the y coordinate of the cell is used to orientate on the airfoil where the shortest distance to the cell center can be found. The y coordinate on the airfoil corresponding to the guessed x coordinate is found and the distance between it and the cell center is calculated. The second guess of the x coordinate moves the guessed x coordinate value a bit along the positive x axis. The

moving distance is dependent by the variable *movedivide*. This variable is dependent on the geometry of the airfoil and a value of 1000 was chosen for this setup. Again the y coordinate on the airfoil is found and the distance to it is calculated. The algorithm now has enough information to on its own make guesses whether to go along the positive or negative x axis and the distance to move. A maximum of 100 guesses of x locations for each cell is made specified using the variable *countMax*. The distance to the airfoil calculation is done in the mod.f. file and can be seen in appendix H.

1.6.2 Derivatives of ϕ close to the airfoil

In all of the available turbulence models the derivative of the variable ϕ is used. The Large Eddy Simulation (LES) model is one of them. It is here part of the calculation of viscosity and the ϕ variable is one of the three velocity components. This can be seen in the file vist_les.f. At the airfoil due to the no slip condition the derivative should be taken with a value of zero at the airfoil surface. When using a one block mesh there exists no values at the airfoil surface. To represent these values when the derivatives next to the surface of the airfoil is calculated. The northern face value of the cells below the airfoil i.e. at *jaf* is set to zero. In the same way the southern face value of the cells on top of the airfoil i.e. at $jaf+1$ is set to zero. For the $jaf+1$ cells also the south areas are specified using the variable ***areanjafp1***. The derivatives are calculated in the files dphidx.f, dphidy.f and dphidz.f. The implementation in all the files are made in the same way and as an example the implementation in the dphidy.f file can be seen in appendix I.

2

Trailing edge flow: validation

To validate the implementation in CALC a flat plate and an airfoil setup is used.

2.1 Flat plate

Two cases are used as validation cases for the flat plate. For each case two simulations are performed, the first a one sided setup where all implementations done is commented away. The second a two sided case where the top and bottom results should be identical. The first test case is a laminar flat plate simulation where external results are used to validate the simulation results. The second case is a RANS simulation of a flat plate flow with comparison between the two simulations.

In all the simulations performed boundary conditions are applied to the planes limiting the x, y and z axis. These planes are assigned names according to fig. 2.1.

2.1.1 Laminar

The laminar case is a steady and incompressible two dimensional flow along a flat plate. The case setup can be found in appendix B in [5]. The given data is obtained using the same code as the implementation is made in i.e. CALC but without the implementation. The basic setup of the case can be seen in fig. 2.2, with the flow simulated 0.19m before

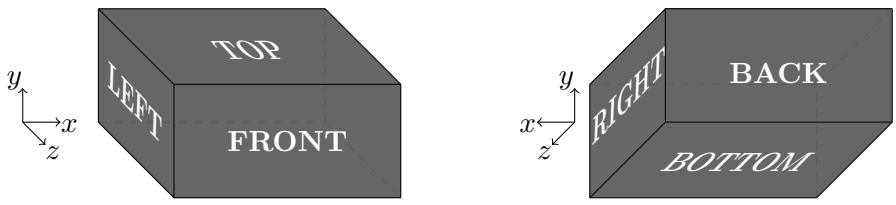


Figure 2.1: Name declaration of the planes in the computational domain.

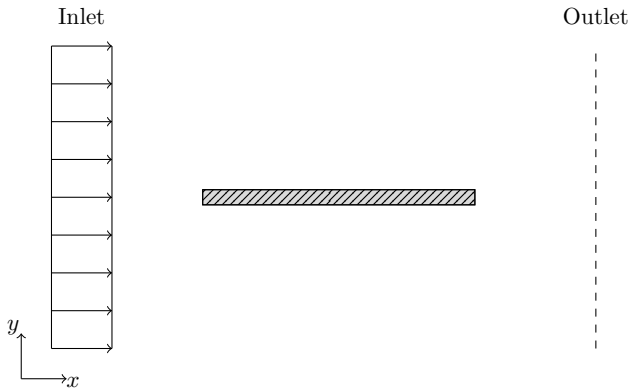


Figure 2.2: The case setup, not drawn to scale.

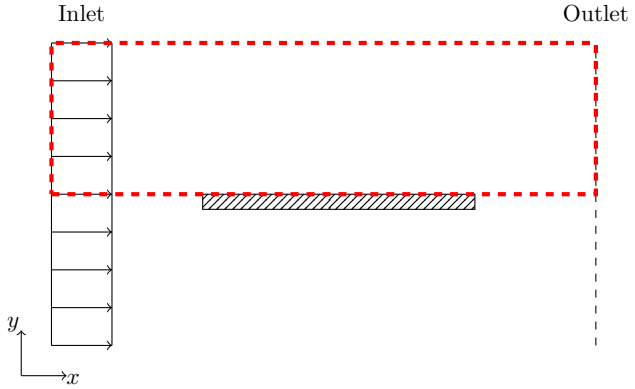


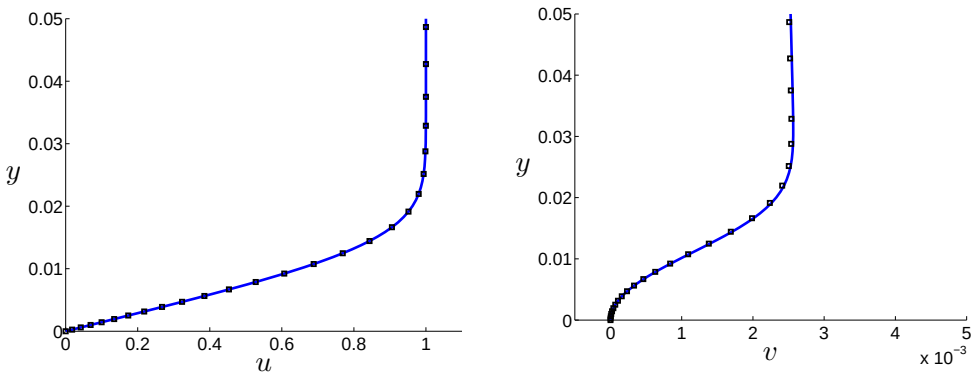
Figure 2.3: The one sided case setup, not drawn to scale.

it reaches the flat plate. The media simulated is air at 20°C at a uniformed velocity of 1m/s. The discretization scheme used is the hybrid scheme, personal communication [6]. The hybrid discretization scheme is also used in all the flat plate simulations performed to validate the implementation.

One sided

When the code is run as a one sided flat plate the whole case seen in fig. 2.2 is not needed to be simulated; only the top part of the plate is used as seen in fig. 2.3 with the computational domain marked in red. It has a length of 2.461m and a height of 1.289m. The mesh used has 252 nodes along the x axis and 200 nodes along the y axis.

The presence of the flat plate is simulated by applying boundary conditions to the bottom boundary named BOTTOM in fig. 2.1. For the velocity components the following boundary conditions are applied. A wall boundary condition is used at the flat plate and a symmetry boundary condition is used before and after the plate. The inlet condition is applied to the plane named LEFT in fig. 2.1. The inlet boundary condition is an



(a) u velocity component (b) v velocity component

Figure 2.4: Comparison of the velocity components profiles at x equal to 1.5645m between the simulation and the given data. ——: Data, □: Simulation.

uniformed u velocity with zero flow along the y axis. On the plane named RIGHT an outlet Neumann boundary condition of zero change of the u and v component of the velocity along the x axis is applied. The boundary named TOP has a symmetry boundary condition along the whole boundary. Cyclic boundary conditions are applied to the planes named FRONT and BACK, these boundary conditions are also used for the pressure on these planes. On all the other planes seen in fig. 2.1 Neumann boundary conditions of zero gradient are applied for the pressure.

The onesided case is run using a numerous of if statements in the implementation and a value of -1 for the variables *iaf* and *jaf*. The implementation is not made to easily handle this case but to make sure that nothing of the original code has been damaged during the implementation this case is run. The u and v velocity profiles from the simulation are compared to the profiles from the given data close to the flat plate at x equal to 1.5645m seen if fig. 2.4. The u velocity profile of the simulation is spot on the given data. A difference between the simulation and the given data can be seen in the v component of the velocity profile. The magnitude of the v velocity should be noted which is 10^{-3} so both the results show acceptably good agreement to the given data.

Two sided

The implementation makes it possible to simulate the flat plate as infinitely thin. This results in that the simulations of both the one sided and two sided flat plate should give the same results. The computational domain used to simulate the two sided flat plate can be seen in fig. 2.5, the black line indicates the location of the flat plate.

The two sided flat plate computational domain is the one sided flat plate but mirrored along the y equal to zero plane. This simulation should thereby give the same results for the profiles of the u and v components of the velocity as the one sided simulation.

The u and v velocity profiles at x equal to 1.5645m are compared to the data given close to the flat plate in fig. 2.6. The mesh used both for the simulation on top and

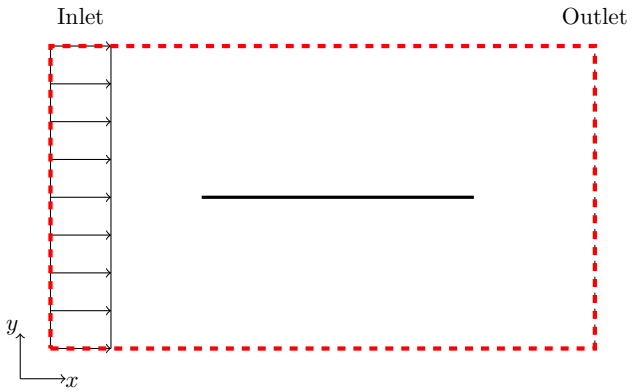
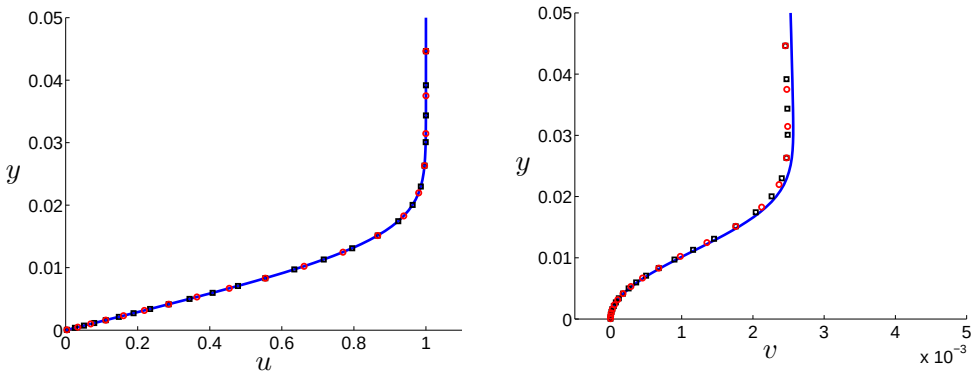


Figure 2.5: The two sided case setup, not drawn to scale.



(a) u velocity component (b) v velocity component

Figure 2.6: Comparison of the u and v velocity profiles at x equal to 1.5645m between the simulation and the given data. ——: Data, □: Simulation on top, ○: Simulation below.

below the plate is identical. The u velocity profiles of the two sided flat plate show good agreement to the data. For the v velocity component profile the values on top of and below the airfoil are spot on each other. But their values differ toward the given data. In the same way as for the one sided flat plate simulation the magnitude of the v velocity component is 10^{-3}, so the seen difference is accepted. Also the two sided simulation show good agreement toward the given data.

2.1.2 RANS

In this validating case a flat plate is also used, but here it starts from the beginning of the computational domain. The RANS $k-\varepsilon$ turbulence model is use to predict the behavior of the flow. The inlet profiles used in this case is not uniformed as can be seen in fig. 2.7.

The length of the flat plate used in this validating case is 6m and the length between

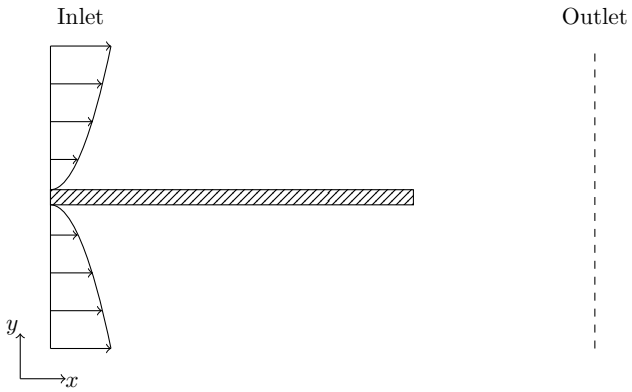


Figure 2.7: The case setup, not drawn to scale.

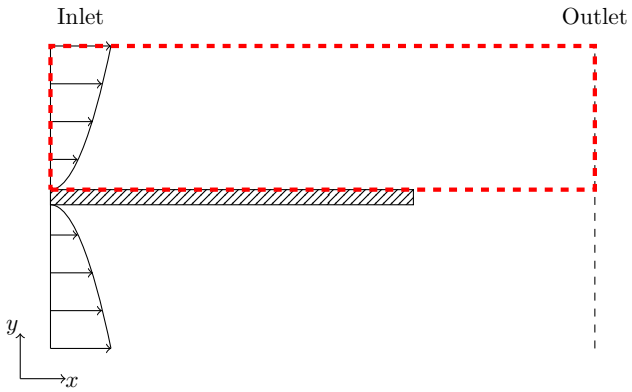


Figure 2.8: The one sided case setup, not drawn to scale.

the inlet and outlet is 12m. The inlet profiles used in the simulations are the profiles that will be used on top of the airfoil at a momentum thickness of 4100 for the airfoil simulation. The simulation for the one sided plate uses the computational domain marked in red in fig. 2.8.

In the same way as for the earlier validating case the flat plate is simulated using a wall boundary condition on the plane in the computational domain named BOTTOM in fig. 2.1. After the flat plate along the x axis a symmetry boundary condition is used. The height of the computational domain is 3.8486m. The mesh used has 50 nodes along the x axis and 70 nodes along the y axis. In the two sided simulation this validation case uses in the same way as the laminar validating case an infinitely thin flat plate. The computational domain is marked in red in fig. 2.9.

The inlet profiles used in the simulation seen in fig. 2.9 are identically the same on top of and below the flat plate with respect to the flat plate i.e. the sign of the v velocity profile is changed for the profile below the flat plate. The profiles for u, v, k and ε of the one sided and on each side of the two sided simulation are compared at x equal to 4.625m.

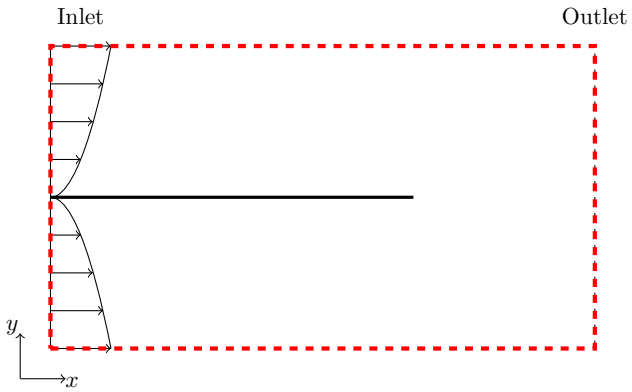
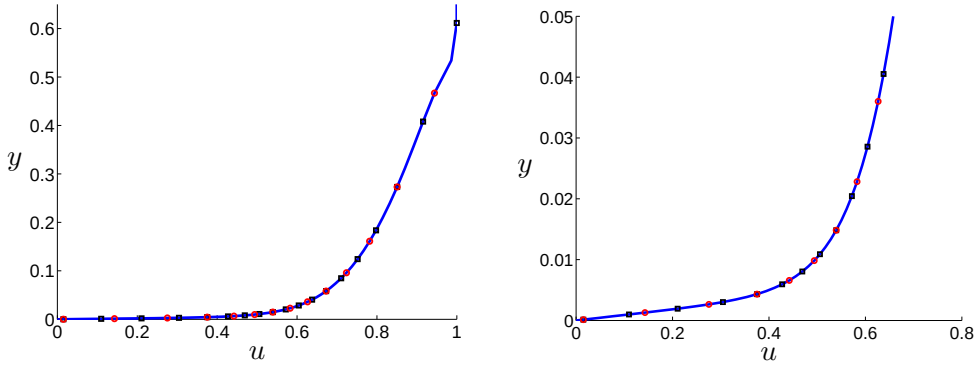


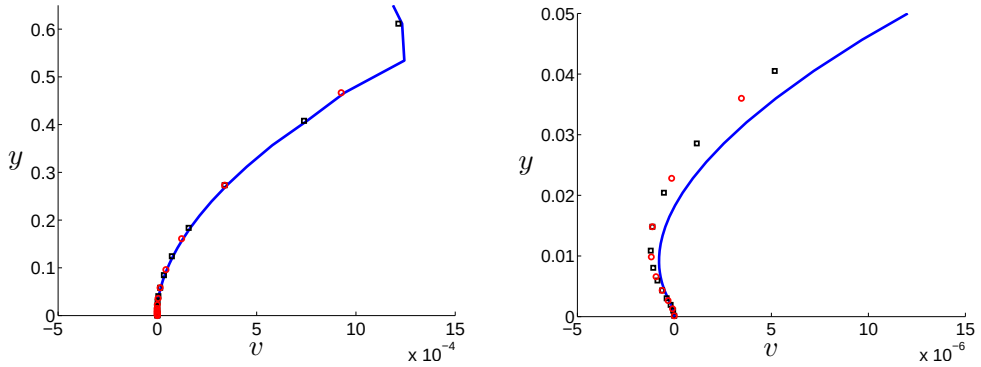
Figure 2.9: The two sided case setup, not drawn to scale.



(a) Boundary layer (b) Zoom near the flat plate

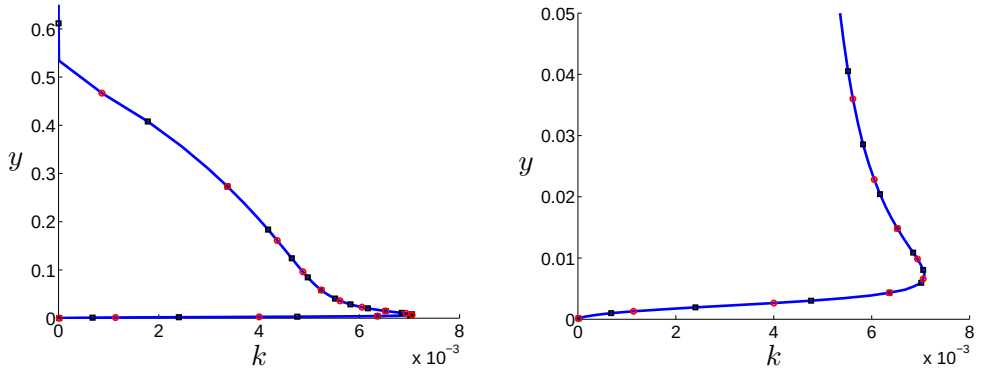
Figure 2.10: Comparison of the u velocity profile at x equal to 4.625m between the simulations. ——: One sided, □: Simulation on top, ○: Simulation below.

The result of the u velocity profiles is seen in fig. 2.10 show that the predicted values both on top of and below the flat plate in the two sided simulation is spot on the predicted values from the one sided simulation. In fig. 2.11 it can be seen that the predicted values of the v velocity profiles on top of and below the airfoil gives more or less the same results. But they both differ compared to the one sided simulation. But as could be seen by the x axis of both the figures shown in fig. 2.11 their magnitudes are small i.e. 10^{-4} for the left figure and 10^{-6} for the right one. So the results given by both the simulations are acceptably close to each other. Even when the k profiles are zoomed in close to the flat plate as seen in the right figure in fig. 2.12 the predicted values from all the simulations seems to give the same results. Also the predicted ε values were compared between the simulations and can be seen in fig. 2.13. In the same way as the k profiles the ε profiles from the simulations seem to predict identically the same results even when zooming in close to the flat plate.



(a) Boundary layer (b) Zoom near the flat plate

Figure 2.11: Comparison of the v velocity profile at x equal to 4.625m between the simulations. ——: One sided, □: Simulation on top, ○: Simulation below.

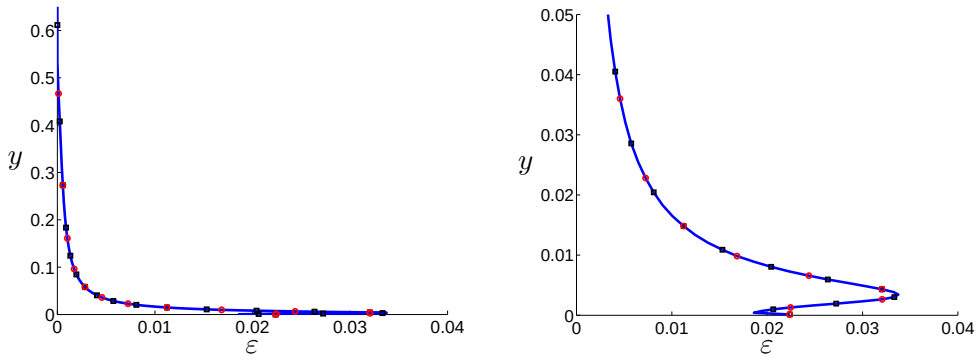


(a) Boundary layer (b) Zoom near the flat plate

Figure 2.12: Comparison of the k profile at x equal to 4.625m between the simulations. ——: One sided, □: Simulation on top, ○: Simulation below.

2.1.3 Conclusions of flat plate cases

All the compared profiles show similar results within accepted difference in the margins between the one and two sided simulations and the given data. This shows that nothing of the original code has been damaged during the implementation. The results obtained from the top of and below the flat plate are identically the same results. So the implementation makes the flow behave in the same way on top of and below the flat plate which is an important point to show for this implementation to be correct.



(a) Boundary layer (b) Zoom near the flat plate

Figure 2.13: Comparison of the ε profile at x equal to 4.625m between the simulations. ——: One sided, □: Simulation on top, ○: Simulation below.

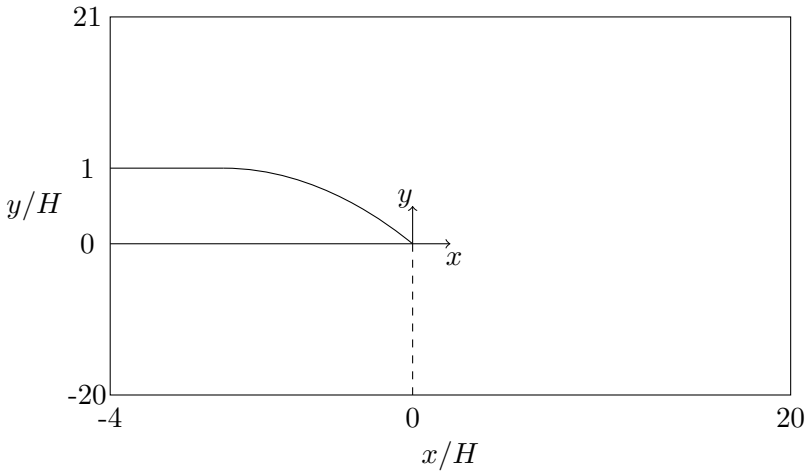


Figure 2.14: Dimensions of the computational domain including the trailing end of the airfoil, not drawn to scale.

2.2 Airfoil

In the same way as the implementation can handle an infinitely thin flat plate it can also handle the trailing edge of an airfoil inside the computational domain. The only thing that is different when the airfoil is run instead of the flat plate is the mesh used.

2.2.1 Computational domain

The computational domain is shaped to be able to compare the results from simulations with the experiments by Blake in [7]. The chosen dimensions of the computational domain can be seen in fig. 2.14.

The origin of the computational domain is located at the trailing edge of the airfoil as seen in fig. 2.14. Along the y axis the computational domain extents $y/H = 20$ units on each side of the airfoil, i.e. it reaches from y/H = -20 to $y/H = 21$. The thickness of the airfoil in the y direction is defined as H. Along the x direction the domain extend from x/H = -4 to $x/H = 20$. The width of the computational domain is chosen to $0.5H$ along the z axis.

