Development of a fast aerodynamic tool using meta-modeling techniques

R. Veldhuizen

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

DEVELOPMENT OF A FAST AERODYNAMIC TOOL USING META-MODELING TECHNIQUES

by

R. Veldhuizen

in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Friday April 17th, 2015 at 13:00.
Thesis Registration Number: 025#15#MT#FPP

Student number:	1369202	
Supervisor:	Dr. ir. R. Vos	
Thesis committee:	Prof. dr. ir. L. L. M. Veldhuis,	TU Delft
	Dr. ir. F. F. J. Schrijer,	TU Delft
	Dr. Ing. M. Weismüller	Airbus Operations GmbH

SUMMARY

There is a need for a wave drag method that combines the speed of handbook methods with the accuracy of computational methods. Especially determining the onset of wave drag as well as the initial drag rise is important in initial design stages. Meta-models allow this by capturing the trends present in previously computed data, providing an accurate and fast representation. In this report, it is investigated what gains can be achieved by applying the meta-modeling method GT-Approx to the aerodynamic tool MSES.
The total drag calculated by MSES for a supercritical airfoil was verified using wind tunnel experiments. It was found that aerodynamic characteristics and pressure distributions are accurate up until $M = 0.76$.

GT-Approx was critically evaluated to determine what accuracies are attainable, and how the resolution of the input data influences these accuracies. Initially the fitting capabilities of GT-Approx are tested. With a maximum error of 0.11% the fit is accurate. The errors increase when GT-Approx is used to predict values for data points that are not present in the data set. Especially in the case of extrapolation, the errors grow exponentially. If GT-Approx is used to predict values in between data-points, thus performing an interpolation, the results are better, with average errors of 2 drag counts. However, when predicting difficult regions, such as the drag coefficient near the drag rise, the errors increase. The prediction errors for high M are large, reaching 6.1 drag counts for c_{d_w} and 5.9 drag counts for c_{d_v}.
In an effort to reduce these errors, the influences of resolution increases are investigated. Upon increasing the resolutions, the errors decrease. Especially increasing the M-resolution and c_l-resolution yields large improvements. The average error for the higher Mach numbers reduces from 6.0 drag counts to 2.0 drag counts if the resolution is quadrupled. The average errors also reduce significantly, as shown in Table 1.

	Initial			**Final**		
Variable	**Resolution**	$\Delta \mathbf{c_{d_v}}[\times \mathbf{10^{-4}}]$	$\Delta \mathbf{c_{d_w}}[\times \mathbf{10^{-4}}]$	**Resolution**	$\Delta \mathbf{c_{d_v}}[\times \mathbf{10^{-4}}]$	$\Delta \mathbf{c_{d_w}}[\times \mathbf{10^{-4}}]$
Re	20.0×10^6	1.85	0.38	5.00×10^6	1.63	0.20
c_l	0.1	1.73	1.00	0.025	1.33	0.42
M	0.1	3.15	3.05	0.025	1.27	0.40
$\frac{t}{c}$	0.02	2.36	0.69	0.01	2.33	0.51

Table 1: Overview of initial and final resolutions and their accuracy.

Two A320 variants are evaluated using GT-Approx and a direct application of MSES. The performance of GT-Approx is good. An average difference of 0.21 drag counts between MSES and GT-Approx was achieved, with an in-calculation computation time of 5.13×10^{-4} s per calculation instead of 5.58 s using a direct application of MSES.
GT-Approx is extended to a quasi-3D method, using the simple sweep method. This quasi-3D method is used to calculate the value of C_{D_w} for two test cases. The calculated values of C_{D_w} are compared with CFD data.
It was found that the region of validity of the quasi-3D method is highly limited. Up until 60% of the wing, root and tip effects, fuselage effects and engine installation effects render any comparison useless. Beyond this value the first test case showed no correlation, whereas the second showed reasonable accuracy. Due to lack of more 3D CFD data, no clear explanation for the difference was found.
In general it is concluded that the combination of an aerodynamic tool with a meta-model is able to combine low computation times with high accuracy, but only if the aerodynamic model is accurate.

ACKNOWLEDGMENTS

This report is part of a Master Thesis, performed as a part of the Master Flight Performance and Propulsion at the Faculty of Aerospace Engineering of the Delft University of Technology. This Master Thesis was performed at the Non-Specific Design Skills - Weights and Aerodynamics department of the Future Projects Office at Airbus Operations GmbH.

First of all, I would like to thank everybody at Airbus. Specifically Michael Weismüller, for making it possible for me to perform my Master Thesis within Airbus and being my daily supervisor. I would like to thank him for providing excellent advice and clear guidance, for providing me with the necessary information and data and all the help given. Thanks to Martin Schimmöller and all my other colleagues of EIXDS for accepting me into the team, and providing a helpful and constructive working environment.
I would also like to thank Bruno Moorthamers and Ali Cabac for helping me find the right contacts within Airbus, leading to this thesis.

I would also like to express my gratitude towards Roelof Vos, my supervisor at the TU Delft. He has proven invaluable. Without his critical remarks, thought provoking questions, useful suggestions and meticulous reading this thesis would not be the same.
Furthermore I would like to thank Ferry Schrijer and Leo Veldhuis for completing my graduation committee.

Finally I would like to thank all my friends in Delft, Tianjin, Hamburg and all other places around the world for making my study time an enjoyable one.
Lastly I would like to thank my family for supporting me during my studies.

Roy Veldhuizen
Delft, April 7, 2015

CONTENTS

List of Figures

List of Tables

NOMENCLATURE

c	=	Airfoil chord, [m]
$c_\perp$	=	Effective airfoil chord, perpendicular to the sweepline, [m]
c_d	=	2D drag coefficient, [-]
c_{d_v}	=	2D viscous drag coefficient, [-]
c_{d_w}	=	2D wave drag coefficient, [-]
c_f	=	2D friction coefficient, [-]
c_l	=	2D lift coefficient, [-]
$c_{l_{des}}$	=	2D design lift coefficient, [-]
c_m	=	2D moment coefficient
c_p	=	2D pressure coefficient, [-]
C_{D_w}	=	3D wave drag coefficient, [-]
C_L	=	3D lift coefficient, [-]
C_P	=	3D pressure coefficient, [-]
h_{0_∞}	=	Free stream specific total enthalpy, [m^2s^2]
$\dot{m}$	=	Mass flow, [kgs^{-1}]
M	=	Mach number, [-]
M_∞	=	Free stream Mach number, [-]
M_{cr}	=	Critical Mach number, [-]
M_{DD}	=	Drag divergence Mach number, [-]
$M_{\text{DD}_{c_l=0.40}}$	=	Drag divergence Mach number at $c_l = 0.40$, [-]
$M_{\text{DD}_{c_l=0.55}}$	=	Drag divergence Mach number at $c_l = 0.55$, [-]
$M_{\text{DD}_{c_l=0.70}}$	=	Drag divergence Mach number at $c_l = 0.70$, [-]
$M_{L_{\text{SH}}}$	=	Mach number just ahead of the shock, [-]
$M_\perp$	=	Effective Mach number, perpendicular to the sweepline, [-]
MAC	=	Mean Aerodynamic Chord, [m]
p_{exit}	=	Pressure at exit plane, [Nm^{-2}]
p_∞	=	Free stream pressure, [Nm^{-2}]
q_{exit}	=	Velocity at exit plane, [ms^{-1}]
$q_{+\infty}$	=	Velocity at infinity behind airfoil, [ms^{-1}]
Re	=	Reynolds number, [-]
Re_{wing}	=	Reynolds number based on the mean aerodynamic chord, [-]
$Re_\perp$	=	Effective Reynolds number, perpendicular to the sweepline, [-]
$\frac{t}{c}$	=	Thickness to chord ratio, [-]
V_∞	=	Free stream velocity, [ms^{-1}]
$V_\perp$	=	Effective free stream velocity, perpendicular to the sweepline, [ms^{-1}]
$\frac{x}{c}$	=	Chordwise distance to chord ratio, [-]
X	=	Distance from the nose along the symmetrical axis of the aircraft, [m]

α	=	Angle of attack, [deg]
$\Delta\frac{t}{c}$	=	Thickness-ratio calculation interval, [-]
$\Delta_{c_{d_{combined}}}$	=	Combined wave drag coefficient error, [-]
$\Delta c_{d_{\Delta c_l}}$	=	Fitting difference as a function of the 2D lift coefficient calculation interval, [-]
$\Delta c_{d_{\Delta M}}$	=	Fitting difference as a function of the Mach number calculation interval, [-]
$\Delta c_{d_{\Delta Re}}$	=	Fitting difference as a function of the Reynolds calculation interval, [-]
$\Delta c_{d_{\Delta\frac{t}{c}}}$	=	Fitting difference as a function of the thickness calculation interval, [-]
Δc_{d_v}	=	2D viscous drag coefficient difference between GT-Approx and MSES, [-]
Δc_{d_w}	=	2D wave drag coefficient difference between GT-Approx and MSES, [-]
Δc_l	=	2D lift coefficient calculation interval, [-]
Δ_{fit}	=	Average fitting difference between data and GT-Approx, [-]
ΔM	=	Mach calculation interval, [-]
ΔRe	=	Reynolds number calculation interval, [-]
η	=	Relative spanwise wing station, based on original A320 semi-span [-]
κ_A	=	Airfoil technology factor, [-]
κ_w	=	Airfoil curvature, $\frac{z-z_{\text{sh}}}{x-x_{\text{sh}}}$, [-]
Λ	=	Sweep angle, [deg]
$\Lambda_{0.25c}$	=	Quarter chord sweep angle, [deg]
$\Lambda_{0.50c}$	=	Half chord sweep angle, [deg]
ρ_∞	=	Free stream air density, [kgm^{-3}]
ν_∞	=	Free stream air kinematic viscosity, [m^{2}s]

1

INTRODUCTION

Fossil fuels are getting scarcer [1], and there is a driving demand to lower the fuel usage of aircraft. Most of the current day transport aircraft fly at transonic speeds [2], and as consequence of this, experience additional drag caused by shock waves. In a good aircraft design, wave drag is a relatively small part of the total drag in cruise [3]. But if a wing is not designed properly, wave drag can become significantly large [4].
Transonic flows are hard to calculate because both subsonic and supersonic flows coexist [5]. It is possible to accurately calculate these flows using high fidelity CFD (Computation Fluid Dynamics) methods, but these methods are not feasible for early design stages. Also, even CFD codes cannot accurately predict flows in high transonic conditions. In the conceptual design phase emphasis lies on quick and low fidelity methods. Such methods exist for both subsonic [6] as well as supersonic flows [7], but no such method is available for transonic flows. This poses difficulties in early design stages, as it is hard to predict loads with fast and accurate methods. In practice, either handbook methods or computational methods are used. However, both are not adequate for this tasks. The handbook methods are quick, as they rely on general trend lines and empirical methods to provide answers, but they lack accuracy [8] and are not able to evaluate new, or radically different airfoils. Computational methods are very flexible, but have significant computing times, and sometimes have issues converging [9].
In the initial design stages many important and far-reaching decisions have to be made [10]. Therefore there is a need for a method which delivers the required accuracy, but without large computation times. This allows a better trade-off to be made in initial design stages.

The main question answered in this thesis is:

What possibilities are there to improve the stability, accuracy and speed of wave drag models by applying meta-modeling techniques?

This main question is broken down into the following subquestions:

- *What wave drag model is able to effectively predict wave drag?*
- *What meta-model method is able to effectively represent the wave-drag model?*
- *What possibilities are there in combining a wave drag model with a wave drag model?*

This project investigates the possibility to combine an accurate wave drag model with meta-modeling methods, thus providing an accurate answer with low computation times. A meta-model is in essence, a model of a model [11]. Thus, several calculations of a model are performed, and from these calculations trends are recognized. Based on these trends, a new model is developed, the meta-model. This method is being used more and more in the last decades [12]. It has also been used before in aerospace optimizations [13–15].
Chapter 2 treats current and proposed methods for wave drag prediction. These methods are compared, and a suitable aerodynamic method is chosen. Then, in Chapter 3 a general overview of meta-modeling methods is given, after which a suitable method is selected. Subsequently, the meta-modeling method is critically evaluated in Chapter 4. The dependency on the input data is thoroughly investigated in Chapter 5. Then the aerodynamic model is combined with the meta-modeling method and extended into a 3D application. This 3D application is then verified against CFD data in Chapter 6. Finally, the conclusions and recommendations are presented in Chapter 7.

2

WAVE DRAG ESTIMATION METHODS

This chapter investigates the different methods that are available for estimating wave drag. Section 2.1 treats handbook methods that are used at Airbus today for airfoil wave drag estimation. Subsequently, Section 2.2 treats computational methods to estimate the wave drag. Finally in Section 2.3 the performance of the different methods is compared. After this an aerodynamic method is selected for further application.

2.1. CURRENT WAVE DRAG ESTIMATION METHOD AT AIRBUS

In this section the current wave drag estimation method in preliminary design stages at Airbus is treated. This method relates airfoil characteristics and flight conditions to the drag divergence Mach number. Subsequently empirical formulas are used to estimate the value of the wave drag coefficient.
The drag divergence Mach number is the Mach number at which the wave drag rapidly increases due to strong shocks and the associated boundary layer separation. For this thesis the value of the drag divergence Mach number M_{DD} is defined as the Mach number at which the value of the derivative of the wave drag first reaches the value of 0.1 [16], as shown in Equation 2.1.

$$\left.\frac{\partial c_{d_w}}{\partial M}\right|_{M=M_{\text{DD}}} = 0.1 \tag{2.1}$$

KORN-LOCK-MASON METHOD (KLM METHOD)

Korn found that the drag divergence Mach number is a function of the thickness to chord ratio $\frac{t}{c}$, lift coefficient c_l, and an airfoil technology factor [17]. The empirical Korn Equation is as follows:

$$M_{\text{DD}} + \frac{c_l}{10} + \frac{t}{c} = \kappa_A \tag{2.2}$$

The value of κ_A is dependent of the airfoil section, and shows how well an airfoil is designed for transonic conditions. For typical current supercritical airfoils it is equal to 0.95. It is possible to apply the simple sweep theory to the Korn equation, as done by Malone and Mason [18]:

$$M_{\text{DD}} = \frac{\kappa_A}{cos\Lambda_{0.25c}} - \frac{\frac{t}{c}}{\cos^2\Lambda_{0.25c}} - \frac{c_l}{10\cdot\cos^3\Lambda_{0.25c}} \tag{2.3}$$

Lock [19] derived an empirical formula for the value of c_{d_w} at Mach numbers higher than M_{cr}, shown in Equation 2.4 .

$$c_{d_w} = 20\,(M - M_{\text{cr}})^4 \tag{2.4}$$

Or combined with the definition of M_{DD} from Equation 2.1:

$$M_{\text{cr}} = M_{\text{DD}} - \left(\frac{0.1}{80}\right)^{1/3} \tag{2.5}$$

It is possible to determine the value of M_{DD} based on airfoil parameters and flight characteristics using Equation 2.3. This value can then be used to calculate the value of M_{cr} using Equation 2.5. Finally it is possible to determine the development of c_{d_w} with M by using Equation 2.4. This method is known as the Korn-Lock-Mason method, and shows reasonable results [5]. At Airbus this method is calibrated to fit existing data, and subsequently used to predict the wave drag for future aircraft.

2.2. COMPUTATIONAL METHODS

For every aerodynamic computation, the Navier-Stokes equations need to be solved. However, difficulties arise in solving these full equations at realistic conditions. Therefore assumptions are applied to simplify the equations. In this subsection two programs using different assumptions are investigated. Both these methods are 2D methods, for simplicity and speed reasons. The two investigated methods are:

- Viscous Garabian-Korn, solution to the full potential equations assuming irrotational, inviscid, isentropic flow, empirically adjusted to incorporate effects of a viscous boundary layer.
- MSES, solving the Euler equations, which ignores viscous terms, coupled with a viscous boundary layer.

VISCOUS-GARABEDIAN-KORN (VGK) METHOD - FULL POTENTIAL EQUATIONS

The VGK-method [20] uses the mixed differencing method. This means that for subsonic flow a central differencing scheme is used, whereas a forward differencing scheme is used when the flow is supersonic. This scheme is applied to the full potential equations coupled with a viscous boundary layer.

Boundary layer evaluation

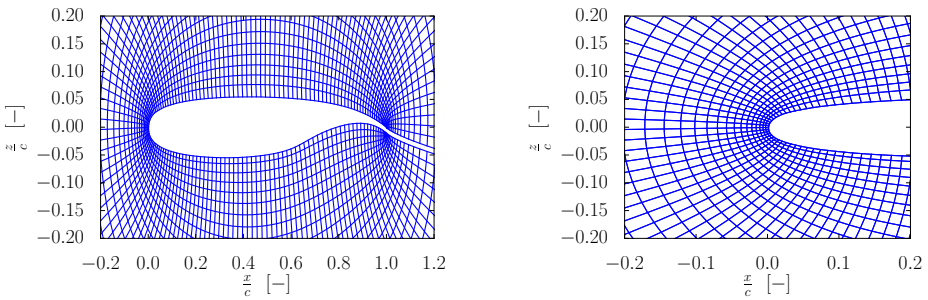
The laminar boundary is calculated with Thwaites' single formula method [21], extended with compressibility effects by the Stewartson-Illingword transformation [22]. VGK employs a turbulent boundary layer method called the lag-entrainment method of Green. In this method, the momentum integral equation, the entrainment equation, and an equation based on the turbulent enerqy equation are solved simultaneously [23].
VGK is a full potential method, and empirical adjustments are used to incorporate the effects of shocks.

Transition

No explicit method is used for transition prediction, and VGK relies heavily on the user to supply the appropriate transition position. Separation prediction is based on the value of the local value of the friction coefficient c_f, following from the boundary layer evaluation. If this value drops below the limit of 2.0×10^{-6}, separation is assumed.

Computational Grid

Initially it performs a number of calculations on a coarse grid with 80 radial lines, and 15 circumferential lines. The resulting potential field is then transferred to the fine grid, which contains 160 radial lines and 30 circumferential lines. As can be seen from Figure 2.1, VGK automatically produces a denser grid near the leading edge and trailing edge, and the circumferential lines are denser near the airfoil.



(a) Computational grid around airfoil

(b) Zoom of the leading edge region

Figure 2.1: Converged computational grid used in MSES.

Wave drag estimation

Wave drag is calculated using the flow field characteristics just ahead of the chock, using the method derived by Lock [24, 25] . The resulting expression is:

$$c_{d_w} = \frac{0.243}{\kappa_w}\left[\frac{1+0.2M_\infty^2}{M_\infty}\right]^3 \frac{\left(M_{L_{\text{SH}}}-1\right)^4\left(2-M_{L_{\text{SH}}}\right)}{M_{L_{\text{SH}}}\left(1+0.2M_{L_{\text{SH}}}^2\right)} \tag{2.6}$$

Where $M_{L_{\text{SH}}}$, is the Mach number just ahead of the shock, and κ_w is the curvature of airfoil around the shock location [20]. Because of its calculation speed, VGK is frequently used in optimizations [26, 27].

MSES - EULER EQUATIONS

MSES 2.95 [28] is an Euler code, coupled with a viscous boundary layer method. These two methods are coupled using the boundary layer thickness. It is a finite volume code, solved on a streamline grid. According to the summary [28]: *"The range of validity includes low-Reynolds numbers and transonic Mach numbers"*. MSES is frequently used for aerodynamic optimizations [29, 30].

Boundary layer evaluation

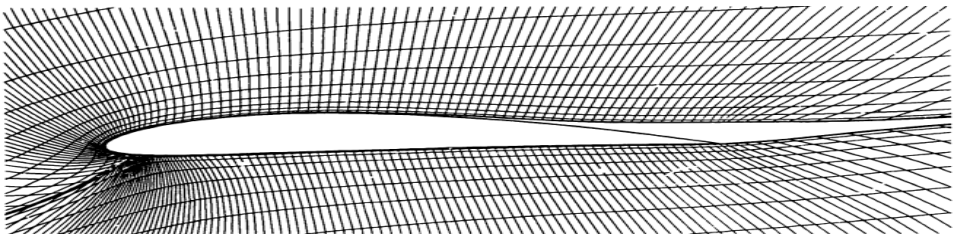
MSES assumes the laminar boundary layer flow to consist of one parameter Falkner-Skan velocity profiles [31]. Based on this assumption, it is possible to provide laminar closure using the momentum and shape parameter equations, together with empirical relations obtained from the solved Falkner-Skan equation [32]. The turbulent flows are modeled following the method of Swafford [33]. To achieve this, an empirical relation for the skin-friction coefficient is used. Using this value, it is possible to construct an estimate for the turbulent velocity profile. This profile is constructed by matching the inner and outer solution. , it consists out of the sum of an inner solution containing the laminar sublayer, and an outer layer, which contains the wake. The turbulent development of the flow is estimated using Green's Lag entrainment method.
Drela states *"The turbulent dissipation coefficient is composed of a wall and wake contribution, each of which is composed of a shear stress scale, and a velocity scale."* [32]
The first of these contributions solely depends on the local conditions, where as the second is only dependent on the upstream history .

Transition

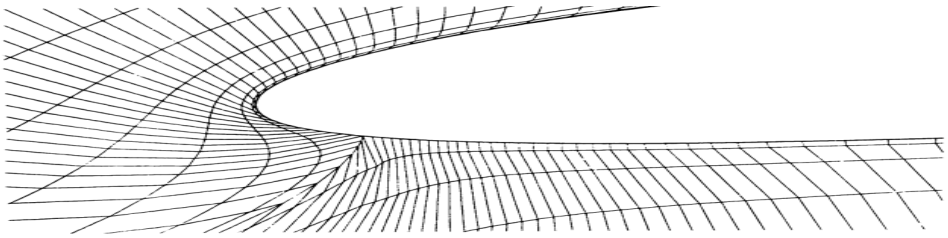
The onset of transition is predicted based on the e^n-method. In order to save computational effort and increase stability, this method is not implemented directly. The amplification equation is discretized and linearized, instead of evaluating the full integral as in the e^n-method.

Computational Grid

The equations are solved on an intrinsic streamline grid, as shown in Figure 2.2. As such, the grid contains a multitude of stream tubes. This greatly simplifies the constant mass and energy equations, as they are transformed to constant mass flux and total enthalpy in a streamtube [32]. Another benefit is that there is no numerical diffusion of entropy or enthalpy, as information only travels along the stream tubes. The only form of interaction between the streamtubes happens via geometry and pressure. It can be seen that around the airfoil the grid is very dense, especially near the leading edge. In MSES, the left edge of the grid is located 4 chords ahead of the leading edge, whereas the right edge of the grid is 5 chords behind the trailing edge, as recommended by the manual [34].



(a) Computational grid around airfoil



(b) Zoom of the leading edge region

Figure 2.2: Converged computational grid used in MSES.

Wave Drag Calculation
Once the computation is converged, the wave drag is calculated in the following way :

1) The pressure and velocity are sampled at the exit plane p_{exit} and q_{exit}
2) The isentropic relation between p_{exit} and p_∞ is used to determine the velocity at infinity, ($q_{+\infty}$):

$$p_{\text{exit}}\left(1-\frac{q_{\text{exit}}^2}{2h_{0_\infty}}\right)^{-\frac{\gamma}{\gamma-1}} = p_\infty\left(1-\frac{q_{+\infty}^2}{2h_{0_\infty}}\right)^{-\frac{\gamma}{\gamma-1}} \tag{2.7}$$

3) This value is used in the integration over all the stream tubes using:

$$c_{d_w} = \frac{2}{\rho_\infty V_\infty^2}\int\left(V_\infty - q_{+\infty}\right)\mathrm{d}\dot{m} \tag{2.8}$$

2.3. PERFORMANCE COMPARISON

In order to assess how accurate all the methods are, they are compared for a well-known test case. To fully asses the transonic capabilities a NASA second phase supercritical airfoil, with a design lift coefficient of 0.7 and a thickness of 11%, the SC(2)-0711 is chosen as the test case. The data is obtained from wind-tunnel experiments performed by Harris and Blackwell [35], airfoil 5. Harris and Blackwell provide a c_d vs M plot, as well as pressure distributions for $c_l = 0.4$, $c_l = 0.55$ and $c_l = 0.70$.

Characteristic	Value
$c_{l_{des}}$	0.70
$\frac{t}{c}$	0.1094
$M_{\text{DD}_{c_l=0.40}}$	0.788
$M_{\text{DD}_{c_l=0.55}}$	-†
$M_{\text{DD}_{c_l=0.70}}$	0.790
$\Lambda_{0.25c}$	0
$\frac{x}{c}$ transition strip	0.05

Table 2.1: Characteristics of SC(2)-0711 test case airfoil.
† The drag divergence criterion was not met.

The exact implementation in Python can be found in Section A.1.
The results of all the investigated methods can be seen in Figure 2.3 and Table 2.2. A plot of c_d versus M for various values of c_l is shown in Figure 2.3. Here it can be seen that the KLM-method, shown by the striped cyan graph, shows good results. However, the drag creep at $c_l = 0.4$ is not present. Also, the sudden drag rise at $c_l = 0.7$ is not predicted. VGK experiences convergence problems with higher Mach numbers.

VGK is the quickest computation method of the two computational methods considered. It requires 0.51 s per calculation, but the results are not accurate. The predicted values of c_d are too low, and convergence problems are experienced. MSES, requiring 5.88 s per calculation, is accurate in predicting the absolute value of c_d. MSES correctly predicts the initial value of c_d as well as the following drag creep. As expected it is the method with the highest fidelity that provides the most accurate results. However, also for MSES issues remain. MSES has problems predicting the sudden drag rise and for higher Mach numbers the error grows. Furthermore, Figure 2.3 shows that convergence is not always guaranteed. This can be seen at $c_l = 0.4, M = 0.75$ and $c_l = 0.7, M = 0.72$.

