Wing-, cube- and aeroelastic simulations in Unicorn

Kenny Hedlund
Master Thesis
Royal Institute of Technology, 100 44, Stockholm, Sweden

December 20, 2010
Abstract

The new FEM computational fluid dynamics software, Unicorn, has been used for aerodynamic force estimation on a cube and a NACA0012 wing spanning a wind tunnel. It has also been used for an elastic plate, showing that aeroelastic problems may be solved in a near future. This has been done, modeling the boundary layer with slip condition. For the wing, the fluid has also been modelled as completely inviscid.

The cube drag computation has indicated convergence towards the experimental drag coefficient of 1.05 within a few percent. The wing simulations on the other hand have proven to give rise to more problems. Here, a set of different boundary conditions, meshes and normal computation methods have been evaluated. Most of the results have shown fairly good lift computations (within a few percent of the experimental values), but all has shown too large drag estimations. They do however tend to decrease when the mesh is refined, with the adaptive mesh refinement functionality. The results have also proven to replicate experiments well in a qualitative manner, like oil flow visualisation and circulation patterns.

I has been concluded that edge conditions should be avoided, in order to get pure slip condition everywhere and that superparametric elements should be used in order to avoid artificial friction from velocity projection at the nodes. Further conclusions are that it is crucial to move added nodes, from adaptive mesh refinement, to the exact geometry and that computations should be made for large aircraft or wings with wing tips, since the too large drag might be vanishingly small in comparison to larger scale drags like wing tip vortices.
Nomenclature

\( b \) = Wing span
\( C \) = Constant
\( c \) = Wing chord
\( C_D \) = Drag coefficient
\( C_L \) = Lift coefficient
\( D(\cdot) \) = Derivation operator w.r.t \( \cdot \)
\( F \) = Displacement rate vector [m/s]
\( f \) = Volume force vector [N]
\( G \) = Structural shear modulus [Pa]
\( h \) = Mesh size [m]
\( I \) = Time domain \([0,T]\)
\( I \) = The identity matrix
\( k \) = Time step [s]
\( K \) = Finite element in mesh
\( M \) = Magnitude of interest
\( n \) = Normal vector
\( p, \dot{p}, P, q \) = Pressures [Pa]
\( \mathbf{p}_p \) = Particle position
\( q_F, v_F \) = Finite element functions
\( Q \) = Space-time domain
\( R \) = Residual vector
\( t \) = Tangential vector
\( t \) = Time [s]
\( T \) = Final time
\( \mathbf{u}, \mathbf{U}, \mathbf{w}, \mathbf{v} \) = Velocity vectors [m/s]
\( \mathbf{X}, \mathbf{x} \) = Particle original position
\( \alpha \) = Angle of attack (a.o.a) [deg]
\( \mu, E \) = Isotropic Young’s modulus [Pa]
\( \nu \) = Kinematic viscosity \([m^2/s]\)
\( \partial \Omega \) = Space boundary
\( \Omega \) = Space domain
\( \varphi \) = Dual velocity
\( \psi \) = Dual magnitude of interest
\( \rho \) = Density [kg/m\(^3\)]
\( \rho_s \) = Structure density [kg/m\(^3\)]
\( \tau_{\text{in}} \) = Finite element mesh
\( \theta \) = Dual pressure
1 Introduction

It is not hard to realise that it is necessary to ensure that the wings should not be subject to structural failure, and that this requires some means of estimating the structural displacement resulting from air loads. One way of doing this is by estimating the aerodynamics with unsteady potential flow theory and linearizing the equations of motion around some given condition. This enables a stability analysis of both the dynamic and static behaviour of the wing and also gives the possibility to estimate static deformations.

The static stability analysis is typically used to predict divergence and reversal. Divergence happens when the aerodynamic forces are reducing the effective stiffness of the wings to zero or further in some direction, and thus, potentially leading to very large deformations. This may of course have catastrophic consequences. Reversal is when the effectiveness of a control surface is reduced because of the moment that twists the lifting surface and thereby changes the angle of attack in such a way that it counteracts the lift produced by the control surface itself.

The dynamic stability is governing what is known as flutter. This is when a perturbation of the deformation results in an oscillating motion which is undamped, or negatively damped. Even though this may be described by a linear system of equations, the stability analysis is not trivial to resolve. The reason for that, is that it results in a nonlinear eigenvalue problem, and therefore requires an iterative solution process. Nevertheless, this method is relatively fast, and therefore a very powerful tool. However, this speed does not come completely without flaws; since it is based on potential flow theory, the accuracy may be quite low, and completely unable to predict the nonlinear regime of the flow. Thus it may be of interest to investigate linear instabilities further, using more accurate CFD methods, so that more accurate results may be obtained. Therefore this report will investigate the possibility to solve the aerodynamics and the structural mechanics simultaneously for an aeroelastic problem.

The software that will be used in this study is capable of solving fluid-structure interaction as one problem, which makes it ideal for space-time simulations of aeroelastic problems. It is called Unicorn, and forms a part of the FEniCS project [1]. This is however based on a rather unconventional approach to obtaining accurate CFD solutions, explained in section 2.1, which is why it will need to be tried for some basic purely aerodynamic problems for justification of the use of this software.
2 Theory

2.1 Aerodynamic Modeling

In today’s fluid dynamics, a no slip BC (Boundary Condition), along with the incompressible Navier-Stokes (ICNS) system of equations, (1), is needed for accurate estimations of subsonic fluid flows. (No slip means that the air has zero relative velocity compared to the boundary on the boundary, while slip allows tangential velocities.)

\[
\dot{u} + (u \cdot \nabla)u - \nu \Delta u + \frac{\nabla p}{\rho} = f, \quad \text{in } Q
\]
\[
\nabla \cdot u = 0, \quad \text{in } Q
\]
\[
u \cdot u = 0, \quad \text{on } \partial \Omega \times I
\]

(1)

\(Q\) is the considered domain for the entire considered time and \(I\) is the time domain \([0,T]\).

Therefore, viscous effects have a large impact on the flow close to the surface, even in almost inviscid fluids, forming a viscous boundary layer (BL). This is of major importance since this provides an explanation to lift and drag. However, a new explanation has been presented by Claes Johnson and Johan Hoffman in [6], where they claim that this may be obtained accurately without boundary layers.

This might be a major break through in fluid dynamics, since it would allow for the flow to be computed without resolving the small boundary layers, present in high Reynolds number problems for large structures, potentially allowing for a great reduction in mesh points.