The size of the computational domain along the y axis is consistent with the three simulations by Wang in [8] and Wang *et al.* in [9, 10]. One of the simulations made by Wang in [8] uses a computational domain of height $82H$ and Gritskevich *et al.* uses in [11] a domain reaching from y/H = -40 to $y/H = 40$. These computational domains are about double the size as the one in fig. 2.14 but in the articles using the smaller computational domain no issues regarding boundary conditions interfering with the simulation seemed to be present. So the smaller computational domain was chosen to save Central Processing Unit (CPU) time. Along the x axis the computational domain uses the same dimensions and positioning of the airfoil as in [11]. The only difference is that in this implementation both the top and bottom inlets will be located at x/H = -4, while [11] has the top inlet at x/H = -4 and the bottom at x/H = -1. Wang *et al.* uses larger dimensions of the domain including the airfoil with x axis lengths of $20H$ [8] and $16.5H$ [8, 9, 10]. In all these simulations more of the flow on the sides of the airfoil is simulated e.g. in the $16.5H$ simulations the inlets are positioned at about x/H = -8. The x/H = -4 inlet was chosen to save CPU time and the inlet conditions from the simulations by Wang *et al.* were recalculated for the new positions, further information of how this was done is presented later in the report. The computational domain in the simulations by Wang *et al.* extends to about $x/H = 8$. The longer distance towards the outflow used in [11] were preferred to make sure that the outflow boundary condition does not affect the results of the simulation. The width of the computational domain used in [8, 9, 10, 11] was $0.5H$ which is also chosen for this simulation. The curve describing the trailing edge of the airfoil used in this simulation is the same as used by Bentaleb *et al.* in [12].

$$\begin{aligned} y_{wall} &= (1 - R_1) + \sqrt{R_1^2 - x^2} \quad \text{for} \quad 0 \leq x/H < 2.3 \\ y_{wall} &= y_2 - \sqrt{\frac{R_1^2}{4} - (x_2 - x)^2} \quad \text{for} \quad 2.3 \leq x/H < 2.835 \\ y_{wall} &= R_2 - \sqrt{R_2^2 - (2.937 - x)^2} \quad \text{for} \quad 2.835 \leq x/H < 2.937 \end{aligned} \tag{2.1}$$

In eq. 2.1 the constants are $R_1 = 4.03H$, $R_2 = 0.334H$ and $y_2 = 1.936H$. The trailing edge of the airfoil is located at $x/H = 0$ and $y/H = 0$ and the computational domain is built around it.

2.2.2 Boundary conditions

The boundary conditions specified for the velocity components and the pressure are not identical at all the planes. The planes of the computational domain where the boundary

conditions are applied can be seen in fig. 2.1.

Velocity

At the planes TOP and BOTTOM seen in fig. 2.1 the boundary condition specified is symmetry. This simulates that the airfoil is located far away from being affected by any other body. The condition means that no flow will pass through this plane and the gradients of all the velocity components at the plane is zero, which is resonable since the flow on each side of the plane is identical to each other if no disturbances are present. This boundary condition has to be applied sufficiently far away from the airfoil so that the presence of the airfoil does not cause any flow through the boundaries. On the planes named FRONT and BACK according to fig. 2.1 cyclic boundary condition are applied. This boundary condition means that the airfoil has an infinitely length along the z axis. The outflow boundary condition applied at plane RIGHT in fig. 2.1 is a Neumann boundary condition with zero change of the velocity component in the normal direction to the plane. The inlet boundary condition used at plane LEFT in fig. 2.1 is taken from a Reynolds Averaged Navier Stokes (RANS) simulation of a flow along a flat plate with data from a location of prescribed local momentum thickness.

Pressure

The boundary condition for the pressure is cyclic on the planes FRONT and BACK in fig. 2.1. On the other planes a Neumann zero gradient boundary condition is applied.

2.2.3 Inlet data

Local momentum thickness

To be able to compare the results obtained from the simulations using this implementation with the experiments performed by Blake in [7] the RANS data used at the inlet have to have the same Reynolds number Re_Θ based on the local momentum thickness and boundary layer edge velocity as the flow in the experiments. Wang *et al.* uses in [9] Re_Θ equal to 2760 for the profiles below the airfoil and a Re_Θ value of 3380 on top of the airfoil to duplicate the experiments performed by Blake. The dimensions of the computational domain used in the simulation performed by Wang *et al.* are different. The inlet used by Wang *et al.* is located at $x/H = -8$. The mesh used in this implementation has an inlet located at $x/H = -4$. At these new locations the momentum thicknesses on top and below the airfoil are needed. They can be found using the data from the experiments performed by Blake and the formulas from Apsley in [13]. Blakes experiments are performed using a flat strut with thickness 2 inches at a velocity of 100 ft/s at the Reynolds number based on the thickness of the flat strut Re_H of $1.02 \cdot 10^5$ [7]. Using the definition of Re_H the kinematic viscosity ν used in the experiments performed by Blake can be found from.

$$Re_H = \frac{u_\infty H}{\nu}$$

Next the momentum thickness Θ for the Re_Θ used by Wang *et al.* is found using the definition of Re_Θ seen below.

$$Re_\Theta = \frac{u_\infty \Theta}{\nu}$$

The x location can be found from the obtained data using the formulas by Apsley in [13]. Here the momentum integral relation for a zero pressure gradient boundary layer and the power law approximation is used to obtain the formulas presented below for the momentum thickness and the boundary layer thickness δ.

$$\Theta = \frac{7}{72}\delta \tag{2.2}$$

$$\frac{\delta}{x} = 0.166 Re_x^{-1/7} \tag{2.3}$$

In eq. 2.3 Re_x is the Reynolds number at position x i.e.

$$Re_x = \frac{u_\infty x}{\nu}$$

By combining eq. 2.2 and 2.3 the x location for the simulation by Wang *et al.* can be found. Now we use the difference in x location towards our inlet and eq. 2.2 and 2.3 is again used; now to find the new momentum thickness. The RANS simulation result used as part of the inlet profiles in this implementation applied at the top of the airfoil has a momentum thickness of 4100 and below the airfoil a momentum thickness of 3500.

Scaling

The inlet data used in the simulation need to be rescaled because the Re_Θ values were the same, but the momentum thicknesses did not have the same values as have been calculated for the locations of the inlets in the mesh. To obtain the same momentum thickness the momentum thickness of the data is calculated and a factor is created to obtain the required momentum thickness. This factor is then applied to the center and corner locations of the nodes in the data.

The data that should be used as inlet profiles in CALC should have a maximum value of u equal to one. The original maximum u value is larger than one and is used as a factor. The factor is applied to the quantities depending on their velocity dependence e.g. u and v is scaled only by the factor while the turbulent kinetic energy k is scaled by the factor up to the power of two. The dissipation ε is needed to be scaled in a different

way. This to fit the boundary condition at the surface of the airfoil presented in [14] and seen below

$$\varepsilon = \frac{2k\nu}{y^2} \tag{2.4}$$

In eq. 2.4 y is defined as the normal distance from the airfoil. To scale the dissipation to fit with the boundary condition the used scale factor is presented below

$$\varepsilon_{factor} = \frac{(\frac{k_{sim}}{k_{data}})(\frac{\nu_{sim}}{\nu_{data}})}{(\frac{\delta_{99,sim}}{\delta_{99,data}})^2} \tag{2.5}$$

In eq. 2.5 k_{sim} is the turbulent kinetic energy that will be used in the simulation in CALC and k_{data} is for the given data. In the same way as for the turbulent kinetic energy ν_{sim} is the kinematic viscosity that will be used in the CALC simulation and ν_{data} is for the given data. The $\delta_{99,sim}$ is the thickness of the boundary layer in the simulation where the limit is when u is 99% of the u velocity infinitely away from the airfoil often defined as u_∞. The $\delta_{99,data}$ is the thickness of the boundary layer for the given data.

The data are interpolated from the given nodes of the data to the nodes of the mesh. Linear interpolation is used. The given data value for the dissipation at the airfoil can not be used in the interpolation if nodes of the mesh are located closer to the airfoil than the data nodes, because this value is not the correct boundary value. To obtain the values of ε the boundary condition eq. 2.4 is used for the nodes in the mesh closer to the airfoil.

The inlet profiles are written to one file for both the inlets. But the data is obtained from RANS simulations that have a wall at the bottom. In the airfoil simulation the wall i.e. the airfoil is located at the bottom for the top profile but the airfoil is on the top of the bottom profile. To represent this for the bottom profile the v velocity profile has to be mirrored.

2.2.4 RANS

In this validating case the RANS $k-\varepsilon$ turbulence model is used to predict the behavior of the flow. The discretization scheme used is the same scheme as used in the simulations of the flat plate validating cases i.e. the hybrid scheme. The u/u_∞ and u_{rms} profiles are available from the experiments performed by Blake in [7] to validate the implementation.

The predicted u/u_∞ velocity profiles are compared with measurements at locations x/H = -3.125, -2.125, -1.625, -1.125 and -0.625 see fig. 2.15. Here it is seen that the velocity profiles all fulfills the no slip condition at the surface of the airfoil. All the profiles from the experiments have similar shapes as those from the simulation, but their magnitudes are different. The focus in this studied was the implementation to be able to run this case setup and get converging results. No case setup that is more simplified than this is available to validate the implementation without changing the physics of the simulation. If simulations should be run to obtain results that is in agree with the experiments by Blake it is recommended to do a mesh study and to change the

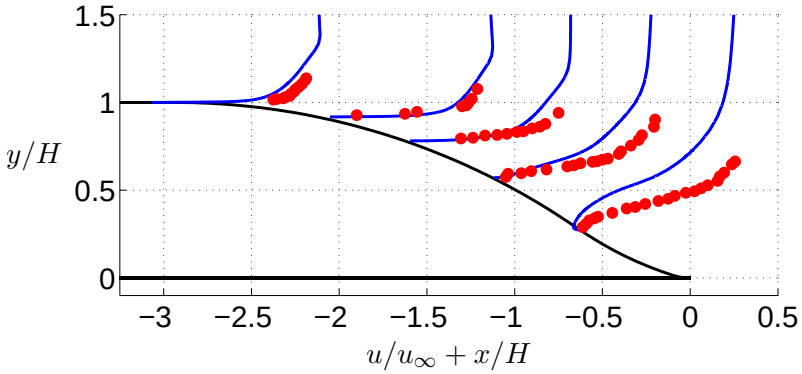


Figure 2.15: Comparison of the u profiles obtain by the simulation and the experiments by Blake at locations, x/H = -3.125, -2.125, -1.625, -1.125 and -0.625. ——: Drawn airfoil, ——: Simulation, ● : Experiments by Blake.

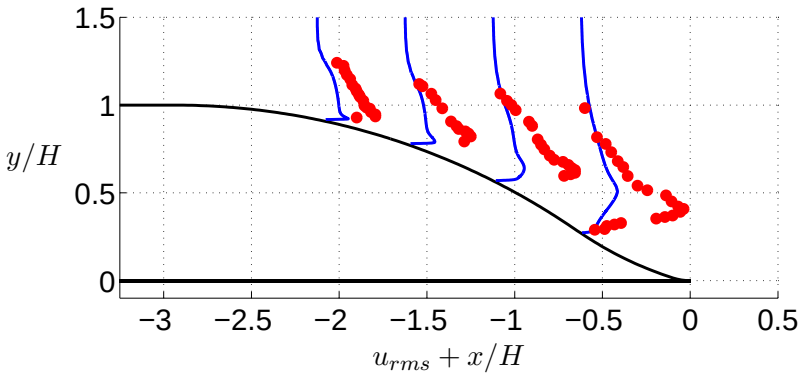


Figure 2.16: Comparison of the $u_{rms} + x/H$ profiles obtain by the simulation and the experiments by Blake at locations, x/H = -2.125, -1.625, -1.125 and -0.625. ——: Drawn airfoil, ——: Simulation, ● : Experiments by Blake.

turbulence model. Nothing of this has been done in this study and it is recommended to read [8, 9, 10, 11] where the results in agreement with the experiments is the main focus, if this is the goal of the study by the reader. In this study the simulation converged and the profiles in fig. 2.15 show that the no slip condition is fulfilled is an acceptable result.

The profiles of u_{rms} are compared from the experiments by Blake and from this simulation at the locations x/H = -2.125, -1.625, -1.125 and -0.625 in fig. 2.16. In the same way as for the u/u_∞ profiles these profiles also show that the no slip condition is fulfilled at the surface of the airfoil. The u_{rms} profiles obtained from the simulation and the experiments show similar behavior but their magnitudes are different in the same way as for the u/u_∞ profiles in fig. 2.15.

2.2.5 Conclusions of airfoil case

Both the comparisons of u/u_∞ and u_{rms} show that the no slip condition is fulfilled at the surface of the airfoil. The simulation itself was converging which is important to show that the implementation in CALC is made in a correct way. So the focus of this study to implement this case setup in CALC is seen as completed.

3

Actuator turbine models

In wind turbine CFD simulations actuator turbine models are often used in favour of solving the flow field over each blade. The main advantage of this is that the boundary layer on each blade does not need to be resolved so the mesh resolution can be reduced significantly. [15]

In actuator turbine models the wind turbine is simulated by drag forces. But the exact representation of the blades could vary in numerous ways e.g. a disk in Actuator Disk Model (ADM) and lines in an Actuator Line Model (ALM). The physical representation of ADM and ALM can be seen in fig. 3.1. In this work an ADM and an ALM were implemented that both apply thrust and tangential force.

3.1 Actuator disk model

Due to that the ADM simulates the wind turbine blades as a disk the flow will be axisymmetric. The ADM is based on the Blade Element Momentum Method (BEM) which calculates the forces on the blades by 2D airfoil profiles. The lift and drag force calculations for the ADM can be seen below.

$$f_{ADM} = \frac{\partial F}{\partial A} = (L_{ADM}, D_{ADM}) = \frac{B}{2\pi r}\frac{1}{2}\rho V_{rel}^2 c(C_L e_L, C_D e_D) \tag{3.1}$$

In ADM the lift and drag forces, L_{ADM} and D_{ADM} are per unit area as can be seen in eq. 3.1. An example of an area section where the force from an ADM routine could be applied on can be seen in fig. 3.2. In eq. 3.1 B represents the number of blades on the wind turbine and r represents the local radius to the center location on the area section. The rest of eq. 3.1 is an ordinary lift and drag force formula for an airfoil where V_{rel} is the local relative velocity and c represents the local chord length. C_L and C_D are the local lift and drag coefficients and e_L and e_D are the unit vectors in the directions of the lift and drag. Further information on the steps from BEM to ADM can be found in [16].

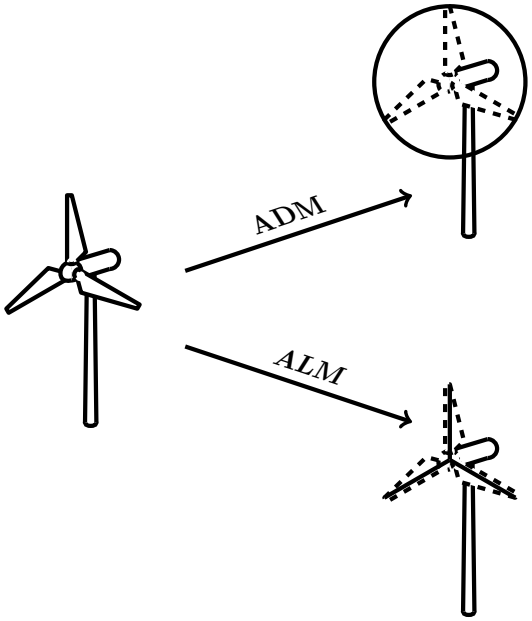


Figure 3.1: Wind turbine to ADM and ALM.

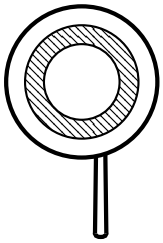


Figure 3.2: Area sections of ADM.

3.1.1 Gaussian function

The forces obtained from the ADM are in the axial plane of the wind turbine. Applying these forces to the Computational Fluid Dynamics (CFD) simulation as a point force will lead to oscillations [1]. To avoid these oscillations the force could be smeared away from the single point in numerous ways. Here the force will be smeared using a Gaussian distribution. For the ADM a 1D Gaussian distribution smearing in the axial direction is recommended [16] and can be seen in fig. 3.3.

$$\eta(p^{1D})_{\varepsilon}^{1D} = \frac{1}{\varepsilon^{1D}\sqrt{\pi}} e^{-\left(\frac{p^{1D}}{\varepsilon^{1D}}\right)^2} \tag{3.2}$$

In eq. 3.2 the 1D Gaussian distribution used in this implementation can be seen. p^{1D} is the axial distance from the point evaluated to the wind turbine. The variable ε^{1D} is a function of ε_i and the mesh resolution along the axial direction Δx.

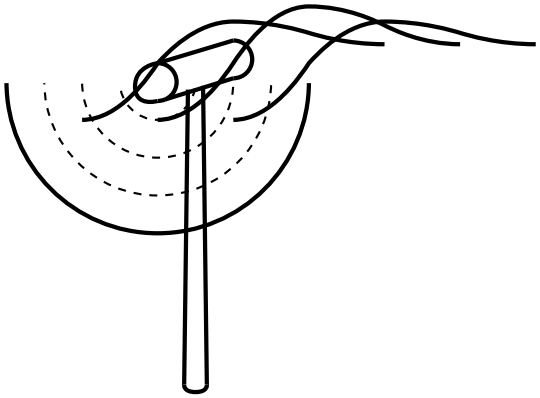


Figure 3.3: 1D gaussian applied.

$$\varepsilon^{1D} = \varepsilon_i \Delta x \tag{3.3}$$

In this 1D smearing case ε^{1D} is described by eq. 3.3. ε_i is recommended to be in the range from 1 to 4 [16] and Δx is the local axial mesh size.

3.1.2 Implementation in CALC

The implementation of ADM in CALC is first visible in the mod.f file where the part with ADM implementation code can be seen in appendix J. Variables that will be constant during the whole simulation are first initialized e.g. the number of nodes radially on the blades. In this section the location of the wind turbine in the mesh is also found.

The velocity in ADM is represented in two dimensions by the axial and the tangential velocity. The CALC code uses Cartesian coordinates with the x axis being the axial direction and the z axis pointing upwards and the y axis specified from the two others. The axial velocity in the ADM is the same as the Cartesian u component of the velocity in CALC. The tangential velocity is obtained from

$$u_{tang} = -vsin(\theta) + wcos(\theta) \tag{3.4}$$

Here u_{tang} is the tangential velocity in ADM and θ is the polar angle coordinate which needs to be found for each cell of the mesh in the y-z plane. This is needed because CALC cannot handle a 2D axisymmetric mesh. When the mesh is axisymmetric it needs a y-z plane formed like a regular polygon due to the multigrid solver for the pressure. A representation of such a mesh can be seen in fig. 3.4. This is followed by reading the dimensions of the blades from the aeroData.inp file followed by the lift, drag and momentum coefficients from the CLCDCMTable.inp file.

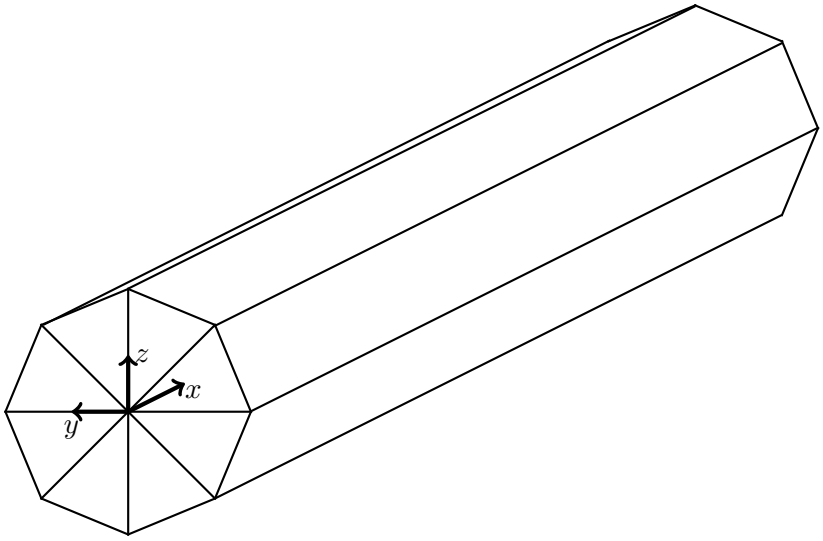


Figure 3.4: The mesh used in ADM and ALM.

In each time step the axial and tangential velocities are averaged over all the cells on the same radii from the center of the wind turbine rotor. This is followed by sending the variables to the force subroutine seen in appendix K. It is first checked if the cell is located on the blade outside of the hub radius. If not only a drag force is applied and returned to the mod.f file. If the cell is located outside of the hub radii the flow angle, the angle of attack and the Mach number of the flow are calculated and by the help of subroutines aeroTable and foilclcdcm the lift and drag coefficients are found. These coefficients are then used together with the flow angle to obtain the coefficients in the normal direction referred to as the axial direction in the mod.f file and the tangential direction. Last in the force routine the normal and the tangential forces are calculated. Here eq. 3.1 is used but in the normal and the tangential directions instead of the lift and drag directions. The reason is that the normal and the tangential velocities are supplied to the force subroutine.

When returning to the mod.f file the normal force (from here on called the axial force) and the tangential force are converted from Newton per meter squared on a circular area to a regular polynomial area due to the mesh. The drag force in the axial and tangential directions will be applied to the momentum equations as source terms. To do this the unit needs to be Newton and not Newton per area section. So the forces are multiplied by the area in the y-z plane of the cell. Next the tangential force is projected to a y and z component and added as source terms in the V and W momentum equations.

The smearing of the force takes place in the mod.f file. Here the Gaussian distribution eq. 3.2 and the ε^{1D} eq. 3.3 equations are used. In the end of mod.f the smeared force is added to the Su source term. To smear the force axial in the computational domain the variable $imiddiff$ is used. It represents the number of cells in the axial direction upstream and downstream of the wind turbine where the smeared force will be added to

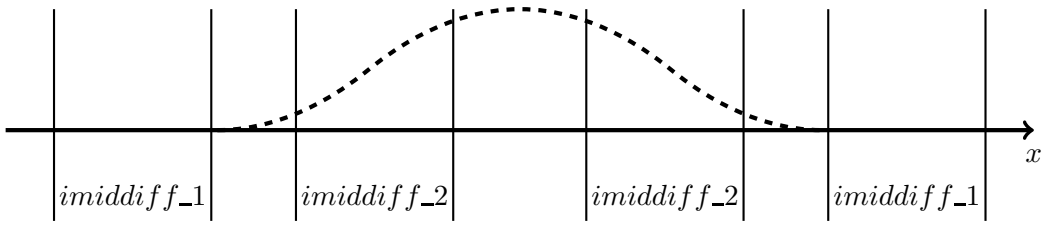


Figure 3.5: Choice of *imiddiff* parameter. Here *imiddiff_1* has a good value but *imiddiff_2* has a value which is too low. The Gaussian function described by eq. 3.2 is shown by the dashed line.

the Su source term. Depending on the magnitude of the force and the parameters of the Gaussian distribution the smeared force will be wider or narrower. When first starting a new setup of the simulation it is important to make sure that the contribution to the source term in the start and end cell is close to zero. Because if it is not this could cause oscillations in the simulation. In fig. 3.5 two dummy variables are shown representing the values of the *imiddiff* variable. *imiddiff_1* has a larger value than *imiddiff_2*. This means that the *imiddiff_1* value choice will start applying the smeared force at a cell located further away from the wind turbine. The Gaussian function in fig. 3.5 has a value close to zero at this location. So *imiddiff_1* is a good choice of value for the *imiddiff* variable. However consider the *imiddiff_2* value of the *imiddiff* variable. It is here seen that the Gaussian function is not close to zero. So a choice representing *imiddiff_2* is not recommended.

Before the force is added to the momentum equations it is smeared with the Gaussian function. The Gaussian is integrated for each radial position along the axial axis. Then when the Gaussian is used to smear the force this integrated Gaussian is used to scale so that the final contribution of the Gaussian is equal to one. Here the force is added to the source term of the u velocity, but the same follows for the v and w velocity by changing the applied force to the y and z forces.

3.1.3 Validation

To validate the ADM implementation the 5-MW reference wind turbine from National Renewable Energy Laboratory (NREL) is used [17]. This wind turbine is used due to the amount of available data e.g. velocity profiles at velocity specific rotational speed of the rotor and pitch angle of the blades. The RANS k-ε turbulence model was used.

This first setup describes a uniformed wind blowing on the rotor blades and the hub and can be seen in fig. 3.6. This representation allows the simulation to be run using axis symmetry. The axisymmetric computational domain can be seen in fig. 3.7 expressed in units of rotor radii. The flow is simulated 10 rotor radii before the wind turbine and 20 after. Radially the flow is simulated 9 rotor radii outside of the wind turbine. Radially the mesh has 240 cells.

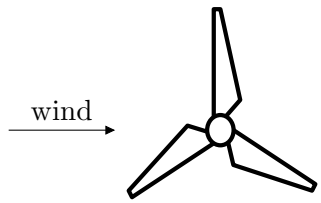


Figure 3.6: First setup.