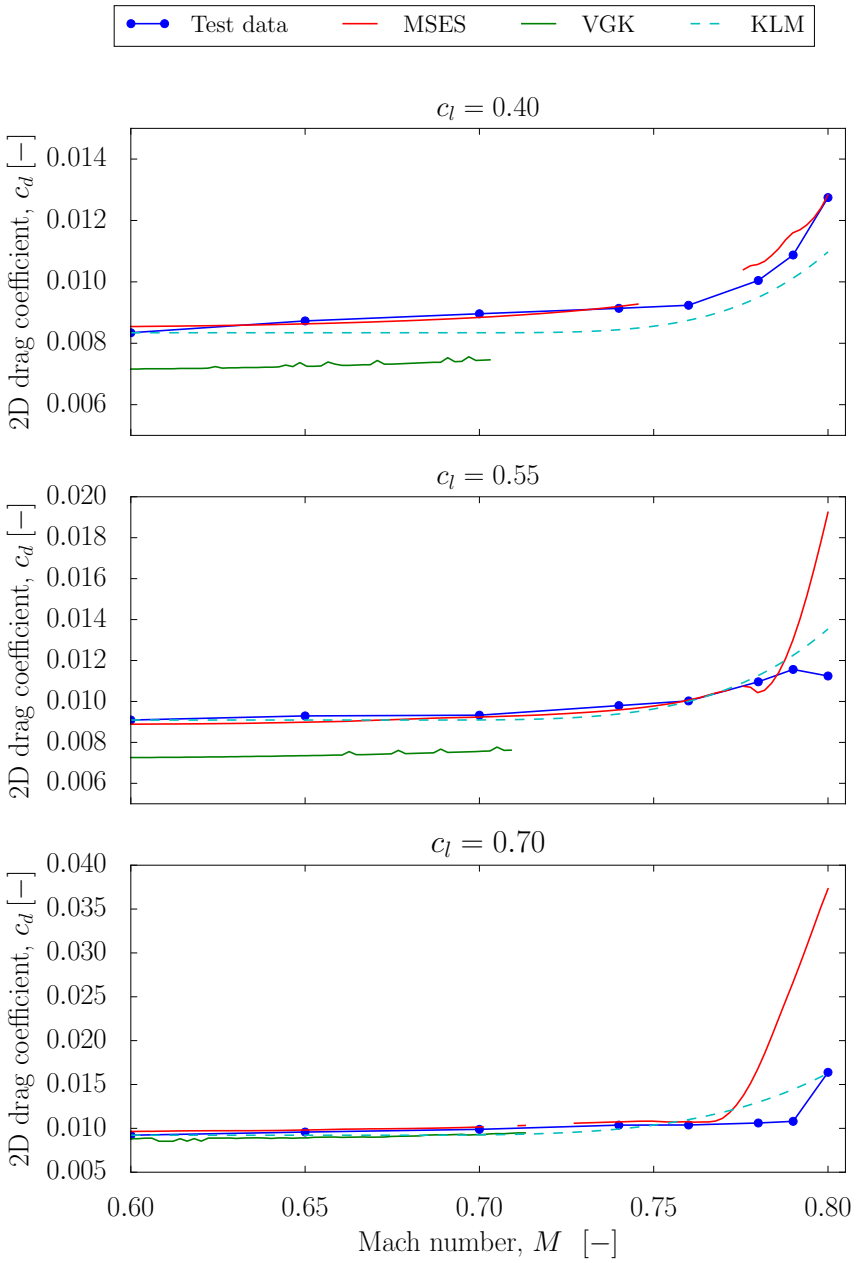
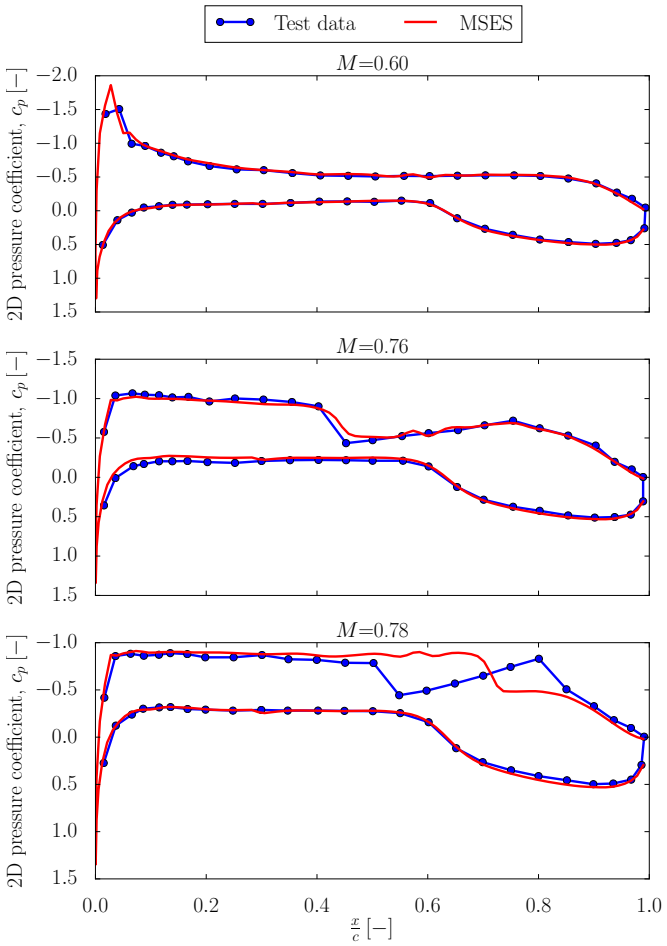


Figure 2.3: Comparison of different wave drag prediction methods.

Another benefit of using a computational method is that these methods can calculate the pressure distribution. In Table 2.2, a comparison of MSES and wind tunnel pressure distributions for $c_l = 0.7$ is shown. It can be seen that the pressure distribution as well as the characteristics are accurate up until $M = 0.76$. Only the value of α shows a large difference. This can be attributed to the interference effects of the wall [35]. After $M = 0.76$, the shocks grow in strength, and MSES is not able to deliver accurate results. Although the pressure distribution for $M = 0.78$ is clearly wrong, the prediction for c_d is accurate. This can only be fortuitous, and stems from two errors canceling out, for example an underestimation of viscous drag counteracting an overestimation of wave drag.

As MSES is able to give a good estimate for the absolute value of c_d, accurately determines the drag-creep and delivers good pressure distributions up until $M = 0.76$, this method is selected as the wave-drag method used. A range of validity up until $M = 0.76$ combined with a typical half chord sweep of 20° [36] means that flight Mach numbers up until 0.81 can be accurately calculated. It should be noted here that this conclusion is based on a single test case. Although MSES has been verified for other test cases as well [30, 32], it is not guaranteed that MSES will deliver accurate results for every case.



Value	α,[°]	$\mathbf{c_d}$, [$\times\mathbf{10^{-4}}$]	$\mathbf{c_m}$, [−]
Test data	2.25	95.0	-0.162
MSES	0.76	96.4	-0.159
Difference	-1.49	1.4	0.003

Value	α, [°]	$\mathbf{c_d}$, [$\times\mathbf{10^{-4}}$]	$\mathbf{c_m}$, [−]
Test data	1.75	105.1	-0.179
MSES	-0.16	107.5	-0.183
Difference	-1.91	2.4	-0.004

Value	α, [°]	$\mathbf{c_d}$, [$\times\mathbf{10^{-4}}$]	$\mathbf{c_m}$, [−]
Test data	1.20	109.8	-0.168
MSES	-0.24	116.7	-0.195
Difference	−1.44	6.9	-0.027

Table 2.2: Comparison between wind tunnel data and data generated by MSES.

3

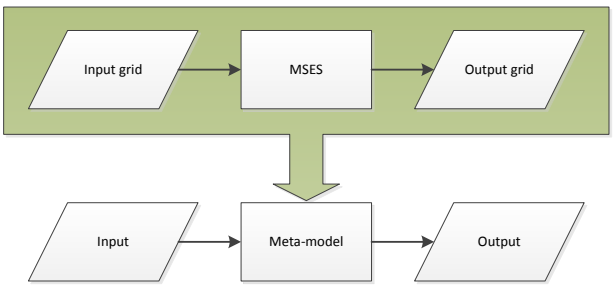
META-MODELING METHOD SELECTION

Now that a suitable aerodynamic model has been found, it is necessary to investigate the possibilities and drawbacks of using meta-models. Section 3.1 discusses the general concept of meta-modeling. Subsequently the application of meta-modeling on wave drag data is discussed in Section 3.2. Finally, the Airbus in-house meta-modeling tools and their application are discussed in Section 3.3.

3.1. META-MODELING CONCEPT

As mentioned before, meta-models are mathematical models of physical models. Normally the physical model is executed for every different set of inputs encountered, as shown in 3.1a.
However, in the case of large numbers of calculations, this can lead to long computation times. A way to reduce the computation time during optimizations is to apply a meta-model. For this application this means that MSES is run for a given range of input parameters (called the input grid) to create an grid of output values, the output grid. Using the input grid and the output grid it is possible to create a mathematical representation of the aerodynamic model, a meta-model, as shown in 3.1b. All the calculations are performed beforehand, and only the fast mathematical model is used inside the optimization loop.

(a) Direct application



(b) Meta-model application

Figure 3.1: Differences between direct calculation and meta-model calculation.

3.2. WAVE DRAG APPLICATION

For the meta-modeling approach it is not possible to have the airfoil coordinates as an input, as this would lead to an exceptional amount on input variables. Therefore, only $\frac{t}{c}$ is chosen as an input parameter. The SC(2)-0711 is scaled based on the thickness, to allow a variation of airfoils.
It should be noted here that the original Supercritical Airfoils collection is not a family of airfoils that is derived by scaling. Therefore, the SC(2)-0711 scaled to 10% thickness, will not look the same as the actual SC(2)-0710. However, as a large variety of thicknesses are required, the scaling method is used to generate Supercritical Airfoils with varying thickness. The input and output for the meta modeling approach are shown in Table 3.1.

Table 3.1: Input and output variables of meta-modeling approach.

Input	Output
Re	α
c_l	c_m
M	c_{d_v}
$\frac{t}{c}$	c_{d_w}
	c_p distribution

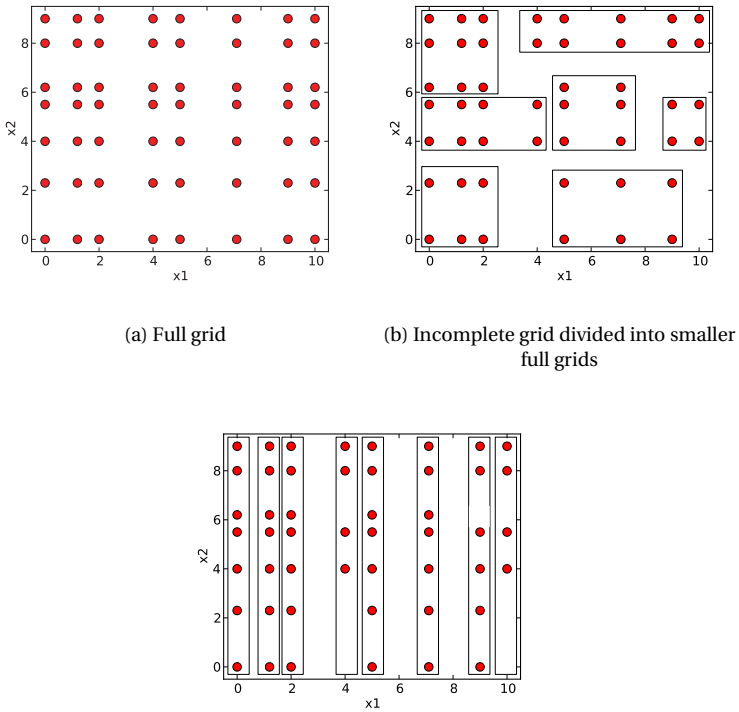
This representation is orders of magnitude quicker than a direct application, but as it is a mathematical model care should be taken to ensure accurate results. One of the limitations of the meta-modeling method is that it can only be used within a pre-determined design space. It is not possible to approximate values for inputs that are outside the input-grid. It is therefore not possible to evaluate radically new designs using an input grid of older airfoils. It is possible to use this method within a pre-determined design space, for example a retwist of an existing wing, or to use it to further optimize an initial design.

3.3. AIRBUS IN-HOUSE META-MODELING: GT-APPROX

At Airbus an in-house algorithmic core called MACROS is present. This core contains a collection of meta-modeling tools. This collection, called GT-Approx [37], is an implementation of many different meta-models. It is developed by DATADVANCE, which is a joint venture between the Airbus-Group and the Institute for Information Transmission Problems of the Russian Academy of Sciences.
In this collection, many meta-modeling methods are available. GT-Approx automatically determines which is the best fitting method based on several features of the input grid. The selection is done based on:

- **Dimension of the input grid**
  This is the number of variables that is given as an input. Some methods can only be used for one dimensional data.
- **Sample size of the input grid**
  This is the number of individual data points that is given to the program. Some methods will lead to exceptionally large computation times for a large amount of data points.
- **Number of non-converged calculations in the input grid**
  Most of the methods require a full factorial grid, as shown in 3.2a. If some data points are not present, the data set is divided into smaller full factorial grids as shown in 3.2b. The fitting method is then applied to these smaller grids. It can be seen that with the omission of 7 data points, many smaller grids are necessary. If a few data points are missing this is possible but if too many calculations did not converge, the number of smaller grids will increase substantially, leading to large computation times. If this is the case, GT-approx will automatically switch to one dimensional spline fitting, as shown in 3.2c. This means that instead of fitting a surface, the data points will be fitted using a collection of one dimensional splines.

The data grid provided by the aerodynamic methods is not a full factorial grid. This is because of non-convergence, calculated values that are considered unrealistic and outliers. The result is a so called incomplete tensor. As such, GT-approx will approximate the data using a collection of one dimensional splines. The method used for fitting these splines is called Splines with Tension.



(a) Full grid

(b) Incomplete grid divided into smaller full grids

(c) Incomplete grid divided into one dimensional sets

Figure 3.2: Overview of examples of different fitting strategies.

This method is a generalization of spline methods. In essence, this is a one dimensional method, that can be applied to each dimension at a time. The heuristic procedure to determine the tension parameters is outlined by Pruess [38]. This method starts with zero tension parameters, calculates derivatives, and adjusts the tension parameters such that local monotonicity and convexity are preserved. This method is well suited for incomplete tensor data. When applied to multi-dimensional data, the result is a collection of Cubic B-Splines [37].

4

META-MODELING METHOD VALIDATION

Now that a meta-model is selected it is important to investigate if the meta-modeling method selected is able to accurately represent the data generated by MSES. Section 4.1 discusses the outlier detection applied. The second section treats the ability of the meta-modeling method to fit the data generated by MSES. In Section 4.3 the predictive capabilities of GT-Approx are treated. Section 4.4 discusses the computation times of GT-Approx. In Section 4.5 the conclusions concerning the abilities of GT-Approx are presented.

4.1. OUTLIER DETECTION

In order to effectively fit the data generated by MSES it is important to apply pre-processing. Any outliers can have a great influence on the fitting accuracy, and should therefore be removed. First, all unphysical data points are removed. This means that any negative values of c_{d_w} or c_{d_v} are deemed unreliable, and therefore removed. Figure 4.1 shows an example of the second phase of outlier selection applied. Here c_{d_w} is plotted against M. The figure shows the 4th order polynomial, and the 3σ-confidence interval. This confidence interval is intentionally made large to ensure that only extreme outliers are removed. It has proven difficult to fit the entire Mach range due to the rapid increase of c_{d_w} at higher values of M. If the entire range of M is fitted, many valid data points near the drag rise are deemed outliers.

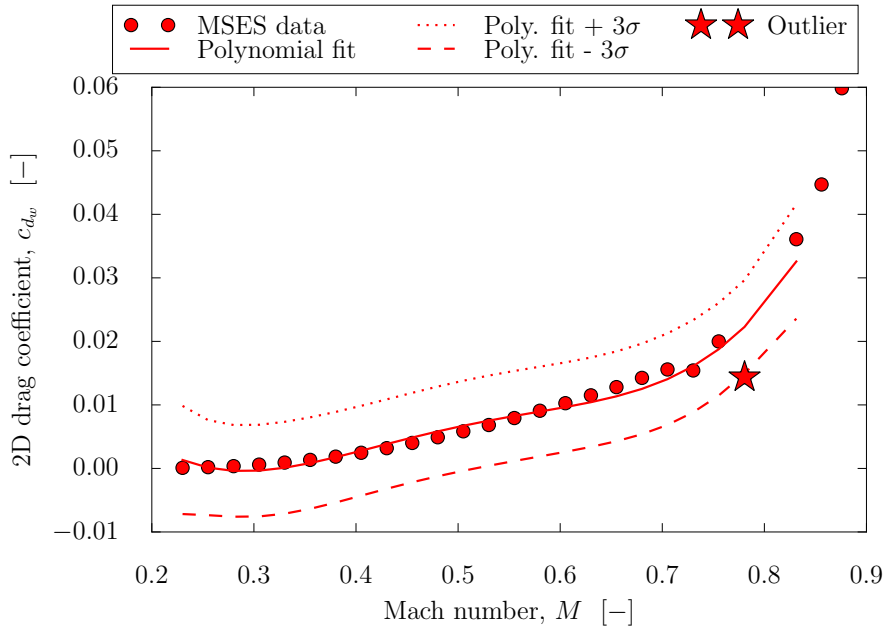


Figure 4.1: Example of outlier detection.

4.2. FITTING CAPABILITIES OF GT-APPROX

In this subsection the fitting capabilities of GT Approx are evaluated. If GT-Approx is not able to generate an accurate fit, it will not be able to produce accurate results for other data points. To this end, a sample batch is generated, with the ranges of parameters as shown in Table 4.1. The SC(2)-0711 airfoil is taken as a basis for scaling. The exact parameters and characteristics of MSES can be found in Equation 2.2.
These data-points are calculated using MSES and subsequently a meta-model of this data is generated by GT-Approx. Then the absolute difference between the source data and GT-Approx-data, denoted by Δ_{fit} is evaluated. All the MSES data is accurately represented by GT-Approx as is shown by Table 4.2 where it can be seen that the relative error does not exceed 0.11%.

Table 4.1: Overview of parameter ranges used to generate the meta-model

Variable	**Minimum, [-]**	**Maximum, [-]**	**Stepsize, [-]**
$\frac{t}{c}$	0.10	0.14	0.01
Re	20.0×10^6	60.0×10^6	10.0×10^6
c_l	0.30	0.80	0.05
M	0.40	0.80	0.05

Table 4.2: Average differences between MSES and GT-Approx fit

Variable	α	$\mathbf{c_{d_v}}$	$\mathbf{c_{d_w}}$	$\mathbf{c_m}$
Average value	1.20 [deg]	7.80×10^{-3} [-]	8.00×10^{-3} [-]	0.18 [-]
Δ_{fit}	1.30×10^{-3} [deg]	1.00×10^{-6} [-]	3.70×10^{-7} [-]	1.30×10^{-5} [-]
Error	0.11 [%]	1.30×10^{-2} [%]	4.60×10^{-2} [%]	7.40×10^{-3} [%]

4.3. PREDICTIVE CAPABILITIES OF GT-APPROX

At this point, the fitting capabilities of GT-Approx have been evaluated. It is necessary to investigate if GT-Approx is also able to predict data. In this section it is investigated how accurate GT-Approx can determine results for data-points that have not been previously calculated with MSES.

4.3.1. METHOD

The following steps are employed to asses the predictive qualities:

1) From the source data as used in Section 4.2 data is removed for a particular value of a single variable (for example all data points containing $Re = 20.0 \times 10^6$).

2) This set is used to create the meta-model.

3) This meta-model is used to calculate the results for the removed data. In case of the example this would mean, the meta-model is used estimate values for $Re = 20.0 \times 10^6$.

4) These values are compared to the results calculated by MSES, and the average difference is calculated. Thus for the example of $Re = 20.0 \times 10^6$, the resulting value is the difference between the GT-Approx prediction and MSES calculation for $Re = 20.0 \times 10^6$, averaged over all the values of c_l, $\frac{t}{c}$ and M.

This procedure is repeated for the whole range of Re, and subsequently plotted. This is done for all the input variables, to see how well GT-Approx is capable of predicting missing values.

4.3.2. PREDICTION OF DRAG AS A FUNCTION OF *Re* VALUES

The prediction error development for values of Re is shown in Figure 4.2. The graphs denote the average value calculated by MSES, averaged over all the values of $\frac{t}{c}$, c_l and M. The error bars show the difference between the MSES calculations and the GT-Approx calculations. From this figure, several conclusions can be drawn:

1) GT-Approx is not good at extrapolating. It can be seen that the errors for the two outer values are significantly higher than the error for the inner values. If one of the outer two values (for example $Re = 20 \times 10^6$) is removed from the source data, and the remaining points are used by GT-Approx to predict results for this value, GT-Approx is not interpolating, but extrapolating. As a consequence the error increases significantly. This also occurs for the other variables. It is concluded that GT-Approx is good at interpolating, but not good at extrapolating.

2) Although the accuracy has decreased compared to the fitting case treated in Section 4.2, in general, the fit for interpolating values is still very good. For the inner values the error bars are hardly distinguishable. The average errors for the inner values are 1.85 drag counts for c_{d_v} and 0.38 drag counts for c_{d_w}.

Further more, the general trends in the graph make sense. The wave drag does not change much with increasing Reynolds number, whereas the viscous drag decreases with increasing Reynolds numbers. The largest part of viscous drag is caused by surface friction because of shear stress in the boundary layer [39]. For a turbulent boundary layer, the shear friction is given by White [40]:

$$\tau = \mu \frac{\partial \overline{u}}{\partial y} - \rho \overline{u'v'} \tag{4.1}$$

It is possible to rewrite Equation 4.1 in terms of the Reynolds number:

$$\tau = \frac{\rho V c}{Re} \frac{\partial \overline{u}}{\partial y} - \rho \overline{u'v'} \tag{4.2}$$

Equation 4.2 shows that for increasing Reynolds numbers the shear friction τ will decrease, leading to a lower value of c_{d_v}. This fact is also present in Figure 4.2.

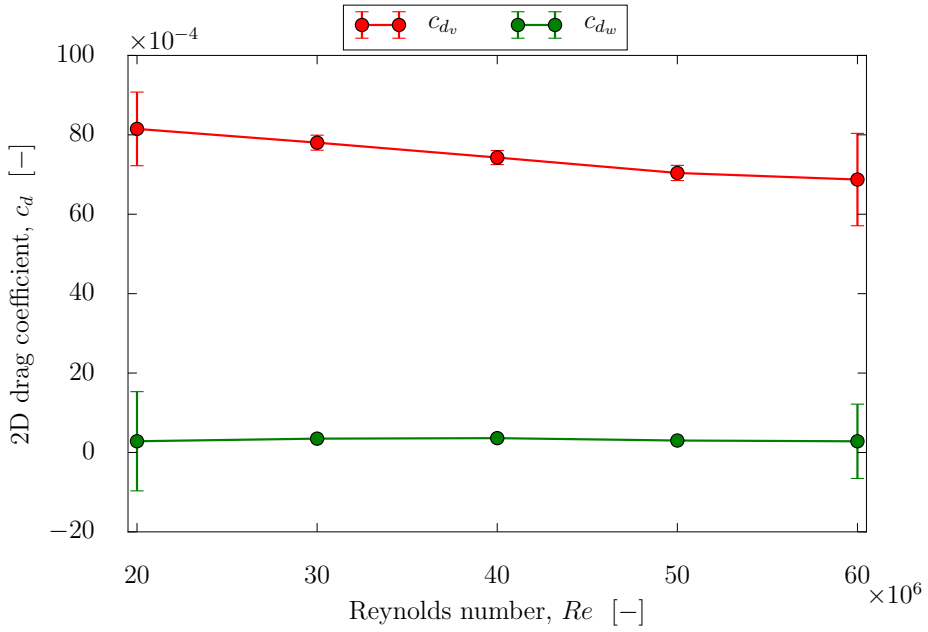


Figure 4.2: Development of the two drag components and the error versus *Re*.

4.3.3. PREDICTION OF DRAG AS A FUNCTION OF $\frac{t}{c}$ VALUES

A similar procedure is performed for the prediction of $\frac{t}{c}$ values. The result of this can be seen in Figure 4.3. This figure shows similar result as before, large errors for extrapolating, and small errors for interpolating. Furthermore it can be seen that the value of c_{d_v} increases with increasing $\frac{t}{c}$. However, contrary to the KLM-method, Figure 4.3 shows that c_{d_w} does not increase with increasing values of $\frac{t}{c}$. This can be attributed to the number of converged calculations. For higher values of $\frac{t}{c}$, convergence will be more difficult. This means that even for moderate values of M, calculations might not converge. Only converged calculations are used for the calculation of the average. If only calculations for low values of M converge, this will lead to a lower value average value of c_{d_w}.

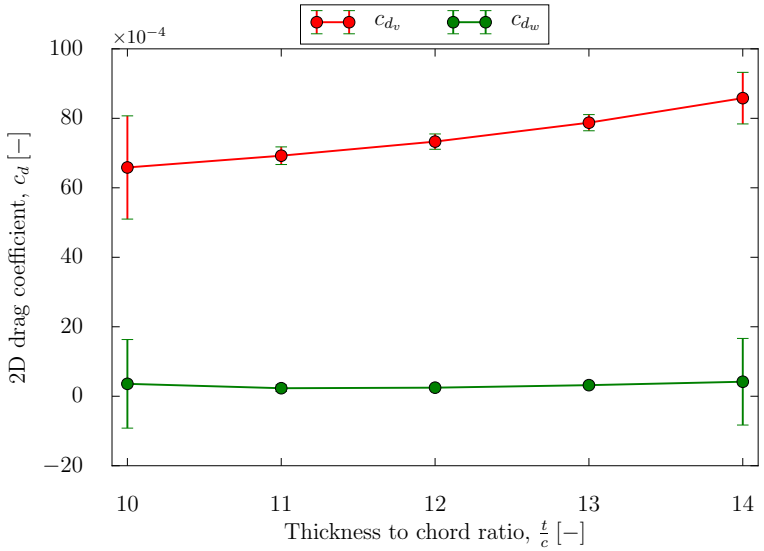


Figure 4.3: Development of the two drag components and the error versus $\frac{t}{c}$.

4.3.4. PREDICTION OF DRAG AS A FUNCTION OF M VALUES

The predicting capabilities of GT-Approx with respect to values of M are investigated. The results can be seen in Figure 4.4. Again, the lines show the values of c_{d_w} calculated by MSES, and averaged over all the values of $\frac{t}{c}$, c_l and Re. The error bars show the difference between the MSES calculations and the GT-Approx calculations. Here it can be seen that the approximation also shows the increase at the edges due to extrapolation, as well as increases towards higher Mach numbers due to more complex data to be modeled. It can be seen that the error increases progressively, and at higher Mach numbers is as large as the value of c_{d_w} itself.

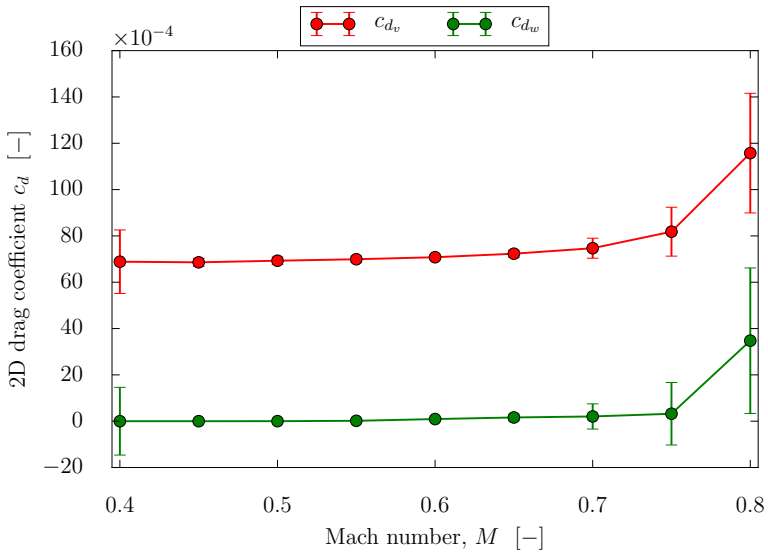


Figure 4.4: Development of the two drag components and the error versus M.

4.3.5. PREDICTION OF DRAG AS A FUNCTION c_l VALUES

Also, the prediction for c_l values is investigated. The result can be found in Figure 4.5. Again, it can be seen that for the value of c_{d_v} the outer predictions are significantly worse than the inner values. However, for c_{d_w} the extrapolation at the lower values of c_l no large extrapolation error occurs. A zoomed plot shown in Figure 4.6 shows that the error for $c_l = 0.3$ is significantly larger than the neighboring interpolated value.
It can also be seen that on average, the prediction increases with increasing values of c_l. This can be explained by the fact that for higher values of c_l, the changes for increasing Mach numbers will become more fierce, and thus more difficult to predict accurately. Again, the general trends show that the value of c_{d_w} and c_{d_v} do not increase significantly with increasing of c_l. This can be explained that the airfoil is optimized for $c_l = 0.7$. Only for high values of c_l the value of c_{d_w} increases significantly.

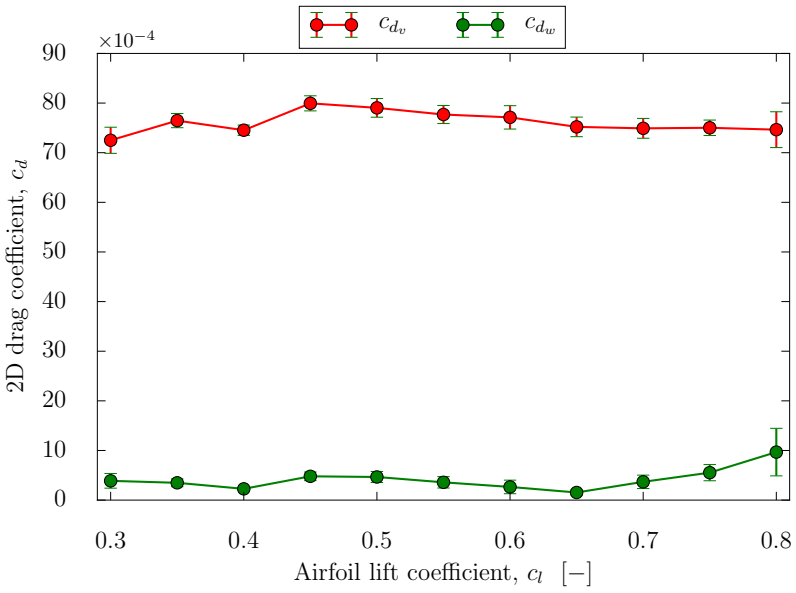


Figure 4.5: Development of the two drag components and the error versus c_l.