2.1.1 Impact of Boundary Layers

If the viscosity is zero and a slip BC is applied, equation 1 turns into the incompressible Euler equations, (2).

\[
\dot{u} + (u \cdot \nabla)u + \frac{\nabla p}{\rho} = f, \quad \text{in } Q
\]
\[
\nabla \cdot u = 0, \quad \text{in } Q
\]
\[
u \cdot u = 0, \quad \text{on } \partial \Omega \times I
\]

(2)
These equations can be solved analytically for e.g. the flow around a wing using potential theory. When this is done, one obtains a clearly unphysical result, namely that the forces from the flow on the wing is zero. BL theory explains this by stating that the boundary layer generates circulation, causing the flow to be faster on the upper side than the lower side. This also creates an up draught upstream of the wing and a down draught downstream. These phenomena result in forces on the body, which should be expected. BL theory may also be used to explain separation and the difference between turbulent and laminar separation.

Johnson and Hoffman, however, claim that the potential solution is unstable to perturbations and that it is therefore it cannot be observed in reality. This may be shown by equations (2)-(5). Consider two solutions to equation (2), \((w, \hat{p})\) and \((u, p)\).

\[
\dot{w} + (w \cdot \nabla)w + \frac{\nabla \hat{p}}{\rho} = f, \quad \text{in } Q
\]
\[
\nabla \cdot w = 0, \quad \text{in } Q
\]
\[
w \cdot n = 0, \quad \text{on } \partial \Omega \times I
\]

By subtracting (2)-(3), one obtains a linearized Euler equation (4).

\[
\dot{v} + (u \cdot \nabla)v + (v \cdot \nabla)u + \frac{\nabla q}{\rho} = 0, \quad \text{in } Q
\]
\[
\nabla \cdot v = 0, \quad \text{in } Q
\]
\[
v \cdot n = 0, \quad \text{on } \partial \Omega \times I
\]

where the quadratic perturbation terms have been omitted, with

\[
v = u - w
\]
\[
q = p - \hat{p}
\]

and since

\[
\Sigma(\text{diag}(A)) = \Sigma(\text{eig}(A)), \quad \text{for any matrix } A
\]
\[
\Rightarrow \Sigma(\text{eig}(\nabla u)) = 0
\]

This means that \(\nabla u\) has positive eigenvalues, unless the real parts of the diagonal are all zero. Therefore, exponential growth of perturbations is expected.

### 2.1.2 FEM

Unicorn uses a Finite Element Method named the G2 (Galerkin 2) - method, by Johnson and Hoffman [6]. This solves (1), but with slip BC instead of no slip.

The time stepping, \(\dot{U}\), can be viewed as Crank-Nicholson for solutions that are piecewise linear in space and time. This is done by iterating (for \(\nu = 0\),

\[
U_{n+1} = U_n + k(0.5f(U_{n+1}^t) + 0.5f(U_n^t)),
\]
\[
f(U_{n+1}) = -U_{n+1} \cdot \nabla U_{n+1} - \nabla p_{n+1}^t,
\]
where $i$, indicates the iteration number and $U_n = U(t_n)$.

The choice of $k$ is made so that it satisfies

$$k \leq \frac{h_{\text{min}}}{|\bar{U}|}$$

(6)

### 2.1.3 Adaptivity

When solving the Euler equations with FEM, several factors provide perturbations from an exact solution, such as the interpolations over the cells and the residuals. On one hand this is good, since it provides the possibility to evaluate reality, which is subject to perturbations, but on the other hand one must control these perturbations, so that they are small, at least in an integral sense. The way this is done with Unicorn is explained thoroughly in [6], but may be explained briefly as follows.

Consider two numerical solutions, $(U, P)$ and $(u, p)$, to equation (2), with residuals, $R_U$ and $R_u$ along with corresponding dual problem (7).

$$-\dot{\varphi} - u \cdot \nabla \varphi + \nabla u^T \varphi + \nabla \theta = \psi$$

(7)

It is then possible to show that

$$M(u, p) - M(U, P) = \int_Q (R_u - R_U) \cdot \varphi \, dx dt,$$

which means that

$$| M(u, p) - M(U, P) | \leq C \parallel \dot{\varphi} \parallel (\parallel hR_u \parallel + \parallel hR_U \parallel),$$

(8)

where $\parallel \cdot \parallel$ indicates an integral of the norm of $\cdot$ over $Q$.

Since the residuals typically grow as $1/\sqrt{h}$, equation (8) typically decrease as $h$ is refined with $\sqrt{h}$. This also allows for $h$ to be decreased only where it affects the magnitude of interest the most, and thus for the mesh to be adaptively refined until some convergence criteria is reached.

To conclude, this means that the method is suitable if integral values are of interest, but not if pointwise solutions are required.

In most cases, the geometry that defines the boundary includes bent surfaces. This might introduce problems if one wants to take full advantage of the adaptive functionality. One is that, if new nodes are to be added to the boundary, they should probably not be added between to other nodes, but rather on the geometry, between the nodes. If this can be done, another problem also becomes apparent. This is the fact that the solver actually computes a solution for the poorly resolved geometry, defined by the mesh, which might have been a good solution for such a geometry, but not for the real one. Therefore, an error estimation model is needed for how the difference between the mesh geometry and the wanted one affects the solution. In this case, since these are still open issues for Unicorn, the way around this problem has been to manually resolve parts of the geometry with high curvature. This is done by choosing an initial mesh which accurately describes the geometry.

### 2.1.4 Slip BC

One approach to explain the origin of lift and drag is that the viscous boundary layer creates vorticity, friction and circulation, that in turn creates lift and drag. The way that circulation creates lift is explained by Kutta-Joukowski. Johnson and Hoffman however claim that this does not describe physics; that there is no large scale circulation around airfoils and no starting vortex.
Instead, they have presented a new explanation that supposedly explains both lift and drag, see figure 2.

![Figure 2: Johnson/Hoffman’s alternative to Kutta-Joukowski.](image)

When two flow streams are converging on the downstream side of 3D objects, a transverse flow is induced, that in turn induces streamwise vorticity. This allows for low pressure vortices to pull down the flow towards the surface and thus creating lift and drag on airfoils.

What might introduce some problems here however, is the fact that the slip boundary condition requires the surface normals. These may be defined as giving the node normals the mean of the surrounding facets’ normals, weighted by their areas. This might, however introduce errors in the normals in a way that they do not exactly correspond to the normals of the actual geometry in the corresponding points. This might be solved by extracting the true normal distribution from the geometry, and importing it in the solver, in order to get the true values in the nodes. That functionality is under development for Unicorn, but not yet incorporated. Therefore also this problem requires curved surfaces to be accurately defined already in the initial mesh.