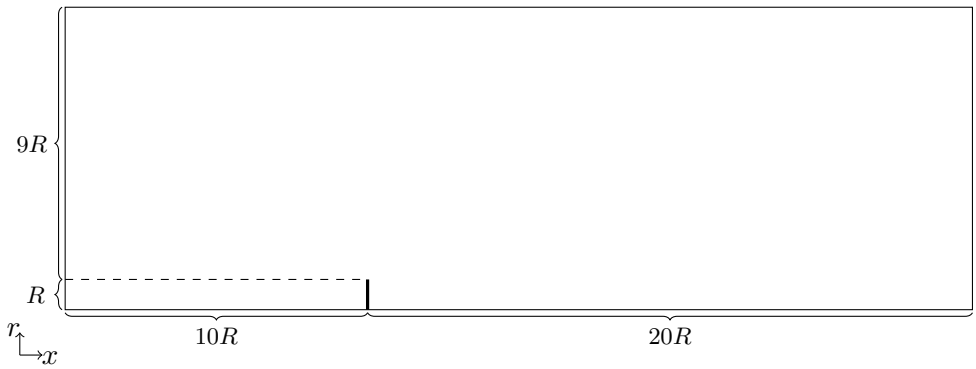


Figure 3.7: Axisymmetric computational domain in units rotor radii. The location of the wind turbine is seen as the thick line inside the computational domain.

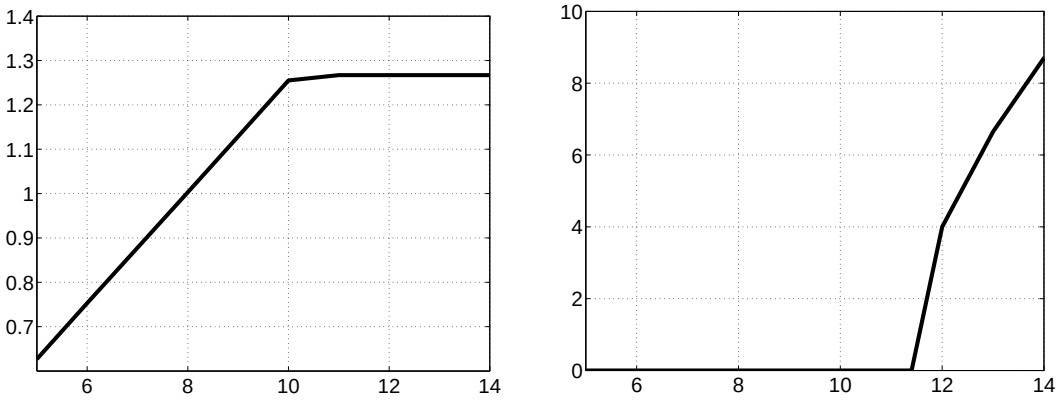
Velocity profiles

From the NREL documentation the velocity profiles can be found. The rotational speed seen in fig. 3.8(a) and the blade pitch angle shown in fig. 3.8(b) were taken from this technical report to be able to compare the behaviour of the ADM implementation for different wind velocities [17]. In the velocity profile simulations the mesh had 160 cells in the axial direction distributed uniformly, which corresponds to a Δx value of 11.81 m.

In fig. 3.9 the predicted rotor power and rotor thrust can be seen as functions of the wind speed. The NREL data using the BEM model is seen in red and the ADM simulation using the RANS turbulence model is seen in blue. The shape of the ADM curves are close to the NREL shape but have a distance in between them. At wind speeds larger than 11.4 m/s the blades start to pitch as seen in fig. 3.8(b) and the influence of this on the rotor power and thrust is also captured by the ADM implementation seen in fig. 3.9. The results from the ADM is not expected to give the exactly same results as the NREL data because different models have been used. Rather it should have the main shape equal to the NREL data which has been shown here.

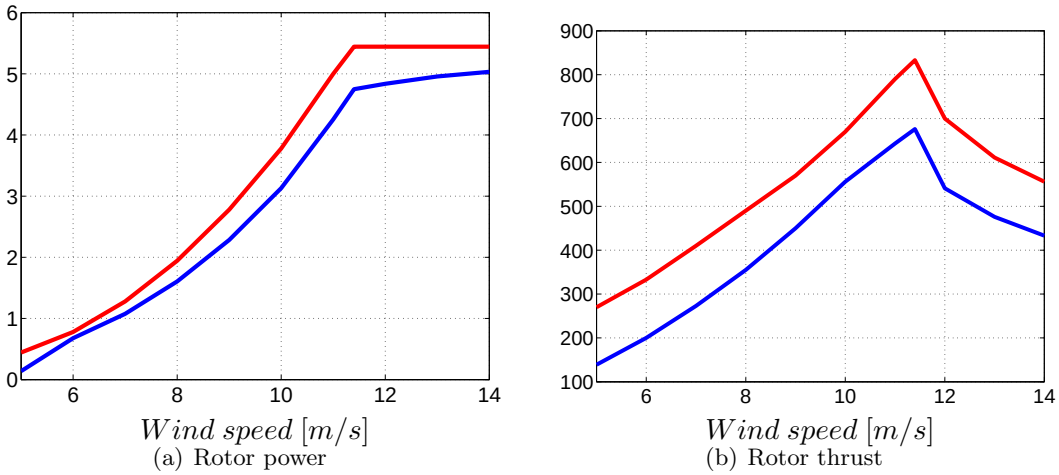
ε parameter

The ε and Δ parameters have been shown to have an impact on the results from the CFD simulation [1, 16]. Simulations when varying the ε and Δ parameters were performed to



(a) Rotational speed (b) Pitch angle

Figure 3.8: Rotational speed $[rad/s]$ and pitch angle on the blade [°] as function of the wind speed from the NREL data [17].



(a) Rotor power (b) Rotor thrust

Figure 3.9: Rotor power $[MW]$ and thrust $[kN]$ as function of the wind speed. ——: NREL BEM, ——: ADM RANS.

investigate the impact on this implementation of ADM. The uniformly inflow was chosen to be 8 m/s and the rotational speed was 1.003 rad/s and zero pitch angle. Here the 1D ε parameter described by eq. 3.3 and the axial mesh size named Δx were varied.

The input used in this simulation was quite close to the input used in [1]. The same wind turbine was used, the inflow velocity was also 8 m/s, the rotational speed was a bit lower 0.959 rad/s. They used a 3D Gaussian distribution which should give slightly different results [1, 16]. The software in [1] was OpenFOAM using LES with the Smagorinsky subgrid model which should provide more accurate results compared to the RANS k-ε model used here but the magnitude of the difference is unknown [5].

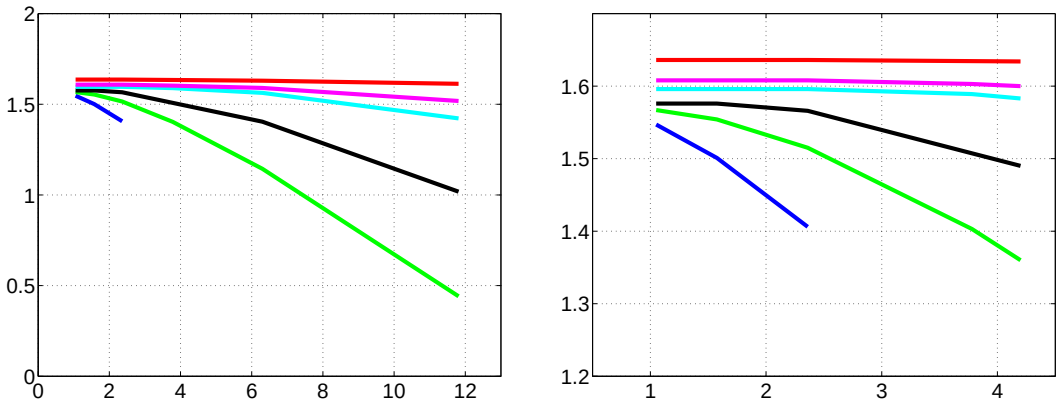
The rotor power results from the simulations performed in this master thesis can be seen in fig. 3.10. The span of data used for the parameters was larger in these simulations than in the simulations made by [1] seen in fig. 3.11. All the data from the present simulations are shown in fig. 3.10(a) and data from a similar span as in [1] can be seen in fig. 3.10(b). The reason that data for the lowest ε value of 2.36 is not presented for meshes with Δx larger than 2.36 is the large oscillations introduce that caused the simulations to diverge. The result from the simulations in [1] show an overall larger amplitude of the results from minimum 1.95 MW with $\varepsilon = 4.2$ m and uniform $\Delta = 4.25$ m to a maximum of 2.25 MW with $\varepsilon = 10.5$ m and uniform $\Delta = 1.1$ m [1]. Comparing to approximately the same parameter values here giving 1.35 MW for $\varepsilon = 3.78$ m and $\Delta x = 4.2$ m to 1.59 MW for $\varepsilon = 11.81$ m and $\Delta x = 1.05$ m. Giving a difference in their simulations of 0.3 MW and in this simulations 0.25 MW. So the difference is more or less the same but it is unknown if the difference is due to the different Gaussian functions or turbulence models used or something else.

The main behaviour of the rotor power when the parameters are changed is the same between the simulations. Keeping ε constant and increasing Δ will decrease the rotor power predicted. Whereas keeping Δ constant and increasing ε will increase the rotor power. The power of the choice of Δ and ε can be seen in fig. 3.10(a). Here the predicted rotor power varies between 0.48 MW and 1.63 MW.

Data for the rotor thrust is also presented in fig. 3.12 in the same way as for the rotor power. The thrust shows the same behaviour as the power keeping ε constant and increasing Δ decreases the predicted rotor thrust. Whereas keeping Δ constant and increasing ε increases the rotor thrust. Fig. 3.12(a) shows the power for different Δ and ε and the trust predictions vary from 200 kN to 360 kN. No comparing data for the rotor thrust are available in [1].

To limit these differences in predicted rotor power and thrust it is in [16] recommended to limit the ε_i parameter in eq. 3.3 between 1 and 4. This will limit the ε^{1D} parameter differently for different choices of Δx i.e. meshes. Showing this in fig. 3.10 and fig. 3.12 proved difficult so the data was rearranged and shown as constant Δx profiles in fig. 3.13 instead of constant ε^{1D} profiles. The part of the profiles in fig. 3.13 that are solid shows a ε_i value between 1 and 4 and the dashed part has values outside of this interval.

It is now clearly seen the ε^{1D} value varies a lot from the mesh with the smallest Δx to the one with the largest Δx. But the rotor power and thrust is for all the



(a) All data (b) Zoomed

Figure 3.10: Rotor power $[MW]$ as function of Δx $[m]$. ——: $\varepsilon^{1D} = 2.36$ m, ——: $\varepsilon^{1D} = 3.78$ m, ——: $\varepsilon^{1D} = 6.30$ m, ——: $\varepsilon^{1D} = 11.81$ m, ——: $\varepsilon^{1D} = 15.12$ m, ——: $\varepsilon^{1D} = 25.20$ m.

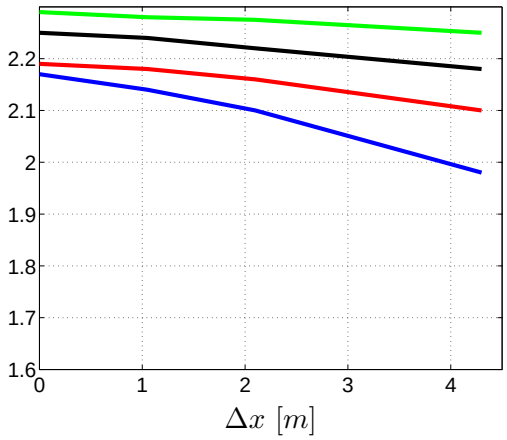
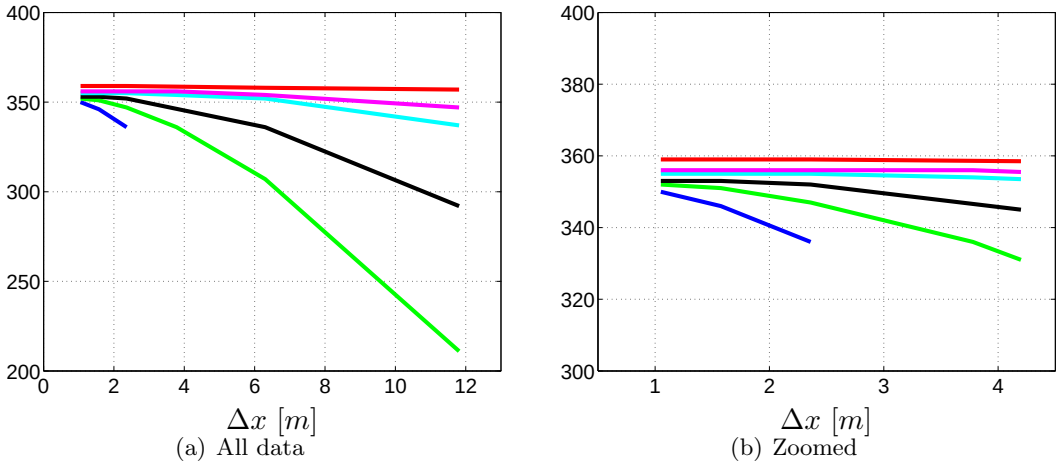
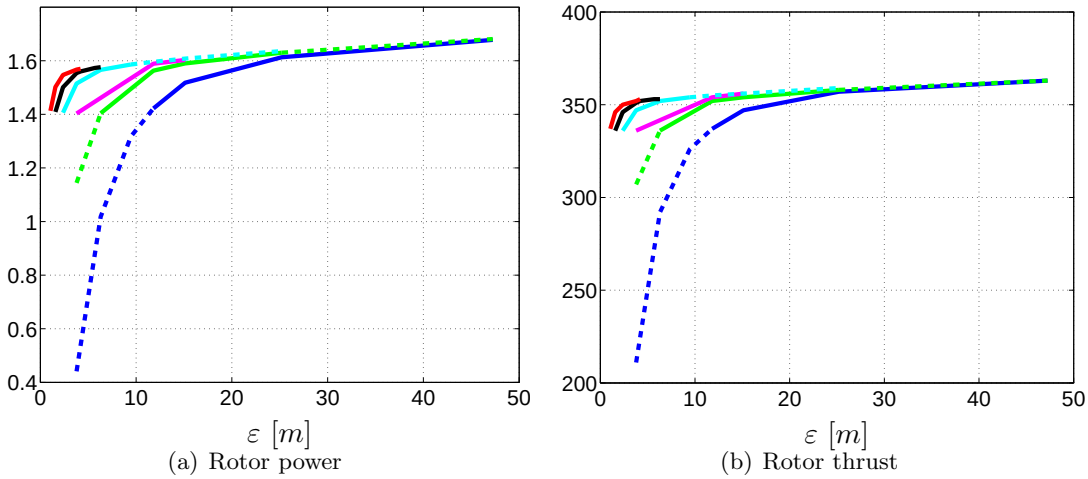


Figure 3.11: Rotor power $[MW]$ as function of the Δx $[m]$, data obtained from [1]. ——: $\varepsilon = 4.2$ m, ——: $\varepsilon = 6.3$ m, ——: $\varepsilon = 8.4$ m, ——: $\varepsilon = 10.5$ m.



(a) All data (b) Zoomed

Figure 3.12: Rotor thrust $[kN]$ as function of the Δx $[m]$. ——: $\varepsilon^{1D} = 2.36$ m, ——: $\varepsilon^{1D} = 3.78$ m, ——: $\varepsilon^{1D} = 6.30$ m, ——: $\varepsilon^{1D} = 11.81$ m, ——: $\varepsilon^{1D} = 15.12$ m, ——: $\varepsilon^{1D} = 25.20$ m.



(a) Rotor power (b) Rotor thrust

Figure 3.13: Rotor power $[MW]$ and thrust $[kN]$ as function of the ε $[m]$. ——: $\Delta x = 1.050$ m, ——: $\Delta x = 1.575$ m, ——: $\Delta x = 2.36$ m, ——: $\Delta x = 3.78$ m, ——: $\Delta x = 6.30$ m, ——: $\Delta x = 11.81$. m

Δx values much more restricted for the recommended ε_i values. The interval of the smallest predicted rotor power is now 1.4 MW to the largest of 1.63 MW. This shows a difference of 0.23 MW compared to 1.15 MW for the non restricted ε_i simulations. In the rotor thrust the smallest predicted value is now 330 kN and the largest 360 kN. So the difference is now 30 kN compared to 160 kN for the non restricted ε_i simulations.

So to be able to have more consistent predictions of the rotor power and thrust

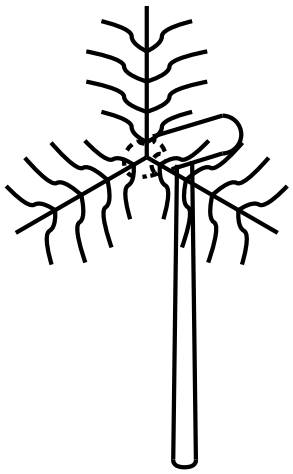


Figure 3.14: 2D Gaussian smears forces both in the axial direction in the same way as the 1D Gaussian seen in fig. 3.3 but now also in the rotor plane.

between different meshes i.e. different Δ values it is more important to keep track of the ε_i parameter than the ε^{1D} parameter in eq. 3.3.

3.2 Actuator line model

In ALM the blades are simulated as lines. This representation makes it not possible to run the simulation axisymmetric as in ADM now a 3D simulation is needed.

$$f_{ALM} = \frac{\partial F}{\partial L} = (L_{ALM}, D_{ALM}) = \frac{1}{2}\rho V_{rel}^2 c(C_L e_L, C_D e_D) \tag{3.5}$$

The ALM formula for lift and drag can be seen in eq. 3.5, as L_{ALM} and D_{ALM}. Compared to the ADM the ALM formula has the unit force per length instead of force per area. ALM can capture the tip and root vortices which ADM cannot. To have a representation of these vortices is important when studying the near wake which is a clear advantage of the ALM over the ADM. The advantage of ADM compared to ALM is the speed of convergence if the near wake is not to be studied. [1]

3.2.1 Gaussian function

The force obtained from the ALM is along a line. The 2D Gaussian distribution recommended from [16] will not only smear the force in the axial direction but also in the tangential direction following a constant radii in the rotor plane. Hence the force will not only be applied as in fig. 3.3 but also in the rotor plane (seen in fig. 3.14).

$$\eta(p^{2D})_{\varepsilon}^{3D} = \frac{1}{(\varepsilon^{3D})^2\pi} e^{-\left(\frac{p^{2D}}{\varepsilon^{3D}}\right)^2} \tag{3.6}$$

The 2D Gaussian distribution can be seen in eq. 3.6. The variable p^{2D} is the distance from the evaluated point to the node on the line with the same radius. The ε^{1D} from ADM is now not only a variable depending on the axial mesh resolution but the whole 3D mesh resolution and is hence denoted by ε^{3D}.

$$\varepsilon^{3D} = \varepsilon_i \sqrt{(R\Delta\theta)^2 \Delta r^2 \Delta x^2} \tag{3.7}$$

Eq. 3.7 shows the formula for ε^{3D} recommended and used here [16]. The variable R is the rotor radius, $\Delta\theta$, Δr and Δx are the local tangential, radial and axial mesh size respectively.

3.2.2 Implementation in CALC

The implementation of ALM in CALC has many similarities to the implementation of ADM. This section will only present changes and further implementations needed for the ALM compared to the ADM.

To run ALM compared to ADM the location of the blades i.e. the lines need to be traced. This is done in the mod.f file seen in appendix L after the airfoil data has been read. The first thing that needs to be done is to trace the angle the blades has turned within one lap to the current k plane. This is done by the variable $Theata2k$ which maps angles to k planes. Finally the angle between the blades are calculated and defined by the $AngBlade$.

Each time step the location of the blades needs to be found. To do this the time step dt, the total time of the simulation at this current time step $tottime$ and the rotational speed $omega$ are used. First the number of whole laps the blades has turned is found. The radians this corresponds to are then subtracted from the total radians turned to find the radians turned within the lap. Then $Theata2k$ and $AngBlade$ variables are used to find the k plane location of each blade and store it in the $Bladek$ variable.

For each blade the local axial and tangential velocity are now found and used as input in the force subroutine for ALM seen in appendix M. The only difference between the force subroutine in ADM and ALM is the two formulas for calculating the normal and tangential force at the end of the routine. In ALM the formulas are given in eq. 3.5.

Like in ADM the ALM forces must have the unit Newton. This is obtained by multiplying the forces obtained from the force subroutine by the radial mesh size of the cells seen in fig. 3.15. The ε^{3D} will vary in the mesh because it includes Δr. So to be able to compare the constant ε^{1D} in ADM to the varying ε^{3D} in ALM the ε^{3D} values are averaged. Observe that this is only for comparison reasons and the mesh local value of ε^{3D} is used in the simulation.

The 2D Gaussian distribution is integrated for each radial position over the axial and the tangential axis. Like in the ADM the integral is scaled so that it is equal to one. This is the definition of a Gaussian, but without scaling the integrated 2D Gaussian is much larger than one. The force from each blade is then added to the source term individually after being smeared. This is done in the mod.f file in appendix L and here eq. 3.6 and eq. 3.7 are used.

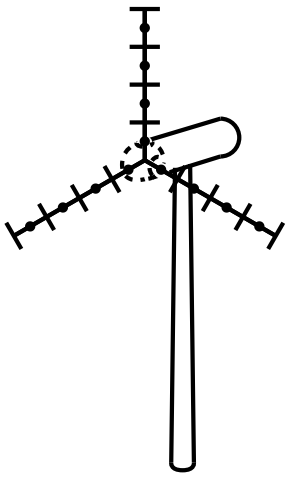


Figure 3.15: 2D gaussian in addition to 1D gaussian also apply forces in the rotor plane

3.2.3 Validation

The ALM implementation is validated in this section. Both the Unsteady Reynolds Averaged Navier Stokes (URANS) and the Partially Averaged Navier Stokes (PANS) turbulence models are used.

Velocity profiles

To validate the ALM implementation the same wind turbine as used in the ADM validation, the 5-MW reference wind turbine from NREL is used [17]. This first setup describes a uniformed wind blowing on the rotor blades and the hub and can be seen in fig. 3.6. This setup allowed the ADM simulation to be carried out using axis symmetry but in the ALM the lines are rotating so the simulation now needs to be run in 3D. The computational domain was the same as in ADM and can be seen in fig. 3.7. The wind speeds were the same as in the ADM simulations. The rotational speed and the blade pitch angle versus wind speed are shown in fig. 3.8.

When the PANS turbulence model is used the LES region is specified as the black region in the computational domain in fig. 3.16 and the URANS region is white. The transitions between the regions take place over a distance of two rotor radius.

In fig. 3.17 the predicted rotor power and rotor thrust are presented as functions of the wind speed. The NREL data using the BEM model is included in red and the ADM simulation using the RANS turbulences model is seen in blue. The ALM simulations using URANS and PANS are also shown. When predicting the rotor thrust ALM predicts somewhat higher values than ADM but not as high as the NREL data. Comparing the ALM simulations to each other the values predicted by the PANS gives slightly lower rotor power and thrust than the URANS simulation. Here the ALM shows values in closer agreement to the NREL data than the ADM results.

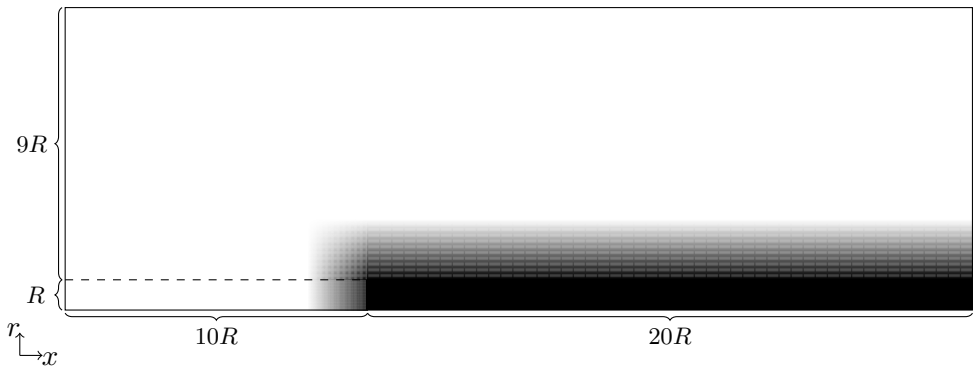
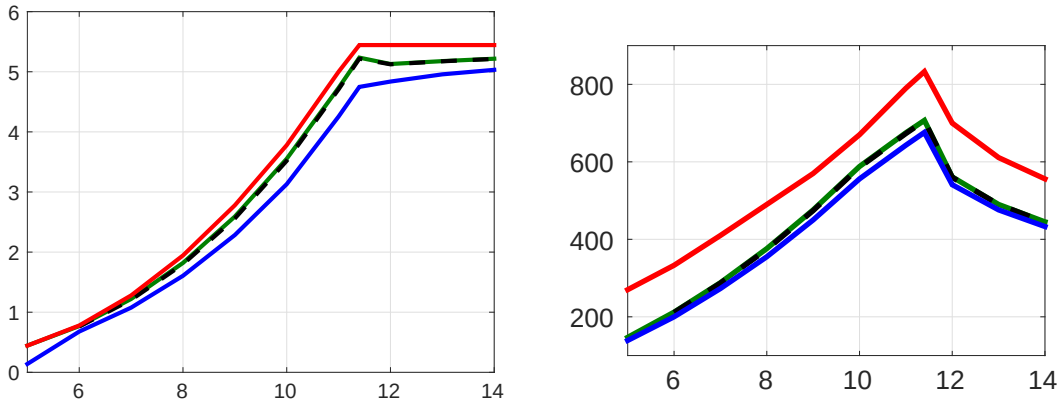


Figure 3.16: The LES region seen in black and URANS region in white. They are shown in the axisymmetric computational domain in units rotor radius.