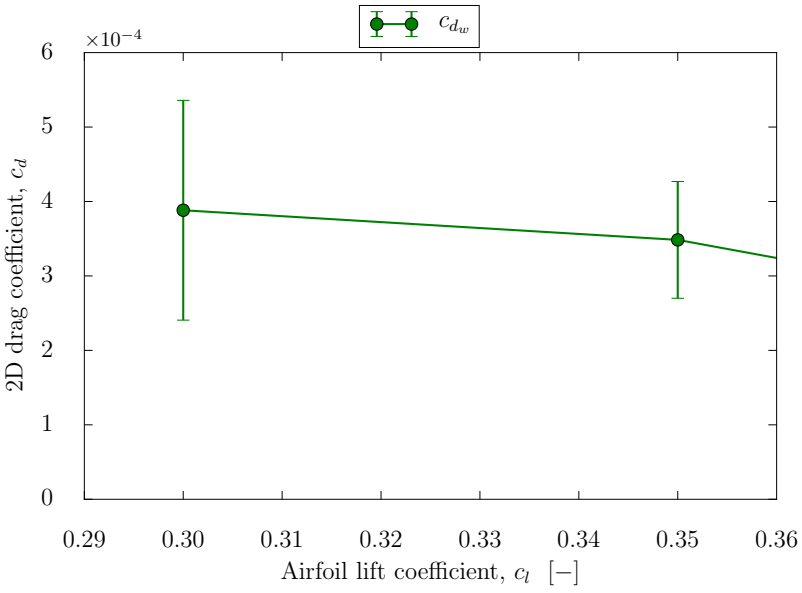


Figure 4.6: Zoom of development of c_{d_w} and the error versus c_l.

4.4. GT-APPROX COMPUTATION TIMES

Not only the fitting capabilities of GT-Approx are important, also the required computational time is important. Therefore it is researched how long it takes to load different datasets, and how fast values can be reproduced once the models is loaded. To test this a varying number of datapoints is used to build a GT-Approx model. This will show the relationship between calculation time and number of data points. The result is shown in Figure 4.7. Here it can be seen that the model building time increases approximately linear with the number of data-points. However, as this only needs to be done once, this is not a significant disadvantage.
The computation time also increases with the number of data points, but this increase is very moderate. At 23301 data points, the computation time is only 2.52×10^{-5} s longer than for a model based on 277 data points.

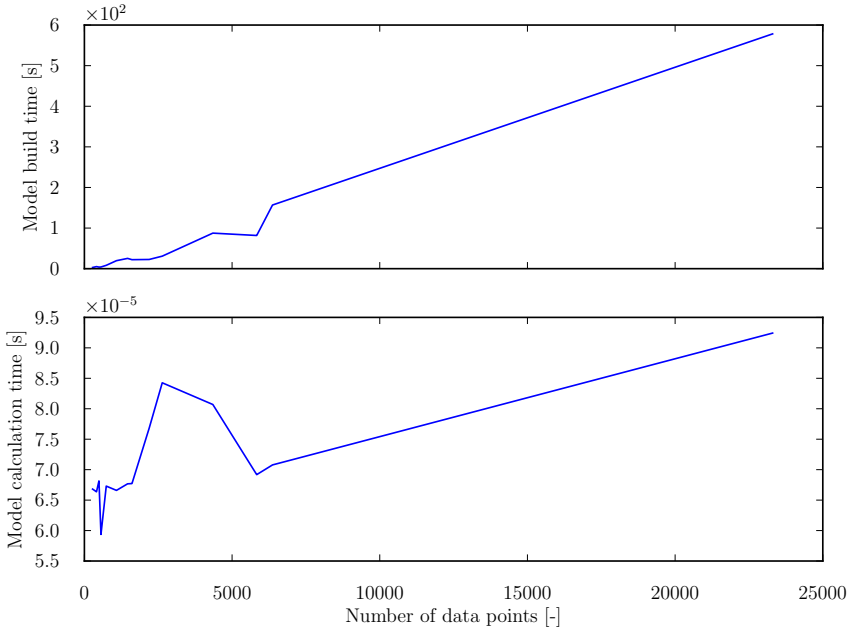


Figure 4.7: Model building time and computation time versus number of data points.

4.5. CONCLUSION CONCERNING GT-APPROX ABILITIES

Based on the verification performed in this chapter, it can be stated that the general predictive capabilities of GT-Approx are generally good, but some issues remain. GT-approx can interpolate the drag values with an accuracy of approximately 2 drag counts. Only for complicated cases, such as high Mach numbers the predictions become less accurate. In almost all cases extrapolation leads to increasing errors, and the accuracy is no longer guaranteed.

5

META MODELING RESOLUTION SENSITIVITIES

As can be seen from Chapter 4, the accuracy of GT-Approx varies significantly with the omission of certain values. This is because the performance of GT-Approx is highly dependent on input grid. Efforts are undertaken to improve the estimation quality, specifically the prediction of c_{d_w} for values of M. This is the only variable that shows large errors for interpolation values.
In this chapter the influence of the various resolutions is investigated to examine what resolution is required to achieve the necessary accuracy. First, the method is explained in Section 5.1. Subsequently increases in resolution are investigated in Section 5.2. Non-uniform resolution increases are discussed in Section 5.3. Finally in Section 5.4 the effect of combined resolution increases is assessed and the final resolutions are presented.

5.1. METHOD

The resolutions are evaluated using the following steps:

1) The source data, generated before as discussed in Section 4.2, is adjusted to contain higher or lower resolution data.

2) For all the variables, the procedure as explained in Section 4.2 is performed.

3) The average relative errors and absolute differences in drag counts for all the estimations are calculated.

4) These values are plotted versus the resolution to discern any trends and locate the origin of the errors.

Using this procedure it is possible to see if adding additional data will change the quality of the predictions and for which variables.

5.2. RESOLUTION INCREASES

Increasing the resolution of the source datawill lead to smaller interpolation intervals, and thus smaller interpolation errors. A smaller resolution might also mitigate the effects caused by missing data points. Because of the nature of the calculations done by MSES it might be the case that the calculation for $M = 0.7, c_l = 0.65$ fails, whereas a calculation for $M = 0.7, c_l = 0.66$ might converge. Therefore, more data for difficult calculations can be obtained by performing more MSES calculations. This allows GT-Approx to better understand the trends, and thus provide better estimations. However, as this requires a large amount of extra data points, it will lead to larger MSES-computation times as well as longer times required to build the meta-models. The following resolution increases are considered:

- Mach resolution
- c_l resolution
- Thickness resolution
- Reynolds number resolution

5.2.1. MACH RESOLUTION

The influence of resolution increases of all the variables have been examined. The influence of ΔM is investigated in order to improve the Mach accuracy. Specifically the higher Mach values showed significant errors. To investigate the influence of the data spacing a range of spacings (denoted by ΔM) are evaluated, with the baseline resolution indicated in bold:

- 0.10
- **0.05**
- 0.025
- 0.0125
- 0.00625

The results are shown in Figure 5.1 and Table 5.1. Substantial gains can be achieved by decreasing the step size ΔM. For example, halving the value of ΔM of the baseline sample reduces the average prediction error of c_{d_v} by 32%, and the average prediction error concerning c_{d_w} by 65%. Upon decreasing ΔM to 0.0125 the improvement with respect to the baseline increases to 60% and 87% for c_{d_v} and c_{d_w} respectively. Further decreasing the value of ΔM to 0.00625 does not result in significant gains.

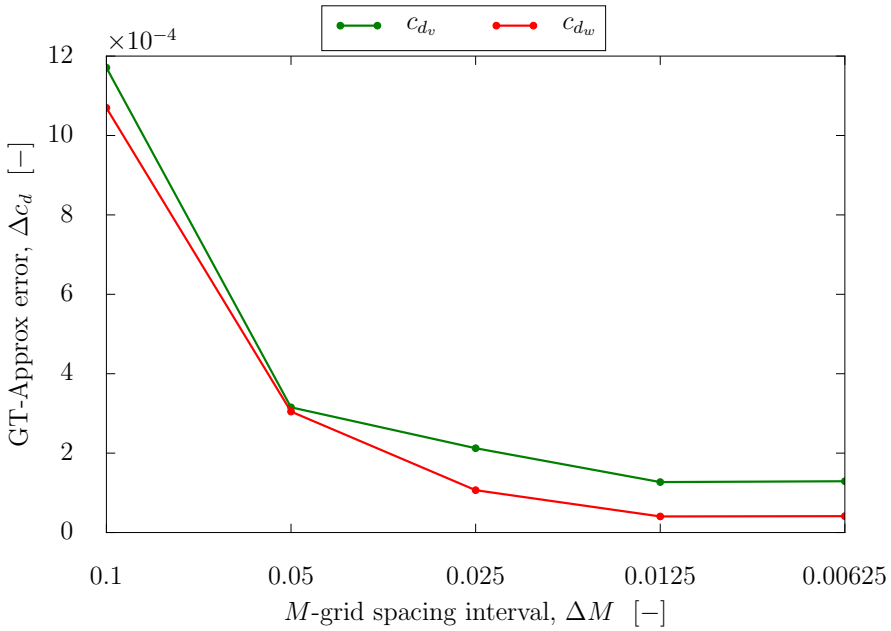


Figure 5.1: GT-Approx prediction error versus M-grid spacing.

Table 5.1: Overview of Δc_{d_v} and Δc_{d_w} versus ΔM

ΔM, [-]	**Δc_{d_v}, [10^{-4}]**	**Improvement, [%]**	**Δc_{d_w}, [10^{-4}]**	**Improvement, [%]**
0.10	11.709	-271.29	10.697	-251.13
0.05	**3.153**	**0.00**	**3.046**	**0.00**
0.025	2.124	32.65	1.066	64.99
0.0125	1.271	59.70	0.405	86.69
0.00625	1.291	59.04	0.412	86.46

Besides evaluating the average differences, it is also important to investigate the influence of the resolution on the ability of GT-Approx to recreate the behavior present in the MSES data. In order to examine this, the behavior of Δc_{d_w} and Δc_{d_v} versus M is plotted for various resolutions, as can be seen from Figure 5.2. Here it can be seen that increasing the M-resolution has allowed GT-Approx to improve the approximation for higher Mach numbers. It can be seen that for the baseline step size of $\Delta M = 0.05$, the difference between MSES and GT-Approx is 4 drag counts for $M = 0.70$, and larger than 10 drag counts for $M = 0.75$.
Halving ΔM has greatly reduced the errors, and for $\Delta M = 0.0125$, both drag parts show an error of less than two drag counts for both $M = 0.70$ and $M = 0.75$. Also, the conclusion that further decreasing the step size does not yield any additional gains is confirmed, as can be seen when the plots for $\Delta M = 0.0125$ and $\Delta M = 0.00625$ are compared. $\Delta M = 0.0125$ is chosen as the best compromise between accuracy and number of computations.

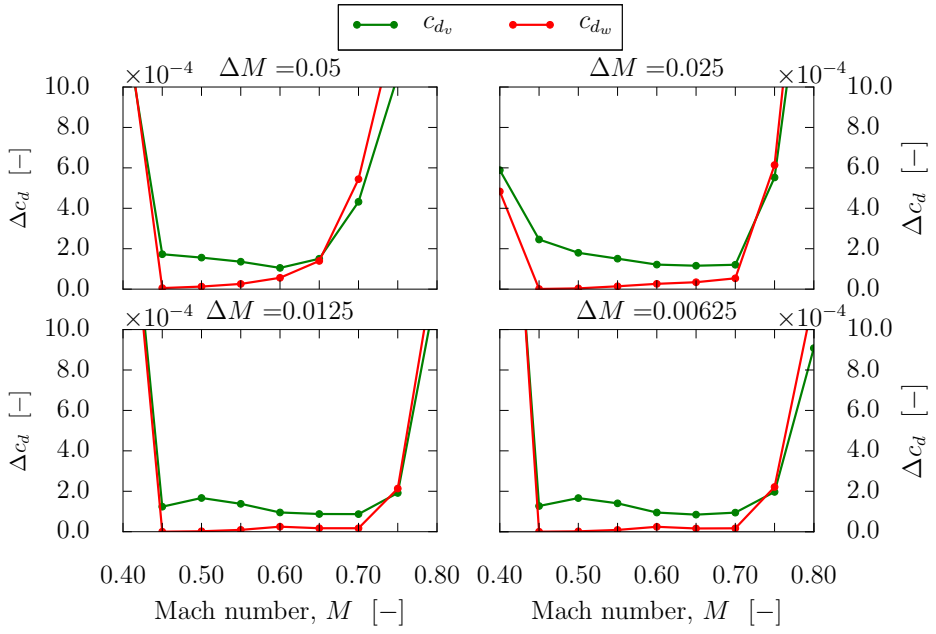


Figure 5.2: GT-Approx prediction error versus M for different resolutions.

5.2.2. THICKNESS RESOLUTION

Although the thickness resolution is already quite large, reasonable results are obtained for the prediction concerning thickness values. Therefore, it might be possible to further reduce the resolution to save calculation time. The following thickness ratio intervals are assessed:

- 0.02
- **0.01**
- 0.005
- 0.0025
- 0.00125

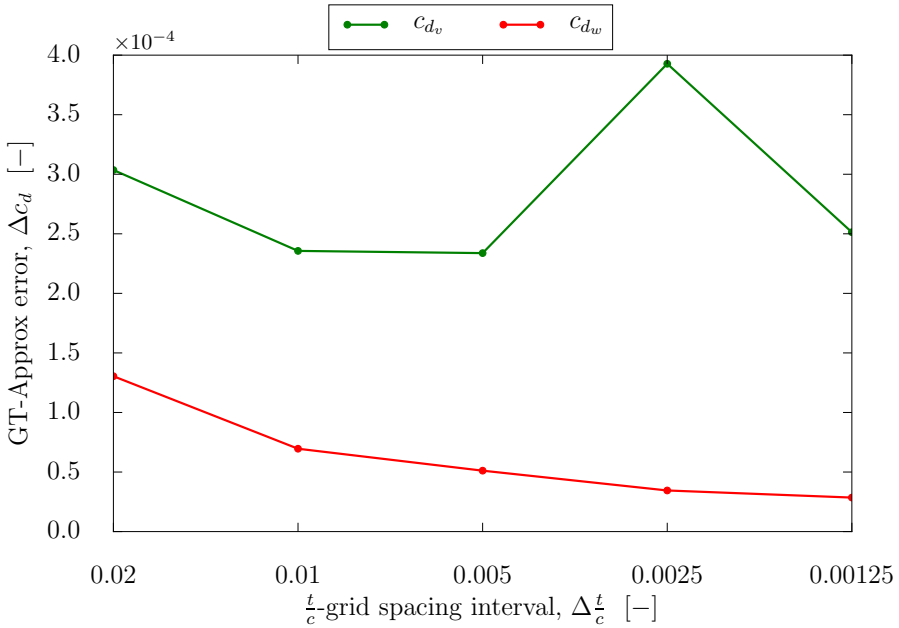


Figure 5.3: GT-Approx prediction error versus $\frac{t}{c}$-grid spacing.

Table 5.2: Overview of Δc_{d_v} and Δc_{d_w} versus $\Delta\frac{t}{c}$

$\Delta\frac{t}{c}$, [-]	Δc_{d_v}, $[\times 10^{-4}]$	Improvement [%]	Δc_{d_w}, $[\times 10^{-4}]$	Improvement, [%]
0.02	3.035	-28.81	1.304	-87.51
0.01	**2.356**	**0.00**	**0.695**	**0.00**
0.005	2.337	0.786	0.511	26.47
0.00025	3.927	-66.66	0.345	50.37
0.000125	-6.65	18.63	0.285	58.94

From Figure 5.3, it can be seen that reducing the resolution is not feasible, as the error grows significantly. Decreasing the resolution causes both the drag errors to increase, although the improvement for c_{d_v} is small. After this, the error for c_{d_w} decreases. However, the value of c_{d_v} is not at a global minimum. As can be seen from Figure 5.4 this is because there is a very large prediction error for the c_{d_v} at $\frac{t}{c} = 0.11$. The resolution of $\Delta\frac{t}{c} = 0.005$ is chosen as the final resolution.

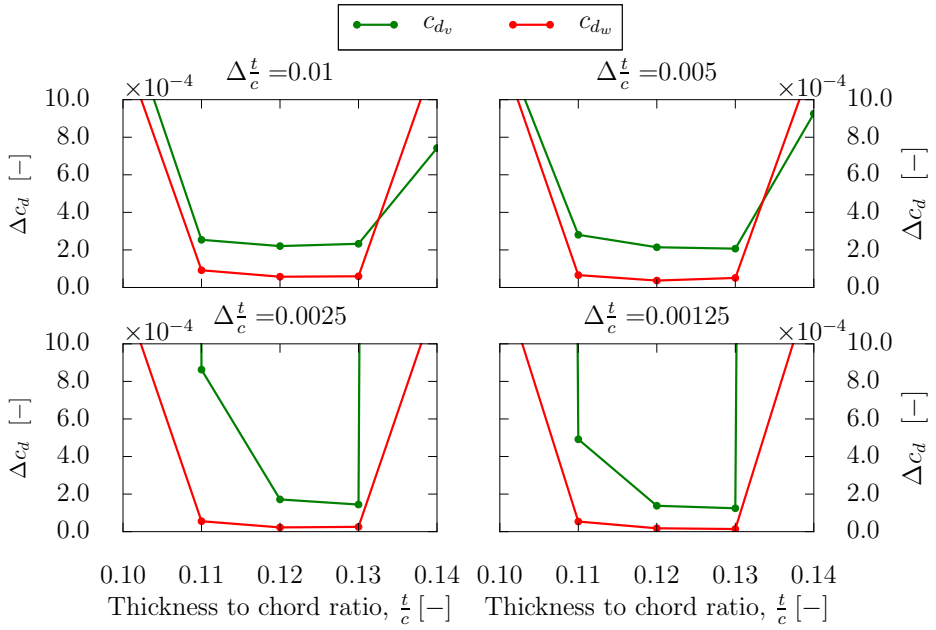


Figure 5.4: GT-Approx prediction error versus $\frac{t}{c}$ for different resolutions.

5.2.3. c_l RESOLUTION

As is shown in Subsection 4.3.5 the baseline resolution shows some difficulties in approximating the values calculated by MSES. Therefore, the influence of the c_l-resolution is investigated to see if it is possible to improve the approximation. The following resolutions are evaluated:

- 0.1
- **0.05**
- 0.025
- 0.0125
- 0.00625

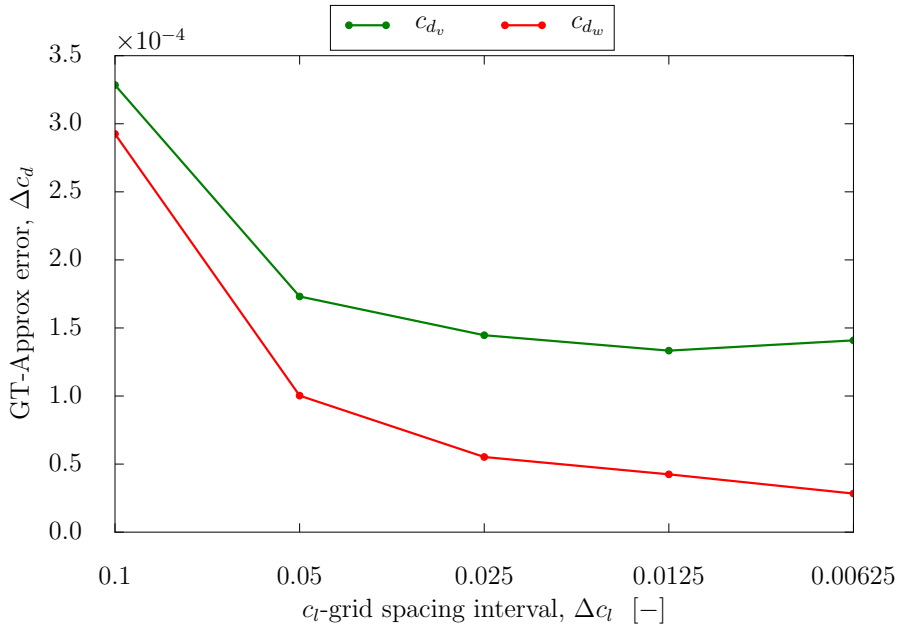


Figure 5.5: GT-Approx prediction error versus c_l-grid spacing.

Table 5.3: Overview of Δc_{d_v} and Δc_{d_w} versus Δc_l

Δc_l, [-]	Δc_{d_v}, [10^{-4}]	Improvement [%]	Δc_{d_w}, [10^{-4}]	Improvement, [%]
0.1	3.284	-89.70	2.925	-191.69
0.05	**1.731**	**0.00**	**1.002**	**0.00**
0.025	1.446	16.44	0.552	44.94
0.0125	1.333	22.98	0.424	57.64
0.00625	1.408	18.63	0.283	71.73

Figure 5.5 and Table 5.3 show that up until $\Delta c_l = 0.0125$ the accuracy improves with increasing resolution. However, when the stepsize is increased to 0.00625, the increase in accuracy is not significant. The error in c_{d_w} still decreases marginally, but the value of c_{d_v} starts to increase. Because the error distribution in the baseline prediction was rather erratic, as shown by Figure 4.5, the improvements in c_l are further investigated. This is done to see if any improvements are achieved for higher c_l or, that the general estimation quality has increased.

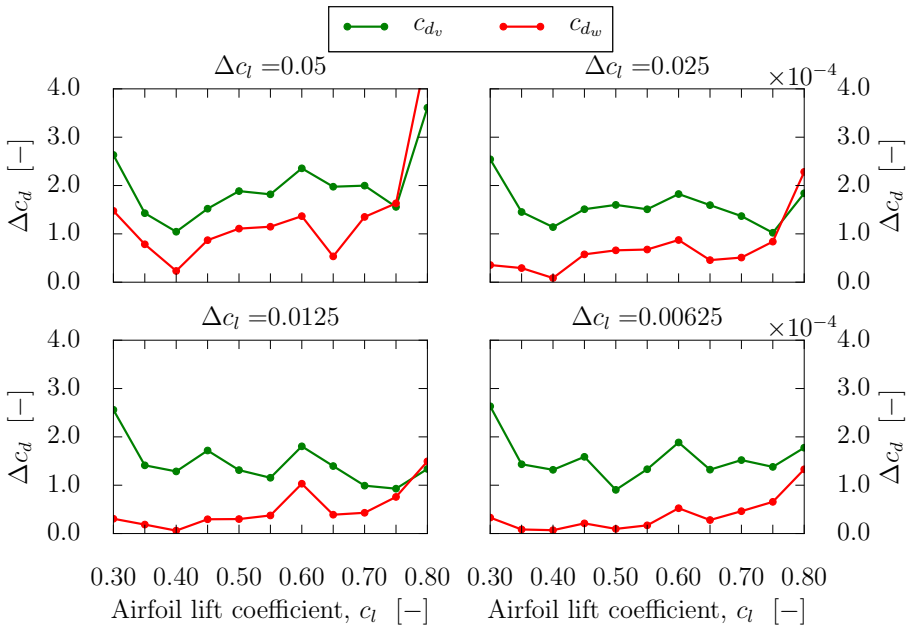


Figure 5.6: GT-Approx prediction error versus c_l for different resolutions.

Figure 5.6 shows that the erratic behavior persists, also for higher resolutions. Furthermore it can be seen that the estimation accuracy of c_{d_w} increases significantly, whereas the estimation improvement of c_{d_v} is less strong. This confirms the trends shown in Figure 5.5. Based on the results presented in Figure 5.5 and Figure 5.6 a value of $\Delta c_l = 0.0125$ is chosen as optimal.

5.2.4. REYNOLDS NUMBER RESOLUTION

For the Reynolds number, the predictions were already accurate. However, it might be possible to improve the accuracy further by increasing the resolution. For every step, the value of ΔRe is halved. The original value is indicated in bold. The resolutions investigated are:

- 20.0×10^6
- $\mathbf{10.0 \times 10^6}$
- 5.00×10^6
- 2.50×10^6
- 1.25×10^6

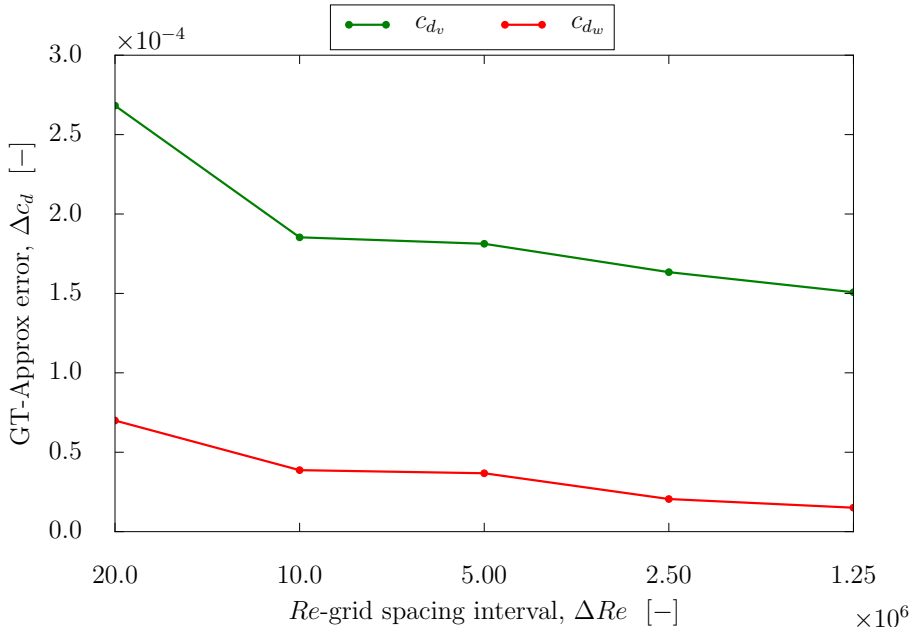


Figure 5.7: GT-Approx prediction error versus Re-grid spacing.

Table 5.4: Overview of Δc_{d_v} and Δc_{d_w} versus ΔRe

ΔRe, [$\times 10^6$]	**Δc_{d_v}, [$\times 10^{-4}$]**	**Improvement [%]**	**Δc_{d_w}, [$\times 10^{-4}$]**	**Improvement, [%]**
20.0	2.682	-44.73	0.699	-80.70
10.0	**1.853**	**0.00**	**0.387**	**0.00**
5.00	1.812	2.19	0.367	5.03
2.50	1.633	11.83	0.205	46.95
1.25	1.507	18.64	0.150	61.05

From Figure 5.7 and Table 5.4 it can be seen that no large gains can be achieved by increasing the Reynolds number. This was expected from the results of Subsection 4.3.2, as it showed a good approximation for all values but the outer two points. It can also be seen that doubling the resolution not give any significant gains, but further increasing the resolution does give moderate improvements. Table 5.4 shows that increasing the stepsize to $\Delta Re = 1.25 \times 10^6$ does not improve the accuracy by much. Figure 5.8 shows that the behavior of the error does not change much. Therefore it is decided to maintain a value of $\Delta Re = 2.50 \times 10^6$.