Furthermore, when a geometry has edges or corners, the normals are not defined exactly on the edge. This might be solved by using the mean value of the surrounding facets as described above. That might however induce a flow through the facets. To solve this, Unicorn has a maximum angle between the face normals, above which the node is defined as an edge node or corner node, depending on the number of directions of nearby facet inclinations. This is done by enforcing slip condition for both surrounding normals (edge) or no slip (corner). The default maximum facet inclination angle in Unicorn is set to $\pi/6$. However, since having an edge at the trailing edge will enforce a stagnation point at the trailing edge and a “bubble of slow air” of the size of the trailing edge cells, investigation is required as to whether the edge condition should be present there. It is also to be investigated whether or not the edge criterium is needed in the boundary between e.g. wings and wind tunnel walls.

### 2.1.5 Turbulence

As stated earlier, argues that the potential solution is unstable to perturbations. This is shown in the analysis of the linearized Euler equations, but may also be shown by the vorticity equation, (9), explaining the large amount of vorticity, present in turbulent flows.

$$\dot{\omega} + (u \cdot \nabla)\omega - (\omega \cdot \nabla) u = 0$$

If one then focuses on the potential solution in the circular cylinder case, $\phi(x_1, x_2) = (r + \frac{1}{r})\cos(\beta)$, being the real part of the analytic function $w = z + \frac{1}{z}$ with $z = x_1 + ix_2$, one gets
Thus, (9) takes the form

\begin{align}
\dot{\omega}_1 + (\mathbf{u} \cdot \nabla)\omega_1 &= 2\omega_1 \\
\dot{\omega}_2 + (\mathbf{u} \cdot \nabla)\omega_2 &= -2\omega_2 \\
\dot{\omega}_3 + (\mathbf{u} \cdot \nabla)\omega_3 &= 0,
\end{align}

close to the point \( z = 1 \) (the downstream stagnation point), which means that \( \omega_1 \) grows exponentially and thus that strong vorticity will be generated close to the rear stagnation point.

Equation (8) shows that the residuals may be pointwise large for small \( h \), without having large impact on the mean value output of the magnitude of interest. This would be critical in case you get blow up when solving a problem, because even if the residual grows as the spatial step size is decreased, the reduction of \( h \) might counteract that growth. This is exactly what [6] claim happens when the NS are solved with the G2 method, see equations (12).

\begin{equation}
\begin{split}
R(U) \sim h^{-1/2} \\
| M(u, p) - M(U, P) | &\leq C||\hat{\varphi}||(||hR_u|| + ||hR_p||) \sim C||\hat{\varphi}h^{1/2},
\end{split}
\end{equation}

This means that you would expect a turbulent solution to blow up and be wrong everywhere, but show a mean value convergence as the mesh is refined.
An example of this is shown in the following scalar linear constant coefficient stationary model problem.

\[ u,1 + u - \nu \delta u = f, \text{ in } \Omega, \ u = 0, \text{ on } \Gamma, \]  

(13)

where \( \nu \) is small and \( \Omega = (0,1)^2 \) with boundary \( \Gamma \), \( u,1 = \frac{\partial u}{\partial x_1} \) and \( f \) is a given function. Then let \( U \) be a continuous piecewise linear G2 solution defined by

\[ (U,1 + U, v + hv_1 = (f, v + hv, 1)) \]  

(14)

where \( (\cdot, \cdot) \) is the \( L_2(\Omega) \)-norm, since \( \nu \ll h \) and the stabilizing term, \( h(v,1 + v) \), has been simplified to \( hv_1 \). With the \( L_2(\Omega) \)-norm \( \| \cdot \| \), since \( \| f \| = 1 \) the following energy estimate,

\[ \| U \|^2 + \| \sqrt{h}U_1 \|^2 \leq (1 + h)\| f \|^2 \approx 1 \]

(15)

shows that the stabilizing term \( \| \sqrt{h}U_1 \|^2 \) will not be small in case the exact solution has layers, because \( U,1 \sim h^{-1} \) in an outflow layer of width \( \sim h \), and \( U,1 \sim h^{-1/2} \) in characteristic layers of width \( h^{1/2} \). Thus

\[ R(U) \approx U,1 + U - f \]

(16)

which shows that \( R(U) \) cannot be small everywhere, but with \( R(U) \sim h^{-1} \) in the outflow layers and \( R(U) \sim 1 \) in the characteristic layers (parallel to boundary), then

\[ | M(u) - M(U) | \sim h^{3/2}, \]  

(17)

if \( M(u) = (u, \psi) \) and \( \psi \) vanishes in the layers. Then again, the error in output of interest may be small even though the residuals are pointwise large ("output of interest" is a quantity of which one may be interested, eg. a space interal of the pressure on one of the boundaries). [6]

2.2 Structural Modeling

The constitutive modeling is derived by first expressing the displacement rate as function of original particle position, time and velocity. This is described in equations (18). The displacement rate in one element contributes its velocity along with the neighbours’ displacement rates, that translates and rotates the element like a rigid body.

\[ D_t F(t, P_p(t, X)) = D_t D_X P_p(t, X) \]

\[ = D_X u(t, P_p(t, X)) \]

\[ = D_p u(t, P_p(t, X)) D_X P_p(t, X) \]

\[ = \nabla u(t, P_p(t, X)) F(t, P_p(t, X)) \]

\[ \Leftrightarrow D_t F = \nabla u F \]

\[ \Rightarrow D_t F^{-1} = -F^{-1} \nabla u \]  

(18)

Using the relation
\[ \mathbf{B} = \mathbf{F} \mathbf{F}^T, \]

equation (18) may be modified to exclude the displacement rate. This may be achieved by using the Neo-Hookean constitutional equation (19).

\[ \sigma = \mu (\mathbf{B} - \mathbf{I}) - p \mathbf{I} \quad (19) \]

If we differentiate the component

\[ \sigma_D = \mu (\mathbf{B} - \mathbf{I}), \quad (20) \]

with respect to time, using the relation

\[ D_t \mathbf{B} = \nabla \mathbf{u} \mathbf{B} + \mathbf{B} \nabla \mathbf{u}^T, \quad (21) \]

and equation (20), we obtain equation (22).

\[ D_t \sigma_D = \mu 2 \varepsilon(\mathbf{u}) + \nabla \mathbf{u} \sigma_D + \sigma_D \nabla \mathbf{u}^T \quad (22) \]

Thus, we may solve a rate problem, which is equivalent to the non rate model (20). Since the aerodynamic problem uses a rate model, this allows for the whole domain to be solved simultaneously given only the model parameters and an indicator that defines which cells are structure and which ones are fluid.

This, does however require some approximations:

- Only choosing the \( \sigma_D \) part of the problem approximates the solid as incompressible.
- Using a scalar \( \mu \) instead of a stiffness matrix, \( \mathbf{E} \), only allows for isotropic material models (which is the case for the code at the moment).