(a) Rotor power (b) Rotor thrust

Figure 3.17: Rotor power $[MW]$ and thrust $[kN]$ as function of the wind speed. —: NREL BEM, —: ADM RANS, —: ALM URANS, —: ALM PANS.

3.3 Conclusions

When comparing the ADM and ALM velocity profiles of the rotor power and the rotor thrust to the NREL data the main shape is the same. This was the focus due to NREL uses the BEM model so no actuator turbine model when predicting the quantities. Both the 1D in ADM and 2D in ALM Gaussian distributions have been normalized so that the integral is equal to one.

When varying the values of the ε and the Δx parameters in the ADM simulation and comparing the predicted rotor power to [1] the difference between the maximum and minimum were more or less the same. But it is unknown if the small difference is due to the different Gaussian distributions or turbulence models or something else. No comparing data for the rotor thrust are available in [1].

To have more consistent predictions of the rotor power and the rotor thrust in ADM between different meshes i.e. different Δx values it is more important to keep track of the ε_i parameter than the ε^{1D} parameter in eq. 3.3.

4

Further work

In the trailing edge flow in the airfoil case a mesh refinement study needs to be done. To obtain velocity profiles that agree with the experiments by Blake to a larger extent than the ones presented here it is also recommended to change the turbulence model.

In the ALM simulation the rotor power and rotor thrust dependence on the mesh resolution and the ε^{2D} is still needed to be found.

Bibliography

[1] L. M. Tossas, S. Leonardi, Wind turbine modeling of computational fluid dynamics, Tech. rep., National Renewable Energy Laboratory, University of Puerto Rico, Mayaguez, Puerto Rico (2013).

[2] H. K. Versteeg, W. Malalasekra, An introduction to computional fluid dynamics and finite volume method, 2nd Edition, Pearson Education Limited, Harlow, Great Britian, 2007.

[3] L. Davidson, B. Farhanieh, Calc-bfc a finte-volume code employing collocated varible arrangement and cartesian velocity components for computation of fluid flow and heat transfer in complex three-dimensional geometrist, Publication no 95/11, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden (1995).

[4] P. Johansson, A three dimensional laminar multigrid method applied to the simplec algorithm, Publication no 92/11, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden (1992).

[5] L. Davidson, Fluid mechanics, turbulent flow and turbulence modeling, 23 dec, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden (2014).

[6] L. Davidson, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden (2014).

[7] W. Blake, A statistical description of pressure and velocity fields at the trailing edges of a flat strut, 4241, David W. Taylor Naval ship research and development center, Bethesda, USA (1975).

[8] M. Wang, Progress in large-eddy simulation of trailing-edge turbulence and aeroacoustics, in: Annual Research Briefs, Center for Turbulent Research, Stanford Univ./NASA Ames Research Center, 1997, pp. 37–49.

[9] M. Wang, P. Moin, Dynamic wall modelling for large eddy simulation of complex turbulent flows, Physics of Fluids 14 (7) (2002) 2043–2051.

[10] M. Wang, P. Moin, Computation of trailing-edge flow and noise using large-eddy simulation, AIAA Journal 38 (12) (2000) 2201–2209.

[11] M. Gritskevich, A. Garbaruk, J. Schütze, F. Menter, Development of ddes and iddes formulations for the k-ω shear stress transport model, Flow, Turbulence and Combustion 88 (2012) 431–449.

[12] Y. Bentaleb, S. Lardeau, M. A. Leschziner, Large-eddy simulation of turbulent boundary layer separation from a rounded step, Journal of Turbulence 13 (4) (2012) 1–28.

[13] D. Apsley, Turbulent boundary layers, Tech. rep., The university of Manchester, Manchester, Great Britain (2009).

[14] L. Davidson, An introduction to turbulence models, Publication no 97/2, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden (2011).

[15] N. Troldborg, Actuator line modeling of wind tubine wakes, Ph.D. thesis, Dept. of Mechanical Engineering, Technical University of Denmark, Lyngby, Denmark (2008).

[16] R. Mikkelsen, Actuator disk methods applied to wind turbines, Ph.D. thesis, Dept. of Mechanical Engineering, Technical University of Denmark, Lyngby, Denmark (2008).

[17] W. M. J. Jonkman, S. Butterfield, G. Scott, Definition of a 5-mw reference wind turbine for offshore system development, Tech. rep., National Renewable Energy Laboratory, Golden, Colorado (2009).

A

CALC part of init.f file including modifications

```
      subroutine init

chapter 0 0 0 0 0 0 0 0 preliminaries 0 0 0 0 0 0
0

      include 'COMMON'
      include 'CASECOM'

      dimension xcmod(it,jt,kt),ycmod(it,jt,kt),zcmod(it,jt,kt)

chapter 1 1 1 1 1 1 geometrical quantities 1 1 1 1 1
1

c-------calculate the nodes of the control volume.

      pi=4.*atan(1.)
      do 100 kk= 1,nk
      do 100 jj= 1,nj
      do 100 ii= 1,ni

         im1=max(ii-1,1)
         jm1=max(jj-1,1)
         km1=max(kk-1,1)

         i=min(ii,nim1)
```

```
          j=min(jj,njm1)
          k=min(kk,nkm1)

          xp(ii,jj,kk)=
     &    0.125*(xc(i,j,k)+xc(im1,j,k)+xc(i,jm1,k)+xc(im1,jm1,k)
     &       +xc(i,j,km1)+xc(im1,j,km1)+xc(i,jm1,km1)+xc(im1,jm1,km1))
          yp(ii,jj,kk)=
     &    0.125*(yc(i,j,k)+yc(im1,j,k)+yc(i,jm1,k)+yc(im1,jm1,k)
     &       +yc(i,j,km1)+yc(im1,j,km1)+yc(i,jm1,km1)+yc(im1,jm1,km1))
          zp(ii,jj,kk)=
     &    0.125*(zc(i,j,k)+zc(im1,j,k)+zc(i,jm1,k)+zc(im1,jm1,k)
     &       +zc(i,j,km1)+zc(im1,j,km1)+zc(i,jm1,km1)+zc(im1,jm1,km1))

c Modified by Matsfelt
c Modify the yp for nodes in the airfoil
      if (ii.le.iaf.and.jj.eq.jaf) then

          yp(ii,jj,kk)=
     &    0.125*(yc(iaf,j,k)+yc(iaf,j,k)+yc(i,jm1,k)+yc(im1,jm1,k)
     &     +yc(iaf,j,km1)+yc(iaf,j,km1)+yc(i,jm1,km1)+yc(im1,jm1,km1))

      end if
c Modified by Matsfelt

 100  continue

c-------calculate the area of the control volume faces
      do 110 k= 1,nkm1
      do 110 j= 1,njm1
      do 110 i= 1,nim1

          im1=max(1,i-1)
          jm1=max(1,j-1)
          km1=max(1,k-1)
c-----------
          ax=xc(i,j,k)-xc(im1,j,k) !
          ay=yc(i,j,k)-yc(im1,j,k) !
          az=zc(i,j,k)-zc(im1,j,k) !
c-----------
          bx=xc(i,j,k)-xc(i,j,km1) !
          by=yc(i,j,k)-yc(i,j,km1) !
          bz=zc(i,j,k)-zc(i,j,km1) !
c-----------
```

```
            cx=xc(i,j,k)-xc(i,jm1,k) !
            cy=yc(i,j,k)-yc(i,jm1,k) !
            cz=zc(i,j,k)-zc(i,jm1,k) !
c-----------
            dx=xc(i,jm1,km1)-xc(i,jm1,k) !
            dy=yc(i,jm1,km1)-yc(i,jm1,k) !
            dz=zc(i,jm1,km1)-zc(i,jm1,k) !
c-----------
            ex=xc(i,jm1,km1)-xc(i,j,km1) !
            ey=yc(i,jm1,km1)-yc(i,j,km1) !
            ez=zc(i,jm1,km1)-zc(i,j,km1) !
c-----------
            ox=xc(im1,j,km1)-xc(im1,j,k) !
            oy=yc(im1,j,km1)-yc(im1,j,k) !
            oz=zc(im1,j,km1)-zc(im1,j,k) !
c-----------
            px=xc(im1,jm1,k)-xc(i,jm1,k) !
            py=yc(im1,jm1,k)-yc(i,jm1,k) !
            pz=zc(im1,jm1,k)-zc(i,jm1,k) !
c-----------
            qx=xc(im1,jm1,k)-xc(im1,j,k) !
            qy=yc(im1,jm1,k)-yc(im1,j,k) !
            qz=zc(im1,jm1,k)-zc(im1,j,k) !
c-----------
            rx=xc(im1,j,km1)-xc(i,j,km1) !
            ry=yc(im1,j,km1)-yc(i,j,km1) !
            rz=zc(im1,j,km1)-zc(i,j,km1) !

c Modified by Matsfelt
c Modify the yc at jaf level for the  locations in the airfoil
          if (i.le.iaf.and.j.eq.jaf) then


            ay=yc(iaf,j,k)-yc(iaf,j,k) !
            by=yc(iaf,j,k)-yc(iaf,j,km1) !
            cy=yc(iaf,j,k)-yc(i,jm1,k) !

            ey=yc(i,jm1,km1)-yc(iaf,j,km1) !
            oy=yc(iaf,j,km1)-yc(iaf,j,k) !

            qy=yc(im1,jm1,k)-yc(iaf,j,k) !
            ry=yc(iaf,j,km1)-yc(iaf,j,km1) !
```

```
      end if
c Modified by Matsfelt

                  ai1x=cy*bz-cz*by
                  ai1y=cz*bx-cx*bz
                  ai1z=cx*by-cy*bx

                  ai2x=ey*dz-ez*dy
                  ai2y=ez*dx-ex*dz
                  ai2z=ex*dy-ey*dx

                  aj1x=by*az-bz*ay
                  aj1y=bz*ax-bx*az
                  aj1z=bx*ay-by*ax

                  aj2x=oy*rz-oz*ry
                  aj2y=oz*rx-ox*rz
                  aj2z=ox*ry-oy*rx

                  ak1x=cy*pz-cz*py
                  ak1y=cz*px-cx*pz
                  ak1z=cx*py-cy*px

                  ak2x=qy*az-qz*ay
                  ak2y=qz*ax-qx*az
                  ak2z=qx*ay-qy*ax


                  areaex(i,j,k)=0.5*(ai1x+ai2x)
                  areaey(i,j,k)=0.5*(ai1y+ai2y)
                  areaez(i,j,k)=0.5*(ai1z+ai2z)

                  areanx(i,j,k)=0.5*(aj1x+aj2x)
                  areany(i,j,k)=0.5*(aj1y+aj2y)
                  areanz(i,j,k)=0.5*(aj1z+aj2z)

                  areahx(i,j,k)=0.5*(ak1x+ak2x)
                  areahy(i,j,k)=0.5*(ak1y+ak2y)
                  areahz(i,j,k)=0.5*(ak1z+ak2z)

110   continue

c-------calculate the volume.
```

```
c       volume=0.333*area times coord. (sum over all faces)
        volsum=0.0
        do 120 k= 2,nkm1
        do 120 j= 2,njm1
        do 120 i= 2,nim1

           im1=i-1
           jm1=j-1
           km1=k-1

c----------east face
           xe=0.25*(xc(i,j,k)+xc(i,jm1,k)+xc(i,jm1,km1)+xc(i,j,km1))
           ye=0.25*(yc(i,j,k)+yc(i,jm1,k)+yc(i,jm1,km1)+yc(i,j,km1))
           ze=0.25*(zc(i,j,k)+zc(i,jm1,k)+zc(i,jm1,km1)+zc(i,j,km1))

c Modified by Matsfelt
c Modify ye for the  locations in the airfoil
        if (i.le.iaf.and.j.eq.jaf) then
          ye=0.25*(yc(iaf,j,k)+yc(i,jm1,k)+yc(i,jm1,km1)+yc(iaf,j,km1))
        end if
c Modified by Matsfelt

           vole=xe*areaex(i,j,k)+ye*areaey(i,j,k)+ze*areaez(i,j,k)

c----------west face
           xw=
     & 0.25*(xc(im1,j,k)+xc(im1,jm1,k)+xc(im1,jm1,km1)+xc(im1,j,km1))
           yw=
     & 0.25*(yc(im1,j,k)+yc(im1,jm1,k)+yc(im1,jm1,km1)+yc(im1,j,km1))
           zw=
     & 0.25*(zc(im1,j,k)+zc(im1,jm1,k)+zc(im1,jm1,km1)+zc(im1,j,km1))

c Modified by Matsfelt
c Modify yw for the  locations in the airfoil
        if (i.le.iaf.and.j.eq.jaf) then
           yw=
     & 0.25*(yc(iaf,j,k)+yc(im1,jm1,k)+yc(im1,jm1,km1)+yc(iaf,j,km1))
        end if
c Modified by Matsfelt

          volw=-xw*areaex(i-1,j,k)-yw*areaey(i-1,j,k)-zw*areaez(i-1,j,k)

c----------north face
```

```
          xn=0.25*(xc(i,j,k)+xc(im1,j,k)+xc(im1,j,km1)+xc(i,j,km1))
          yn=0.25*(yc(i,j,k)+yc(im1,j,k)+yc(im1,j,km1)+yc(i,j,km1))
          zn=0.25*(zc(i,j,k)+zc(im1,j,k)+zc(im1,j,km1)+zc(i,j,km1))

c Modified by Matsfelt
c Modify yn for the  locations in the airfoil
      if (i.le.iaf.and.j.eq.jaf) then
        yn=0.25*(yc(iaf,j,k)+yc(iaf,j,k)+yc(iaf,j,km1)+yc(iaf,j,km1))
      end if
c Modified by Matsfelt

          voln=xn*areanx(i,j,k)+yn*areany(i,j,k)+zn*areanz(i,j,k)

c----------south face
          xs=
     & 0.25*(xc(i,jm1,k)+xc(im1,jm1,k)+xc(im1,jm1,km1)+xc(i,jm1,km1))
          ys=
     & 0.25*(yc(i,jm1,k)+yc(im1,jm1,k)+yc(im1,jm1,km1)+yc(i,jm1,km1))
          zs=
     & 0.25*(zc(i,jm1,k)+zc(im1,jm1,k)+zc(im1,jm1,km1)+zc(i,jm1,km1))

        vols=-xs*areanx(i,j-1,k)-ys*areany(i,j-1,k)-zs*areanz(i,j-1,k)

c----------high face
          xh=0.25*(xc(i,j,k)+xc(i,jm1,k)+xc(im1,jm1,k)+xc(im1,j,k))
          yh=0.25*(yc(i,j,k)+yc(i,jm1,k)+yc(im1,jm1,k)+yc(im1,j,k))
          zh=0.25*(zc(i,j,k)+zc(i,jm1,k)+zc(im1,jm1,k)+zc(im1,j,k))

c Modified by Matsfelt
c Modify yh for the  locations in the airfoil
      if (i.le.iaf.and.j.eq.jaf) then
          yh=0.25*(yc(iaf,j,k)+yc(i,jm1,k)+yc(im1,jm1,k)+yc(iaf,j,k))
      end if
c Modified by Matsfelt

          volh=xh*areahx(i,j,k)+yh*areahy(i,j,k)+zh*areahz(i,j,k)

c----------low face
          xl=
     & 0.25*(xc(i,j,km1)+xc(i,jm1,km1)+xc(im1,jm1,km1)+xc(im1,j,km1))
          yl=
     & 0.25*(yc(i,j,km1)+yc(i,jm1,km1)+yc(im1,jm1,km1)+yc(im1,j,km1))
          zl=
```

```
     & 0.25*(zc(i,j,km1)+zc(i,jm1,km1)+zc(im1,jm1,km1)+zc(im1,j,km1))

c Modified by Matsfelt
c Modify yl for the  locations in the airfoil
      if (i.le.iaf.and.j.eq.jaf) then
        yl=0.25*(yc(iaf,j,km1)+yc(i,jm1,km1)
     &          +yc(im1,jm1,km1)+yc(iaf,j,km1))
      end if
c Modified by Matsfelt

       voll=-xl*areahx(i,j,k-1)-yl*areahy(i,j,k-1)-zl*areahz(i,j,k-1)

       vol(i,j,k)=abs(vole+volw+voln+vols+volh+voll)/3.

c Modified by Matsfelt
c The volumes of jaf+1 cells are modified later in the code
c    and will then be added to volsum
       if (j.ne.(jaf+1)) then
         volsum=volsum+vol(i,j,k)

       end if
c Modified by Matsfelt


 120  continue


c Modified by Matsfelt
c Modify the areas and volumes for the nodes at jaf+1
      j=jaf

c------calculate the area of the control volume faces

      do k= 1,nkm1
      do i= 1,(iaf+1)

         im1=max(1,i-1)
         jm1=max(1,j-1)
         km1=max(1,k-1)
c---------
         ax=xc(i,j,k)-xc(im1,j,k) !
         ay=yc(i,j,k)-yc(im1,j,k) !
         az=zc(i,j,k)-zc(im1,j,k) !
```

```
c-----------
            bx=xc(i,j,k)-xc(i,j,km1)  !
            by=yc(i,j,k)-yc(i,j,km1)  !
            bz=zc(i,j,k)-zc(i,j,km1)  !
c-----------
            cx=xc(i,j,k)-xc(i,jm1,k)  !
            cy=yc(i,j,k)-yc(i,jm1,k)  !
            cz=zc(i,j,k)-zc(i,jm1,k)  !
c-----------
            dx=xc(i,jm1,km1)-xc(i,jm1,k)  !
            dy=yc(i,jm1,km1)-yc(i,jm1,k)  !
            dz=zc(i,jm1,km1)-zc(i,jm1,k)  !
c-----------
            ex=xc(i,jm1,km1)-xc(i,j,km1)  !
            ey=yc(i,jm1,km1)-yc(i,j,km1)  !
            ez=zc(i,jm1,km1)-zc(i,j,km1)  !
c-----------
            ox=xc(im1,j,km1)-xc(im1,j,k)  !
            oy=yc(im1,j,km1)-yc(im1,j,k)  !
            oz=zc(im1,j,km1)-zc(im1,j,k)  !
c-----------
            px=xc(im1,jm1,k)-xc(i,jm1,k)  !
            py=yc(im1,jm1,k)-yc(i,jm1,k)  !
            pz=zc(im1,jm1,k)-zc(i,jm1,k)  !
c-----------
            qx=xc(im1,jm1,k)-xc(im1,j,k)  !
            qy=yc(im1,jm1,k)-yc(im1,j,k)  !
            qz=zc(im1,jm1,k)-zc(im1,j,k)  !
c-----------
            rx=xc(im1,j,km1)-xc(i,j,km1)  !
            ry=yc(im1,j,km1)-yc(i,j,km1)  !
            rz=zc(im1,j,km1)-zc(i,j,km1)  !


                  ai1x=cy*bz-cz*by
                  ai1y=cz*bx-cx*bz
                  ai1z=cx*by-cy*bx

                  ai2x=ey*dz-ez*dy
                  ai2y=ez*dx-ex*dz
                  ai2z=ex*dy-ey*dx

                  aj1x=by*az-bz*ay
```

```
                    aj1y=bz*ax-bx*az
                    aj1z=bx*ay-by*ax

                    aj2x=oy*rz-oz*ry
                    aj2y=oz*rx-ox*rz
                    aj2z=ox*ry-oy*rx

                    ak1x=cy*pz-cz*py
                    ak1y=cz*px-cx*pz
                    ak1z=cx*py-cy*px

                    ak2x=qy*az-qz*ay
                    ak2y=qz*ax-qx*az
                    ak2z=qx*ay-qy*ax


                    areaexjafp1(i,j,k)=0.5*(ai1x+ai2x)
                    areaeyjafp1(i,j,k)=0.5*(ai1y+ai2y)
                    areaezjafp1(i,j,k)=0.5*(ai1z+ai2z)

                    areanxjafp1(i,j,k)=0.5*(aj1x+aj2x)
                    areanyjafp1(i,j,k)=0.5*(aj1y+aj2y)
                    areanzjafp1(i,j,k)=0.5*(aj1z+aj2z)

                    areahxjafp1(i,j,k)=0.5*(ak1x+ak2x)
                    areahyjafp1(i,j,k)=0.5*(ak1y+ak2y)
                    areahzjafp1(i,j,k)=0.5*(ak1z+ak2z)


      end do
      end do

c-------calculate the volume.
      j=jaf+1

      do k= 2,nkm1
      do i= 2,(iaf+1)

         im1=i-1
         jm1=j-1
         km1=k-1

c----------east face
```

```
            xe=0.25*(xc(i,j,k)+xc(i,jm1,k)+xc(i,jm1,km1)+xc(i,j,km1))
            ye=0.25*(yc(i,j,k)+yc(i,jm1,k)+yc(i,jm1,km1)+yc(i,j,km1))
            ze=0.25*(zc(i,j,k)+zc(i,jm1,k)+zc(i,jm1,km1)+zc(i,j,km1))

            vole=
     & xe*areaex(i,j,k)+ye*areaey(i,j,k)+ze*areaez(i,j,k)


c----------west face
            xw=
     & 0.25*(xc(im1,j,k)+xc(im1,jm1,k)+xc(im1,jm1,km1)+xc(im1,j,km1))
            yw=
     & 0.25*(yc(im1,j,k)+yc(im1,jm1,k)+yc(im1,jm1,km1)+yc(im1,j,km1))
            zw=
     & 0.25*(zc(im1,j,k)+zc(im1,jm1,k)+zc(im1,jm1,km1)+zc(im1,j,km1))

            volw=
     &    -xw*areaex(i-1,j,k)-yw*areaey(i-1,j,k)
     &    -zw*areaez(i-1,j,k)


c----------north face
            xn=0.25*(xc(i,j,k)+xc(im1,j,k)+xc(im1,j,km1)+xc(i,j,km1))
            yn=0.25*(yc(i,j,k)+yc(im1,j,k)+yc(im1,j,km1)+yc(i,j,km1))
            zn=0.25*(zc(i,j,k)+zc(im1,j,k)+zc(im1,j,km1)+zc(i,j,km1))

            voln=
     & xn*areanx(i,j,k)+yn*areany(i,j,k)+zn*areanz(i,j,k)


c----------south face
            xs=
     & 0.25*(xc(i,jm1,k)+xc(im1,jm1,k)+xc(im1,jm1,km1)+xc(i,jm1,km1))
            ys=
     & 0.25*(yc(i,jm1,k)+yc(im1,jm1,k)+yc(im1,jm1,km1)+yc(i,jm1,km1))
            zs=
     & 0.25*(zc(i,jm1,k)+zc(im1,jm1,k)+zc(im1,jm1,km1)+zc(i,jm1,km1))

c Using the areanyjafp1 for the south areeas
            vols=
     &    -xs*areanxjafp1(i,j-1,k)-ys*areanyjafp1(i,j-1,k)
     &    -zs*areanzjafp1(i,j-1,k)
```

```
c----------high face
          xh=0.25*(xc(i,j,k)+xc(i,jm1,k)+xc(im1,jm1,k)+xc(im1,j,k))
          yh=0.25*(yc(i,j,k)+yc(i,jm1,k)+yc(im1,jm1,k)+yc(im1,j,k))
          zh=0.25*(zc(i,j,k)+zc(i,jm1,k)+zc(im1,jm1,k)+zc(im1,j,k))

          volh=
     & xh*areahx(i,j,k)+yh*areahy(i,j,k)+zh*areahz(i,j,k)


c----------low face
          xl=
     & 0.25*(xc(i,j,km1)+xc(i,jm1,km1)+xc(im1,jm1,km1)+xc(im1,j,km1))
          yl=
     & 0.25*(yc(i,j,km1)+yc(i,jm1,km1)+yc(im1,jm1,km1)+yc(im1,j,km1))
          zl=
     & 0.25*(zc(i,j,km1)+zc(i,jm1,km1)+zc(im1,jm1,km1)+zc(im1,j,km1))

          voll=
     &    -xl*areahx(i,j,k-1)-yl*areahy(i,j,k-1)
     &    -zl*areahz(i,j,k-1)


          vol(i,j,k)=abs(vole+volw+voln+vols+volh+voll)/3.

          volsum=volsum+vol(i,j,k)

        end do
        end do
c Modified by Matsfelt


c-------------------------------------------------------------------------------
        do 140 k= 2,nkm1
        do 140 j= 2,njm1
        do 140 i= 2,nim1

          im1=i-1
          jm1=j-1
          km1=k-1

          xw=
     & 0.25*(xc(im1,j,k)+xc(im1,jm1,k)+xc(im1,j,km1)+xc(im1,jm1,km1))
          xe=0.25*(xc(i,j,k)+xc(i,jm1,k)+xc(i,j,km1)+xc(i,jm1,km1))
```

```
            yw=
     &  0.25*(yc(im1,j,k)+yc(im1,jm1,k)+yc(im1,j,km1)+yc(im1,jm1,km1))
            ye=0.25*(yc(i,j,k)+yc(i,jm1,k)+yc(i,j,km1)+yc(i,jm1,km1))
            zw=
     &  0.25*(zc(im1,j,k)+zc(im1,jm1,k)+zc(im1,j,km1)+zc(im1,jm1,km1))
            ze=0.25*(zc(i,j,k)+zc(i,jm1,k)+zc(i,j,km1)+zc(i,jm1,km1))

            xs=
     &  0.25*(xc(i,jm1,k)+xc(im1,jm1,k)+xc(i,jm1,km1)+xc(im1,jm1,km1))
            xn=0.25*(xc(i,j,k)+xc(im1,j,k)+xc(i,j,km1)+xc(im1,j,km1))
            ys=
     &  0.25*(yc(i,jm1,k)+yc(im1,jm1,k)+yc(i,jm1,km1)+yc(im1,jm1,km1))
            yn=0.25*(yc(i,j,k)+yc(im1,j,k)+yc(i,j,km1)+yc(im1,j,km1))
            zs=
     &  0.25*(zc(i,jm1,k)+zc(im1,jm1,k)+zc(i,jm1,km1)+zc(im1,jm1,km1))
            zn=0.25*(zc(i,j,k)+zc(im1,j,k)+zc(i,j,km1)+zc(im1,j,km1))

            xl=
     &  0.25*(xc(i,j,km1)+xc(im1,j,km1)+xc(i,jm1,km1)+xc(im1,jm1,km1))
            xh=0.25*(xc(i,j,k)+xc(im1,j,k)+xc(i,jm1,k)+xc(im1,jm1,k))
            yl=
     &  0.25*(yc(i,j,km1)+yc(im1,j,km1)+yc(i,jm1,km1)+yc(im1,jm1,km1))
            yh=0.25*(yc(i,j,k)+yc(im1,j,k)+yc(i,jm1,k)+yc(im1,jm1,k))
            zl=
     &  0.25*(zc(i,j,km1)+zc(im1,j,km1)+zc(i,jm1,km1)+zc(im1,jm1,km1))
            zh=0.25*(zc(i,j,k)+zc(im1,j,k)+zc(i,jm1,k)+zc(im1,jm1,k))

c Modified by Matsfelt
c Modify y for the  locations in the airfoil
          if (i.le.iaf.and.j.eq.jaf) then
            yw=
     &  0.25*(yc(iaf,j,k)+yc(im1,jm1,k)+yc(iaf,j,km1)+yc(im1,jm1,km1))
            ye=0.25*(yc(iaf,j,k)+yc(i,jm1,k)+yc(iaf,j,km1)+yc(i,jm1,km1))

            yn=0.25*(yc(iaf,j,k)+yc(iaf,j,k)+yc(iaf,j,km1)+yc(iaf,j,km1))

            yl=0.25*(yc(iaf,j,km1)+yc(iaf,j,km1)
     &           +yc(i,jm1,km1)+yc(im1,jm1,km1))
            yh=0.25*(yc(iaf,j,k)+yc(iaf,j,k)+yc(i,jm1,k)+yc(im1,jm1,k))
          end if
c Modified by Matsfelt

c---------calculate the weightfunctions
```

```
      del1=
     &    sqrt((xe-xp(i,j,k))**2+(ye-yp(i,j,k))**2+(ze-zp(i,j,k))**2)
      del2=sqrt((xe-xp(i+1,j,k))**2+(ye-yp(i+1,j,k))**2
     &          +(ze-zp(i+1,j,k))**2)
      fx(i,j,k)=del1/(del1+del2)

      del1=
     &    sqrt((xn-xp(i,j,k))**2+(yn-yp(i,j,k))**2+(zn-zp(i,j,k))**2)
      del2=sqrt((xn-xp(i,j+1,k))**2+(yn-yp(i,j+1,k))**2
     &          +(zn-zp(i,j+1,k))**2)
      fy(i,j,k)=del1/(del1+del2)

      del1=sqrt((xh-xp(i,j,k))**2+(yh-yp(i,j,k))**2
     &          +(zh-zp(i,j,k))**2)
      del2=sqrt((xh-xp(i,j,k+1))**2+(yh-yp(i,j,k+1))**2
     &          +(zh-zp(i,j,k+1))**2)
      fz(i,j,k)=del1/(del1+del2)
  140 continue


c--- Volumes and the weight functions for boundary nodes ...

chapter 2 2 2 2 2 set variables to small vzpue 2 2 2 2 ...


      return
      end
```