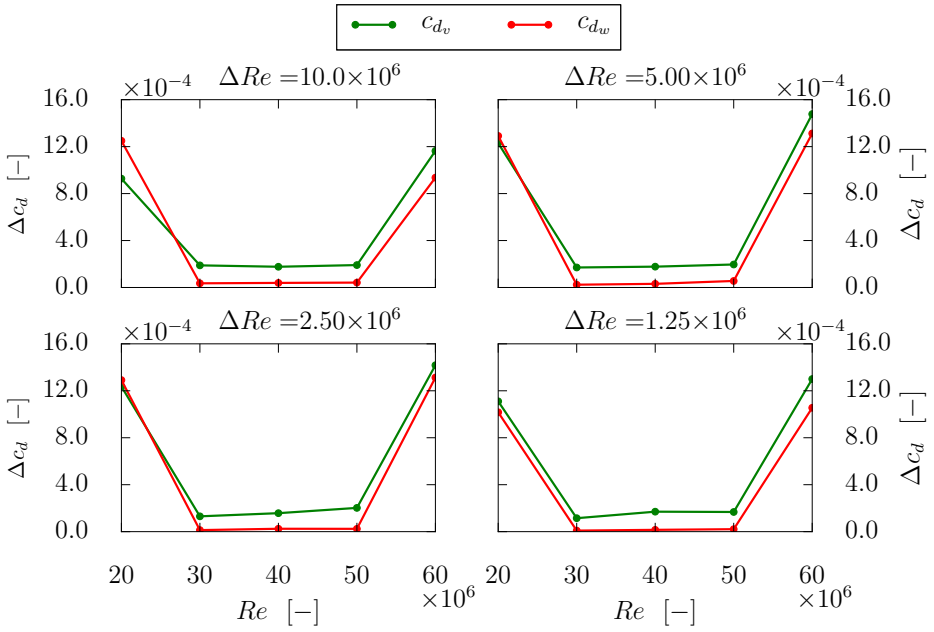


Figure 5.8: GT-Approx prediction error versus Re for different resolutions.

5.2.5. FINAL RESOLUTIONS

From Subsection 5.2.4 through Subsection 5.2.2 the following resolutions have been obtained:

Variable	Resolution
Re	2.50×10^6
C_l	0.0125
M	0.0125
$\frac{t}{c}$	0.005

Table 5.5: Overview of resolutions obtained to assured adequate prediction accuracy.

However, it should be noted that these resolution do not correspond directly to the resolutions necessary to achieve the accuracies listed. This is because of the method used to asses the differences between calculations and approximations. For example, for a ΔM of 0.01, the following would occur in the difference assessment: a value was removed (say $M = 0.70$, shown in gray in the image), and based on the other values, an estimate was created for the value at $M = 0.70$. This means that the interpolation distance is equal to $(\Delta M)_{int} = 0.01$, see Figure 5.9 upper half. Thus the listed accuracies are for values $(\Delta M)_{int}$.

Variable	Resolution
Re	2.50×10^6
c_l	0.0125
M	0.0125
$\frac{t}{c}$	0.005

Table 5.6: Overview of final resolutions.

However, in the actual application of the model, all the calculated points will be present, see Figure 5.9 lower half. This means that for a value of $\Delta M = 0.01$ in the application the interpolation distance is equal to $\frac{\Delta M}{2} = 0.005$. Thus, in the application, a stepsize of $\Delta M = 0.01$ would achieve the error associated $\Delta M = 0.005$ in the error estimation performed in the previous subsections.

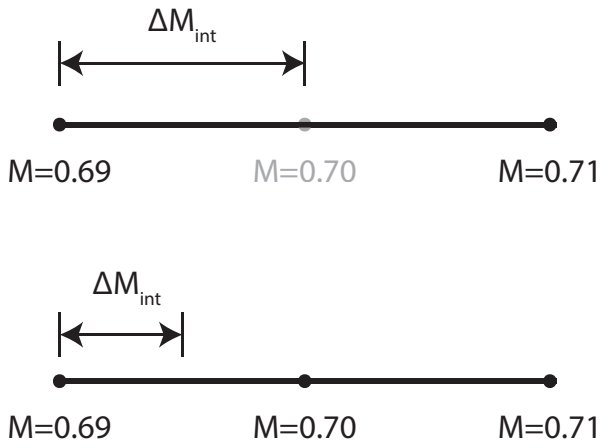


Figure 5.9: Interpolation distance for the error estimation (above) and the application (below).

Thus the accuracy that belongs to a certain ΔM mentioned in Subsection 5.2.1 corresponds to the actual resolution twice ΔM. This reasoning also holds for the other variables, making the final recommended resolutions:

Table 5.7: Overview of final recommended resolutions and their accuracy.

Variable	**Resolution**	$\Delta \mathbf{c_{d_v}}$ $[\times \mathbf{10^{-4}}]$	$\Delta \mathbf{c_{d_w}}$ $[\times \mathbf{10^{-4}}]$
Re	5×10^6	1.63	0.20
c_l	0.025	1.33	0.42
M	0.025	1.27	0.40
$\frac{t}{c}$	0.01	2.33	0.51

5.3. TARGETED RESOLUTION INCREASES

Besides linearly increasing the resolutions it is also possible to increase the number of data points for a region where more accuracy is required. For the low Mach numbers the development is rather smooth, whereas for higher Mach numbers, sudden and fierce increases might be present. To save computation time, it is proposed to increase the resolution at locations where more accuracy is needed: at high Mach numbers. Further more it was found in Subsection 5.2.1 that the main improvements in the approximation where achieved for higher Mach numbers. For the lower Mach numbers, increasing the resolution yielded little to no improvement. Therefore it seems logical to omit the lower Mach resolution-increases. Therefore, it is proposed to only add increased resolution data for values larger than $M = 0.6$. Again the absolute errors versus resolution are compared, this can be seen in Figure 5.10.

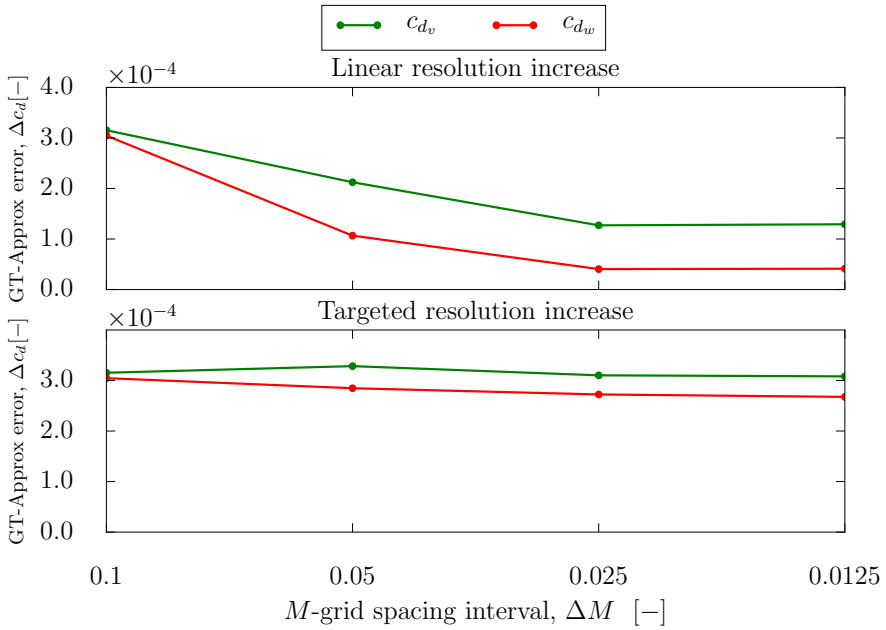


Figure 5.10: Estimation error for the linear resolution increase (above) and the targeted resolution increase (below) versus M-resolution.

It can be seen that the targeted resolution increase does not have the expected behavior. Increasing the resolution hardly increases the accuracy. This can be explained by the method used by GT Approx. In case of an incomplete tensor approach, GT Approx tries to reconstruct the complete tensor, but with wide margins. By applying a targeted resolution increase this behavior is invoked. GT Approx is expecting the same resolution for both higher and lower Mach numbers. Since these values are not present, GT-Approx tries to reconstruct these values. Following this strategy it can be seen that the quality hardly increases for targeted resolution increases. Therefore this option is dismissed as a feasible option for accuracy improvement.

5.4. FINAL RESOLUTIONS

The procedure as explained in Subsection 5.2.1 is also applied to the other variables. The resulting step sizes are shown in Table 5.8.

Variable	**Resolution**	$\Delta \mathbf{c_{d_v}} [\times \mathbf{10^{-4}}]$	$\Delta \mathbf{c_{d_w}} [\times \mathbf{10^{-4}}]$
Re	5.00×10^6	1.63	0.20
c_l	0.025	1.33	0.42
M	0.025	1.27	0.40
$\frac{t}{c}$	0.01	2.33	0.51

Table 5.8: Overview of recommended resolutions and their accuracy.

It is important to investigate how all the individual resolutions and their accuracies combine to a total accuracy. It is assumed that the errors are independent, and thus according to Dekking et al [41], the error for all the different variables combined can be estimated using Equation 5.1.

$$\Delta_{c_{d_{combined}}} = \sqrt{\left(\Delta_{c_{d_{\Delta Re}}}\right)^2 + \left(\Delta_{c_{d_{\Delta c_l}}}\right)^2 + \left(\Delta_{c_{d_{\Delta M}}}\right)^2 + \left(\Delta_{c_{d_{\Delta \frac{t}{c}}}}\right)^2} \tag{5.1}$$

To validate the combined error, the approximation accuracy is calculated for a range of test-locations. This will determine what the accuracy is when interpolation for multiple dimensions is necessary. To this end, a so called offset-dataset was generated. For example, for the Mach-numbers, if the first three calculated points are $M = 0.4, 0.425$ and 0.475, the offset values are taken as $M = 0.4125, 0.4375$. This maximizes the interpolation distance. For all these points, the GT Approx values are compared with data calculated by MSES. This leads to the average errors, which are compared to the expected errors in Table 5.9.

Table 5.9: Overview of predicted and actual accuracy.

Variable	$\Delta \mathbf{c_{d_v}}, [\times \mathbf{10^{-4}}]$	$\Delta \mathbf{c_{d_w}}, [\times \mathbf{10^{-4}}]$
Predicted	3.35	0.81
Offset test case	2.01	0.82

As can be seen from Table 5.9, the error for the wave drag is as expected, where the viscous drag is lower than the expected value. This is because the different interpolation accuracies are not completely independent. This means that the estimation for a certain value of M is not only a function of ΔM, but also a function of ΔRe, $\Delta \frac{t}{c}$ and Δc_l. This invalidates the assumption made in Equation 5.1, thus explaining the difference. From this test it can be concluded that the errors behave approximately as expected. and that the combined resolutions will indeed provide enough accuracy.

6

3D METHOD EXTENSION

Up until this point the tool developed was only focused on 2D airfoil prediction. However, to make this tool a suitable tool for preliminary aircraft design and evaluation, it is necessary to extend the method into the third dimension. The extension into the third dimension is done by applying a quasi 3D method. This method entails dividing the wing into a number of 2D sections. The local 3D conditions for these sections are then calculated. These local conditions are then used to estimate the 3D performance of the wing.
First in Section 6.1 the region of validity of the quasi-3D method is discussed. In Section 6.2 the implementation of the model is treated. Then, the test cases are evaluated using a direct application of MSES and using GT-Approx in Section 6.3. Finally in Section 6.4 the results of GT-Approx are compared with the CFD data.

6.1. REGION OF VALIDITY

This method is fairly accurate for an entire clean wing. It is assumed that a well designed wing will minimize cross flow, thereby allowing accurate results for the entire wing, minus the outer most parts of the wing, where root- and tip-effects occur [42]. However, this will not affect the accuracy that much, because the highest values of c_l (and therefore the strongest shocks) will occur in the outboard part of the wing, before the tip. If the total circulation distribution is elliptical, the effect of tapering will increase the value of c_l towards the tip. On the outboard part of the wing, the combination of an elliptical circulation distribution and the effect of tapering result in the highest values of c_l, this is confirmed by Figure 6.1

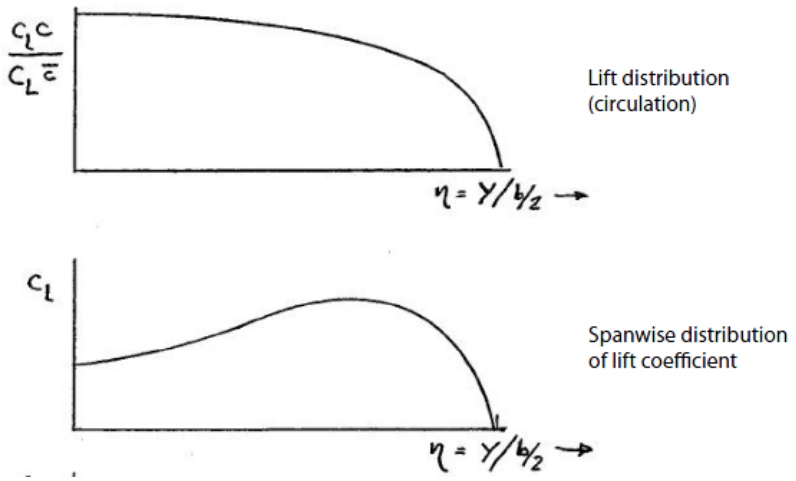


Figure 6.1: Development of circulation and local lift coefficient over the wing [43].

For a realistic wing, which includes a fuselage and engine, the region of validity is smaller. As the quasi 3D method assumes a simple sweep decomposition it will only give reliable results in the regions where the flow

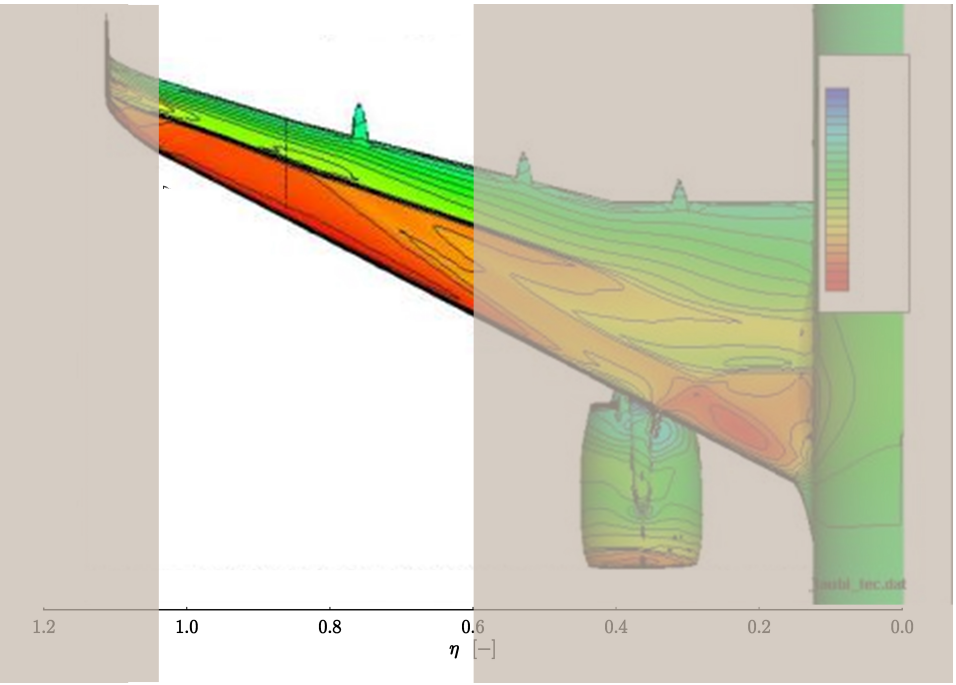


Figure 6.2: Region of validity of quasi 3D method.

is perpendicular to the chord line. To asses for which regions this is the case, the isobars for a typical Airbus aircraft are examined. Here it can be seen that for inboard regions the flow is highly three dimensional, and the isobars are not aligned with the sweep line. The region around $\eta = 0.34$ is heavily influenced by the engine placement. The shock occurs on the engine, leaving the area of the wing behind the engine in relatively low speeds, preventing shock waves on the wing. The effect of the engine spreads outboard and inboard, and strong influences can be seen from the root up until $\eta = 0.6$. These effects prevent any useful comparison between the quasi 3D method and CFD calculations for values of $\eta \leq 0.6$.

6.2. 3D METHOD IMPLEMENTATION

To investigate the accuracy of the 3D extension, two test cases are investigated:

- A320
  The original A320 wing
- A320-PL7a
  A modification of the original A320 wing, incorporating a chord and span extension, wing retwist and modified airfoil shapes

The main goal for this modification was to reduce the induced drag by achieving a more elliptic loading. To achieve this more loading was shifted outboard.
This will increase the value of c_l on the outboard wing resulting in an increase in wave drag. For both these cases, calculations using DLR'S TAU CFD code [44] have been performed. For both wings, CFD results are available for $Re = 2.57 \times 10^7$, $M = 0.78$, C_L values ranging from 0.4 to 0.75 in steps of 0.025.

6.2.1. AIRFOIL GEOMETRY

For both the wings, the wing planforms are known. In this thesis, not the entire wing is evaluated. Near the tip the wing is shaped according to 3D flow phenomena like the tip effect. This leads to distorted airfoil profiles, which have no meaning in a quasi-3D method like this one. The relevant parts of both planforms are shown are shown in Figure 6.3.

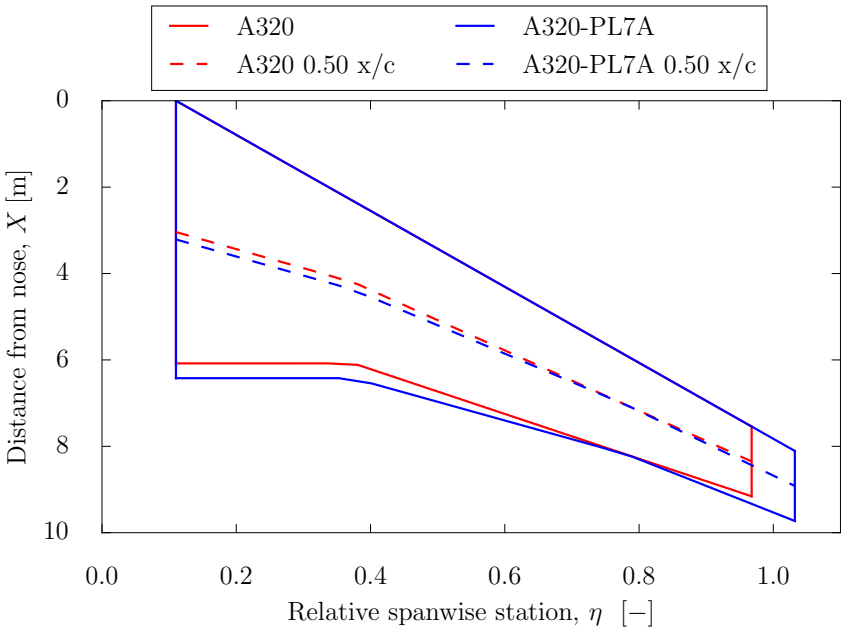


Figure 6.3: Comparison of the two test case geometries.

It should be noted here that η is based on the original A320 reference span, as per Airbus standard. For twenty equidistant locations on the η-range shown in the image airfoil geometries are extracted. The distribution of airfoil stations of the A320 planform is shown in Figure 6.4.

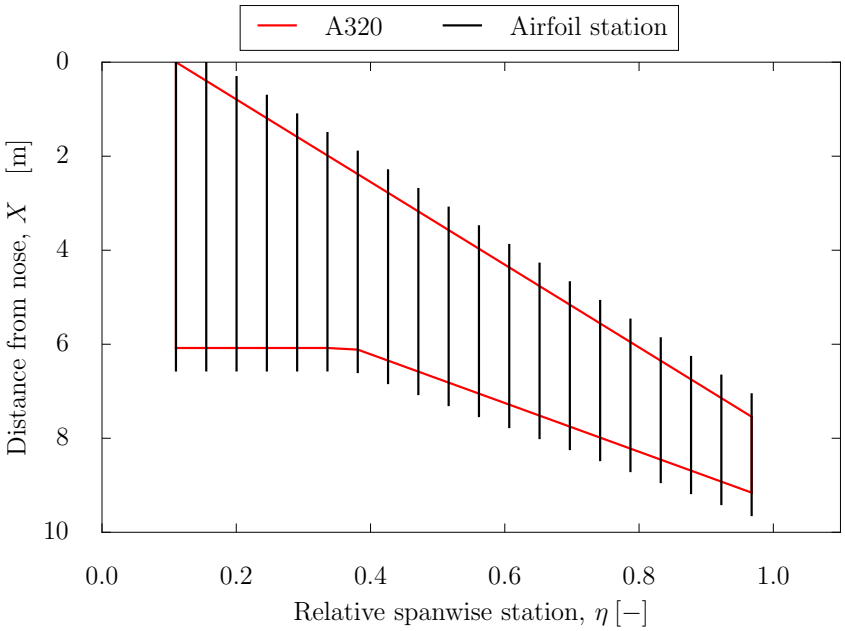


Figure 6.4: A320 planform with airfoil stations.

6.2.2. EFFECTIVE MACH NUMBER

To incorporate the effect of wing sweep, a correction is necessary. The simple sweep theory proposes to use the speed perpendicular to the sweep line, as shown by Equation 6.1.

$$M_\perp = M_\infty \cos \Lambda_{0.25c} \tag{6.1}$$

However, based on experience and empirical results, Torenbeek [45] proposes the following correction:

$$M_\perp = M_\infty \sqrt{\cos \Lambda_{0.25c}} \tag{6.2}$$

However, it was found that a better agreement with data was obtained using $\Lambda_{0.50c}$. This makes sense, as the shock usually lies at 50% chord [46]. The effective Mach number calculation is adjusted to:

$$M_\perp = M_\infty \sqrt{\cos \Lambda_{0.50c}} \tag{6.3}$$

6.2.3. EFFECTIVE REYNOLDS NUMBER

The test-case data calculations were performed for a Reynolds number of 2.57×10^7. The formula for the Reynolds number is given by:

$$Re_{wing} = \frac{\rho_\infty V_\infty \, MAC}{\nu_\infty} \tag{6.4}$$

Since MAC is known, it is possible to calculate the local Reynolds number as:

$$Re_\perp = \frac{\rho_\infty V_\perp c_\perp}{\nu_\infty} = \left(\frac{\rho_\infty V_\infty \cos \Lambda_{0.50c}}{\nu_\infty} \right) c \cos \Lambda_{0.50c} = \left(\frac{Re_{wing}}{MAC} \right) c \cos^2 \Lambda_{0.50c} = Re_{wing} \left(\frac{c}{MAC} \right) \cos^2 \Lambda_{0.50c} \tag{6.5}$$

6.2.4. c_l INPUT

In order to ensure comparability with the test-case, the values generated by the CFD code are used as input for the GT-approx model. As mentioned, lift distributions for $C_L = 0.4 - 0.75$ are available. Two of the circulations as calculated by the CFD code are shown in Figure 6.5.

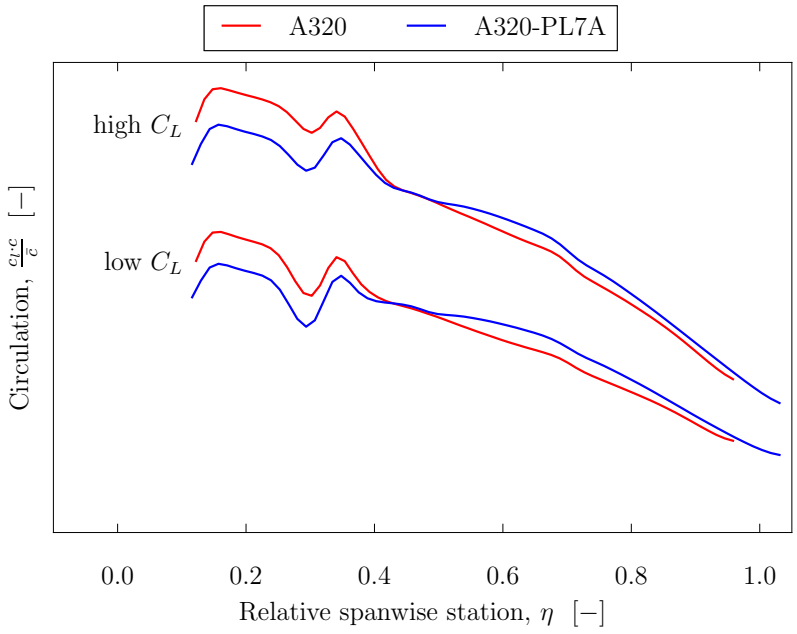


Figure 6.5: Selection of circulations for the two planforms.

From Figure 6.5 it can be seen that the modifications have indeed increased the value of the circulation in the outboard section on the wing, as was targeted. This has made the circulation distribution more elliptical, and will decrease the induced drag. To obtain the local lift coefficient, the effect of the local chord needs to be incorporated. The values from Figure 6.5 are multiplied with the average chord, and divided by the local chord to obtain the local lift coefficient, as shown in Figure 6.6.

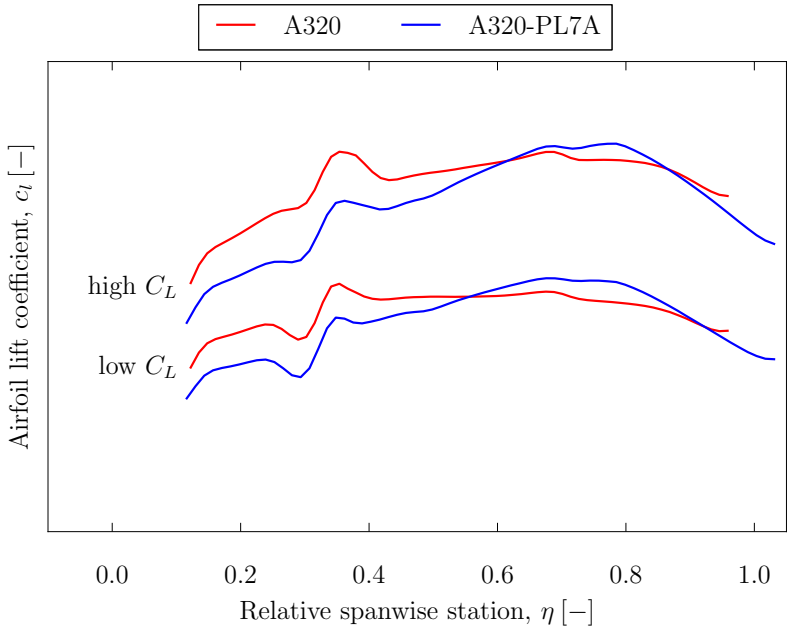


Figure 6.6: Selection of lift coefficient distribution.

Figure 6.6 shows that the local lift distribution looks like a typical lift distribution, as shown in Figure 6.1. The outboard increase of the circulation has led to higher values of the local lift coefficient. This will increase the wave drag.

6.2.5. C_{D_w} DETERMINATION

Now that all the input variables have been corrected to the effective values, it is possible to determine the value of effective $c_{d_{w_\perp}}$. This value needs to be converted back to the streamwise direction, to determine the actual drag experienced. This is done using a method proposed by Drela [47]. This method proposes the following formula:

$$C_{D_w} = c_{d_{w_\perp}} \cos^3 \Lambda_{0.50c} \tag{6.6}$$

6.3. TEST CASE: COMPARISON BETWEEN MSES AND GT-APPROX

For this test case, the following algorithm was executed:

1) Based on the geometry, for every of the 20 spanwise locations the following values are determined: $Re_\perp, M_\perp, c_{l_{local}}$.
2) From these values the boundaries are determined, $Re_{\perp_{\min}}, Re_{\perp_{\max}}, M_{\perp_{min}}$, $M_{\perp_{max}}$ and $c_{l_{local_{min}}}, c_{l_{local_{max}}}$.
3) For all spanwise locations the airfoils are extracted
4) For the variable ranges as determined in step 2), MSES is used to calculate values at the intervals as determined in Section 5.4 to form the input data-set for GT-Approx.
5) Based on this data-set, a GT-Approx model is generated.
6) For every spanwise location the actual value of $Re_\perp, M_\perp, c_{l_{local}}$ and η are given as input to the GT-Approx model, in order to predict outputs.
7) For every spanwise location the calculation is also performed with MSES using the same input- parameters.

The final result is shown against data calculated by MSES directly in Figure 6.7. From this it can be seen that GT-Approx is able to predict the results with good accuracy. The average difference between GT-Approx and MSES for values of $\eta > 0.60$ is equal to 0.21 drag counts.