### 2.3 ALE

In aerodynamic problems, an Eulerian representation is most convenient, while in structural problems, a Lagrangian representation is easier. Therefore, an ideal way of solving a fluid-structure interaction problem would be to use Eulerian coordinates for the fluid and Lagrangian coordinates for the structure. This may be done by moving the structural part of the mesh inside the fluid part and adding the mesh velocity, \( \mathbf{w} \) to the fluid equations of motion, see equation (23).

\[ (\dot{\mathbf{u}}) + ((\mathbf{u} - \mathbf{w}) \cdot \nabla)(\mathbf{u}) + \frac{\nabla p}{\rho} = 0 \quad (23) \]

This however, requires that the mesh must be smoothed in some way, in order to avoid bad or inverted elements. One way of doing this is by elastic mesh smoothing, which is briefly described by equations (24).

\[ \sigma = E (\mathbf{I} - (\mathbf{F} \mathbf{F}^T)^{-1}) \]

\[ D_t \mathbf{F}^{-1} = -\mathbf{F}^{-1} \nabla \mathbf{w}, \quad (24) \]

with the initial condition on \( \mathbf{F} \) that it corresponds to the deformation gradient with regard to a scaled equilateral reference cell.
3 Method

3.1 Mesh generation

The mesh generator chosen was Salome [2], because it is open source and because it is possible to load coordinate files for creating geometries. Another advantage with Salome is the script functionality, allowing the user to change parameters without having to redraw/remesh manually. It also allows exchanging scripts of a few kB instead of meshes tens or hundreds of MB, when more than one person is involved in the process. An example of a script for creating a uniform tetrahedral mesh of a cube in a tunnel is presented in figure 4.

```python
import smesh
import geompy
import salome
import StdMeshers
import NETGENPlugin
import math
gg = salome.ImportComponentGUI("GEOM")

box1 = geompy.MakeBox(-0.1, -0.1, -0.1, 0.1, 0.1, 0.1)
box2 = geompy.MakeBox(-2.0, -0.5, -0.5, 5.0, 0.5, 0.5)
cut = geompy.MakeCut(box2, box1)
geompy.addToStudy( cut, "cut" )

print 'Geometry computed'
Mesh_1 = smesh.Mesh(cut)
Regular_1D = Mesh_1.Segment()
Max_Size_1 = Regular_1D.MaxSize(0.2)
MEFISTO_2D = Mesh_1.Triangle(algo=smesh.NETGEN_2D)
Tetrahedron_Netgen = Mesh_1.Tetrahedron(algo=smesh.NETGEN)

if not isDone:
    print 'Mesh computation failed'

print "Done"
```

Figure 4: Simple cube mesh script for Salome.

However, since the dolfinXML mesh format is not available in Salome, a file format conversion program has been created. This simply reads the relevant data in a unv-file (available in Salome) to save in dolfinXML, see figure 5.

Corresponding dolfinXML file would then be the one described by figure 6.

The cube case was solved using a uniform tetrahedral mesh with 3750 nodes as initial mesh. This mesh was provided by Murtazo Nazarov.

The wing section was generated by interpolating the coordinates shown in table 11 in the appendix using bezier interpolation, after scaling, rotating and translating the coordinates around the geometric center of the wing section to fit the corresponding test case, see equation (25).
### Node number

<table>
<thead>
<tr>
<th>Node</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9699</td>
<td>0.8199</td>
<td>2.5000</td>
</tr>
<tr>
<td>2</td>
<td>0.5590</td>
<td>0.7347</td>
<td>2.4954</td>
</tr>
</tbody>
</table>

### Node coordinates

<table>
<thead>
<tr>
<th>Node</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Element number

<table>
<thead>
<tr>
<th>Element</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>41</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Nodes in element

<table>
<thead>
<tr>
<th>Node</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>2411</td>
<td>9.9699</td>
<td>-7.8199</td>
<td>2.5000</td>
</tr>
<tr>
<td>23004</td>
<td>9.5590</td>
<td>-6.7347</td>
<td>-2.4954</td>
</tr>
</tbody>
</table>

---

Figure 5: UNV-mesh example from Salome. Necessary mesh information for dolfinXML files encircled (only 3D elements are needed).

\[
\begin{align*}
  x_{\text{transformed}} &= c \left( (x_{\text{table}} - \frac{1}{2}) \cos(\alpha) + y_{\text{table}} \sin(\alpha) + \frac{1}{2} \right) \\
  y_{\text{transformed}} &= c \left( -(x_{\text{table}} - \frac{1}{2}) \sin(\alpha) + y_{\text{table}} \cos(\alpha) \right) \\
  z_{\text{transformed}} &= z
\end{align*}
\]  

For the wing meshes, the attempts to use the NETGEN 1D-2D-3D generator with mesh optimization were unsuccessful, which is why manual choices of mesh sizes for different domains were needed. The two different 1-D mesh size configurations that have been used in the project are presented in table 1.

Four subdomains were used in order to enable different cell sizes at different locations in the mesh. These are presented in figure 7.

To create the leading edge subdomains, the same procedure was performed for the same wing, scaled and translated to form a surrounding layer for an amount of points corresponding to the length of the domain. Then all faces were extruded with the span of the wing and put inside a tunnel with some appropriate cut/merge operations.

The flutter plate mesh was created using the NETGEN 1D-2D-3D algorithm. In order to avoid having to have no slip boundary conditions at the junction between the flutter plate and
Figure 6: DolfinXML file corresponding to the UNV file in figure 5.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>20k node mesh value</th>
<th>100k node mesh value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain 1 maximum 1D length</td>
<td>0.4 [m]</td>
<td>0.2 [m]</td>
</tr>
<tr>
<td>Domain 2 maximum 1D length</td>
<td>0.1 [m]</td>
<td>0.05 [m]</td>
</tr>
<tr>
<td>Domain 3 maximum 1D length</td>
<td>0.02 [m]</td>
<td>0.01 [m]</td>
</tr>
<tr>
<td>Domain 4 maximum 1D length</td>
<td>0.01 [m]</td>
<td>0.005 [m]</td>
</tr>
<tr>
<td>1D mesh generation algorithm</td>
<td>Regular 1D</td>
<td>Regular 1D</td>
</tr>
<tr>
<td>2D mesh generation algorithm</td>
<td>Mephisto 2D triangle</td>
<td>Mephisto 2D triangle</td>
</tr>
<tr>
<td>3D mesh generation algorithm</td>
<td>Netgen 3D tetrahedron</td>
<td>Netgen 3D tetrahedron</td>
</tr>
</tbody>
</table>

Table 1: Mesh specifications for wing case sub domains. Geometry and mesh generated with Salome [2].

the wind tunnel wall, the wind tunnel was divided into two sub domains, where the structural one formed the plate and the wall, see figure 8 and table 2.