B

CALC part of conv.f file including modifications

```
      do k=2,nkm1
      do j=2,njm1
      do i=2,nim1

c east ...

c north
         jp2=min(j+2,nj)
         jp1=min(j+1,njm1)
         jp1=j+1
         jm1=j-1

c Modified by Matsfelt
         if (i.le.iaf) then
            if (j.eq.(jaf-1)) then
               jp2=jp1
            else if (j.eq.jaf) then
               jp2=j
               jp1=j
            else if (j.eq.(jaf+1)) then
               jm1=j
            end if
         end if
c Modified by Matsfelt
```

```
            voln=0.5*(vol(i,jp1,k)+vol(i,j,k))

            ann=(areanx(i,j,k)**2+areany(i,j,k)**2+
     .           areanz(i,j,k)**2)/voln

            un=fy(i,j,k)*phi(i,j+1,k,u)+(1.-fy(i,j,k))*phi(i,j,k,u)
            vn=fy(i,j,k)*phi(i,j+1,k,v)+(1.-fy(i,j,k))*phi(i,j,k,v)
            wn=fy(i,j,k)*phi(i,j+1,k,w)+(1.-fy(i,j,k))*phi(i,j,k,w)

            convnc= denn*(areanx(i,j,k)*un+areany(i,j,k)*vn
     &                   +areanz(i,j,k)*wn)


            dp=0.25*(phi(i,jp2,k,p)-phi(i,j,k,p)+phi(i,jp1,k,p)
     .          -phi(i,jm1,k,p))

            convn(i,j,k)=convnc+ann*dp*dtt

c high ...

          end do
          end do
          end do
```

C

CALC dpdyf.f file including modifications

```
      function dpdyf(i,j,k)
      include 'COMMON'

c the derivative dpdy at node i,j,k is calculated here using
c Neumann boundary condition

      jm1=max(j-1,2)
      jp1=min(j+1,njm1)

      phie=fx(i,j,k)*phi(i+1,j,k,p)+(1.-fx(i,j,k))*phi(i,j,k,p)
      phiw=fx(i-1,j,k)*phi(i,j,k,p)+(1.-fx(i-1,j,k))
     &     *phi(i-1,j,k,p)
      phin=fy(i,j,k)*phi(i,jp1,k,p)+(1.-fy(i,j,k))*phi(i,j,k,p)
      phis=fy(i,j-1,k)*phi(i,j,k,p)+(1.-fy(i,j-1,k))
     &     *phi(i,jm1,k,p)
      phih=fz(i,j,k)*phi(i,j,k+1,p)+(1.-fz(i,j,k))*phi(i,j,k,p)
      phil=fz(i,j,k-1)*phi(i,j,k,p)+(1.-fz(i,j,k-1))
     &     *phi(i,j,k-1,p)


      dpdyf=(phie*areaey(i,j,k)-phiw*areaey(i-1,j,k)
     &      +phin*areany(i,j,k)-phis*areany(i,j-1,k)
     &      +phih*areahy(i,j,k)-phil*areahy(i,j,k-1))/vol(i,j,k)
```

```
c Modified by Matsfelt
c To break the connection between the cell jaf+1 above the airfoil
c   and jaf below.
      if (i.le.iaf.and.j.eq.(jaf+1)) then

      phis=phi(i,j,k,p)

      dpdyf=(phie*areaey(i,j,k)-phiw*areaey(i-1,j,k)
     &      +phin*areany(i,j,k)-phis*areanyjafp1(i,j-1,k)
     &      +phih*areahy(i,j,k)-phil*areahy(i,j,k-1))
     &      /vol(i,j,k)

      end if
c Modified by Matsfelt

      return
      end
```

D

CALC part of peter_init.f file including modifications

```
      subroutine peter_init(xa,ya,za,iin,jin,kin,id,jd,kd,
     .                      cycli,cyclj,cyclk,iaf,jaf,ifirst,ilast)

      include 'PETER_COMMON'
      common /peter/kst(100),kst_2d(100)
      dimension xa(id,jd,kd),ya(id,jd,kd),za(id,jd,kd)
      logical cycli,cyclj,cyclk
      integer walwidth


      pi=4.*atan(1.)

ccccccccc Calculate the number of levels in the multigrid ccccc

      ipa=iin
      jpa=jin
      kpa=kin
      nlev=1
      ipb=ipa
      jpb=jpa
      kpb=kpa
      i1c=ifirst
      i2c=ilast

c Modified by Matsfelt
```

```
      iairfoil=iaf
      jairfoil=jaf
c Modified by Matsfelt

      do i=1,100
         if(mod(ipa-2,2).eq.0)then
         if(mod(jpa-2,2).eq.0)then
         if(mod(kpa-2,2).eq.0)then
         ipa=(ipa-2)/2+2
         jpa=(jpa-2)/2+2
         kpa=(kpa-2)/2+2
         walwidth=i2c-i1c
         if(ipa.ge.4.and.jpa.ge.4.and.kpa.ge.4.and.nlev.le.3)then
            ipb=ipa
            jpb=jpa
            kpb=kpa
            i1c=(i1c)/2+1
            i2c=(i2c-1)/2+1

c Modified by Matsfelt
            iairfoil=(iairfoil-1)/2+1
            jairfoil=(jairfoil-1)/2+1
c Modified by Matsfelt

            nlev=nlev+1
            write(6,*)'nlev,ipb,jpb,kpb' ,nlev, ipb, jpb, kpb
            write(6,*)'nlev,i1c,i2c' , nlev,i1c,i2c
         end if

         end if
         end if
         end if
      end do
      write(6,*)'NUMBER OF LEVELS=',nlev

ccccccc Setup pointer system and define coarse grids ccccccccccccc

      ni(1)=ipb
      nj(1)=jpb
      nk(1)=kpb
      kst(1)=1
      do klev=2,nlev
         ni(klev)=2*(ni(klev-1)-2)+2
```

```
          nj(klev)=2*(nj(klev-1)-2)+2
          nk(klev)=2*(nk(klev-1)-2)+2
          kst(klev)=kst(klev-1)+ni(klev-1)*nj(klev-1)*nk(klev-1)
        end do


c Modified by Matsfelt
c Trace the iaf and jaf through the levels

        iafp(1)=iairfoil
        jafp(1)=jairfoil

        write(6,*)'klev=1:iafp(1),jafp(1)',
       .            iafp(1),jafp(1)

        do klev=2,nlev
          iafp(klev)=2*(iafp(klev-1)-1)+1
          jafp(klev)=2*(jafp(klev-1)-1)+1

          write(6,*)'klev,iafp,jafp',
       .          klev,iafp(klev),jafp(klev)
        end do
c Modified by Matsfelt

        call key(nlev,lst,ist,ni(nlev),nj(nlev),nk(nlev))

c$doacross local(k,j,i,io)
        do k=1,nk(nlev)
        do j=1,nj(nlev)
        do i=1,ni(nlev)
          io=lst(k)+ist(i)+j
          p(io)=xa(i,j,k)
          t(io)=ya(i,j,k)
          pm(io)=za(i,j,k)
        end do
        end do
        end do


        do klev=nlev,2,-1
          klevc=klev-1

          call key(klevc,lstc,istc,ni(klevc),nj(klevc),nk(klevc))
```

```
          call key(klev,lst,ist,ni(klev),nj(klev),nk(klev))

c x,y,z on coarse meshes
          do k=1,nk(klevc)-1
             do j=1,nj(klevc)-1
                do i=1,ni(klevc)-1
                   io=lstc(k)+istc(i)+j
                   p(io)=p(lst(2*k-1)+ist(2*i-1)+2*j-1)
                   t(io)=t(lst(2*k-1)+ist(2*i-1)+2*j-1)
                   pm(io)=pm(lst(2*k-1)+ist(2*i-1)+2*j-1)

                end do
             end do
          end do
       end do


 88    format(1x,15(1pe13.6,2x))


ccccccc Calculate the coefficients on all meshes  ccccccccccccc

       do klev=1,nlev
       call key(klev,lst,ist,ni(klev),nj(klev),nk(klev))

c---   Area calculation

       do k=1,nk(klev)-1
             do i=1,ni(klev)-1
                do j=1,nj(klev)-1
                   im1=max(1,i-1)
                   jm1=max(1,j-1)
                   km1=max(1,k-1)

                   io=lst(k)+ist(i)+j
                   il=lst(km1)+ist(i)+j
                   iw=lst(k)+ist(im1)+j
                   iwl=lst(km1)+ist(im1)+j
                   is=lst(k)+ist(i)+jm1
                   isw=lst(k)+ist(im1)+jm1
                   isl=lst(km1)+ist(i)+jm1

                   ax=p(io)-p(iw)
```

```
ay=t(io)-t(iw)
az=pm(io)-pm(iw)

bx=p(io)-p(il)
by=t(io)-t(il)
bz=pm(io)-pm(il)

cx=p(io)-p(is)
cy=t(io)-t(is)
cz=pm(io)-pm(is)

dx=p(isl)-p(is)
dy=t(isl)-t(is)
dz=pm(isl)-pm(is)

ex=p(isl)-p(il)
ey=t(isl)-t(il)
ez=pm(isl)-pm(il)

ox=p(iwl)-p(iw)
oy=t(iwl)-t(iw)
oz=pm(iwl)-pm(iw)

px=p(isw)-p(is)
py=t(isw)-t(is)
pz=pm(isw)-pm(is)

qx=p(isw)-p(iw)
qy=t(isw)-t(iw)
qz=pm(isw)-pm(iw)

rx=p(iwl)-p(il)
ry=t(iwl)-t(il)
rz=pm(iwl)-pm(il)

ai1x=cy*bz-cz*by
ai1y=cz*bx-cx*bz
ai1z=cx*by-cy*bx

ai2x=ey*dz-ez*dy
ai2y=ez*dx-ex*dz
ai2z=ex*dy-ey*dx
```

```
                aj1x=by*az-bz*ay
                aj1y=bz*ax-bx*az
                aj1z=bx*ay-by*ax

                aj2x=oy*rz-oz*ry
                aj2y=oz*rx-ox*rz
                aj2z=ox*ry-oy*rx

                ak1x=cy*pz-cz*py
                ak1y=cz*px-cx*pz
                ak1z=cx*py-cy*px

                ak2x=qy*az-qz*ay
                ak2y=qz*ax-qx*az
                ak2z=qx*ay-qy*ax


c Modified by Matsfelt
c Modify the a..y parameters for if the location is
c  in the airfoil.
            if (i.le.iafp(klev).and.j.eq.jafp(klev)) then

                ioiaf=lst(k)+ist(iafp(klev))+j
                iliaf=lst(km1)+ist(iafp(klev))+j
                iwiaf=lst(k)+ist(iafp(klev))+j
                iwliaf=lst(km1)+ist(iafp(klev))+j

                ax=p(io)-p(iw)
                ay=t(ioiaf)-t(iwiaf)
                az=pm(io)-pm(iw)

                bx=p(io)-p(il)
                by=t(ioiaf)-t(iliaf)
                bz=pm(io)-pm(il)

                cx=p(io)-p(is)
                cy=t(ioiaf)-t(is)
                cz=pm(io)-pm(is)

                ex=p(isl)-p(il)
                ey=t(isl)-t(iliaf)
                ez=pm(isl)-pm(il)
```

```
                ox=p(iwl)-p(iw)
                oy=t(iwliaf)-t(iwiaf)
                oz=pm(iwl)-pm(iw)

                qx=p(isw)-p(iw)
                qy=t(isw)-t(iwiaf)
                qz=pm(isw)-pm(iw)

                rx=p(iwl)-p(il)
                ry=t(iwliaf)-t(iliaf)
                rz=pm(iwl)-pm(il)

                ai1x=cy*bz-cz*by
                ai1y=cz*bx-cx*bz
                ai1z=cx*by-cy*bx

                ai2x=ey*dz-ez*dy
                ai2y=ez*dx-ex*dz
                ai2z=ex*dy-ey*dx

                aj1x=by*az-bz*ay
                aj1y=bz*ax-bx*az
                aj1z=bx*ay-by*ax

                aj2x=oy*rz-oz*ry
                aj2y=oz*rx-ox*rz
                aj2z=ox*ry-oy*rx

                ak1x=cy*pz-cz*py
                ak1y=cz*px-cx*pz
                ak1z=cx*py-cy*px

                ak2x=qy*az-qz*ay
                ak2y=qz*ax-qx*az
                ak2z=qx*ay-qy*ax

            end if
c Modified by Matsfelt

                aw(io)=0.5*(ai1x+ai2x)
                ae(io)=0.5*(ai1y+ai2y)
                as(io)=0.5*(ai1z+ai2z)
```

```
                an(io)=0.5*(aj1x+aj2x)
                al(io)=0.5*(aj1y+aj2y)
                ah(io)=0.5*(aj1z+aj2z)

                ap(io)=0.5*(ak1x+ak2x)
                su(io)=0.5*(ak1y+ak2y)
                co(io)=0.5*(ak1z+ak2z)

             end do
          end do
       end do

c Modified by Matsfelt
c Modify the the northern area of the jaf cells because they are
c used in the jaf+1 volume calculation.

c copy the northern areas to the jafp1 matrixes
      do k=1,nk(klev)-1
            do i=1,ni(klev)-1
               do j=1,nj(klev)-1

                  io=lst(k)+ist(i)+j

                  anjafp1(io)=an(io)
                  aljafp1(io)=al(io)
                  ahjafp1(io)=ah(io)
               end do
            end do
      end do


c---  Area calculation
      do k=1,nk(klev)-1
            do i=1,(iafp(klev)+1)
                j=jafp(klev)

                  im1=max(1,i-1)
                  jm1=max(1,j-1)
                  km1=max(1,k-1)

                  io=lst(k)+ist(i)+j
                  il=lst(km1)+ist(i)+j
                  iw=lst(k)+ist(im1)+j
```

```
iwl=lst(km1)+ist(im1)+j
is=lst(k)+ist(i)+jm1
isw=lst(k)+ist(im1)+jm1
isl=lst(km1)+ist(i)+jm1


ax=p(io)-p(iw)
ay=t(io)-t(iw)
az=pm(io)-pm(iw)

bx=p(io)-p(il)
by=t(io)-t(il)
bz=pm(io)-pm(il)

cx=p(io)-p(is)
cy=t(io)-t(is)
cz=pm(io)-pm(is)

dx=p(isl)-p(is)
dy=t(isl)-t(is)
dz=pm(isl)-pm(is)

ex=p(isl)-p(il)
ey=t(isl)-t(il)
ez=pm(isl)-pm(il)

ox=p(iwl)-p(iw)
oy=t(iwl)-t(iw)
oz=pm(iwl)-pm(iw)

px=p(isw)-p(is)
py=t(isw)-t(is)
pz=pm(isw)-pm(is)

qx=p(isw)-p(iw)
qy=t(isw)-t(iw)
qz=pm(isw)-pm(iw)

rx=p(iwl)-p(il)
ry=t(iwl)-t(il)
rz=pm(iwl)-pm(il)

ai1x=cy*bz-cz*by
```

```
                ai1y=cz*bx-cx*bz
                ai1z=cx*by-cy*bx

                ai2x=ey*dz-ez*dy
                ai2y=ez*dx-ex*dz
                ai2z=ex*dy-ey*dx

                aj1x=by*az-bz*ay
                aj1y=bz*ax-bx*az
                aj1z=bx*ay-by*ax

                aj2x=oy*rz-oz*ry
                aj2y=oz*rx-ox*rz
                aj2z=ox*ry-oy*rx

                ak1x=cy*pz-cz*py
                ak1y=cz*px-cx*pz
                ak1z=cx*py-cy*px

                ak2x=qy*az-qz*ay
                ak2y=qz*ax-qx*az
                ak2z=qx*ay-qy*ax


                anjafp1(io)=0.5*(aj1x+aj2x)
                aljafp1(io)=0.5*(aj1y+aj2y)
                ahjafp1(io)=0.5*(aj1z+aj2z)


             end do
          end do
c Modified by Matsfelt


c---- Aspect ratio calculation ...

c---- Choice of smoother ...


c---- Calculate volumes

        do k=2,nk(klev)-1
           do i=2,ni(klev)-1
```

```
            do j=2,nj(klev)-1
               io=lst(k)+ist(i)+j
               iw=lst(k)+ist(i-1)+j
               is=lst(k)+ist(i)+j-1
               il=lst(k-1)+ist(i)+j
               iws=lst(k)+ist(i-1)+j-1
               iwl=lst(k-1)+ist(i-1)+j
               isl=lst(k-1)+ist(i)+j-1
               iwsl=lst(k-1)+ist(i-1)+j-1

               xw=0.25*(p(iw)+p(iws)+p(iwl)+p(iwsl))
               xe=0.25*(p(io)+p(is)+p(il)+p(isl))
               yw=0.25*(t(iw)+t(iws)+t(iwl)+t(iwsl))
               ye=0.25*(t(io)+t(is)+t(il)+t(isl))
               zw=0.25*(pm(iw)+pm(iws)+pm(iwl)+pm(iwsl))
               ze=0.25*(pm(io)+pm(is)+pm(il)+pm(isl))

               xs=0.25*(p(is)+p(iws)+p(isl)+p(iwsl))
               xn=0.25*(p(io)+p(iw)+p(il)+p(iwl))
               ys=0.25*(t(is)+t(iws)+t(isl)+t(iwsl))
               yn=0.25*(t(io)+t(iw)+t(il)+t(iwl))
               zs=0.25*(pm(is)+pm(iws)+pm(isl)+pm(iwsl))
               zn=0.25*(pm(io)+pm(iw)+pm(il)+pm(iwl))

               xl=0.25*(p(il)+p(iwl)+p(isl)+p(iwsl))
               xh=0.25*(p(io)+p(iw)+p(is)+p(iws))
               yl=0.25*(t(il)+t(iwl)+t(isl)+t(iwsl))
               yh=0.25*(t(io)+t(iw)+t(is)+t(iws))
               zl=0.25*(pm(il)+pm(iwl)+pm(isl)+pm(iwsl))
               zh=0.25*(pm(io)+pm(iw)+pm(is)+pm(iws))

c Modified by Matsfelt
c Modify the y dependent location parameters if the location is
c  in the airfoil.
            if (i.le.iafp(klev).and.j.eq.jafp(klev)) then

               ioiaf=lst(k)+ist(iafp(klev))+j
               iwiaf=lst(k)+ist(iafp(klev))+j
               iliaf=lst(k-1)+ist(iafp(klev))+j
               iwliaf=lst(k-1)+ist(iafp(klev))+j

               yw=0.25*(t(iwiaf)+t(iws)+t(iwliaf)+t(iwsl))
               ye=0.25*(t(ioiaf)+t(is)+t(iliaf)+t(isl))
```

```
                yn=0.25*(t(ioiaf)+t(iwiaf)+t(iliaf)+t(iwliaf))

                yl=0.25*(t(iliaf)+t(iwliaf)+t(isl)+t(iwsl))
                yh=0.25*(t(ioiaf)+t(iwiaf)+t(is)+t(iws))

             end if
c Modified by Matsfelt

                ew=sqrt((xe-xw)**2+(ye-yw)**2+(ze-zw)**2)
                sn=sqrt((xn-xs)**2+(yn-ys)**2+(zn-zs)**2)
                hl=sqrt((xh-xl)**2+(yh-yl)**2+(zh-zl)**2)

                vole=xe*aw(io)+ye*ae(io)+ze*as(io)
                volw=-xw*aw(iw)-yw*ae(iw)-zw*as(iw)
                voln=yn*al(io)+zn*ah(io)+xn*an(io)
                vols=-ys*al(is)-zs*ah(is)-xs*an(is)
                volh=zh*co(io)+xh*ap(io)+yh*su(io)
                voll=-zl*co(il)-xl*ap(il)-yl*su(il)


                sr(io)=abs(vole+volw+voln+vols+volh+voll)/3.
                if(sr(io).lt.1e-20)write(6,*)'VOL=0',i,j,k,klev,
     .                vole,volw,vols,voln,volh,voll

             end do
          end do
       end do

c Modified by Matsfelt
c Modify the jaf+1 volumes until iaf+1
c---- Calculate volumes

       do k=2,nk(klev)-1
          do i=2,(iafp(klev)+1)
                j=jafp(klev)+1


                io=lst(k)+ist(i)+j
                iw=lst(k)+ist(i-1)+j
                is=lst(k)+ist(i)+j-1
                il=lst(k-1)+ist(i)+j
                iws=lst(k)+ist(i-1)+j-1
```

```
      iwl=lst(k-1)+ist(i-1)+j
      isl=lst(k-1)+ist(i)+j-1
      iwsl=lst(k-1)+ist(i-1)+j-1

      xw=0.25*(p(iw)+p(iws)+p(iwl)+p(iwsl))
      xe=0.25*(p(io)+p(is)+p(il)+p(isl))
      yw=0.25*(t(iw)+t(iws)+t(iwl)+t(iwsl))
      ye=0.25*(t(io)+t(is)+t(il)+t(isl))
      zw=0.25*(pm(iw)+pm(iws)+pm(iwl)+pm(iwsl))
      ze=0.25*(pm(io)+pm(is)+pm(il)+pm(isl))

      xs=0.25*(p(is)+p(iws)+p(isl)+p(iwsl))
      xn=0.25*(p(io)+p(iw)+p(il)+p(iwl))
      ys=0.25*(t(is)+t(iws)+t(isl)+t(iwsl))
      yn=0.25*(t(io)+t(iw)+t(il)+t(iwl))
      zs=0.25*(pm(is)+pm(iws)+pm(isl)+pm(iwsl))
      zn=0.25*(pm(io)+pm(iw)+pm(il)+pm(iwl))

      xl=0.25*(p(il)+p(iwl)+p(isl)+p(iwsl))
      xh=0.25*(p(io)+p(iw)+p(is)+p(iws))
      yl=0.25*(t(il)+t(iwl)+t(isl)+t(iwsl))
      yh=0.25*(t(io)+t(iw)+t(is)+t(iws))
      zl=0.25*(pm(il)+pm(iwl)+pm(isl)+pm(iwsl))
      zh=0.25*(pm(io)+pm(iw)+pm(is)+pm(iws))


      ew=sqrt((xe-xw)**2+(ye-yw)**2+(ze-zw)**2)
      sn=sqrt((xn-xs)**2+(yn-ys)**2+(zn-zs)**2)
      hl=sqrt((xh-xl)**2+(yh-yl)**2+(zh-zl)**2)

      vole=xe*aw(io)+ye*ae(io)+ze*as(io)
      volw=-xw*aw(iw)-yw*ae(iw)-zw*as(iw)
      voln=yn*al(io)+zn*ah(io)+xn*an(io)

      vols=-ys*aljafp1(is)-zs*ahjafp1(is)-xs*anjafp1(is)

      volh=zh*co(io)+xh*ap(io)+yh*su(io)
      voll=-zl*co(il)-xl*ap(il)-yl*su(il)

      sr(io)=abs(vole+volw+voln+vols+volh+voll)/3.
      if(sr(io).lt.1e-20)write(6,*)'VOL=0',i,j,k,klev,
     .     vole,volw,vols,voln,volh,voll
```

```
         end do
      end do
c Modified by Matsfelt


c--- Coefficient assembly for d2p/dx2+d2p/dy2=f

c$doacross local(k,i,j,io)
      do k=1,nk(klev)
         do i=1,ni(klev)
            do j=1,nj(klev)
               io=lst(k)+ist(i)+j
               p(io)=aw(io)**2+ae(io)**2+as(io)**2
               t(io)=al(io)**2+an(io)**2+ah(io)**2
               pm(io)=co(io)**2+ap(io)**2+su(io)**2
            end do
         end do
      end do

      do k=2,nk(klev)-1
         do i=2,ni(klev)-1
            do j=2,nj(klev)-1
               io=lst(k)+ist(i)+j
               in=io+1
               is=io-1
               ie=lst(k)+ist(i+1)+j
               iw=lst(k)+ist(i-1)+j
               il=lst(k-1)+ist(i)+j
               ih=lst(k+1)+ist(i)+j

            vole=0.5*(sr(ie)+sr(io))+1.e-30
            if (i.eq.ni(klev)-1) vole=0.5*sr(io)
            ae(io)=p(io)/vole

            volw=0.5*(sr(io)+sr(iw))+1.e-30
            if (i.eq.2) volw=0.5*sr(io)
            aw(io)=p(iw)/volw

            voln=0.5*(sr(in)+sr(io))+1.e-30
            if (j.eq.nj(klev)-1) voln=0.5*sr(io)
            an(io)=t(io)/voln

            vols=0.5*(sr(io)+sr(is))+1.e-30
```

```
            if (j.eq.2) vols=0.5*sr(io)
            as(io)=t(is)/vols

            volh=0.5*(sr(ih)+sr(io))+1.e-30
            if (k.eq.nk(klev)-1) volh=0.5*sr(io)
            ah(io)=pm(io)/volh

            voll=0.5*(sr(io)+sr(il))+1.e-30
            if (k.eq.2) voll=0.5*sr(io)
            al(io)=pm(il)/voll

c Modified by Matsfelt
c Modify the y dependent location parameters if the location is
c  in the airfoil and save in the parameter locations from above.
          if (i.le.iafp(klev).and.j.eq.jafp(klev)) then

            an(io)=0.

         elseif (i.le.iafp(klev).and.j.eq.(jafp(klev)+1)) then

            as(io)=0.

          end if
c Modified by Matsfelt

             end do
           end do
         end do


c--- Boundary condition dp/dn=0 ...

c--- Diagonal coefficient evaluation ...

         end do
         return
         end
```