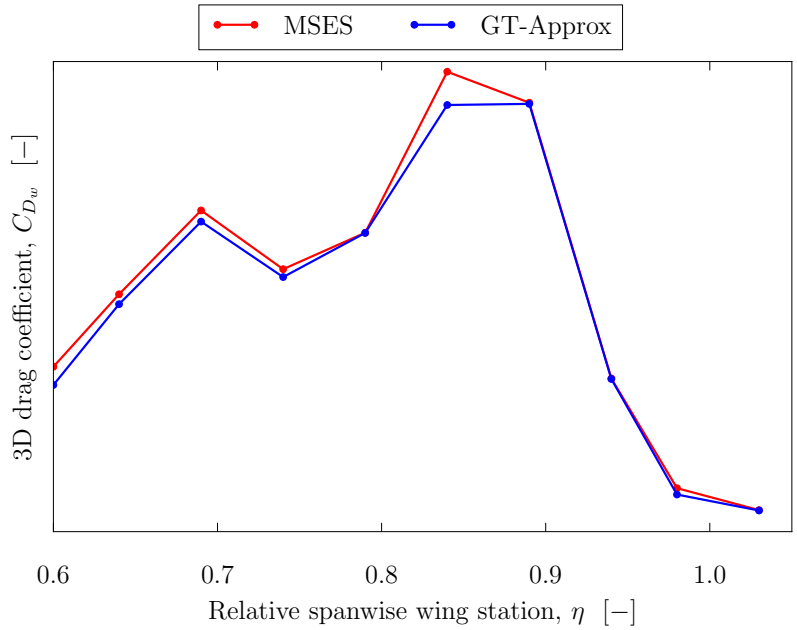


Figure 6.7: Comparison between MSES and GT-Approx approximation for A320-PL7A with $C_L = 0.625$.

The pressure distributions generated by GT-Approx and MSES are also compared. The result of this can be seen in Figure 6.8. The general comparison is good. GT-Approx accurately predicts the pressures. However, some features look somewhat different. At $\eta = 0.6$, around $X = 21$m, some differences can be seen. However, these differences are small.

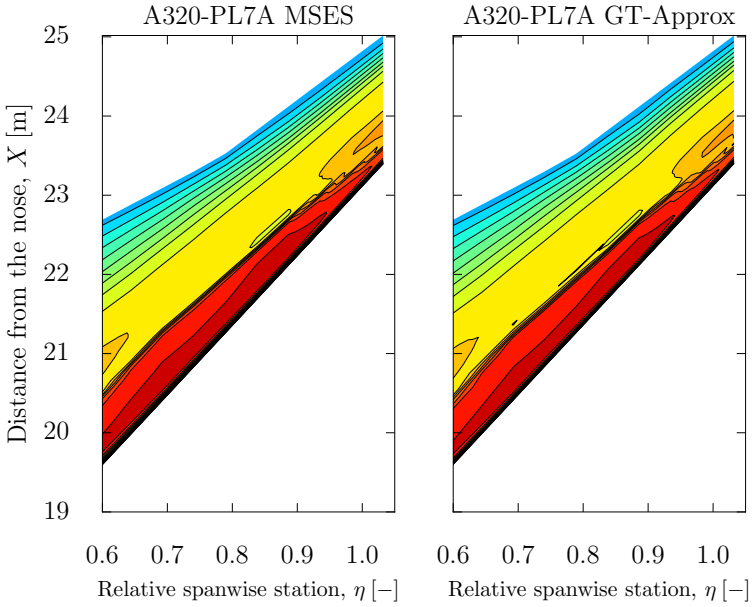


Figure 6.8: Comparison between pressures calculated by MSES and GT-Approx for $C_L = 0.625$.

When the running times are compared, the advantage of GT-Approx becomes apparent. For GT-Approx, three different computation times are required:

- Model preparation time
  The time needed to generate the output grid using MSES. For this calculation, a grid of 2.16×10^4 data points was used. A typical calculation of MSES requires 5.58 s, thus giving a model preparation time of 33 hours, 25 minutes and 43 seconds. As these calculations are not dependent on each other it possible to greatly reduce the computation by using parallel computing.

- Model loading time
  The time needed to initialize the model, which required once. For this calculation, a model loading time of 12 minutes and 34 seconds was needed.

- Model calculation time
  The time required to generate a single calculation, thus generating a single set of outputs from a single set of inputs. 5.13×10^{-4} *s* was needed to perform a single calculation in this case.

The benefit of using GT-Approx is that both the model preparation time and the model loading time can be performed outside an iterative multi disciplinary design loop. An issue using iterative design loops is that the disciplines are dependent on each other, and it is not possible to perform calculation simultaneously, therefore it is of vital importance to reduce the computation time per iteration. For one wing evaluation (using 20 span wise locations) MSES required 1.12×10^2 s. For the same calculation, GT-Approx required 1.03×10^{-2} s.
If only the Model loading time and the model calculation time are included (because the model preparation can be done before), GT-Approx is faster for computations that contain more than 7 wing evaluations.
In general it is concluded that GT-Approx is able to achieve the accuracy required for initial design stages, but with greatly reduced computation times per iteration.

6.4. TEST CASE: COMPARISON BETWEEN GT-APPROX AND CFD

In the previous subsection it was proven that GT-Approx is able to recreate the data generated by MSES. To asses the quality of GT-Approx it is validated against CFD calculations. The C_{D_w} calculated by GT-Approx is compared with CFD calculations in Figure 6.9

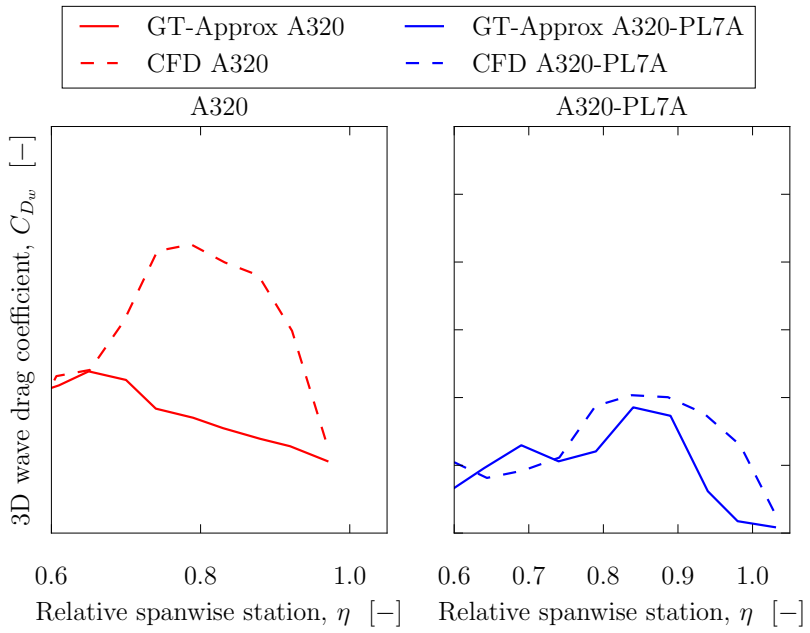


Figure 6.9: Comparison between GT-Approx and CFD calculations for both test cases with $C_L = 0.625$.

6.4.1. A320

For the A320, the calculations lack any resemblance. The wave drag predicted by GT-Approx shows a strong correlation with the c_l distribution, whereas the CFD-calculations do not. The c_l, shown in Figure 6.6 peaks at $\eta = 0.7$, and shows a moderate descent after that. This is also visible in the wave drag calculated by GT-Approx. The CFD calculations show a different distribution, with a maximum wave drag at $\eta = 0.8$. From this it is concluded that the quasi 3D application of MSES is too sensitive to c_l, and not sensitive enough for the effects of the local geometry. On the outboard section the quasi 3D and full 3D method should achieve similar results, as shown by Mariens [27]. One possible explanation could be that Mariens compared the total drag whereas in this work only the value of C_{D_w} is compared. As no further data for the 3D case is present, detailed research is not possible.

6.4.2. A320-PL7A

For the A320-PL7A, the result is more accurate. Figure 6.9 shows that both the wave drag predicted by CFD calculations and GT-Approx peak around $\eta = 0.9$, at approximately the same value of C_{D_w}. After that, GT-Approx predicts lower wave drag, whereas the CFD calculations show a higher value. This can be attributed to tip effects, which are present in the CFD calculations, but not in GT-Approx. These tip effects can be seen near the tip in Figure 6.10. Here the isobars show a highly irregular pattern, invalidating the simple sweep assumption.

Although the wave drag values in Figure 6.9 look accurate, from the pressure distribution comparison in Figure 6.10 it is clear that the pressure distibutions are not similar. Again, due to the lack of more detailed 3D CFD information, it is not possible to say what the cause of the difference is. However, as Figure 6.8 showed that the results for GT-Approx and MSES are very similar, it is clear that the errors stem from the inaccuracies in the quasi 3D method, rather than from the meta-modeling procedure.

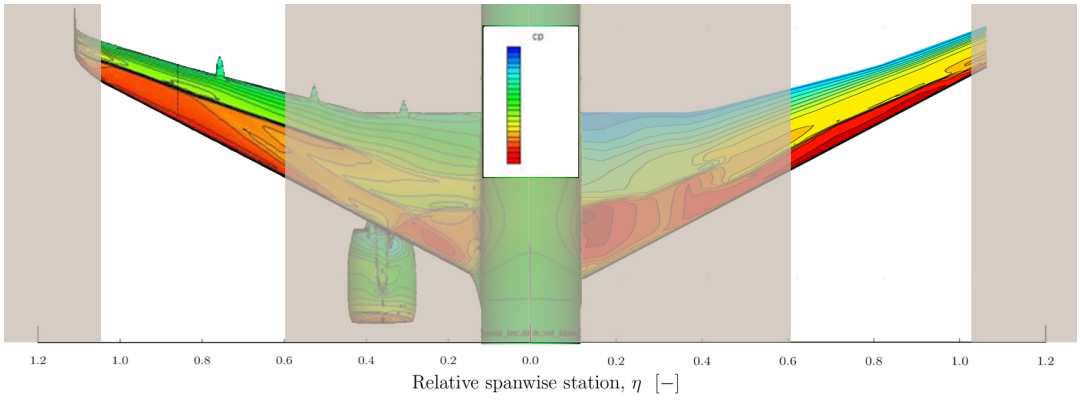


Figure 6.10: Comparison between CFD pressure distribution of a similar case [36] (left) and GT-Approx A320-PL7A (right), regions of invalidity indicated in gray.

7

CONCLUSION AND RECOMMENDATIONS

7.1. CONCLUSION

It is concluded that aerodynamic tool MSES is an accurate 2D method and that the meta-modeling method GT-Approx is able to accurately reproduce complex data generated by MSES. It is possible to improve the accuracy of the meta-modeling method by adding more data points. Accuracies of 0.21 drag counts are achieved, whilst reducing the in-the-loop computation time from 5.58 s to 5.13×10^{-4} s per calculation.
However, the comparison of MSES data reproduced by GT-Approx with CFD data showed that a quasi 3D application of MSES does not lead to accurate results. For one of the two test cases reasonable results for the wave drag prediction were achieved, whereas the second test case showed no correlation. The pressure distribution also lacked resemblance. Because of lack of more detailed data no cause for this difference was found.
In general it is concluded that the combination of an aerodynamic tool with a meta-model can be a successful one, but that is highly dependent on the accuracy and the application of the aerodynamic model.

7.2. RECOMMENDATIONS

As is shown in this work, the combination of a meta-modeling method with a computational aerodynamic method is not a feasible one. The main reason for this lies in the errors generated by the quasi 3D methodology. The transition from 2D to 3D can not be modeled in a simple way using the simple sweep theorem. It is therefore recommended to further investigate methodologies that are able to accurately represent the transition from 2D to 3D.
Furthermore it was established that the predictive capabilities of GT-Approx are highly dependent on the data provided. The accuracy was greatly increased by simply adding more data points. However, using Design of Experiment methodologies it is possible to add additional points in an intelligent way such that the increase in accuracy is maximized. This can greatly reduce the amount of data points required, whilst still achieving similar accuracies.
Another possibility to improve the meta-modeling procedure is so-called Data Fusion. These methods are geared towards combining lower and higher fidelity data. Data Fusion allows the usage of higher fidelity data for a few difficult cases with lower fidelity cases for easier cases. MSES experienced many problems for difficult cases, such as high M, high c_l calculations. For this case, Data Fusion could employ MSES for easier calculations combined with CFD calculations for the more difficult cases. This could greatly improve the accuracy or alternatively decrease the number of points required to reach a certain accuracy.

BIBLIOGRAPHY

[1] S. SHARIAR and E. TOPAL, *When will fossil fuel reserves be diminished?* Energy Policy **37**, pp. 181 (2009).

[2] L. JENKINSON, P. SIMPKIN, and D. RHODES, *Civil Jet Aircraft Design* (Arnold, 1999).

[3] J. ROSKAM and E. CHUAN-TAU, *Airplane Aerodynamics and Performance* (DARcorporation, 1997).

[4] G. SCHRAUF, *Katnet, key aerodynamic technologies for aircraft performance improvement,* in *Fifth Community Aeronautical Days* (2006).

[5] O. GUR, W. MASON, and J. SCHETY, *Full-configuration drag estimation,* Journal of Aircraft **47**, pp. 1356 (2010).

[6] M. DRELA, *Xfoil: An analysis and design system for low reynolds number airfoils,* in *Proceedings of the Conference Notre Dame, Indiana, USA*, Vol. 54, edited by T. MUELLER (1989) pp. pp 1–12.

[7] J. D. ANDERSON JR., *Fundamentals of Aerodynamics* (McGraw-Hill, 2001).

[8] D. SCHOLZ and S. CIORNEI, *Mach number, relative thickness, sweep and lift coefficient of the wing - an empirical investigation of parameters and equations,* in *Deutscher Luft- und Raumfahrtkongreß* (2005).

[9] J. SCHULZE, *Development of a drag prediction method for flexible aircraft over a complete cruise flight mission*, Master's thesis, Technische Universität Darmstadt (2013).

[10] American Institute of Aeronautics and Astronautics, *White paper on industrial experience with mdo,* AIAA Technical Committee on Multidisiplinary Design Optimization (MDO) (1999).

[11] J. KLEIJNEN, *Regression metamodels for generalizing simulation results,* IEEE Transactions on Systems Man and Cybernetics **9**, pp. 93 (1979).

[12] F. VIANA and R. HAFTKA, *Using multiple surrogates for metamodeling,* in *7th ASMO-UK/ISSMO International Conference on Engineering Design Optimization* (2008).

[13] J. AUDET, C. DENNIS, D. MOORE, A. BOOKER, and P. FRANK, *A surrogate-model-based method for constrained optimization,* in *8th Symposium on Multidisciplinary Analysis and Optimization* (2000).

[14] S. KOZIEL and L. LEIFSSON, *Multi-objective airfoil design using variable-fidelity cfd simulations and response surface surrogates,* in *10th AIAA Multidisciplinary Design Optimization Specialist Conference* (2014).

[15] S. KOZIEL and L. LEIFSSON, *Multi-fidelity design optimization of transonic airfoils using physics-based surrogate modeling and shape-preserving response prediction,* Journal of Computational Science **1**, pp. 98 (2010).

[16] J. ROSKAM, *Airplane Design: Part VI: Preliminary Calculation of Aerodynamic, Thrust and Power Characteristics* (DARcorporation, 2000).

[17] P. GARABEDIAN and D. KORN, *A Theory of Supercritical Wing Sections: With Computer Programs and Examples, Volume 2* (Springer, 1975).

[18] B. MALONE and W. MASON, *Multidisciplinary optimization in aircraft design using analytic technology models,* Journal of Aircraft **32**, pp. 431 (1995).

[19] W. HILTON, *High Speed Aerodynamics* (Longmans, Green & Co., Ltd.;, 1952).

[20] M. FREESTONE, *VGK method for two-dimensional aerofoil sections*, IHS ESDU, part 1: principles and results ed. (2004).

[21] B. THWAITES, *Approximate calculation of the laminar boundary-layer,* Aerodynamics Quarterly **1**, 245 (1949).

[22] K. STEWARTSON, *Correlated compressible and incompressible boundary layers,* Proc. R. Soc. Lond. Series A **200**, 84 (1949).

[23] J. GREEN, D. WEEKS, and J. BROOMAN, *Prediction of turbulent boundary-layers and wakes in compressible flow by a lag-entrainment method,* R.A.E. Farnborough, aeronautical research council reports & memoranda no 3791 ed. (1977).

[24] R. LOCK, *A method of determining the wave drag and its spanwise distribution on a finite wing in transonic flow*, IHS ESDU (1987).

[25] R. LOCK, *Prediction of the drag of wings at subsonic speeds by viscous/inviscid interaction techniques,* AGARD Report , 723 (1985).

[26] M. PADULO, J. MAGINOT, and M. GUENOV, *Airfoil design under uncertainty with robust geometric parameterization,* in *50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference* (2009).

[27] J. MARIENS, A. ELHAM, and M. VAN TOOREN, *Influence of weight modelling on the outcome of wing design using multidisciplinary design optimisation techniques,* Aeronautical Journal **117**, pp. 871 (2013).

[28] M. DRELA, *Mses multi-element airfoil desing/analysis software summary,* (1994).

[29] D. LEE, L. GONZALEZ, J. PERIAUX, and E. ONATE, *Robust aerodynamic design optimisation of morphing aerofoil/wing using distributed moga,* in *28TH INTERNATIONAL CONGRESS OF THE AERONAUTICAL SCIENCES* (2012).

[30] J. DRIVER and Z. D.W., *Numerical aerodynamic optimization incorporating laminar-turbulent transition prediction,* AIAA JOURNAL **45**, pp. 1810 (2007).

[31] D. R. HARTREE, *On an equation occurring in falkner and skan's approximate treatment of the equations of the boundary layer,* Mathematical Proceedings of the Cambridge Philosophical Society **33**, pp 223 (1937).

[32] M. DRELA, *Two-Dimensional Transonic Aerodynamic Desing and Analysis Using The Euler Equations*, Ph.D. thesis, Massachusets Institute of Technology (1985).

[33] T. SWAFFORD, *Analytical approximation of two-dimensional separated turbulent boundary-layer velocity profiles,* AIAA Journal **21**, pp 923 (1983).

[34] M. DRELA, *A User's Guide to MSES 3.05*, MIT Department of Aeronautics and Astronautics (2007).

[35] C. HARRIS and J. BLACKWELL JR., *Wind-Tunnel Investigation Of Effects of Rear Upper Surface Modification On An NASA Supercritical Airfoil*, Tech. Rep. (NASA, 1972).

[36] R. NORTHAM, *36m Baseline TEX Development - Aero Design - Draft*, Tech. Rep. (Airbus, 2014).

[37] D. YAROTSKY, M. BELYAEV, A. KRYMOVA, E. ZAYTSEV, Y. YANOVICH, and E. BURNAEV, *GT Approx Generic Tool for Approximation*, DATADVANCE, 1st ed. (2013), inhouse document.

[38] S. PRUESS, *An algorithm for computing smoothing splines in tension,* Computing **19**, pp. 365 (1977).

[39] E. TORENBEEK, *Advanced Aircraft Design* (Wiley, 2013).

[40] F. M. WHITE, *Viscous fluid flow*, edited by L. BEAMESDERFER and J. Morriss (McGra, 1991).

[41] F. DEKKING, C. KRAAIKAMP, H. LOPUHAÄ, and L. MEESTER, *A Modern Introduction to Probability and Statistics: Understanding Why and How* (Springer-Verlag, 2005).

[42] J. MARIENS, *Wing Shape Multidisciplinary Design Optimization*, Master's thesis, Delft University of Technology (2012).

[43] E. OBERT, *Aerodynamic design of transport aircraft* (IOS Press, 2009).

[44] Unknown, *Description of DLR TAU Code*, DLR (Retrieved: Oct 2014).

[45] E. TORENBEEK, *Synthesis of Subsonic Airplane Design* (Kluwer Academic Publishers, 1984).

[46] D. Schuster, *Lessons learned in the selection and development of test cases for the aeroelastic prediction workshop: Rectangular supercritical wing,* in *Aerospace Sciences Meetings* (American Institute of Aeronautics and Astronautics, 2013) pp. –.

[47] M. Drela, *N+3 Aircraft Concept Designs and Trade Studies - Volume 2 (Appendices): Design Methodologies for Aerodynamics, Structures, Weight, and Thermodynamic Cycles (Appendix A: TASOPT - Transport Aircraft System OPTimization)*, Tech. Rep. Final Report NASA/CR-2010-216794/VOL2 (NASA, 2010).

A

PYTHON CODE

A.1. TEST CASE IMPLEMENTATION

This section contains the code used to verify the different methods against windtunnel data.

comparefig.py This script loads the test-data, and generates the plots to compare the computational methods with the data.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 23 14:43:31 2014

@author: THC0LQ
"""
import cPickle as pickle
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate as interpolate
from scipy.interpolate import interp1d
import matplotlib as mpl
mpl.rc('text', usetex = True)
mpl.rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})
#testdata
[TestCD040,TestM040]=pickle.load( open( "CNtest-0.4.p", "rb" ) )
[TestCD055,TestM055]=pickle.load( open( "CNtest-0.55.p", "rb" ) )
[TestCD070,TestM070]=pickle.load( open( "CNtest-0.7.p", "rb" ) )

#handbook
text_file= open("handbook04.txt", "r")
handbook = text_file.read().split(',')
handbook040 = [float(x) for x in handbook][1:]
handbook040=handbook040[:3]+handbook040[4:]
text_file= open("handbook055.txt", "r")
handbook = text_file.read().split(',')
handbook055 = [float(x) for x in handbook][1:]
handbook055=handbook055[:3]+handbook055[4:]
text_file= open("handbook070.txt", "r")
handbook = text_file.read().split(',')
handbook070 = [float(x) for x in handbook][1:]
handbook070=handbook070[:3]+handbook070[4:]

#import pdb
```

```
#pdb.set_trace()
import os
#os.get
#computational
resultsmat070 = pickle.load( open( os.getcwd()+"\\res 100\\savetgoed-0.7.p", "rb" )
    ↪ )
resultsmat055 = pickle.load( open( os.getcwd()+"\\res 100\\savet-0.55.p", "rb" ) )
resultsmat040 = pickle.load( open( os.getcwd()+"\\res 100\\savetgoed-0.4.p", "rb" )
    ↪ )
sizes=np.shape(resultsmat070)
Machrange = np.linspace(0.6,0.81,sizes[1])
p=[]
q=[]
r=[]
plt.rc('figure', figsize=(11.5,15))
fix,ax = plt.subplots(3, 1, sharex='col',sharey='row')
fix.subplots_adjust(top=0.85)
f = plt.gcf()
f.set_size_inches(8.27,11.69)

ax[0].plot(TestM040,TestCD040,linewidth=2.0)
p.extend(ax[0].plot(Machrange,resultsmat040[0,:,2],linewidth=2.0,c='r'))
p.extend(ax[0].plot(Machrange,resultsmat040[1,:,2],linewidth=2.0,c='g'))
p.extend(ax[0].plot(Machrange,resultsmat040[2,:,2],linewidth=2.0,c='c'))
ax[0].tick_params(axis='y', labelsize=18)
#ax[0].text(0.56, 0.011, '$C_L$=0.40', fontsize=40,
#          bbox={'facecolor':'white', 'alpha':0, 'pad':10})
handbookf=handbook040
color=['b','g','r','c','m','y','k']
import numpy as np
y=np.linspace(-0.01,0.07,100)
import pdb
from matplotlib.lines import Line2D
markers = []

for m in Line2D.markers:
    try:
        if len(m) == 1 and m != ' ':
            markers.append(m)
    except TypeError:
        pass
del markers[4]
del markers[2]
ax[1].set_ylim([-0.0025, 0.015])
#pdb.set_trace()
h,=ax[0].plot(np.array([float(handbookf[0])]*len(y)),y,c='b',linestyle='--')
#d.append(h)
h,=ax[0].plot(np.array([float(handbookf[1])]*len(y)),y,c='r',linestyle='--')
#d.append(h)
h,=ax[0].plot(np.array([float(handbookf[2])]*len(y)),y,c='g',linestyle='--')
#d.append(h)
h,=ax[0].plot(np.array([float(handbookf[3])]*len(y)),y,c='#ffa500',linestyle='--')
#d.append(h)
h,=ax[0].plot(np.array([float(handbookf[4])]*len(y)),y,c='b',linestyle=':')
#d.append(h)
h,=ax[0].plot(np.array([float(handbookf[5])]*len(y)),y,c='r',linestyle=':')
```

```
#d.append(h)
h,=ax[0].plot(np.array([float(handbookf[6])]*len(y)),y,c='g',linestyle=':')
#d.append(h)
#     else:
#         ax[0].plot([float(handbookf[k]),float(handbookf[k])],[-0.01,0.07],linestyle
    ↪ ='--',linewidth=4.0)
TestM=TestM040
TestCD=TestCD040
data=resultsmat040[0,:,2]
CDdat=data[np.isfinite(data)]
Mdat=Machrange[np.isfinite(data)]
Mdat=Mdat[-sizes[1]/3:]
CDdat=CDdat[-sizes[1]/3:]

tck=interpolate.splrep(Mdat, CDdat, s=0)
#reconstruction of data
fitCD=interpolate.splev(Mdat, tck, der=0)
#derivative of calc data
fitCDd=interpolate.splev(Mdat, tck, der=1)
# fit to adjusted data
tcka=interpolate.splrep(Mdat, fitCDd-0.1, s=0)
#fitTest=interpolate.splev(TestM, tck2, der=0)
root=min(interpolate.sproot(tcka, mest=10))
f1=interp1d(Mdat, CDdat)
CDroot=f1(root)
ax[0].set_ylabel('$C_d \: [-]$',fontsize=18)
ax[0].set_ylim([-0.0025, 0.015])
tck2=interpolate.splrep(TestM, TestCD, s=0)
#reconstruction of testdata
fitCDtest=interpolate.splev(TestM, tck2, der=0)
#derivative of calc data
fitCDtestd=interpolate.splev(TestM, tck2, der=1)
# fit to adjusted data
tckta=interpolate.splrep(TestM, fitCDtestd-0.1, s=0)
#fitTest=interpolate.splev(TestM, tck2, der=0)
roottest=interpolate.sproot(tckta, mest=10)
f2=interp1d(TestM, TestCD)
CDroottest=f2(roottest)
ax[0].plot(root,CDroot,'r*',markersize=20)
ax[0].plot(roottest,CDroottest,'b*', markersize=20)
ax[0].set_title(r'$C_l=0.40$', fontsize=18)
ax[0].tick_params(axis='y', pad=15)
#box = ax[0].get_position()
#ax[0].set_position([box.x0, box.y0, box.width * 0.8, box.height])




ax[1].plot(TestM055,TestCD055,linewidth=2.0)
p.extend(ax[1].plot(Machrange,resultsmat055[0,:,2],linewidth=2.0,c='r'))
p.extend(ax[1].plot(Machrange,resultsmat055[1,:,2],linewidth=2.0,c='g'))
p.extend(ax[1].plot(Machrange,resultsmat055[2,:,2],linewidth=2.0,c='c'))
ax[1].set_ylabel('$C_d \: [-]$',fontsize=18)
ax[1].tick_params(axis='y', labelsize=18)
ax[1].tick_params(axis='y', pad=15)
```

```
#ax[1].text(0.56, 0.011, '$C_L$=0.55', fontsize=40,
#         bbox={'facecolor':'white', 'alpha':0, 'pad':10})
handbookf=handbook055
ax[1].set_ylim([-0.0025, 0.015])
h,=ax[1].plot(np.array([float(handbookf[0])]*len(y)),y,c='b',linestyle='--')
#d.append(h)
h,=ax[1].plot(np.array([float(handbookf[1])]*len(y)),y,c='r',linestyle='--')
#d.append(h)
h,=ax[1].plot(np.array([float(handbookf[2])]*len(y)),y,c='g',linestyle='--')
#d.append(h)
h,=ax[1].plot(np.array([float(handbookf[3])]*len(y)),y,c='#ffa500',linestyle='--')
#d.append(h)
h,=ax[1].plot(np.array([float(handbookf[4])]*len(y)),y,c='b',linestyle=':')
#d.append(h)
h,=ax[1].plot(np.array([float(handbookf[5])]*len(y)),y,c='r',linestyle=':')
#d.append(h)
h,=ax[1].plot(np.array([float(handbookf[6])]*len(y)),y,c='g',linestyle=':')
#d.append(h)