3.2 Manual wing surface normal computation

In order to manually compute the surface normals for the NACA0012 profile, one needs the wing section equation (26) [5].

\[
\frac{y}{c} = \pm 0.6[0.2969 \sqrt{x/c} - 0.126x/c - 0.3516(x/c)^2 + 0.2843(x/c)^3 - 0.1015(x/c)^4]
\] (26)

However, in order for equation (26) to be valid for a given node, that node’s coordinates have to be rotated back from the present angle of attack to 0 degrees (see section sec:mesh). Then, in order to get the node normal in the rotated coordinate system, one may simply derivate the wing section equation and choose a vector, perpendicular to the derivative vector, see equations (27) and (28).
Figure 7: Mesh sub domains.

\[
D_{(x/c)} \left[ \frac{y}{c} \right] = \pm 0.6 \left[ \frac{0.2969^{-0.5x/c}}{2} - 0.126 - 2 \times 0.3516(x/c) + 3 \times 0.2843(x/c)^2 - 4 \times 0.1015(x/c)^3 \right]
\]

\[\Rightarrow \mathbf{t}^* = \begin{bmatrix} 1, \pm 0.6[0.14845^{-0.5x/c} - 0.126 - 0.7032(x/c) + 0.8529(x/c)^2 - 0.406(x/c)^3], 0 \end{bmatrix}^T \]

\[\mathbf{t} = \mathbf{t}^*/\|\mathbf{t}^*\|\]  

(27)
Table 2: Flutter plate mesh specifications. Geometry and mesh generated with Salome [2].

Since the $z$-component is zero, one may then simply switch places of the $x$- and $y$-components and times on of them with $-1$.

\[ \mathbf{n} = \begin{bmatrix} \pm 0.6[0.14845^{-0.5x/c} - 0.126 - 0.7032(x/c) + 0.8529(x/c)^2 - 0.406(x/c)^3], -1, 0 \end{bmatrix}^T / \text{norm}(\mathbf{t}^*) \]

(28)
4 Case Setups

4.1 Cube

The cube test case is made to try out the capabilities of the flow part of the solver to catch flow scenarios with large amounts of separation. Therefore, a cube aligned with the xyz-axes and corners in (0.6,0.4,0.4) and (0.8,0.6,0.6) is placed in a square tunnel, also aligned with the coordinate axes, with corners in (0,0,0) and (3,1,1). The Initial mesh is composed of 18 000 uniformly sized tetrahedrons. The parameters used for simulating the flow solution are shown in table 3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nu )</td>
<td>([2 \cdot 10^{-3}, 2 \cdot 10^{-5}, 2 \cdot 10^{-7}] ) [m(^2)/s]</td>
</tr>
<tr>
<td>( \rho )</td>
<td>1 [kg/m(^3)]</td>
</tr>
<tr>
<td>Number of adaptive iterations</td>
<td>10</td>
</tr>
<tr>
<td>Percentage of cells to be refined/adaptive iteration</td>
<td>10</td>
</tr>
<tr>
<td>Time span</td>
<td>([0,10]) [s]</td>
</tr>
<tr>
<td>Inflow velocity</td>
<td>((1,0,0)) [m/s]</td>
</tr>
<tr>
<td>Outflow</td>
<td>( p = 0 ) [Pa]</td>
</tr>
<tr>
<td>Tunnel wall &amp; cube</td>
<td>Slip</td>
</tr>
</tbody>
</table>

Table 3: Parameters used for cube flow computation.

4.2 NACA0012 Airfoil

In order to validate also lift and drag, and their origins, the flow around a NACA0012 will also be computed.

The geometric and aerodynamic characteristics are defined in table 4 and the BCs and computational parameters in table 5.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Span</td>
<td>2.7 [m]</td>
</tr>
<tr>
<td>Chord</td>
<td>0.76 [m]</td>
</tr>
<tr>
<td>Wing profile</td>
<td>NACA0012*</td>
</tr>
<tr>
<td>Tunnel corners in</td>
<td>((-1.52,-1.35,-1.35)) and ((3.8,1.35,1.35)) [m]</td>
</tr>
<tr>
<td>Geometric center of wing</td>
<td>((0.38,0,0)) [m]</td>
</tr>
<tr>
<td>Axis of rotation</td>
<td>((0.38,0,x)) [m]</td>
</tr>
<tr>
<td>( \nu )</td>
<td>0 [m(^2)/s]</td>
</tr>
<tr>
<td>( \rho )</td>
<td>1 [kg/m(^3)]</td>
</tr>
</tbody>
</table>

Table 4: \( y_c = \pm 0.6[0.2969\sqrt{x/c} - 0.126x/c - 0.3516(x/c)^2 + 0.2843(x/c)^3 - 0.1015(x/c)^4]\)[5]

4.3 Flutter plate

In order to validate the fluid structure interaction functionality, a flutter experiment has been set up. This test case is based on a flutter experiment made by Majid and Basri [9], but with the approximations that the material is incompressible (instead of poisson's number = 0.143) and isotropic (instead of transversally isotropic) since, at the time of the simulations, this is what the fluid structure interaction functionality could handle. The structural, aerodynamical
Table 5: Boundary conditions for the NACA0012 test case.

and geometrical parameters used for this test case are presented in table 6, mesh information is specified in table 2 and Boundary conditions and computational parameters in table 7.

Table 6: Flutter plate geometric, aerodynamic and structure parameters.

Table 7: Flutter plate Boundary conditions and computational parameters.

The reason that no slip is needed between the flutter plate and the wind tunnel wall is that it otherwise would allow for the wing to slide on the wall.

These specifications did however prove to require a lot of effort to fulfill. This, because of the fact that the geometry of the output corresponded not exactly, but roughly to the input, requiring trial and error specification of location of slip and no slip BCs. Therefore, a new method of fulfilling the no slip BC between plate and wall was developed. This was by making solid walls surrounding the fluid tunnel and using no slip on the entire outer boundary.
5 Results

5.1 Cube

The cube flow simulations gave $C_D = 1.04$ and $C_D = 1.06$, after 10 iterations in a mean value sense, for the cases with $\nu = 2 \cdot 10^{-5}$ and $2 \cdot 10^{-7}$, respectively (giving Reynolds numbers of $10^4$ and $10^6$). This is in accordance with [10] and [3], who state that $C_D$ for a cube should be 1.05 for $Re > 10^4$ and $Re = 10^5$ respectively. In order to show more apparent convergence, another 4 adaptive iterations were made for the $Re=10^4$ case with the parallel solver. The convergence is presented in figure 9.