E

CALC part of peter_multi.f file including modifications

```
      subroutine peter_multi(resin,iin,jin,kin,
     .           tryck,sormax,icycmax,cycli,cyclj,cyclk)

      include 'PETER_COMMON'
      logical cycli,cyclj,cyclk
      dimension resin(iin,jin,kin),tryck(iin,jin,kin)
      integer*4 lock(0:1000)


c--- Initialization ...

c--- Bilinear restriction ...


c--- Bilinear prolongation

c go from the coarsest to the finest grid
      do 800 klev=2,nlev
         call key(klev,lst,ist,ni(klev),nj(klev),nk(klev))
         klevc=klev-1
         call key(klevc,lstc,istc,ni(klevc),nj(klevc),nk(klevc))

c fix Neumann boundary conditions
         do k=1,nk(klevc)
            do i=1,ni(klevc)
```

```
            in=lstc(k)+istc(i)+nj(klevc)
            inm1=lstc(k)+istc(i)+nj(klevc)-1
            is=lstc(k)+istc(i)+1
            ism1=lstc(k)+istc(i)+2
            p(in)=p(inm1)
            p(is)=p(ism1)
         end do
      end do
      do k=1,nk(klevc)
         do j=1,nj(klevc)
            in=lstc(k)+istc(ni(klevc))+j
            inm1=lstc(k)+istc(ni(klevc)-1)+j
            is=lstc(k)+istc(1)+j
            ism1=lstc(k)+istc(2)+j
            p(in)=p(inm1)
            p(is)=p(ism1)
         end do
      end do
      do i=1,ni(klevc)
         do j=1,nj(klevc)
            ih=lstc(nk(klevc))+istc(i)+j
            il=lstc(1)+istc(i)+j
            ihm1=lstc(nk(klevc)-1)+istc(i)+j
            ilm1=lstc(2)+istc(i)+j
            p(ih)=p(ihm1)
            p(il)=p(ilm1)
         end do
      end do

     call peter_cyclic(istc,lstc,ni(klevc),nj(klevc),nk(klevc),
     .                  klevc,cycli,cyclj,cyclk)

      do k=2,nk(klevc)
      lstc0=lstc(k)
      lstc1=lstc(k-1)
      lst2=lst(2*k-2)
      lst3=lst(2*k-3)
      do i=2,ni(klevc)
      istc0=istc(i)
      istc1=istc(i-1)
      ist2=ist(2*i-2)
      ist3=ist(2*i-3)
      do j=2,nj(klevc)
```

```
c           coarser mesh location parameters
c North
             ihnec=lstc0+istc0+j
             ihnwc=lstc0+istc1+j
             ilnec=lstc1+istc0+j
             ilnwc=lstc1+istc1+j

c South
             ihswc=lstc0+istc1+j-1
             ihsec=lstc0+istc0+j-1
             ilswc=lstc1+istc1+j-1
             ilsec=lstc1+istc0+j-1

c           finer mesh location parameters
c North
             ihneb=lst2+ist2+2*j-2
             ihnwb=lst2+ist3+2*j-2
             ilneb=lst3+ist2+2*j-2
             ilnwb=lst3+ist3+2*j-2

c South
             ihswb=lst2+ist3+2*j-3
             ihseb=lst2+ist2+2*j-3
             ilswb=lst3+ist3+2*j-3
             ilseb=lst3+ist2+2*j-3

c Modified by Matsfelt
c If the cell above the airfoil should not be affected
c  by the cell below the airfoil.
          if (i.le.iafp(klevc).and.j.eq.(jafp(klevc)+1)) then
c Prolongation to the finer mesh above the airfoil
c           coarser mesh location parameters
c North
             ihnec=lstc0+istc0+j
             ihnwc=lstc0+istc1+j
             ilnec=lstc1+istc0+j
             ilnwc=lstc1+istc1+j

c South
             ihswc=lstc0+istc1+j
             ihsec=lstc0+istc0+j
             ilswc=lstc1+istc1+j
```

```
            ilsec=lstc1+istc0+j

          end if
c Modified by Matsfelt

c prolongate the pressure from a coarse to a fine grid
c North
            p(ihnwb)=p(ihnwb)+(27.*p(ihnwc)+9.*p(ihnec)
     .           +9.*p(ihswc)+3.*p(ihsec)+9.*p(ilnwc)
     .           +3.*p(ilnec)+3.*p(ilswc)+1.*p(ilsec))/64.

            p(ihneb)=p(ihneb)+(9.*p(ihnwc)+27.*p(ihnec)
     .           +3.*p(ihswc)+9.*p(ihsec)+3.*p(ilnwc)
     .           +9.*p(ilnec)+1.*p(ilswc)+3.*p(ilsec))/64.

            p(ilnwb)=p(ilnwb)+(9.*p(ihnwc)+3.*p(ihnec)
     .           +3.*p(ihswc)+1.*p(ihsec)+27.*p(ilnwc)
     .           +9.*p(ilnec)+9.*p(ilswc)+3.*p(ilsec))/64.

            p(ilneb)=p(ilneb)+(3.*p(ihnwc)+9.*p(ihnec)
     .           +1.*p(ihswc)+3.*p(ihsec)+9.*p(ilnwc)
     .           +27.*p(ilnec)+3.*p(ilswc)+9.*p(ilsec))/64.


c Modified by Matsfelt
c If the cell above the airfoil should not be affected
c  by the cell below the airfoil.
          if (i.le.iafp(klevc).and.j.eq.(jafp(klevc)+1)) then
c Prolongation to the finer mesh below the airfoil
c          coarser mesh location parameters
c North
            ihnec=lstc0+istc0+j-1
            ihnwc=lstc0+istc1+j-1
            ilnec=lstc1+istc0+j-1
            ilnwc=lstc1+istc1+j-1

c South
            ihswc=lstc0+istc1+j-1
            ihsec=lstc0+istc0+j-1
            ilswc=lstc1+istc1+j-1
            ilsec=lstc1+istc0+j-1

          end if
```

```
c Modified by Matsfelt

c prolongate the pressure from a coarse to a fine grid
c South
            p(ihswb)=p(ihswb)+(9.*p(ihnwc)+3.*p(ihnec)
     .           +27.*p(ihswc)+9.*p(ihsec)+3.*p(ilnwc)
     .           +1.*p(ilnec)+9.*p(ilswc)+3.*p(ilsec))/64.

            p(ihseb)=p(ihseb)+(3.*p(ihnwc)+9.*p(ihnec)
     .           +9.*p(ihswc)+27.*p(ihsec)+1.*p(ilnwc)
     .           +3.*p(ilnec)+3.*p(ilswc)+9.*p(ilsec))/64.

            p(ilswb)=p(ilswb)+(3.*p(ihnwc)+1.*p(ihnec)
     .           +9.*p(ihswc)+3.*p(ihsec)+9.*p(ilnwc)
     .           +3.*p(ilnec)+27.*p(ilswc)+9.*p(ilsec))/64.

            p(ilseb)=p(ilseb)+(1.*p(ihnwc)+3.*p(ihnec)
     .           +3.*p(ihswc)+9.*p(ihsec)+3.*p(ilnwc)
     .           +9.*p(ilnec)+9.*p(ilswc)+27.*p(ilsec))/64.
         end do
         end do
         end do

c if not on the finest grid (=nlev), solve the pressure
         if(klev.ne.nlev) then
            call peter_relax(1,cycli,cyclj,cyclk)

         end if
  800 continue


      return
      end
```

F

CALC part of mg_2d.f file including modifications

```
      subroutine mg_2d(aw,ae,as,an,ap,t,p,sr,su,nlev,ni,nj,al,
     .                 ah,iplane,iwal1,iwal2,iafp,jafp,
     .                 cycli,cyclj,cyclk,wallj,taf)

      dimension  aw(*),ae(*),as(*),an(*),ap(*),t(*),p(*),sr(*),su(*)
     .     ,ist(1000),istc(1000),ni(*),nj(*),al(*),ah(*),iwal1(100),
     . iwal2(100),iafp(100),jafp(100)
      logical cycli,cyclj,cyclk,wallj,taf

c---- Initialization ...

c---- If the 3D-MG has reached the coarsest level ...

c---- V(1,1)-cycle without postsmoothing on the coarsest level ...
c---- Only one cycle is performed ...

c---- Bilinear restriction (from fine to coarse) ...


c---- Bilinear prolongation

      do klev=2,nlev
         call key2(klev,ist,ni(klev),nj(klev))
         klevc=klev-1
         call key2(klevc,istc,ni(klevc),nj(klevc))
```

```
          do i=1,ni(klevc)
                in=istc(i)+nj(klevc)
                inm1=istc(i)+nj(klevc)-1
                is=istc(i)+1
                ism1=istc(i)+2
                p(in)=p(inm1)
                p(is)=p(ism1)
          end do
          do j=1,nj(klevc)
                in=istc(ni(klevc))+j
                inm1=istc(ni(klevc)-1)+j
                is=istc(1)+j
                ism1=istc(2)+j
                p(in)=p(inm1)
                p(is)=p(ism1)
          end do

c... Periodic boundary conditions:

          call peter_2d_cyclic(p,klevc,istc,ni,nj,
     .                         iplane,iwal1,iwal2,cycli,cyclj,cyclk,wallj)

          do i=2,ni(klevc)
          istc0=istc(i)
          istc1=istc(i-1)
          ist2=ist(2*i-2)
          ist3=ist(2*i-3)
          do j=2,nj(klevc)


c           coarser mesh location parameters
c North
             ilnec=istc0+j
             ilnwc=istc1+j

c South
             ilswc=istc1+j-1
             ilsec=istc0+j-1


c           finer mesh location parameters
c North
```

```
            ilneb=ist2+2*j-2
            ilnwb=ist3+2*j-2

c South
            ilswb=ist3+2*j-3
            ilseb=ist2+2*j-3


c Modified by Matsfelt
c The prolongation are not allowed to cross the limits of the airfoil
c  Special cases for each plane:
c    iplane=1 => z-y plane, taf true x value ok i.e. airfoil, check y
c    iplane=2 => x-z plane, not affected by the airfoil configuration
c    iplane=3 => x-y plane, check here if x and y is at the airfoil

          if (iplane.ne.2) then
       if ((iplane.eq.1.and.taf.and.j.eq.(jafp(klevc)+1)).or.
      .(iplane.eq.3
      ..and.i.le.iafp(klevc).and.j.eq.(jafp(klevc)+1))) then

c           coarser mesh location parameters
c North
            ilnec=istc0+j
            ilnwc=istc1+j

c South
            ilswc=istc1+j
            ilsec=istc0+j

         end if
         end if
c Modified by Matsfelt


c prolongate the pressure from a coarse to a fine grid
c North
            p(ilnwb)=p(ilnwb)+(9.*p(ilnwc)
     .           +3.*p(ilnec)+3.*p(ilswc)+1.*p(ilsec))/16.

            p(ilneb)=p(ilneb)+(3.*p(ilnwc)
     .           +9.*p(ilnec)+1.*p(ilswc)+3.*p(ilsec))/16.
```

```
c Modified by Matsfelt
c The prolongation are not allowed to cross the limits of the airfoil
c  Special cases for each plane:
c    iplane=1 => z-y plane, taf true x value ok i.e. airfoil, check y
c    iplane=2 => x-z plane, not affected by the airfoil configuration
c    iplane=3 => x-y plane, check here if x and y is at the airfoil

          if (iplane.ne.2) then
       if ((iplane.eq.1.and.taf.and.j.eq.(jafp(klevc)+1)).or.
      . (iplane.eq.3
      ..and.i.le.iafp(klevc).and.j.eq.(jafp(klevc)+1))) then

c           coarser mesh location parameters
c North
            ilnec=istc0+j-1
            ilnwc=istc1+j-1

c South
            ilswc=istc1+j-1
            ilsec=istc0+j-1

         end if
         end if
c Modified by Matsfelt


c prolongate the pressure from a coarse to a fine grid
c South
            p(ilswb)=p(ilswb)+(3.*p(ilnwc)
     .           +1.*p(ilnec)+9.*p(ilswc)+3.*p(ilsec))/16.

            p(ilseb)=p(ilseb)+(1.*p(ilnwc)
     .           +3.*p(ilnec)+3.*p(ilswc)+9.*p(ilsec))/16.
         end do
         end do
         if(klev.ne.nlev)call peter_2d_relax(klev,maxa,aw,ae
     .      ,as,an,ap,t,p,sr,ist,ni,nj,iplane,iwal1,iwal2,cycli,cyclj,
     .       cyclk,wallj)
       end do
       end do

       return
       end
```

G

CALC part of peter_relax.f file including modifications

```
      subroutine peter_relax(mmm,cycli,cyclj,cyclk)

      include 'PETER_COMMON'
      logical cycli,cyclj,cyclk,wallj,taf
      common /peter/kst(100),kst_2d(100)
      dimension bw(io2dmax),be(io2dmax),bs(io2dmax),bn(io2dmax)
      dimension bp(io2dmax),bsu(io2dmax),btry(io2dmax)
      dimension bl(io2dmax),bh(io2dmax)
      dimension sl1(io2dmax),sl2(io2dmax)
      dimension a(1000),b(1000),c(1000),d(1000)
      dimension pnew(iomax)
      integer*4 lock(0:1000)  ! should be max(it,jt,kt)

      maxit2=mmm
      if(klev.eq.1.and.nlev.gt.1)maxit2=25

      do 100 iter=1,maxit2

      call peter_cyclic(ist,lst,ni(klev),nj(klev),nk(klev),
     .                  klev,cycli,cyclj,cyclk)

c----- Symmeteric point-gs (forward and backward sweep) ...

c----- X-line-gs ...
```

```
c------ Y-line-gs ...

c------ Z-line-gs ...


c------ X-plane-gs

      if(sol(5).eq.1)then

      call key(klev,lst,ist,ni(klev),nj(klev),nk(klev))
      kst_2d(1)=1


      call peter_cyclic(ist,lst,ni(klev),nj(klev),nk(klev),
     .                  klev,cycli,cyclj,cyclk)

      do i=2,ni(klev)-1
         icou=0
         do kl=1,klev
            kst_2d(kl+1)=kst_2d(kl)+nj(kl)*nk(kl)
         end do
         isti=ist(i)
         istip=ist(i+1)
         istim=ist(i-1)
         do k=1,nk(klev)
         lstk=lst(k)
         icoutmp=kst_2d(klev)-1 + (k-1)*nj(klev)
         do j=1,nj(klev)
            icou = icoutmp + j
            io=lstk+isti+j
            iw=lstk+istim+j
            ie=lstk+istip+j
            bw(icou)=al(io)
            be(icou)=ah(io)
            bs(icou)=as(io)
            bn(icou)=an(io)
            bp(icou)=ap(io)
            bl(icou)=aw(io)
            bh(icou)=ae(io)
            bsu(icou)=t(io)+sr(io)+ae(io)*p(ie)+aw(io)*p(iw)
            btry(icou)=p(io)
         end do
         end do
```

```
c Modified by Matsfelt
c    If the i coordinate show that the airfoil will be part
c        of the set taf to true for the in z-y plane.
         if (i.le.iafp(klev).and.j.eq.(jafp(klev)+1)) then
            taf=.true.
         else
            taf=.false.
         end if
c Modified by Matsfelt

         call mg_2d(bw,be,bs,bn,bp,bsu,btry,sl1,sl2,klev,
     .              nk,nj,bl,bh,1,iwal1,iwal2,iafp,jafp,
     .              cycli,cyclj,cyclk,wallj,taf)

         kl=klev
         do k=nk(kl),1,-1
         lstk=lst(k)
         icoutmp=kst_2d(klev)-1 + (k-1)*nj(klev)
         do j=1,nj(kl),1
             icou = icoutmp + j
             io=lstk+isti+j
             p(io)=btry(icou)
         end do
         end do
       end do

       call peter_cyclic(ist,lst,ni(klev),nj(klev),nk(klev),
     .                   klev,cycli,cyclj,cyclk)

       end if


c------ Y-plane-gs

       if(sol(6).eq.1)then
        call key(klev,lst,ist,ni(klev),nj(klev),nk(klev))
        kst_2d(1)=1
       do j=2,nj(klev)-1
          icou=0
          do kl=1,klev
             kst_2d(kl+1)=kst_2d(kl)+ni(kl)*nk(kl)
```

```
          end do
          kl=klev

          do i=1,ni(klev)
          isti=ist(i)
          icoutmp=kst_2d(klev)-1 + (i-1)*nk(klev)
          do k=1,nk(klev)
             lstk=lst(k)
             icou=icoutmp + k
             io=lstk+isti+j
             in=io+1
             is=io-1
             bw(icou)=aw(io)
             be(icou)=ae(io)
             bs(icou)=al(io)
             bn(icou)=ah(io)
             bp(icou)=ap(io)
             bl(icou)=as(io)
             bh(icou)=an(io)
             bsu(icou)=t(io)+sr(io)+an(io)*p(in)+as(io)*p(is)
             btry(icou)=p(io)
          end do
          end do

c Modified by Matsfelt
c    The prolongation in the x-z plane in mg_2d will not be
c         affected by the airfoil configuration. The specified taf
c         value has no importance only set to have it initialized.
          taf=.false.

          call mg_2d(bw,be,bs,bn,bp,bsu,btry,sl1,sl2,klev,
     .               ni,nk,bl,bh,2,iwal1,iwal2,iafp,jafp,
     .               cycli,cyclj,cyclk,wallj,taf)
c Modified by Matsfelt

          kl=klev
          do i=1,ni(kl),1
          isti=ist(i)
          icoutmp=kst_2d(klev)-1 + (i-1)*nk(klev)
          do k=1,nk(kl),1
             lstk=lst(k)
             io=lstk+isti+j
             icou=icoutmp + k
```

```
            p(io)=btry(icou)
          end do
          end do
        end do

        call peter_cyclic(ist,lst,ni(klev),nj(klev),nk(klev),
     .                    klev,cycli,cyclj,cyclk)

        end if


c------ Z-plane-gs

        if(sol(7).eq.1)then
         call key(klev,lst,ist,ni(klev),nj(klev),nk(klev))
         kst_2d(1)=1
        do k=2,nk(klev)-1
          icou=0
          do kl=1,klev
            kst_2d(kl+1)=kst_2d(kl)+ni(kl)*nj(kl)
          end do

          lstk=lst(k)
          lstkm=lst(k-1)
          lstkp=lst(k+1)

          do i=1,ni(klev)
          isti=ist(i)
          icoutmp = kst_2d(klev)-1+(i-1)*nj(klev)
          do j=1,nj(klev)
            icou=icoutmp+j
            io=lstk+isti+j
            ih=lstkp+isti+j
            il=lstkm+isti+j
           bw(icou)=aw(io)
            be(icou)=ae(io)
            bs(icou)=as(io)
            bn(icou)=an(io)
            bl(icou)=al(io)
            bh(icou)=ah(io)
            bp(icou)=ap(io)
            bsu(icou)=t(io)+sr(io)+ah(io)*p(ih)+al(io)*p(il)
            btry(icou)=p(io)
```

```
          end do
          end do

c Modified by Matsfelt
c     The mg_2d code will loop over the whole x-y plane and on its
c         own feel if the airfoil is part of the prolongation or
c         not. The specified taf value has no importance only set
c         to have it initialized.
             taf=.true.

          call mg_2d(bw,be,bs,bn,bp,bsu,btry,sl1,sl2,klev,
     .                ni,nj,bl,bh,3,iwal1,iwal2,iafp,jafp,
     .                cycli,cyclj,cyclk,wallj,taf)
c Modified by Matsfelt

          kl=klev
          do i=1,ni(kl),1
          isti=ist(i)
          icoutmp = kst_2d(klev)-1+(i-1)*nj(klev)
          do j=1,nj(kl),1
             io=lstk+isti+j
             icou=icoutmp + j
             p(io)=btry(icou)
          end do
          end do
        end do

       call peter_cyclic(ist,lst,ni(klev),nj(klev),nk(klev),
     .                   klev,cycli,cyclj,cyclk)

        end if


 100   continue
       return
       end
```