#    if k<4:
#        ax[1].plot([float(handbookf[k]),float(handbookf[k])],[-0.01,0.07],linestyle
    ↪ =':',linewidth=4.0)
#    else:
#        ax[1].plot([float(handbookf[k]),float(handbookf[k])],[-0.01,0.07],linestyle
    ↪ ='--',linewidth=4.0)
#box = ax[1].get_position()
#ax[1].set_position([box.x0, box.y0, box.width * 0.8, box.height])



TestM=TestM055
TestCD=TestCD055
data=resultsmat055[0,:,2]
CDdat=data[np.isfinite(data)]
Mdat=Machrange[np.isfinite(data)]
Mdat=Mdat[-sizes[1]/3:]
CDdat=CDdat[-sizes[1]/3:]

tck=interpolate.splrep(Mdat, CDdat, s=0)
#reconstruction of data
fitCD=interpolate.splev(Mdat, tck, der=0)
#derivative of calc data
fitCDd=interpolate.splev(Mdat, tck, der=1)
# fit to adjusted data
tcka=interpolate.splrep(Mdat, fitCDd-0.1, s=0)
#fitTest=interpolate.splev(TestM, tck2, der=0)
root=min(interpolate.sproot(tcka, mest=10))
f1=interp1d(Mdat, CDdat)
CDroot=f1(root)


tck2=interpolate.splrep(TestM, TestCD, s=0)
#reconstruction of testdata
fitCDtest=interpolate.splev(TestM, tck2, der=0)
#derivative of calc data
fitCDtestd=interpolate.splev(TestM, tck2, der=1)
```

```
# fit to adjusted data
tckta=interpolate.splrep(TestM, fitCDtestd-0.1, s=0)
#fitTest=interpolate.splev(TestM, tck2, der=0)
roottest=interpolate.sproot(tckta, mest=10)
f2=interp1d(TestM, TestCD)
CDroottest=f2(roottest)
ax[1].plot(root,CDroot,'r*',markersize=20)
ax[1].plot(roottest,CDroottest,'b*', markersize=20)
ax[1].set_title(r'$C_l=0.55$', fontsize=20)



h0,=ax[2].plot(TestM070,TestCD070,linewidth=2.0)
p.extend(ax[2].plot(Machrange,resultsmat070[0,:,2],linewidth=2.0,c='r'))
p.extend(ax[2].plot(Machrange,resultsmat070[1,:,2],linewidth=2.0,c='g'))
p.extend(ax[2].plot(Machrange,resultsmat070[2,:,2],linewidth=2.0,c='#ffa500'))
ax[2].set_ylabel('$C_d \: [-]$',fontsize=18)
ax[2].tick_params(axis='y', labelsize=18)
ax[2].tick_params(axis='x', labelsize=18)
ax[2].tick_params(axis='x', pad=15)
#ax[2].text(0.56, 0.011, '$C_L$=0.70', fontsize=40,
#          bbox={'facecolor':'white', 'alpha':0, 'pad':10})
#box = ax[2].get_position()
#ax[2].set_position([box.x0, box.y0, box.width * 0.8, box.height])
handbookf=handbook070
#h=np.zeros(9)
d=[]
#plt.xlim((0.65, 0.90001))

#for k,nr in enumerate(handbookf):
h,=ax[2].plot(np.array([float(handbookf[0])]*len(y)),y,c='b',linestyle='--')
d.append(h)
h,=ax[2].plot(np.array([float(handbookf[1])]*len(y)),y,c='r',linestyle='--')
d.append(h)
h,=ax[2].plot(np.array([float(handbookf[2])]*len(y)),y,c='g',linestyle='--')
d.append(h)
h,=ax[2].plot(np.array([float(handbookf[3])]*len(y)),y,c='#ffa500',linestyle='--')
d.append(h)
h,=ax[2].plot(np.array([float(handbookf[4])]*len(y)),y,c='b',linestyle=':')
d.append(h)
h,=ax[2].plot(np.array([float(handbookf[5])]*len(y)),y,c='r',linestyle=':')
d.append(h)
h,=ax[2].plot(np.array([float(handbookf[6])]*len(y)),y,c='g',linestyle=':')
d.append(h)
#h,=ax[2].plot(np.array([float(handbookf[7])]*len(y)),y,c='#ffa500',linestyle=':')
#d.append(h)
#    h,=ax[2].plot(np.array([float(handbookf[k])]*len(y)),y,linestyle='None',marker=
    ↪ markers[k])
#    h,=ax[2].plot(np.array([float(handbookf[k])]*len(y)),y,linestyle='None',marker=
    ↪ markers[k])
#    h,=ax[2].plot(np.array([float(handbookf[k])]*len(y)),y,linestyle='None',marker=
    ↪ markers[k])

#    if k<4:
```

```
#          h,=ax[2].plot([float(handbookf[k]),float(handbookf[k])],[-0.01,0.07],
    ↪ linestyle=':',linewidth=4.0)
#      d.append(h)
#      else:
#          h,=ax[2].plot([float(handbookf[k]),float(handbookf[k])],[-0.01,0.07],
    ↪ linestyle='--',linewidth=4.0)
#          d.append(h)

TestM=TestM070
TestCD=TestCD070
data=resultsmat070[0,:,2]
CDdat=data[np.isfinite(data)]
Mdat=Machrange[np.isfinite(data)]
Mdat=Mdat[-sizes[1]/3:]
CDdat=CDdat[-sizes[1]/3:]

tck=interpolate.splrep(Mdat, CDdat, s=0)
#reconstruction of data
fitCD=interpolate.splev(Mdat, tck, der=0)
#derivative of calc data
fitCDd=interpolate.splev(Mdat, tck, der=1)
# fit to adjusted data
tcka=interpolate.splrep(Mdat, fitCDd-0.1, s=0)
#fitTest=interpolate.splev(TestM, tck2, der=0)
root=min(interpolate.sproot(tcka, mest=10))
f1=interp1d(Mdat, CDdat)
CDroot=f1(root)
print len(d)

tck2=interpolate.splrep(TestM, TestCD, s=0)
#reconstruction of testdata
fitCDtest=interpolate.splev(TestM, tck2, der=0)
#derivative of calc data
fitCDtestd=interpolate.splev(TestM, tck2, der=1)
# fit to adjusted data
tckta=interpolate.splrep(TestM, fitCDtestd-0.1, s=0)
#fitTest=interpolate.splev(TestM, tck2, der=0)
roottest=interpolate.sproot(tckta, mest=10)
f2=interp1d(TestM, TestCD)
CDroottest=f2(0.790)
h1,=ax[2].plot(root,CDroot,'r*',markersize=20)
h2,=ax[2].plot(roottest,CDroottest,'b*', markersize=20)
ax[2].set_title(r'$C_l=0.70$', fontsize=20)
len(d)
#pdb.set_trace()
namepool = ['Test data','MSES','VGK','TSFOIL2','Toorenbeek Extended','Shevell','Kroo
    ↪ ','KLM','Jenkinson','Raymer',r'B\"{o}ttger','$M_{DD}$ MSES','$M_{DD}$ Test
    ↪ Case']
plt.figlegend((h0,p[0],p[1],p[2],d[0],d[1],d[2],d[3],d[4],d[5],d[6],h1,h2),namepool,
    ↪  loc = 'upper center',prop={'size':15}, ncol=3, labelspacing=0. )


plt.xlim((0.65, 0.90001))
plt.ylim((-0.0025, 0.015))
plt.xlabel('$M \:\:\: [-]$',fontsize=18)
```

```
#ax[i].tick_params(axis='y', pad=15)
#plt.ylabel('$C_D$',fontsize=40)
#figManager = plt.get_current_fig_manager()
#figManager.window.showMaximized()
plt.gcf().subplots_adjust(left=0.15)
f.savefig('MDDcomp.pdf')
#f.savefig('Z:\\Thesis\\Thesis Report\\Images\\MDDcomp.pdf')
plt.show()
```

A.1.1. MSES IMPLEMENTATION

MSESmain.py

This is the master file used to generate the MSES output, it calls the other slave functions.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 09 15:37:28 2014

@author: THC0LQ
"""
def MSESmain(airfoil,state,calctype):
    import pdb
#if 1==1:


    import MSESblade
    import MSESMSETrun
    import MSESMSESwrite
    import MSESMSESrun
    import MSESMSESread
    #print airfoil
    runname='try2'
    numit=1000
    MSESblade.MSESblade(airfoil,runname)
    MSESMSETrun.MSESMSETrun(state)

    state=map(float, state)
    MSESMSESwrite.MSESMSESwrite(state,runname,calctype)
#    pdb.set_trace()
    MSESMSESrun.MSESMSESrun(runname,numit)
    Coeffs=MSESMSESread.MSESMSESread()
    return Coeffs
```

importcode.py

This script loads the airfoil data from the text files.

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 08 16:51:47 2014

@author: THC0LQ
"""
def importpy(files):
    import numpy as np
    for nr,name in enumerate(files):
        li = [i.strip().split() for i in open(name).readlines()]
        myarray = np.asarray(li)
#        print myarray
```

```
        matrix = myarray.astype(np.float)
    xcor = matrix[:,0]
    yup = matrix[:,1]
    ylow = matrix[:,2]
    airfoil=np.vstack((xcor, yup, ylow))
    airfoil = airfoil.T
    return airfoil
```

MSESblade.py
This script generates the MSET blade input file.

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 12 16:58:35 2014

@author: THC0LQ
"""


def MSESblade(airfoil,runname):
    import numpy as np
    import os
    x=np.concatenate((airfoil[::-1,0],airfoil[:,0]))
    vals = np.concatenate((airfoil[::-1,2],airfoil[:,1]))
    airfoil=np.vstack((x,vals))
    airfoil=airfoil.T
    f= open(os.path.dirname(os.path.realpath(__file__))+'\\blade.'+runname, 'wt')
    f.write('A359_0520 \n')
    f.write('  -4.00000    5.00000   -5.5    5.5 \n')
    for i,name in enumerate(airfoil):
        if airfoil[i][1]<=0:
            f.write('    '+"{:1.5f}".format(airfoil[i][0])+'   '+"{:1.5f}".format(
                ↪ airfoil[i][1])+'\n')
        else:
            f.write('    '+"{:1.5f}".format(airfoil[i][0])+'    '+"{:1.5f}".format(
                ↪ airfoil[i][1])+'\n')
    f.close()
```

MSESMSESwrite.py
This script calls MSET, to convert the MSET blade input file to an MSES blade input file.

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 12 13:16:51 2014

@author: THC0LQ
"""
def MSESMSESwrite(state,runname,calctype):
    import os
    f = open(os.path.dirname(os.path.realpath(__file__))+'\\mses.'+runname, 'wt')
    f.write('  3   4   5\n')
    if 'alpha' in calctype:
        f.write('  3   4   5\n')
    if 'CL' in calctype:
        f.write('  3   4   6\n')
    f.write('    '+"{:1.5f}".format(state[3])+'      '+"{:1.5f}".format(state[1])+'
        ↪     '+"{:1.5f}".format(state[2])+'   16.50000 \n')
    f.write('  4   2\n')
```

```
    f.write(' '+"{:1.4f}".format(state[0]/(10**8))+'E+08 5.0 \n')
    f.write('     '+"{:1.5f}".format(state[4])+'      '+"{:1.5f}".format(state[4])+' \
        ↪ n')
    f.write('     .99000     1.00000 \n')
    f.close
```

MSESMSESrun.py
This script calls executes MSES.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 09 14:55:30 2014

@author: THC0LQ
"""
import os, fnmatch
def find(pattern, path):
    result = []
    for root, dirs, files in os.walk(path):
        for name in files:
            if fnmatch.fnmatch(name, pattern):
                result.append(os.path.join(root, name))
    return result

def MSESMSESrun(runname,numit):
#if 1==1:
#    runname='try2'
    import os
    curdir = os.getcwd()
    from subprocess import CalledProcessError, Popen, PIPE
    #delete previous files
    resfiles = [os.path.join(dirpath, f)
                for dirpath, dirnames, files in os.walk(curdir)
                for f in fnmatch.filter(files, 'da*')]
#    print resfiles
    for name in resfiles:
        os.remove(name)
    cmd = os.path.dirname(os.path.realpath(__file__))+"\\mses.exe "+runname
#    print cmd
    input_data = os.linesep.join([str(numit), '0'])
    p = Popen(cmd, stdin=PIPE,stdout=PIPE, bufsize=0)
    p.communicate(input_data.encode('ascii'))
    if p.returncode != 0:
       raise CalledProcessError(p.returncode, cmd)
```

MSESMSESread.py
After a MSES run, this file reads the output file into Python.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 09 15:38:52 2014

@author: THC0LQ
"""
def MSESMSESread():
#if 1==1:
    import os
    import numpy as np
```

```
f = open(os.path.dirname(os.path.realpath(__file__))+"\\daflow.try2", "r")
Co2 = {'CD1':[], 'CD2':[], 'CDV':[], 'CDP':[], 'CDF':[], 'CL':[], 'CM':[], 'ALP':[], '
    ↪ DISTRI':[]}
xcor=[]
pres=[]
try:
    while f.tell() != os.fstat(f.fileno()).st_size:
        line = f.readline()
        if '   MA          RE          ALP         CA          CM' in line:
            data=f.readline()
            splitted=data.split()
            Co2['CD1']=np.inf
            Co2['CD2']=float(splitted[7])
            Co2['CDV']=float(splitted[8])
            Co2['CDP']=np.inf
            Co2['CDF']=np.inf
            Co2['CL']=float(splitted[3])
            Co2['CM']=float(splitted[4])
            Co2['ALP']=float(splitted[2])
        if "#    X/C         CP          Z/C         D1S         ME          THETA       CFUE
            ↪         H12" in line:
            end = False
            while end is False:
                values=f.readline()
                split=values.split()
                try:
                    if float(split[0])<1:
                        xcor=np.append(xcor,float(split[0]))
                        pres=np.append(pres,float(split[1]))
                    else:
                        1==1
                except:
                    end = True
    distri=np.concatenate((xcor.reshape(-1,1),pres.reshape(-1,1)),axis=1)
    Co2['DISTRI']=distri
except:
    Co2['CD1']=np.inf
    Co2['CD2']=np.inf
    Co2['CDV']=np.inf
    Co2['CDP']=np.inf
    Co2['CDF']=np.inf
    Co2['CL']=np.inf
    Co2['CM']=np.inf
    Co2['ALP']=np.inf
    Co2['DISTRI']=np.inf
return Co2
```

A.1.2. VGK IMPLEMENTATION

vgk_control_multiple_inputs.py

This is the master file used to generate the VGK output, it calls the other slave functions.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 10:10:50 2014

@author: THC0LQ
```

```
"""
#if 1==1:
def vgk(airfoil,state,calctype):
#
#
    import os
    import time
    import vgk_delete_input_files as delete_in
    import vgk_create_nam_file as nam
    import vgk_create_sir_file as sir
    import vgk_create_vin_file as vin
    import vgk_read_results as read
    import vgk_delete_result_files as delete_res
    from subprocess import CalledProcessError, Popen, PIPE
    if 'alpha' in calctype:
        calcpar = [state[4], state[4], False]
    if 'CL' in calctype:
        calcpar = [state[4], state[4], True]
#    print calcpar
    names=[None]*4
    #write airfoil to .dat

    AF=['Airfoil 5','SC20411','airfoil.dat']
    f = open('airfoil.dat','wt')
    f.write(AF[0]+'\n')
    iorg = len(airfoil)
    f.write('0    '+str(iorg)+'  '+str(iorg)+'    0.0        1.0        \n\n')

    for i,nr in enumerate(airfoil):
        if airfoil[i,1]>=0:
            f.write('  '+"{:1.6f}".format(airfoil[i,0])+'   '+"{:1.6f}".format(airfoil
                ↪ [i,1])+'\n')
        else:
            f.write('  '+"{:1.6f}".format(airfoil[i,0])+' '+"{:1.6f}".format(airfoil[
                ↪ i,1])+'\n')
    for i,nr in enumerate(airfoil):
        if airfoil[i,2]>=0:
            f.write('  '+"{:1.6f}".format(airfoil[i,0])+'   '+"{:1.6f}".format(airfoil
                ↪ [i,2])+'\n')
        else:
            f.write('  '+"{:1.6f}".format(airfoil[i,0])+' '+"{:1.6f}".format(airfoil[
                ↪ i,2])+'\n')
    f.close()
    names[0]='matlvgk'
    names[1] = AF[0]
    names[2] = AF[2]
    names[3] = AF[1]+'_Re'+str(state[0]/(10**6))+'e6'
    sir.vgk_create_sir_file(names)
    nam.vgk_create_nam_file(names)
    try:
        vin.vgk_create_vin_file(names,state,calcpar)
    except:
          time.sleep(0.2)
          vin.vgk_create_vin_file(names,state,calcpar)
    os.system("vgk.exe" + " < "+names[0]+".sir")
    #read files
```

```
    Coeffs = read.vgk_read_results(names[0]+".FUL")

    return Coeffs
```

vgk_create_sir_file.py
This script generates a SIR file, which is contains the name of the project.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 14:23:36 2014

@author: THC0LQ
"""

def vgk_create_sir_file(names):
    import glob
    import os
    files = glob.glob('*.sir')
    for i in files:
        os.remove(i)
     #create file name
    sirdir=os.getcwd()
    #create file name
    filename = sirdir + '/'+ names[0]+'.SIR'
     #open file
    f = open(filename, 'wt')
    f.write(names[0]+'\n')
    f.close
```

vgk_create_nam_file.py
This script generates a NAM file, which contains a directory which contains the name of all the necessary files.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 12:31:00 2014

@author: THC0LQ
"""
def vgk_create_nam_file(names):
    import glob
    import os
    #remove old .nam
    files = glob.glob('*.nam')
    for i in files:
        os.remove(i)
    #create file name
    namdir=os.getcwd()
    filename = namdir + '/'+names[0]+'.NAM'

    #open file
    f = open(filename, 'wt')
    f.write(names[0]+'.NAM\n')
    f.write(names[0]+'.VIN\n')
    f.write(names[0]+'.IEH\n')
    f.write(names[0]+'.VEH\n')
    f.write(names[0]+'.FUL\n')
    f.write(names[0]+'.BRF\n')
```

```
    f.write(names[0]+'.GRD\n')
    f.write(names[0]+'.UNF\n')
    f.write('1\n')
    f.write(names[2]+' '+'\n')
    f.write('1\n')
    f.write('1\n')
    f.close()
```

vgk_create_vin_file.py
This script generates a VIN file, which contains a the flight parameters, as well as configuration parameters.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 14:18:17 2014

@author: THC0LQ
"""
def vgk_create_vin_file(names,state,calcpar):

    import glob
    import os

    files = glob.glob('*.VIN')
    for i in files:
        os.remove(i)
    #create file name
    vindir=os.getcwd()
    filename =vindir + '/' + names[0]+'.VIN'

    #open file
    f = open(filename, 'wt')
    f.write('    4      '+names[1]+'       \n')
    f.write('\n')
    f.write('  160     0'+'     '+"{:1.4f}".format(state[3]))
    f.write('     '"{:1.4f}".format(state[2])     +'     0.00001.9001.0000.8000.250
        ↪ 0'+'\n')
    f.write('01000    30     1   0.000001     100.00'+'\n')
    f.write(' 1000     1'+'\n')
    f.write('   -1     1'+'\n')
    f.write('  100     1     0.0000     0.0000     0.00001.9001.0000.8000.250     1'+'\n'
        ↪ )
    f.write('    1     1     0    81     1 '+str(state[2])+'    0.00000     ') #stringcat
        ↪ problem?
    print calcpar[2]
    if calcpar[2]:
        f.write('1     ')
        f.write(str("{:6.4f}".format(state[1])))
    else:
        f.write('0     0.0000')
    f.write('     0.5000'+'\n')
    f.write('     0.1500     0.0010     0'+' '+str("{:1.3E}".format(state[0]))+'     '+"
        ↪ {:5.4f}".format(calcpar[0])+'     '+"{:5.4f}".format(calcpar[1])+'
        ↪ 0.0500     1'+'\n')
    f.write('    10     5     0 0.0000000 0.0000000'+'\n')
    f.write('     1    -1'+'\n')
    f.write('  200     1     0.0000     0.0000     0.00001.9001.0000.8000.250     1'+'\n'
        ↪ )
```

```
    f.write('     1     1     0   160     1 '+str(state[2])+'    0.00000     ') #stringcat
        ↪ problem?
    if calcpar[2]:
        f.write('1     ')
        f.write(str("{:6.4f}".format(state[1])))
    else:
        f.write('0     0.0000')
    f.write('     0.5000'+'\n')
    f.write('     0.0750     0.0010     1'+' '+str("{:1.3E}".format(state[0]))+'     '+"
        ↪ {:5.4f}".format(calcpar[0])+'     '+"{:5.4f}".format(calcpar[1])+'
        ↪ 0.0500     1'+'\n')
    f.write('    10     5     0 0.0000000 0.0000000'+'\n')

    f.write('     0     0'+'\n')
    f.close()
```

vgk_read_results.py
After a VGK run, this file reads the output file into Python.

```
## -*- coding: utf-8 -*-
#"""
#Created on Thu Feb 20 12:27:26 2014
#
#@author: THC0LQ
#"""
# Function definition is here
def vgk_read_results(FUL_file):
#    Add both the parameters and return them."

    import os
    import numpy as np
    from operator import itemgetter




    # initialise distri list
    distri=[]
    # initialise end criterion
    end = False
    # initialise dictionaries
    Co2 = {'CD1':[],'CD2':[],'CDV':[],'CDP':[],'CDF':[],'CL':[],'CM':[],'ALP':[],'
        ↪ DISTRI':[]}

    f = open(FUL_file, 'r')
    #obtain last 10 lines
    f.seek(-101,2)
    line=f.readline()
    #set convergence to diverged
    convergence='diverged'

    #loop to determine convergence
    while f.tell() != os.fstat(f.fileno()).st_size:
        # if CD1 or CD2 is found -> converged
        if '*******' in line:
            convergence='diverged'
        elif 'CDV+CD2' in line or 'CDV+CD1' in line:
```

```
            convergence='converged'
        # else -> diverged

        line=f.readline()

    # diverged loop
    if convergence is 'diverged':
            # write infinities
             Co2['CD1']=np.inf
             Co2['CD2']=np.inf
             Co2['CDV']=np.inf
             Co2['CDP']=np.inf
             Co2['CDF']=np.inf
             Co2['CL']=np.inf
             Co2['CM']=np.inf
             Co2['ALP']=np.inf
             Co2['DISTRI']=np.inf



    # converged loop
    if convergence is 'converged':
            # begin of file
            f.seek(0,0)
            while f.tell() != os.fstat(f.fileno()).st_size:
                    if " SUM OF UPPER-SURFACE AND LOWER-SURFACE WAVE-DRAG
                        ↪ CONTRIBUTIONS" in line:
                        #move to next line
                        f.readline()
#                         print(f)
                        #place values in string
                        data = f.readline();
                        #takeout data
                        CDW= [float(data[7:13]),float(data[19:25])];

                    if "     T        X         Z          CP        P/P0     EML"
                        ↪ in line:
                        while end is False:

                            #save line
                            values=f.readline()
                            #split in parts
                            split=values.split()
                            try:
                                #convert first indice to float
                                float(split[0])
                            except:
                                #if first indice is not a float, end loop
                                end = True
                            #add parts to distribution
                            try:
                                distri.append([split[1],split[3]])
                            except:
                                    1==1
#                             print distri
                    line=f.readline()
```

```
            distri = np.array(distri,'float')
            zero =  distri[1:,0].argmin()

            lower=distri[zero+1:,:]
            upper=distri[:zero,:]
            distri=np.concatenate((upper[::-1],lower[::-1]),axis=0)

            #go to a bit before the end
            f.seek(-400,2)
            linem=f.readline()
            while f.tell() != os.fstat(f.fileno()).st_size:
                #check for EM, write all values to variables
                if " EM =" in linem:
                    EMline = linem.split()
#                    EM = EMline[2]
                    ALP = EMline[5]
                    lineCL = f.readline()
                    CLline = lineCL.split()
                    CL = CLline[2]
                    CM = CLline[5]
                    lineCDP = f.readline()
                    CDPline = lineCDP.split()
                    CDP = CDPline[2]
                    CDF = CDPline[5]
                    lineCDV = f.readline()
                    CDVline = lineCDV.split()
                    CDV = CDVline[2]
                    break


                linem =f.readline()
            #write all values to struct
            Co2['CD1']=float(CDW[0])
            Co2['CD2']=float(CDW[1])
            Co2['CDV']=float(CDV)
            Co2['CDP']=float(CDP)
            Co2['CDF']=float(CDF)
            Co2['CL']=float(CL)
            Co2['CM']=float(CM)
            Co2['ALP']=float(ALP)
            Co2['DISTRI']=distri

    return Co2
```

vgk_delete_input_files.py
After a the data has been read, this script deletes all files generated.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 27 13:54:49 2014

@author: THC0LQ
"""

# -*- coding: utf-8 -*-
```

```
"""
Created on Thu Feb 27 13:46:26 2014

@author: THC0LQ
"""
def vgk_delete_input_files():
    import glob
    import os
    types = ('*.NAM','*.SIR','*.VIN')
    files_grabbed = []
    for files in types:
        files_grabbed.extend(glob.glob(files))
    for i in files_grabbed:
        os.remove(i)
```

A.2. META-MODELING METHOD VALIDATION IMPLEMENTATION

main.py

This file calls all the scripts used in the meta-modeling method validation implementation.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 28 14:04:14 2014

@author: THC0LQ
"""

#Fast Aerodynamics Tool using MSES and MACROS combination
#Developed by Roy Veldhuizen roy.veldhuizen@airbus.com
#EIXDS

from A320airfoils import airfoilcap
from Grid_gen import MSESrun
from gtapp import fitrun
import numpy as np
import pickle
import cPickle as cpickle
import pdb
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import matplotlib as mpl
from outlier import cleanup

mpl.rc('text', usetex = True)



DoMSES=0
GTAPP=1
mode= 'fitting' #or predicting
meaneval=1
print 'MSES calc '+ str(DoMSES)
print 'GTAPP '+ str(GTAPP)
print 'Mode '+mode
print 'Value Evaluation ' + str(meaneval)

#var=0 #0 for Re, 1 for CL, 2 for M, 3 for t
```

```
nrofvals=4
avg=np.zeros((nrofvals,4))
avgd=np.zeros((nrofvals,2))
selectionresult=np.zeros((5,4))
selectionresultd=np.zeros((5,2))

for selnr,selectiongrade in enumerate(np.array([0])):
#    for var in range(nrofvals):
    var=0
    if 1==1:
        vareval=2 #evaluated variable, 1 for CDV, 2 for CDW,10 for CDW+CDV 99 for
            ↪ pressure
        MinRe = 20*10**6
        MaxRe = 60*10**6
        dRe = 10*10**6

        Mincl = 0.3
        Maxcl = 0.8
        dcl = 0.05


        MinMach = 0.4
        MaxMach =0.80
        dMach = 0.05

        Mint=10
        Maxt=14
        dt=1

        etaorg=np.array([0,0.093,0.186,0.233,0.279,0.651,0.884])
        corr=1-np.max(etaorg)
        etarange=np.add(etaorg,corr)

        trange = np.arange(Mint,Maxt+dt,dt)
        Rerange = np.arange(MinRe,MaxRe+dRe,dRe)
        CLrange = np.arange(Mincl*1000,(Maxcl*0.999+dcl)*1000,dcl*1000)/1000
        Machrange = np.arange(MinMach*100,(MaxMach*0.999+dMach)*100,dMach*100)/100

        if var==3:
            varrange=trange
        #    varrange=etarange
        if var==0:
            varrange=Rerange
        if var==1:
    #        varrange=CLrange[[0,2,5,8,10]]
            varrange=CLrange
        if var==2:
            varrange=Machrange
    #        varrange=Machrange[[0,2,4,6,8]]

        try:
            varrange=varrange[iterables]
        #    varrange=varrange[iterables]

        #    varrange=[0.65]
            print 'evaluated vals ' + str(varrange)
```

```
except:
    varrange=varrange
    print 'no iterables selected, doing full range'



if DoMSES:
    NrofX=300
    Airfoilfile='A320_wing_old.plb'
    airfoilcap(NrofX,Airfoilfile)
    #extracts the airfoils, and writes them to a .p file
    #format of .p file:
    # 0     u       l
    # .     p       o
    # .     p       w
    # .     e       e
    # 1     r       r
    # stacked in the third dimension