![Figure 9: Drag coefficient time plots for serial and parallel computations at different levels of refinement. (Updated drag computation algorithm)](image)

These results led to the suspicion that the updated force computation algorithm in the parallel solver might be wrongly implemented or wrong. Therefore the old algorithm, which is simply a numerical surface integral of the pressure on the object was tried as well. Those results are presented in figure 10. Figure 11 shows the logarithmic drag convergence with the error in iteration $i$ estimated as $|C_D^n - C_D^i|$ where $n$ is the maximum amount of iterations (in this case 11).
Figure 10: Drag coefficient time plots for serial and parallel computations at different levels of refinement. (Simple drag computation algorithm)

Figure 11: Logarithmic drag coefficient convergence plot (Simple drag computation algorithm)
5.2 NACA0012 Airfoil

5.2.1 Lift and drag

The initial mesh for the simulations was generated for five different angles of attack from 0 to 14 degrees. These were simulated using the serial unicorn version with initial mesh sizes of roughly 20000 nodes and domains 1, 2, 3 and 4, described in section 3.1. This resulted in the convergence pattern described by figure 12.

![Figure 12: Lift and drag coefficients as functions of angle of attack for different iterations.](image)

It clearly shows that the drag gets highly over estimated as the angle of attack is increased. It also shows that the lift curve initially has (even though a slightly premature stall) fairly good agreement with experiments, but as the mesh is adaptively refined, stall shows tendencies of getting more and more premature, see figures 30 and 31 in the appendix.

![Figure 13: Longitudinal velocity component distribution at t_{final} and 12 degrees a.o.a. Iteration 1 above and Iteration 0 below.](image)
As figure 13 shows, the separation is more intense for the refined mesh. This result agrees with the lift curves but contradicts the drag curves, since more separation tends to reduce the lift and increase the drag. Therefore, it was believed that the refinement created more rapid changes in normals, while it also was refined the wake, so that the "vortex sausages" got more resolved, see figure 14. Another idea was that the fact that the normals were computed from the mesh might have caused problems, being slightly different from the ones of the true geometry. Furthermore it was reasoned that the leading edge might not have been resolved enough. In order to rule out the normal error as reason for the difference between the simulation and the experiments, the mesh normals were exchanged for the analytical one as described in section 3.2, for the 10 degrees a.o.a. As can be seen in figure 15, this did not make a considerable change.

Figure 14: Cells marked for refinement at 12 degrees a.o.a. Iteration 1 is shown as wireframe and Iteration 0 as surface and red means marked.

Therefore more resolved meshes were used instead, which also required use of the parallel unicorn version in order for the simulations to be finished in reasonable time. Since a limited amount of servers were available, these simulations were only made for 10, 15 and 18 degrees angle of attack. This because the larger angles were the ones with the most pronounced error in lift and drag. The mesh size was chosen to be around 100 000 nodes, see table 1. While this was made, another issue was however encountered, namely the fact that using edges enforced no slip-like conditions at the trailing edge, and therefore also creating a "low speed bubble" around the trailing edge, see figure 16. This lead to the conclusion that edges might destroy the solution and thus should not be used. Note that the edge/corner conditions were put there by the software, for all edges/corners between neighboring surface elements that had an inclination of π/6 or more.

However a new problem arised from taking them away entirely. When there were no edges in the junction between the wing and the wind tunnel walls, jets of span wise flow appeared, see figure 17. Though not wanted, neither of the artifacts had a major impact on the mean value results. The lift and drag coefficients of the cases are presented in table 5.2.1.

The results for above mentioned angles of attack and edge conditions on the wind tunnel
The reason that results from no more iterations are available (especially for 15 deg) is that at this point in the simulation, the difference between the aerodynamic force computations between the serial solver and the parallel one, pointed out in section 5.1, was discovered. Therefore, and in order to save time, another, smaller mesh was used only for 10 degrees a.o.a, with unit chord and span and wind tunnel corners in $[-2, -0.5, -0.5]$ and $[5, 0.5, 0.5]$. This mesh was used for both aerodynamic force computation algorithms. The resulting force coefficient plots are presented in figure 18. Here, the supercomputer results have been plotted as 8 iterations, since the uniform refinement corresponds to at least 4 adaptive refinements.

In order to check if the results are converging towards some final result, a mesh with 800K...
Table 8: Lift and drag coefficients for different edge boundary conditions.

<table>
<thead>
<tr>
<th>Edge boundary condition</th>
<th>$C_L$</th>
<th>$C_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge conditions on all edges</td>
<td>1.11</td>
<td>0.13</td>
</tr>
<tr>
<td>No edge conditions at all</td>
<td>0.93</td>
<td>0.10</td>
</tr>
<tr>
<td>No edge condition on TE only</td>
<td>0.93</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Figure 17: Span wise jet, from no edge condition at wing/wall junction. Red is high velocity magnitude.

Figure 18: Comparison between simple, new force computation algorithm for 10 degrees a.o.a for short span wing and experiments (figures 30 and 31).
nodes have been used on a 264 core computer as well. This resulted in the lift coefficients 1.05 and 1.07 and drag coefficients 0.0576 and 0.0622 for the simple and new algorithms respectively.

### 5.2.2 Circulation

The Circulation patterns were also investigated for the wing simulations, and clearly showed the presence of both starting vortices and wing section circulation. In order to show the circulation qualitatively, the mean velocity was subtracted from the computed velocity field and then plotted. Figure 19 shows the initial result. As time goes, the starting vortex is translated away in the flow direction and finally the flow pattern shows only the large scale circulation around the wing seen in figure 20.

Figure 19: Circulation initially including starting vortex. $U - U_{\text{mean}}$ plotted.

This can be compared by the experimental picture of aluminum powder, photographed by a camera which is at rest relative to the undisturbed fluid, see figure 21.

In order to check if the circulation also corresponds to the lift quantitatively as well a rough estimation has been done. The path chosen for integration is shown in figure 22. This path was chosen simply because it is easy to integrate the velocity along the path, if the path is aligned with the coordinate axes. The integration gave a section lift coefficient of 0.55, which should be compared with the 10 degree a.o.a $C_L$ of table 9, iteration 2. in order to to check that the

<table>
<thead>
<tr>
<th>$C_D$</th>
<th>$\alpha = 10$</th>
<th>15</th>
<th>18</th>
<th>$C_L$</th>
<th>$\alpha = 10$</th>
<th>15</th>
<th>18</th>
<th>$L/D$</th>
<th>$\alpha = 10$</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter 0</td>
<td>0.105</td>
<td>0.283</td>
<td>0.360</td>
<td>0.930</td>
<td>1.03</td>
<td>0.638</td>
<td>8.83</td>
<td>3.62</td>
<td>1.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iter 1</td>
<td>0.0683</td>
<td>0.155</td>
<td>0.231</td>
<td>0.693</td>
<td>0.673</td>
<td>0.747</td>
<td>10.1</td>
<td>4.33</td>
<td>3.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iter 2</td>
<td>0.0592</td>
<td>-</td>
<td>0.210</td>
<td>0.624</td>
<td>-</td>
<td>0.746</td>
<td>10.5</td>
<td>-</td>
<td>3.55</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Lift and drag coefficients computed as mean from $t_{\text{final}}/2$ to $t_{\text{final}}$. A.o.a given in degrees.
integration path is independent of trajectory, two more paths have been integrated. The path corners and integration results are shown in table 5.2.2.

in order to check if the integration path is independent of trajectory, two more paths have been integrated. The path corners and integration results are shown in table 5.2.2.