H

CALC distance to airfoil calculation in the mod.f file

```
      subroutine modify
      include 'COMMON'
      include 'CASECOM'
      include 'MOD_COM'
      dimension utemp(jt,kt),vtemp(jt,kt),wtemp(jt,kt)

      common/inlet1/umean(jt),rkmean_in(jt),epsmean_in(jt),
     .   ufluct(jt,kt),vfluct(jt,kt),wfluct(jt,kt),vmean(jt),
     .   u2_prim,uv_fluct(jt),uv_rans(jt),uu_synt2d(jt),vv_synt2d(jt),
     .   ww_synt2d(jt),uv_synt2d(jt),ommean_in(jt)

      common/cmean/umeanv(it,jt),vmeanv(it,jt),umean2v(it,jt),
     .             vmean2v(it,jt),wmean2v(it,jt),ss_u


      character filny*140,test*140,name_end*110

c deltan2d related variables
      dimension xTop(it),yTop(it)
      integer countSize,count,ifound
      logical flag,flag1


c-- section 0 ------initialization ------------------------------------
      entry modini
```

```
c Modified by Matsfelt
! Calculates the closest distance from each cell center to the
!  airfoil is needed in vist_pans.f and calced_pans to run RANS
!  the original variable name is dist but is here named deltan2d

      diff2org=2.937 !Diff in the top equation has to move the curve.
      k=nk/2

      movedivide=1000. ! Factor to divide the moving distance with
      countMax=100. ! Number of guesses to find closest
                    ! location on the curve


! Take the x and y values of the top of the airfoil to be used in
! linear interpolation between.

      countSize=1
      do i=2,(iaf+1)
        if (xc(i,jaf,k).gt.(-diff2org)) then
          xTop(countSize)=xc(i-1,jaf,k)
          yTop(countSize)=yc(i-1,jaf,k)

          countSize=countSize+1

        end if
      end do

      countSize=countSize-1

       do i=1,ni
           do j=1,nj
               count=1
               flag=0
              ! Default 0, set to 1 for locations where no iteration
              ! is needed

               ysource=yp(i,j,k)
               if (j.eq.jaf.and.i.le.iaf) then
                   ! The real y value of the nodes just above the
                   ! airfoil is y2d(iaf,jaf)
                   ysource=yp(iaf,jaf,k)
               end if
```

```
      do while (count.le.countMax.and.flag.eq.0)

         if (count.eq.1) then
            xn=xp(i,j,k)     ! First guess
            ifound=2
         elseif (count.eq.2) then
            ! Second guess move 1/movedivide along the
            ! positive x axis
            deltanold=deltan
            xn=xn+1/(count*movedivide)
         else
            xn=(deltanold-deltan)/(count*movedivide)+xn
            deltanold=deltan
        end if

         if (ysource.ge.yc(iaf,jaf,k)) then ! Top equation:
            if (xn.lt.(-diff2org)) then
               ! The top surface of the airfoil
               yn=yc(1,jaf,k)
            elseif (xn.le.xTop(countSize)) then
               ! The curved surface of the airfoil
               ! Find the location where xn is larger
               !  than the value of xTop to the
               !  left and larger than that
               !  to the right.
               flag1=0
             do while (ifound.le.countSize.and.flag1.eq.0)
                  if (xTop(ifound-1).le.xn
     .               .and.xTop(ifound).ge.xn) then
                   ! If the xn value is located in
                   !  between the xTop values.
                    yn=((yTop(ifound)-yTop(ifound-1))/
     .                 (xTop(ifound)-xTop(ifound-1)))*
     .                 (xn-xTop(ifound-1))+yTop(ifound-1)
                    flag1=1
                  end if
                  if (flag1.eq.0) then
                     if (xTop(ifound-1).lt.xn
     .                  .and.ifound.ne.countSize) then
                        ifound=ifound+1
                     else if (ifound.eq.2) then
                     ! The ifound value is not allowed
```

```
                                !  to be lower than 2 so
                                !  the same ifound value
                                !  is used in the next guess.
                                else if (ifound.eq.countSize) then
                                ! To allow the the last number
                                !  to be guessed again
                                else
                                    ifound=ifound-1
                                end if
                            end if
                        end do
                    else if (xn.gt.xTop(countSize)) then
                        ! The section above and behind the airfoil
                        xn=xTop(countSize)
                        yn=yTop(countSize)
                    end if
                else if ((ysource.gt.yc(iaf,jaf-1,k))
     .               .and.(xp(i,j,k).gt.xc(iaf,jaf-1,k))) then
                        ! Side equation
                        yn=ysource
                        ! The shortest distance to the location
                        !  will allways have the same y value.
                         xn=xTop(countSize)
                         flag=1
                else  ! Bottom equation
                    yn=yc(iaf,jaf,k)
                    ! The real location of the bottom
                    !  of the airfoil
                    if (xp(i,j,k).gt.xc(iaf,jaf-1,k)) then
                        ! The location has a x value larger
                        ! than the arifoil
                        xn=xc(iaf,jaf-1,k)
                    end if
                    flag=1
                end if

                 deltan=(((abs(xp(i,j,k)-xn))**2.
     .                   +(abs(ysource-yn))**2.))**0.5
                    ! Differnce between the predicted location
                    !  and the i,j.

                if (count.eq.1) then
                    deltan2d(i,j)=deltan
```

```
                    else
                        if (deltan2d(i,j).lt.deltan) then
                        ! if the new guessed deltan value is lower
                        !  than that specified in deltan2d put it in
                        !  deltan2d.
                            deltan2d(i,j)=deltan
                        end if
                    end if
                   count=count+1
                end do
            end do
        end do
c Modified by Matsfelt


c ...
```

I

CALC dphidy.f file including modifications

```
      function dphidy(i,j,k,nphi)
      include 'COMMON'

c the derivative dphidy at node i,j,k for variable nphi is calculated here

      phie=fx(i,j,k)*phi(i+1,j,k,nphi)+(1.-fx(i,j,k))*phi(i,j,k,nphi)
      phiw=fx(i-1,j,k)*phi(i,j,k,nphi)+(1.-fx(i-1,j,k))
     &     *phi(i-1,j,k,nphi)
      phin=fy(i,j,k)*phi(i,j+1,k,nphi)+(1.-fy(i,j,k))*phi(i,j,k,nphi)
      phis=fy(i,j-1,k)*phi(i,j,k,nphi)+(1.-fy(i,j-1,k))
     &     *phi(i,j-1,k,nphi)
      phih=fz(i,j,k)*phi(i,j,k+1,nphi)+(1.-fz(i,j,k))*phi(i,j,k,nphi)
      phil=fz(i,j,k-1)*phi(i,j,k,nphi)+(1.-fz(i,j,k-1))
     &     *phi(i,j,k-1,nphi)


      dphidy=(phie*areaey(i,j,k)-phiw*areaey(i-1,j,k)
     &       +phin*areany(i,j,k)-phis*areany(i,j-1,k)
     &       +phih*areahy(i,j,k)-phil*areahy(i,j,k-1))/vol(i,j,k)

c Modified by Matsfelt
c To break the connection between the cell jaf+1 above the airfoil
c   and jaf below.
      if (i.le.iaf.and.j.eq.jaf) then
```

```
      phin=0.

      dphidy=(phie*areaey(i,j,k)-phiw*areaey(i-1,j,k)
     &       +phin*areany(i,j,k)-phis*areany(i,j-1,k)
     &       +phih*areahy(i,j,k)-phil*areahy(i,j,k-1))/vol(i,j,k)


      elseif (i.le.iaf.and.j.eq.(jaf+1)) then

      phis=0.

      dphidy=(phie*areaey(i,j,k)-phiw*areaey(i-1,j,k)
     &       +phin*areany(i,j,k)-phis*areanyjafp1(i,j-1,k)
     &       +phih*areahy(i,j,k)-phil*areahy(i,j,k-1))
     &       /vol(i,j,k)

      end if
c Modified by Matsfelt

      return
      end
```

J

CALC part of ADM mod.f file

```
      subroutine modify
      include 'COMMON'
      include 'CASECOM'
      include 'MOD_COM'

      integer, parameter :: nSpanLoc = 19, nBlade = 3,
     1 NSPANM = 30, NANGM = 365, NMACHM = 3
      double precision, parameter :: rHub = 1.50d0, rTip = 63.00d0,
     1 vInf = 8.0d0, rho = 1.225d0,
     1 soundSpeed = 340.0d0, pitch = 0.0d0
      double precision, dimension(nSpanLoc) :: rSpan, cSpan, tSpan
      double precision, dimension(jt) :: rNode, afa, tfa
      double precision :: temp, um, wm, axialForce, tangentForce, omega
      double precision :: rcp, twist, chord, AngleOfAttack
      integer :: NSPANB, NMACHB, counter
      integer, dimension(NSPANM) :: NANGB
      double precision, dimension(NMACHM) :: AMACHTB
      double precision, dimension(NSPANM) :: RCB
      double precision, dimension(NSPANM, NANGM) :: AATB
      double precision, dimension(NSPANM, NMACHM, NANGM) :: CLTB,CDTB,
     1 CMTB

      integer counterj

c     ADM varibale for mod.f
      common/admmod/imid,jmin,jmax,kmin,kmax,xp1,xpWT,
     .              csmear,nksec,avgk,phiavgu(jt),phiavgtheta(jt),
```

```
     .                thetayz(jt,kt),Fyadm(jt,kt),Fzadm(jt,kt),
     .                testtheta,vv,vw,testr,testdiffrtheta,
     .                phiavgr(jt),diffphiavgrphiavgtheta(jt)

c      Gaussian smearing related variables
       common/gauss/imiddiff,gaussianmag(jt,kt),
     .              gaussianmagcheck(jt,kt)


c- section 0 ------initialization ------------------------------------------
       entry modini

c Angular velocity of the rotation of the blades.
       omega = 1.003d0  !  [rad/s]

c      Gaussian smearing nodes
       imiddiff=50


c      Wind turbine location
c      find imid, the i location of the wind turbine
       do i=1,ni
          if (xp(i,1,1).le.0.and.xp(i+1,1,1).ge.0) ii=i
       end do
       imid=ii

       xpWT = 0.5*(xc(imid,1,1)+xc(imid-1,1,1))  ! streamwise position

c      Rotor dimensions
       jmin=2

c      Find jmax from the defined mesh
       hub=63.
       do j=1,nj
          if (yp(1,j,1).le.hub.and.yp(1,j+1,1).ge.hub) jj=j
       end do
       jmax=jj

       kmin=2
       kmid=nk/2
       kmax=nkm1
```

```
c     The number of k positions that are averaged over.
      avgk=kmax-kmin+1

c     Smearing function (convolution)
      csmear = 2.0

c     Write the center locations of the nodes of the
      open(unit=50,file='controlNode.inp',status='unknown')
      do j=jmin,jmax
        write(50,*)yp(1,j,1)
      end do
      close(50)

c     Number of nodes on the blad
      nBladeNodes=jmax-jmin+1

c     Number of sections that the force should be applied on
      nksec=nk-2

c     Approximation of pi Machin-like formulae
      pi=4.*(4.*atan(1./5.)-atan(1./239.))

      do j=jmin,jmax
        counterj=j-jmin+1
      do k=kmin,kmax
c     Tangential velocity,using polar coordinates conversion in y-z plane
c     Find Theta angle [rad]
        if (yp(imid,j,k).gt.0.0) then
          thetayz(j,k)=atan(zp(imid,j,k)/yp(imid,j,k))
        else if (yp(imid,j,k).lt.0.0.and.zp(imid,j,k).ge.0.0) then
          thetayz(j,k)=atan(zp(imid,j,k)/yp(imid,j,k))+pi
        else if (yp(imid,j,k).lt.0.0.and.zp(imid,j,k).lt.0.0) then
          thetayz(j,k)=atan(zp(imid,j,k)/yp(imid,j,k))-pi
        else if (yp(imid,j,k).eq.0.0.and.zp(imid,j,k).gt.0.0) then
          thetayz(j,k)=pi/2
        else if (yp(imid,j,k).eq.0.0.and.zp(imid,j,k).lt.0.0) then
          thetayz(j,k)=-pi/2
        else if (yp(imid,j,k).eq.0.0.and.zp(imid,j,k).eq.0.0) then
          thetayz(j,k)=0.
          write(6,*)'Theta angle in mod.f is not defined at (0,0)'
        end if
      end do
      end do
```

```
!-----------------------------------------------------------------------
! The aerodynamic characteristics of blade are read.
! rSpan: number of blade spanwise section including airfpoil profile
! cspan: local chord length [m]
! tSpan: local twist [degree]
!-----------------------------------------------------------------------
      open(1, file = 'aeroData.inp')
      do i = 1, nSpanLoc
         read(1,*) rSpan(i), cSpan(i), tSpan(i), temp
      end do
      close(1)

!-----------------------------------------------------------------------
! The blade control points are read. (Blade spanwise discretization)
!-----------------------------------------------------------------------
      open(2, file = 'controlNode.inp')
      do i = 1, nBladeNodes
         read(2,*) rNode(i)
      end do
      close(2)

!-----------------------------------------------------------------------
! The tabulated airfoil data, CL, CD and CM are read.
!-----------------------------------------------------------------------
        open(3, file = 'CLCDCMTable.inp' )
        read(3,*)
        read(3,*)
        read(3,*)
        read(3,*) NSPANB
        read(3,*) NMACHB

        write(6,*)'modini NMACHB',NMACHB
        flush(6)

        do k = 1, NMACHB
           read  (3,*)  AMACHTB (k)
        enddo

        read  (3,*)
        do j = 1, NSPANB
         read(3,*) RCB(j)
         read(3,*) NANGB(j)
```

```
          read(3,*)
          do i = 1, NANGB (j)
             read(3,*) AATB (j,i), (CLTB (j,k,i), CDTB (j,k,i),
     1                  CMTB (j,k,i), k=1, NMACHB)
        end do
          read(3,*)
        end do
          close(3)

          counter = maxval(NANGB)

        return

c- section 1 ------properties-----------------------------------------------

c- section 2 ------convection-----------------------------------------------

c- section 3 ------ u-velocity-----------------------------------------------
        entry modu

c ---------------------------------------------------------------------------
c     ACTUATOR DISK MODEL
c ---------------------------------------------------------------------------
c  Average the velocities at all the k levels because of the axisymmetry
c  convert u,v,w velocity components to axial and tangential components.
        do j=jmin,jmax
          counterj=j-jmin+1

          phiavgu(counterj)=0.       ! initialize
          phiavgtheta(counterj)=0.   ! initialize
          phiavgr(counterj)=0.       ! initialize

        do k=kmin,kmax

c         Axial velocity
          phiavgu(counterj)=phiavgu(counterj)+phi(imid,j,k,u)

c         Tangential velocity
          phiavgtheta(counterj)=phiavgtheta(counterj)
     .                         -phi(imid,j,k,v)*sin(thetayz(j,k))
     .                         +phi(imid,j,k,w)*cos(thetayz(j,k))

        end do
```

```
c         The average over the k sections
          phiavgu(counterj)=phiavgu(counterj)/avgk
          phiavgtheta(counterj)=phiavgtheta(counterj)/avgk

        end do

!-------------------------------------------------------------------------
! This part calculates the aerodynamic forces.
!-------------------------------------------------------------------------

        do j = 1, nBladeNodes
         counterj=jmin+j-1

         um = phiavgu(j)
         wm = phiavgtheta(j)
         rcp = rNode(j)

! The 'interp' subroutine calculate the local chord and twist
! by interpolation from the aerodynamic characteristic table.
          call interp(nSpanLoc, rcp, rSpan, cSpan, tSpan, chord, twist)

          AngleOfAttack=0.0
! The 'force' subroutine calculates the axial and tangential force at
! each spanwise section [N/m2].
          call force(um, wm, omega, nBlade, rho, rHub, rTip, soundSpeed,
       1 pitch, rcp, twist, chord, counter, NSPANM, NMACHM, NANGM,NSPANB,
       1 NMACHB, NANGB, RCB, AATB, AMACHTB, CLTB, CDTB, CMTB, axialForce,
       1 tangentForce,AngleOfAttack)

c Convert from circule to a regular polynom
          axialForce=axialForce*pi/
       . (8.*(1.+((2.)**(1./2.)))*((sin(pi/float(nksec)))**2.))

          tangentForce=tangentForce*pi/
       . (8.*(1.+((2.)**(1./2.)))*((sin(pi/float(nksec)))**2.))

         afa(j)=axialForce          ! [N/m2]
         tfa(j)=tangentForce        ! [N/m2]

c Have the forces in [N] by multipling by the area sections
         afa(j)=afa(j)*areaex(imid,counterj,kmin)      ! [N]
         tfa(j)=tfa(j)*areaex(imid,counterj,kmin)      ! [N]
```

```
      end do

c Convert from tangential force to the v and w component.
      do j=jmin,jmax
      counterj=j-jmin+1
      do k=kmin,kmax

c       v component
        Fyadm(j,k)=-tfa(counterj)*sin(thetayz(j,k))

c       w component
        Fzadm(j,k)=tfa(counterj)*cos(thetayz(j,k))

      end do
      end do


c Check that the Gaussian has a total value of one
      do j=jmin,jmax
      counterj=j-jmin+1
      do k=kmin,kmax
      gaussianmag(counterj,k)=0.
      do i=imid-imiddiff,imid+imiddiff

        xp1         = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx          = xc(i,1,1)-xc(i-1,1,1)

c       Streamwise distance to wind turbine
        dstream     = sqrt((xp1-xpWT)**2)

c       Epsilon
        epsi        = csmear*dx

c       1D Gaussian function
        eta_eps     = 1./(epsi*pi**0.5)*exp(-(dstream/epsi)**2)

c       Original magnitude of the local sum gaussian function
        gaussianmag(counterj,k)=gaussianmag(counterj,k)+
     .              eta_eps*dx

      end do
      end do
```

```
      end do


c Apply the force
      fmax=0.
      do j=jmin,jmax
      counterj=j-jmin+1
      do k=kmin,kmax
      gaussianmagcheck(counterj,k)=0.
      fsum=0.0
      do i=imid-imiddiff,imid+imiddiff

        xp1         = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx          = xc(i,1,1)-xc(i-1,1,1)

c       Streamwise distance to wind turbine
        dstream     = sqrt((xp1-xpWT)**2)

c       Epsilon
        epsi        = csmear*dx

c       1D Gaussian function
        eta_eps     = 1./(epsi*pi**0.5)*exp(-(dstream/epsi)**2)

c       Axial force
        fp          = afa(counterj)

c       Apply the gaussian and normalize to have a sum of 1
        fdist       = fp*eta_eps*dx/gaussianmag(counterj,k)
        su(i,j,k) = su(i,j,k) - fdist

c       Sum gaussian function to ensure that the sum is one
        gaussianmagcheck(counterj,k)=gaussianmagcheck(counterj,k)+
     .                  eta_eps*dx/gaussianmag(counterj,k)

      end do
      end do
      end do

      return

c- section 4 ------v-velocity-----------------------------------------
      entry modv
```

```
c ------------------------------------------------------------------------
c       ACTUATOR DISK MODEL
c ------------------------------------------------------------------------

        fmax=0.
        do j=jmin,jmax
        counterj=j-jmin+1
        do k=kmin,kmax
        gaussianmagcheck(counterj,k)=0.
        fsum=0.0
        do i=imid-imiddiff,imid+imiddiff

          xp1          = 0.5*(xc(i,1,1)+xc(i-1,1,1))
          dx           = xc(i,1,1)-xc(i-1,1,1)

c         Streamwise distance to wind turbine
          dstream      = sqrt((xp1-xpWT)**2)

c         Epsilon
          epsi         = csmear*dx

c         1D Gaussian function
          eta_eps      = 1./(epsi*pi**0.5)*exp(-(dstream/epsi)**2)

c         v component of the tangential force
          fp           = Fyadm(j,k)

c         Apply the gaussian and normalize to have a sum of 1
          fdist        = fp*eta_eps*dx/gaussianmag(counterj,k)
          su(i,j,k) = su(i,j,k) - fdist

        end do
        end do
        end do

        return

c- section 5 -------w-velocity-------------------------------------------
        entry modw

c ------------------------------------------------------------------------
c       ACTUATOR DISK MODEL
```

```
c ------------------------------------------------------------------------

      fmax=0.
      do j=jmin,jmax
      counterj=j-jmin+1
      do k=kmin,kmax
      gaussianmagcheck(counterj,k)=0.
      fsum=0.0
      do i=imid-imiddiff,imid+imiddiff

        xp1          = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx           = xc(i,1,1)-xc(i-1,1,1)

c       Streamwise distance to wind turbine
        dstream      = sqrt((xp1-xpWT)**2)

c       Epsilon
        epsi         = csmear*dx

c       1D Gaussian function
        eta_eps      = 1./(epsi*pi**0.5)*exp(-(dstream/epsi)**2)

c       w component of the tangential force
        fp           = Fzadm(j,k)

c       Apply the gaussian and normalize to have a sum of 1
        fdist        = fp*eta_eps*dx/gaussianmag(counterj,k)
        su(i,j,k) = su(i,j,k) - fdist

      end do
      end do
      end do

      return

c ...
```

K

CALC ADM force.f file

```
      subroutine force(u, w, rotVel, nb, airDen, hubRad, tipRad, snVel,
     . bPtc, rLoc, locTwist, locChord, maxCounter, nSpanG, nMachG,
     . nAngleG, nSpanTable, nMachTable, nAngleTable, rLocTable,
     . aoaTable, machTable, clTable, cdTable, cmTable, Fn, Ft, aoa)
!-----------------------------------------------------------------------
      implicit none
!-----------------------------------------------------------------------
      double precision, parameter :: nacelleDragCoeff = 0.50d0
      integer :: nb, nSpanG, nMachG, nAngleG, nSpanTable, nMachTable,
     . maxCounter
      integer, dimension(nSpanG) :: nAngleTable
      double precision :: rotVel, airDen, hubRad, tipRad, snVel, bPtc,
     . rLoc, locTwist, locChord, Fn, Ft, u, w, CL, CD, CM, pi, criteria,
     . flowAngle, aoa, liftCoeff, dragCoeff, normalCoeff, tangentCoeff,
     . sigma, fTipCorr, fHubCorr, Ftip, Fhub, Ftot, aInd, apInd, Vx, Vt,
     . VxUpdate, VtUpdate, deltaVx, deltaVt, vTotal2, machNumber,
     . aIndUpdate, apIndUpdate, deltaaInd, deltaapInd,
     . acInd, KaInd
      double precision, dimension(nMachG) :: machTable
      double precision, dimension(nSpanG) :: rLocTable
      double precision, dimension(nAngleG) :: aoaInterp, clInterp,
     . cdInterp, cmInterp
      double precision, dimension(nSpanG, nAngleG) :: aoaTable
      double precision, dimension(nSpanG,nMachG,nAngleG) ::
     . clTable, cdTable, cmTable
!-----------------------------------------------------------------------
      Vx = u
```

```
! [m/s], from Navier-Stokes solution, axial velocity perpendicular to
! the rotor plane
        Vt = w
! [m/s], from Navier-Stokes solution, tangential velocity perpendicular
! to the rotor plane

        pi = 4.0d0 * atan(1.0d0)

          if (rLoc .le. hubRad) then
             Ft = 0.0d0
             Fn = 0.50d0 * airDen * (u ** 2) * nacelleDragCoeff
             goto 20
          endif
! For the radii smaller than hub radius, the constant drag coefficient
! for the nacelle (Cd_nacelle=0.5) is considered. So, in that case,
! only the axial force acts on the actuator disk.
! The dimension of the Ft and Fn are [N/m2].
!-----------------------------------------------------------------------
         flowAngle = atand(Vx / abs(-rotVel * rLoc - Vt))
! [degree], flow angle calculation based on the axial and tangential
! velocities.
        aoa = flowAngle - (locTwist + bPtc)
! aoa [degree], angle of attack calculation by means of flow angle,
! local twist and blade pitch angle.
!-----------------------------------------------------------------------
! Lift and drag coefficients based on the linear airfoil theory.
!-----------------------------------------------------------------------
!        liftCoeff = 2.0d0 * pi * (aoa * pi / 180.0d0)
!        dragCoeff = 0.0d0
!-----------------------------------------------------------------------
        machNumber = sqrt((Vx ** 2.0d0) + ((-rotVel * rLoc - Vt)
     .              ** 2.0d0)) / snVel
! machNumber is the Mach number which is used when reading and
!interpolating airfoil data.
!-----------------------------------------------------------------------
          call aeroTable(rLoc, machNumber, rLocTable, machTable, aoaTable,
     .       clTable, cdTable, cmTable, nSpanG, nMachG, nAngleG,
     .       nSpanTable, nMachTable, nAngleTable, clInterp, cdInterp,
     .       cmInterp, aoaInterp)

          call foil_clcdcm(aoa, maxCounter, nAngleG, aoaInterp,
     .       clInterp, cdInterp, cmInterp, CL, CD, CM)
```

```
! These two subroutines calculate lift (CL) and drag (CD) coefficients
! for each spanwise section.
!-----------------------------------------------------------------------
          liftCoeff = CL ! Lift coefficient for each spanwise section.
          dragCoeff = CD ! Drag coefficient for each spanwise section.

        normalCoeff = (liftCoeff * cosd(flowAngle))
     .              + (dragCoeff * sind(flowAngle))
! normalCoeff is the projection of lift and drag coefficients into the
! normal direction (perpendicular to the rotor plane, Axial direction)

        tangentCoeff = (liftCoeff * sind(flowAngle))
     .               - (dragCoeff * cosd(flowAngle))
! tangentCoeff is the projection of lift and drag coefficients into the
! tangential direction (tangential to the rotor plane)
!-----------------------------------------------------------------------
        vTotal2 = Vx ** 2.0d0 + (abs(-rotVel * rLoc - Vt)) ** 2.0d0
! The magnitude of the total velocity is computed.
!-----------------------------------------------------------------------
        Fn = (1.0d0 / (2 * pi * rLoc)) * 0.50d0 * airDen * vTotal2
     .     * locChord * nb * normalCoeff
! [N/m2], The axial force (perpendicular to the rotor plane).
        Ft = (1.0d0 / (2 * pi * rLoc)) * 0.50d0 * airDen * vTotal2
     .     * locChord * nb * tangentCoeff
! [N/m2], The tangential force (tangential to the rotor plane).

20    continue

        return
        end subroutine
```