    MSESrun(Airfoilfile,Rerange,CLrange,Machrange,varrange)

if GTAPP:
    if selectiongrade==-1:
        dataten=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Multiple runs folder\\Comparison campaign\\whole t\\
            ↪ resultstencomb14.p", "rb" ))
        datapten=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Multiple runs folder\\Comparison campaign\\whole t\\
            ↪ distritencomb14.p", "rb" ))
        dataten=dataten[::2,:,:,:,:] #check for right selection
        datapten=datapten[::2,:,:,:,:]
        dataten,datapten,xcors,limits = cleanup(dataten,datapten)

    if selectiongrade==0:
        dataten=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Multiple runs folder\\Comparison campaign\\whole t\\
            ↪ resultstencomb14.p", "rb" ))
        datapten=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Multiple runs folder\\Comparison campaign\\whole t\\
            ↪ distritencomb14.p", "rb" ))


        dataten,datapten,xcors,limits = cleanup(dataten,datapten)
        dataten=np.vstack((dataten))
        datapten=np.vstack((datapten))

        fig = plt.figure(1)
        ax1 = fig.add_subplot(1,2,1,projection='3d')
```

```
    viewold=dataten[(dataten[:,0]==20e6) & (dataten[:,-1]==14),:]
    l1,=ax1.plot(viewold[:,1],viewold[:,2],viewold[:,4],'o',markersize
        ↪ =5, zdir='z', c='r')



if selectiongrade>=1:
    datatenx=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All
        ↪ \\10_Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
        ↪ Multiple runs folder\\Comparison campaign\\double t\\
        ↪ resultstencomb14.5.p", "rb" ))
    dataptenx=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All
        ↪ \\10_Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
        ↪ Multiple runs folder\\Comparison campaign\\double t\\
        ↪ distritencomb14.5.p", "rb" ))
    datatenx,dataptenx,xcorsx,limitsx=cleanup(datatenx,dataptenx)
    dataten=np.vstack((dataten,datatenx))
    datapten=np.vstack((datapten,dataptenx))



if selectiongrade==2:
    datatenx2=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All
        ↪ \\10_Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
        ↪ Multiple runs folder\\Comparison campaign\\quad t\\
        ↪ resultstencomb14.25.p", "rb" ))
    dataptenx2=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All
        ↪ \\10_Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
        ↪ Multiple runs folder\\Comparison campaign\\quad t\\
        ↪ distritencomb14.25.p", "rb" ))
    datatenx2,dataptenx2,xcorsx2,limitsx2=cleanup(datatenx2,dataptenx2)
    dataten=np.vstack((dataten,datatenx,datatenx2))
    datapten=np.vstack((datapten,dataptenx,dataptenx2))



if selectiongrade==3:
    datatenx3=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All
        ↪ \\10_Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
        ↪ Multiple runs folder\\Comparison campaign\\oct t\\old pickle\\
        ↪ resultstencomb14.125.p", "rb" ))
    dataptenx3=pickle.load(open( "O:\\ENGINEERING\\EIX\\DS\\3_Team_All
        ↪ \\10_Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
        ↪ Multiple runs folder\\Comparison campaign\\oct t\\old pickle\\
        ↪ distritencomb14.125.p", "rb" ))
    datatenx3,dataptenx3,xcorsx3,limitsx3=cleanup(datatenx3,dataptenx3)
    dataten=np.vstack((dataten,datatenx,datatenx2,datatenx3))
    datapten=np.vstack((datapten,dataptenx,dataptenx2,dataptenx3))

datapten=datapten[:,0:5]
if vareval==99:
    pickle.dump(np.zeros((3,1,7+np.shape(datapten)[1])), open( "means.p"
        ↪ , "wb" ) )
```

```
        else:
            pickle.dump(np.zeros((3,1,7)), open( "means.p", "wb" ) )

        if mode=='fitting':
            fitting=1
        else:
            fitting=0

        #compare varrange with available values
        if var==3:
            varrange= [x for x in np.unique(datateN[:,-1]) for y in varrange if
                ↪ x == y]
        else:
            varrange= [x for x in np.unique(datateN[:,var]) for y in varrange if
                ↪  x == y]
        for itervar in [varrange[index] for index in [0,-1]]:
            print varrange

            datateN[:,[0,1,2,-1]]=np.round(datateN[:,[0,1,2,-1]],3)
            fitrun(itervar,fitting,var,varrange,vareval,datateN,datapteN,limits,
                ↪ xcors)
        figManager = plt.get_current_fig_manager()
        figManager.window.showMaximized()
        plt.savefig('Z:\\Thesis\\Thesis Report\\Images\\fitplot.pdf')
        plt.savefig('Z:\\Thesis\\Exec Summary transfer\\Images\\fitplot.pdf')
        pdb.set_trace()

    if meaneval:
        plt.rc('font', **{'size':'30'})
        divs=np.zeros((1,4))
        datams=np.zeros((1,2))

        charvals=pickle.load(open( "means.p", "rb" ))
        for t in np.unique(charvals[0,charvals[0,:,1]!=0,1]):

            data=charvals[0,charvals[0,:,1]==t,:]
            datam=np.mean(data,axis=0)
            order=charvals[1,charvals[0,:,1]==t,:]
            orderm=np.mean(order,axis=0)
            div=np.divide(datam,orderm)

            print t
            print datam
            print div

            divs=np.vstack((divs,div[2:6])) #take out relevant values
            datams=np.vstack((datams,datam[[2,3]]))

        divs=np.delete(divs,0,0)
        datams=np.delete(datams,0,0)

        rel=divs[1:-1,:] # ignore sides
        reld=datams[1:-1,:]
        avg[var,:]=np.mean(rel,axis=0)
        avg[var,2]= np.mean(rel[rel[:,2]!=float('inf'),2])
        avgd[var,:]=np.mean(reld,axis=0)
```

```
        result=np.mean(avg,axis=0)
        resultd=np.mean(avgd,axis=0)
        selectionresult[selnr,:]=result
        selectionresultd[selnr,:]=resultd
        pickle.dump(selectionresult, open( "selres.p", "wb" ) )
        pickle.dump(selectionresultd, open( "selresd.p", "wb" ) )
        pdb.set_trace()
```

Grid_gen.py
This script generates the input data necessary for meta-modeling.

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 08 17:13:18 2014

@author: THC0LQ
"""

#import importcode as ip
import os
import numpy as np
#import matplotlib.pyplot as plt
#import cPickle as pickle
#import hickle as hkl
import pickle
import MSESblade
import MSESMSETrun
import MSESMSESwrite
import MSESMSESread
import MSESMSESrun
import pdb
#import sPickle
#import tables
import timeit

import glob
def MSESrun(filename,Rerange,CLrange,Machrange,varrange):
    airfoils=pickle.load(open( filename+".p", "rb" ))
#    pdb.set_trace()
    start = timeit.default_timer()


    #pdb.set_trace()
    #mint=10
    #maxt=14
    #dt=1
#    trange= range(np.shape(airfoils)[2])



    runnumber = len(Machrange)

    print varrange
    print Machrange
    print Rerange
    print CLrange
```

```
    resultsten=np.zeros((len(varrange),len(Rerange),len(CLrange),len(Machrange)
        ↪ ,1,9,))
    distriten=np.zeros((len(varrange),len(Rerange),len(CLrange),len(Machrange)
        ↪ ,2,181))

    trans=0.28



    runname='a359_0268_cl030'
    print Machrange
    np.savetxt('machs.'+str(runname), Machrange, delimiter=',',fmt='%1.4f')
    run=1
#     pdb.set_trace()

    if run==1:
        for a,t in enumerate(varrange):
#             pdb.set_trace()
            airfoilfac=airfoils[:,:,a]
            for name in glob.glob('mblade*.*'):
                os.remove(name)
            MSESblade.MSESblade(airfoilfac,runname)
            for name in glob.glob('mdat*.*'):
                os.remove(name)
            MSESMSETrun.MSESMSETrun(runname,1)
            for b,Re in enumerate(Rerange):
                for c,CL in enumerate(CLrange):

                    state=[Re,CL,1,0.6,trans]
                    print [t, Re,CL,1,0.6,trans]
                    for name in glob.glob('msweep*.*'):
                        os.remove(name)
                    for name in glob.glob('da*.*'):
                        os.remove(name)


                    MSESMSESwrite.MSESMSESwrite(state,runname)
                    os.system("mpolarbatch.bat")


                    resultsmat,distrimat = MSESMSESread.MSESMSESread(runname,
                        ↪ runnumber,Machrange)


                    ind=np.where(resultsmat[:,0,2]==0)[0]
                    for p in ind:
                        state=[Re, CL,1,Machrange[p],trans]
                        print [t, Re,CL,1,Machrange[p],trans]
                        for name in glob.glob('msweep*.*'):
                            os.remove(name)
                        for name in glob.glob('da*.*'):
                            os.remove(name)
                        MSESMSESwrite.MSESMSESwrite(state,runname)
                        MSESMSESrun.MSESMSESrun(runname,100)
                        resultsmats,distrimats = MSESMSESread.MSESMSESread(runname
                            ↪ ,1,Machrange)
```

```
                resultsmat[p,:,:] = resultsmats
                distrimat[p,:,:] = distrimats


            resultsmat[:,0,8] = t


            resultsten[a,b,c,:,:,:] = resultsmat
            distriten[a,b,c,:,:,:] = distrimat
            pickle.dump(resultsten, open( "resultstencomb"+str(t)+".p", "wb"
                ↪ ) )
            pickle.dump(distriten, open('distritencomb'+str(t)+".p","wb"))



    stop = timeit.default_timer()
    tottime = stop - start
    print tottime
```

outlier.py
This script removes outliers based on a curve fitting procedure.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 26 20:02:45 2014

@author: Roy
"""
from __future__ import print_function
from __future__ import division
import pdb
#pdb.set_trace()
#from read import read
#from sweep import sweep
import numpy as np



from scipy import arange, array, exp
#from Grid_gen import MSESrun
import numpy as np
import matplotlib.pyplot as plt

import os


import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std




def extrap1d(interpolator):
    xs = interpolator.x
```

```
    ys = interpolator.y

    def pointwise(x):
        if x < xs[0]:
            return ys[0]+(x-xs[0])*(ys[1]-ys[0])/(xs[1]-xs[0])
        elif x > xs[-1]:
            return ys[-1]+(x-xs[-1])*(ys[-1]-ys[-2])/(xs[-1]-xs[-2])
        else:
            return interpolator(x)

    def ufunclike(xs):
        return array(map(pointwise, array(xs)))

    return ufunclike
def maxoccur(input):
    uniqw, inverse = np.unique(input, return_inverse=True)
    freq=np.bincount(inverse)
    maxval=max(uniqw[freq>len(input)/50])
    minval=min(uniqw[freq>len(input)/50])
    return maxval,minval

from pylab import *
from warnings import warn

class OutlierDetector(object):

    def __init__(self, N):
        """
        Generates some simple statistics on the previous N data points

        S = OutlierDetector()
        S.update(4)
        S.update(120)
        """
        self.__index = 0
        self.__N = N
        self.__x = np.zeros(N)      # previous N data points
        self.__x2 = np.zeros(N)     # square of each of the previous N data points
        self.__mean = 0
        self.__var = 0
        self.__cnt = 0

    def get_mean(self):
        return self.__mean
    def get_var(self):
        return self.__var
    def get_std(self):
        return np.sqrt(self.__var)
    def get_data(self):
        return self.__x

    def update(self, point):
        """
        Adds a new datapoint and updates the statistics
        """
        outlier = False
```

```
        if self.__cnt > self.__N:
            if point > self.__mean + 3*np.sqrt(self.__var):
                point = self.__mean + 3*np.sqrt(self.__var)
                outlier = True
            elif point < self.__mean - 3*np.sqrt(self.__var):
                point = self.__mean - 3*np.sqrt(self.__var)
                outlier = True
        else:
            self.__cnt += 1
        prev_point = self.__x[self.__index]

        try:
            self.__x[self.__index] = point
            self.__x2[self.__index] = point ** 2
        except:
            warn("Invalid datapoint, skipping.")
            return False

        self.__mean = self.__mean - prev_point/self.__N + point/self.__N
        self.__var = sum(self.__x2) / self.__N - (self.__mean ** 2)  # could be
            ↪ faster

        self.__index = (self.__index + 1) % self.__N
        return outlier


def cleanup(resultsmatn,distrimatn):
    dataten=np.zeros([1, 9])
    distriten=np.zeros([1, 181])
    nrofstd=3

    xcors=distrimatn[0,0,0,0,:]

    clmax,clmin = maxoccur(resultsmatn[...,1][resultsmatn[...,1]!=0])
    Mmax,Mmin = maxoccur(resultsmatn[...,2][resultsmatn[...,2]!=0])
    Remax, Remin = maxoccur(resultsmatn[...,0][resultsmatn[...,0]!=0])
    tmax, tmin= maxoccur(resultsmatn[...,-1][resultsmatn[...,-1]!=0])
    limit2=np.round([[Remin, clmin, Mmin, tmin],[Remax, clmax, Mmax, tmax]],3)

    for t,tval in enumerate(np.unique(resultsmatn[...,-1])):

        for Re,Reval in enumerate(np.unique(resultsmatn[...,0])[1:]):

            for M,Mval in enumerate(np.unique(resultsmatn[...,2])[1:]):

                if 1==1:
                    sampleold=np.hstack((resultsmatn[t,Re,:,M,0,:],distrimatn[t,Re
                        ↪ ,:,M,1,:])).T
                    sample=sampleold[:,sampleold[0,:]!=0]
                    if np.shape(sample)[1]==0:
                        continue



                    maxpos=sample[5,:].argmax()
```

```
mask=np.equal(range(len(sample[5,:]))>maxpos,False)
sample=sample[:,mask]
maxpos=sample[4,:].argmax()
mask=np.equal(range(len(sample[4,:]))>maxpos,False)
sample=sample[:,mask]
minpos=sample[5,:].argmin()
mask=np.equal(range(len(sample[5,:]))<minpos,False)
sample=sample[:,mask]
minpos=sample[4,:].argmin()
mask=np.equal(range(len(sample[4,:]))<minpos,False)
sample=sample[:,mask]
if np.shape(sample)[1]<9:
    continue

mask=(sample[0,:]!=0) & (sample[0,:]==0)
border=np.where(sample[5,:]>=max(sample[5,:])/1.5)[0][0]

if (border>len(sample[1,:])-2) or (border<2):
    border=np.round(len(sample[1,:])*0.75)

x1 = sample[1,:][:border]
X1 = np.column_stack((x1,x1**2,x1**3,x1**4))
X1 = sm.add_constant(X1)
y1 = sample[5,:border]
model1 = sm.OLS(y1, X1)
results1 = model1.fit()
prstd1, iv_l, iv_u = wls_prediction_std(results1)
outliers1=abs(results1.fittedvalues-y1)>nrofstd*prstd1
mask[:len(outliers1)]=mask[:len(outliers1)]+outliers1


if np.any(mask==True):


    fig = plt.figure(1)
    fig.subplots_adjust(left=0.15,top=0.85,bottom=0.15)
    ax = fig.add_subplot(1, 1, 1)

    mpl.rc('text', usetex = True)
    mpl.rc('font', **{'family': 'serif', 'serif': ['Computer
        ↪ Modern']})

    ax.tick_params(axis='y', pad=15)
    ax.tick_params(axis='x', pad=15)
    ax.tick_params(axis='y', labelsize=12)
    ax.tick_params(axis='x', labelsize=12)
    ax.set_ylabel('2D drag coefficient, $c_{d_w} \:\:\: [-]$',
        ↪ fontsize=12)
    ax.set_xlabel('Mach number, $M \:\:\: [-]$',fontsize=12)
    data,=ax.plot(sample[1,:], sample[5,:], 'ro', label="data")
    fit,=ax.plot(x1, results1.fittedvalues, 'r', label="OLS")
    boundl,=ax.plot(x1, results1.fittedvalues-nrofstd*prstd1, 'r
        ↪ --')
    bound,=ax.plot(x1, nrofstd*prstd1+results1.fittedvalues, 'r:
        ↪ ')
```

```
                    outl,=ax.plot(sample[1,mask],sample[5,mask],'*r',markersize
                        ↪ =15)
                    fig.legend((data,fit,bound,boundl,outl),[r'MSES data',r'
                        ↪ Polynomial fit',r'Poly. fit + 3$\sigma$',r'Poly. fit -
                        ↪  3$\sigma$',r'Outlier'],loc = 'upper center',prop={'
                        ↪ size':12}, ncol=3, labelspacing=0.)

                    plt.savefig(os.getcwd()+'\\Outliers\\eta-'+str(tval)+'RE-'+
                        ↪ str(Reval)+'M-'+str(Mval)+'.pdf')
#                    pdb.set_trace()
                    plt.close()

                sample=sample[:,mask==False]
                sample[5,sample[5,:]<0]=0
                datamat=sample[:9,:].T
                datapmat=sample[9:,:].T
                dataten=np.vstack((dataten,datamat))
                distriten=np.vstack((distriten,datapmat))

    return dataten[1:,:],distriten[1:,:],xcors,limit2
```

cl0.py
This script collects all the differences, and generates the pictures, in this case as a function of c_l.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 06 10:40:17 2014

@author: THC0LQ
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Sep 30 11:04:23 2014

@author: THC0LQ
"""
import pickle
import pdb
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl
from matplotlib.ticker import ScalarFormatter, FormatStrFormatter
import matplotlib as mp
#mp.rcParams['font.size'] = font_size
mp.rcParams['axes.linewidth'] = 0.1
mp.rcParams['patch.linewidth'] = 0.1
mp.rcParams['xtick.minor.width'] = 0.1
mp.rcParams['xtick.major.width'] = 0.1
mp.rcParams['ytick.minor.width'] = 0.1
mp.rcParams['ytick.major.width'] = 0.1
class FixedOrderFormatter(ScalarFormatter):
    """Formats axis ticks using scientific notation with a constant order of
    magnitude"""
    def __init__(self, order_of_mag=0, useOffset=True, useMathText=False):
        self._order_of_mag = order_of_mag
```

```
        ScalarFormatter.__init__(self, useOffset=useOffset,
                                 useMathText=useMathText)
    def _set_orderOfMagnitude(self, range):
        """Over-riding this to avoid having orderOfMagnitude reset elsewhere"""
        self.orderOfMagnitude = self._order_of_mag
mpl.rc('text', usetex = True)
mpl.rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})
vardim='CL'
meshlabels=[r'$0.02$',r'$0.01$',r'$0.005$',r'$0.0025$',r'$0.00125$']
import os
selresmin2=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\low - 2\\selres.p", "rb
    ↪ " ))
selresmin2d=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\low - 2\\selresd.p", "rb
    ↪ " ))
selresmin2data=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\low - 2\\database.p", "
    ↪ rb" ))

selres2=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\2\\selres.p", "rb" ))
selres2d=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\2\\selresd.p", "rb" ))
selres2data=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\2\\database.p", "rb" ))

selres3=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\3\\selres.p", "rb" ))
selres3d=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\3\\selresd.p", "rb" ))
selres3data=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\3\\database.p", "rb" ))




results=selresmin2
results[-2,:]=selres2[0,:]
results[-1,:]=selres3[0,:]

resultsd=selresmin2d
resultsd[-2,:]=selres2d[0,:]
resultsd[-1,:]=selres3d[0,:]
#datatype=1
alldata=np.column_stack((selresmin2data[:,1:,:],selres2data[:,1:,:],selres3data
    ↪ [:,1:,:]))

zeros=np.where(alldata[0,:,0]==0)
var=1
datamin1=alldata[:,zeros[0][var]+1:zeros[0][var+1],:]
data0=alldata[:,zeros[0][var+4]+1:zeros[0][var+5],:]
data1=alldata[:,zeros[0][var+8]+1:zeros[0][var+9],:]
data2=alldata[:,zeros[0][var+12]+1:zeros[0][var+13],:]

#pdb.set_trace()
```

```
if var==3:
        data3=alldata[:,zeros[0][var+16]+1::,:]
else:
        data3=alldata[:,zeros[0][var+16]+1:zeros[0][var+17],:]

varrange=np.unique(data1[0,:,1])

means=np.zeros((3,len(varrange),np.size(datamin1,2),5))

for nr,varreq in enumerate(varrange):
#     pdb.set_trace()
    means[:,nr,:,0]=np.mean(datamin1[:,datamin1[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,1]=np.mean(data0[:,data0[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,2]=np.mean(data1[:,data1[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,3]=np.mean(data2[:,data2[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,4]=np.mean(data3[:,data3[0,:,1]==varreq,:],axis=1)

meansr=np.mean(means[:,1:-1,:,:],axis=1)
test=np.mean(means[:,2:-2:2,:,0],axis=1)
meansr[:,:,0]=test
results2=np.divide(meansr[0,:],meansr[1,:])
fig = plt.figure(1)
fig.subplots_adjust(top=0.9,bottom=0.15)

ax = fig.add_subplot(1, 1, 1)
ax.yaxis.set_major_formatter(FixedOrderFormatter(-4))
a=plt.errorbar(np.unique(data1[0,:,1]),means[1,:,3,1], yerr=means[0,:,3,1],  fmt='ro
    ↪ -')
b=plt.errorbar(np.unique(data1[0,:,1]),means[1,:,4,1], yerr=means[0,:,4,1],  fmt='go
    ↪ -')
plt.setp(a[0],color='r')
plt.setp(a[1],color='g')
plt.xlim([0.29,0.81])
plt.ylabel(r'2D drag coefficient, $c_d$',fontsize=12)
ax.tick_params(axis='y', pad=15)
ax.tick_params(axis='x', pad=15)
ax.yaxis.set_major_formatter(FixedOrderFormatter(-4))
plt.tick_params(axis='y', which='major', labelsize=12)
plt.tick_params(axis='x', which='major', labelsize=12)
fig.legend([a,b],[r'$c_{d_v}$',r'$c_{d_w}$'],prop={'size':12},loc='upper center',
    ↪ ncol=4)
plt.xlabel(r'Airfoil lift coefficient, $c_l \:\:\: [-]$',fontsize=12)

plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) ) )+'\\
    ↪ Thesis Report\\Images\\clverlooptest.pdf')
plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) ) )+'\\Exec
    ↪  Summary transfer\\Images\\clverlooptest.pdf')

fig = plt.figure(2)
fig.subplots_adjust(top=0.9,bottom=0.15)
ax = fig.add_subplot(1, 1, 1)
ax.yaxis.set_major_formatter(FixedOrderFormatter(-4))
b=plt.errorbar(np.unique(data1[0,:,1]),means[1,:,4,1], yerr=means[0,:,4,1],  fmt='go
    ↪ -')
plt.setp(a[0],color='r')
plt.setp(a[1],color='g')
```

```
plt.xlim([0.29,0.36])
plt.ylim([0,6e-4])
plt.ylabel(r'2D drag coefficient, $c_d$',fontsize=12)
ax.tick_params(axis='y', pad=15)
ax.tick_params(axis='x', pad=15)
ax.yaxis.set_major_formatter(FixedOrderFormatter(-4))
plt.tick_params(axis='y', which='major', labelsize=12)
plt.tick_params(axis='x', which='major', labelsize=12)
fig.legend([b],[r'$c_{d_w}$'],prop={'size':12},loc='upper center',ncol=4)
plt.xlabel(r'Airfoil lift coefficient, $c_l \:\:\: [-]$',fontsize=12)

plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) ) )+'\\
    ↪ Thesis Report\\Images\\clverlooptestz.pdf')
plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) ) )+'\\Exec
    ↪  Summary transfer\\Images\\clverlooptestz.pdf')




#plt.show()

#pdb.set_trace()
#for grade in range(1,np.size(means,3)):
#    print grade
##     pdb.set_trace()
#    fig = plt.figure(1)
#    fig.subplots_adjust(top=0.9)
#    ax = fig.add_subplot(2, 2, grade)
##     ax = fig.add_subplot(2, 1, 1)
##     pdb.set_trace()
##     plt.ylabel(r'$\frac{\Delta_{fit}}{Order} \:\:\: [-]$',fontsize=30)
##     plt.gca().xaxis.set_major_locator(MaxNLocator(8,prune='both'))
###     pdb.set_trace()
###     varrange,np.divide(means[0,:,3,grade],means[1,:,3,grade]),'c'
##    A=plt.plot(varrange[...,~np.isnan(means[0,:,2,grade])],np.divide(means[0,:,2,
    ↪ grade],means[1,:,2,grade])[...,~np.isnan(means[0,:,2,grade])],'b',varrange
    ↪ [...,~np.isnan(means[0,:,2,grade])],np.divide(means[0,:,3,grade],means[1,:,3,
    ↪ grade])[...,~np.isnan(means[0,:,2,grade])],'g',varrange[...,~np.isnan(means
    ↪ [0,:,2,grade])],np.divide(means[0,:,4,grade],means[1,:,4,grade])[...,~np.isnan
    ↪ (means[0,:,2,grade])],'r',varrange[...,~np.isnan(means[0,:,2,grade])],np.
    ↪ divide(means[0,:,5,grade],means[1,:,5,grade])[...,~np.isnan(means[0,:,2,grade
    ↪ ])],'c')
##     plt.tick_params(axis='both', which='major', labelsize=30)
##     plt.yscale('log')
##     ax = fig.add_subplot(3, 1, 2)
##     plt.yscale('log')
##     plt.ylabel(r'$\Delta C_d \:\:\: [counts]$',fontsize=30)
##     ax.set_xlim([0.4, 0.8])
##     plt.plot(varrange,datams[:,0]*10**4,'g',varrange,datams[:,1]*10**4,'r',)
##     plt.gca().xaxis.set_major_locator(MaxNLocator(8,prune='both'))
##
##     ax = fig.add_subplot(2, 1, 2)
##     ax.set_xlim([0.4, 0.8])
#    ax.set_ylim([0,10])
##     plt.yscale('log')
#
```

```
##     ax.set_ylim([0, 6])
##     pdb.set_trace()
#     A=plt.plot(varrange[...,~np.isnan(means[0,:,2,grade])],means[0,:,3,grade][...,~
    ↪ np.isnan(means[0,:,2,grade])]*10**4,'g',varrange[...,~np.isnan(means[0,:,2,
    ↪ grade])],means[0,:,4,grade][...,~np.isnan(means[0,:,2,grade])]*10**4,'r',)
##     pdb.set_trace()
##     ax.set_ticks([])
#
#     plt.text(0.5,0.8, r'$\Delta \frac{t}{c}=$'+meshlabels[grade],
    ↪ horizontalalignment='center',fontsize=30,transform = ax.transAxes)
#     if grade==4 or grade==3 or grade==2 or grade==1:
#         if grade==4 or grade==3:
#             plt.xlabel(r'$\frac{t}{c} \:\:\: [-]$',fontsize=40)
#
#         plt.xticks([0.10*100,0.11*100,0.12*100,0.13*100,0.14*100],[r'$0.10$',r'$0
    ↪ .11$',r'$0.12$',r'$0.13$',r'$0.14$'])
#         ax.tick_params(axis='x', pad=15)
#         plt.tick_params(axis='x', which='major', labelsize=30)
#     else:
#         plt.setp( ax.get_xticklabels(), visible=False)
#     if grade==1 or grade==3:
#         plt.ylabel(r'$\Delta C_d \:\:\: [counts]$',fontsize=30)
#         ax.tick_params(axis='y', pad=15)
#         plt.tick_params(axis='y', which='major', labelsize=30)
#     if grade==2 or grade==4:
#         plt.ylabel(r'$\Delta C_d \:\:\: [counts]$',fontsize=30)
#         ax.tick_params(axis='y', pad=15)
#         plt.tick_params(axis='y', which='major', labelsize=30)
#         ax.yaxis.tick_right()
#         ax.yaxis.set_label_position("right")
```

A.3. META-MODELING RESOLUTION SENSITIVITIES IMPLEMENTATION

additionCL.py

This script adjusts the input data to contain a higher resolution, and determines the improvement in differences, in this case as a function of c_l.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 06 12:15:59 2014

@author: THC0LQ
"""