<table>
<thead>
<tr>
<th>Path corners</th>
<th>$C_t$ from circulation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[-0.1, -0.2, 0], [1, 0.2, 0]$</td>
<td>0.55</td>
</tr>
<tr>
<td>$[-0.5, -0.5, 0], [0.5, 0.5, 0]$</td>
<td>0.40</td>
</tr>
<tr>
<td>$[-1, -1, 0], [2, 1, 0]$</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Table 10: Circulation for different rectangular integration paths.

5.2.3 Stream wise vorticity

The tube shaped vortices described in section 2.1.4 are indeed present in the simulations, and are depicted in figure 23.

Note however, the fact that these sausages are of the same size for the x-component of the vorticity as for the y-component and that they are present mainly where there is separation (see figure 24).

Another result that might be of interest is the fact that the upper side of the wing has considerably more vorticity than the lower one. This is depicted in figure 25 and 26.

5.2.4 Surface flow

The results presenting the stream wise-/normal to surface- vorticity go in line with the experiments shown in figure 32 in the appendix. This is clarified in figure 27, showing the normal
to surface vortices, responsible for smearing out the china clay in circular patterns and creating what [5] refer to as stall cells.

Though the figures look similar to the experiments, it should be noted that these separation patterns occur much earlier for the simulations than in the experiments. Figure 27 is taken from a low resolution 10 degree a.o.a run with edge condition at the trailing edge. For the high resolution simulations these patterns seem to emerge at somewhat higher angles of attack (after 10 but before 15 degrees). Note that the experiments showed these patterns at 15-17 degrees a.o.a [5].

5.3 Flutter plate

The flutter plate simulation was unsuccessful in the sense that stationary solutions were obtained for the 0 a.o.a simulations. Tries were however made for plates with large angles of attack, which showed oscillatory motion of the plate as it converged towards some mean deflection. Since these results have no experiments for comparison, they may work as a technology demonstration rather than as a validation of the solver. The angular and vertical displacements of the two cases can be seen in figures 28 and 29.
Figure 22: Path for integration of circulation, top view and side view.

Figure 23: x- and y-components of the vorticity.
Figure 24: Blue regions represent separated flow and red ones attached.

Figure 25: Vorticity magnitude, lower surface.
Figure 26: Vorticity magnitude, upper surface.

Figure 27: Computed surface flow during stall. Snap shot below and 5 seconds of low opacity velocity vectors above.
Figure 28: Computed vertical and angular displacements for 0 degree a.o.a plate tip.

Figure 29: Computed vertical and angular displacements for 15 degree a.o.a plate tip.
6 Discussion

6.1 Circulation and streamwise vorticity

 Apparently Johnson and Hoffman’s theory results in both large scale circulation around the section of a wing and a starting vortex. Furthermore, the circulation is of the right order of magnitude compared to the lift even though not independent of where it is integrated. Truly remarkable however, is the fact that this is a result from inviscid computations, since the common explanation of generation of circulation comes from the viscous BL, and gives rise to some questions that might need further investigation to be answered.

- Are the vortex sausages present in the computations from other state of the art software and can they be seen in experiments?

- Is there a way to make simplified models of airflow by connecting the x-y-plane vorticity to the chordwise circulation and create more accurate vortex lattice types of aerodynamic force estimation methods?

- Is it possible to avoid this friction by allowing all velocity directions not crossing the surface at the nodes? or could it be possible to avoid by using curved elements (superparametric)?

- Why is the circulation reducing as the integration path is moved away from the wing?

- It seems, qualitatively like the streamwise vorticity mainly is present where separation is present. Could this be the explanation of separation rather than lift and drag?

6.2 Lift and drag

 It seems like the lift coefficient converges toward the experimental values, but that drag estimations are too high. A list of proposed explanations is presented as follows:

- It is argued in [7] to be a result from the wind tunnel tests showing results from low Reynolds number flows, and comparisons are being made between L/D for Unicorn simulations and for full scale airliners. The author however, argues that the difference in L/D between these wind tunnel tests and full scale aircraft is mainly due to wing tip vortices, fuselage drag and drag from horizontal and vertical stabilizers. Therefore, the author suggests that simulations should be made for a mesh of a full scale, low speed aircraft with Unicorn.

- One proposition is that a small error in the total aerodynamic force vector direction would, as a result of the large difference between lift and drag, give a large impact on the drag but not on the lift.

- Another proposed explanation for the difference in drag, compared to [5] is that the trailing edge and wake might need to be further resolved in order for the wake, especially close to the wing to include smaller vortices.

- Finally, one idea for explaining the drag is that it could be a result from artificial friction, created by the velocity vector projection made by the slip condition at the nodes.
6.3 Edge conditions

The fact that edges seem to create no slip like conditions, could according to the author be an explanation to the cube simulations converging towards the real drag coefficient, even though the difference between edge and no edge was quite small in some wing cases. However, in order to look closer into the validity of this explanation, the author suggests that the cube simulations should also be performed without edges, to see if this results in a highly different drag coefficient. Furthermore, since these simulations have been made with non zero viscosity, it would also be preferred to try no edge in combination with zero viscosity. One way to solve the edge issue entirely could be to use a cube with filleted edges. Another example of bad influence from edge conditions in the simulations is that this condition has in some cases been applied on the leading edge, due to poor resolution, creating premature stall.

6.4 Geometric error

In order to avoid a too sharp leading edge creating premature stall, a lot of effort has been put into using a mesh which is fine enough at the leading edge. This is a problem, both because it takes a lot of time to get a satisfactory mesh and because of the fact that it is likely to result in an unnecessarily fine mesh. Therefore it would be a large advantage if it would be possible to use a truly coarse mesh and a geometry file that ensures the added nodes from the adaptive refinement are placed on the actual geometry rather than in between the nodes on the mesh.

6.5 Flutter plate

The stationary solution of the scenario that in experiments has proved to generate flutter may be a result of many reasons, and the ones the author can think of are listed below.

- The computations have been made for isotropic structure, while the experiment was made with glass fiber-epoxy composite.
- The total force perturbations of the stationary solution might have been too small for flutter to be triggered.
- The flow part of the problem might estimate \( \frac{\partial C_l}{\partial \alpha} \) and the aerodynamic damping in a non-conservative way. This might be a result of a too poorly resolved fluid.