L

CALC part of ALM mod.f file

```
      subroutine modify
      include 'COMMON'
      include 'CASECOM'
      include 'MOD_COM'

      integer, parameter :: nSpanLoc = 19, nBlade = 3,
     1 NSPANM = 30, NANGM = 365, NMACHM = 3
      double precision, parameter :: rHub = 1.50d0, rTip = 63.00d0,
     1 vInf = 8.0d0, rho = 1.225d0,
     1 soundSpeed = 340.0d0, pitch = 0.0d0
      double precision, dimension(nSpanLoc) :: rSpan, cSpan, tSpan
      double precision, dimension(jt) :: rNode
      double precision, dimension(jt,3) :: afa, tfa
      double precision :: temp, um, wm, axialForce, tangentForce, omega
      double precision :: rcp, twist, chord, AngleOfAttack, magcheck
      integer :: NSPANB, NMACHB, counter
      integer, dimension(NSPANM) :: NANGB
      double precision, dimension(NMACHM) :: AMACHTB
      double precision, dimension(NSPANM) :: RCB
      double precision, dimension(NSPANM, NANGM) :: AATB
      double precision, dimension(NSPANM, NMACHM, NANGM) :: CLTB,CDTB,
     1 CMTB

      integer counterj

c     ADM varibale for mod.f
      common/admmod/imid,jmin,jmax,kmin,kmax,xp1,xpWT,
```

```
     .                csmear,nksec,avgk,phiavgu(jt),phiavgtheta(jt),
     .                thetayz(jt,kt),Fyadm(jt,kt),Fzadm(jt,kt),
     .                testtheta,vv,vw,testr,testdiffrtheta,
     .                phiavgr(jt),diffphiavgrphiavgtheta(jt)

c    ALM parameters
      common/alm/tottime,radturnedtot,radturnedtotB(3),
     .           lapturnedtotB(3),radturnedB(3),lapturnedB(3),
     .           thetamesh,Theta2k(jt,jt),AngBlade,flagkplane

      integer, dimension(3) :: Bladek
      integer :: Bladep

c     Gaussian smearing related variables
      common/gauss/imiddiff,gaussianmag(it,jt),gaussianmagcheck(it,jt)

c     Epsi avgerage parameters
      common/epsiavgpar/epsiavgcounter,epsiavgsum,epsiavg


c- section 0 ------initialization ------------------------------------------
      entry modini

c Angular velocity of the rotation of the blades.
      omega = 1.003d0  !  [rad/s]

c     Gaussian smearing nodes
      imiddiff=50

c     Wind turbine location
c     find imid, the i location of the wind turbine
      do i=1,ni
         if (xp(i,1,1).le.0.and.xp(i+1,1,1).ge.0) ii=i
      end do
      imid=ii

      xpWT = 0.5*(xc(imid,1,1)+xc(imid-1,1,1))  ! streamwise position

c     Rotor dimensions
      jmin=2

c     Find jmax from the defined mesh
      hub=63.
```

```
      do j=1,nj
         if (yp(1,j,1).le.hub.and.yp(1,j+1,1).ge.hub) jj=j
      end do
      jmax=jj

      kmin=2
      kmid=nk/2
      kmax=nkm1


c     The number of k positions that are averaged over.
      avgk=kmax-kmin+1

c     Smearing function (convolution)
      csmear = 2.0

c     Write the center locations of the nodes of the
      open(unit=50,file='controlNode.inp',status='unknown')
      do j=jmin,jmax
        write(50,*)yp(1,j,1)
      end do
      close(50)

c     Number of nodes on the blad
      nBladeNodes=jmax-jmin+1

c     Number of sections that the force should be applied on
      nksec=nk-2

c     Approximation of pi Machin-like formulae
      pi=4.*(4.*atan(1./5.)-atan(1./239.))


      do j=jmin,jmax
        counterj=j-jmin+1
      do k=kmin,kmax
c       Tangential velocity,using polar coordinates conversion in y-z plane
c       Find Theta angle [rad]
        if (yp(imid,j,k).gt.0.0) then
          thetayz(j,k)=atan(zp(imid,j,k)/yp(imid,j,k))
        else if (yp(imid,j,k).lt.0.0.and.zp(imid,j,k).ge.0.0) then
          thetayz(j,k)=atan(zp(imid,j,k)/yp(imid,j,k))+pi
        else if (yp(imid,j,k).lt.0.0.and.zp(imid,j,k).lt.0.0) then
```

```
          thetayz(j,k)=atan(zp(imid,j,k)/yp(imid,j,k))-pi
        else if (yp(imid,j,k).eq.0.0.and.zp(imid,j,k).gt.0.0) then
          thetayz(j,k)=pi/2
        else if (yp(imid,j,k).eq.0.0.and.zp(imid,j,k).lt.0.0) then
          thetayz(j,k)=-pi/2
        else if (yp(imid,j,k).eq.0.0.and.zp(imid,j,k).eq.0.0) then
          thetayz(j,k)=0.
          write(6,*)'Theta angle in mod.f is not defined at (0,0)'
        end if
      end do
      end do

!-----------------------------------------------------------------------
! The aerodynamic characteristics of blade are read.
! rSpan: number of blade spanwise section including airfpoil profile
! cspan: local chord length [m]
! tSpan: local twist [degree]
!-----------------------------------------------------------------------
      open(1, file = 'aeroData.inp')
      do i = 1, nSpanLoc
         read(1,*) rSpan(i), cSpan(i), tSpan(i), temp
      end do
      close(1)

!-----------------------------------------------------------------------
! The blade control points are read. (Blade spanwise discretization)
!-----------------------------------------------------------------------
      open(2, file = 'controlNode.inp')
      do i = 1, nBladeNodes
         read(2,*) rNode(i)
      end do
      close(2)

!-----------------------------------------------------------------------
! The tabulated airfoil data, CL, CD and CM are read.
!-----------------------------------------------------------------------
        open(3, file = 'CLCDCMTable.inp' )
        read(3,*)
        read(3,*)
        read(3,*)
        read(3,*) NSPANB
        read(3,*) NMACHB
```

```
          write(6,*)'modini NMACHB',NMACHB
          flush(6)

          do k = 1, NMACHB
             read  (3,*)  AMACHTB (k)
          enddo

          read  (3,*)
          do j = 1, NSPANB
           read(3,*) RCB(j)
           read(3,*) NANGB(j)
           read(3,*)
           do i = 1, NANGB (j)
              read(3,*) AATB (j,i), (CLTB (j,k,i), CDTB (j,k,i),
     1                   CMTB (j,k,i), k=1, NMACHB)
         end do
           read(3,*)
         end do
           close(3)

           counter = maxval(NANGB)

c        Tracing the blades
c        Mesh turning
         thetamesh=2.0*pi/float(nksec)

c        Matrix mapping thetamesh to k plane
         do i=1,nksec

c         Radians turned smaller than the k plane used
          Theta2k(i,1)=i*thetamesh

c         k plane correspond to radians turned smaller than
          Theta2k(i,2)=i+1

         end do

c        Angle between the blades, here 3 blades
         AngBlade=2.0*pi/3.0

         return

c- section 1 ------properties-----------------------------------------------
```

```
c- section 2 ------convection-----------------------------------------------

c- section 3 ------ u-velocity-----------------------------------------------
      entry modu

c ---------------------------------------------------------------------------
c     ACTUATOR LINE MODEL
c ---------------------------------------------------------------------------

c     Blades turning
      tottime=0.0
      do locitstep=1,itstep
          tottime=tottime+dt(locitstep)
      end do


      radturnedtot=omega*tottime

c     Find radians each blade has rotated whitin the lap
      do k=1,3 ! 3 blades

        radturnedtotB(k)=radturnedtot+(k-1)*AngBlade
        lapturnedtotB(k)=radturnedtotB(k)/(2.0*pi)

c       Floor rounds to greatest integer less than, to obtain
c        the full laps by the blades
        lapturnedB(k)=FLOOR(real(lapturnedtotB(k)))
        radturnedB(k)=radturnedtotB(k)-(2.0*pi*lapturnedB(k))

      end do

c     Find the k plane location of the three blades
      do k=1,3  ! Each blade
        flagkplane=0
        i=1
        do while (i.le.nksec.and.flagkplane.eq.0)
          if (radturnedB(k).lt.Theta2k(i,1)) then
            Bladek(k)=Theta2k(i,2)
            flagkplane=1
          end if
          i=i+1
        end do
```

```
      end do

!-----------------------------------------------------------------------
! This part calculates the aerodynamic forces for the 3 blades
!-----------------------------------------------------------------------
      do k=1,3  ! 3 blades

      do j = 1, nBladeNodes

       counterj=jmin+j-1

c      Local axial velocity
       um = phi(imid,counterj,Bladek(k),u)

c      Local tangential velocity
       wm = -phi(imid,counterj,Bladek(k),v)
     .       *sin(thetayz(counterj,Bladek(k)))
     .      +phi(imid,counterj,Bladek(k),w)
     .       *cos(thetayz(counterj,Bladek(k)))

       rcp = rNode(j)

! The 'interp' subroutine calculate the local chord and twist
! by interpolation from the aerodynamic characteristic table.
       call interp(nSpanLoc, rcp, rSpan, cSpan, tSpan, chord, twist)

       AngleOfAttack=0.0

! The 'force' subroutine calculates the axial and tangential force at
! each spanwise section [N/m2].
       call force(um, wm, omega, nBlade, rho, rHub, rTip, soundSpeed,
     1 pitch, rcp, twist, chord, counter, NSPANM, NMACHM, NANGM,NSPANB,
     1 NMACHB, NANGB, RCB, AATB, AMACHTB, CLTB, CDTB, CMTB, axialForce,
     1 tangentForce,AngleOfAttack)

       afa(j,k)=axialForce          ! [N/m]
       tfa(j,k)=tangentForce        ! [N/m]

c  Have the forces in [N] by multipling by delta r of the mesh
       afa(j,k)=afa(j,k)
     .          *(yc(1,counterj,1)-yc(1,counterj-1,1))! [N]
       tfa(j,k)=tfa(j,k)
     .          *(yc(1,counterj,1)-yc(1,counterj-1,1))! [N]
```

```
      end do
      end do

c       Convert from tangential force to the v and w component.
      do j=jmin,jmax
      counterj=j-jmin+1
c      do k=kmin,kmax
      do k=1,3

c       v component
        Fyadm(j,k)=-tfa(counterj,k)*sin(thetayz(j,Bladek(k)))

c       w component
        Fzadm(j,k)=tfa(counterj,k)*cos(thetayz(j,Bladek(k)))

      end do
      end do

c     Check that the Gaussian has a total value of one
      epsiavgcounter=0.0
      epsiavgsum=0.0
      do Bladep=1,3
      do j=jmin,jmax
      counterj=j-jmin+1
      gaussianmag(Bladep,counterj)=0.0
      do k=kmin,kmax
      do i=imid-imiddiff,imid+imiddiff

        xp1          = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx           = xc(i,1,1)-xc(i-1,1,1)
        dr           = yc(imid,j,1)-yc(imid,j-1,1)
        drThetaloc    = (2*pi/nksec)*yp(imid,j,1)
        dRTheta     = (2*pi/nksec)*hub

c       3D distance to wind turbine blade
        dstream     = sqrt((xp1-xpWT)**2+
     .                     (yp(imid,j,Bladek(Bladep))-yp(imid,j,k))**2+
     .                     (zp(imid,j,Bladek(Bladep))-zp(imid,j,k))**2)

c       Epsilon
        epsi        = csmear*sqrt(dRTheta**2+dr**2+dx**2)
```

```
c       2D Gaussian function
        eta_eps    = 1./(epsi**2*pi)*exp(-(dstream/epsi)**2)

c       Original magnitude of the local sum gaussian function
        gaussianmag(Bladep,counterj)=gaussianmag(Bladep,counterj)+
     .                          eta_eps*dx*drThetaloc

c       Average epsi used to define epsilon of the simulation
        epsiavgcounter=epsiavgcounter+1.0
        epsiavgsum=epsiavgsum+epsi

      end do
      end do
      end do
      end do

c     Averagaged espi value in the simulation
      epsiavg=epsiavgsum/epsiavgcounter
      if (itstep.eq.1.and.iter.eq.1) then
        write(6,*)'Average epsilon value in the simulation is '
     .             ,epsiavg
      end if

c     Apply the force
      do Bladep=1,3
      fmax=0.
      do j=jmin,jmax
      counterj=j-jmin+1
      gaussianmagcheck(Bladep,counterj)=0.0
      do k=kmin,kmax
      fsum=0.0
      do i=imid-imiddiff,imid+imiddiff

        xp1         = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx          = xc(i,1,1)-xc(i-1,1,1)
        dr          = yc(imid,j,1)-yc(imid,j-1,1)
        drThetaloc    = (2*pi/nksec)*yp(imid,j,1)
        dRTheta    = (2*pi/nksec)*hub

c       3D distance to wind turbine blade
        dstream    = sqrt((xp1-xpWT)**2+
     .                    (yp(imid,j,Bladek(Bladep))-yp(imid,j,k))**2+
     .                    (zp(imid,j,Bladek(Bladep))-zp(imid,j,k))**2)
```

```
c       Epsilon
        epsi       = csmear*sqrt(dRTheta**2+dr**2+dx**2)

c       2D Gaussian function
        eta_eps    = 1./(epsi**2*pi)*exp(-(dstream/epsi)**2)

c       Axial force
        fp         = afa(counterj,Bladep)

c       Apply the gaussian and normalize to have a sum of one
        fdist      = fp*eta_eps*dx*drThetaloc
     .                  /gaussianmag(Bladep,counterj)
        su(i,j,Bladek(Bladep)) = su(i,j,Bladek(Bladep)) - fdist

c       Sum gaussian function to ensure that the sum is one
        gaussianmagcheck(Bladep,counterj)=
     .                  gaussianmagcheck(Bladep,counterj)+
     .                  eta_eps*dx*drThetaloc
     .                  /gaussianmag(Bladep,counterj)

      end do
      end do
      end do
      end do


      return

c- section 4 ------v-velocity-----------------------------------------------
      entry modv

c ---------------------------------------------------------------------------
c     ACTUATOR LINE MODEL
c ---------------------------------------------------------------------------

      do Bladep=1,3
      fmax=0.
      do j=jmin,jmax
      counterj=j-jmin+1
      gaussianmagcheck(Bladep,counterj)=0.0
      do k=kmin,kmax
      fsum=0.0
```

```
      do i=imid-imiddiff,imid+imiddiff

        xp1          = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx           = xc(i,1,1)-xc(i-1,1,1)
        dr           = yc(imid,j,1)-yc(imid,j-1,1)
        drThetaloc   = (2*pi/nksec)*yp(imid,j,1)
        dRTheta      = (2*pi/nksec)*hub

c       3D distance to wind turbine blade
        dstream      = sqrt((xp1-xpWT)**2+
     .                      (yp(imid,j,Bladek(Bladep))-yp(imid,j,k))**2+
     .                      (zp(imid,j,Bladek(Bladep))-zp(imid,j,k))**2)

c       Epsilon
        epsi         = csmear*sqrt(dRTheta**2+dr**2+dx**2)

c       2D Gaussian function
        eta_eps      = 1./(epsi**2*pi)*exp(-(dstream/epsi)**2)

c       v component of the tangential force
        fp           = Fyadm(j,Bladep)

c       Apply the gaussian and normalize to have a sum of one
        fdist        = fp*eta_eps*dx*drThetaloc
     .                    /gaussianmag(Bladep,counterj)
        su(i,j,Bladek(Bladep)) = su(i,j,Bladek(Bladep)) - fdist

        end do
        end do
        end do
        end do


      return

c- section 5 -------w-velocity-------------------------------------------
      entry modw

c ------------------------------------------------------------------------
c     ACTUATOR LINE MODEL
c ------------------------------------------------------------------------

      do Bladep=1,3
```

```
      fmax=0.
      do j=jmin,jmax
      counterj=j-jmin+1
      gaussianmagcheck(Bladep,counterj)=0.0
      do k=kmin,kmax
      fsum=0.0
      do i=imid-imiddiff,imid+imiddiff

        xp1          = 0.5*(xc(i,1,1)+xc(i-1,1,1))
        dx           = xc(i,1,1)-xc(i-1,1,1)
        dr           = yc(imid,j,1)-yc(imid,j-1,1)
        drThetaloc   = (2*pi/nksec)*yp(imid,j,1)
        dRTheta      = (2*pi/nksec)*hub

c       3D distance to wind turbine blade
        dstream      = sqrt((xp1-xpWT)**2+
     .                      (yp(imid,j,Bladek(Bladep))-yp(imid,j,k))**2+
     .                      (zp(imid,j,Bladek(Bladep))-zp(imid,j,k))**2)

c       Epsilon
        epsi         = csmear*sqrt(dRTheta**2+dr**2+dx**2)

c       2D Gaussian function
        eta_eps      = 1./(epsi**2*pi)*exp(-(dstream/epsi)**2)

c       w component of the tangential force
        fp           = Fzadm(j,Bladep)

c       Apply the gaussian and normalize to have a sum of one
        fdist        = fp*eta_eps*dx*drThetaloc
     .                    /gaussianmag(Bladep,counterj)
        su(i,j,Bladek(Bladep)) = su(i,j,Bladek(Bladep)) - fdist

        end do
        end do
        end do
        end do


      return


c ...
```

M

CALC ALM force.f file

```
      subroutine force(u, w, rotVel, nb, airDen, hubRad, tipRad, snVel,
     1 bPtc, rLoc, locTwist, locChord, maxCounter, nSpanG, nMachG,
     1 nAngleG, nSpanTable, nMachTable, nAngleTable, rLocTable,
     1 aoaTable, machTable, clTable, cdTable, cmTable, Fn, Ft, aoa)
!-----------------------------------------------------------------------
      implicit none
!-----------------------------------------------------------------------
      double precision, parameter :: nacelleDragCoeff = 0.50d0
      integer :: nb, nSpanG, nMachG, nAngleG, nSpanTable, nMachTable,
     1 maxCounter
      integer, dimension(nSpanG) :: nAngleTable
      double precision :: rotVel, airDen, hubRad, tipRad, snVel, bPtc,
     1 rLoc, locTwist, locChord, Fn, Ft, u, w, CL, CD, CM, pi, criteria,
     1 flowAngle, aoa, liftCoeff, dragCoeff, normalCoeff, tangentCoeff,
     1 sigma, fTipCorr, fHubCorr, Ftip, Fhub, Ftot, aInd, apInd, Vx, Vt,
     1 VxUpdate, VtUpdate, deltaVx, deltaVt, vTotal2, machNumber,
     1 aIndUpdate, apIndUpdate, deltaaInd, deltaapInd,
     1 acInd, KaInd
      double precision, dimension(nMachG) :: machTable
      double precision, dimension(nSpanG) :: rLocTable
      double precision, dimension(nAngleG) :: aoaInterp, clInterp,
     1 cdInterp, cmInterp
      double precision, dimension(nSpanG, nAngleG) :: aoaTable
      double precision, dimension(nSpanG,nMachG,nAngleG) ::
     1 clTable, cdTable, cmTable
!-----------------------------------------------------------------------
      Vx = u
```

```
! [m/s], from Navier-Stokes solution, axial velocity perpendicular to
! the rotor plane
        Vt = w
! [m/s], from Navier-Stokes solution, tangential velocity perpendicular
! to the rotor plane

        pi = 4.0d0 * atan(1.0d0)

          if (rLoc .le. hubRad) then
             Ft = 0.0d0
             Fn = 0.50d0 * airDen * (u ** 2) * nacelleDragCoeff
             goto 20
          endif
! For the radii smaller than hub radius, the constant drag coefficient
! for the nacelle (Cd_nacelle=0.5) is considered. So, in that case,
! only the axial force acts on the actuator disk.
! The dimension of the Ft and Fn are [N/m2].
!-----------------------------------------------------------------------
         flowAngle = atand(Vx / abs(-rotVel * rLoc - Vt))
! [degree], flow angle calculation based on the axial and tangential
! velocities.
        aoa = flowAngle - (locTwist + bPtc)
! aoa [degree], angle of attack calculation by means of flow angle,
! local twist and blade pitch angle.
!-----------------------------------------------------------------------
! Lift and drag coefficients based on the linear airfoil theory.
!-----------------------------------------------------------------------
!        liftCoeff = 2.0d0 * pi * (aoa * pi / 180.0d0)
!        dragCoeff = 0.0d0
!-----------------------------------------------------------------------
        machNumber = sqrt((Vx ** 2.0d0) + ((-rotVel * rLoc - Vt)
     1             ** 2.0d0)) / snVel
! machNumber is the Mach number which is used when reading and
!interpolating airfoil data.
!-----------------------------------------------------------------------
          call aeroTable(rLoc, machNumber, rLocTable, machTable, aoaTable,
     1       clTable, cdTable, cmTable, nSpanG, nMachG, nAngleG,
     1       nSpanTable, nMachTable, nAngleTable, clInterp, cdInterp,
     1       cmInterp, aoaInterp)

          call foil_clcdcm(aoa, maxCounter, nAngleG, aoaInterp,
     1       clInterp, cdInterp, cmInterp, CL, CD, CM)
```

```
! These two subroutines calculate lift (CL) and drag (CD) coefficients
! for each spanwise section.
!-----------------------------------------------------------------------
          liftCoeff = CL ! Lift coefficient for each spanwise section.
          dragCoeff = CD ! Drag coefficient for each spanwise section.

        normalCoeff = (liftCoeff * cosd(flowAngle))
     1              + (dragCoeff * sind(flowAngle))
! normalCoeff is the projection of lift and drag coefficients into the
! normal direction (perpendicular to the rotor plane, Axial direction)

        tangentCoeff = (liftCoeff * sind(flowAngle))
     1               - (dragCoeff * cosd(flowAngle))
! tangentCoeff is the projection of lift and drag coefficients into the
! tangential direction (tangential to the rotor plane)
!-----------------------------------------------------------------------
        vTotal2 = Vx ** 2.0d0 + (abs(-rotVel * rLoc - Vt)) ** 2.0d0
! The magnitude of the total velocity is computed.
!-----------------------------------------------------------------------
        Fn = 0.50d0 * airDen * vTotal2 * locChord * normalCoeff
! [N/m], The axial force (perpendicular to the rotor plane).
        Ft = 0.50d0 * airDen * vTotal2 * locChord * tangentCoeff
! [N/m], The tangential force (tangential to the rotor plane).

20     continue

          return
          end subroutine
```