# -*- coding: utf-8 -*-
"""
Created on Tue Sep 30 11:04:23 2014

@author: THC0LQ
"""
import pickle
import pdb
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mp
font_size=12
#mp.rcParams['font.size'] = font_size
```

```
mp.rcParams['axes.linewidth'] = 0.1
mp.rcParams['patch.linewidth'] = 0.1
mp.rcParams['xtick.minor.width'] = 0.1
mp.rcParams['xtick.major.width'] = 0.1
mp.rcParams['ytick.minor.width'] = 0.1
mp.rcParams['ytick.major.width'] = 0.1
from matplotlib.ticker import ScalarFormatter, FormatStrFormatter

class FixedOrderFormatter(ScalarFormatter):
    """Formats axis ticks using scientific notation with a constant order of
    magnitude"""
    def __init__(self, order_of_mag=0, useOffset=True, useMathText=False):
        self._order_of_mag = order_of_mag
        ScalarFormatter.__init__(self, useOffset=useOffset,
                                 useMathText=useMathText)
    def _set_orderOfMagnitude(self, range):
        """Over-riding this to avoid having orderOfMagnitude reset elsewhere"""
        self.orderOfMagnitude = self._order_of_mag
import matplotlib as mpl
mpl.rc('text', usetex = True)
mpl.rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})

vardim='CL'
meshlabels=[r'$0.1$',r'$0.05$',r'$0.025$',r'$0.0125$',r'$0.00625$']
import os
#[20**10*6,10*10**6,5*10**6,2.5*10**6,1.25*10**6]
selresmin2=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\low - 2\\selres.p", "rb
    ↪ " ))
selresmin2d=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\low - 2\\selresd.p", "rb
    ↪ " ))
selresmin2data=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\low - 2\\database.p", "
    ↪ rb" ))

selres2=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\2\\selres.p", "rb" ))
selres2d=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\2\\selresd.p", "rb" ))
selres2data=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\2\\database.p", "rb" ))

selres3=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\3\\selres.p", "rb" ))
selres3d=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\3\\selresd.p", "rb" ))
selres3data=pickle.load(open( os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\
    ↪ picture_gen\\Pred_pics_dev\\Multiple runs\\"+vardim+"\\3\\database.p", "rb" ))


results=selresmin2
results[-2,:]=selres2[0,:]
results[-1,:]=selres3[0,:]

resultsd=selresmin2d
```

```
resultsd[-2,:]=selres2d[0,:]
resultsd[-1,:]=selres3d[0,:]
#datatype=1
alldata=np.column_stack((selresmin2data[:,1:,:],selres2data[:,1:,:],selres3data
    ↪ [:,1:,:]))
zeros=np.where(alldata[0,:,0]==0)
var=1
datamin1=alldata[:,zeros[0][var]+1:zeros[0][var+1],:]
data0=alldata[:,zeros[0][var+4]+1:zeros[0][var+5],:]
data1=alldata[:,zeros[0][var+8]+1:zeros[0][var+9],:]
data2=alldata[:,zeros[0][var+12]+1:zeros[0][var+13],:]
data3=alldata[:,zeros[0][var+16]+1:zeros[0][var+17],:]

varrange=np.unique(data1[0,:,1])

means=np.zeros((3,len(varrange),np.size(datamin1,2),5))

for nr,varreq in enumerate(varrange):
    means[:,nr,:,0]=np.mean(datamin1[:,datamin1[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,1]=np.mean(data0[:,data0[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,2]=np.mean(data1[:,data1[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,3]=np.mean(data2[:,data2[0,:,1]==varreq,:],axis=1)
    means[:,nr,:,4]=np.mean(data3[:,data3[0,:,1]==varreq,:],axis=1)


meansr=np.mean(means[:,1:-1,:,:],axis=1)
test=np.mean(means[:,2:-2:2,:,0],axis=1)
meansr[:,:,0]=test
results2=np.divide(meansr[0,:],meansr[1,:])

for grade in range(1,np.size(means,3)):
    print grade
    fig = plt.figure(1)
    fig.subplots_adjust(top=0.85,bottom=0.15,right=0.85)
    ax = fig.add_subplot(2, 2, grade)

    ax.set_xlim([0.3, 0.8])
    ax.set_ylim([0,4.0e-4])
    fig.subplots_adjust(hspace = 0.3)

    A=plt.plot(varrange[...,~np.isnan(means[0,:,2,grade])],means[0,:,3,grade][...,~
        ↪ np.isnan(means[0,:,2,grade])],'g.-',varrange[...,~np.isnan(means[0,:,2,
        ↪ grade])],means[0,:,4,grade][...,~np.isnan(means[0,:,2,grade])]*1,'r.-',)

    if grade==4 or grade==3 or grade==2 or grade==1:
        ax.set_title(r'$\Delta c_l=$'+meshlabels[grade],fontsize=12)
        if grade==4 or grade==3:
            plt.xlabel(r'Airfoil lift coefficient, $c_l \:\:\: [-]$',fontsize=12)
        plt.yticks(np.multiply([0,1,2,3,4],1e-4),[r'$0.0$',r'$1.0$',r'$2.0$',r'$3.0$
            ↪ ',r'$4.0$'])
        plt.xticks([0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8],[r'$0.30$',' '
            ↪ ,r'$0.40$',' ',r'$0.50$',' ',r'$0.60$',' ',r'$0.70$',' ',r'$0.80$'])
        ax.tick_params(axis='x', pad=15)
        plt.tick_params(axis='x', which='major', labelsize=12)
    else:
        plt.setp( ax.get_xticklabels(), visible=False)
```

```
        plt.setp( ax.get_yticklabels(), visible=False)
    if grade==1 or grade==2:
        plt.setp( ax.get_xticklabels(), visible=False)
    if grade==1 or grade==3:
        plt.ylabel(r'$\Delta c_d \:\:\: [-]$',fontsize=12)
        ax.tick_params(axis='y', pad=15)
        plt.tick_params(axis='y', which='major', labelsize=12)


        ax.yaxis.set_major_formatter(FixedOrderFormatter(-4))
        locs,labels = plt.yticks()
        plt.yticks(np.multiply([0,1,2,3,4],1e-4),[r'$0.0$',r'$1.0$',r'$2.0$',r'$3.0$
            ↪ ',r'$4.0$'])


    if grade==2 or grade==4:
        locs,labels = plt.yticks()
        plt.yticks(locs, map(lambda x: "%.1f" % x, locs*1e4))

        ax.text(0.88, 1.02, r' $\times 10^{-4}$', fontsize=12, transform = ax.
            ↪ transAxes)

        plt.ylabel(r'$\Delta c_d \:\:\: [-]$',fontsize=12)
        ax.tick_params(axis='y', pad=15)
        plt.tick_params(axis='y', which='major', labelsize=12)
        ax.yaxis.tick_right()
        ax.yaxis.set_label_position("right")
        ax.yaxis.set_ticks_position('both')

    fig.legend(A,[r'$c_{d_v}$',r'$c_{d_w}$'],prop={'size':12},loc='upper center',
        ↪ ncol=4)

plt.savefig( os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) )) +'\\
    ↪ Thesis Report\\Images\\CLCLresverg.pdf')
plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) )) +'\\Exec
    ↪  Summary transfer\\Images\\CLCLresverg.pdf')
plt.close()
fig=plt.figure(2)
fig.subplots_adjust(top=0.90,bottom=0.15)
ax = fig.add_subplot(1, 1, 1)
ax.tick_params(axis='x', pad=15)
ax.tick_params(axis='y', pad=15)
plt.ylabel(r'$\frac{\Delta_{fit}}{Order} \:\:\: [-]$',fontsize=12)
plt.ylabel(r'GT-Approx error, $\Delta c_d$',fontsize=12)
plt.xticks([0,1,2,3,4],meshlabels)
plt.tick_params(axis='both', which='major', labelsize=12)
plt.xlabel(r'$c_l$-grid spacing interval, $\Delta c_l\:\:\: [-]$',fontsize=12)
ax.yaxis.set_major_formatter(FixedOrderFormatter(-4))

a=plt.plot([0,1,2,3,4],np.multiply(meansr[0,3,:],1),'g.-',[0,1,2,3,4],np.multiply(
    ↪ meansr[0,4,:],1),'r.-')
fig.legend(a,[r'$c_{d_v}$',r'$c_{d_w}$'],prop={'size':12},loc='upper center',ncol=4)

plt.savefig( os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) )) +'\\
    ↪ Thesis Report\\Images\\CLresverg.pdf')
```

```
plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()) )) +'\\Exec
    ↪ Summary transfer\\Images\\CLresverg.pdf')
plt.close()
```

A.4. 3D METHOD EXTENSION IMPLEMENTATION

prototype_combined-etags.py

This script adjusts extends the method into 3D, and compares the result with previously generated CFD data.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 09 13:33:48 2014

@author: THC0LQ
"""
from __future__ import division
from da.macros import gtapprox
from A320airfoils import airfoilcap
from Grid_gen import MSESrun
from gtapp import fitrun
import numpy as np
import cPickle as pickle
import hickle as hkl
#import pickle
import pdb
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import matplotlib as mpl
from outlier import cleanup
import os
from read import read
from sweep import sweep
from scipy import interpolate
from math import log10, floor
import math
from scipy.interpolate import griddata
font_size=12
import matplotlib as mp
font_size=12
mp.rcParams['font.size'] = font_size
mp.rcParams['axes.linewidth'] = 0.1
mp.rcParams['patch.linewidth'] = 0.1
mp.rcParams['xtick.minor.width'] = 0.1
mp.rcParams['xtick.major.width'] = 0.1
mp.rcParams['ytick.minor.width'] = 0.1
mp.rcParams['ytick.major.width'] = 0.1
#!import code; code.interact(local=vars())
from matplotlib.ticker import ScalarFormatter, FormatStrFormatter

class FixedOrderFormatter(ScalarFormatter):
    """Formats axis ticks using scientific notation with a constant order of
    magnitude"""
    def __init__(self, order_of_mag=0, useOffset=True, useMathText=False):
        self._order_of_mag = order_of_mag
        ScalarFormatter.__init__(self, useOffset=useOffset,
                                 useMathText=useMathText)
    def _set_orderOfMagnitude(self, range):
```

```
        """Over-riding this to avoid having orderOfMagnitude reset elsewhere"""
        self.orderOfMagnitude = self._order_of_mag
mpl.rc('text', usetex = True)
#mpl.rc('text', usetex = True)
mpl.rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})
#pdb


DoMSES=0
GTAPP=1
mode= 'fitting' #or predicting
meaneval=1
A320app=1
Aircraft='A320'
A320test=1
geocap=1

print 'MSES calc '+ str(DoMSES)
print 'GTAPP '+ str(GTAPP)
print 'Mode '+mode
print 'Value Evaluation ' + str(meaneval)
#pdb.set_trace()

from scipy import arange, array, exp

def extrap1d(interpolator):
    xs = interpolator.x
    ys = interpolator.y

    def pointwise(x):
        if x < xs[0]:
            return ys[0]+(x-xs[0])*(ys[1]-ys[0])/(xs[1]-xs[0])
        elif x > xs[-1]:
            return ys[-1]+(x-xs[-1])*(ys[-1]-ys[-2])/(xs[-1]-xs[-2])
        else:
            return interpolator(x)

    def ufunclike(xs):
        return array(map(pointwise, array(xs)))

    return ufunclike

def round_to_n(x, n):
        if n < 1:
                raise ValueError("number of significant digits must be >= 1")
        # Use %e format to get the n most significant digits, as a string.
        format = "%." + str(n-1) + "e"
        as_string = format % x
        return float(as_string)

#import numpy as np
def find_nearest(array,value):
    idx = (np.abs(array-value)).argmin()
    return array[idx]
```

```
filename=Aircraft+'.plb'
owndir=True
selection=[1,2,4,5,6,11,14,19]
roundval=2
wingdir=os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\A320\\Wing\\"

if geocap:
    NrofX=100
    airfoilcap(NrofX,'A320')
    airfoilcap(NrofX,'A320neo')
    airfoilsneo=hkl.load('A320neo.hkl')
    airfoils=hkl.load('A320.hkl')
    philocal,Airfoildata=sweep(wingdir)
    deltaY=Airfoildata[0,0,0,1]-Airfoildata[1,0,0,1]
    Airfoildata[0,:,0,1]=Airfoildata[0,:,0,1]-deltaY
    bref=16.95658
    airfoils[0,:,:]=etarange=np.round(Airfoildata[0,:,0,1]/bref,roundval)
    airfoilsneo[0,:,:]=etarangeneo=np.round(Airfoildata[1,:,0,1]/bref,roundval)
    hkl.dump(airfoils,'A320.hkl')
    hkl.dump(airfoilsneo,'A320neo.hkl')


if A320app:
        #gather input:
    CLdir=os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\A320\\Airfoils\\Drag\\"
    CDdir=os.path.dirname(os.path.dirname(os.getcwd()) ) +"\\A320\\CFD\\"

    philocal,Airfoildata=sweep(wingdir)
    deltaY=Airfoildata[0,0,0,1]-Airfoildata[1,0,0,1]
    Airfoildata[0,:,0,1]=Airfoildata[0,:,0,1]-deltaY

    CLy=read(CDdir)
#     pdb.set_trace()
    MAC=4.19
    cbar=3.609
    Reflight=25.7*10**6
    Mflight=0.78
    rhovmu=Reflight/MAC

    maxY=np.max(Airfoildata[:,:,0,1],axis=1)
    minY=np.min(Airfoildata[:,:,0,1],axis=1)
    cloc=np.max(Airfoildata[:,:,1:,0],axis=2)-np.min(Airfoildata[:,:,1:,0],axis=2)
    Reloc=rhovmu*cloc

    offsetY=Airfoildata[:,:-1,0,1]+0.5*np.diff(Airfoildata[0,:,0,1])
    fold=extrap1d(interpolate.interp1d(offsetY[0,:],philocal[0,:]))
    fnew=extrap1d(interpolate.interp1d(offsetY[1,:],philocal[0,:]))
    phiint=np.concatenate((fold(Airfoildata[0,:,0,1])[np.newaxis,...],fnew(
        ↪ Airfoildata[1,:,0,1])[np.newaxis,...]),axis=0)
    Rel=Reloc*np.cos(phiint)**2
    Ml=Mflight*np.sqrt(np.cos(phiint))
    CLyorg=np.copy(CLy)
    CLy[0,:,1:,:][np.logical_or(CLy[0,:,1:,0]>=maxY[0]  ,CLy[0,:,1:,0]<=minY[0])]=0
```

```
    CLy[1,:,1:,:][np.logical_or(CLy[1,:,1:,0]>=maxY[1]  ,CLy[1,:,1:,0]<=minY[1])]=0


    ymax=np.linspace(minY[0],maxY[0],20)
    ymaxn=np.linspace(minY[1],maxY[1],20)
    fmax=extrap1d(interpolate.interp1d(Airfoildata[0,:,0,1],cloc[0,:]))
    fmaxn=extrap1d(interpolate.interp1d(Airfoildata[1,:,0,1],cloc[1,:]))
    clocintmax=np.concatenate((fmax(ymax)[np.newaxis,...],fmaxn(ymaxn)[np.newaxis
        ↪ ,...]),axis=0)
    phiintmax=np.concatenate((fold(ymax)[np.newaxis,...],fnew(ymaxn)[np.newaxis
        ↪ ,...]),axis=0)
    Relocmax=rhovmu*clocintmax
    Relmax=Relocmax*np.cos(phiintmax)**2
    Mlmax=Mflight*np.sqrt(np.cos(phiintmax))

    #remove tag
    CLy=CLy[:,:,1:,:]

    #interpolate chords for cl locations
    fold=interpolate.interp1d(Airfoildata[0,:,0,1],cloc[0,:])
    fnew=interpolate.interp1d(Airfoildata[1,:,0,1],cloc[1,:])
#     pdb.set_trace()

    cold=fold(CLy[0,0,CLy[0,0,:,0]!=0,0])
    cnew=fnew(CLy[1,0,CLy[1,0,:,0]!=0,0])
    #remove c dependency

    cllocold=np.divide(CLy[0,:,CLy[0,0,:,0]!=0,1:]*cbar*bref,cold[:,None,None])
    cllocnew=np.divide(CLy[1,:,CLy[1,0,:,0]!=0,1:]*cbar*bref,cnew[:,None,None])
    cllocold[:,:,1]=np.divide(CLy[0,:,CLy[0,0,:,0]!=0,2]*bref,1)
    cllocnew[:,:,1]=np.divide(CLy[1,:,CLy[1,0,:,0]!=0,2]*bref,1)

    boundaries=np.zeros((2,3,2))
    boundaries[:,0,0]=np.min(Rel,axis=1)
    boundaries[:,0,1]=np.max(Rel,axis=1)
    boundaries[:,1,0]=np.min(Ml,axis=1)
    boundaries[:,1,1]=np.max(Ml,axis=1)
    boundaries[:,2,0]=[np.min(cllocold),np.min(cllocnew)]
    boundaries[:,2,1]=[np.max(cllocold),np.max(cllocnew)]

    if Aircraft=='A320':
            boundaries=boundaries[0]
    if Aircraft=='A320neo':
            boundaries=boundaries[1]
    MinRe = boundaries[0,0]
    MaxRe = boundaries[0,1]
    dRe = 5*10**6


    Mincl=0.2
    Maxcl=0.8
    dcl = 0.1
#     dcl=0.1


    MinMach =    0.5
```

```
    MaxMach =    boundaries[1,1]
    dMach = 0.1


    Rerange = np.sort(np.append(np.arange(MinRe-dRe,MaxRe+2*dRe,dRe),MaxRe))
    CLrange = np.sort(np.append(np.arange(Mincl-dcl,Maxcl+2*dcl,dcl),Maxcl))
    Machrange = np.sort(np.append(np.arange(MinMach-dMach,MaxMach+2*dMach,dMach),
        ↪ MaxMach))
#    pdb.set_trace()
else:
    MinRe = 20*10**6
    MaxRe = 60*10**6
    dRe = 10*10**6

if DoMSES:

            numcal=len(Rerange)*len(etarange)*len(Machrange)*len(CLrange)
            trans=0
            a,b=MSESrun(Aircraft,Rerange,CLrange,Machrange,etarange,trans)
if GTAPP:
    print 'GTAPP in progress'
    plotval=99 #evaluated variable, 1 for CDV, 2 for CDW,10 for CDW+CDV 99 for
        ↪ pressure
    plot=1
    #values that need to be varied per plot
#     pdb.set_trace()

#     pdb.set_trace()
#     pdb.set_trace()

    if mode=='fitting':
        fitting=1
    else:
        fitting=0

    #load data, either from this directory or from other directory
    if DoMSES==1:
        dataten=hkl.load(os.getcwd()+'\\resultstentest'+str(etarange[-1])+'.hkl')
        datapten=hkl.load(os.getcwd()+'\\distritentest'+str(etarange[-1])+'.hkl')
    else:
        if owndir==True:

                datatendorg=hkl.load( os.path.dirname(os.path.dirname(os.getcwd())
                    ↪ ) +"\\database - improved single\\detail - neo\\
                    ↪ resultstentest1.032.hkl")
                dataptendorg=hkl.load( os.path.dirname(os.path.dirname(os.getcwd())
                    ↪  ) +"\\database - improved single\\detail - neo\\
                    ↪ distritentest1.032.hkl")
                datatenrorg=hkl.load( os.path.dirname(os.path.dirname(os.getcwd())
                    ↪ ) +"\\database - improved single\\range - neo\\
                    ↪ resultstentest1.032.hkl")
                dataptenrorg=hkl.load( os.path.dirname(os.path.dirname(os.getcwd())
                    ↪  ) +"\\database - improved single\\range - neo\\
                    ↪ distritentest1.032.hkl")
```

```
        datatend,dataptend,xcorsd,limitsd = cleanup(datatendorg,
            ↪ dataptendorg)
        datatenr,dataptenr,xcorsnr,limitsr = cleanup(datatenrorg,
            ↪ dataptenrorg)

        datatenneo=np.vstack((datatend,datatenr))
        dataptenneo=np.vstack((dataptend,dataptenr))

        limitsneo=limitsd
        xcorsneo=xcorsd

        datatendorgold=hkl.load( os.path.dirname(os.path.dirname(os.getcwd
            ↪ ()) ) +"\\database - improved single\\detail - A320\\
            ↪ resultstentest0.968.hkl")
        dataptendorgold=hkl.load( os.path.dirname(os.path.dirname(os.getcwd
            ↪ ()) ) +"\\database - improved single\\detail - A320\\
            ↪ distritentest0.968.hkl")
        datatenrorgold=hkl.load( os.path.dirname(os.path.dirname(os.getcwd
            ↪ ()) ) +"\\database - improved single\\range - A320\\
            ↪ resultstentest0.968.hkl")
        dataptenrorgold=hkl.load( os.path.dirname(os.path.dirname(os.getcwd
            ↪ ()) ) +"\\database - improved single\\range - A320\\
            ↪ distritentest0.968.hkl")

        datatendold,dataptendold,xcorsdold,limitsdold = cleanup(
            ↪ datatendorgold,dataptendorgold)
        datatenrold,dataptenrold,xcorsnrold,limitsrold = cleanup(
            ↪ datatenrorgold,dataptenrorgold)
        datatenold=np.vstack((datatendold,datatenrold))
        dataptenold=np.vstack((dataptendold,dataptenrold))


        limitsold=limitsdold
        xcorsold=xcorsdold

    else:

        datatenx=hkl.load("O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Protoype CL clean - 0025\\resultstentest0.209.hkl")
        dataptenx=hkl.load("O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Protoype CL clean - 0025\\distritentest0.209.hkl")
        dataten=hkl.load("O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10_Student_Work
            ↪ \\20140304_Veldhuizen_Roy\\Thesis\\Software\\Protoype CL clean -
            ↪ 005\\resultstentest0.209.hkl")
        datapten=hkl.load("O:\\ENGINEERING\\EIX\\DS\\3_Team_All\\10
            ↪ _Student_Work\\20140304_Veldhuizen_Roy\\Thesis\\Software\\
            ↪ Protoype CL clean - 005\\distritentest0.209.hkl")
        datatenrest,dataptenrest,xcorsrest,limitsrest = cleanup(dataten,
            ↪ datapten)
        datatenx,dataptenx,xcorsx,limitsx = cleanup(datatenx,dataptenx)

        dataten=np.vstack((datatenx,datatenrest))
        datapten=np.vstack((dataptenx,dataptenrest))
```

```
    if Aircraft=='A320neo':
        datateN=datatenneo
        datapteN=dataptenneo
        limits=limitsneo
        xcors=xcorsneo
    if Aircraft=='A320':
        datateN=datatenold
        datapteN=dataptenold
        limits=limitsold
        xcors=xcorsold

    datateN[:,[0,1,2,8]]=np.round(datateN[:,[0,1,2,8]],roundval)
    plotvar1=3 #0 for Re, 1 for CL, 2 for M, 3 for t

    plotvar1req=np.round(0.1160,roundval)
    plotvar2=0 #0 for Re, 1 for CL, 2 for M, 3 for t

    plotvar2req=np.round(18.1e6,roundval)
    plotvar3=1
    plotvar3req=10

    etarangesel=np.unique(datateN[:,-1])
    Rerangesel=np.unique(datateN[:,0])
    plotvar1req=np.array([etarangesel[0]])
    plotvar2req=np.array([Rerangesel[-1]])

if A320test:
    print 'A320test in progress'


    inputmatrix=np.zeros((len(ymax),4))
    inputmatrix[:,0]=Relmax[0,:]
    inputmatrix[:,2]=Mlmax[0,:]
    inputmatrix[:,3]=ymax/bref

    inputmatrixneo=np.zeros((len(ymaxn),4))
    inputmatrixneo[:,0]=Relmax[1,:]
    inputmatrixneo[:,2]=Mlmax[1,:]
    inputmatrixneo[:,3]=ymaxn/bref

    resultsneo=np.zeros((8,np.shape(inputmatrixneo)[0],9))
    distrineo=np.zeros((8,np.shape(inputmatrixneo)[0],2,181))
    resultsold=np.zeros((8,np.shape(inputmatrixneo)[0],9))
    distriold=np.zeros((8,np.shape(inputmatrixneo)[0],2,181))

    CLnum=14

    CLs=CLyorg[0,:,0,0]
    CLnum=10
    np.arange(0,15,2)

    for CLnum in [9]:
        CL=CLyorg[0,CLnum,0,0]
        f1=extrap1d(interpolate.interp1d(np.unique(CLy[0,:,CLy[0,0,:,0]!=0,0]),
            ↪ cllocold[:,CLnum,0]))
```

```
fCDW1=extrap1d(interpolate.interp1d(np.unique(CLy[0,:,CLy[0,0,:,0]!=0,0]),
    ↪ cllocold[:,CLnum,1]))

inputmatrix[:,1]=f1(ymax)
CDWexold=fCDW1(ymax)

outputmatrix=hkl.load('GTout.hkl')
inputmatrix=hkl.load('GTin.hkl')

f2=extrap1d(interpolate.interp1d(np.unique(CLy[1,:,CLy[1,0,:,0]!=0,0]),
    ↪ cllocnew[:,CLnum,0]))
fCDW2=extrap1d(interpolate.interp1d(np.unique(CLy[1,:,CLy[1,0,:,0]!=0,0]),
    ↪ cllocnew[:,CLnum,1]))

inputmatrixneo[:,1]=f2(ymaxn)
CDWex=fCDW2(ymaxn)

inputmatrixneo=np.round(inputmatrixneo,roundval)

outputmatrixneo=hkl.load('GToutneo.hkl')
inputmatrixneo=hkl.load('GTinneo.hkl')

resultsneo=hkl.load('resultsneo.hkl')
distrineo=hkl.load('distrineo.hkl')
resultsold=hkl.load('resultsold.hkl')
distriold=hkl.load('distriold.hkl')

fig=plt.figure(7)
fig.subplots_adjust(top=0.79,bottom=0.15)
ax = fig.add_subplot(1, 2, 1)
plt.ylabel(r'3D wave drag coefficient, $C_{D_w} \: \: \: [-]$',fontsize=12)
plt.xlabel(r'Relative spanwise station, $\eta \: \: \: [-]$',fontsize=12)


MSES2,=plt.plot(np.round(inputmatrix[:,-1],roundval),np.multiply(resultsold[
    ↪ CLnum/2,0:,5]*10**4,np.power(np.cos(phiintmax[0,:]),3)),'r')
CDWexold=fCDW1(ymax)
exper2,=plt.plot(ymax/bref,CDWexold,'r--')
plt.tick_params(axis='y', which='major', labelsize=12)
plt.tick_params(axis='x', which='major', labelsize=12)
ax.tick_params(axis='y', pad=15)
ax.tick_params(axis='x', pad=15)
ax.set_xticks(np.arange(0,1.21,0.2))
ax.set_ylim([0,30])
ax.set_xlim([0.6,1.2])
ax.set_title('A320',fontsize=12)


ax = fig.add_subplot(1, 2, 2)
ax.set_title('A320-PL7A',fontsize=12)
plt.tick_params(labelleft='off')
plt.xlabel(r'Relative spanwise station, $\eta \: \: \: [-]$',fontsize=12)
ax.tick_params(axis='x', pad=15)
plt.tick_params(axis='x', which='major', labelsize=12)
```

```
MSES,= plt.plot(np.round(inputmatrixneo[:,-1],roundval),np.multiply(
    ↪ resultsneo[CLnum/2,:,5]*10**4,np.power(np.cos(phiintmax[1,:]),3)),'b')
CDWex=fCDW2(ymaxn)
ax.set_ylim([0,30])
ax.set_xlim([0.6,1.2])
exper,= plt.plot(ymaxn/bref,CDWex,'b--')

fig.legend((MSES2,exper2,MSES,exper),[r'GT-Approx A320','CFD A320',r'GT-
    ↪ Approx A320-PL7A','CFD A320-PL7A'],prop={'size':12},loc='upper center'
    ↪ ,ncol=2)
plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()))
    ↪ +'\\Thesis Report\\Images\\CFDcomp.pdf')
plt.savefig(os.path.dirname(os.path.dirname(os.path.dirname(os.getcwd()))
    ↪ +'\\Exec Summary transfer\\Images\\CFDcom.pdf')
```