This shows that the problem might be too complex for an early validation of a fluid-structure interaction solver. An option that would probably be more efficient is to first validate the flow solver with a simpler structural model, either by using a mass-spring model or by setting a prescribed motion of the plate/wing and comparing these results with corresponding experiments.

7 Conclusions

- The Cube simulations have been very successful, showing convergence towards experimental drag coefficients within a few percent.
- The results presented in [6] have been reproduced with a new software framework, and lift seems to be accurate in comparison with experiments.
- In order to straighten out the argument regarding the impact from wing tips and fuselage, etcetera, a simulation should be made for a full scale aircraft.
• In order to give answers to the questions regarding the impact from edges creating parts of no slip like conditions, no edge simulations of the flow around a cube should be carried out. For now, however it seems like no edges and fillets would be the best way, to avoid the need of ad hoc edge placement.

• In order to avoid artifacts like the jet in figure 17, the use of fillets on edges, rounding them off is proposed.

• The automatic movement of new nodes to the exact geometry at adaptive refinement is concluded to be crucial for effective use of adaptivity and initial mesh size.

• In order to validate the flow solver for unsteady boundaries, the author recommends either using a simple well tried structural model, like the pitch-plunge case, or a prescribed boundary movement like the one described by [4].

• For the fluid-structure interaction functionality to be useful for aeroelastic stability analysis, the author recommends trying some way of including perturbations that may trigger an unstable behaviour.

• One conclusion to be made however, can be drawn from [8]. This is the fact that the fluid-structure interaction version of unicorn does reproduce the results of the Hron-Turek 2D benchmark problem [11] within a few percent.
References


Appendix

Figure 30: Experimental $C_L - \alpha$-curves. [5]
<table>
<thead>
<tr>
<th>x</th>
<th>±y</th>
<th>x</th>
<th>±y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.5120819</td>
<td>0.0521620</td>
</tr>
<tr>
<td>0.0005839</td>
<td>0.0042603</td>
<td>0.5362174</td>
<td>0.0505161</td>
</tr>
<tr>
<td>0.0023342</td>
<td>0.0084289</td>
<td>0.5602683</td>
<td>0.0487619</td>
</tr>
<tr>
<td>0.0052468</td>
<td>0.0125011</td>
<td>0.5841786</td>
<td>0.0469124</td>
</tr>
<tr>
<td>0.0093149</td>
<td>0.0164706</td>
<td>0.6078921</td>
<td>0.0449802</td>
</tr>
<tr>
<td>0.0145291</td>
<td>0.0203300</td>
<td>0.6313537</td>
<td>0.0429778</td>
</tr>
<tr>
<td>0.0208771</td>
<td>0.0240706</td>
<td>0.6545085</td>
<td>0.0409174</td>
</tr>
<tr>
<td>0.0283441</td>
<td>0.0276827</td>
<td>0.6773025</td>
<td>0.0388109</td>
</tr>
<tr>
<td>0.0369127</td>
<td>0.0311559</td>
<td>0.6996823</td>
<td>0.0366700</td>
</tr>
<tr>
<td>0.0465628</td>
<td>0.0344792</td>
<td>0.7215958</td>
<td>0.0345058</td>
</tr>
<tr>
<td>0.0572720</td>
<td>0.0376414</td>
<td>0.7429917</td>
<td>0.0323294</td>
</tr>
<tr>
<td>0.0690152</td>
<td>0.0406310</td>
<td>0.7638202</td>
<td>0.0301515</td>
</tr>
<tr>
<td>0.0817649</td>
<td>0.0434371</td>
<td>0.7840324</td>
<td>0.0279828</td>
</tr>
<tr>
<td>0.0954915</td>
<td>0.0460489</td>
<td>0.8035813</td>
<td>0.0258337</td>
</tr>
<tr>
<td>0.1101628</td>
<td>0.0484567</td>
<td>0.8224211</td>
<td>0.0237142</td>
</tr>
<tr>
<td>0.1257446</td>
<td>0.0506513</td>
<td>0.8405079</td>
<td>0.0216347</td>
</tr>
<tr>
<td>0.1422005</td>
<td>0.0526251</td>
<td>0.8742554</td>
<td>0.0176353</td>
</tr>
<tr>
<td>0.1594921</td>
<td>0.0543715</td>
<td>0.8898372</td>
<td>0.0157351</td>
</tr>
<tr>
<td>0.1775789</td>
<td>0.0558856</td>
<td>0.9045085</td>
<td>0.0139143</td>
</tr>
<tr>
<td>0.1964187</td>
<td>0.0571640</td>
<td>0.9182351</td>
<td>0.0121823</td>
</tr>
<tr>
<td>0.2159676</td>
<td>0.0582048</td>
<td>0.9309849</td>
<td>0.0105485</td>
</tr>
<tr>
<td>0.2361799</td>
<td>0.0590081</td>
<td>0.9427280</td>
<td>0.0090217</td>
</tr>
<tr>
<td>0.2570083</td>
<td>0.0595755</td>
<td>0.9534372</td>
<td>0.0076108</td>
</tr>
<tr>
<td>0.2784042</td>
<td>0.0599102</td>
<td>0.9630873</td>
<td>0.0063238</td>
</tr>
<tr>
<td>0.3003177</td>
<td>0.0600172</td>
<td>0.9716559</td>
<td>0.0051685</td>
</tr>
<tr>
<td>0.3226976</td>
<td>0.0599028</td>
<td>0.9791229</td>
<td>0.0041519</td>
</tr>
<tr>
<td>0.3454915</td>
<td>0.0595747</td>
<td>0.9854709</td>
<td>0.0032804</td>
</tr>
<tr>
<td>0.3686463</td>
<td>0.0590419</td>
<td>0.9906850</td>
<td>0.0025595</td>
</tr>
<tr>
<td>0.3921079</td>
<td>0.0583145</td>
<td>0.9947532</td>
<td>0.0019938</td>
</tr>
<tr>
<td>0.4158215</td>
<td>0.0574033</td>
<td>0.9976658</td>
<td>0.0015870</td>
</tr>
<tr>
<td>0.4397317</td>
<td>0.0563200</td>
<td>0.9994161</td>
<td>0.0013419</td>
</tr>
<tr>
<td>0.4637826</td>
<td>0.0550769</td>
<td>1.0000000</td>
<td>0.0012600</td>
</tr>
<tr>
<td>0.4879181</td>
<td>0.0536866</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11: NACA0012 coordinates normalized with respect to chord length.
Figure 31: Experimental $C_D - \alpha$-curves. [5]

Figure 32: Experimental china clay surface low visualization for NACA0012 at 17 degrees a.o.a. [5]